University of Pisa

School of Engineering

Master of Science in Artificial Intelligence
and Data Engineering

Final report

Italian Summer School at Fermilab

Software and Hardware development for the Mu2e trigger and data acquisition system

---

Student:
Tommaso Baldi

Academic year 2022-2023

Contents

1 Introduction

1.1 The Mu2e Experiment

The Mu2e experiment at Fermilab will be 10,000 times more sensitive than previous experiments looking for muon-to-electron conversion [1]. This precise and complex experimental apparatus will produce 200 million billion muons per year. The accelerator complex repurposes elements of the infrastructure that produced high energy proton and anti-protons beams for the Tevatron experiments to produce the high-intensity muon beams necessary for Mu2e and the Muon (g-2) experiments.

The Fermilab Booster will accelerate protons to the 8 GeV needed to produce the intense muon beam employed by Mu2e.

The protons will travel from the Booster to the Recycler where they will be stacked, bunched, and extracted to the Delivery ring. The Delivery ring is located in the repurposed Debuncher. Once in the Delivery ring the protons will be slow extracted and delivered to the Mu2e apparatus. A system of three superconducting solenoidal magnets (Production Solenoid, Transport Solenoid and Detector Solenoid) will transport the intense low-energy muon beam to the experimental area where Mu2e is located.

The 8 GeV protons will arrive in bunches from the Delivery ring and enter the Mu2e Production Solenoid at a slight angle to its axis and strike a tungsten production target about the size of a pencil. These collisions will create a cascade of particles, including pions that decay into muons. The magnetic field of the Production Solenoid will capture some of the muons and spiral them into the Transport Solenoid. Only about 1 in 300 protons that collide with the production target will generate a muon that moves into the Transport Solenoid. Throughout the experiment's projected three-year running period, roughly 10 billion muons per second will be stopped.

Muons in the Transport Solenoid will travel inside an evacuated vessel towards the Mu2e aluminum target. The Mu2e detector is the particle physics detector embedded inside the Detector Solenoid that provides a magnetic field in the detector region that allows the momentum of the conversion electrons to be accurately determined. The Mu2e detectors consist of two main parts: the magnetic spectrometer to measure particles momentum, and the electromagnetic calorimeter to measure particles energy and time of impact.

Improvements to the accelerator could extend the initial Mu2e sensitivity by a factor of ten or more. This is comparable to Mu2e initially producing a number of muons equivalent to all the grains of sand on the Earth's beaches. This would provide a valuable tool for physics research whether or not Mu2e discovers muon-to-electron conversion during its first, lower-intensity phase. If Mu2e does observe charged lepton conversion, an upgraded accelerator would enable Mu2e to study in depth the details of the conversion by providing more data. If Mu2e does not observe the conversion, the collaboration could continue the search with a wider net and also search for signs of never-before-seen physics in rare processes that have previously been out of reach of physics machines.

1.2 The Mu2e Experiment Facility

The Mu2e apparatus is extensively documented in the conceptual Design and Technical Report.

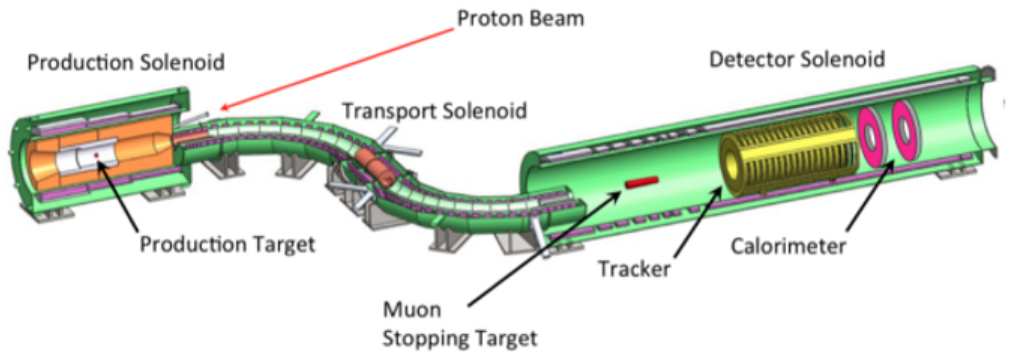

Figure 1: Mu2e apparatus: the proton beam enters from the right at the junction between the Production Solenoid and the Transport Solenoid, and strikes the production target. The cosmic ray veto system, which surrounds the Detector Solenoid, and the muon stopping monitor are not shown in this scheme (source: Mu2e experiment data center.

1.2.1 Production Solenoid

The Mu2e magnet system consists of three large superconducting solenoids. The first one in the chain of magnets is the Production Solenoid (PS), whose role is to collect and focus pions and muons generated in interactions of an 8-GeV proton beam with a tilted high-Z target, by supplying a peak axial field between 4.6 T and 5.0 T and an axial field gradient of about 1 T/m, within a 1.5 m warm bore.

The PS is a challenging magnet because of the relatively high magnetic field and a harsh radiation environment that requires the state-of-the-art conductor, both in terms of the current-carrying capacity and structural strength. The PS coil is protected by a massive Heat and Radiation Shield (HRS).

1.2.2 Transport Solenoid

The role of the S-shaped Transport Solenoid (TS) is to filter and transport the muon beam (approximately 1011 muon/s) to the Detector Solenoid. It is composed of 14 superconducting units (solenoids and toroids) and is divided in five sections:

- a 1 m long straight section
- a 90-degree elbow, with 3 meters radius of curvature
- a second 2 m long straight section
- a second 90-degree elbow, similar to the first, that turns the beam line in a direction parallel to the first one
- a final 1 m long straigth section

The resulting length of the Transport Solenoid is 13 m. To improve the purity of the muon beam, the Transport Solenoid has an absorber placed in its central part that stops charged particles (mainly antiprotons). A state-of-the-art collimator system is placed in the same zone to select only low energy muons with momentum below 0.08 GeV/c0. Moreover, the S-shape of the solenoid removes neutral particles that travel in a straight direction.

1.2.3 Detector Solenoid

The Detector Solenoid (DS) is a 11 m long component, with a decreasing magnetic field in the first sector (from 2 T to 1 T). It hosts the muon stopping target, schematically represented in figure **??**.

Muons impacting the disks come to rest and replace the electrons lying in the 1s orbit of the aluminium atoms. The lifetime of the muon in the muonic atom is 864 ns. The non uniformity of magnetic field plays an important role in reducing the background coming from high energy electrons transported to the Detector Solenoid. The magnetic field gradient is generated introducing spacers to change the winding density of the superconducting cable, which is made of aluminum-stabilized $NiTi$.

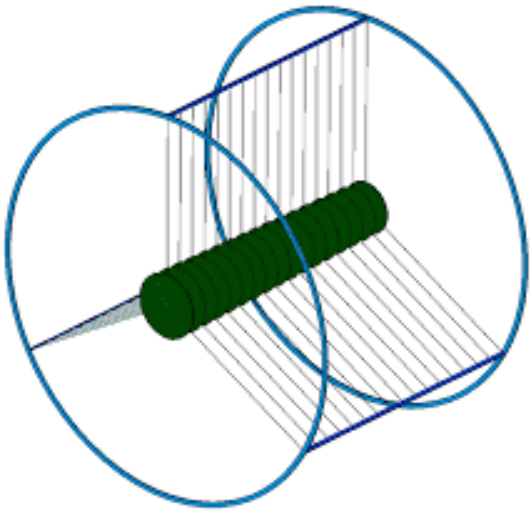

Figure 2: The Mu2e stopping target is made of 17 aluminium disks, 0.2 mm thick, spaced 5.0cm apart along the Detector Solenoid axis. The disks radii decrease from 8.3 cm at the upstream end to 6.53 cm at the downstream end (source: Mu2e experiment data center).

1.2.4 The Tracker and the Electromagnetic Calorimeter

The Mu2e detector is located inside the evacuated warm bore of the Detector Solenoid in a nearly uniform 1 Tesla magnetic field and is designed to efficiently and accurately identify and analyse the helical trajectories of ∼105 MeV electrons in the high-rate time varying environment of Mu2e. The detector consists of a Tracker and an electromagnetic calorimeter that provide redundant energy/momentum, timing, and trajectory measurements. A cosmic ray veto, consisting of both active and passive elements, surrounds the Detector Solenoid and nearly half of the Transport Solenoid and is used to identify possible background events due to cosmic-rays.

The Mu2e Tracker is designed to accurately reconstruct the helical trajectories of electrons in a uniform 1 Tesla magnetic field and measure their momenta.

Given that multiple scattering in the Tracker material dominates the resolution on the measurement of the helix parameters, the mechanical structure of the detector has been made extremely light. The Tracker is made of straw drift tubes; and is called T-Tracker because the straws are transverse to the axis of the Detector Solenoid. The basic detector element is made of a 20 μm sense wire inside a straw tube filled with gas. The straws are 5 mm diameter tubes, made of 15 μm thick metallic Mylar. The Tracker is made of approximately 2000 straws arranged along 18 stations across the 3 m Tracker length. One Tracker plane consists of two layers of straws to improve the reconstruction efficiency and help to overcome the classic left-right ambiguity. A 1 mm gap between straws allows for manufacturing tolerance and expansion due to the internal pressure. A larger radius ring outside the active detector region supports the straws and the electronics boards.

Each straw has one preamplifier and one time-to-digital converter (TDC) placed on each tip, in order to measure the signal arrival time on both sides. It uses also analog to digital converters (ADC) to measure the total integrated charge, providing useful information for particle identification.

The Tracker has been designed to observe only electrons with energy greater than 53 MeV. Electrons below this threshold travel undetected in the central un-instrumented volume of the Tracker. They are approximately 3% of the total electron flux coming from muon decays. Since momentum resolution is a crucial factor to suppress critical backgrounds, the Tracker is required to have a momentum resolution better than 180 keV for 100 MeV electrons.

The Mu2e calorimeter provides additional energy, position, and timing information for particles' trajectories reconstructed by the Tracker. The two detectors use different physical and technological processes to perform their measurements, to rely on uncorrelated error sources. This helps to reduce backgrounds and provides a cross check to verify the quality of signal events.

The calorimeter operates in the same solenoidal magnetic field and vacuum level as the Tracker. It handles a large flux of particles, mostly a low energy background of protons, neutrons and gamma rays produced by muon captures in the stopping target. It also manages a large flux of electrons coming from muons decays in the aluminium stopping target, and other produced particles during the beam injection.

1.2.5 Cosmic Ray Veto

Cosmic ray muons may initiate processes and produce particles that interact with the detectors and produce unwanted backgrounds. The simulation show that approximately one background event generated by cosmic ray muons may be erroneously reconstructed as a conversion electron signal per day. This source of background can be reduced to a negligible level by introducing passive and active shielding.

The *Cosmic Ray Veto* (CRV) surrounds the entire volume occupied by the Detector Solenoid and the downstream part of the Transport Solenoid. It consists of four layers of extruded scintillator strips with silicon photosensors and aluminum absorbers.

The cosmic ray induced background rate will be monitored between beam spills and when the beam will be turned off. This allows to perform a direct measurement of the background level. The study of the background rate will be initiated as soon as the Detector Solenoid and the cosmic ray veto are in place.

1.3 Trigger and Data Acquisition System

The Mu2e Trigger and Data Acquisition (TDAQ) subsystem provides necessary components to collect digitised data from the Tracker, Calorimeter, Cosmic Ray Veto and Beam Monitoring systems (Stopping Target Monitor and Extinction Monitor), and deliver that data to the online and offline processors for further analysis. It is also responsible for detector synchronisation, control, monitoring, and operator interfaces. The Mu2e TDAQ is based on a "streaming" readout. This means that Tracker and Calorimeter detector data is digitised, zero-suppressed in front-end electronics, and then transmitted off the detector to the TDAQ system. While this approach results in a higher off-detector data rate, it also provides greater flexibility in data analysis and filtering, as well as a simplified architecture. The Mu2e TDAQ architecture is further simplified by the integration of all off-detector components in a "TDAQ Server" which functions as a centralised controller, data collector and data processor. A single TDAQ Server can be used as a complete standalone data acquisition/processing system or multiple TDAQ Servers can be connected together to form a highly scalable system.

1.3.1 Requirements

The TDAQ monitors, selects, and validates physics and calibration data from the Mu2e detector for final stewardship by the offline computing systems. The TDAQ combines information from about 450 detector data sources and applies filters to reduce the average data volume by a factor of at least 100 before it can be transferred to offline storage. The TDAQ also provides a timing and control network for precise synchronisation and control of the data sources and readout, along with a Detector Control System (DCS) for operational control and monitoring of all Mu2e subsystems. Figure **??** shows a full view with focus on the interfaces connected to TDAQ: Tracker and Calorimeter ROCs, Detector Hall, WH Control Room.

TDAQ requirements are based on the following experiment attributes:

- Environment: the TDAQ system will be located in the surface level electronics room and connected to the detector by optical fiber. There are no radiation or temperature issues. The TDAQ will however be exposed to a magnetic fringe field from the detector solenoid at a level of about 20-30 gauss.
- Beam Structure: supercycle is the temporal window between two proton beams (1.4 s). Beam is delivered to the detector during the first 467 ms of each supercycle. During this period there can be up to eight 54 ms spills. Spills are proton pulses delivered to the target in the Production Solenoid. Each spill contains approximately 32 000 "$\mu Bunches$", for a total of 256 000 $\mu Bunches$ in a 1.4 second supercycle. A $\mu Bunch$ is 1695 ns. Readout Controllers store data from the digitizers during the "live gate". The live gate width is programmable but is nominally the last 1000 ns of each $\mu Bunch$.
- Detectors: the TDAQ system receives data from the following subdetectors:
  - Tracker – 20 736 straw tubes: 96 tubes per "panel", 12 panels per "station" and 18 stations. There are 216 Readout Controllers (one for each panel) located inside the cryostat. Straw tubes are read from both ends to determine hit location along the wire. The readout produces two TDC values (16 bits each) and typically six ADC values (10 bits each) per hit. The ADC values are the analog sum of both ends of the straw.
  - Calorimeter – 1610 crystals in 2 disks. There are 192 Readout Controllers located inside the cryostat. Each crystal is connected to two avalanche photodiodes (APDs). The readout produces approximately 25 ADC values (12 bits each) per hit.
  - Cosmic Ray Veto system – 21 504 Silicon Photomultipliers (SiPMs). There are 336 front-end boards (64 channels each), and 14 Readout Controllers (24 front-end boards each). The readout generates approximately 12 bytes for each hit. CRV data is used in the offline reconstruction, so readout is only necessary for timestamps that have passed the Tracker and Calorimeter filters.
  - Extinction and Target Monitors – monitors will be implemented as standalone systems with local processing. A subset of information will be forwarded to the TDAQ for inclusion in the run conditions database and optionally in the event stream.
- Data rate: the detector will generate an estimated 150 Kbytes of zero-suppressed data per $\mu Bunch$, for an average data rate of about 90 *Gbytes/s* when beam is present. To reduce TDAQ bandwidth requirements, this data is buffered in Readout Controller (ROC) memory during the spill period, and transmitted to the TDAQ over the full supercycle for an average data rate of about 28 *Gbytes/s*.
- Processing: the TDAQ system provides online processing to perform Tracker and Calorimeter filters. The goal of these filters is to reduce the data rate by a factor of at least 100, limiting the offline data storage to less than 7 *Petabytes/year*. Based on preliminary estimates, the online processing requirement is approximately 30 *TeraFLOPS*.

1.3.2 Architecture

Readout Controllers digitise and zero-suppress data at the detector. The data is then transmitted over optical links to TDAQ Servers in the surface level electronics room. Control information is sent from the TDAQ Servers to the Readout Controllers over the same bidirectional optical links. Data is exchanged between TDAQ Servers (via the Event Building Network) to form complete events. The TDAQ Servers filter these events and forward a small subset of them to offline storage.

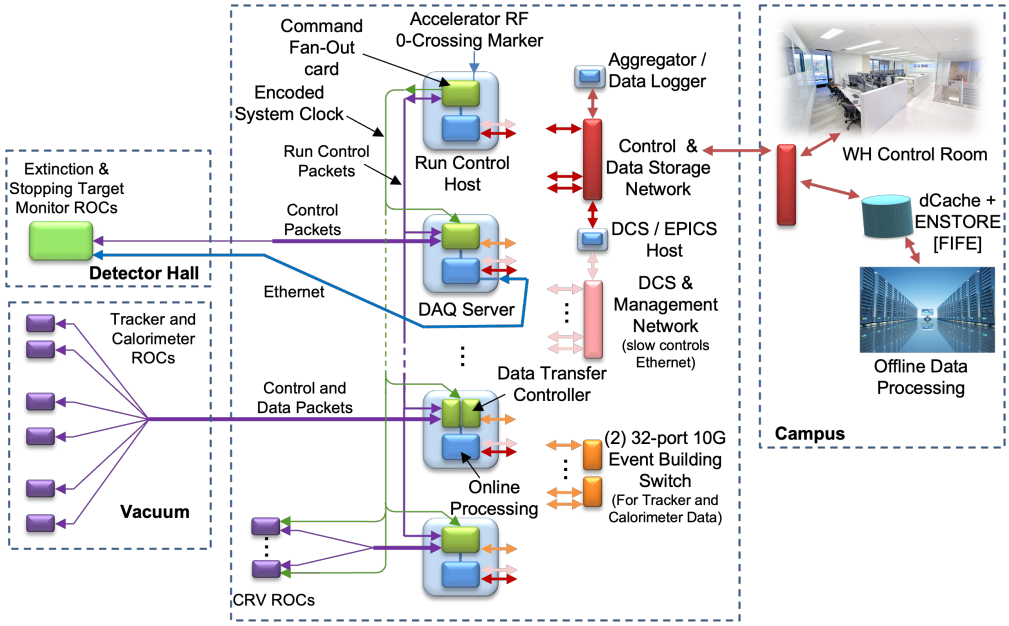

Figure 3: Full view of TDAQ and its interfaces with subsystems

Subproject	Interfaces
Tracker, Calorimeter, CRV	The TDAQ connects to detector readout controllers via optical links which carry fast control, slow control and data. The Timing system supplies an encoded System Clock to each detector subsystem.
Solenoids, Beamline	The TDAQ provides the infrastructure for slow control and monitoring, and readout of the stopping target monitor.
Accelerator	The TDAQ receives beam timing and status information from the accelerator for timing system synchronization. The TDAQ also provides the infrastructure for slow control and readout of the extinction monitor.
Civil	The civil construction subproject provides the surface level electronics room, power, and air conditioning for the TDAQ. It also supplies cable chases for connecting the detector hall electronics to the electronics room.

1.3.3 Readout Controllers

Readout Controllers (ROCs) are not part of the TDAQ system, but rather are included separately in each detector subsystem.

Readout Controllers have the main purpose of data collection, buffer management and processing. They are based on an FPGA architecture. This FPGA provides the high-speed serial transceivers (SERDES) for the optical links and manages all kind of communications: both data transfer then Detector Control System (DCS) "slow control" operations. ROC's firmware is still in development: one of the possible approach is to embed a microcontroller, which handles and is responsible for initializing the FPGA. Because all communication is normally routed to or through the FPGA, there must be a failsafe way to reload the FPGA in the event of firmware corruption. A watchdog timer will restart the microcontroller on loss of System Clock, or if any of several FPGA and microcontroller check signals are outside nominal timing windows. This will automatically reload a "golden" version of the FPGA and microcontroller firmware from dedicated SPI memory, providing a known-good DCS connection. DCS commands can then be used to remotely load new software/firmware into the application program memory. A DCS "run" command must be sent to the microcontroller to cause it to switch from golden to application program memory. Readout Controllers in or near the detector will be exposed to a high neutron flux. SRAM based FPGAs are sensitive to radiation induced single-event upset (SEU) in the configuration and application memory. Mu2e Readout Controllers in higher radiation areas will use Microsemi PolarFire series FPGAs which provide on-chip microcontroller and SERDES and a number of features to mitigate SEU, including flash based configuration and ECC protected memory and registers. Commercial integrated circuits can typically tolerate total dose of at least 100 Gy without significant degradation. In the region where the Tracker and Calorimeter ROCs are located, total dose is estimated at 10 Gy/yr.

Data from the digitizers is zero-suppressed, formatted and written to the ROC Data Buffer during the beam spill. Data packets are read from the Data Buffer and transmitted on the optical link during the full accelerator supercycle. The buffer is large enough to hold at least 1 second of ROC output data, and uses ECC memory for SEU mitigation.

1.3.4 Data Transfer Controller

The Mu2e Data Transfer Controller (DTC) [25] collects data from multiple detector Readout Controllers, optionally performing event building and data preprocessing. The DTC module provides an interface between the Mu2e Readout Controller (ROC) modules, and the Trigger and Data Acquisition (TDAQ) servers running the TDAQ online software framework. For Mu2e, the DTC (figure 3.5) is implemented using a commercial PCIe (Peripheral Component Interconnect Express) card located in the TDAQ Server. It is based on the HiTech Global Kintex-7 (HTG-K700) PCI Express expansion card. This card features an eight lane Gen 2 PCI Express interface, a DDR3 SODIMM socket, and a 400 pin FMC connector, all wired to a Xilinx K325T Kintex-7 FPGA. The FMC connector allows the installation of an FMC card with the optical fiber interface. This provides for multi-gigabit serial links for up to six ROC Links, a port for data exchange for hardware event building, and a port for the Command Fan-Out (CFO) interface. Firmware for the DTC's FPGA is based on a modified reference design provided by Xilinx.

The central component of the Mu2e TDAQ system is a commercial 3U server, which manages data collection from the Readout Controllers, Event Building, and Online processin. There are a total of 36 TDAQ servers, occupying four racks in the electronics room. The servers used for pilot system development are Supermicro X10DRD-iTP with dual E5-2680v3 processors and four 8GB ECC DDR3 2133 memory modules.

High-speed serial ports are provided by an adapter module which plugs into the FMC (FPGA Mezzanine Card) connector on the PCIe card. This adapter has eight bidirectional SFP+ (enhanced Small Form-factor Pluggable) ports, and can be used with optical or copper cabling. Six of the ports are used to connect to Readout Controller rings optical links. One port can be used to connect to the Event Building Network to exchange data between DTCs. The last port is used to communicate with the Run Control Host computer.

The DTC receives Heartbeat packets from the Run Control Host. These packets are forwarded on each attached ROC ring link. Data packets from the Readout Controllers are returned on the same links. The DTC multiplexes data from six links into one timeslice which is then transferred to the Server over PCIe, or to other DTCs via the Event Building Network.

1.3.5 Run Control Host

The Run Control Host receives beam status and timing information from the Accelerator Controls network, and operator commands from the remote control room. The Command Fanout (CFO) module in the Run Control Host is responsible for generating and synchronising Heartbeat packets. It sends a Heartbeat control packet for each event window. The CFO contains a set of standard Heartbeat packet templates (normal readout, calibration, no operation, etc.), and a default list mapping these packets to each of the ∼800,000 potential event window periods in a 1.4 second supercycle. The CFO also maintains the System Timestamp which it sends with each Heartbeat packet. The Run Control Host can instruct the CFO to override the default packet on any clock or series of clocks.

1.3.6 CFO - Command Fan-Out

The Command Fan-Out Module (CFO Module) [24] provides an interface between the CFO Host and the DTCs. The CFO is based on the HiTech Global Kintex-7 (HTG-K700) PCI Express expansion card. This card features an eight lane Gen 2 PCI Express interface, a DDR3 SODIMM socket, and a 400 pin FMC connector, all wired to a Xilinx K325T Kintex-7 FPGA. The FMC connector allows the installation of an FMC card with eight SFP+ slots. This provides for multi-gigabit serial links for up to eight DTC Links, and ports for the System Clock and Super Cycle Start inputs. Firmware for the CFO's FPGA is based on a modified version of the DTC code.

The CFO module firmware is based on the Kintex-7 FPGA Targeted Reference Design. This reference design is provided by Xilinx Corporation to allow designs utilising the high bandwidth capabilities of PCI Express, DDR3 memory, and high-speed I/O to be implemented with significantly less design time required. To do this, the reference design provides code that implements the PCI Express Interface, the DDR3 memory interface, and a high bandwidth DMA engine. Two high bandwidth ports are provided to allow user developed code to be connected to the reference design. In the case of the CFO module firmware, the DTC link controller code is connected to the two high bandwidth user ports. One port is dedicated for DTC data, and the other is dedicated to DTC status and slow controls.

A DMA channel provided by the Northwest Logic DMA Back-End Core provides a DMA engine to transfer data between PCI Express and the DDR3 memory on the CFO card. DMA Channel 0 is used to write the clock cycle specific Readout Request information to the Readout Request Information Table in CFO memory.

Once configured, the CFO will issue a Heartbeat Packet for each System Clock cycle ($\mu Bunch$) of a Super Cycle. A System Clock period is 1695 ns. Four bytes of cycle specific data are taken from the Heartbeat Information Table and output along with the timestamp in the Heartbeat Packet. For loopback diagnostic functionality, a given readout link's SERDES can be put into loopback mode. All packets queued for transmission on that link will then appear in the input buffer of the associated readout link.

From a software point of view, all packets are 16 bytes in length. Note that the hardware appends four bytes to the packet before transmitting it over a high speed link. These four extra bytes are then stripped from the packet by the hardware upon reception. The extra bytes are noted in the packet descriptions below as a two byte 8b-10b K/D character header and two CRC bytes. The K/D character header, and the CRC bytes are not included when generating the CRC value. These characters will not appear in the input buffer with received packets when in loopback mode. About 8b-10b codification a more in deep section is dedicated later in this document.

The CFO Instruction Table is stored in memory on the CFO module. Populating the table is done via a DMA transfer to DMA port 0. Each table entry is 64 bits wide and corresponds to one CFO Instruction. The size of the table (corresponding to the number of instructions) is configurable via the CFO Instruction Table Size Register.

The CFO module receives the timestamp preset value from the CFO Host. The interface mirrors the DTC Link interfaces, in that it is an SFP+ socket populated by a multi-mode optical fiber transceiver operating at either 2.0 Gbps or 2.5 Gbps decoded (2.5 Gbps or 3.125 Gbps 8b10b encoded). Of the eight SFP+ channels on the CFO, all are available for connection to DTC links. The CFO also connects to the System Clock and the Super Cycle Start signals. All Readout Request Packets are queued by the CFO until the next positive System Clock edge.

1.4 Timing System

The TDAQ will generate a continuous Mu2e System Clock with frequency of 40 MHz at the Run Control Host Clock Fanout module (CFO). The CFO also receives the RF-cavity 0-crossing marker from the Accelerator. Note that this marker signal is synchronous with the arrival of proton pulses every 1695 ns to Mu2e and is only active through a ∼43 ms spill. There will be at least 100 μs of markers without proton pulses to start each spill. The time between spills is arbitrary, minimum is on order ∼5 ms.

The CFO outputs a "punched" or "encoded" clock indicating the start of the Mu2e event window, which is synchronous with the Accelerator marker during beam ON to 10 ns accuracy - the "punch" or "marker" is a change of the duty cycle of two cycles of the clock to either 25%/75% duty cycle, or 75%/25% duty cycle. These two encodings are alternated so that a lost marker can be identified by the ROCs.

This encoded clock will be fanned out and distributed to outside the cryostat in the detector hall. Because of the grounding requirements, the TDAQ will distribute the signal optically from the TDAQ room to the Detector Hall (i.e. 20 optical fibers). Then the optical can be converted to electrical co-ax (SMA) in the detector hall on detector ground. The detector subsystems are responsible for distribution inside the detector vacuum and further fanout stages as needed (i.e. 18 encoded clock signals to 216 Tracker ROCs and 192 Calorimeter ROCs).

The end of the marker on the encoded clock marks the start of the next event window. Event windows are contiguous in time although detectors may have a "live gate" within an event window. During beam ON, the event windows will have duration 1695 ns, and during beam OFF, the event windows can be different duration but a multiple of the System Clock period. Identifying beam ON versus beam OFF can be done independently per 43.1 ms spill using two configurable timeout parameters such as 100 μs leading into beam ON and 5 μs leading into beam OFF.

The data links to the front ends will be used to send a 16 byte Heartbeat packet describing the next event window before each event window begins. This packet includes an 8 bit TDC value predicting the relative phase of the next event window and the System Clock to better than 1 ns resolution. This packet also provides "live gate" info and a 48 bit Mu2e System Timestamp (no wrap-around for 15 years) labeling the next event window. Heartbeat packets are not transmitted during the period 50 ns before and after the event window marker. The front ends can use the information distributed from the TDAQ however needed (i.e. if 10 ns resolution is enough, the front ends can ignore the 8 bit TDC value). There are at least two ways to recover the System Clock at the front ends. Most FPGAs can recover a clock to better than 200 ps jitter. External to FPGA clock recovery and jitter cleaner circuits can recover the clock to better than 1 ps jitter.

The System Clock is not guaranteed to arrive to all detector ROCs phase aligned. Phase alignment at individual ROCs is accomplished by an adjustable delay at the ROC clock input. Additional alignment is provided by software or firmware calibration and may result in internal timestamp synchronisation offsets. Each ROC generates its own internal high speed digitisation clocks, phase locked to the System Clock. A Clock

Generator on the ROC is programmed via the DCS connection to drive the digitises at any N/M multiple of the System Clock. Each ROC also generates an internal timestamp for timing data within the Event window. This phase alignment and internal timestamp synchronisation offset can be calibrated across detectors using cosmic rays or proton pulses.

To accommodate the new point-to-point topology for the Tracker and Calorimeter, forced by the design decision to go to a single bi-directional VTRx at the ROC, the System Clock and event marker will be transmitted over the serial link running at 4.0 Gbps. The 40 MHz System Clock will be represented by a special clock marker. Both the clock marker and the event marker will be represented by two 8b-10b K-characters, that are only transmitted on a System Clock edge. By only transmitting on System Clock edges, the System Clock can be extracted and event markers can be extracted associated with a System Clock edge. The markers must be received with fixed latency with respect to the source to maintain ROC-to-ROC synchronisation. Fixed latency over the serial links is achieved by removing as much elastic buffering as allowed in the SERDES throughout the data path. Note that after extracting the 40 MHz System Clock at the timestamping front-end ROC, the clock can be scaled up by integer multiples (e.g. 200 MHz) to be used to timestamp data. Accounting for the variable latency of the serial datapath from ROC to ROC is handled by coarse and fine granularity delay offsets implemented at the timestamping front-end. Determining the offset to apply is achieved by loopback to determine the latency of the datapath.

1.4.1 Timestamps

Each Readout Controller generates its own internal timestamp for data within an event window. This internal timestamp counter is driven by the digitisation clock and is reset at the beginning of an event window. It may be 1 or 2 bytes, depending on the resolution of the detector. The digitisation clock frequency is determined by the ROC and can be different across different detectors. In addition to the internal timestamp generated by each ROC, there is a System Timestamp generated by the Command Fanout Module in the Run Control Host. This is a six byte value which increments for each event window. It has a range of at least 15 years. It can be stopped and restarted at any value as long as the new start value is higher than the previous stop value. The System Timestamp can be correlated with actual calendar time, or the high bytes can represent a Run Number, supercycle, etc. The System Timestamp is sent by the CFO to the DTCs at each System Clock. The DTCs broadcast the timestamp to all attached ROCs in a Heartbeat packet. The DTCs also send a System Timestamp as part of Data Request packets, and the ROCs return the System Timestamp in the Data Header packet. Sending the System Timestamp directly to the ROCs for each event window (instead of relying on a timestamp generated from the ROC itself) avoids loss of ROC event synchronisation with the rest of the system as a result of missing or extra decoded event windows. The System Timestamp counter increments for every event window. The event windows are contiguous during a run, whether or not there is beam. This allows readout (e.g., acquisition of calibration or pedestal data) at any time in the accelerator supercycle. The System Timestamp is the only value used for event identification in the TDAQ system. Events are not renumbered following various stages of filtering.

1.5 TDAQ Software: artdaq

The software architecture is based on *artdaq*. This software runs on TDAQ servers and on dedicated control and monitoring computers. artdaq is a toolkit of C++ 2011 libraries and programs for use in the construction of TDAQ systems. It provides functionality that includes the following:

- management of the readout and configuration of the TDAQ hardware. This makes use of experiment-supplied software components.
- routing of data between threads within a process, between different processes, and between different machines, and for assembling complete events from these data.
- encapsulation of the data being routed, and support for experiment-specific raw data formats to provide type-safe data access.
- event analysis and filtering using the art event-processing framework.
- basic control and monitoring applications.
- infrastructure for distributing configuration data to TDAQ processes.

The *artdaq* data acquisition toolkit is used to build the Mu2e TDAQ software system. *artdaq* provides software applications for managing the data flow as well as libraries and applications for encapsulating the data, analyzing the data, and performing other basic data acquisition functions. The core data-flow applications in *artdaq* consist of the following:

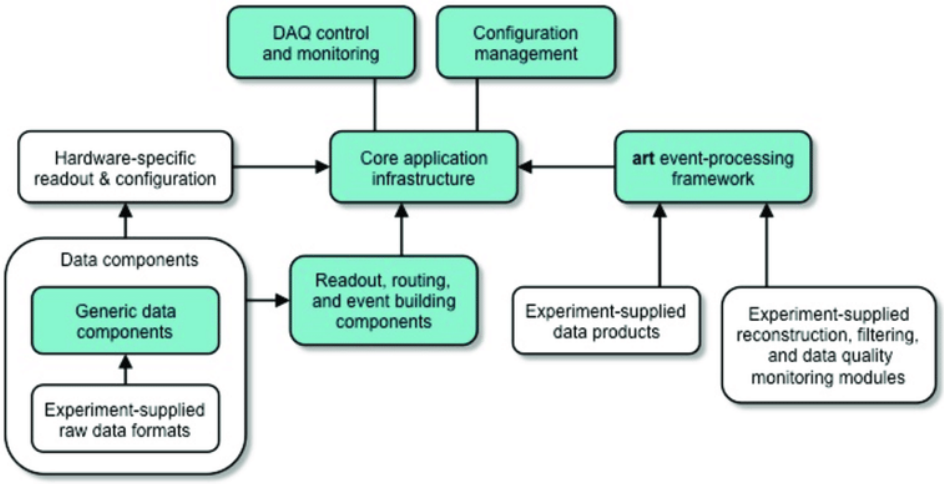

Figure 4: *artdaq* Architecture: core components are shown with a blue background, while experiment-supplied components are shown with a white background.

- BoardReaders that configure and read out hardware modules, and send data fragments to EventBuilders;
- EventBuilders that assemble full events and pass the events to instances of the art analysis framework for reconstruction and filtering;
- Aggregators that organise events in time order, write them to disk, and analyse them to monitor the quality of the data.

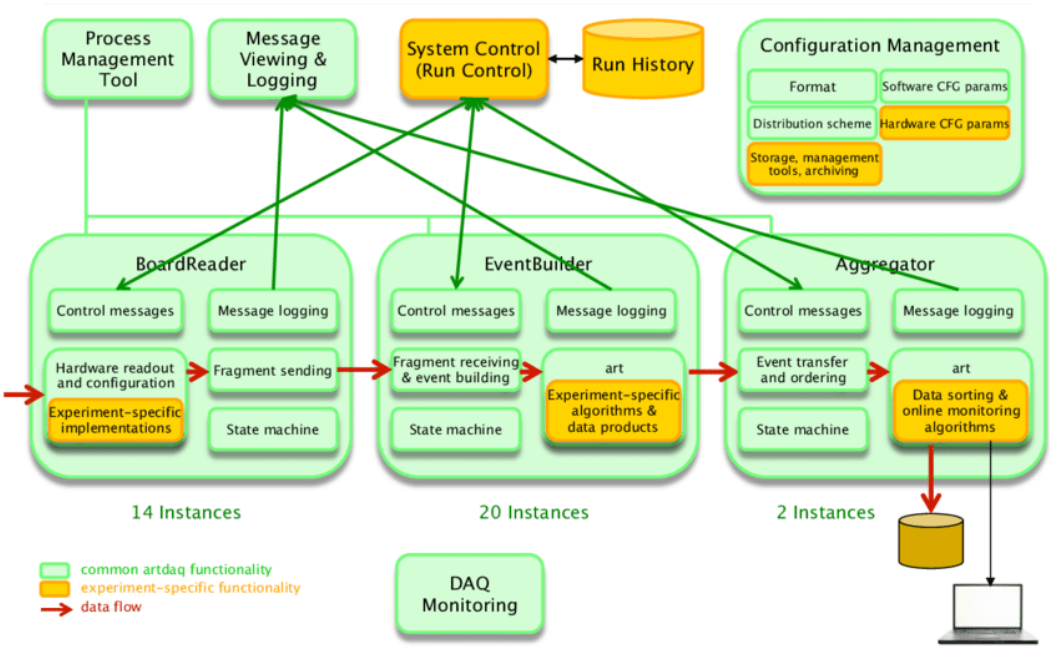

Figure 5: *artdaq* components. Applications and infrastructure components that are shown in green are part of the core *artdaq* toolkit. Components shown in orange are modules that experiments provide to read out their specific hardware and perform their specific analyses and monitoring.

These applications are shown in Figure 5 along with additional components that are part of artdaq. The additional components include infrastructure for sending and receiving control messages, managing the state of individual processes and the full system, logging messages to central loggers and viewers, and the sending and parsing of configuration parameters.

The toolkit is designed to provide core functionality while allowing experiments to customise the hardware readout and event analysis as needed.

Key term are of the TDAQ Software are: online DAQ software, *artdaq*, *art* and *otsdaq*.

The term "online DAQ (data acquisition) software" refers to the software used to monitor, select, and validate physics and calibration data for the experiment. It is easy to creep the scope beyond above. For example, the above often involves the need for some control of the front-end electronics, so the extreme would be for all control of the front-end electronics to go through the online DAQ software. Other scope creeping features might include configuration parameters, configuration change tracking, user access permissions, user preferences, etc.

Acronym for "*art* data acquisition" *artdaq* is a data acquisition toolkit which provides functionality for data transfer, event building, event reconstruction and analysis (using the *art* analysis framework), process management, system and process state behavior, control messaging, local message logging (status and error messages), DAQ process and *art* module configuration, and the writing of event data to disk in ROOT format. In general, the *artdaq* toolkit includes one or many ways to do things, and it is left to the experiment to choose the tools from the toolkit and provide the glue for a complete system.

art is not an acronym, it is an event-processing framework for particle physics experiments, like Mu2e. Experiments use the *art* framework to build programs that process data in a variety of contexts: high-level software filters, online data monitoring, calibration, reconstruction, simulation, and analysis. Mu2e offline uses *art*, Mu2e online uses *artdaq* and thus *art* as well. For example, the Mu2e online trigger decision is made by a set of *art* modules running in the online environment (but primarily developed in the offline environment).

otsdaq is an acronym for "off-the-shelf data acquisition." *ots* for short. It is the online DAQ software framework that Mu2e has chosen. *otsdaq* uses the artdaq DAQ framework under-the-hood to provide data handling flexibility and scalability. In addition, *otsdaq* provides a web interface to configure, control, and monitor the online DAQ software entities. In general, *otsdaq* has chosen the tools from the artdaq toolkit for the experiment, and provided the glue for a coherent experience for all users (shifters, experts, etc.) from Chrome or Firefox.

otsdaq and *artdaq* are developed by the Fermilab Scientific Computing Division and developments are in two directions: server side and web side. About the online DAQ software development, server side is C++. User code is added through plugins (C++ classes inheriting from the appropriate class). Types of Mu2e online DAQ software plugins are:

- Front-end interfaces - code to communicate with an external device, e.g. there's a plugin for the DTC, and for each type of ROC
- art modules - e.g. trigger modules, online monitor modules
- artdaq Fragment Generators - code to decode data and transmit to artdaq event builders
- Data processors - code for custom data handling, e.g. datastream-to-ROOT for Visualizer
- Configuration table handlers - code for custom handling of configuration data, e.g. to output FHiCL, or provide helper-abstraction functions like getVolume() of object with size specified in configuration parameters.

Web side is HTML and JavaScript. User code is added in the form of web-apps through .html files (including the appropriate .js and .css files). Any custom user web-apps for Mu2e is not been generated yet, but the facility is present. For example, overlaying Calorimeter ROC temperature color-coded on a 3-D representation of the detector with slider controls to set thresholds, this would be a custom user web-app.

All data filtering and triggering in the Mu2e TDAQ architecture is done in firmware or software. The production TDAQ will use 36 dual-CPU servers. The online processing system must handle a total rate of 200,000 events per second, an average of 5,600 events per second per server. The *art* analysis framework will be used as the environment in which the online processing algorithms are executed. It provides the infrastructure for running software modules that are provided by experimenters and managing the data that is analyzed and produced by the analysis modules. It has been developed at Fermilab for use in current and future intensity frontier and cosmic frontier experiments, and it is currently used in the offline environments of the Mu2e, NOvA, LBNE, and other experiments. It is also currently used in the TDAQ system of the DarkSide-50 experiment, which is also *artdaq*-based. The use of the same analysis framework online and offline has substantial advantages, most notably the ability for physicists to develop algorithms independently of the full TDAQ system and move them to the online environment when they are ready. Within the TDAQ system, EventBuilder processes handle the starting of *art* threads and the transfer of full events to *art* for analysis. It also handles the configuration of the *art* framework and the analysis modules using the configuration parameters that it receives from Run Control. The same configuration language is used to configure *artdaq* processes as is used to configure *art*.

As part of the software interface to the DTC, a Linux device driver for communicating over the server PCIe bus is being developed. The driver will be responsible for managing the buffers into which the data is written when it is received from the ROC, responding to the interrupts when DMA transfers complete, notifying the user code that data is available, and delivering the data to the user code.

1.5.1 otsdaq

otsdaq is the Ready-to-Use data-acquisition (DAQ) solution aimed at test-beam, detector development, and other rapid-deployment scenarios. As stated earlier, *otsdaq* uses the *artdaq* DAQ framework under-the-hood, providing flexibility and scalability to meet evolving DAQ needs and provides a library of supported front-end boards and firmware modules which implement a custom UDP protocol. Additionally, an integrated Run Control GUI and readout software are provided, preconfigured to communicate with *otsdaq* firmware. *otsdaq* comes as a web page. The ots web desktop environment is your portal to all of the possibilities of *otsdaq*. Briefly, desktop features are: same user on multiple browser tabs, monitors and computers, configurable desktop window icons and folders with access permissions, window layout presets (Global and per user) and window manipulations (Tile, resize, move, minimize, maximize, refresh, close).

Front-end interfaces are plugins that are considered to be the specifics for how (i.e. C++ to write and read) to interface to a device external to *otsdaq*. In particular, *otsdaq* is used to control the DTCs. The DTC Client library is the low level interface code to the TDAQ Data Transfer Controllers. It provides the PCIe interface functionality, and implements handling the packet protocol that the DTCs and readout controllers use in Mu2e. The *otsdaq* front-end interface plugin for the DTC is a wrapper around the DTC Client library and the DTC Board Reader. *otsdaq* presents a State Machine, visible in figure 3.16, that allows to easily configure and run DTCs. Macro Maker is a tool that allows the user to execute front-end interface writes and reads, and build sequences of writes and reads, i.e. macros. Macros can be saved per user or made public for all users. Macro Maker is useful for low level debugging of front-end interfaces, and early development. Macros can be exported to C++ or directly to a target plugin as a FE Macro. FE Macros are C++ member functions of a front-end interface plugin class. The primary utility is that, with no user effort, FE Macros are available through the web-interface - through the FE Macro Test web-app or custom user web-apps. FE Macros have strings or numbers as input and output arguments. The FE Macro Test web app also runs generic private and public macros from Macro Maker. The concept of Macro Maker mode is that anyone (e.g. a firmware developer) who just wants to use front-end interface plugins with FE Macros, or generic macros, could use this simplified mode without tracking configuration changes or using the state machine. A FHiCL parameter file is used to import the configuration. When Macro Maker mode is launched, the state machine is automatically transitioned through to the Configured state.

Another feature is the configuration tree. The configuration tree defines the hierarchical relationship between all entities in the online DAQ system, and all of their parameters. When *otsdaq* is launched, the executables that start are the ones enabled in the configuration tree for that node. Then later, when the state machine transitions to the Configured state, the children of the executables are instantiated based on the parameters defined by the chosen configuration alias. Configuration alias maps to a configuration tree which fully defines the online DAQ configuration (likely, the configuration alias string and the translation to group name and group key gets recorded in the run conditions database). A configuration tree can have multiple roots and multiple branches (as shown on the bottom-right). At the lowest level, *otsdaq* stores configuration data in tables and tracks table changes as versions. The configuration tree is an abstraction extracted from groups of tables. Any entity that needs configuration parameters can have read-only access to the configuration tree API - with calls like getChildren(), getNode(), and getValue() - and access to the table plugins and their helper abstraction functionality.

Other features are the console and the code editor. The console web-app help users exist remotely and remove the need to access the linux terminal. The console core functionality is built on *artdaq* message facility. Messages have labels, line numbers, and severity; they can be filtered, and user preferences are saved per user. Printouts to the terminal, log files, or the web console can be generated from any user plugin code by using the *ots* output macros. The code editor web-app is a tool that allows for editing and viewing of source code and text files. Configurable permission levels give write access or not. The code editor has vertical and horizontal view split, and can spawn multiple browser tabs and windows. The code editor might help developers standardise code format, encourage collaboration, and allow for remote development.

ots uses *artdaq* database as its external database interface. *artdaq* database is a JSON document based database which can be persisted on the filesystem or by *mongodb*. When a new table version is created by *ots*, a new JSON document is created in *artdaq* database. For redundancy and high availability, *mongodb* is used. The approach is to have replica sets which each maintain the same data set, and then a cron job that automates daily backups to a directory tracked by TiBS (Fermilab Core Computing backup/restore service).

Data processing is the primary responsibility of the online DAQ. Mu2e's event window data will be processed through *artdaq* modules. However ots allows for data processor plugins in general (i.e. interfacing to *artdaq* makes use of particular data processor plugins provided by *otsdaq* core functionality). Data processor plugins inherit generic data handler functionality, and can add custom handling beyond that. For example, an aspect of the ots visualization tools make use of specialized data processor plugins that generate ROOT objects that can then be viewed in the web desktop. Users can make ots data processor plugins for any purpose they dream up. When the *artdaq* data processor plugin is used in *otsdaq*, users have access to the flexibility and

scalability of *artdaq*. The *artdaq* data processor plugin instantiates an *artdaq* Board Reader with a Fragment Generator plugin. Based on the configuration of the online DAQ system, the user can also instantiate *artdaq* Event Builders, Dispatchers, and Data Loggers. For Mu2e, there will be a Board Reader for each DTC, one Tracker/Calorimeter Event Builder per server, (each running the trigger algorithm with as many art analyzer processes as fit on the server, around 20), a second-level Event Builder which will integrate CRV data, several Data Loggers on dedicated nodes for writing data to online storage, and several Dispatchers to provide online data quality monitoring. *artdaq* tracks a large number of metrics covering pretty much everything about event rate and dataflow, which can be enabled at the metric plugin level; the user can send a subset of metrics to EPICS, everything to *Ganglia*, and only the most important ones to a file, all at the same time.

2 Internship Tasks

During the internship, I have mainly worked on the software development of *otsdaq*. The developments of *otsdaq* are in two directions:

- Server side: User code is added through plugins (C++ classes inheriting from the appropriate class).
- Web side: The user can design custom web-apps to introduce further functionalities.

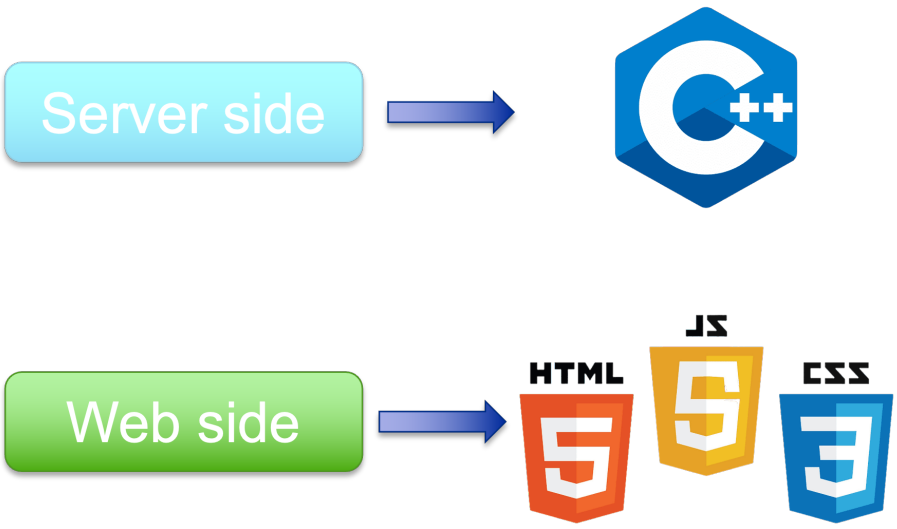

Figure 6: *otsdaq* code base structure.

The idea behind *otsdaq* is to encapsulate one or more instances of *artdaq* and allows the users to interact with them through a user-friendly web-app. As already explained in the introduction, both *artdaq* and *otsdaq* are are open source and developed by the Fermilab Scientific Computing Division. In the past, each Fermilab's experiment used to develop its own platform for data acquisition, there was no standard. Therefor, they decided to develop a unique framework which could be extended with further functionalities based on the requirements of the experiment. In this way, each new experiment can leverage on what has been already developed by past experiment without starting every time from scratch, it is easy to understand that this approach aims to have more robust and less error prone software.

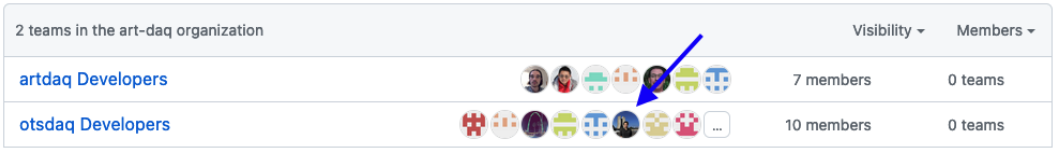

Figure 7: Me listed as one of the contributors of otsdaq.

In this section I am going to explain the main task on which I work on and how they helped TDAQ subsystem.

2.1 Vertical Slice Test

A vertical slice test is a top to bottom, fully implemented and tested piece of functionality. Instead of testing the whole architecture, which could be really expensive in terms of complexity, we focus only on all the components involved in the architecture without taking in consideration how they scale, but just considering a single instance. This test is really important in order to understand if each component and the designed protocol of communication are working properly.

In our case, the vertical slice test consists of data transmission between the detector and the DAQ, which is composed by:

- Data request, starting from the server;
- Data reply, coming from the ROC;

Figure 8: Components involved in the vertical slice test.

- Check if the response is the one expected.

The module of *ots* used for the test is the **FE Macros**, which allows the user to interface with the electronic boards. The macro implemented is called "buffer test", which can send a request for a user-specified number of events and wait for the respective responses. The response has been encoded in JSON format which is human readable and easy to understand allowing the users to check if the various headers of the protocol and payload are the expected one.

The test was successful, and we were able to send a request for up to 2300 events per time.

```
Read 1: Events returned by the DTC: 1
Request event tag:         0x0011
Response event tag:         0x0001
"DTC_EventHeader": {
        "inclusive_event_byte_count": 712,
        "event_tag_low": 1,
        "event_tag_high": 0,
        "num_dtcs": 1,
        "event_mode": 0xfffffffffff,
        "dtc_mac": 0,
        "partition_id": 0,
        "evb_mode": 0,
        "evb_id": 0,
        "evb_status": 0,
        "emtdc": 1
}
Subevents count: 1
Subevent [0]:
"DTC_SubEventHeader": {
        "inclusive_subevent_byte_count": 68
        "event_tag_low": 1,
        "event_tag_high": 0,
        "num_rocs": 1,
        "event_mode": 0xfffffffffff,
        "dtc_mac": 0,
        "partition_id": 0,
        "evb_mode": 0,
        "source_dtc_id": 0,
        "link0_status": 0,
        "link1_status": 0,
        "link2_status": 0,
        "link3_status": 0,
        "link4_status": 0,
        "link5_status": 0,
        "emtdc": 1
}
Number of Data Block: 1
Data block [0]:
"DataHeaderPacket": {
        "byteCount": 0x290,
        "isValid": 1,
        "subsystemID": 0x0,
        "linkID": 0,
        "packetType": 5,
        "hopCount": 0x0,
        "packetCount": 40,
        "timestamp": 1,
        "status": 85,
        "packetVersion": 0,
        "DTC ID": 0,
        "evbMode": 0x0
}
Data payload:
        0x00a0
        0x00a1
        0x00a2
        0x00a3
        0x00a4
        0x00a5
        0x00a6
        0x00a7
        0x00a8
        0x00a9
        0x00aa
        0x00ab
        0x00ac
```

Figure 9: Example of the response returned by a buffer test. Note that the payload contains a counter because we are in the hardware development phase of the boards.

Thanks to this macro, we discovered that if we try to ask for more events the boards start sending back timeout signal, which is not the expected behaviour and it needs to be deeply explored by the subsystem team that is working on the development of the ROCs.

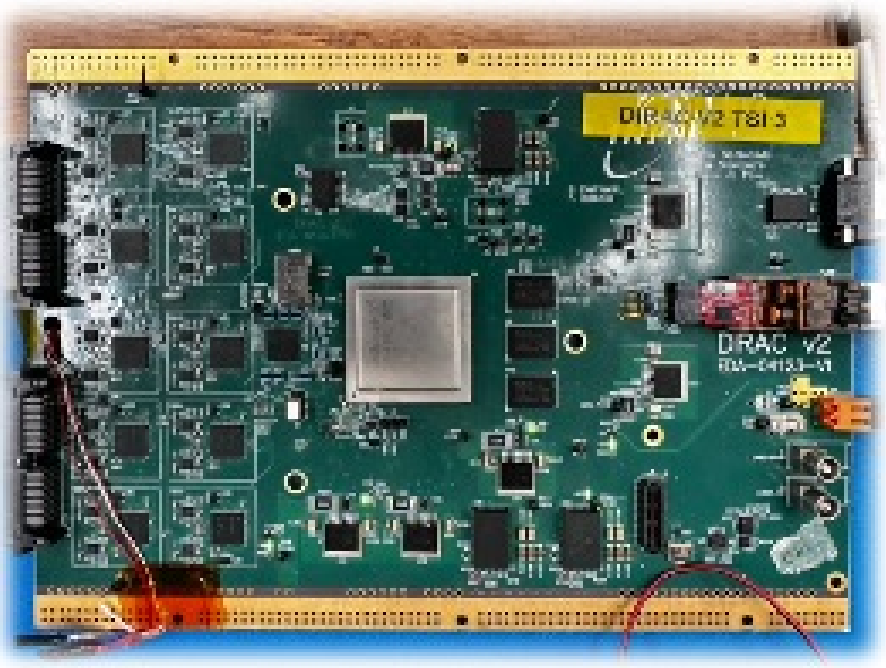

Figure 10: DIRAC, developed by the INFN of Pisa, used during the test.

2.2 FEMacro History

One the functionalities of *otsdaq* I have implemented is the history view of the **FE Macros** module. The development team was pushing a lot for this functionality, because it allows the user to quickly check the last command run with the respective parameters. Having this view makes life easier during the debugging phase, because we can check which combination of commands raise the error and being able to reproduce it is the first step to solve it.

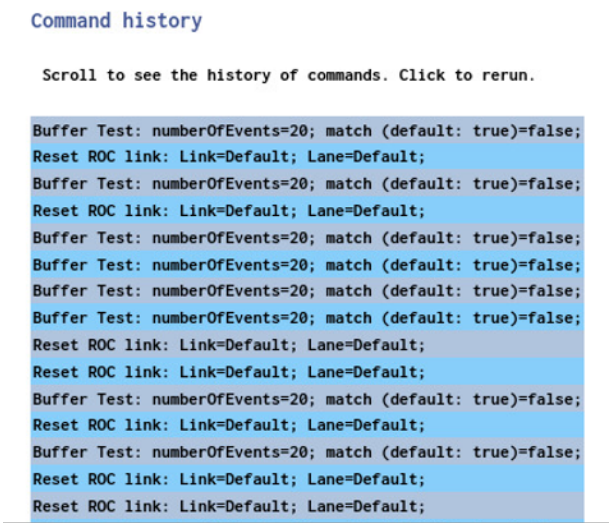

Figure 11: History view of the FE macros module.

The history is stored on the server side as file to keep it consistent across different sessions of the user. There is a history file for each user and they are handled independently, in this way users can also share the history's file among them to test different solutions to a given sequence of commands. The user can also clear the history by pressing a button of the UI.

2.3 Timing Synchronisation

As explained in the introduction, the clock synchronisation among all the boards is fundamental to bind an event to a certain timestamp. The CFO spreads the clock first to the Data Transfer Controller (DTC) boards.

The DTCs are configured in a chain: the first one extracts the 200 MHz clock and passes it to the next one. To prevent jitter accumulation, all the DTCs are also receiving a 200 MHz Reference Clock from the RTF module.

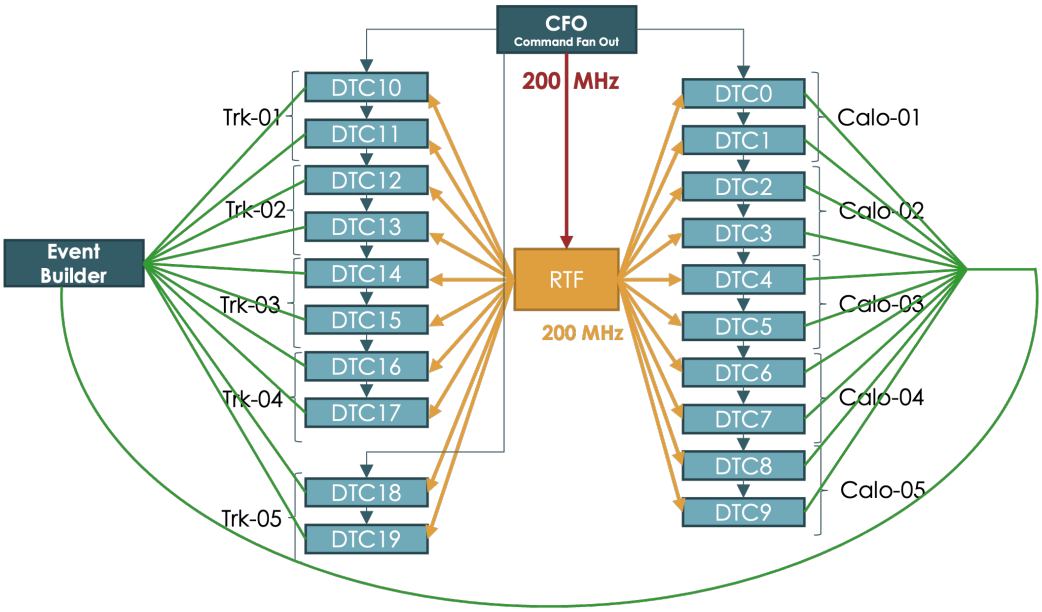

Figure 12: Snapshot of the TDAQ architecture deployed in DAQ room. We can see how tricky the Event Window synchronisation can be.

The various ROCs involved in the architecture can have different clocks because the signal travel through different number of boards and through fibers or cables of different length.

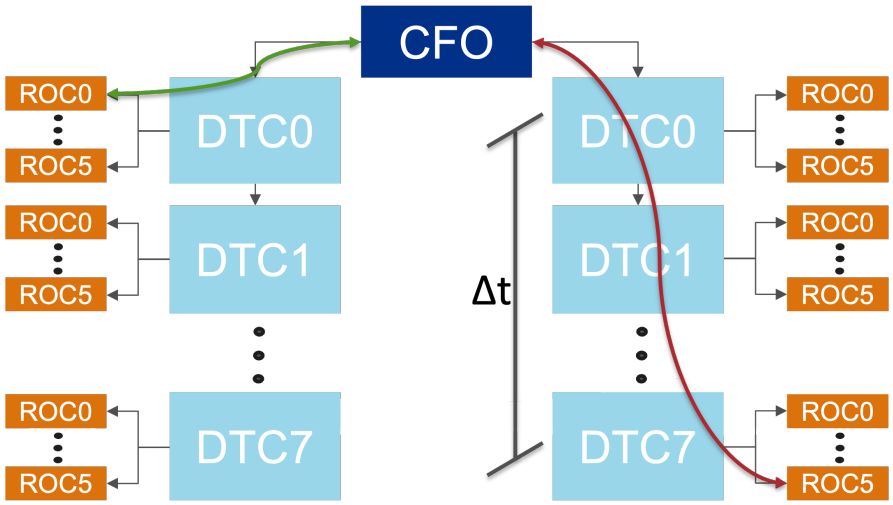

Figure 13: Example of ROCs with different clock.

To line up Event Windows the approach is to delay each front-end to match front-end with longest latency. To calculate the delay to introduce in each board we use the loopback signal, so we can determine the round-trip time (RTT) by sending a marker to the ROCs and measure the time which the marker spend to come back to the CFO. We can repeat this operation many times to compute an average RTT. Once we know the timing path latency for each front-end, we can calculate each front-end's delay offset:

$$\text{delay offset} = \text{longest timing path latency} – \text{individual timing path latency}$$

The loopback test is performed during the "start" phase of the state machine, and it is divided in steps. At each step:

Figure 14: Scheme of the RTT computation.

- DTCs prepare the path which will be follow by the marker to reach the target ROC of the current step.
- CFO sends the markers on its links.

We assume to have a full topology, so 8 DTCs for each chain and 6 ROCs for each DTC, this allows an easier configuration where the user has only to specify the number of chain and the links of the CFO involved.

I proposed to use the same mechanism to create a new mode, which I call "Discovery mode", which can be use to asses the actual topology of the TDAQ deployed. By sending the marker across the topology, we can trace which ROCs are sending back the responses. At the end of the process the system gives a detailed report about which ROCs are reachable and which are not. This report can be used to check if the actual topology is equal to the expected one. In this way, it's easier to spot if *ots* is properly configured or if some component is not working. To understand if ROC is present or we send a marker to each link and we wait for an echo reply, if we wait for too long the CFO rise a timeout and we record the link as dead. Unfortunately, the CFO framework was not ready to handle timeout of the **loopback signal** so I was not able to test the new mode, but it is already implemented in software so I assume that it will be eventually test.

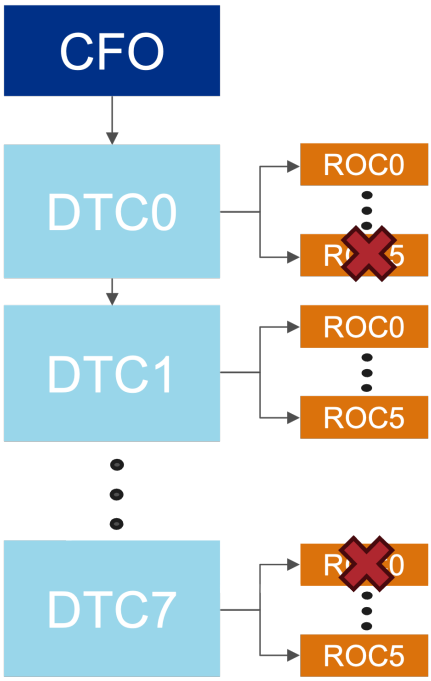

Figure 15: Example scheme of the Discovery mode.

2.4 FEMacro Sequence

One of my last tasks aim to help the working-groups who are focusing on the hardware development of the ROCs. They frequently update their boards by adding new registers or changing configuration procedure. Therefore, they need new macros every time they change the design of the boards to check if the new design is working properly. Writing new macros for each new change is not the best approach, so we decide to allow the user to build their own configuration sequences by chaining together simple macros. Once they find a sequence which is working fine it will be easily converted in a new atomic macro.

The sequence of commands is stored similarly to the history file, in this case each sequence is stored on the server side as a single file. The various working groups can exchange the various sequences among them and easily work together. The sequence is composed by FE Macros with the respective parameter, the user can run the whole sequence in an atomic way or to go through the sequence step-by-step. This last option could be

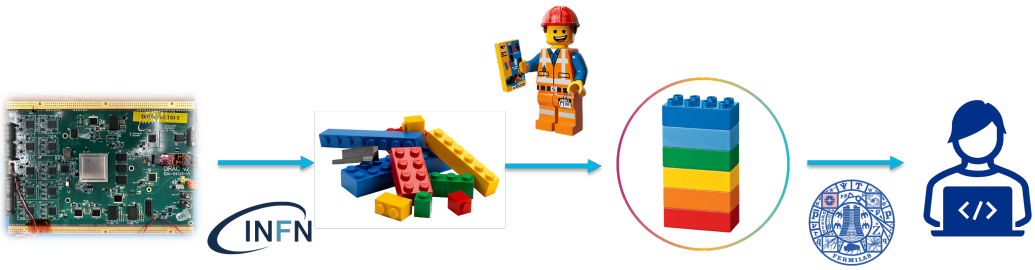

Figure 16: Scheme of the development process.

useful during debugging. Between the macros, the user can also add delays which may be helpful to leave time to the boards to run internal configuration procedures by itself. The result of a sequence is shown as a list of all the results of each step.

2.5 Search bar

One minor task which I work on is the implementation of a search bar in the tree-view configuration menu. *otsdaq* is known to be difficult to configure, because it is really customizable so there could be many different configuration fields. Therefore, looking for a specific field could be difficult. In order to speed up the configuration of *otsdaq* I add a search bar to the GUI which highlight the fields containing the specified key word.

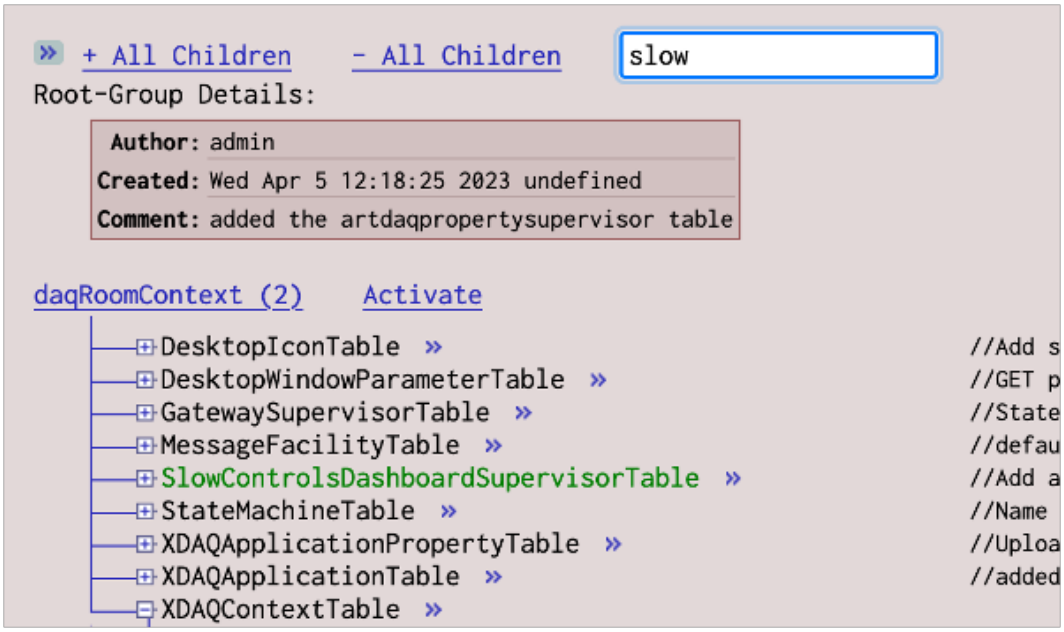

Figure 17: Configuration tree-view with the search bar.

3 Conclusion

My experience with the Mu2e TDAQ team was incredible. I have learnt a lot, not only from software point of view but also from the hardware side and how it is important to have an all around knowledge of all the components of the experiment in order to design the most efficient software possible. Every component of the team has been always available to spend time with me explaining the details of each specific module or component. In addition to the technical skills, I have also trained few soft skills such as time organisation of multiple projects and presentation skills, really important when you have to show to all the team the progresses achieved during the week. It was really interesting to work in a team composed by members with really different background and years of experience. I appreciate the support of Professor Mambelli in all the organisational stuff of the summer school and a special thanks goes to my supervisor, Micol Rigatti, who does not only supported me, but she also trusted me by giving to me roles of responsibility in the design and development of the TDAQ. Can't wait to see the Mu2e experiment up and running.

References

[1] Mu2e experiment. `https://mu2e.fnal.gov/`.