Available on CMS information server **CMS IN -2021/003**

CMS Internal Note

The content of this note is intended for CMS internal use and distribution only

26 May 2021

Setup and Operation of a GLIB-Based DAQ System for CMS GEM Detectors

S. Butalla and M. Hohlmann

Abstract

With the GE2/1 gas electron multiplier detector recently approved for mass production, the electronics integration effort for software and firmware development is complete. With the sparsity of available CTP7 advanced mezzanine cards, we present the setup and operation of an alternative backend advanced mezzanine card for use with electronics integration test stands. Namely, we present the setup procedure and operation of a Gigabit Link Interface Board based data acquisition system for use with the GE2/1 detector. First, we briefly review the frontend and backend electronics for the GE2/1 detector. Next, we discuss instrumenting the GEM electronics board with the frontend electronics. We then provide the procedure for installing and configuring the software and firmware for the data acquisition system. Finally, we discuss the connectivity testing and calibration routines using the `CMS-GEM-DAQ-Project` software for the GE2/1 GEM detector.

1 Introduction

The High Luminosity Upgrade of the LHC (HL-LHC) is set to increase the integrated luminosity by a factor of five [1]. With this increase in luminosity, the muon trigger rates will increase in the forward region of the experiment. The Phase-2 Muon System Upgrade of the CMS experiment is underway to help cope with this increase in muon trigger rates by increasing the redundancy of the muon system [1]. Recently approved for mass production, the GE2/1 gas electron multiplier (GEM) detector has transitioned from the prototyping phase to the final version.

Electronics integration is paramount to nominal and reliable performance of these detectors—we need to ensure that the data acquisition (DAQ) system can communicate with the frontend electronics, and also that they are calibrated. With the limited supply of Calorimeter Trigger Processing (CTP7) Cards, we present an alternative DAQ system setup using a Gigabit Link Interface Board (GLIB) to function as the backend advanced mezzanine card (AMC). In the next sections, we provide an overview of the GE2/1 detector system, and we discuss both the frontend and backend electronics of the GE2/1 GEM detector. We then cover the setup of a GLIB-based DAQ system, and the operation of an electronics integration test stand using the `CMS-GEM-DAQ-Project` software.

2 The GE2/1 GEM Detector

The GE2/1 GEM detector is a composite detector made of four trapezoidal modules with an opening angle of 20°, and has an active area that covers 1.45 m^2. The back GE2/1 detector is made up of independent modules, named M1-M4, where the smallest module is the M1 module, and the largest is the M4. The front chamber is made up of modules M5-M8, where, keeping with the same convention, the M5 module is the smallest, and the M8 is the largest (see Fig. 1). Here, we denote the GE2/1 front chamber as the detector that is closest to (i.e., facing) the interaction point. The front and back modules, when mounted in the superchamber configuration, will overlap such that there will redundant areas where both chambers overlap, and the gaps between the front and back chambers will minimize the acceptance (see Fig. 1). The front and back chambers will be mounted together to form a superchamber, which will be installed at the second muon station in the endcap of the CMS experiment (see Fig. 2). Seventy-two superchambers will be constructed in total, with 36 superchambers per endcap.

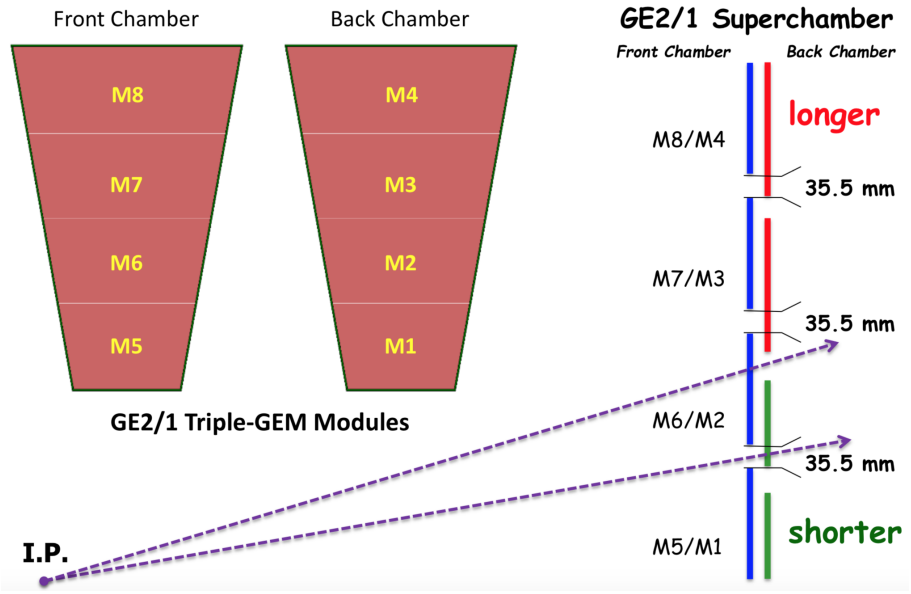


Figure 1: A schematic of the front and back GE2/1 detectors, and the superchamber configuration [1]. Note the configuration such that there are no gaps in the active area of the superchamber.

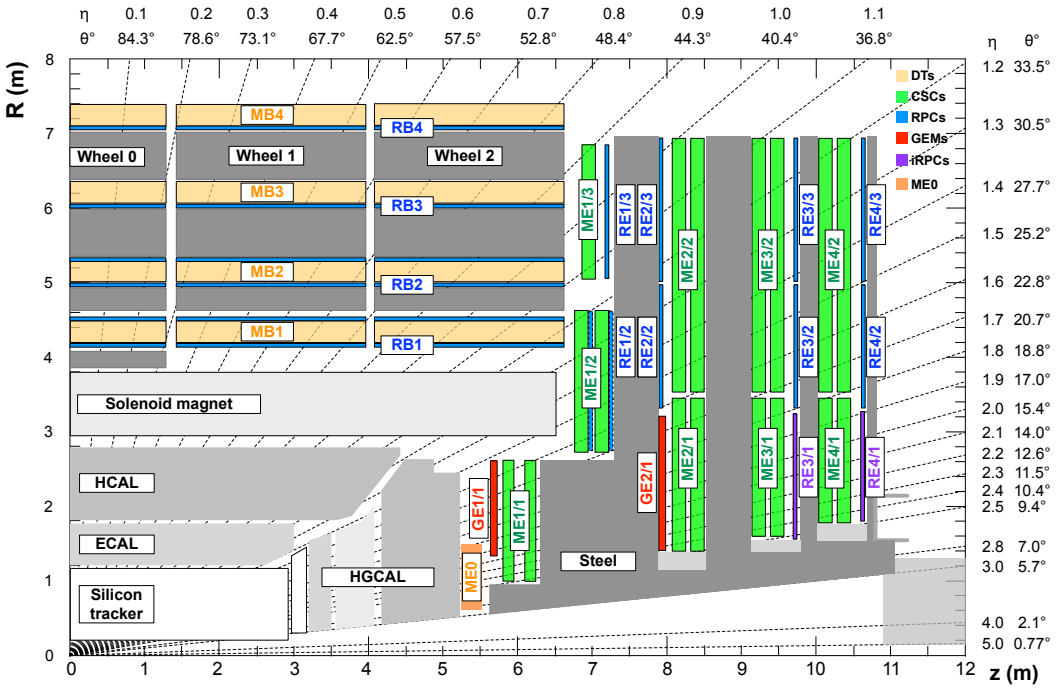


Figure 2: A schematic of a quadrant of the CMS experiment [1].

3 Frontend Electronics

This section provides an abridged overview of the essential frontend electronics used with the GE2/1 GEM detector. While it is not meant to cover every detail of the hardware, the most salient points necessary for test stand operation are included. The curious reader is referred to references cited in each section for more information.

3.1 The GEM Electronics Board

The GEM electronics board (GEB) is the base for all of the frontend electronics, and serves as an interface by routing the signals between the optohybrid board (see Sec. 3.2) and the 12 VFAT3 hybrid cards (see Sec. 3.3), and also distributing the power to all of the components on the GEB. Low voltage is provided to the GEB via the power standoffs (terminals) on the board, and is distributed by 5 FEASTMP_CLP (colloquially referenced as simply the *FEAST*) DC-DC converters (see Sec. 3.4). Each GE2/1 module has its own GEB, so a total of 8 different varieties of these PCBs will be produced. In Fig. 3, we see an example of a GE2/1 back chamber with the GEB fixed on the M1 module, and in Fig. 4, we see a closeup of a single GEB board sans frontend electronics.

Figure 3: A photo of the first fully assembled GE2/1 chamber [2]. Note the un-instrumented GEB board fixed to the M1 (smallest) module of this GE2/1 back chamber.

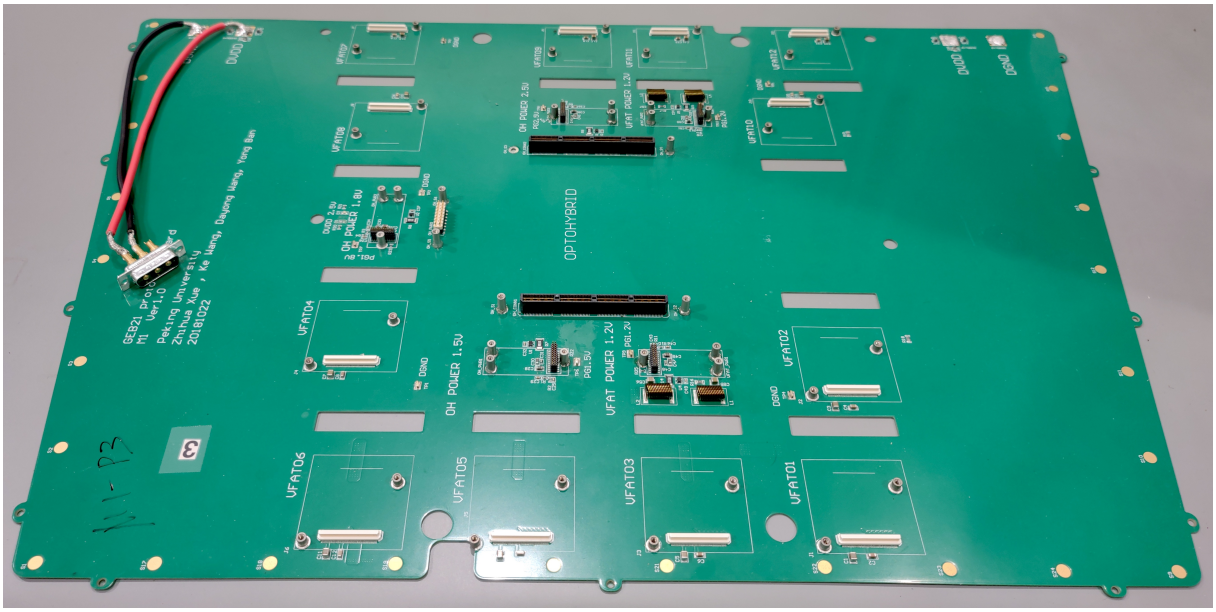


Figure 4: An example of an un-instrumented GEB.

3.2 The Optohybrid

The Optohybrid (OH) provides a means of communicating with the Very Forward ATLAS and TOTEM (VFAT3) application specific integrated circuit (ASIC) hybrid cards mounted on the GEB and the backend electronics. Each module of the GE2/1 will have its own OH board. This OH board was redesigned from the previous incarnation for the GE1/1 GEM detector to better accommodate the needs of the modular GE2/1 GEM detector. It features an Artix-7 [3] field programmable gate array (FPGA) and two gigabit transfer (GBTx) ASICs [4], which communicate via the Versatile Link Transmitter/Receiver (VTRx) and the Versatile Link Dual Transmitter (VTTx) (see Fig. 5 below). The two VTTXs serve as trigger paths for the GEM and cathode strip chamber (CSC) triggers, and the VTRx is a transceiver (bidirectional), used for communication between the backend and frontend electronics. The GBT Slow Control ASIC (GBT-SCA) is dedicated solely to slow control applications (see [5]).

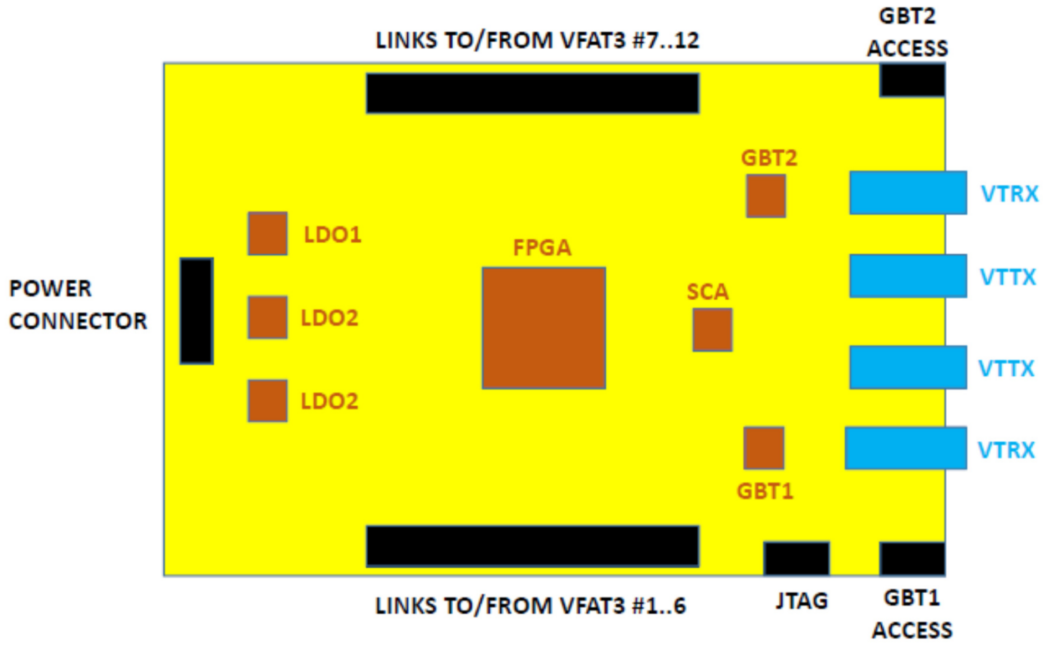


Figure 5: Schematic of the OH [6].

3.2.1 GBTx

The GBTx is a radiation-hard ASIC that interfaces with the FPGA on the optohybrid and allows for three paths of communication on a single optical link: slow control (SC), timing and trigger control (TTC), and data acquisition (DAQ) [4]. This chip interfaces with an LC-LC duplex LC-LC, allowing for bidirectional communication, which can provide both data transmission/monitoring and communication. It is a versatile chip which allows for many modes of operation [4] and can be operated with both commercial, off-the-shelf components, and radiation hard electrical components which allow for it's use in the harsh environment of the LHC.

The first GBT (GBT0 in the software; GBT1 in the diagram in Fig. 5) communicates with VFATs 1-6 and also the GBT-SCA. If communication is not established with this GBT, communication will not be able to be established with GBT1[1)]. GBT1 (GBT2 in the diagram in Fig. 5), communicates with VFATs 7-12. The VTRx transceivers connect these GBTs to the backend electronics.

[1)] This is known as the GBT not being "locked." One can manually check this by running the `gem_reg.py` script and checking the GBT statuses, or, more easily, one can look at the monitored current on the low voltage (LV) power supply: if the current is less than 1.2 A with 6.5 V applied, the first (or second) GBT is not locked. A common reason for this is that the fibers are dirty or they are incorrectly cabled.

3.3 The VFAT3 Hybrid Card

The Very Forward ATLAS and TOTEM (VFAT3) ASIC hybrid card [7] is responsible for integrating and processing the signals induced on the readout (RO) strips (see Fig. 6 for an image of the top and bottom of the hybrid card).

(a) Picture of the top of the VFAT card.

(b) Picture of the bottom of the VFAT card.

Figure 6: The top and bottom of the VFAT3 hybrid card.

This chip has 128 channels, all of which have their own integrating and shaping circuit, and can be individually calibrated to ensure uniformity across the chip via the calibration, bias, and monitoring (CBM) unit. This ASIC also has two SRAM chips that can store signal information. The block diagram of the ASIC is reproduced from [7] below in Fig. 7.

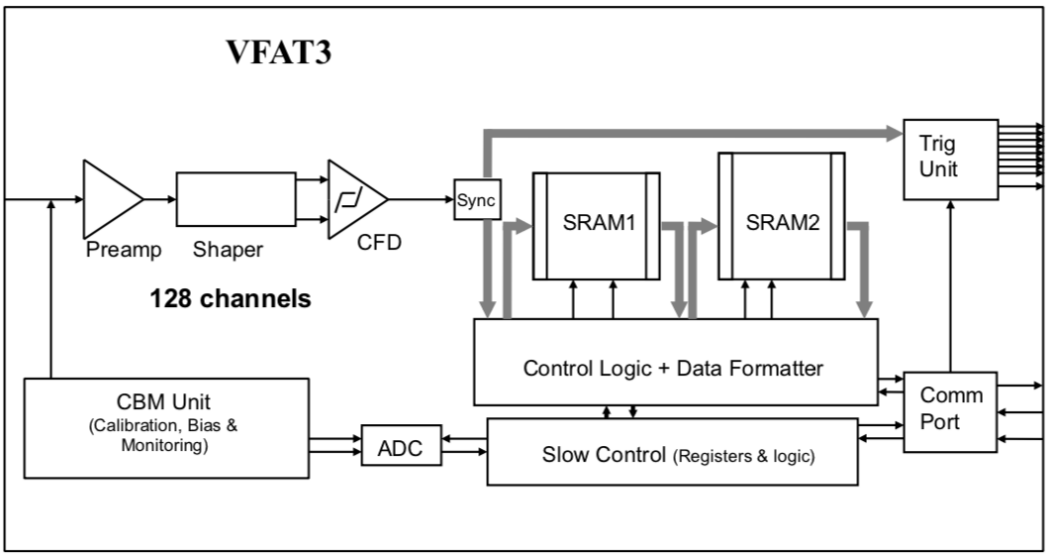


Figure 7: Block diagram of the VFAT3 ASIC [7].

3.3.1 Registers

Below in Tab. 1, we list the programmable registers. Some of the nominal register values are pulled from the CMS production database (outlined in Sec. 7.3), and some are found by the DAC scans (outlined in Sec. 7.4).

Table 1: Programmable Registers of the VFAT3

Register	Description
CFG_IREF	Reference current generated by the digital-to-analog (DAC) converters after the bandgap circuit
CFD_Bias1	Constant Fraction Discriminator (CFD) bias current 1
CFD_Bias2	CFD bias current 2
CFD_ThZCC	Zero-crossing comparator (ZCC) bias current
CFD_ThArm	Arming DAC comparator bias current
CFD_Hyst	Hysteresis DAC bias current
CFG_BIAS_PRE_I_BIT	Preamplifier bias input transistor bias current
CFG_BIAS_PRE_I_BSF	Preamplifier bias source follower bias current
CFG_BIAS_PRE_I_BLCC	Preamplifier bias leakage compensation bias current
CFG_BIAS_PRE_VREF	Preamplifier reference voltage
CFG_BIAS_SH_I_BFCAS	Shaper folded cascode bias current
CFG_BIAS_SH_I_BDIFF	Shaper input pair bias current
CFG_BIAS_SD_I_BDIFF	SD input pair bias current
CFG_BIAS_SD_I_BFCAS	SD folded cascode bias current
CFG_BIAS_SD_I_BSF	SD source follower bias current

3.4 The FEASTMP_CLP DC-DC Converter

In order to transmit the appropriate voltage to the correct components on the GEB board, the FEASTMP_CLP [8] (or just *FEAST* for short) is used to convert the applied voltage to the power terminals on the GEB to the correct voltage for the VFATs and the optohybrid. For all GEBs for the GE2/1 system, there are five FEASTs: (2) 1.2 V FEASTs for powering the VFATs, (1) 1.5 V FEAST, (1) 1.8 V FEAST, and (1) 2.5 V FEAST, the last three powering the OH. The FEAST is a radiation-hard component and has many protection mechanisms employed in its design (e.g., over-current and temperature protection, shielding to reduce interference with the data transmission on the GEB, a four amp load capacity, etc.) [8].

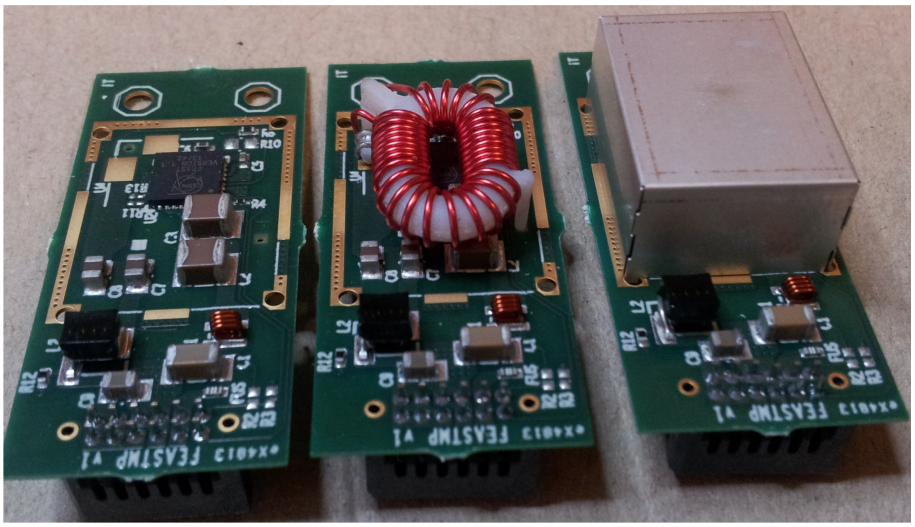


Figure 8: Picture of the FEAST with the shielding and it's main inductor removed [8].

4 Backend Electronics

This section provides an overview of the backend electronics. Here, we discuss the Aspiro 2U power supply, the MicroTCA© crate, the NAT-MicroTCA Carrier HUB (NAT-MCH), the GLIB AMC, and the CAEN SY5527 mainframe, as well as the CAEN A2519 low voltage (LV) and A515 high voltage (HV) boards. See Fig. 9 below for a diagram of the rack which houses all of the backend electronics.

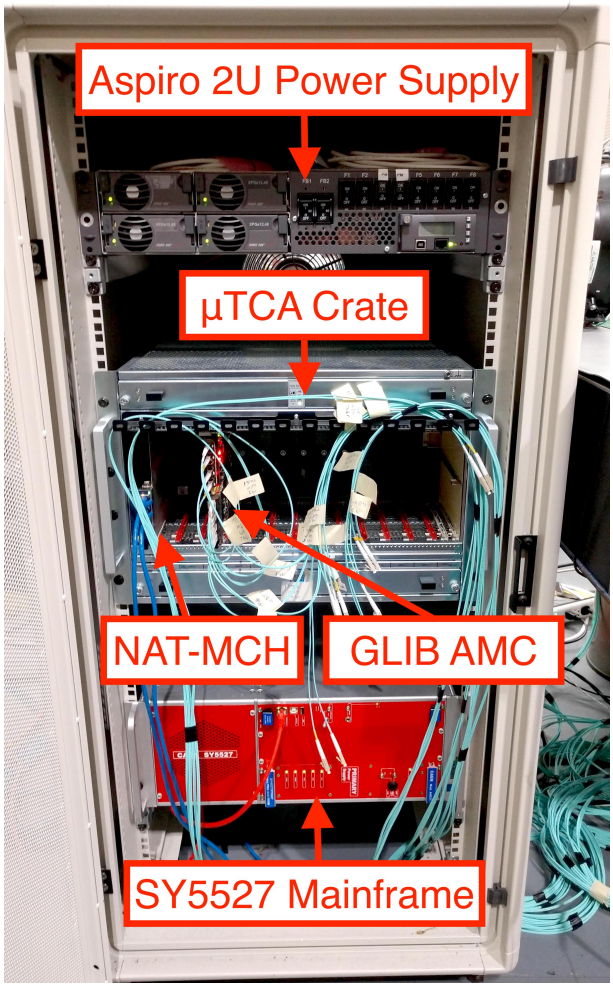


Figure 9: A picture of the rack at Florida Tech with all of the backend components labeled.

4.1 The Aspiro 2U Power Supply

The Aspiro 2U power supply [11] is used to power the MicroTCA crate. This power supply has a maximum current output of 60A, and can provide 100-120/200-240 single-phase AC, as well as 230/400 triple-phase AC. One +48 V line and one -48 V line are connected from the power supply to a power module in the μTCA crate to provide power to the MCH and the GLIB.

4.2 The MicroTCA© Crate

MicroTCA© (also written μTCA©) architecture is a computing standard which facilitates the use of advanced mezzanine cards (AMCs). This standard essentially defines the system and architecture in which the AMCs are placed (i.e., the card cage), the cooling system, management/communication of the AMCs, and power distribution [12]. The MicroTCA.4 shelf at Florida Tech has 12 AMC slots and two full-size MCH slots (see [13] for more information). In order to communicate with the AMCs in the μTCA© crate, the NAT-MCH must used.

4.3 The N.A.T.-MicroTCA© Carrier HUB

The NAT-MCH [14] is essentially a managed switch, and functions as a manager between the μTCA© crate (and the AMCs within the crate) and the DAQ PC. It allows for communication between the DAQ PC and up to 12 AMCs housed in the μTCA© crate. The MCH, just like a managed switch, has an IP address so that one can communicate and configure the MCH via the DAQ PC. Secure Shell (`ssh`) communication is disabled by default, so once the IP address for the MCH is set or is known, one can use the `telnet` protocol to communicate with the MCH. I.e., run

```
telnet 192.168.X.YY
```

to access the MCH. The commands for the MCH's Command Line Interface (CLI) are listed in [14].

4.4 The GLIB AMC

The gigabit link interface board is a backend AMC that serves as the interface between the frontend electronics and the DAQ PC. Figure 10 below is reproduced from [9] and displays a schematic of the link system on the board.

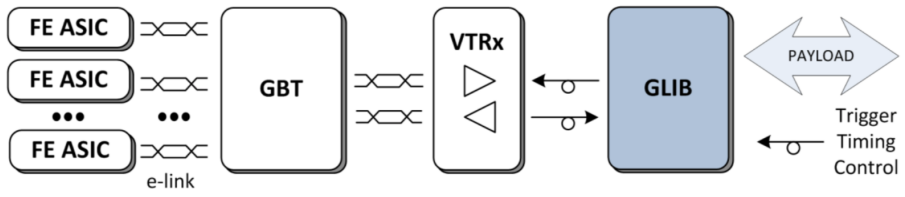


Figure 10: Schematic of the link system with the GLIB [9].

The GLIB has four small form-factor pluggable (SFP) sockets built-in on the AMC. There is an option to expand the number of SFPs by adding up to two AMC extensions cards which are connected to the GLIB via FMC connectors, and have four SFP ports a piece. The GLIB at Florida Tech uses Avago AFBR-57R5APZ SFP transceivers [10] and uses LC to LC OM3 10 Gigabit Multi-Mode Duplex 50/125 LSZH fiber optic cables for communication between the OH and the GLIB. The fibers for the first link are mapped to the top two SFPs on the GLIB, and the fibers for the second link are mapped to the bottom two SFPs[2)]. The GLIB can be operated with two separate trigger configurations when the AMC extension card is installed: (1) Each of the two OHs can have the GEM trigger links connected, or (2) one OH can have both the GEM and CSC trigger links connected. For diagrams and a more detailed discussion, see Sec. 5.2.1.

4.5 Power Supply

Low voltage and high voltage (HV) are provided by two different boards, housed in a CAEN SY5527 mainframe [15]. In the following subsections, we discuss the LV and HV boards, and the electrical connections to the GEBs and the modules of a GE2/1 detector. For information on interfacing and controlling the power supply, see Sec. 7.1.

4.5.1 Low Voltage

The low voltage power supply is provided by a CAEN A2519 low voltage board [16]. This board is equipped with eight floating channels, which allow for on-load grounding for noise reduction [16]. This LV board has a voltage range of 5 to 15 volts, with a maximum current output of 5 A. The board also features a connection for sense wires, which allows for monitoring the voltage at the load, which, in turn, allows the system to compensate for the voltage drop over the power cable.

The setup at Florida Tech powers four GEBs via an Amphenol D-subminiature (D-sub) eight pin female housing plug (part number: DC8W8SA00LF [17]), along with eight, male, 12 AWG D-sub pins (part number: 8638PPS2005LF [18]), which is connected to one of the LV output connectors on the LV board.

The connection to the GEB is made via an Amphenol, three pin, female D-sub housing plug (part number:

[2)] Note that if an AMC extension is used to support trigger links, the mapping is reverse, i.e., the two bottom most SFPs are for the first link and the top two SFPs are for the second link.

DA3W3SA00LF [19]), instrumented with two Amphenol socket pins (part number: 8638PSS2005LF [20]), from the power supply. The connector from the GEB side to the power supply side is made via an Amphenol 3 position D-sub housing plug (part number: DA3W3PA00LF [21]) instrumented with two, 12 AWG pins [18]. The physical connection to the GEB is made by lugs crimped onto the two wires from the male connector, which are then screwed into the power standoffs on the GEB.

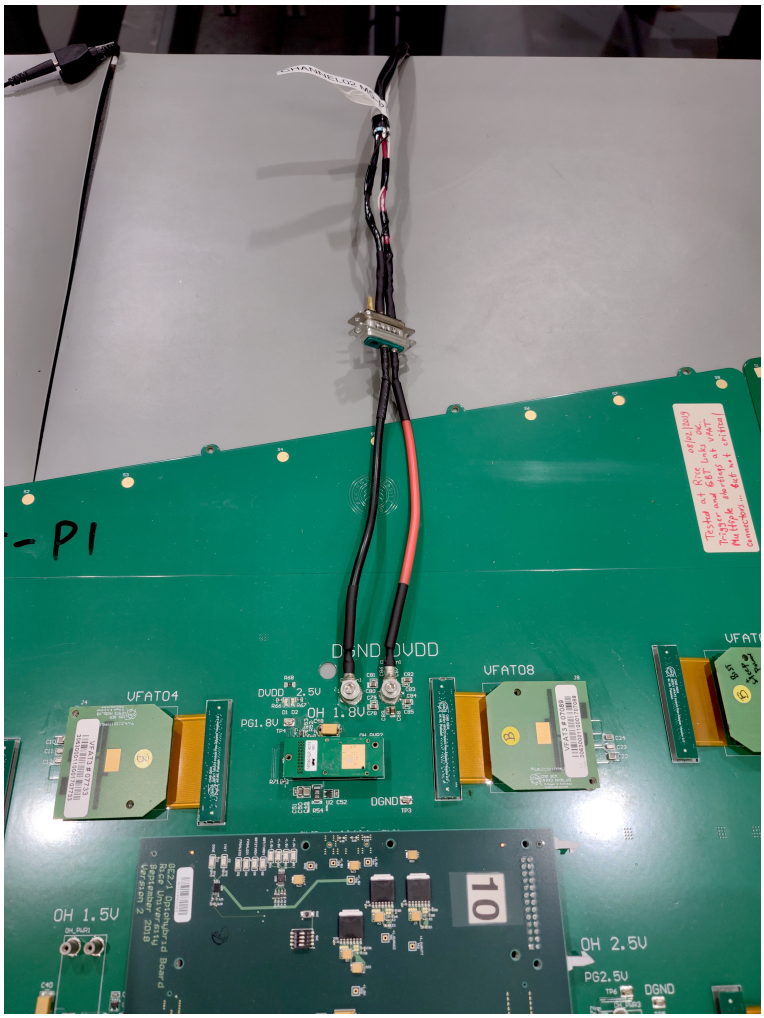


Figure 11: Low voltage connection from the A2519 to a GEB.

The sense wire connection is made via an Amphenol, 26 pin, male D-sub connector (part number: 10090769-P264ALF [22]), and is connected to the service connector on the LV board [16]. These wires are routed to the appropriate connectors for each GEB so that the voltage across the load can be monitored, and the voltage drop over the cable can be accounted for.

4.6 High Voltage

The high voltage power supply is provided by a CAEN A1515 high voltage board. This board features 16 floating HV channels, which allows for providing HV to two, triple-GEM detectors. The maximum voltage per channel is 1.4 kV, with a maximum output current of 1 mA (operated in the high power mode) or 100 μA (operated in the high resolution mode).

The connector at the power supply is a CAEN Mod. A996 52 pin HV cable connector (compatible with the Radiall 691803004 type connector) [24]. On the cable from the power supply is an Amphenol H401 female, 9-pin connector. This connector plugs into the receptacle on the patch panel, which is an Amphenol H403 male, 9-pin connector. The leads of this connector connect to the filter box, which houses the HV lowpass circuits for each channel. The tops and bottoms of each GEM foil, along with the drift and readout board (ROB), have their own channel, which allows for the precise control of the HV to all elements of the detector.

5 Instrumenting a Module with Electronics

Here, we discuss the procedure of instrumenting the frontend electronics on a single module of the GE2/1 GEM detector, and making the optical connections to the GLIB. We also review the power distribution test for the FEASTs.

5.1 Instrumenting the GEB with FEASTs and Testing the Power Distribution

First, the GEB is fixed to the ROB of the module via six screws. From here, one connects all five FEASTs to the board. Before installing the remaining electronics (i.e., the VFATs or the OH), it is important to test the power distribution on the GEB. This will ensure that the OH is not receiving an over-voltage which can damage the FPGA and GBT ASICs and reduce its operational lifetime. This test also ensures that the VFATs are receiving an adequate voltage (there is a voltage drop across the GEB which can lead to an under-voltage at the VFAT power connectors). Table 2 lists the range of safe voltages that the FEASTs can provide to the GEB. If the voltages on the test points are excluded from these ranges, a new FEAST must be selected.

Table 2: Voltage Tolerance of the FEASTs

Nominal Voltage (V)	Tolerance Range (V)
1.20	[1.17, 1.27]
1.55	[1.47, 1.59]
1.86	[1.76, 1.91]
2.58	[2.45, 2.66]

To check the power distribution of the FEASTs, instrument the board with only the FEASTs and connect the LV supply, then power the GEB with 6.5 V. Place the negative probe of a multimeter on the digital ground (DGND) test point on the GEB, and place the positive lead on the test point for each respective FEAST (see Fig. 12).

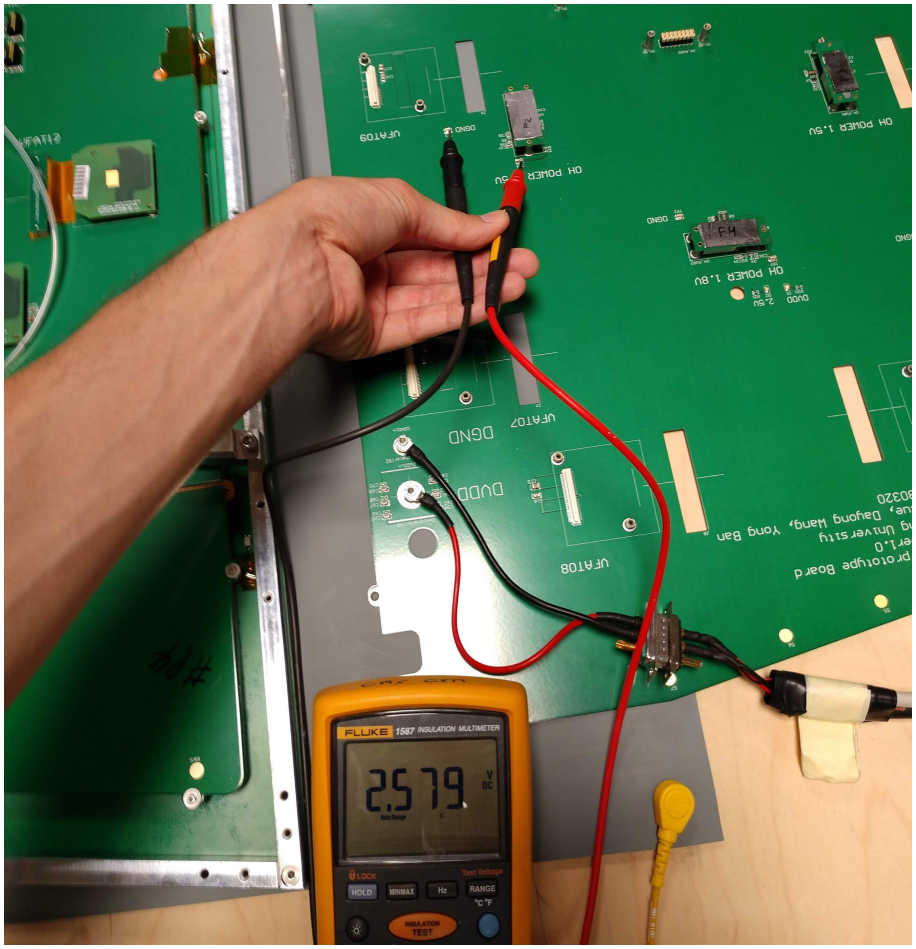


Figure 12: Picture of the measurement of the voltages produced by the FEASTs on the GEB.

If the voltage is distributed correctly based on the voltage ranges listed in Tab. 2, one can move to instrumenting the board with the OH and the VFATs once the LV is switched off.

5.2 Optical Fiber Connections

This section discusses the optical fiber connections that are made to the OH and to the backend AMC for version 1 electronics. First, we discuss the on-chamber fiber connections which are made to the OH and to the patch panel, and then the connection to the backend AMC. Finally, we discuss the actual method of connecting the fibers.

5.2.1 On-Chamber Fiber Plant and Backend AMC Connections

The on-chamber fiber plant, when installed in the CMS experiment, will consist of two Multi-fiber Termination Push-on (MTP®) to Lucent Connector[3)] (LC) fanout fibers. These cables are laser-optimized mutlimode fibers (LOMMF) following the optical multimode (OM) 3 standard (OM3)[4)].

One fiber plant is dedicated completely for GEM RO and trigger applications, with another reserved for the CSC/optical trigger motherboard (OTMB)[5)]. At the FIT test stand, the fiber bundle responsible for the GEM RO and trigger is a male MTP24 (24 fibers integrated into one connector) to 24 LC fiber fanout with type B (rollover) polarity. Typically, LC connectors are mounted in a duplex configuration via plastic molding (one fiber for transmission purposes, and the other for receiving purposes). For the GEM fiber plant, one duplex is connected to the

[3)] An LC connector is a miniature version of the subscriber connector (SC), which houses a single optical fiber.

[4)] The OM3 designation is a fiber standard which specifies the core size (50 μm), bandwidth (1 Gb/s), and maximum transmission distance (300 m). The jacket color for OM3 fibers is aqua.

[5)] Recall that the GE2/1 will be installed in the second muon station in the endcaps where the ME2/1 CSC chambers are located.

GBT0 VTRx, the GBT1 VTRx, and the GEM trigger VTTx on the OH. The male MTP24 side is connected to a standard footprint MTP optical coupler (key-up-to-key-down) mounted at on the patch panel.

The CSC/OTMB trigger links are provided by a male MTP12 to 12 LC fiber fanout. Here, one LC duplex is connected to the CSC VTTx on the OH. Note that only eight of the twelve fibers will be used. Like the MTP24M fiber, this cable is connected to the other MTP optical coupler mounted on the patch panel. These on-chamber connections are displayed in the Fig. 13.

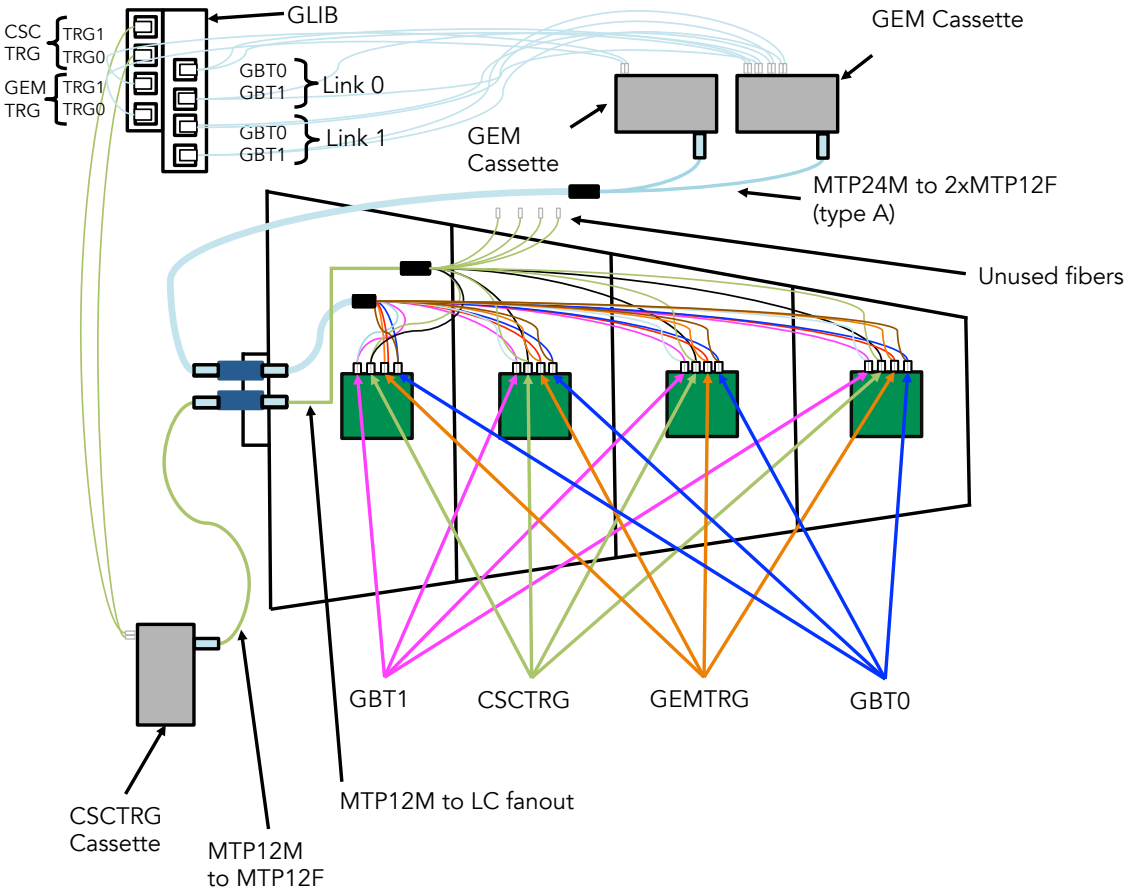


Figure 13: The on-chamber fiber plants and the associated connections. Note that each connection to the VTRx/VTTxs on the OH are duplex LC connections.

The connections to the backend AMC (in this case, the GLIB), are provided by LC connections to the SFPs mounted in the cages of the GLIB. This can be accomplished by using another male MTP24 to LC fanout and male MTP12 to LC fanout for the GEM data/trigger and CSC trigger links, respectively, connected to the MTP optical coupler at the patch panel. Alternatively, to increase the versatility of the setup, and allow for connections to a CVP13 backend card, the configuration in Fig. 13 can be adopted. This is accomplished by MTP to MTP interconnecting patch fibers, which run from the patch panel to the cassettes. In the case of the GEM data and trigger links, a male MTP24 to 2 female MTP12 fanout fibers are connected to two 12 LC (6 LC duplex) cassettes, from which standard LC fibers are connected to the GLIB. For the CSC trigger links, an male MTP12 to female MTP12 fiber is connected from the patch panel to a 12 LC cassette, from which standard LC fibers are connected to the GLIB. The standard GLIB mapping is shown in Fig. 14 below, and for the alternate configuration with the CSC trigger links, the mapping is shown in Fig. 15.

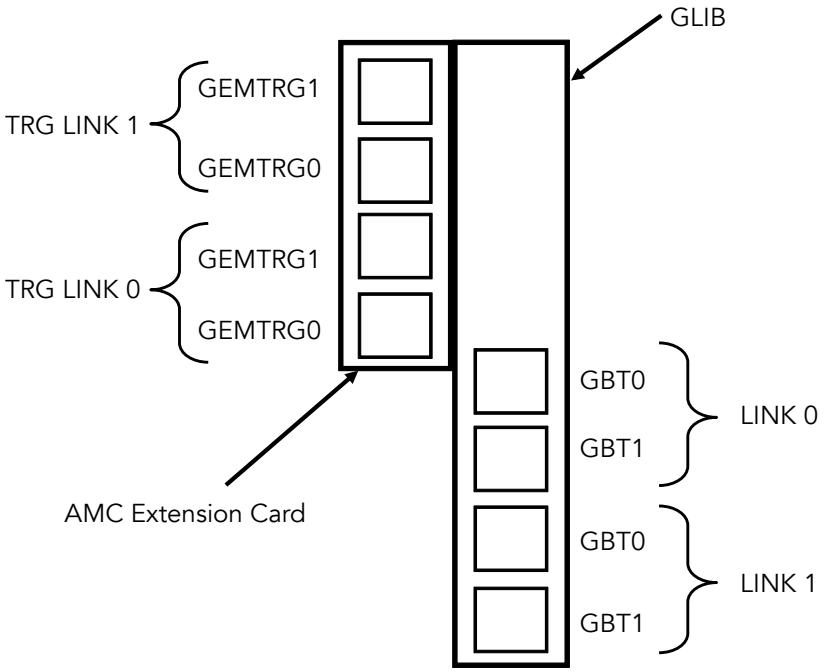


Figure 14: Fiber mapping configuration for the GLIB with both GEM trigger links connected.

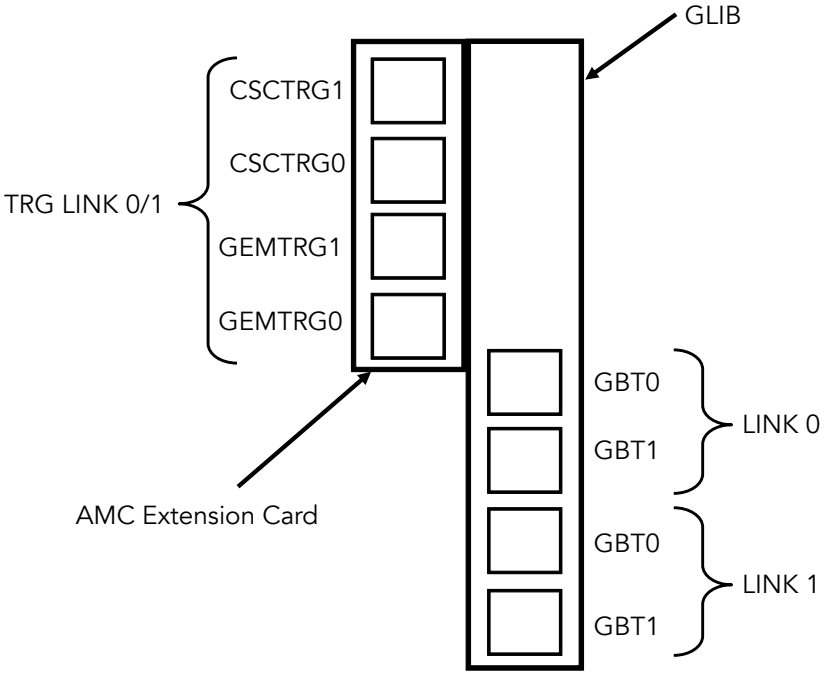


Figure 15: Fiber mapping configuration for the GLIB with the GEM and CSC trigger links connected (for a single GEB).

5.2.2 The Optohybrid Connections

Before instrumenting the GEB with any electronics, ensure that the GEB is powered off. To connect the OH to the GEB, simply align the power connector pins on the GEB with the power connector on the OH, and then press lightly on the sides where the Samtec connector is.

When connecting the optical fibers, make sure that the fibers are appropriately mapped (i.e., connected to the appropriate GBT or trigger link), and that the fibers are in the correct ports (recall that the GBTx is operated in bidirectional mode, and so connections to the VTRxs require one line for transmission and one line for receiving).

See Fig. 16 for a diagram of the fiber connections on the OH.

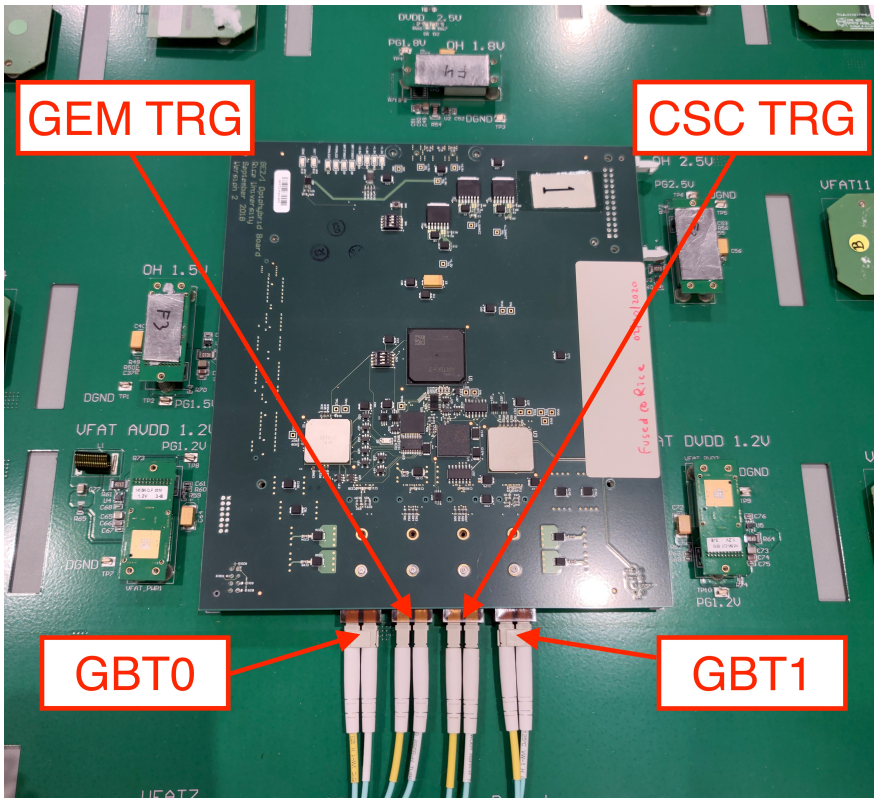


Figure 16: Picture of the OH with the GBT and trigger connections labeled [6]. Note that the fibers for the GEM and CSC triggers are split since they are mapped to two separate SFPs on the GLIB (see Sec. 4.4)

To ensure that the receiver and transmitter are connected in the correct ports, look at the VTRx ports. One should see a light in both the fiber and the connector; the fiber that is illuminated is to be connected into the port that is not illuminated (i.e., the receiver), and the fiber that is not illuminated should be connected to the port that is illuminated (i.e., the transmitter); see Figs. 17 and 18. Note that both of the VTTx ports are illuminated since these serve as transmitters only (for the GEM and CSC triggers).

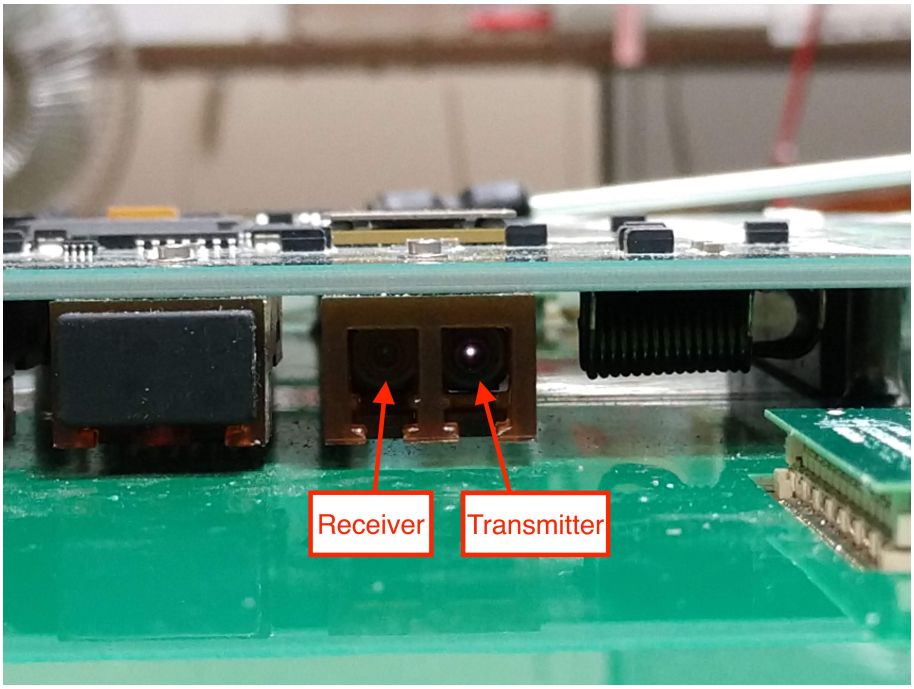


Figure 17: A VTRx on an OH. Note the illuminated transmitter on the right.

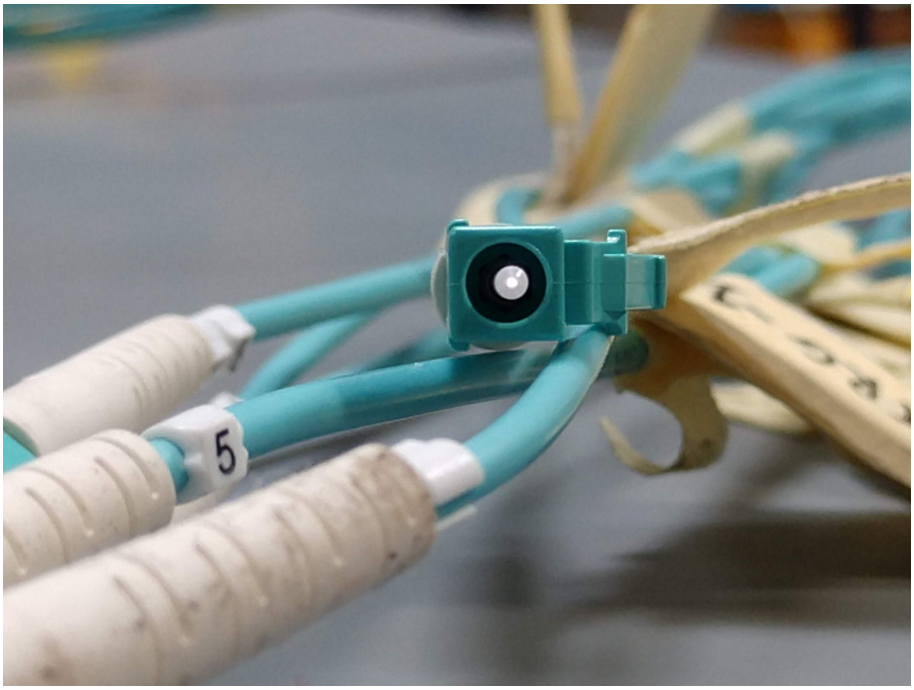


Figure 18: A transmitting, illuminated optical fiber which is to be connected to the receiver port on the VTRx which is not illuminated.

5.3 Instrumenting the VFAT3 Hybrid Cards

The VFAT3 Hybrid Cards feature two Panasonic connectors (one, 130-pin female connector, and one, 110-pin female connector). To make the connection on the GE2/1 module, it is necessary to use a Panasonic-to-Hirose adapter, also known as the FlexPCB adapter, since the ROB of the GE2/1 features Hirose connectors. The adapter features one male Hirose 140 connector for the connection to the ROB of the GE2/1, and a male, 130-pin Panasonic connector [25]. To instrument the VFAT3 hybrid cards on the GEB, simply plug the Panasonic-to-Hirose (FlexPCB) adapter into the 130-pin female Panasonic connector on the VFAT3 hybrid card. Once the adapter and the VFAT are joined, connect the 110-pin female connector on the hybrid to the male Panasonic connector on the

GEB, and then insert the Hirose connector into the connector on the ROB. See Fig. 19 for a picture of the VFAT and FlexPCB adapter instrumented on a GEB.

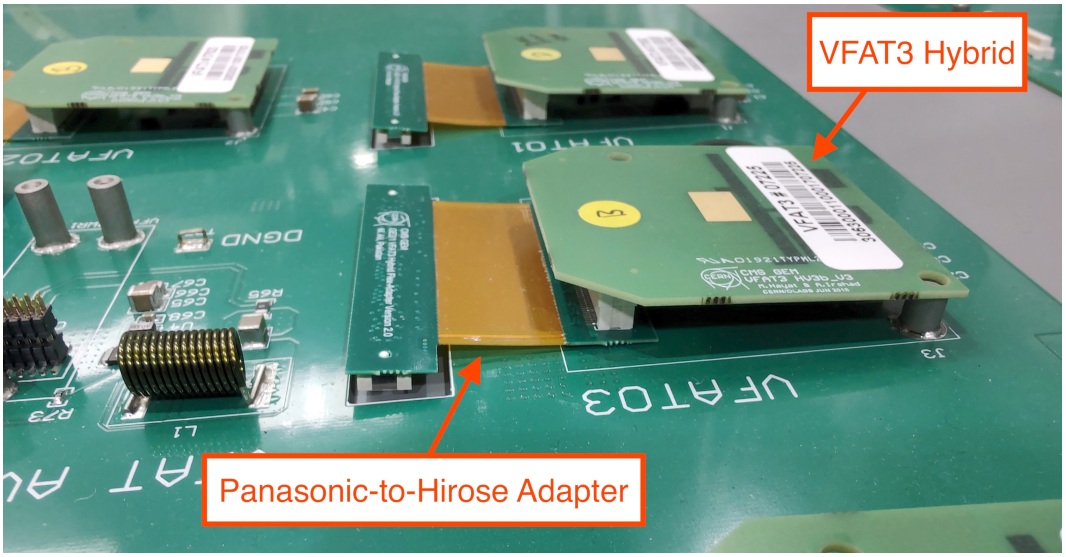


Figure 19: The FlexPCB adapter and the VFAT3 hybrid card on the GEB.

6 DAQ System Setup

This installation will be specific to the `CentOS 7` operating system. This section will cover installation of the `CMS-GEM-DAQ-Project` Software dependencies, the `CMS-GEM-DAQ-Project` software, the `gem-emulator`, and setup of the GLIB. This section derives much inspiration and content from Laurent Pétré's original GEM DAQ setup document [26].

6.1 `CMS-GEM-DAQ-Project` Software Dependency Installation

This subsection outlines the general dependencies that need to be installed on the DAQ PC before installation of the `CMS-GEM-DAQ-Project` software.

6.1.1 Software Updating, Upgrading, and Installation

Before the installation of any software, we need to update/upgrade the OS and all of the packages present on the machine. Run

```
$ yum update
$ yum upgrade
```

We also need to install the `centos-release-scl` package. The Software Collections (SCL) repository ensures compatibility between older and newer versions of software. See [27].:

```
$ yum install centos-release-scl
```

6.1.2 Installing/Upgrading `pip`

We will need to install `pip` [28] if it has not been installed yet. In order to do this, we need to first enable/install the `EPEL` (Extra Packages for Enterprise Linux) [29] repository and then install `pip`:

```
$ sudo yum --enablerepo=extras install epel-release
$ sudo yum install -y --enablerepo"epel" python-pip
```

Now make sure `pip` is the latest version:

```
$ sudo pip install --upgrade pip
```

6.1.3 Installing the MySQL Development Package

The development package for mysql [30] is an open-source, structured query language (SQL) used for managing databases, and is necessary for cmsgemos-gemutils software. To install this dependency, run

```
$ sudo yum install mysql mysql-devel
```

6.1.4 Installing the Developer Toolset 7.0/8.0

When we go to make the rpms of the various repositories of the CMS-GEM-DAQ-Project software, we will need higher versions of the pre-installed compilers on the machine (e.g., to install the xhal repository, we'll need g++ $\geq$ v.4.9). To install the 7$^{\text{th}}$ and 8$^{\text{th}}$ major releases of the devtoolset package, run the command

```
$ sudo yum install devtoolset-{7,8}
```

This package is necessary for installing the xdaq (pronounced "cross DAQ") software, as it needs version 8.2.1 of the gcc compiler.

6.1.5 Installing the boost C++ libraries

The boost [31] libraries are an open-source collection of libraries that extend the functionality of C++. They are required for some subpackages in the cmsgemos repository. To install, simply execute

```
$ sudo yum install boost boost-thread boost-devel
```

6.2 Installing pugixml

The pugixml library [32] is an XML parser and provides support for XML Path Language (XPath). This package is necessary for the cmsgemos-gemhardware repository. To install, execute

```
$ sudo yum install pugixml pugixml-devel
```

6.2.1 Installing ROOT

ROOT [33] is the standard processing and plotting software in high energy physics, and is used extensively by the CMS-GEM-DAQ-Project software.

Installing the Dependencies

Unfortunately, installing ROOT is not as straightforward as a single yum command. This is because the binaries installed under the yum repositories root and python2-root do not contain many of the additional packages in ROOT, such as the Toolkit for Multivariate Analysis (TMVA). These are necessary for installing root_numpy (see Sec. 6.2.9). To install, we will download the latest source code directly from the ROOT project's GitHub, and build ourselves. Note that the installation procedure presented below has been adapted from ROOT's official installation page [34].

Before we start, we need to install all of the dependencies for ROOT. Since many of the dependencies of ROOT come pre-installed on CC7, we only need to install a select number of software packages. (For a full list of dependencies, see [35].) To install all of the dependencies with one command, run:

```
$ sudo yum install binutils libX11-devel libXpm-devel libXft-devel libXext-devel openssl-devel
```

Since we need cmake3 for building the ROOT source code, we will install that package as well:

```
$ sudo yum install cmake3
```

Note that `cmake2` is the default `cmake` version, and issues can arise when trying to build `ROOT`. In theory, one can make a symbolic link, pointing the `cmake2` binary file to the `cmake3` file in `/usr/bin/`. If this doesn't work, one can simply uninstall using `yum remove cmake`, so that only the `cmake3` is available for building the source.

Building ROOT From Source

Now that the dependencies are installed, we can download the source[6] from GitHub:

```
$ git clone --branch v6-22-00-patches https://github.com/root-project/root.git root_src
```

With the source code cloned, we make our build and installation directories (you can name them whatever you want, just make sure there is a distinction between the directory where `ROOT` is built and installed):

```
$ mkdir buildDir installDir
```

Now, we will set the installation and build directories:

```
$ cmake3 -DCMAKE_INSTALL_PREFIX=../buildDir ../installDir
```

Now, we will build the source (the `-j` flag specifies the amount of cores to be used during the install; if you have more than four cores, feel free to increase the number):

```
$ cmake3 --build . -- install -j4
```

After `ROOT` has been installed, we add the following line to our `.bash_profile` so that the `ROOT` environment variables are sourced whenever a new `bash` session is started:

```
source path/to/installDir/bin/thisroot.sh
```

(If your default shell is `fish` or `csh`, simply modify the file type from `.sh` to either `.fish` or `.csh`.)

6.2.2 Installing the xdaq Repositories

The `xdaq` software platform is used in facilitating the design and implementation of the `CMS-GEM-DAQ-Project` software (see [37] for more information on the `xdaq` software). Before running the `yum install` command, we need to configure a `.repo` file for the `xdaq` repository. In the `/etc/yum.repos.d/` directory, create the following file: `xdaq.repo` and add the following contents:

```
[xdaq]
name=XDAQ Software Base
baseurl=http://xdaq.web.cern.ch/xdaq/repo/14/cc7/x86_64/base/RPMS/
enabled=1
gpgcheck=0

[xdaq-sources]
name=XDAQ Software Base Sources
baseurl=http://xdaq.web.cern.ch/xdaq/repo/14/cc7/x86_64/base/SRPMS/
enabled=1
gpgcheck=0

[xdaq-updates]
name=XDAQ Software Updates
baseurl=http://xdaq.web.cern.ch/xdaq/repo/14/cc7/x86_64/updates/RPMS/
enabled=1
gpgcheck=0

[xdaq-updates-sources]
name=XDAQ Software Updates Sources
baseurl=http://xdaq.web.cern.ch/xdaq/repo/14/cc7/x86_64/updates/SRPMS/
enabled=1
gpgcheck=0

[xdaq-extras]
name=XDAQ Software Extras
baseurl=http://xdaq.web.cern.ch/xdaq/repo/14/cc7/x86_64/extras/RPMS/
enabled=1
gpgcheck=0
```

[6] At the time of this writing, v.6-22/00 was the latest stable release. Refer to `ROOT`'s official GitHub [36] for the latest version.

We also need to define the environment variables. Add a file named `xdaq.sh` in the `/etc/profile.d/` directory with the following contents:

```
export XDAQ_OS=linux
export XDAQ_PLATFORM=x86_64_centos7
export XDAQ_ROOT=/opt/xdaq
export XDAQ_DOCUMENT_ROOT=${XDAQ_ROOT}/htdocs
export LD_LIBRARY_PATH=${XDAQ_ROOT}/lib:$LD_LIBRARY_PATH
export PATH=${XDAQ_ROOT}/sbin:${XDAQ_ROOT}/bin:$PATH
```

We also need to download and group install the associated packages with `xdaq`. Run:

```
sudo yum groups install extern_coretools coretools extern_powerpack powerpack
sudo yum --skip-broken groups install general_worksuite database_worksuite hardware_worksuite dcs_worksuite
```

6.2.3 Installing the Wisconsin RPC Service (`wiscrpcsvc`) Binary

First, download the precompiled binary RPM from https://github.com/mexanick/wiscrpcsvc/releases/download/1.0.0/wiscrpcsvc-1.0.0-1.testing.centos7.x86_64.rpm. To install, use `yum install`:

```
$ sudo yum install wiscrpcsvc-1.0.0-1.testing.centos7.x86_64.rpm
```

Like before, we need to define an environment variable. Add a file named `wiscrpcsvc.sh` in the `/etc/profile.d/` with the following definition of the `LD_LIBRARY_PATH` environment variable:

```
export LD_LIBRARY_PATH=/opt/wiscrpcsvc/lib:$LD_LIBRARY_PATH
```

6.2.4 Installing the `uhal` Libraries

First, we need to create a `.repo` file in `/etc/yum.repos.d/` named `ipbus-sw.repo` with the following contents:

```
[ipbus-sw-base]
name=IPbus software repository
baseurl=http://www.cern.ch/ipbus/sw/release/2.6/repos/centos7_x86_64/base/RPMS
enabled=1
gpgcheck=0

[ipbus-sw-updates]
name=IPbus software repository updates
baseurl=http://www.cern.ch/ipbus/sw/release/2.6/repos/centos7_x86_64/updates/RPMS
enabled=1
gpgcheck=0
```

To group install `uhal`, execute

```
sudo yum group install uhal
```

6.2.5 Installing the `cactus` Framework

The computing package `cactus` is a framework for scientific computing applications, see [38] for documentation. The packages for the `cactus` framework are installed in part from the installation of the `uhal` and `amc13` packages. Again, we make a file in the `/etc/profile.d/` directory named `cactus.sh` and add the following environment variables:

```
export uHALROOT=/opt/cactus
export AMC13_ADDRESS_TABLE_PATH=${uHALROOT}/etc/amc13
export LD_LIBRARY_PATH=${uHALROOT}/lib:$LD_LIBRARY_PATH
export PATH=${uHALROOT}/bin/amc13:${uHALROOT}/bin:$PATH
```

6.2.6 Installing the amc13 Libraries

Create the file /etc/yum.repos.d/amc13.repo and add the following contents:

```
[cactus-amc13-base]
name=CACTUS Project Software Repository for amc13 packages
baseurl=http://www.cern.ch/cactus/release/amc13/1.2/uhal2.6/centos7_x86_64/base/RPMS
enabled=1
gpgcheck=0

[cactus-amc13-updates]
name=CACTUS Project Software Repository Updates for amc13 packages
baseurl=http://www.cern.ch/cactus/release/amc13/1.2/uhal2.6/centos7_x86_64/updates/RPMS
enabled=1
gpgcheck=0
```

Now, yum group install:

```
sudo yum group install amc13
```

6.2.7 Installing python cx_Oracle

The cx_Oracle package is an extension module that allows python to interface with Oracle databases. This module is required for pulling the VFAT3 production calibration data from the database before running DAC scans (see Sec. 7.3). To download and install, simply run

```
$ python -m pip --user install cx_Oracle --upgrade
```

If this has already been installed, it will upgrade this module. If not, the binary will be downloaded and installed. See [39] for more information.

6.2.8 Installing the tabulate module

This python module is necessary for text formatting for the CMS GEM software. Install via pip:

```
$ pip install tabulate
```

Note: when installing xhal (Sec. 6.3.2), one might face an issue where yum or rpm thinks that the tabulate package isn't installed. You can check this by running:

```
$ locate tabulate
```

if the correct path is returned, execute

```
$ rpm -qf path/to/tabulate
```

If the output is that tabulate isn't owned by any package, then you will need to install tabulate via the precompiled rpm hosted at CERN. Run:

```
$ wget http://cmsgemdaq.web.cern.ch/cmsgemdaq/sw/repos/centos7_x86_64/
    extras/RPMS/tabulate-0.8.2-1.noarch.rpm
$ sudo yum install tabulate-0.8.2-1.noarch.rpm
```

6.2.9 Installing the root_numpy module

This python module is necessary for python to easily interface with ROOT files. Install via pip (where the pip version must be lower than v. 21):

```
$ pip install --user root_numpy
```

6.3 Installing `CMS-GEM-DAQ-Project` Software

We now review installation and configuration of the `CMS-GEM-DAQ-Project` [40] software. Create a new directory where all of the repositories will be cloned to. From this base directory, run the `git clone <url>` command to download the repositories. Examples are shown, in order, for installing the `CMS-GEM-DAQ-Project` software. To make the rpm, we will run the `make rpm` command. Note that there are many branches of the software currently available; if you are setting up a test stand and need functionality for only GE1/1 and GE2/1 GEM detectors, the `legacy` release in each of the repositories is ideal; if you need to install the repositories for development purposes, change to your preferred branch after cloning the repository. After building the `rpms`, you will need to navigate to the directory where they were built, usually in

```
~/${BUILD_HOME}/<CMS-GEM-DAQ-PROJECT-SOFTWARE-REPOSITORY>/rpm/
```

and then install the appropriate binary (usually, this is the `*.noarch.rpm`, but, depending on the specific repository, it could be the `*.x86_64.rpm` for Intel architecture of `*.arm.rpm` for ARM architecture).

Before getting started, create a directory where all of the GEM DAQ software will be built. Set the environment variable in the `.bash_profile` and export as follows:

```
BUILD_HOME="$path/to/your/GEMDAQ_SW_directory"
export BUILD_HOME
```

6.3.1 Installing `reg_utils`

First, clone the directory,

```
$ git clone https://github.com/cms-gem-daq-project/reg_utils.git
```

Then pull the source for any submodules,

```
$ cd reg_utils
$ git submodule update --init
```

Now we need to make and install two separate `rpms`. The first is `rwreg`. From the base of `reg_utils`, execute

```
$ cd rwreg/x86_64
$ make rpm
$ cd rpm
$ sudo yum install rwreg-devel-*x86_64.rpm
```

(Note if your CPU is based on ARM architecture, navigate to `rwreg/arm` and repeat the steps above.) Prepend to your `$PATH` the following path in your `~/.bash_profile`:

```
/opt/reg_utils/bin/
```

6.3.2 Installing `xhal`

Again, we need to make two different `rpms` and install separately. First, we'll clone the repository and update to retrieve a submodule.

```
$ git clone https://github.com/cms-gem-daq-project/xhal.git
$ cd xhal
$ git submodule update --init
```

Now,

```
cd xhalcore
make rpm
```

Navigate to `rpm/` and install via `yum`.

Now, go to `xhal/python`, and execute `make rpm`. Change directories to the `rpm/` directory and install the binary file:

```
sudo yum install reg_interface_gem-*.noarch.rpm
```

(If you are receiving errors that the package `tabulate` is missing, please refer to Sec. 6.2.8 and download the `rpm` from the CMSGEMDAQ server, and then install the pre-compiled `rpm`.)

Finally, prepend the code below to your `$PATH` environment variable in your `.bash_profile`:

```
/opt/xhal/bin/
```

As always, `source` your `.bash_profile` for the changes to take effect.

6.3.3 Installing `cmsgemos`

First, clone the directory,

```
$ git clone https://github.com/cms-gem-daq-project/cmsgemos.git
```

Now change directories, check for submodules, and build the `rpm`:

```
$ cd cmsgemos
$ git submodule update --init
$ make rpm
```

Since many `rpm`s are built with this command, execute

```
$ find . -name '*.rpm'
```

from the base directory of `cmsgemos` to recursively list all of the generated `rpm`s. Then, install all `rpm`s ending in `*.x86_64.rpm` with `sudo yum install`.

6.3.4 Installing `gem-plotting-tools`

```
$ git clone https://github.com/cms-gem-daq-project/gem-plotting-tools.git
$ cd gem-plotting-tools
$ git submodule update --init
$ make rpm
```

Finally,

```
$ sudo yum install <NAME_OF_RPM>.x86_64.rpm
```

6.3.5 Installing `vfatqc-python-scripts`

```
$ git clone https://github.com/cms-gem-daq-project/vfatqc-python-scripts.
    git
$ cd vfatqc-python-scripts
$ git submodule update --init
$ make rpm
```

Finally, change directories to where the `rpm`s were built, install the

```
$ sudo yum install <NAME_OF_RPM>.noarch.rpm
```

Note that if the following dependencies were not already on the existing machine, they will be installed when `vfatqc-python-scripts` `rpm` is installed: `blas`, `lapack`, `numpy`, `python-nose`.

6.3.6 Cloning `sw_utils`

This repository only needs to be cloned (no building of the `rpm` necessary).

```
$ git clone https://github.com/cms-gem-daq-project/sw_utils.git
```

6.3.7 Setting the Remaining Environment Variables

In the `bash_profile` add the following environment variables with the appropriate paths. An example is given below for the GE2/1 DAQ PC at Florida Tech:

```
#Set environment variables
export BUILD_HOME="$HOME/repos/"
export PATH="/opt/xhal/bin/:/opt/cmsgemos/bin/:/opt/reg_utils/bin/:$PATH:$HOME/.local/bin:$HOME/bin"
export DATA_PATH="/home/user/data/data/"
export ELOG_PATH="/home/user/data/elog/"
export GEM_ADDRESS_TABLE_PATH="/home/user/data/address_table/"
export PYTHONPATH="/home/user/data/config/"
export GBT_SETTINGS="/home/user/data/gbt/"
export LD_LIBRARY_PATH="/opt/rwreg/lib/:/opt/cmsgemos/lib/:/opt/xhal/lib/:$LD_LIBRARY_PATH"
export GEM_ONLINE_DB_CONN="CMS_GEM_APPUSER_R/GEM_Reader_2015@"
export GEM_ONLINE_DB_NAME="INT2R_LB_LOCAL"
```

Always remember to `source` the `bash_profile` after making changes in all running shells.

6.3.8 Updating the `CMS-GEM-DAQ-Project` Software

To update the software for a specific repository, change directories to the repository. Now, we can check if there have been any updates using `git fetch`. If there are changes, we can pull and install the new version of the same branch currently installed by running

```
$ git pull
$ git submodule update --init
$ make rpm
```

After the `rpm` has been built, change directories to `rpm/`. From here, locate the new `rpm` and run

```
$ yum install name_of_new_rpm.noarch.rpm
```

6.3.9 Configuring the System Specifics File

This file is essentially self-explanatory: it contains the specific system settings of various hardware configuration values, as well as the name of the chambers on a certain slot, shelf, and link (geographical location). Note that this *must* be configured before scans can be run. See below for an example of the file: The system configuration file contains important global parameters for all VFATs, as well as the GEB information for the geographical location of the links. The current system configuration file,
`/home/user/data/config/system_specific_constants.py` is reproduced below:

```
chamber_config = {
    (1,3,0):"GE21-M5-P1", #(shelf, slot, link):"name of module"
    (1,3,1):"GE21-M1-P3",
}

GEBtype = {
    (1,3,0):"m5",
    (1,3,1):"m1",
}

registers2apply = {
    #Latency
    "CFG_LATENCY":57,
    #"CFG_LATENCY":10,
    #Pulse Stretch
    "CFG_PULSE_STRETCH":3,
    #Ensure the cal pulse is off
    "CFG_CAL_MODE":0,
    #Provide a slight offset to the ZCC comparator baseline voltage
    "CFG_THR_ZCC_DAC":10,
    #High VFAT3 preamp gain
    #"CFG_RES_PRE":1,
    #"CFG_CAP_PRE":0,
    #Medium VFAT3 preamp gain
    "CFG_RES_PRE":2,
    "CFG_CAP_PRE":1,
    #Low VFAT3 preamp gain
    #"CFG_RES_PRE":4,
    #"CFG_CAP_PRE":3,
    #Comparator Mode - CFD
    "CFG_PT":0xf,
    "CFG_FP_FE":0x7,
    "CFG_SEL_COMP_MODE":0,
    "CFG_FORCE_EN_ZCC":0
    #Comparator Mode - ARM
    #"CFG_SEL_COMP_MODE":1,
    #"CFG_FORCE_EN_ZCC":0
    #Comparator Mode - ZCC
    #"CFG_SEL_COMP_MODE":2,
    #"CFG_FORCE_EN_ZCC":1
}

"""
Keys should be a tuple of (shelf,slot,link)
"""
import copy
chamber_vfatDACSettings = { key:copy.deepcopy(registers2apply) for key,val in chamber_config.iteritems()}
#chamber_vfatDACSettings[(1,4,3)]["CFG_LATENCY"]=58
```

6.3.10 Connecting to the CMS GEM Database

For some electronics integration procedures, we need to connect to the database for VFAT calibration info. If you are not on CERN's network, a tunnel needs to be opened using L. Pétré's script reproduced below:

```
#!/bin/bash
#Scrip to create a tunnel to
#the CMS DBs

ssh \
    -L 10131:cmsrac31-v.cern.ch:10121 \
    -L 10132:cmsrac32-v.cern.ch:10121 \
    -L 10141:cmsrac41-v.cern.ch:10121 \
    -L 10142:cmsrac42-v.cern.ch:10121 \
    -L 10101:itrac1601-v.cern.ch:10121 \
    -L 10109:itrac1609-v.cern.ch:10121 \
  user.name@lxplus7.cern.ch
```

where `user.name` is your `lxplus` username.

6.4 Setting up the GLIB

Before we can flash the GLIB's programmable read-only memory (PROM) with the firmware through the IPBus connection, we need to temporarily program the board so that it is assigned an IP address, which we will use later for permanent programming. From here, we can use the `PyChips` software to pull the board info, view if the

CDCE is locked, etc., and determine if the GLIB's firmware needs to be updated. If an update is needed, we can then use a script within the `PyChips` directory to flash the PROM.

6.4.1 Programming the `.bit` File

For temporary programming[7] of the FPGA, we use the appropriate `.bit` file. In order to program a `.bit` file to the FPGA, we will need Xilinx's `iMPACT v. 14.7`, which can be downloaded for Windows 10. Once the software has been installed, connect the JTAG connector on the red box programmer to the JTAG connector underneath the heatsink[8]. The following procedure has been reproduced and slightly modified from [41].

1. Start the `iMPACT` software. Select **no** when prompted if you would like to load the last saved project.
2. Select **yes** when prompted if you would like the system to automatically save a project file for you.
3. A new window will open with a set of options. Choose the "Configure devices using Boundary-Scan (JTAG)." Click **ok** to initialize the boundary-scan.
4. If the FPGA has been successfully identified, you will receive an "Identify Succeeded" message along with an additional window prompting if you would like to assign configuration files. Select **no**.
5. A device programming properties screen will open showing the FPGA connected. Close this window.
6. To assign the `.bit` file to the GLIB, right click on the device (the green box with the Xilinx logo) in the boundary scan window. Select "Assign New Configuration File."
7. A new window will appear and ask if you want to attach an SPI or BPI PROM to the device. Select **no**. Now select the appropriate file.
8. Right click again on the green box in the boundary scan window and select the "Program" option, then select "Ok." The device will then be programmed. If successful, you will receive a "Program Succeeded" message.

6.4.2 Installing `PyChips`

The `PyChips` software will allow us to access the GLIB, retrieve information about the board, and also flash the PROM with the firmware. Note that this needs to be done before we can use the `gem-emulator` software.

```
$ git clone https://github.com/cms-gem-daq-project/GLIB.git
```

Write the IP address of the GLIB in this file `~/repos/glib-pychips/scripts/ippaddr.dat`. Note that if there is a newline character at the end of the IP address the script to flash the PROM will not work. The easiest way to edit this file properly is

```
$ echo -n '192.168.80.3' > ippaddr.dat
```

Because we will only use the following commands a few times in the current section, we will skip editing the `bash_profile` and simply assign the environment variable `PYTHONPATH` for each command executed[9]. To print the information about the board, run

```
$ PYTHONPATH=$PWD/../src python glib_board_info.py
```

Now, we need to flash the PROM. Run

```
$ PYTHONPATH=$PWD/../src python glib_flash_golden_prom_rw.py name_of_file.
    mcs
```

[7] I.e., the memory will be erased after a power cycle.

[8] While the GLIB is vertically inserted in the MicroTCA crate, the JTAG connector under the heatsink is for the FPGA, while the JTAG connector to the right of the heat sink on the GLIB is for the CPLD.

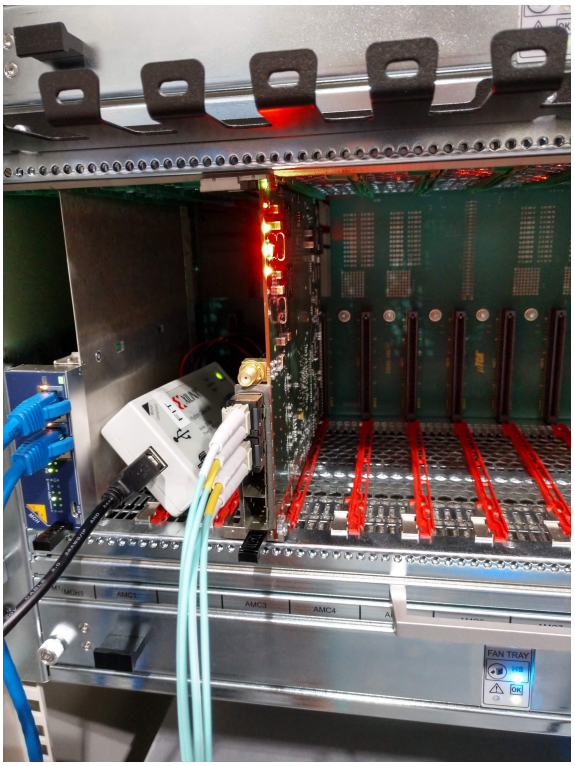
[9] This is done simply for convenience if your `PYTHONPATH` environment variable has already been set, as `export` will overwrite the set variable. (To check, echo the environmental variable by executing `echo $PYTHONPATH`). If this variable has not been set, simply run `export PYTHONPATH=$PWD/../src`.

We now wish to run the command below which will effectively power cycle the GLIB and confirm that the changes we wanted to make are permanent:

```
$ PYTHONPATH=$PWD/../src python glib_program_b_assert.py
```

The IP address of this board is now 192.168.81.3. Now, run the board info `python` script again to confirm that the changes were made. Confirm that the CDCE6205 clock synthesizer (responsible for generating the clocks used for the FPGA) is locked. We can read and write to the CDCE registers using the `glib_cdce_read.py` and `glib_cdce_write.py` scripts, respectively. Now, reset the GLIB by pulling the tab on the bottom of the board. The LEDs at the top of the board will turn off and a blue LED will be illuminated on the bottom (see Fig. 20a below). Push the tab back in. The lights at the top of the board will again be illuminated, and the blue LED at the bottom will turn off (see Fig. 20b).

(a) GLIB card with the bottom tab pulled out. Note the illuminated blue LED at the bottom of the card.



(b) GLIB card with the bottom tab pushed in. Note the illuminated green LED and the red LEDs at the top.

Figure 20: GLIB card with the bottom tab pulled out (a) and pushed in (b).

6.4.3 Creating a Private Network for the GLIB and `gem-emulator` Container

We need to create a private network for the GLIB and the `gem-emulator` container to communicate on. In the `/etc/NetworkManager/system-connections` directory, create the following file, `Private\ Network`, with the following contents:

```
[connection]
id=Private Network
uuid=ddce1799-18cf-470b-ae72-31c6ff4e16d3
type=macvlan
interface-name=private
permissions=
timestamp=1572286380

[macvlan]
mode=2
parent=enp1s1

[ipv4]
address1=192.168.0.180/16
dns-search=
ignore-auto-dns=true
method=manual
never-default=true

[ipv6]
addr-gen-mode=stable-privacy
dns-search=
ignore-auto-dns=true
method=ignore
never-default=true
```

Note that for the `uuid`, you should generate your own, unique `uuid`. This can be done via the command line with the command

```
$ uuidgen
```

6.5 Installing the `gem-emulator`

This software emulates the `Zynq SoC` of the CTP7. The installation procedure listed below has been adapted from [42]. To install, add a new file `/etc/yum.repos.d/gem-emulator.repo` with the following contents:

```
[gem-emulator]
name=GEM emulator software repository
baseurl=https://lpetre.web.cern.ch/lpetre/gem/repos/el7/x86_64
enabled=1
gpgcheck=0
```

To install, execute

```
$ yum install gem-emulator{,-selinux}
```

Now, we enable and start the `IPBus ControlHub`,

```
$ systemctl enable --now controlhub.service
```

Add a new entry to the `/etc/hosts` file for the controlhub service:

```
###### IPBus
192.168.0.180 gem-shelf01-controlhub
```

And also add a host entry for where the specific AMC, which takes the generic name

```
###### Host for AMC
192.168.<shelf>.<slot+40> gem-shelfXX-slotYY
```

So for an AMC on shelf 1 in slot 3, we add the following line to the `/etc/hosts/` file:

```
192.168.1.43 gem-shelf01-slot03
```

Start the service for the particular AMC (slot) in the μTCA crate (shelf). An example is given below for the GLIB in shelf 1 slot 3:

```
$ systemctl start gem-emulator@1-3:glib:0
```

Now we're ready to `ssh` into the container, which, in version 0, has password-less authentication:

```
$ ssh gemuser@gem-shelf01-slot03
```

Now, we add the address tables, firmware, RPC modules, and supporting software to the container. The `gem-emulator` container is a (mostly) blank canvas that needs to be populated with the specific files for our applications. The following sections will overview what files need to be added. Note that for many directories, `root` permissions are required to `scp` the files. So it is easiest for all files to be transferred as `root` (see the `scp` example in the following section).

6.5.1 Setting up the `vfat3` Directory

We need to add a useful script to generate the configuration files for the VFATs associated with OH0 and OH1. First, retrieve the script `setLinks.py` from the `cms-gem-daq-project/gemctp7user/scripts/` repository here. Now, transfer the script using `scp`:

```
$ scp setLinks.py root@gem-shelf01-amc03:/mnt/persistent/gemdaq/vfat3
```

Before generating the configuration files, we need to place an example VFAT3 configuration file this directory (`/mnt/persistent/gemdaq/vfat3`), which is reproduced below (name this file `conf.txt`):

```
dacName/C:dacVal/I
PULSE_STRETCH           0
SYNC_LEVEL_MODE         0
SELF_TRIGGER_MODE       0
DDR_TRIGGER_MODE        0
SPZS_SUMMARY_ONLY       0
SPZS_MAX_PARTITIONS     0
SPZS_ENABLE             0
SZP_ENABLE              0
SZD_ENABLE              0
TIME_TAG                0
EC_BYTES                0
BC_BYTES                0
FP_FE                   0
RES_PRE                 1
CAP_PRE                 0
PT                      1
EN_HYST                 1
SEL_POL                 0
FORCE_EN_ZCC            0
FORCE_TH                0
SEL_COMP_MODE           0
VREF_ADC                1
MON_GAIN                0
MONITOR_SELECT          0
IREF                   32
THR_ZCC_DAC            11
THR_ARM_DAC            32
HYST                    5
LATENCY                 0
CAL_SEL_POL             0
CAL_PHI                 0
CAL_EXT                 0
CAL_DAC                 0
CAL_MODE                0
CAL_FS                  0
CAL_DUR               200
BIAS_CFD_DAC_2         40
BIAS_CFD_DAC_1         40
BIAS_PRE_I_BSF          1
BIAS_PRE_I_BIT          1
BIAS_PRE_I_BLCC        15
BIAS_PRE_VREF          86
BIAS_SH_I_BFCAS         1
BIAS_SH_I_BDIFF         1
BIAS_SH_I_BFAMP         0
BIAS_SD_I_BDIFF         1
BIAS_SD_I_BSF           1
BIAS_SD_I_BFCAS         1
```

Finally, we are ready to generate the VFAT configuration files (with symbolic links) for both of our OH links. Execute the following command from inside of the `/mnt/persistent/gemdaq/vfat3` directory:

```
$ python setLinks.py -g 0
$ python setLinks.py -g 1
```

These configuration files contain all of the DAC register values for the VFATs, and are updated using the `replace_parameter.sh` script discussed in the next subsection.

6.5.2 Adding the `replace_parameter.sh` Script

Retrieve the `bash` script `replace_parameter.sh` from the `cms-gem-daq-project/gemctp7user/scripts/` repository here. Transfer the script to `/mnt/persistent/gemdaq/scripts` in the container.

6.5.3 Adding the Address Tables

Two address tables in total need to be added to the `/mnt/persistent/gemdaq/xml/` directory: one `.xml` file for the OH and one `.xml` file for the GLIB. Transfer the appropriate tables into this directory. At the time of creating this document, the compatible files used were

```
glib_address_table_3_9_6.xml
oh_registers_3.2.2.2A.xml
```

Now, we need to make symbolic links for these two files:

```
$ ln -s oh_registers_3.2.2.2A.xml optohybrid_registers.xml
$ ln -s glib_address_table_3_9_6.xml gem_amc_top.xml
```

You can see that the symbolic linking is successful by listing the contents of the directory. You will see two new symbolic links of the form:

```
[gemuser@glib-shelf01-slot03 xml]$ ll
total 212
lrwxrwxrwx. 1 root root     28 Apr  3 07:07 gem_amc_top.xml -> glib_address_table_3_9_6.xml
-rw-r--r--. 1 root root 145383 Apr  3 07:04 glib_address_table_3_9_6.xml
-rw-r--r--. 1 root root  69320 Apr  3 07:04 oh_registers_3.2.2.2A.xml
lrwxrwxrwx. 1 root root     25 Apr  3 07:07 optohybrid_registers.xml -> oh_registers_3.2.2.2A.xml
```

6.5.4 Adding the `rpcmodules`

Now, transfer the RPC modules to `/mnt/persistent/rpcmodules`. The following library files are necessary:

```
amc.so
calibration_routines.so
daq_monitor.so
extras.so
gbt.so
memhub.so
memory.so
optohybrid.so
utils.so
vfat3.so
```

6.5.5 Updating the `lib` Directory

Add the following libraries to the `/mnt/persistent/gemdaq/lib` directory:

```
libreedmuller.so
librpcman.so
libxhal.so
```

6.5.6 Updating the `reset_gtx.py` file for GE2/1 Compatibility

The polarity of transmission for the GE2/1 OH is inverted with respect to the GE1/1 OH. So, we need to make two changes in the script `/usr/local/bin/reset_gtx.py`. The changes, provided by L. Pétré, are marked by the `# <-----` comments in the code. Update the file so that it is now

```
#!/bin/env python

import uhal

import os

glibAddr = os.environ["GEM_IPBUS_CONNECTION"]
glib = uhal.buildClient("GLIB", glibAddr)

#### Configure the GTX's
## GBT: invert TX polarity
for i in range(4):
    glib.write(0x50000000+2*i, 0x00440008) # <------

## Trigger: only reset RX
for i in range(4):
    glib.write(0x50000008+2*i, 0x00040000)

glib.dispatch()

## Release resets
for i in range(4):
    glib.write(0x50000000+2*i, 0x00400000) # <------

for i in range(4):
    glib.write(0x50000008+2*i, 0x00000000)

glib.dispatch()
```

6.5.7 Adding the Firmware

Add the most current version (`OH_3.2.2.2A.bit`) of the OH firmware to the `/mnt/persistent/gemdaq/fw` directory.

6.6 Configuring the GLIB

For the first installation or if there is a power cycle, the GLIB needs to be reconfigured. First, we reset the GTXs. Run the following command from within the container:

```
$ reset_gtx.py
```

A successful output of resetting the GTX is shown below:

```
[gemuser@glib-shelf01-slot03 ~]$ reset_gtx.py
03-04-20 07:40:54.833952 [7f312409d740] INFO - URI "chtcp-2.0://gem-shelf01-controlhub:10203?target=192.168.80.3:50001"
parsed as:
 > protocol : chtcp-2.0
 > hostname : gem-shelf01-controlhub
 > port : 10203
 > path :
 > extension :
 > arguments :
   > target = 192.168.80.3:50001

03-04-20 07:40:54.834541 [7f312409d740] INFO - Converted IP address string "192.168.80.3:50001"
to 192.168.80.3:50001 and converted this to IP 0xC0A85003, port 0xC351
```

```
03-04-20 07:40:54.835477 [7f312409d740] INFO - Attempting to create TCP connection
to 'gem-shelf01-controlhub' port 10203.
03-04-20 07:40:54.835994 [7f312409d740] INFO - TCP connection succeeded
```

Now, we will update the database within the container and inhibit Time, Trigger, and Control (TTC) communication[10] from the backplane. Execute the following commands from the local machine (i.e., not inside of the container):

1. First, run

   ```
   $ gem_reg.py
   ```

   which will open a connection to the CTP7's Zynq-SoC emulator (`gem-emulator`).

2. Now, connect to registers on the glib by running

   ```
   CTP7 > connect gem-shelf01-amc03
   ```

3. The address tables must be updated anytime the firmware is updated (or if running for the first time). We also need to update the database after updating or symbolically linking the address tables for the first time. Execute

   ```
   gem-shelf01-amc03 > update_lmdb /mnt/persistent/gemdaq/xml/gem_amc_top.
       xml
   ```

   Successful output looks like

   ```
   gem-shelf01-amc03 > update_lmdb /mnt/persistent/gemdaq/xml/gem_amc_top.xml
   LMDB address table updated
   ```

4. Now, we write a 1 to the `GEM_AMC.TTC.CTRL.INHIBIT_TTC_FROM_BACKPLANE` node:

   ```
   gem-shelf01-amc03 > write GEM_AMC.TTC.CTRL.INHIBIT_TTC_FROM_BACKPLANE 1
   ```

From inside of the `/mnt/persistent/gemdaq/fw` directory, run

```
$ load_gemloader_bitfile.py OH_3.2.2.2A.bit
```

to load the `.bit` file into the temporary memory of the GLIB. Successful programming of the OH is shown below:

```
[gemuser@glib-shelf01-slot03 fw]$ load_gemloader_bitfile.py OH_3.2.2.2A.bit
03-04-20 08:18:22.997896 [7f060c4fb740] INFO - URI "chtcp-2.0://gem-shelf01-controlhub:10203?target=192.168.80.3:50001"
parsed as:
 > protocol : chtcp-2.0
 > hostname : gem-shelf01-controlhub
 > port : 10203
 > path :
 > extension :
 > arguments :
   > target = 192.168.80.3:50001

03-04-20 08:18:22.998429 [7f060c4fb740] INFO - Converted IP address string "192.168.80.3:50001"
to 192.168.80.3:50001 and converted this to IP 0xC0A85003, port 0xC351
('Bitstream size : ', 3825896)
03-04-20 08:18:24.267676 [7f060c4fb740] INFO - Attempting to create TCP connection to 'gem-shelf01-controlhub' port 10203.
03-04-20 08:18:24.268636 [7f060c4fb740] INFO - TCP connection succeeded
File correctly written.
```

7 Electronics Integration Procedure

This section provides an overview of the various tests we run for the GE2/1 frontend electronics. An overview of the LV control and the connectivity testing and calibration routines will be discussed.

[10] TTC communication is the LHC's system for distributing all timing related signals; i.e., the Level 1 trigger, synchronous timing, etc., and was developed by the RD12 collaboration [43].

7.1 Low Voltage Control

Low voltage and high voltage are provided by a CAEN SY5527 mainframe [15]. The LV board is a CAEN A2519 [16] and the HV board is a CAEN A1515 [23]. To connect via `ssh`, perform the following commands:

1. Connect to the administrator account via `ssh`

```
$ ssh admin@192.168.0.1
```

2. Hit `return` to select the `Main` option, and then select `Channels`:

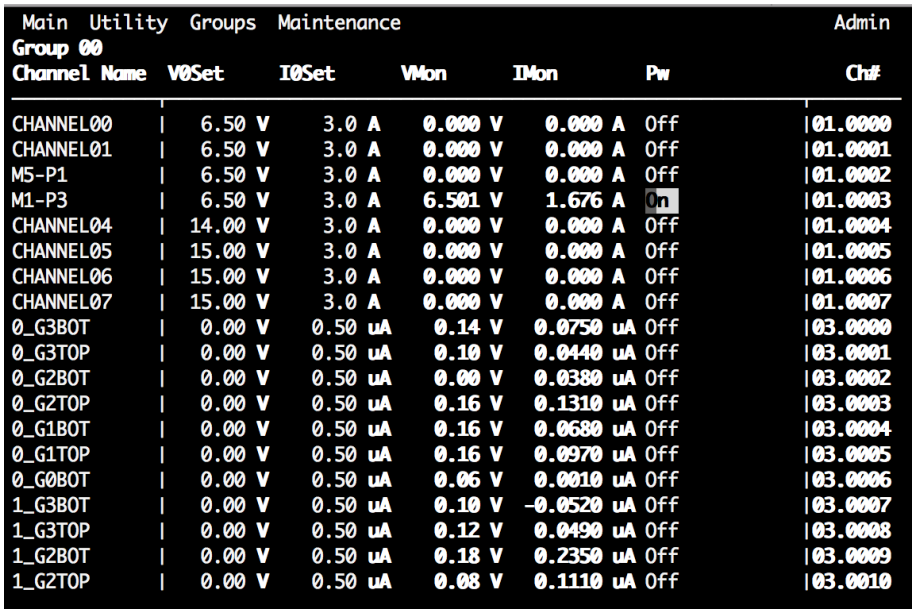


Figure 21: The power supply control window.

To navigate, use the arrows and the return key for setting voltages and turning the channels on and off. To log out (which is important after the end of a session as only one person can access the power supply at a time), use the `tab` key, which will move you to the menu bar at the top of the screen. From here select `Main > Logout`.

7.2 Connectivity Testing

When performing electronics integration, an important step is the initial connectivity testing of the frontend electronics; i.e., making sure the frontend and backend can communicate properly. First, we would like to determine if we can communicate with the GBT chips, which facilitate communication between the frontend electronics and the backend DAQ PC. This is the first step of the all-in-one script, `testConnectivity.py`. During the first step, the GBTs are tested both before and after programming. The next step consists of establishing communication with the SCA chip, which is responsible for the slow control protocol communication. The third step involves programming the FPGA on the optohybrid with the firmware, and checking if the trigger links are active. Next, the GBT phase scans are executed, which scans the clock phases of all VFATs, and determines which phase aligns to the common clock phase of the GBTs. This ensures that communication is synchronized between the OH and each VFAT chip. After the best phases are found, the phases are programmed, and the VFATs are checked for any synchronization errors.

7.2.1 Executing the Connectivity Test for the Frontend Electronics

To run this script, execute the following commands:

1. First, run the script presented in section 6.3.10. When prompted, enter your CERN credentials. This `bash` script opens a tunnel to the databases (DBs) hosted at CERN.

2. To test *just*[11)] the connectivity of the OH/VFATs, navigate to the home directory (`~/`), and run

```
$ testConnectivity.py <flags> --skipDACScan --skipS-curve --gemType <ge11, ge21, me0> --detType <long,
      short, m1,...,m8> <shelf> <slot> <OH mask> 2>&1 | tee logs/log_name_GEB_OHX_YYYYMMDD.txt
```

For example, the command will look like (the example given here is for M5-P1 GEB and the command is executed from the user's `home` directory, $\sim$/):

```
$ testConnectivity.py -a --gemType ge21 --detType m5 --skipDACScan --skipS-curve 1 3 0x1 2>&1 | tee
     logs/testConnectivity_M5-P1_OH10_20191219.txt
```

Here, we included the `-a` flag to accept bad trigger links and skipped both the DAC scan and S-curves. Although optional, it is a best practice to always `tee` the standard error and output to a log file.

Note that all of the scripts used for electronics integration can be run with the `help` flag, i.e.,

```
$ scriptName.py -h
```

which will display the help menu.

Successful output looks like:

```
[user@localhost ~]$ testConnectivity.py -a --gemType ge21 --detType m8 --skipDACScan --skipS-curve 1 3 0x1\
2>&1 | tee logs/M8-P2_20200814.txt
Open pickled address table if available  /home/data/user/address_table/amc_address_table_top.pickle...
Initializing AMC gem-shelf01-amc03
====================
Step 1: Checking GBT Communication
====================
Checking GBT Communication (Before Programming GBTs)
Programming GBTs
Checking GBT Communication (After Programming GBTs)
GBT Communication Established
====================
Step 2: Checking SCA Communication
====================
An exception has been caught while attempting to disable the ADC monitoring.
If you use a CTP7 with a firmware version higher than 3.8.3 you can safely ignore this warning.

---- Reseting the SCA ----
Writing masked reg GEM_AMC.SLOW_CONTROL.SCA.CTRL.SCA_RESET_ENABLE_MASK failed. Exiting...
wReg output -1
--========================================--
-> GEM SYSTEM SCA INFORMATION
--========================================--

READY             0x00000001
CRITICAL_ERROR    0x00000000
NOT_READY_CNT_OH00 0x00000002
NOT_READY_CNT_OH01 0x00000001
SCA Communication Established
====================
Step 3: Programming FPGA & Checking Trigger Links
====================
14 Aug 2020 13:10:24.684 [7f8e49148740] INFO - wrappers::runCommandWithOutput <> -
Executing command: ssh gemuser@gem-shelf01-amc03 sh -c "mpeek 0x6a000000"
--========================================--
-> GEM SYSTEM SCA INFORMATION
--========================================--

READY             0x00000001
CRITICAL_ERROR    0x00000000
NOT_READY_CNT_OH00 0x00000002
NOT_READY_CNT_OH01 0x00000001
FPGA Communication Established
Checking trigger link status:
Reset trigger module on OH0
Reseting trigger module on CTP7
--========================================--
-> GEM SYSTEM TRIGGER LINK INFORMATION
--========================================--

                      |  OH0
LINK0_MISSED_COMMA_CNT  | 0xffff
LINK1_MISSED_COMMA_CNT  | 0xffff
LINK0_OVERFLOW_CNT      |  0x0
LINK1_OVERFLOW_CNT      |  0x0
LINK0_UNDERFLOW_CNT     | 0xffff
LINK1_UNDERFLOW_CNT     | 0xffff
LINK0_SBIT_OVERFLOW_CNT |  0x0
LINK1_SBIT_OVERFLOW_CNT |  0x0
Trigger link of OHs: [0] failed, but I was told to accept bad trigger links
Trigger Link Successfully Established
====================
Step 4: Checking VFAT Communication
====================
Checking GBT Communication (After Programming FPGA)
GBT Communication Is Stil Good
Scanning GBT Phases, this may take a moment please be patient
====================
Phase Scan Results for OH0
====================
  Phase    VFAT0    VFAT1    VFAT2    VFAT3    VFAT4    VFAT5    VFAT6    VFAT7    VFAT8    VFAT9    VFAT10    VFAT11
-------  -------  -------  -------  -------  -------  -------  -------  -------  -------  -------  --------  --------
```

[11] If you want to perform a preliminary DAC scan and S-curves, then omit the options `--skipDACScan` and `--skipS-curve`.

```
      0       50       50       50       50       50        0       50       50        0       49       50       50
      1       50       50        0       50        0       49       50       50       50       50       50       50
      2       50       50       50       50       50       50       50       50       50       50        0       49
      3       50       50       50       50       50       50       50       50       50       50       50       50
      4       50       50       50       50       50       50        0       49       50       50       50       50
      5       50       50       50       50       50       50       50        0       50        0       50       50
      6       50       50       50        0       49       50       50       50       50       50       50       50
      7        0       50       50       50       50       50       50       50        0       50       50       50
      8       50       50       50       50       50        0       49       50       50       50       50       50
      9       50       50       50       50        0       49       50       50       50       50       50       50
     10       50       50       50       50       50       50       50       50       50       50        0       50
     11       50       50       50       50       50       50       50       50       50       50       50       50
     12       50       50       50       50       50       50       50       50       50       50       50       50
     13       50       50       50       50       50       50       50        0       50        0       50       50
     14       50       50       50        0       50       50       50       50       50       50       50       50
     15        0        0        0        0        0        0        0        0        0        0        0        0
Phase 11 will be used for (OH0,VFAT0)
Phase 0 will be used for (OH0,VFAT1)
Phase 5 will be used for (OH0,VFAT2)
Phase 10 will be used for (OH0,VFAT3)
Phase 5 will be used for (OH0,VFAT4)
Phase 5 will be used for (OH0,VFAT5)
Phase 0 will be used for (OH0,VFAT6)
Phase 0 will be used for (OH0,VFAT7)
Phase 4 will be used for (OH0,VFAT8)
Phase 4 will be used for (OH0,VFAT9)
Phase 6 will be used for (OH0,VFAT10)
Phase 6 will be used for (OH0,VFAT11)
Writing Found Phases to frontend
GBT Phases Successfully Writtent to Frontend
14 Aug 2020 13:13:24.483 [7f8e49148740] INFO - utils::_init_num_threads <> - NumExpr defaulting to 4 threads.
====================
Step 5: Checking VFAT Synchronization
====================
--========================================--
-> GEM SYSTEM VFAT INFORMATION
--========================================--

                   |  OH0
VFAT0.SYNC_ERR_CNT  |  0x0
VFAT1.SYNC_ERR_CNT  |  0x0
VFAT2.SYNC_ERR_CNT  |  0x0
VFAT3.SYNC_ERR_CNT  |  0x0
VFAT4.SYNC_ERR_CNT  |  0x0
VFAT5.SYNC_ERR_CNT  |  0x0
VFAT6.SYNC_ERR_CNT  |  0x0
VFAT7.SYNC_ERR_CNT  |  0x0
VFAT8.SYNC_ERR_CNT  |  0x0
VFAT9.SYNC_ERR_CNT  |  0x0
VFAT10.SYNC_ERR_CNT |  0x0
VFAT11.SYNC_ERR_CNT |  0x0
VFATs are properly synchronized
Checking VFAT Communication
VFAT Communication Successfully Established
Connectivity Testing Passed for OH's in 0x1
Goodbye
```

An unsuccessful scan can occur for many reasons. If the procedures listed in the previous sections have been followed exactly, the issue is most likely a result of the hardware and not the software or firmware. The scan can fail at any point in the five steps, so troubleshooting should be done a case-by-case basis. However, two common issues for the connectivity testing routine to fail are either having an improper fiber connection (i.e., the fibers are not connected to the appropriate GBTs[12)]), or the connectors on the optical fibers are dirty. Another issue could be a bad VFAT; if the GBT phases are not staying synced, or if the routine fails find the proper phase for a particular VFAT, that VFAT should be swapped. A good first step is to power cycle the GEB and try the routine again before pursuing other options. Output is displayed below for failure at the first step:

```
[user@localhost ~]$ testConnectivity.py -a --gemType ge21 --detType m6 --skipDACScan --skipS-curve\
1 3 0x2 2>&1 | tee logs/M6-P3_20200815.txt
Open pickled address table if available  /home/data/user/address_table/amc_address_table_top.pickle...
Initializing AMC gem-shelf01-amc03
====================
Step 1: Checking GBT Communication
====================
Checking GBT Communication (Before Programming GBTs)
Programming GBTs
Checking GBT Communication (After Programming GBTs)
GBT Communication was not established successfully
        Try checking:
                1. Fibers from GE1/1 patch-panel to OH have correct jacket color ordering
                2. Fibers from GE1/1 patch-panel to OH are fully inserted
                3. OH3 screw is properly screwed into standoff
                4. OH3 standoff on the GEB is not broken
                5. Voltage on OH3 standoff is within range [1.47,1.59] Volts
Connectivity Testing Failed
Goodbye
```

In this case, a power cycle fixed the issue.

12) To check if the fibers are improperly cabled, one can quickly check the monitored current being drawn at the power supply: if the current is not around 1 A (and not the nominal 1.2 A) with 6.5 V applied to the terminals of the GEB, then the GBTs are not "locked," and indicates that the GBT links are not properly connected. Another issue for the GBTs not locking is that the transmission line polarity is not inverted (see Sec. 6.5.6).

7.3 Retrieving VFAT Calibration Data

After establishing connectivity with the front-end electronics, the calibration data determined from the VFAT quality control tests need to be retrieved from the database. These calibration values are for the global registers on the VFAT; without programming the appropriate values to these registers, nominal operation is impossible. In Tab. 3 below, these registers are listed with a brief description.

Table 3: Global Registers

Nominal Voltage (V)	Tolerance Range (V)
IREF	Reference current after the bandgap circuit in the CBM (used to generate **all** references for the front-end and the CFD)
ADC0	ADC for monitoring the CBM unit; internally generated bias and programmable
ADC1	ADC for monitoring the CBM unit; referenced from the power supply
cal_DAC	Register that controls the calibration DAC (for generating calibration pulses)
VREF_ADC	Voltage reference for the ADC

The procedure below enumerates this process:

1. Connect to the DB using the script presented in section 6.3.10.
2. Now, execute

```
$ getCalInfoFromDB.py --write2File --gemType <ge11, ge21, me0> --detType <long, short, m1,...,m8> <
    shelf> <slot> <link number>
```

An example is given for M1-P3:

```
$ getCalInfoFromDB.py --write2File --gemType ge21 --detType m1 1 3 1
```

Typical output of this script looks like

```
[user@localhost ~]$ getCalInfoFromDB.py --gemType ge21 --detType m1 1 3 1 --write2File
Open pickled address table if available  /home/user/data/address_table//amc_address_table_top.pickle...
Initializing AMC gem-shelf01-amc03
Length of returned view does not match length of input vfat List
VFATs not found: []
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12 entries, 0 to 11
Data columns (total 10 columns):
vfatN            12 non-null int64
vfat3_ser_num    12 non-null object
vfat3_barcode    12 non-null object
iref             12 non-null int64
adc0m            12 non-null float64
adc1m            12 non-null float64
adc0b            12 non-null float64
adc1b            12 non-null float64
cal_dacm         12 non-null float64
cal_dacb         12 non-null float64
dtypes: float64(6), int64(2), object(2)
memory usage: 1.0+ KB
    vfatN vfat3_ser_num vfat3_barcode  iref    adc0m    adc1m    adc0b  \
0       0        0x2060          8288    29  1.89749  2.25444 -331.047
1       1        0x1dfa          7674    31  1.88417  2.27807 -322.126
2       2        0x1ae0          6880    37  1.88539  2.20609 -303.542
3       3        0x1c15          7189    31  1.88738  2.22184 -320.445
4       4        0x1e8d          7821    32  1.88890  2.21770 -316.424
5       5        0x1c87          7303    32  1.91411  2.26294 -322.830
6       6        0x1b0a          6922    39  1.87150  2.24606 -315.997
7       7        0x1c70          7280    34  1.84515  2.26428 -294.113
8       8        0x1ee7          7911    31  1.89948  2.24375 -327.146
9       9        0x1e58          7768    30  1.93774  2.25831 -316.149
10     10        0x1a9d          6813    34  1.89549  2.28178 -318.150
11     11        0x1eaf          7855    33  1.88489  2.22407 -307.662

      adc1b  cal_dacm  cal_dacb
0  -494.560 -0.257674   63.5795
1  -515.364 -0.262623   64.8303
2  -487.140 -0.272533   67.5885
3  -489.001 -0.227987   56.0251
4  -479.794 -0.251086   62.4459
5  -497.602 -0.263442   65.0627
6  -502.871 -0.243721   59.9056
7  -508.646 -0.228780   56.4081
```

```
8  -501.236 -0.245757   60.6185
9  -496.356 -0.222763   54.7832
10 -504.218 -0.219229   53.8976
11 -476.732 -0.264201   65.4966
Writing 'CFG_IREF' to file: /home/user/data/data//GE21-M1-P3/NominalValues-CFG_IREF.txt
Writing 'ADC0' Calibration file: /home/user/data/data//GE21-M1-P3/calFile_ADC0_GE21-M1-P3.txt
Writing 'CAL_DAC' Calibration file: /home/user/data/data//GE21-M1-P3/calFile_calDac_GE21-M1-P3.txt
goodbye
```

Now, we will need to update the `CFG_IREF` register values for all VFATs on the GEB. We can accomplish this by using the `replace_parameter.sh` script in the `/mnt/persistent/gemdaq/scripts` directory. First, we need to transfer the calibration file produced by `getCalInfoFromDB.py` saved in the module folder (i.e., `$DATA_PATH/GE21-MX-PY/`, to the base directory of the `gemuser` profile in the `gem-emulator` (`/mnt/persistent/gemuser`). After transferring, change directories to `/mnt/persistent/gemdaq/scripts` and run the `replace_parameter.sh` script. An example is provided below for the `IREF` values of the M6-P3 GEB:

```
[gemuser@glib-shelf01-slot03 scripts]$ ./replace_parameter.sh -f ../../gemuser/NominalValues-CFG_IREF.
    txt IREF 1
```

Usage of the script is reproduced below:

```
./replace_parameter.sh [-f <FILENAME>] <REGISTER> <LINK> <VALUE>

REGISTER - register to be updated dropping the "CFG_" substring
VALUE - value in decimal to be assigned to REGISTER
FILENAME - name of a file containing a list of vfat number/value pairs
Examples:
./replace_parameter.sh PULSE_STRETCH 1 4
./replace_parameter.sh -f /path/to/NominalDacValues.txt PULSE_STRETCH 1
```

7.4 Performing a DAC Scan

DAC register values are needed for proper calibration and operation of the VFATs. These registers are discussed in Sec. 3.3.1. DAC scans are performed in parallel for each programmable DAC on the VFAT3 ASIC. For each DAC, a range of register values are programmed, and the resulting bias current/voltage is read by the internal ADC on the ASIC. It is vital that the production calibration values are programmed, as the internal reference current (`CFG_IREF`) is responsible for generating all other reference currents/voltages—without this register value properly set, no other DAC values can be properly determined.

One can perform a DAC scan during the connectivity testing routine, omitting the flag `--skipDACscan`, or, one can take a scan manually using the `run_scans.py` tool[13]:

1. Run

```
$ run_scans.py --gemType ge21 --detType <m1,...,m8> dacScanV3 <shelf>
    <slot> <link>
```

An example is given for the M8-P2 GEB:

```
$ run_scans.py --gemType ge21 --detType m8 dacScanV3 1 3 0x1
```

Successful output is displayed below:

```
[user@localhost ~]$ run_scans.py --gemType ge21 --detType m8 dacScanV3 1 3 0x1
14 Aug 2020 13:39:06.540 [7f2333107740] INFO - wrappers::runCommand <>\
 - Executing command: mkdir -p /home/data/data/dacScanV3/2020.08.14.13.39
14 Aug 2020 13:39:06.560 [7f2333107740] INFO - wrappers::runCommand <>\
 - Executing command: chmod g+rw /home/data/data/dacScanV3/2020.08.14.13.39
14 Aug 2020 13:39:06.565 [7f2333107740] INFO - wrappers::runCommand <>\
 - Executing command: unlink /home/data/data/dacScanV3/current
14 Aug 2020 13:39:06.577 [7f2333107740] INFO - wrappers::runCommand <>\
 - Executing command: ln -s 2020.08.14.13.39 /home/data/data/dacScanV3/current
```

[13] This scan can only be taken after connectivity has been successfully established.

```
14 Aug 2020 13:39:06.591 [7f2333107740] INFO - wrappers::runCommand <>\
 - Executing command: dacScanV3.py --gemType=ge21 --detType=m8 -f\
  /home/data/data/dacScanV3/2020.08.14.13.39/dacScanV3.root 1 3 0x1
14 Aug 2020 14:25:06.933 [7f2333107740] INFO - wrappers::runCommand <>\
 - Executing command: chmod -R g+rw /home/data/data/dacScanV3/2020.08.14.13.39
Finished DAC scans for optohybrids on shelf1 slot3 in ohMask: 0x1
Good-bye
```

2. Once the DAC scan has been completed, transfer all calibration files (`*.txt`) in the module folder, i.e., `$DATA_PATH/GE21-MX-PY/dacScans/current`, to the `gemuser` directory of the `gem-emulator`. The calibration files are reproduced below:

```
NominalValues-CFG_BIAS_SD_I_BSF.txt
NominalValues-CFG_BIAS_PRE_I_BLCC.txt
NominalValues-CFG_BIAS_SD_I_BDIFF.txt
NominalValues-CFG_BIAS_PRE_I_BSF.txt
NominalValues-CFG_HYST.txt
NominalValues-CFG_BIAS_CFD_DAC_1.txt
NominalValues-CFG_BIAS_SH_I_BFCAS.txt
NominalValues-CFG_BIAS_CFD_DAC_2.txt
NominalValues-CFG_BIAS_PRE_I_BIT.txt
NominalValues-CFG_BIAS_SD_I_BFCAS.txt
NominalValues-CFG_BIAS_SH_I_BDIFF.txt
NominalValues-CFG_BIAS_PRE_VREF.txt
```

An example DAC scan summary plot is reproduced in Fig. 22 below. (Note that the independent variable (programmed DAC value) is on the y-axis.)

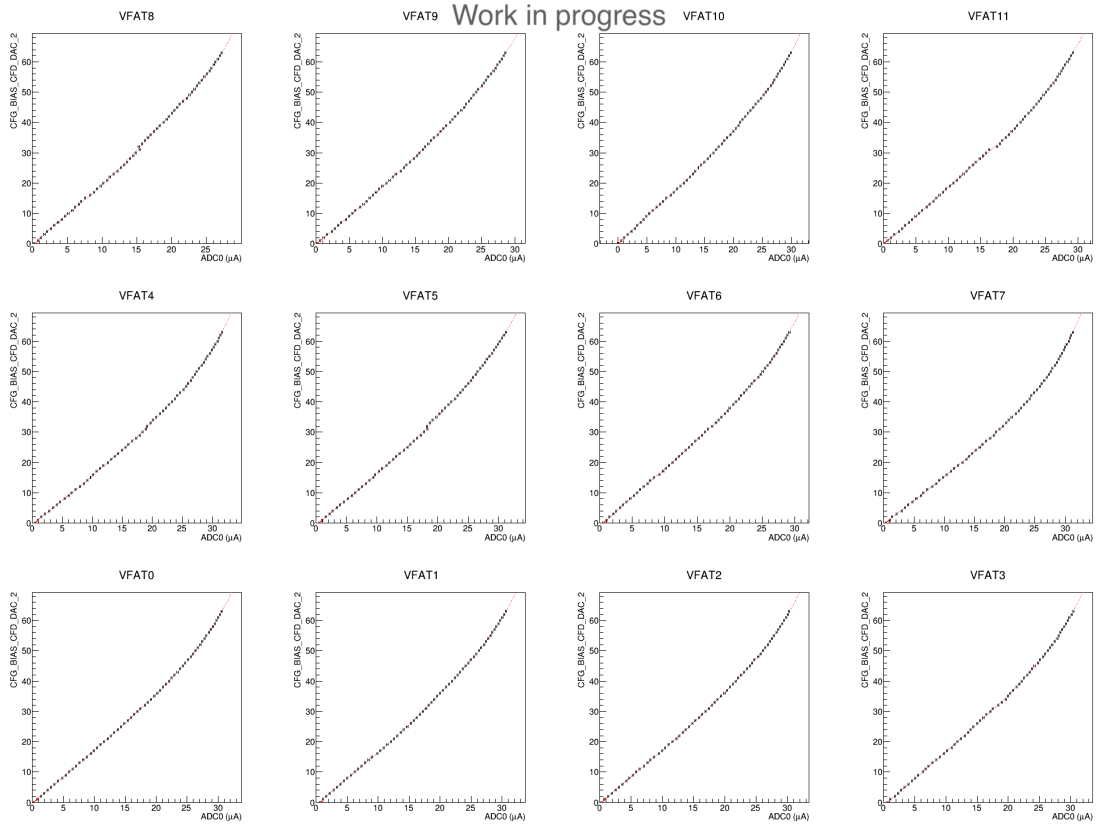


Figure 22: Example DAC scan summary plot for the second DAC of the CFD [44].

7.5 S-curves

This procedure provides us with the equivalent noise charge (ENC) in the electronics/detector system. The ENC of the system is the noise present in the detector pre-amplifier/amplifier chain. To take an S-curve, first the `THR_ARM_DAC` value is set. A calibration pulse of fixed charge is injected into one channel on each VFAT. The voltage comparator then records whether there is a response or not (theoretically, if the pulse is below the threshold, the comparator will not sense any signal). This is repeated multiple times for increasing levels of injected charge. Then, the procedure is repeated for the next channel on the VFAT, until all 128 channels are scanned.

Theoretically, if we make a histogram of the ideal voltage comparator's response for a set threshold and, with the x-axis bins representing the injected charge, we would see a simple step function (nothing below the threshold is sensed, and everything above the threshold is recorded). Physically, however, there is noise in the system, which, near the threshold, will influence the comparator's response (sensing a pulse that is actually below the threshold). Because of this effect, the curve is actually a sigmoid function (see Fig. 23).

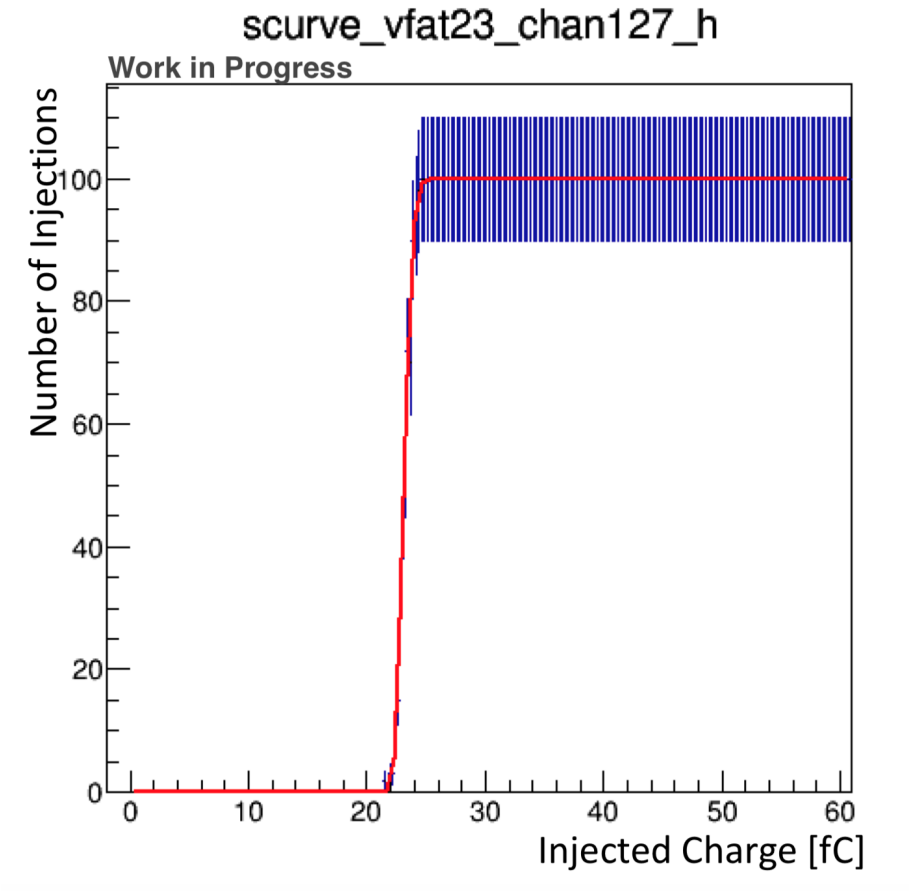


Figure 23: S-curve for a single channel on one VFAT. Adapted from [45].

The data for each channel are fit with a modified error function, given by (1) below [45].

$$f(q) = A \operatorname{erf}\left(\frac{\max(P, q) - \mu}{\sigma\sqrt{2}}\right) \tag{1}$$

where q is the injected charge, A is the number corresponding to half of the total number of charge injections, P is the pedestal (the amount of comparator responses at zero injected charge), μ is the set threshold, i.e., the error function's mean value (located at the half-max point on the curve), and σ is the ENC.

The resulting sigma of this fit (essentially quantifying the "width" of this curve), gives us an estimate of the noise. In Fig. 24, we can see the S-curve sigma distribution for all VFATs at a particular `THR_ARM_DAC` level. In Fig. 25, we can see a 2D histogram of the S-curves for each channel of one VFAT.

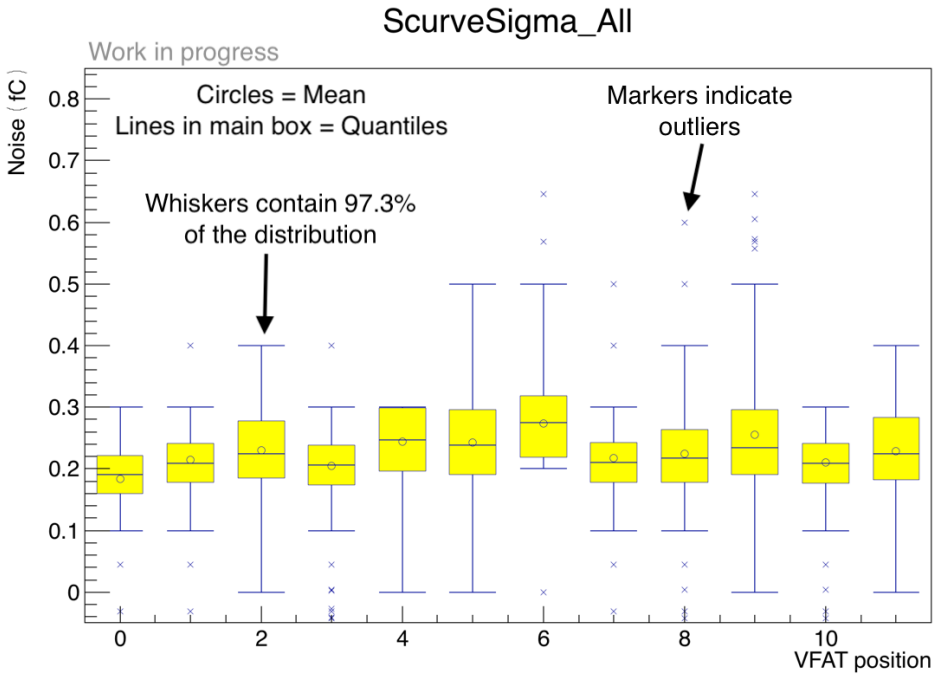


Figure 24: An example S-curve sigma distribution summary plot. Note that outliers above 0.85 fC are not displayed on this plot.

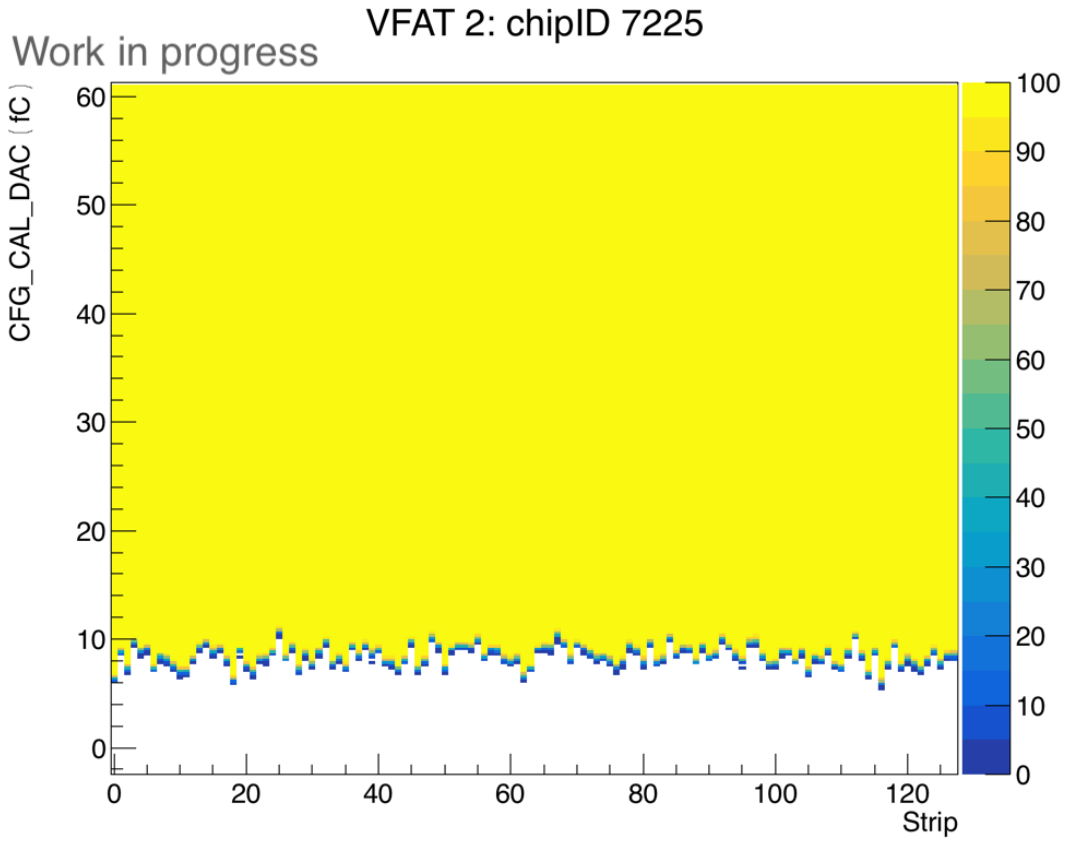


Figure 25: An example two-dimensional histogram of S-curves for one VFAT [44].

7.5.1 Taking an S-curve

To take one S-curve, connectivity testing needs to be successful (see section 7.2). First we configure the chamber, placing the VFATs into run mode (command and successful output shown below):

```
[user@localhost ~]$ confChamber.py --gemType ge21 --detType m8 --vfatmask=0xfff000 --shelf 1 -s 3 -g 0 --run
2020.04.07.15.27
Open pickled address table if available  /home/user/data/address_table//amc_address_table_top.pickle...
Initializing AMC gem-shelf01-amc03
opened connection
Configuring VFATs on (shelf1, slot3, OH0) with chamber_vfatDACSettings dictionary values
biased VFATs on (shelf1, slot3, OH0)
Set CFG_THR_ARM_DAC to 100
VFATs on (shelf1, slot3, OH0) set to run mode
Chamber Configured
```

Now, launch the scans with the `run_scans.py` tool:

```
$ run_scans.py --gemType ge21 --detType m8 S-curve 1 2 0x1
```

To analyze the S-curves, run

```
$ anaUltraS-curve.py -o NameOfOutputFile.root -e /home/user/repos/gem-plotting-tools/mapping/
    shortChannelMap_VFAT3-HV3b-V3.txt --calFile path/to/calFile/calFile_calDac_GE21-M6-P3.txt path/to/S-
    curve/S-curveData.root m6
```

Note here that since the GE2/1 mapping was not available at the time this note was written, we used the GE1/1 short channel mapping.

7.6 Taking S-curves at Multiple `THR_ARM_DAC`

To take and analyze multiple S-curves at different `THR_ARM_DAC` values, we use the `calibrateArmDac.sh` script, to run this procedure automatically.

1. Change directories to `repos/sw_utils/scripts` and run the following command:

   ```
   ./calibrateArmDac.sh -g <gemType> -D <detType> -d <GEB> -S <shelf> -s <slot> -l <OH link number> -m <
       vfat mask> -L <comma-separated list of THR_ARM_DAC values>
   ```

   A full example looks like:

   ```
   ./calibrateArmDac.sh -g ge21 -D m1 -d GE21-M1-P3 -S 1 -s 3 -l 1 -m 0xfff000 -L
       10,20,30,40,50,60,70,80,90,100,110,120,130,140,150,160,170,180,190,200,210,220,230,240,250
   ```

2. The output S-curve files will be located at a path in the form of:

   ```
   /data/data/{Detector Name}/S-curve/{YYYY.MM.DD.hh.mm}
   ```

3. The summary plots of the aggregated data for all S-curves will be located at:

   ```
   /data/data/{Detector Name}/S-curve/{YYYY.MM.DD.hh.mm}/S-curveData/
       Summary
   ```

For post-analysis, the `ROOT` macro `plotS-curve_utils.cpp`, located at S. Butalla's GitHub, can be used to plot the sigma distributions vs. the `THR_ARM_DAC` value for all VFATs.

7.7 Manually Reading and Writing to Registers using `gem_reg.py`

Manually checking the registers using the `gem_reg.py` utility allows one to quickly spot any issues with the VFATs or OH. With the `gem-emulator` running, launch the command line tool `gem_reg.py`. From here, we can read and write to the registers.

7.7.1 Connecting to the GLIB Command Line Interface

1. To access the command line of the emulator, execute

```
$ gem_reg.py
```

2. Now, connect to the GLIB by executing

```
> connect gem-shelf01-amc03
```

If successfully connected, you will see that the local host information changed from `CTP7 >` to `gem-shelf01-amc03 >`.

7.7.2 Useful Commands

Outlined below are useful commands (adapted from [46] and [47]):

- `kw` reads all of the registers in the nodes for a certain substring. Example:

```
gem-shelf01-amc03 > kw OH_LINKS.OH1.GBT
```

- `rwc` reads all of the registers in the nodes for a certain substring with the use of wildcards. Example:

```
gem-shelf01-amc03 > rwc *OH0*VFAT*SYNC_ERR*
```

- `read` reads a specific node name. Example:

```
gem-shelf01-amc03 > read GEM_AMC.OH_LINKS.OH1.VFAT0.LINK_GOOD
```

- `write` writes a value to a specific node name. Example:

```
gem-shelf01-amc03 > write GEM_AMC.OH.OH1.GEB.VFAT0.CFG_IREF 0x0000001D
```

- `doc` prints more information for a command or a register. Example:

```
gem-shelf01-amc03 > doc GEM_AMC.OH.OH1.GEB.VFAT0.CFG_IREF
```

7.7.3 Reading and Writing to the Registers

- To issue a link reset to the GBTs, run the following command:

```
gem-shelf01-amc03 > write GEM_AMC.GEM_SYSTEM.CTRL.LINK_RESET 1
```

- To check if there are any synchronization errors for the VFATs (for OH1 in this example), execute:

```
gem-shelf01-amc03 > rwc *OH1*VFAT*SYNC_ERR*
0x65800840 r    GEM_AMC.OH_LINKS.OH1.VFAT0.SYNC_ERR_CNT                0x00000000
0x65800848 r    GEM_AMC.OH_LINKS.OH1.VFAT1.SYNC_ERR_CNT                0x00000000
0x65800850 r    GEM_AMC.OH_LINKS.OH1.VFAT2.SYNC_ERR_CNT                0x00000000
0x65800858 r    GEM_AMC.OH_LINKS.OH1.VFAT3.SYNC_ERR_CNT                0x00000000
0x65800860 r    GEM_AMC.OH_LINKS.OH1.VFAT4.SYNC_ERR_CNT                0x00000000
0x65800868 r    GEM_AMC.OH_LINKS.OH1.VFAT5.SYNC_ERR_CNT                0x00000000
0x65800870 r    GEM_AMC.OH_LINKS.OH1.VFAT6.SYNC_ERR_CNT                0x00000000
0x65800878 r    GEM_AMC.OH_LINKS.OH1.VFAT7.SYNC_ERR_CNT                0x00000000
0x65800880 r    GEM_AMC.OH_LINKS.OH1.VFAT8.SYNC_ERR_CNT                0x00000000
0x65800888 r    GEM_AMC.OH_LINKS.OH1.VFAT9.SYNC_ERR_CNT                0x00000000
0x65800890 r    GEM_AMC.OH_LINKS.OH1.VFAT10.SYNC_ERR_CNT               0x00000000
0x65800898 r    GEM_AMC.OH_LINKS.OH1.VFAT11.SYNC_ERR_CNT               0x00000000
```

(note that only VFATs 0-11 are displayed).

- To check the SCA status on the OH:

```
gem-shelf01-amc03 > kw GEM_AMC.SLOW_CONTROL.SCA.STATUS.READY
0x66c00400 r    GEM_AMC.SLOW_CONTROL.SCA.STATUS.READY                  0x00000002
gem-shelf01-amc03 > kw GEM_AMC.SLOW_CONTROL.SCA.STATUS.CRITICAL_ERROR
0x66c00404 r    GEM_AMC.SLOW_CONTROL.SCA.STATUS.CRITICAL_ERROR         0x00000000
```

- Check the GBT links on the OH (for OH1 in this example):

```
gem-shelf01-amc03 > kw OH_LINKS.OH1.GBT
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT0_READY                          0x00000001
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT1_READY                          0x00000001
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT2_READY                          0x00000000
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT0_WAS_NOT_READY                  0x00000000
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT1_WAS_NOT_READY                  0x00000000
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT2_WAS_NOT_READY                  0x00000001
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT0_RX_HAD_OVERFLOW                0x00000000
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT1_RX_HAD_OVERFLOW                0x00000000
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT2_RX_HAD_OVERFLOW                0x00000000
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT0_RX_HAD_UNDERFLOW               0x00000000
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT1_RX_HAD_UNDERFLOW               0x00000000
0x65800800 r    GEM_AMC.OH_LINKS.OH1.GBT2_RX_HAD_UNDERFLOW               0x00000001
```

  The `kw` command is an alias for the `readKW` command, where `kw` stands for keyword[14]. Note that the number in front of `OH` needs to be changed depending on the OH link you are using (i.e., if you are using OH0, then the command will be `kw OH_LINKS.OH0.GBT`.

- Now, read all of the registers (using wildcards) using the following command:

```
CTP7 > rwc *OH1*VFAT*SYNC_ERR*
```

  If there are no sync errors, the hex value for that register will be zero (e.g., `0x000000`). If there are sync errors, first issue another link reset. If the problem persists, check if the VFATs are simply out of phase by running the `testConnectivity.py` script.

- Finally, check if all of the VFATs are in run mode:

```
CTP7 > kw CFG_RUN 1
0x65440c00 rw   GEM_AMC.OH.OH1.GEB.VFAT0.CFG_RUN                         0x00000000
0x65442c00 rw   GEM_AMC.OH.OH1.GEB.VFAT1.CFG_RUN                         0x00000000
0x65444c00 rw   GEM_AMC.OH.OH1.GEB.VFAT2.CFG_RUN                         0x00000000
0x65446c00 rw   GEM_AMC.OH.OH1.GEB.VFAT3.CFG_RUN                         0x00000000
0x65448c00 rw   GEM_AMC.OH.OH1.GEB.VFAT4.CFG_RUN                         0x00000000
0x6544ac00 rw   GEM_AMC.OH.OH1.GEB.VFAT5.CFG_RUN                         0x00000000
0x6544cc00 rw   GEM_AMC.OH.OH1.GEB.VFAT6.CFG_RUN                         0x00000000
0x6544ec00 rw   GEM_AMC.OH.OH1.GEB.VFAT7.CFG_RUN                         0x00000000
0x65450c00 rw   GEM_AMC.OH.OH1.GEB.VFAT8.CFG_RUN                         0x00000000
0x65452c00 rw   GEM_AMC.OH.OH1.GEB.VFAT9.CFG_RUN                         0x00000000
0x65454c00 rw   GEM_AMC.OH.OH1.GEB.VFAT10.CFG_RUN                        0x00000000
0x65456c00 rw   GEM_AMC.OH.OH1.GEB.VFAT11.CFG_RUN                        0x00000000
```

  here we expect to see either a `0x00000000` or `0x00000001` if the VFAT is in run mode. If not, the register will read `0xdeaddead`.

8 Summary and Conclusion

This note presents the setup and operation of a DAQ system for a GE2/1 GEM detector-specific test stand using a GLIB AMC for the backend interface. We provide a comprehensive overview of the frontend electronics for the GE2/1 GEM detector, the installation and configuration of a GLIB-based DAQ system, and the instructions for connectivity testing, and taking DAC scans and S-curves.

Acknowledgments

We would like to thank Laurent Pétré (ULB) for his extensive help with setting up the electronics integration test stand at FIT, Mykhailo Dalchenko for sharing his knowledge of GE2/1 electronics integration, Francesco Licciulli (INFN, Bari) for his discussions on the VFAT3 hybrid card, and Jérémie Alexandre and Michele Bianco for their discussions and guidance on the GE2/1 detector system. We would also like to thank Mehdi Rahmani and Jacob Chesslo (FIT), and Mike Matveev (Rice), for their detailed review of this work.

[14] For more information on the executable scripts on the CTP7 register interface, see [46].

References

[1] CMS Collaboration, "The Phase-2 Upgrade of the CMS Muon Detectors Technical Design Report," Technical Report CERN-LHCC-2017-012, CMS-TDR-016, CERN, 2017.

[2] Photo courtesy of M. Bianco.

[3] *Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics*, 2018, https://www.xilinx.com/support/documentation/data_sheets/ds181_Artix_7_Data_Sheet.pdf.

[4] P. Moreira et. al, *GBTx Manual*, 2018, https://espace.cern.ch/GBT-Project/GBTX/Manuals/gbtxManual.pdf.

[5] A. Gabrielli, G. De Robertis, D. Fiore, F. Loddo and A. Ranieri, "Architecture of a Slow-Control ASIC for Future High-Energy Physics Experiments at SLHC," in *IEEE Transactions on Nuclear Science*, **56**(3), pp. 1163-1167, June 2009, doi:10.1109/TNS.2008.2009937.

[6] *GE2/1 Optohybrid Board*, 2019, http://padley.rice.edu/cms/OH_GE21/OH_spec_012819.pdf.

[7] P. Aspell et al., *Preliminary VFAT3 User Manual*, 2018, https://espace.cern.ch/cms-project-GEMElectronics/VFAT3/VFAT3%20User%20Manual%20v2.2.pdf.

[8] CERN DCDC Project, *FEASTMP_CLP Datasheet rev. 1.0*, https://espace.cern.ch/project-DCDC-new/Shared%20Documents/FEAST2CLP_Datasheet.pdf.

[9] *Gigabit Link Interface Board (GLIB)*, https://cds.cern.ch/record/1359270/files/JINST5.C11007.pdf.

[10] Avago AFBR-57R5APZ Digital Diagnostic SFP Datasheet, https://datasheet.octopart.com/AFBR-57R5APZ-Avago-datasheet-5395050.pdf.

[11] UNIPOWER, *Power Supply System, Aspiro 2U Front Access, Instruction Manual*, 2019, https://unipowerco.com/pdf/aspiro2u-man.pdf.

[12] PICMG, MicroTCA©Overview, 2020, https://www.picmg.org/openstandards/microtca/.

[13] nVent Schroff, *User Manual: MTCA.4 Shelf*, 2016, https://schroff.nvent.com/wcsstore/ExtendedSitesCatalogAssetStore/Attachment/SchroffAttachments/Documents/63972-338.pdf.

[14] NAT-Gmbh, *NAT-MCH Users? Manual, Version 1.38*, https://www.nateurope.com/manuals/nat_mch_man_usr.pdf.

[15] CAEN SY5527 Mainframe Manual, 2019, https://www.caen.it/products/sy5527/.

[16] CAEN A2519 LV Board Manual, 2019, https://www.caen.it/products/a2519/.

[17] Digikey, Digikey product page, Amphenol ICC (FCI) DC8W8SA00LF, 8 position D-subminiature housing plug (male), https://www.digikey.com/products/en?keywords=DC8W8SA00LF.

[18] Digikey, Digikey product page, Amphenol ICC (FCI) 8638PPS2005LF, D-subminiature connector pin (male), 12 AWG, https://www.digikey.com/products/en?keywords=8638PPS2005LF.

[19] Digikey, Digikey product page, Amphenol ICC (FCI) DA3W3SA00LF, D-subminiature, 3 position D-subminiature housing plug (female), https://www.digikey.com/products/en?keywords=DA3W3SA00LF.

[20] Digikey, Digikey product page, Amphenol ICC (FCI) 8638PSS2005LF, D-subminiature connector pin (female), 12 AWG, https://www.digikey.com/products/en?keywords=8638PSS2005LF.

[21] Digikey, Digikey product page, Amphenol ICC (FCI) DA3W3PA00LF, D-subminiature, 3 position D-subminiature housing plug (male), https://www.digikey.com/products/en?keywords=DA3W3PA00LF.

[22] Digikey, Digikey product page, Amphenol ICC (FCI) 10090769-P264ALF, 26 pin high density D-sub connector (male), https://www.digikey.com/products/en?keywords=%E2%80%8B10090769-P264ALF%E2%80%8B.

[23] CAEN A1515 HV Board Manual, 2019, https://www.caen.it/products/a1515/.

[24] CAEN Mod. A996 HV Connector, 2019, https://www.caen.it/products/a996/.

[25] M. Ali, "FlexPCB & PlugInCard design & production status report," Presented at the *GEM Phase-2 Electronics meeting: Triad*, Jan. 14th, 2019, https://indico.cern.ch/event/788186/contributions/3274911/.

[26] L. Pétré, *Getting started with the CMS GEM DAQ*, 2018, https://lpetre.web.cern.ch/lpetre/doc/.

[27] T. Hemsley, "The Software Collections (SCL) Repository," The CENTOS Project, 2020,https://wiki.centos.org/AdditionalResources/Repositories/SCL.

[28] pip, v. 21.0.1, https://pypi.org/project/pip/.

[29] EPEL, v. 7, 2021, https://fedoraproject.org/wiki/EPEL.

[30] MySQL (subsidiary of the Oracle corporation), https://dev.mysql.com/.

[31] boost C++ Libraries, v. 1.75.0, https://www.boost.org/.

[32] A. Kapoulkine, pugixml v.1.11, https://pugixml.org/.

[33] The ROOT Project, "ROOT Data Analysis Framework," https://root.cern/.

[34] The ROOT Project, "Building ROOT from source," https://root.cern/install/build_from_source/.

[35] The ROOT Project, "Dependencies," https://root.cern/install/dependencies/.

[36] The ROOT Project, GitHub Repository: root-project/root, v.6.22/00, https://github.com/root-project/root.

[37] "XDAQ Twiki Page," https://twiki.cern.ch/twiki/bin/view/XdaqWiki/WebHome.

[38] *Cactus 4.0 Users Guide*, http://cactuscode.org/documentation/UsersGuide.pdf.

[39] "Introduction to cx_Oracle," https://cx-oracle.readthedocs.io/en/latest/user_guide/introduction.html.

[40] The CMS GEM DAQ Project Collaboration, GitHub Repository CMS-GEM-DAQ-PROJECT, 2020, https://github.com/cms-gem-daq-project.

[41] S. Colafranceschi, "Data Acquisition System for CMS GEM", 2018.

[42] L. Pétré, "Software emulator for the GEM backends," 2020, https://gitlab.cern.ch/lpetre/gem-emulator/-/blob/master/README.md.

[43] RD12

[44] S. Butalla & M. Hohlmann, "Frontend Electronics Integration for the GE2/1 GEM Detector for the Phase-2 Muon System Upgrade of the CMS Experiment," April 2020 APS Meeting, Washington, D.C..

[45] B. Dorney, "Explanation of S curve Fit Algorithm," Internal CMS Presentation at the GEM Phase-2 Electronics Meeting, https://indico.cern.ch/event/780422/contributions/3252280/attachments/1771243/2879820/BDorney_GEMDAQMtg_20181213_S-curve.pdf.

[46] CMS GEM DAQ Project, "Using ge_reg.py," 2020, https://test-gemdaq-ci.web.cern.ch/test-gemdaq-ci/guides/expertguide/electronics/backend/utca/ctp7/gem-reg.html.

[47] B. Dorney, A. Levin, L. Petre, R. Sharma, B. Radburn-Smith, v3ElectronicsUserGuide.md, 2019, https://github.com/cms-gem-daq-project/sw_utils/blob/develop/v3ElectronicsUserGuide.md#programming-oh-fpga.