Travelling modules & BDAQ53

Instructions for testing an ATLAS ITk travelling module with BDAQ53

Michael Daas

University of Bonn
Physikalisches Institut
Nussallee 12
53115 Bonn

Mail: daas@physik.uni-bonn.de

Setting up BDAQ53

Hardware

To power the BDAQ board, you can use either USB or a dedicated 5V powersupply. Select the powering method accordingly with the jumper next to the VDC input on the board. When using USB, also a USB 2.0 cable is fine, but the USB powersupply or port has to provide at least 1A of current.

Connect the BDAQ53 base board to the readout PC using a standard Ethernet cable (>= CAT6). For the best experience, use an additional ethernet interface on your PC (either PCIe or USB3, though it has to be 1000BASE-T – Gigabit ethernet) and the "ETH" port on the BDAQ board next to the USB3 port.

Software

Make sure you have a valid and clean Python 3.7 environment. The recommended method is to install [conda](https://conda.io/miniconda.html).

Install dependencies using conda:

```
# conda install numpy bitarray pyyaml scipy numba pytables matplotlib tqdm pyzmq blosc psutil coloredlogs
```

Go to a directory of your choice and clone the BDAQ53 repository using git:

```
# git clone https://gitlab.cern.ch/silab/bdaq53.git
# cd bdaq53
# git checkout travelling_module
# python setup.py develop
```

Flashing the correct firmware

First, make sure Xilinx Vivado is installed on your system. Connect the BDAQ board to the PC via USB (USB 2.0 is enough).

The firmware can be downloaded and flashed using a simple command:

```
# bdaq53 --firmware travelling_module_BDAQ53_1LANE_RX640
```

Alternatively, you can download the firmware manually from [here](https://gitlab.cern.ch/silab/bdaq53/wikis/uploads/e27b7af2ca9c12d6072628e8ddec592c/v0.13.0_travelling_module_BDAQ53_1LANE_RX640.tar.gz) and use Xilinx Vivado to flash (the recommended method is to flash the FPGA persistently using the .mcs file) to the FPGA using the USB interface of the BDAQ base board. More instructions for manual flashing can be found [here](https://gitlab.cern.ch/silab/bdaq53/wikis/Hardware/FPGA-configuration#permanent-configuration)

Testing the connection

Configure the network interface on your computer, that the BDAQ board is connected to as follows:

> *IP Address: 192.168.10.10*
> *Subnet Mask: 255.255.255.0*
> *Gateway: 0.0.0.0*

Make sure, no jumper is set on the "PMOD" header on the BDAQ board. See if the communication between the PC and the board works by running

> *ping 192.168.10.12*

Preparing the measurements

Connect the DP1 connector on the single chip card (SCC) to the "DP_ML 1" connector on the BDAQ board using a standard DisplayPort (DP) cable.

Make sure the SCC is jumpered for the correct powering mode and your powersupply is setup accordingly.

Observe the BDAQ board. On the FPGA daughterboard there should be 4 LEDs labeled 0-3. LEDs 0 and 1 should be lit up, while LED 3 should be off and LED 4 should be flashing. Once you turn on the powersupply of the SCC, LED4 should stop flashing and either turn off permanently or all four LEDs should be constantly on.
If the flashing continues after you turn on the module, don't panic just yet. Some chips only lock once you actually try to communicate with them. Go on with the next step.

Trimming your chip

In order to have your chip working at optimal conditions, you have to trim IREF, VREF_A and VREF_D, as well as VREF_ADC. IREF is the reference current for all DACs on the chip. VREF_A/D are the regulator reference voltages, which determine the analog and digital supply voltage of the chip (VDDA and VDDD) that will be generated by the regulators in (shunt-)LDO mode. VREF_ADC is the reference voltage for the internal ADC circuit as well as the charge injection circuit. If this reference is not trimmed correctly, the chip charge calibration and therefore the electron scale in any plot will be wrong! You can acquire these settings by either using the **calibrate_vref.py** and **calibrate_vref_adc.py** routines of BDAQ53 or manual trimming (see [wiki](https://gitlab.cern.ch/silab/bdaq53/wikis/User-guide/Trimming-VREF)).

Preparing the module configuration file

Navigate to **bdaq53/bdaq53/** and create a copy of **rd53a_default.cfg.yaml,** naming it according to the chip you are testing. For example, if you have travelling module #1 with chip serial number 0x0494, name your copy **0x0494.cfg.yaml.** Open your freshly created copy and enter the optimal trimbit settings you acquired in the last step in the **trim** section of the file.

Make sure that the global threshold DACs are set to a safe threshold of >3000e: e.g. VTH_SYNC: 390, Vthreshold_LIN: 415, VTH1_DIFF: 600.

The file will be automatically picked up if its name matches the chip serial number you enter into **testbench.yaml** in the next step.

Preparing the output directory

Create a directory on your PC that will be used for all BDAQ53 output files. Open **bdaq53/bdaq53/testbench.yaml** with any text editor and in the **general** section, set

- **chip_sn** to the serial number of the chip / module you are going to test,
- **output_directory** to the directory you just created.
- Make sure, **chip_configuration** is set to **'auto'.**

Copy the configuration file you downloaded or created in the previous step to the output directory. This way, the configuration file is used for the very first scan. After that, the configuration is passed through consecutive scans.

Testing the setup

To make sure everything works as expected, test the setup by changing to **bdaq53/bdaq53/scans** and opening **scan_digital.py.** Verify that the region of interest is defined as

```
'start_column': 0,
'stop_column': 400,
'start_row': 0,
'stop_row': 192,
```

This means that the whole pixel matrix will be scanned. Start the scan by running

```
python scan_digital.py
```

Check the output: In the very beginning it should tell you which chip configuration file it is using. Verify that this is the file you downloaded or created in the previous section. Let the scan finish and open the output .pdf file. Double check that the correct chip serial number is displayed in the first sentence on the first page. Scroll to the second page. The **Event status** histogram should be empty. Scroll to the third page. The **Occupancy** map should be homogeneously yellow, showing an occupancy of 100 hits for every pixel. The total amount of hits should be Σ = 7680000, or 76800 pixels with 100 hits each.

Congratulations, you are ready to begin testing the module.

Testing the module

Pre-tuning scans

Navigate to **bdaq53/bdaq53/scans** and open **scan_analog.py.** Make sure that the region of interest is defined as

```
'start_column': 0,
'stop_column': 400,
'start_row': 0,
'stop_row': 192,
```

so the whole matrix is scanned. To inject a target charge of 10000e, check that *VCAL_MED* is set to 500 and *VCAL_HIGH* is set to 1450. The difference between these two values determines the injected charge in Delta VCAL. 950 Delta VCAL corresponds to about 10ke. Save the file and start the scan by running

```
python scan_analog.py
```

Check the output file in your defined output directory. The occupancy map should look rather homogeneous with a possible exception of the DIFFERENTIAL frontend. Some inefficiencies are expected here due to the chosen settings. Next, check the ToT histogram. The peak of the distribution is probably not at the desired value of 8. We will take care of this later.

Tuning the synchronous front-end

The SYNC flavor is the easiest flavor to tune, since it doesn’t require manual tuning of the local pixel thresholds.

TLDR

The scan order to tune the SYNC flavor is

1. *tune_global_threshold.py*
2. *tune_tot.py*
3. *tune_global_threshold.py*
4. *scan_threshold.py*
5. *scan_analog.py*

Global threshold tuning

Open **tune_global_threshold.py** and set the region of interest to

'start_column': 0,
'stop_column': 128,
'start_row': 0,
'stop_row': 192,

to only scan the SYNC flavor. The target threshold should be set up as follows:

'VCAL_MED’: 500,
'VCAL_HIGH': 580,

A target threshold of 80 Delta VCAL corresponds roughly to 1000e. Save the file and start the scan by running

python tune_global_threshold.py

The scan will log the optimal setting for *VTH_SYNC***,** the global threshold DAC of the SYNC flavor. You can note this value for your documentation, but it will be picked up by the next scan automatically.

Threshold scan

Open **scan_threshold.py**, set the region of interest to

'start_column': 0,
'stop_column': 128,
'start_row': 0,
'stop_row': 192,

to only scan the SYNC flavor and

'VCAL_MED': 500,
'VCAL_HIGH_start': 500,

```
'VCAL_HIGH_stop': 700,
'VCAL_HIGH_step': 5,
```

to scan charges between 0 and 200 Delta VCAL (~ 200 to 2200e). Save the file and start the scan by running

```
python scan_threshold.py
```

The threshold scan will take up to 2 minutes. In the output file, check the “Threshold distribution for enabled pixels”: It shows the mean threshold as μ and the threshold dispersion as σ. Your mean threshold should now be around 1000e with a reasonable dispersion well below 100e. Also check the mean noise in the “Noise distribution for enabled pixels” plot. The noise should also be well below 100e.

Tune ToT

In order to get the desired ToT response (ToT value of 8 for a charge of 10ke), open **tune_tot.py** and again, set the region of interest to the SYNC flavor as above. the other settings should be fine by default. *target_tot* is the desired ToT value, so 8, while *VCAL_MED: 500* and *VCAL_HIGH: 1450* define the target charge for the desired value. A Delta VCAL of 950 again corresponds to a charge of about 10ke. Save the file and start the tuning by running

```
python tune_tot.py
```

You can note down the logged optimal value for *IBIAS_KRUM_SYNC* and again, it will be picked up automatically by the next scan.

Scan threshold

Run another threshold scan like above. You will see that the ToT tuning shifted the measured threshold up by about 100e.

Tune global threshold (again)

To mitigate the effect of the ToT tuning, run *tune_global_threshold.py* again and note the new, lower value for *VTH_SYNC.*

Scan threshold

Another threshold scan will show that the threshold is now back to its target value of about 1000e.

Scan analog

To confirm the effect of the ToT tuning, run an analog scan. Remember to set the region of interest in *scan_analog.py* to only the SYNC flavor. The peak of the ToT distribution should now be at 8 when injecting a charge of 10ke.

Tuning the linear front-end

Due to the necessary manual tuning of the pixel threshold settings (TDACs), tuning the linear (and differential) flavor consists of a few more steps than tuning the synchronous flavor. Before you run a scan on a new flavor, always remember to set the region of interest accordingly. The correct region of interest for the LIN flavor is

> *'start_column': 128,*
> *'stop_column': 264,*
> *'start_row': 0,*
> *'stop_row': 192.*

All other scan settings like target thresholds or -charges are the same for all front-ends.

TLDR

The scan order to tune the LIN flavor is

1. *tune_global_threshold.py*
2. *meta_tune_local_threshold.py*
3. *tune_tot.py*
4. *tune_global_threshold.py*
5. *meta_tune_local_threshold.py*
6. *scan_analog.py*

Tune global threshold

Set the region of interest to the LIN flavor and the target charge to 80 Delta VCAL. Note down the optimal value for *Vthreshold_LIN*, the global threshold DAC of the LIN flavor.

Scan threshold

A threshold scan of the linear flavor should show a mean threshold around the desired 1000e. Notice the very broad dispersion of about 350e. This problem is tackled with the next step.

Tune local thresholds

In order to tune the local pixel threshold settings or TDACs, use *meta_tune_local_threshold.py*, a script that uses several standard threshold scans to iteratively adjust the TDACs of all pixels within the defined region of interest. Make sure the region of interest is set to the LIN flavor and run the script:

```
python meta_tune_local_threshold.py
```

This script may run up to 20 minutes. After it finishes you should see a file with a *.masks.h5* ending in your output directory. This file contains the optimal TDAC settings for all pixels.

Look at the output file of the last threshold scan, that was performed by the script. The mean threshold might have shifted up slightly, but the dispersion should now be much smaller, in the order of 50e.

Tune ToT
Run the ToT tuning for the LIN flavor by adjusting the region of interest. Leave the other values as before. Note the optimal setting for *KRUM_CURR_LIN* from the log output.

Scan threshold
Run another threshold scan and look at the output. The mean threshold is probably still shifted towards a little higher that target value. The dispersion might have also increased a bit with regard to the value before ToT tuning. this is expected since the *KRUM_CURR_LIN* DAC, which was changed by the ToT tuning also has a slight influence on the measured threshold. To mitigate this, run another iteration of tunings

Tune global threshold (again)
Run *tune_global_threshold.py* once again and note the slightly lower optimal value for *Vthreshold_LIN*.

Scan threshold
Run another threshold scan. You should see the mean threshold now bat at around 1000e. The dispersion might still be a bit higher than the optimal value we achieved 4 steps ago.

Tune local thresholds (again)
Run *meta_tune_local_threshold.py* one more time. This time, it will load the TDAC mask from last time and iterate a few times in order to optimize the settings. Once the script finishes, look at the output of the last threshold scan. The mean threshold should still be close to the target value of about 1000e, while the dispersion should have gone back town to an optimal value of about 50e.

Scan analog
Adjust the region of interest for the *scan_analog.py* to the LIN flavor and run the scan. The peak of the ToT distribution should be at 8.

Tuning the differential front-end

The procedure of tuning the differential flavor is very similar to the one for the linear flavor. It might be necessary to run a second iteration of ToT tuning after the second iteration of TDAC tuning though. The region of interest for the DIFF flavor is

> *'start_column': 264,*
> *'stop_column': 400,*
> *'start_row': 0,*
> *'stop_row': 192.*

All other scan settings should remain unchanged.

Scan order

The scan order to tune the DIFF flavor is

1. *tune_global_threshold.py*
2. *meta_tune_local_threshold.py*
3. *tune_tot.py*
4. *tune_global_threshold.py*
5. *meta_tune_local_threshold.py*
6. *tune_tot.py*
7. *scan_threshold.py*
8. *scan_analog.py*

Feel free to run threshold scans between every step to see the results. Run an analog scan between steps 5 and 6 to see why the second iteration of ToT tuning is necessary. The second TDAC tuning iteration shifts the ToT response to 10ke charge to a value of 9. The slight adjustment from the second iteration of ToT tuning does not harm the threshold distribution much.

Results

After running the tuning in the order listed above, a threshold dispersion in the order of 40e at a mean threshold around the target value of 1000e should be achievable, while the ToT response to 10ke is 8.