The GE2/1 Electronics Integration Test Stand at FIT

An Overview of the Hardware, Software, and the Procedure of Electronics Integration

Stephen D. Butalla

December 20, 2019

Contents

Chapter 1

Introduction

This manual, written during the summer of 2019, documents the current version of the hardware, software, and the procedure for electronics integration for the GE2/1 gas electron multiplier (GEM) detector. The first version of this manual was originally written at CERN and has since been adapted to the specific procedures at FIT. Individual connectivity tests and sundry material are covered in the appendices.

It is best stay up to date with the current repositories of code used in this project; please see the GitHub repositories [1, 2]. This manual is meant to be as holistic and comprehensive as possible; if you have any questions or ways this manual can be improved, please feel free to contact me at `stephen.butalla@cern.ch`.

Chapter 2

Hardware

This section provides an abridged overview of the essential electronics used for electronics integration. 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.

2.1 The Optohybrid

The Optohybrid (OH) provides a means of communicating with the VFAT3 (Very Forward ATLAS and TOTEM) application specific integrated circuit (ASIC) chips mounted on the GEM electronics board (GEB). This OH board was redesigned for the GE2/1 to better accommodate the needs of the GE2/1. It features an Artix-7 [3] field programmable gate array (FPGA) and two gigabit transfer (GBTX) chips which receive communication through the VTRX (receiver) and the VTTX (transmitter) (see figure 2.1 below). The SCA is an ASIC dedicated solely to slow control applications. The GE2/1, originally designed for just one OH per two modules [5], now will include four OHs per chamber. Two optohybrids will be joined together by a master/slave connector, and will reduce the number of output optical links to the patch panel.

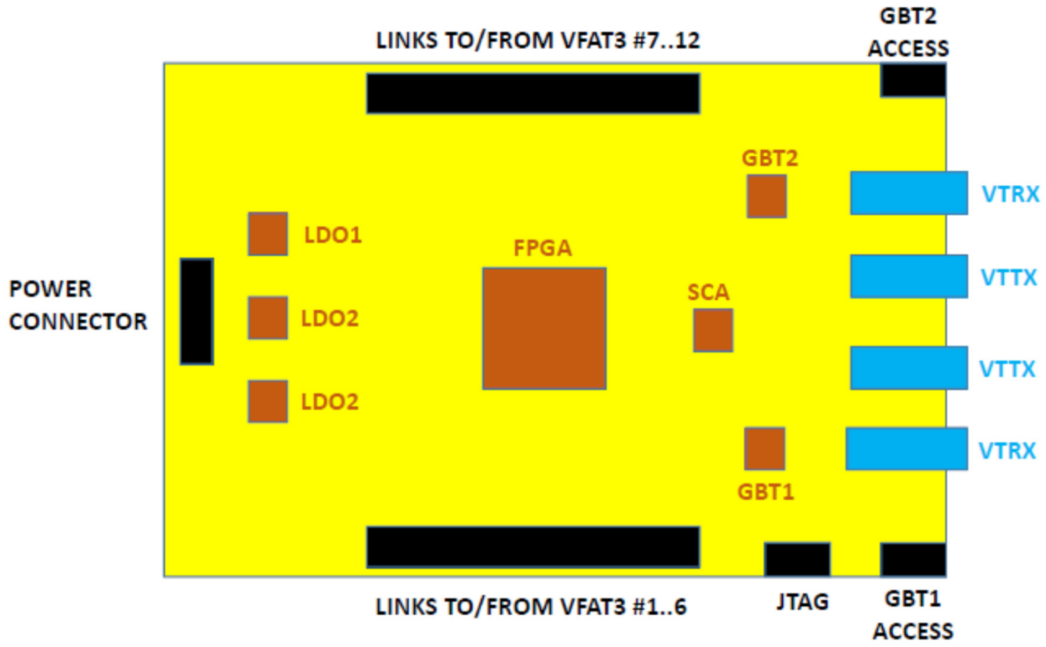

Figure 2.1: Schematic of the OH [4].

In the subsequent subsections, the different hardware components that comprise the OH will be briefly detailed.

2.1.1 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) [6]. This chip interfaces with two optical fibers, allowing it to be bidirectional, which can provide both data transmission/monitoring and communication. It is a versatile chip which allows for many modes of operation [6] and can be operated with both 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 figure 2.1) communicates with VFATs 1-6 and also the SCA. If communication is not established to this GBT, communication will not be able to be established with GBT1[1]. GBT1 (GBT2 in the digram in the diagram in figure 2.1), communicates with VFATs 7-12. The VTRX transceivers connect these GBTs to the backend electronics (i.e., the μTCA). The two optical transceivers in between these VTRXs are the VTTX transmitters.

2.1.2 Mounting the OH on the GEB and Connecting the Optical Fibers

Before attaching any electronics, ensure that the GEB is powered off. Connecting the OH to the GEB is a simple task: simply alight the power connector pins on the GEB with the power connector on the OH, and then press lightly on the sides where the Hirose connector is. To connect the fibers, we need to make sure that we are connecting to the appropriate GBT, and the fibers are in the correct ports (recall that the GBTx is operated in bidirectional mode. When looking at the OH from the top (while it is connected to the GEB), GBT0 is on the left, and GBT1 is on the right (see figure 2.2). The VTTX closest to the GBT0 VTTX is responsible for the GEM trigger, while the other VTTX is responsible for the cathode strip chamber (CSC) trigger.

[1] This is known as have the GBT not "locked." One can manually check this in the `docker` container by running the `gem_reg.py` script, or, more easily, one can look at the monitored current on the LV power supply: if the current is less than 1.2 A, the first (or second) GBT is not locked. A common reason for this is that the fibers are dirty.

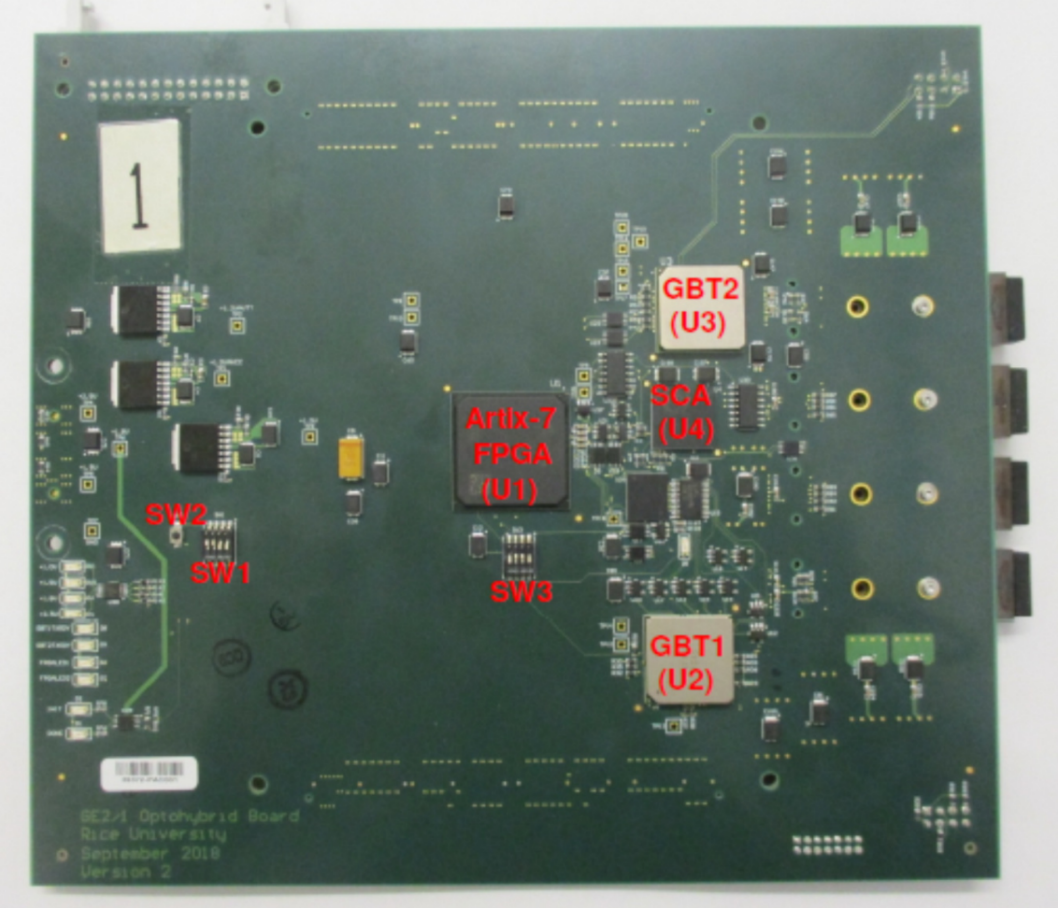

Figure 2.2: Picture of the OH with GBTs labeled [4].

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 receiver that is not illuminated; see figures 2.3 and 2.4.

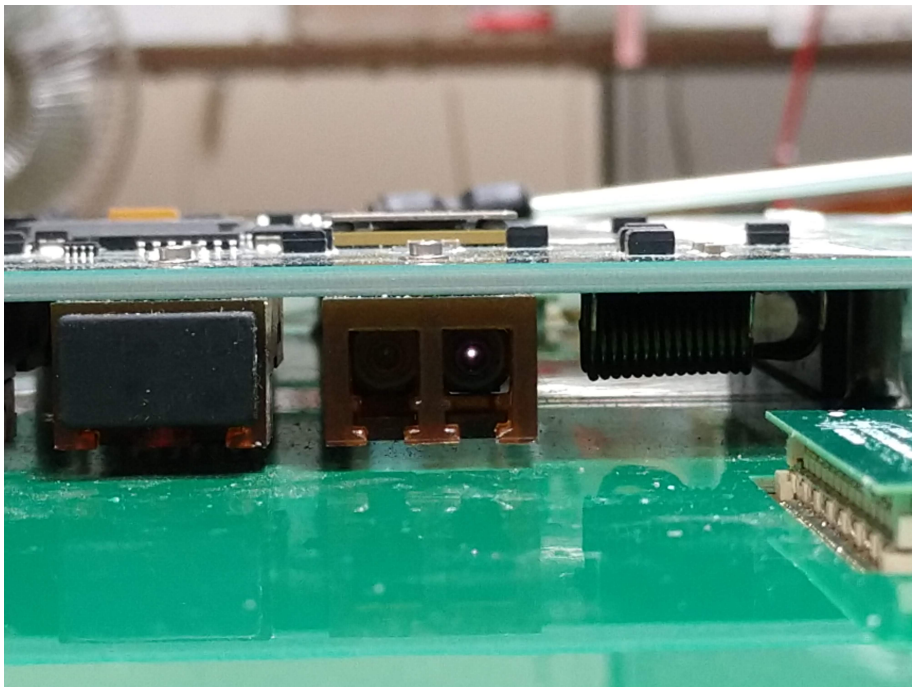

Figure 2.3: Illuminated receiver on the OH.

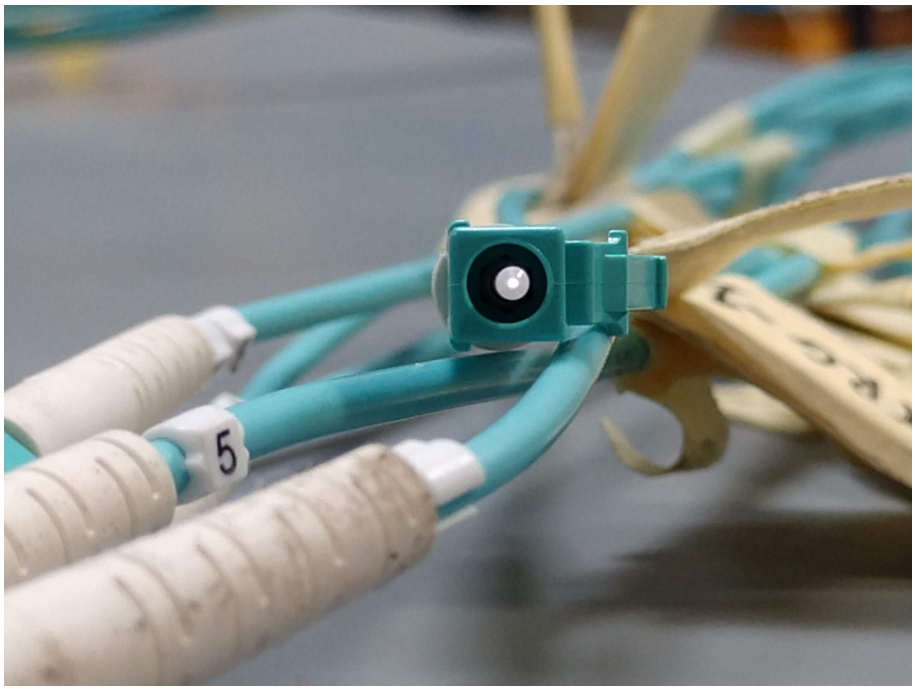

Figure 2.4: Illuminated optical fiber which is to be connected to the receiver port which is not illuminated.

2.2 The VFAT3 ASIC

Of main importance to electronics integration is the high flexibility of this chip; it can be programmed to ensure the uniform response of all 128 channels, which can be accomplished through the internal

calibration system present on the chip [7]. The primary goal of electronics integration is, of course, to establish and optimize communication with these VFAT chips on the GEB, so this chip is one of the central aspects of this manual.

The VFAT interfaces with the GEB board and the readout board of the detector through a Hirose/Panasonic connector converter card[2]. It has 128 channels, all of which can be calibrated individually.

2.2.1 Registers

Below in Tab. 2.1, I list some important registers, and then briefly discuss their purpose and function. The CFD is used to provide timing information that is independent of the pulse amplitude. Essentially,

Table 2.1: Voltage Tolerance of the FEASTs

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

the zero-crossing point of the bipolar pulse is detected by subtracting the part of the unipolar input signal with a time delayed copy of this signal. The input signal is sent to both a passive circuit which converts it into a bipolar signal with amplitude-independent zero crossing time [7] and also the arming comparator. The arming comparator responds only when the signal greater than the threshold set here; when the signal is above the threshold, the comparator enables CFD output. This signal is then sent to an amplifier which restores amplitude of the pulse which was attenuated due to the shaping network. Dynamic offset compensation (DOC) is also applied. This pulse is then sent to the ZCC, which outputs a digital signal when the differential input crosses the baseline. In normal operation, the signal is then sent from the CFD to a multiplexer, which outputs the signal. (Note that the multiplexer can bypass the CFD's output and pass the signal from the arming comparator, which is the "true" signal without time-walk correction.) See Fig. 2.5 below for the block diagram.

The arming comparator exhibits hysteresis, and so this DAC register can be programmed to control the hysteresis loop.

[2]Currently the FlexPCB card; this will eventually be replaced by the Plug-In Card.

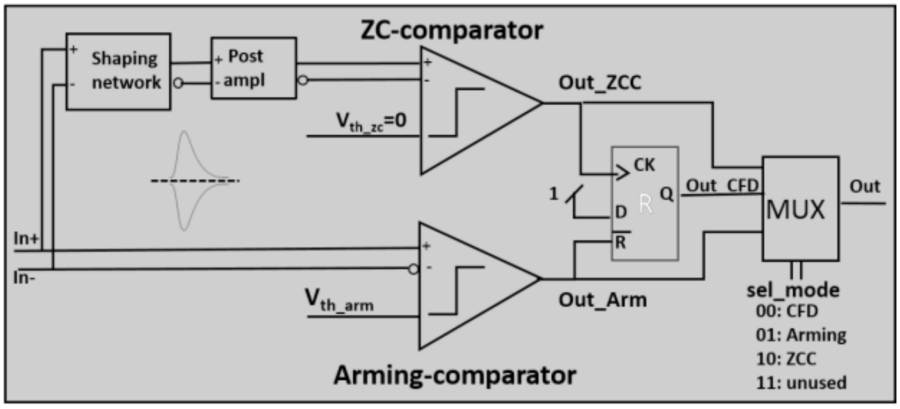

Figure 2.5: Block diagram of the CFD circuit.

2.3 The FEAST DC-DC Converter

In order to transmit the appropriate voltage to the correct components on the GEB board, the FEAST 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 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 being used to provide power to the OH. The FEAST is critical to this project because it is radiation-hard 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].

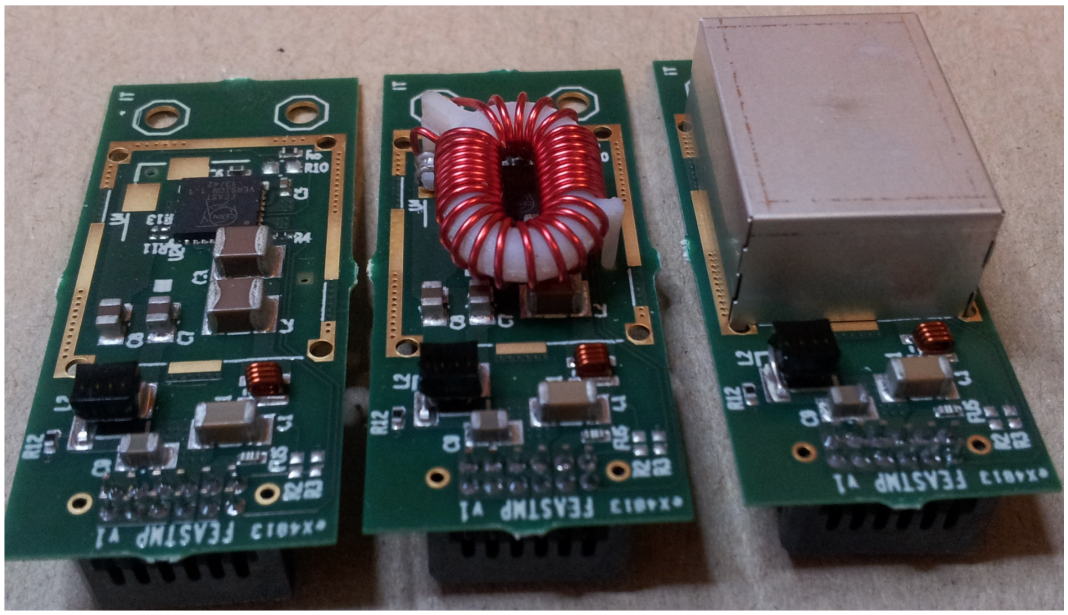

Figure 2.6: Picture of the FEAST with the shielding and it's main inductor removed [8].

2.3.1 Testing the Power Distribution of the FEASTs

It is important to test the power distribution on the GEB. This will ensure that (1) the OH is not receiving an over-voltage which can damage the chip and reduce its operational lifetime, and (2) 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 connectors). Table 2.2 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 2.2: 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 to the LV supply. 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. 2.7). Record the voltages.

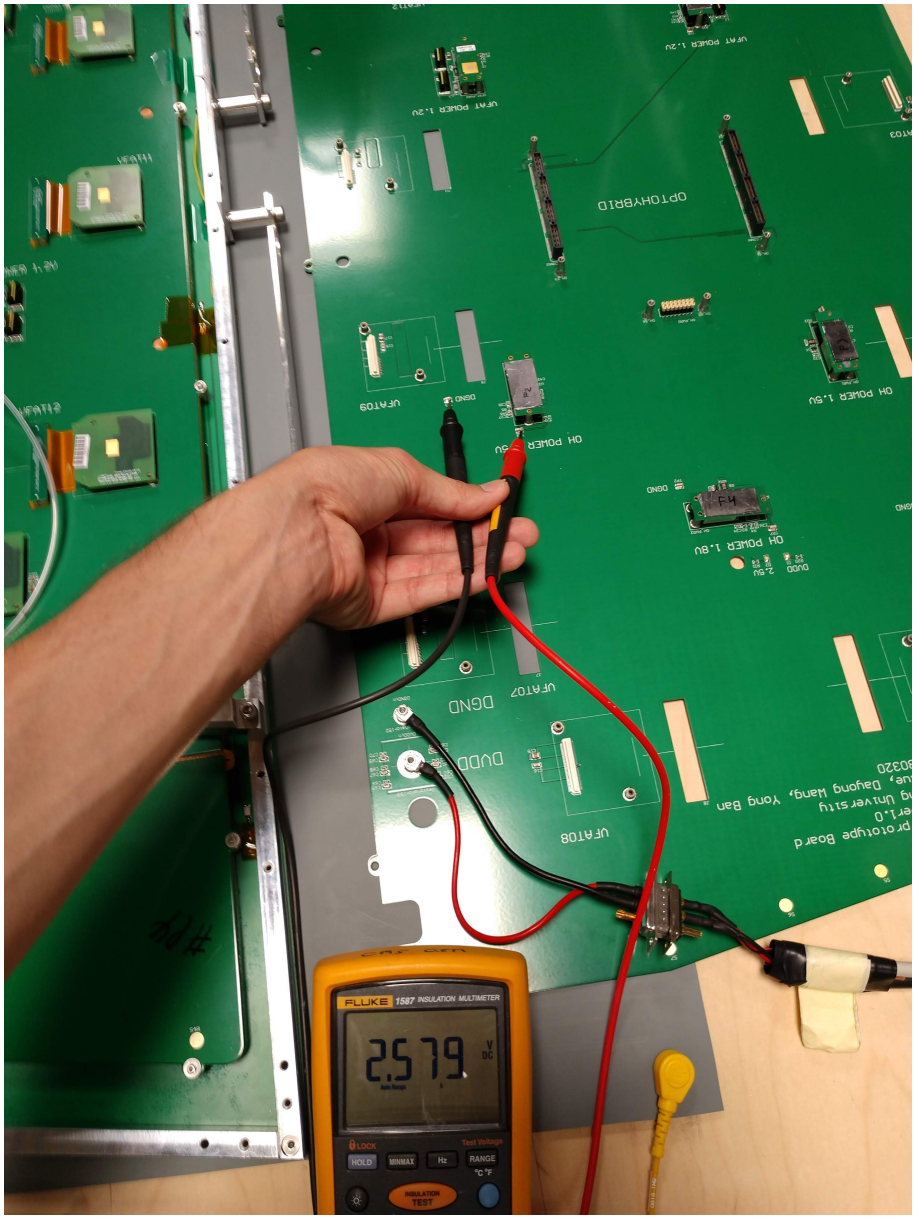

Figure 2.7: Picture of the measurement of the voltages produced by the FEASTs on the GEB.

2.4 The GLIB Card

The gigabit link interface board (GLIB) is an advanced mezzanine card (AMC) that serves as the interface between the frontend electronics (FEs) and the DAQ PC. The GLIB has four small form-factor pluggable

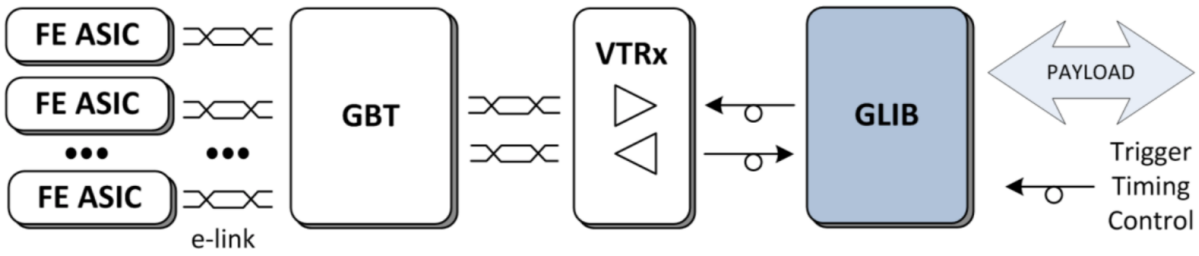

Figure 2.8: Schematic of the link system with the GLIB [9].

(SFP) sockets built standard on the card. There is an option to expand the number of SFPs by adding up to two FPGA Mezzanine Cards (FMCs), which have four sockets a piece. The GLIB at FIT uses Avago AFBR-57R5APZ SFP transceivers [10] and uses LC to LC OM3 10 Gigabit Multi-Mode Duplex 50/125 LSZH fiber optic cables to for communication between the OH and the GLIB. The current firmware (2019/11/28) maps the fibers for the first link to the top two SPFs on the GLIB, and maps the fibers for the second link to the bottom two SFPs[3].

[3]Note that if an FMC is used, the mapping is reverse, i.e., the two bottommost SFPs are for the first link and the top two SFPs are for the second link.

Chapter 3

Electronics Integration Procedure

This chapter provides an overview of the various tests we run for the GE2/1 GEBs (as of 2019/11/28, only the M5-P1 and M1-P3 GEBs are at FIT). An overview of the low voltage (LV) control, connectivity testing, `ARM_DAC` calibration routine, and the S-bit mapping and rate scan will be given.

Generally, multiple panes are kept running in a `tmux` (Terminal Multiplexer) session for convenience[1]. If connecting via `ssh`, attach to the `tmux` session by executing

```
$ tmux attach -t 0
```

To move between windows, first enter `ctrl` + `b` simultaneously, then hit `n` for next pane, or `p` for the previous pane. To open a new pane, enter `ctrl` + `b` and `c`. To exit from the session (without killing the process), enter `ctrl` + `b` and `d` (for detach). To kill a pane, enter `ctrl` + `b` and `x`.

3.1 Low Voltage Control

Low voltage and HV are provided by a CAEN SY5527 mainframe [11][2]. The LV board is a CAEN A2519 [12] and the HV board is a CAEN A1515 [13]. One can either use the `GECO2020` software (SW) on the Windows PC or connect through `ssh` on the `GEMDAQ PC`. To connect via `ssh`, perform the following commands:

1. `$ ssh admin@192.168.0.1`

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

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 PS controls 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`.

3.2 Testing the Connectivity of the VFATs

An important script, `testConnectivity.py`, performs many vital functions. It tests the communication of the GBT and SCA, programs the FPGA and trigger links, and checks the communication and synchronization (i.e., GBT phase scans and the programming of a common clock phase) of the VFATs. This multipurpose script performs some of the tests presented in sections B.1 and B.4, and also performs preliminary S-curves. To run this script, execute the following commands:

1. First, run `./connectDB.sh`[3]. When prompted, enter your CERN credentials. This `bash` script opens a tunnel to two databases (DBs) hosted at CERN. Note that if this script is not executed before beginning the scans, you will be unable to perform them.

2. To test *just*[4] the connectivity of the OH/VFATs, navigate to the home directory (∼`/`), and run

[1] I.e., one pane is devoted to the PS, one for the CTP7 Zynq SoC emulator, one for the `docker` container, and one for running scans, for example.

[2] To download the manuals, you must create an online account with CAEN.

[3] Each user will need to edit this file with their `LXPLUS` details (i.e., `user.name@lxplus.cern.ch`).

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

Main Utility Groups Maintenance Admin

Group 00

Channel Name	V0Set	I0Set	VMon	IMon	Pw	Ch#
CHANNEL00	6.50 V	3.0 A	0.000 V	0.000 A	Off	01.0000
CHANNEL01	6.50 V	3.0 A	0.000 V	0.000 A	Off	01.0001
M5-P1	6.50 V	3.0 A	0.000 V	0.000 A	Off	01.0002
M1-P3	6.50 V	3.0 A	6.501 V	1.676 A	On	01.0003
CHANNEL04	14.00 V	3.0 A	0.000 V	0.000 A	Off	01.0004
CHANNEL05	15.00 V	3.0 A	0.000 V	0.000 A	Off	01.0005
CHANNEL06	15.00 V	3.0 A	0.000 V	0.000 A	Off	01.0006
CHANNEL07	15.00 V	3.0 A	0.000 V	0.000 A	Off	01.0007
0_G3BOT	0.00 V	0.50 uA	0.14 V	0.0750 uA	Off	03.0000
0_G3TOP	0.00 V	0.50 uA	0.10 V	0.0440 uA	Off	03.0001
0_G2BOT	0.00 V	0.50 uA	0.00 V	0.0380 uA	Off	03.0002
0_G2TOP	0.00 V	0.50 uA	0.16 V	0.1310 uA	Off	03.0003
0_G1BOT	0.00 V	0.50 uA	0.16 V	0.0680 uA	Off	03.0004
0_G1TOP	0.00 V	0.50 uA	0.16 V	0.0970 uA	Off	03.0005
0_G0BOT	0.00 V	0.50 uA	0.06 V	0.0010 uA	Off	03.0006
1_G3BOT	0.00 V	0.50 uA	0.10 V	-0.0520 uA	Off	03.0007
1_G3TOP	0.00 V	0.50 uA	0.12 V	0.0490 uA	Off	03.0008
1_G2BOT	0.00 V	0.50 uA	0.18 V	0.2350 uA	Off	03.0009
1_G2TOP	0.00 V	0.50 uA	0.08 V	0.1110 uA	Off	03.0010

Figure 3.1: The PS control window.

```
$ testConnectivity.py <flags> --skipDACScan --skipScurve --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 and the command is executed from the `home` directory, ∼/):

```
$ testConnectivity.py -a --gemType ge21 --detType m5 --skipDACScan --skipScurve 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 Scurves.

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
```

3.2.1 Retrieving DAC Values

After establishing connectivity with the front-end electornics, the DAC values determined from the VFAT quality control tests need to be retrieved from the database. The procedure below enumerates this process:

1. Run `connectDB.sh`[5].

2. Now, execute

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

An example is give for M1-P3:

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

Typical output of this script looks like

[5]This script is located in `/usr/local/bin` and can be executed from anywhere on the machine.

```
[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
```

3.3 Performing a DAC Scan

DAC register values are needed for proper calibration and operation of the VFATs. These registers are discussed in §2.2.1.

1. First, run `./connectDB.sh`.

2. To perform a DAC scan, it is important that connectivity is established (this is mandatory if there has been a power cycle of the OH). Execute the following command:

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

   For example, the command will look like the example given here is for M1-P3 and the command is executed from the `home` directory, ∼/):

   ```
   $ testConnectivity.py -a --gemType ge21 --detType m1--skipScurve 1 3 0x2 2>&1 | tee logs/
       testConnectivity_M1-P3_OH10_20191219.tx
   ```

3. Once the DAC scan has been completed, the `home` directory of the `docker` container will be populated with the following configuration files:

   ```
   NominalValues-CFG_IREF.txt
   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
```

The `testConnectivity.py` routine automatically updates these values automatically. (Except, as of 2019/12/19, the IREF register values. To perform this, see section §B.2.)

3.4 Taking Scurves

This procedure provides us with the equivalent noise charge (ENC) in the electronics/detector. The ENC of the system is the noise present in the detector/pre-amplifier/amplifier chain. To take an Scurve, first the `THR_ARM_DAC` value is set (essentially the threshold). 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 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, 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. 3.4).

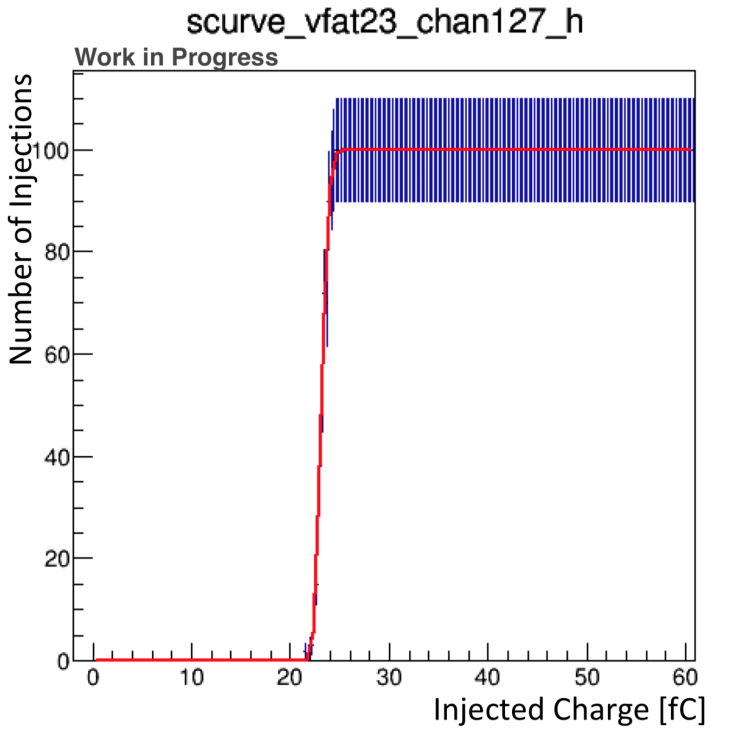

Figure 3.2: Scurve for a single channel on one VFAT [15].

The data for each channel are fit with a modified error function. The resulting sigma of this fit (essentially quantifying the "width" of this curve), gives us an estimate of the noise. Output plots are generated, the most important of which will be described in depth below.

In Fig. 3.3, we can see the Scurve sigma distribution for all VFATs at a partiular `THR_ARM_DAC` level. In

Fig. 3.4, we can see a 2D histogram of the Scurve for all VFATs. Note that if this plot was viewed in 3D, this would simply be a three dimensional sigmoid function.

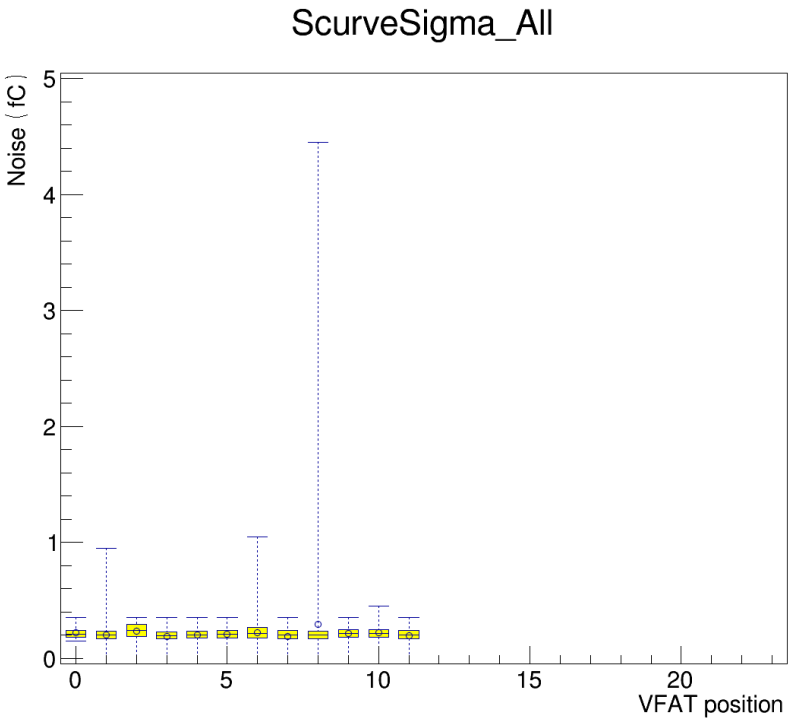

Figure 3.3: Scurve sigma distribution summary.

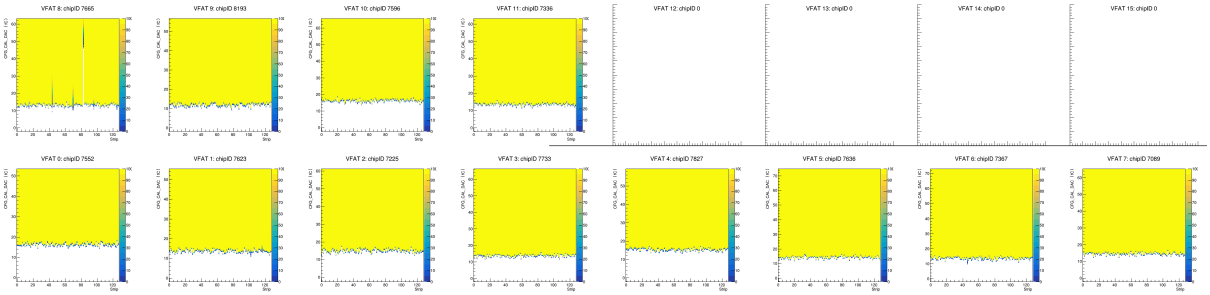

Figure 3.4: Two-dimensional histograms of the Scurves for all VFATs on the M5-P1 GEB.

3.5 Taking Scurves at Multiple THR_ARM_DAC

To take and analyze multiple Scurves at different THR_ARM_DAC values, we use a bash script, modified for the GE2/1 detector, to run this procedure automatically. The command is as follows

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>
   ```

   So, a full example would look 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}/Scurve/{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}/SCurve/{YYYY.MM.DD.hh.mm}/SCurveData/Summary`

For post analysis, the `ROOT` macro `plotScurve_utils.cpp`, located at `https://github.com/sbutalla/scurvePlottingUtilities.git`, can be used to plot the sigma distributions vs. the `THR_ARM_DAC` value for all VFATs. After these plots are generated, one can use `ImageMagick`, a command line tool, to combine all plots into one plot:

```
convert \( VFAT0_sigmaDistribution.png VFAT1_sigmaDistribution.png VFAT2_sigmaDistribution.png
    VFAT3_sigmaDistribution.png +append \) \
       \( VFAT4_sigmaDistribution.png VFAT5_sigmaDistribution.png VFAT6_sigmaDistribution.png
           VFAT7_sigmaDistribution.png +append \) \
       \( VFAT8_sigmaDistribution.png VFAT9_sigmaDistribution.png VFAT10_sigmaDistribution.png
           VFAT11_sigmaDistribution.png +append \) \
       -background none -append VFAT_sCurveSigmaDist.png
```

3.6 Taking a Latency Scan

Currently, we do not have an AMC13, so we cannot perform this scan at the FIT test stand.

3.7 Taking an Sbit Mapping and Rate Measurement

Currently, we do not have an AMC13, so we cannot perform this scan at the FIT test stand.

Bibliography

[1] J. Sturdy et al., GitHub Repository `cms-gem-daq-project/vfatqc-python-scripts`, 2019, `https://github.com/cms-gem-daq-project/vfatqc-python-scripts`.

[2] J. Sturdy et al., GitHub Repository `cms-gem-daq-project/cmsgemos`, 2019, `https://github.com/cms-gem-daq-project/cmsgemos/graphs/contributors`.

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

[5] 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.

[6] P. Moreira et. al, *GBTx Manual*, 2018, `https://espace.cern.ch/GBT-Project/GBTX/Manuals/gbtxManual.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] "FEASTMP: Radiation and magnetic field tolerant 10W DC/DC converter module," `https://project-dcdc.web.cern.ch/project-dcdc/public/Documents/FEASTMod_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] CAEN SY5527 Mainframe Manual, 2019, `https://www.caen.it/products/sy5527/`.

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

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

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

[15] S. Butalla & M. Hohlmann, "Calculation and Measurement of the Interstrip Capacitance and Its Correlation With Measured ENC for the CMS GE2/1 GEM Detector," 2019 April APS Meeting, Denver, CO, `https://research.fit.edu/media/site-specific/researchfitedu/hep/heplaba/documents/conferences-and-workshops/national-amp-international/APS_ButallaHohlmann_FinalDraft.pdf`.

[16] A. Svetek et al., "The Calorimeter Trigger Processor Card: the next generation of high speed algorithmic data processing at CMS," *JINST*, **11** C02011, `doi:10.1088/1748-0221/11/02/C02011`.

[17] M. Dalchenko, E. Juska, R. King, A. Peck, J. Sturdy, "Detailed documentation: Executable scripts," 2018, `http://test-gemdoc.web.cern.ch/test-gemdoc/docs/reg_utils/latest/reg_interface/reg_interface.html#module-reg_utils.scripts.reg`.

[18] "Collisions," *LHC Machine Outreach Website*, `https://lhc-machine-outreach.web.cern.ch/lhc-machine-outreach/collisions.htm`.

[19] B. Stone, "GE21 User Manual," 2019.

[20] 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`.

Appendices

Appendix A

System Configuration

This chapter includes important information on many topics. For instance, properly configuring the `system_configuration_settings.yml` file, properly updating the address tables for the OH and GLIB

A.1 Setting the System Configuration 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",
    (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
```

A.2 Updating the Address Tables

The address tables must be updated when a new version of GLIB or OH firmware (fw) are used. We will first cover the OH address tables and then review the GLIB address tables. For simplicity, we use symbolic links to point to the address tables so we don't have to hardcode the file names each time in the SW. While inside of the `docker` container, move the new address tables (one for the OH and one for the GLIB AMC) to `/mnt/persistent/gemdaq/xml/`. Then, create the symbolic links:

```
$ 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
```

A.3 The Current Environmental Variable Paths

Reproduced below are the environmental variables set in the `bash_profile` on the `GEMDAQ` PC:

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

Appendix B

Individual Connectivity Tests and Scans

B.1 Manually Establishing Communication with the VFATs

Manually checking the registers on the Zynq SoC emulator allows you to quickly spot any issues with the VFATs or OH. We can check the registers on the CTP7 Zynq SoC emulator running in the `docker` container. With the `docker` container for the emulator and the control hub containers started, we can use the following procedures to connect to the container, launch the command line tool `gem_reg.py`, and read and write to the registers.

B.1.1 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 >`.

B.1.2 Useful Commands

Outlined below are useful commands (adapted from [20]):

- `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
  ```

B.1.3 Reading and Writing to the Registers

- To issue a link reset to the GBTs. This command and normal output looks like:

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

- To check if there are any synchronization errors for the VFATs, execute:

```
gem-shelf01-amc03 > rwc *OH0*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:

```
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[1]. 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, we need to check if the VFATs are simply out of phase or if there is a real communication problem.

- 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 [19]. If not, the register will read `0xdeaddead`.

[1] For more information on the executable scripts on the CTP7 register interface, see [17]

B.2 Manually Setting IREF Values

Before configuring the chamber and beginning the various scans, one needs to update the IREF registers for the VFATs. IREF is the reference current generated by the digital-to-analog (DAC) converters after the bandgap circuit [7]. From the quality control tests of the VFATs after production, each VFAT is characterized by a unique IREF value[2]. This can be obtained from the database (DB), by either querying the DB with a script or viewing the tables on the CMS OMS: https://cmsomsdet.cern.ch/gem/vfat_3/new_page. To find the correct IREF values, you will need to first convert the barcode number on the VFAT from decimal to hexadecimal (i.e., VFAT 4085 = 0xff5 in hex), search for the VFAT, and then update the value. To manually[3] update the IREF values, perform the following actions:

1. In the docker container, change directories to
   /mnt/persistent/gemdaq/vfat3
2. IREF values can be changed manually in the config_OHX_VFATY.txt files, or you can execute the following command from the /mnt/persistent/gemdaq/scripts directory:
   ```
   $ ./replace_parameter.sh [-f <FILENAME>] <REGISTER> <LINK> <VALUE>
   ```
   Note that one can either write a single value to the configuration file or one can provide a file with register values for all VFATs For this example, we are replacingIREF values for all of the VFATs connected to the OH on link 1:
   ```
   $ ./replace_parameter.sh -f /mnt/persistent/gemuser/NominalValues-CFG_IREF.txt CFG_IREF 1
   ```

B.3 Taking a Latency Scan

Currently, we do not have an AMC13, so we cannot perform this scan at the FIT test stand.

A latency scan will tell us the response of the VFATs to a trigger signal. Because it takes a certain amount of time for the signal to reach the frontend electronics, we need to understand the response of the VFATs. Essentially, when the trigger signal is detected, we want to look a certain amount of time into the "past" (i.e., into a specific register in the VFAT's memory buffer) when that trigger signal occurred. The purpose of the latency scan, therefore, is to observe the response of the VFATs when a simulated trigger signal is sent.

Normally, we use the units of BX, or bunch crossings[4]. We want the VFATs to respond to the trigger signal with a value of 33 bunch crossings, meaning that the VFAT sends the 33rd register in its memory buffer when it is triggered. To take a latency scan, we follow the following procedure:

1. Configure the chamber:
   ```
   $ confChamber.py -s 8 --shelf 2 -g 1 --vfatmask=0xfff000
   ```
2. Run the scan using
   ```
   $ run_scans.py lat -i -m 3 --vfatmask 0xfff000 2 8 2
   ```
   Here, the -i flag is the internal calibration mode, the -m flag sets the CFG_PULSE_STRETCH register[5], and the arguments after the flags correspond to run_scans.py --flags [Shelf] [Slot] [OHmask].
3. Again, the data will be saved to a directory with the time stamp of the time the scan is initiated.
4. To analyze the data, we run the analysis script:
   ```
   $ anaUltraLatency.py -i /data/bigdisk/GEM-Data-Taking/GE21_Integration/GE21-M2/
       latency/trk//time.stamp.of.scan/LatencyScanData.root
   ```
5. After this is complete, scp the plots (Summary.png, LatSumOverAllVFATs.png, and MaxHitsPerLatByVFAT.png) to your machine.

[2] For more information of how to determine the IREF value, see [7, p. 42]
[3] One can use the bash script replace_parameter.sh, but for changing only one or a few values it is quicker to just edit the file. See [19] for more information.
[4] The LHC operates with a peak bunch crossing rate of 40 MHz, so there is a bunch crossing every 25 ns [18].
[5] The "m" denotes monostable pulse length/width in BX units.

B.4 Taking a DAC Scan

Outlined here is the more detailed version of the ARM DAC calibration given in section 3.5. Here, each step is performed separately and is, of course, not automated.

1. First, power up the GEB using the LV controls.
2. Configure the chamber with the proper `IREF` values:

```
$ confChamber.py -s 3 --shelf 1 -g 1 --vfatmask=0xfff000
2019.06.07.10.56
Open pickled address table if available  /opt/cmsgemos/etc/maps//
    amc_address_table_top.pickle...
Initializing AMC gem-shelf02-amc08
opened connection
Configuring VFATs on (shelf1, slot3, OH1) with chamber_vfatDACSettings dictionary
    values
biased VFATs on (shelf1, slot3, OH1)
Set CFG_THR_ARM_DAC to 100
Chamber Configured
```

3. To start the DAC scan, run

```
$ run_scans.py dacScanV3 1 3 0x2
```

   The DAC scan will produce a `.root` file with all of the data, usually located at `/data/data/GEBNAME/dacScanV3/time.stamp.of.scan/`, where the time stamp will be in `year.month.day.hour.minute` form (e.g., `2019.06.07.11.01`).

4. To run the analysis on the DAC scan, run the following command:

```
$ anaDACScan.py --calFileList /path/to/calFileList/GE21-M1-P4/calFileList_ADC0_GE21
    .txt -p /data/data/GEBNAME/dacScanV3/time.stamp.of.scan/dacScanV3.root
```

B.5 Taking S-Curves

An S-Curve, named after the shape of the modified error function that is fit to the data, provides a measure of the noise in the system. During the process of an S-curve, a calibration pulse of fixed charge is injected into each of the 128 channels of the VFAT (this charge is increased after all channels are scanned, and then the channels are scanned again for this calibration pulse). A voltage comparator, set to respond to a signal of a given voltage, records whether the signal is above or below the threshold. The channels are scanned in parallel for each of the twelve VFATs on the GEB. In an ideal system, we would expect the curve to be a step function; for a signal below the voltage value of the comparator, there will be no response. Once the threshold is crossed, the comparator will respond to the calpulse. Because there is inherent noise in the system, the curve will not be an exact step function, but will instead have a sort-of "transition" region, where the the curve assumes an "S" shape instead of a step function. The sigma of this fit is a direct measure of the noise.

The procedure here will be bifurcated to give an overview of the procedure for taking just one s-curve, and taking multiple s-curves (the standard procedure).

B.5.1 Taking a Single S-curve

To take a single s-curve, we simply run the `run_scans.py` with the `scurve` argument:

```
$ run_scans.py scurve -m 3 --vfatmask=0xfff000 2 8 2
```

Similarly, we analyze using the `anaUltraScurve.py` script, using the correct directory (time-stamp):

```
$ anaUltraScurve.py scurve -i /data/bigdisk/GEM-Data-Taking/GE21_Integration/GE21-M2/
    scurve/time.stamp.of.scan/SCurveData.root
```