ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP05

INTEGRATION OF QUENCH DETECTION SOLUTION INTO FAIR'S FESA CONTROL SYSTEM

Matic Marn, Andrej Debenjak, Cosylab d.d., Ljubljana, Slovenia
Michal Dziewiecki, GSI Helmholtzzentrum für Schwerionenforschung, Darmstadt, Germany

Abstract

Facility for Antiproton and Ion Research (FAIR) is going to make wide use of superconducting magnets for its components: the SIS100 synchrotron, the Superconducting Fragment Separator (SFRS) and Atomic, Plasma Physics and Applications (APPA) experiments. For all these magnets, uniform quench detection (QuD) electronics have been developed to protect them in case of uncontrolled loss of superconductivity. The QuD system will contain ca. 1500 electronic units, each having an Ethernet interface for controls, monitoring, data acquisition, and time synchronization. The units will be grouped into sub-networks of ca. 100 units and interfaced via dedicated control computers to the accelerator network. The interfacing software used to expose QuD functions to the FAIR controls framework is implemented as a Front-End Software Architecture (FESA) class. The software provides a solution for the constant collection of the data and monitoring of the system, storing the complete snapshot in the case a quench event is detected, and prompt notification of a quench to other components of the FAIR facility. The software is developed with special attention to robustness and reliability.

INTRODUCTION

Quench (uncontrolled superconductivity loss) events are considered a major threat in superconductor technology. During a quench of a superconducting magnet, the energy stored in the magnetic field is rapidly released in the quench area, which may cause severe damage. However, damage can be avoided by quench protection devices like quench heaters or dumping resistors. To activate them right in time, QuD systems are used.

FAIR's SIS100 synchrotron, the SFRS and APPA will employ a number of superconducting magnets for their operation [1]. For these magnets, voltage-measurement-based QuD methods are utilized: direct voltage measurement for current leads, bridge measurement for high-current magnets and pickup coils for low-current magnets [2]. Voltages over all superconducting components are monitored, mostly redundantly, by assigned QuD units. Each unit consists of a custom analog front-end (specific for each monitored part type), a set of comparators (quench detection relies on comparing acquired signals against pre-programmed thresholds), and circuitry for device control and data acquisition.

The data acquisition feature is not involved in quench detection itself, but collecting signals allows in-depth analysis of magnet operation and 'post-mortem' data are crucial for understanding magnet failures once they occur.

FAIR's control system is FESA-based, and the integration of the QuD units is done via a FESA class. The QuD FESA class was designed to control up to 96 QuD units by a single equipment class, however, there is no technical limitation to extend this number further as far as enough processing and memory resources are allocated to the hosting computer.

FESA is a C++ framework used for developing real-time (RT) software for devices that have to be integrated into a control system – software that controls and monitors accelerator equipment and performs data acquisition. It comes with a rich environment for designing, developing, deploying, and testing. It offers wide possibilities in data acquisition, data handling, and standardized generic interfaces called properties, which can be specified for each device being integrated. An acquisition property allows for acquired data to be published (i.e., notified) to the users, while command and setting properties facilitate the user to provide input to a FESA class. In addition, FESA provides tight and seamless integration of the FAIR's White Rabbit [3] timing system and focuses on real-time execution.

FAIR QUENCH DETECTION UNITS

Each QuD unit can be divided into two parts: the analog QuD circuitry (which is virtually independent of the control system or any other high-level electronics) and a module for device control and data acquisition. The latter is built around a Cyclone IV FPGA and runs a NIOS-II soft-core with FreeRTOS-based software. Its functions range from simple operations (like remote forcing or resetting of the quench signal) through controlling quench thresholds to continuous 10kHz signal acquisition. Further, it provides numerous diagnostics features and remote updates.

Digital Interface

All units are controlled via a fast Ethernet interface running multiple protocols on top of IP v.4. For generic device control, the Simplified Universal Serial Interface (SUSI) protocol [4] has been developed, which utilizes a variable size register model for device programming. For data download, a custom real-time data transfer protocol has been implemented directly in FPGA hardware to offload the softcore. Further, some generic protocols (DHCP, mDNS) are used to support IP connectivity.

It's worth noting that the quench signal is transmitted over a dedicated digital line and it's independent of the network interface.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP05

Data Format

QuD units sample 8 channels of analog data with a nominal frequency of 10 kHz along with further 16 binary status signals. These are packed into frames consisting of 64 samples and a summary. The summary contains mean values of analog channels, a snapshot of the binary status, measured quench threshold values, frame number, and last but not least, its timestamp. Each frame counts 1328 bytes which fit into a single UDP frame, allowing easy and efficient data transmission.

Time Synchronization

Local data timestamping is used to synchronize streams coming from QuD units. Timestamps are added to data at the earliest possible stage in the unit hardware rather than in the global data aggregator. This way effects of random delays on the network are eliminated. To provide this, all units must be equipped with local clocks and they must be synchronized to an external global clock. The synchronization is achieved utilizing the Precision Time Protocol (PTP).

Internally, units use sample and frame triggers to control the data acquisition. These triggers can be free-running or synchronized to the global clock. In the latter case, frames coming from all synchronized units will have the same timestamps, which makes data aggregation much easier.

SYSTEM OUTLINE

Hardware Arrangement

QuD units are logically (and mostly also physically) grouped into subsets of up to 96 units with a single control server. The local network between the server and units is isolated from the general accelerator network so that direct access to units from outside is not possible. Each server runs an instance of a dedicated FESA class as described below to provide QuD control and data acquisition.

FESA Architecture

To integrate the QuD units into the FAIR control system a FESA class application was designed. The class supports real-time data acquisition from all the units which stream the data by User Datagram Protocol (UDP) packets. The class periodically monitors the units and subscribes or unsubscribes from the unit data streams based on the operational status of each unit. This design allows robust and reliable data availability. The data must be made available to any FESA client for a certain amount of time after it has been acquired. For that reason, once the data is received it is put inside a First-In, First-Out (FIFO) data acquisition (DAQ) buffer which supports a single-write-multiple-read interface with minimal locking required for synchronization. The data is available on demand as either raw frame data or processed scaled data until it is overwritten in the buffer. The data is also made available to the interested clients continuously as it is received at a 10 Hz update rate. The class monitors received data and as soon as a quench event is detected on one of the units it signals an interlock to the FAIR interlock system. In addition to the DAQ capabilities, the QuD FESA class also provides command and setting properties for the user to setup and configure the QuD units.

Even though each QuD FESA class instance interacts with up to 96 QuD units, the FESA class only implements one software device-instance. While from the FESA perspective it would be most appropriate that one device-instance represents one QuD unit, this is not the case for this particular implementation. With the complete QuD system containing around 1500 QuD units and each unit being represented by a corresponding FESA device instance, this would introduce a large amount of FESA software devices to the control network which is inconvenient and unbefitting.

IMPLEMENTATION DETAILS

DAQ Buffer

The QuD FESA class receives the real-time data from up to 96 QuD units. The data needs to be aggregated, aligned, and stored for a predefined period (typically several minutes). The rate of received data is substantial and was one of the main aspects affecting the overall design of the buffer.

The whole buffer is pre-allocated in advance to avoid expensive memory allocations and moves during run-time. Memory for individual units is located in a continuous fashion to increase locality and reduce the number of cache misses.

Data from different units are time-aligned by the timestamp of the frames. For the frame to be inserted into the buffer the timestamp has to match exactly. This means all the units need to be configured to use the global clock as a trigger reference.

Performance tests have shown that a single writer thread can easily handle the expected load so the buffer is designed to be used by a single writer thread. This way, the synchronization between multiple writers is avoided.

The buffer is split into sections called blocks. Each block consists of multiple DAQ frames. At any time, the data may be written into the buffer only in the sliding writer window which is 2-block wide. When the data is written one block past the writing window the window slides by one block forward to include the new block. The old block which is no longer included in the writing window may not be written to anymore and may be read from.

Reader threads do not block the writer thread while they are reading the data. This is achieved by the reader threads reading the current position of the writing window before and after the data was read. With that, the reader thread can determine if the data was overwritten while being read. Reading and updating the writer window position is synchronized with a mutex lock, which affects the performance negligibly.

Communication Sockets

The QuD FESA class implements two types of sockets to communicate with units. First, a general purpose SUSI socket is used to write to and read from a QuD unit. The

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP05

communication is entirely done using UDP packets with two-way communication for each interaction. Appropriate synchronization mutex is used to make sure that each interaction is correctly handled. The second implemented socket is a real-time socket which is only used to receive the streamed real-time data. It exposes the Linux socket file-descriptor which allows to efficiently poll for new data for all units.

Data Acquisition

A FESA custom event source is used to receive the streamed real-time data. When an acquisition start command is sent to the QuD units, they start streaming UDP packets containing the real-time data. Due to the relatively large number of units and to improve the performance, the Linux poll command is used. Whenever one of the sockets has available data, the data is read, deserialized to a DAQ frame structure, and put inside the DAQ buffer. As soon as a new block is written into the DAQ buffer, the custom event source triggers the acquisition RT action. Before adding the frame to the buffer, it is also checked whether a quench status flag is set. If that is the case, the quench handling RT action is immediately triggered. The custom event source thread only does the absolutely most critical work required and other less time-sensitive operations are handled in other lower-priority threads.

Data Processing

Some data acquisition modes require the data to be processed before being sent to the interested clients. There are two processing procedures implemented. Namely, the data may be decimated and scaled. For decimation, a user-specified number of data samples is averaged. For scaling, the data can be scaled from raw ADC values to volts according to preset calibration coefficients.

Data Acquisition Modes

QuD data is acquired in the form of raw DAQ frames, deserialized, and stored in the DAQ buffer. It resides in the buffer until it is requested by the user or a continuous notification is triggered. Three modes of publishing the data, are supported i.e., *Raw*, *Continuous* and *Post-mortem*.

Raw UDP packet data (bytes), can be requested with *GetRawData* command FESA property. The user has to supply the timestamp of the first frame of interest and the number of frames required. As long as the requested data is available in the DAQ buffer, it is serialized back to raw bytes and sent to the user.

In addition to the raw data, the user can subscribe to continuous data notifications. This allows getting a continuous stream at 10 Hz update rate. The data is decimated and scaled to Volts before sending to the user.

The last available data acquisition mode is post-mortem acquisition. Since a quench event may be triggered at any time, it is paramount for the post-mortem analysis to have access to the acquired data before and after the quench happened. In this case, the user may request the post-mortem data with the *GetPostMortemData* command property. The user supplies the timestamp of the first frame of interest and the number of frames. The data is read from the DAQ buffer, scaled, and sent to the user.

Robustness and Reliability

A single class instance works with multiple units. One or more malfunctioning units should not degrade the performance of the class or the ability to use other units.

To achieve the desired reliability, a health monitoring system is implemented. The QuD FESA class maintains internal statistics of the received data frames. With each received packet the statistics are updated. These statistics are then periodically monitored and appropriate action is taken for each unit in case of detected misbehavior. For example, when a unit is correctly configured and is not sending data when it should, the acquisition is restarted on the unit. Similarly, when the unit is misconfigured and is sending data, the acquisition is stopped on the unit, since the sent data is considered invalid. In addition, periodical checks are implemented that units are correctly synchronized and configured at all times.

Each unit can be masked (i.e., easily excluded from the system), in case there is some issue with the unit that cannot be fixed remotely and physical interaction is required. When a unit is masked it effectively means that it is ignored by the QuD FESA class and as such the unit is not operational.

Standard DAQ API

The class exposes the acquired data via the FAIR standardized DAQ Application Programming Interface (API). This allows a common client Graphical User Interface (GUI) implementation to be used across multiple FAIR accelerator control solutions.

CONCLUSION

QuD units are an essential part of preventing damage to the accelerator devices in case of a quench event. As such they are integrated into the FESA control system. Care is taken to provide a performant, robust and reliable solution. At the same time, the implementation offers the user full access to the settings and configuration of the QuD units. Data acquired from the QuD units is buffered, processed, and made available to interested FESA clients.

REFERENCES

[1] W. F. Henning, "FAIR - an International Accelerator Facility for Research With Ions and Antiprotons", in *Proc. EPAC'04*, Lucerne, Switzerland, paper TUXCH02, pp. 50–53, July 2004.

[2] P. Szwanguber *et al.*, "Study on Mutual-Inductance-Based Quench Detector Dedicated to Corrector Magnets of SIS100", *IEEE Trans. Appl. Supercond.*, vol. 29, nr. 4, pp. 1-5, 2019. doi:10.1109/TASC.2019.2896139

[3] *The White Rabbit Project - official web page*, August 2022. https://white-rabbit.web.cern.ch

[4] M. Dziewiecki, "Quench Detection System Developer's Handbook", Internal GSI document, Darmstadt, 2022.