Preparing to run your first calibration

Download the software

… compile your own version of Ph2ACF

- If you would like to set-up your computer to develop and run Ph2ACF natively then please follow the instructions in the README[1] for installing Ph2ACF on centos7

Setup on CC7 (scroll down for instructions on setting up on SLC6)

1. Check which version of gcc is installed on your CC7, it should be > 4.8 (could be the default on CC7):

```
$> gcc --version
```

2. On CC7 you also need to install boost v1.53 headers (default on this system) and pugixml as they don't ship with uHAL any more:

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

3. Install uHAL. SW tested with uHAL version up to 2.7.1

```
Follow instructions from
https://ipbus.web.cern.ch/ipbus/doc/user/html/software/install/yum.html
```

4. Install CERN ROOT

```
$> sudo yum install root
$> sudo yum install root-net-http root-net-httpsniff  root-graf3d-gl root-phy
```

5. Install CMAKE > 2.8:

```
$> sudo yum install cmake
```

- Clone the Github branch for the school
  - git clone —single-branch —branch daqSchool2021 https://:@gitlab.cern.ch:8443/cms_tk_ph2/Ph2_ACF.git
- Create the build directory and compile the code : source setup.sh; cd build; cmake ..; cd../; make -C build -j4
- Stay in the Ph2_ACF directory for the remainder of the exercise

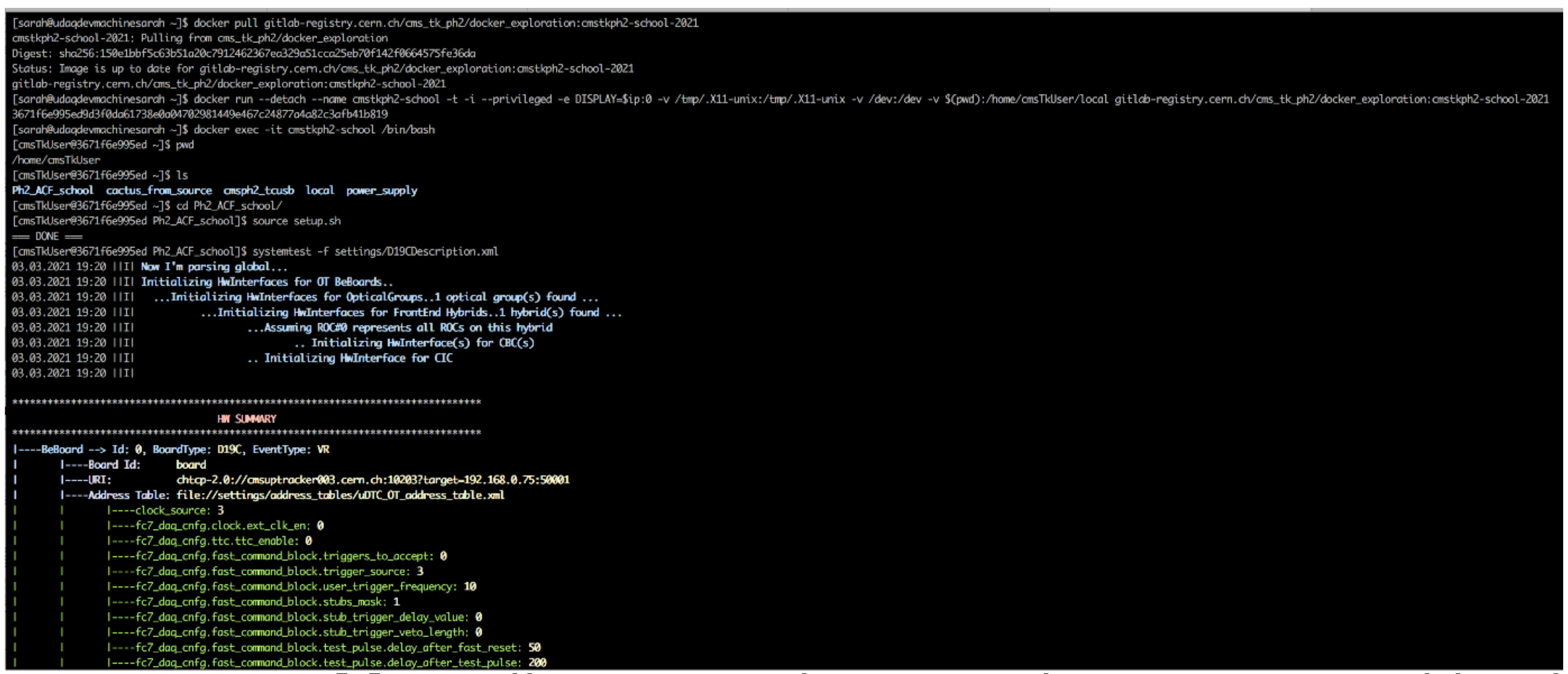
[1] https://gitlab.cern.ch/cms_tk_ph2/Ph2_ACF/

Download the software

… *run the software inside a docker container*

- If you would like to use docker to run the software then follow the instructions for installing docker and generating an access token from [2] then :
  - log-in to the gitlab registry to obtain access to the images [docker login --username $username gitlab-registry.cern.ch]
  - pull the image with the pre-compiled software branch prepared for the school [docker pull gitlab-registry.cern.ch/cms_tk_ph2/docker_exploration:cmstkph2-school-2021]
  - launch a docker container [docker run --detach --name cmstkph2-school -t -i --privileged -e DISPLAY=$ip:0 -v /tmp/.X11-unix:/tmp/.X11-unix -v /dev:/dev -v $(pwd):/home/cmsTkUser/local gitlab-registry.cern.ch/cms_tk_ph2/docker_exploration:cmstkph2-school-2021]
  - refer to README[2] for instructions on how to forward X11 to host
  - connect to the container to complete the exercise [docker exec -it cmstkph2-school /bin/bash]

systemtest run from inside docker container



[2] https://gitlab.cern.ch/cms_tk_ph2/docker_exploration/-/blob/master/README.md

Connecting the software to your front-end object

… *modify the hardware description xml*

- First, modify the connection id in the hardware description xml to point to your FC7 :
    - host-address : address of the computer physically connected to the ethernet port of the FC7
    - fc7-address : ip address of the FC7 physically connected to your front-end object

```xml
<?xml version="1.0" encoding="utf-8"?>
<HwDescription>
  <BeBoard Id="0" boardType="D19C" eventType="VR">
    <connection id="board" uri="chtcp-2.0://cmsuptracker003.cern.ch:10203?target=192.168.0.75:50001" address_table="file://settings/address_tables/uDTC_OT_address_table.xml" />
    <CDCE configure="0" clockRate="120"/>
```

*Example hardware description files for OT hardware be found in Ph2_ACF/settings/D19cDescription**

- Verify that you can communicate with the FC7 using the FPGA configuration binary provided by Ph2ACF
    - make sure that you have set-up the environment [source setup.sh]
    - list the images available on the sd-card in your FC7 [fpgaconfig -c $hardware_description_xml -l]

```
[teledaq@teledaq002:Ph2_ACF]$ fpgaconfig -c settings/D19CDescription.xml -l
03.03.2021 10:40 |I| Now I'm parsing global...
03.03.2021 10:40 |I| Initializing HwInterfaces for OT BeBoards..
03.03.2021 10:40 |I|   ...Initializing HwInterfaces for OpticalGroups..1 optical group(s) found ...
03.03.2021 10:40 |I|          ...Initializing HwInterfaces for FrontEnd Hybrids..1 hybrid(s) found ...
03.03.2021 10:40 |I|                 ...Assuming ROC#0 represents all ROCs on this hybrid
03.03.2021 10:40 |I|                        .. Initializing HwInterface(s) for CBC(s)
03.03.2021 10:40 |I|                 .. Initializing HwInterface for CIC
03.03.2021 10:40 |I| 10 firmware images on SD card:
03.03.2021 10:40 |I|  - GoldenImage.bin
03.03.2021 10:40 |I|  - d19c_8xCBC3_13032019.bin
03.03.2021 10:40 |I|  - 15042019_8CBC3.bin
03.03.2021 10:40 |I|  - d19c_8xCBC3_11062019.bin
03.03.2021 10:40 |I|  - d19c_8xCBC3_20082019.bin
03.03.2021 10:40 |I|  - d19c_8cbc3.bin
03.03.2021 10:40 |I|  - 2s_mux_cic2.bin
03.03.2021 10:40 |I|  - uDTC_2SSA_test.bin
03.03.2021 10:40 |I|  - fc7_test.bin
03.03.2021 10:40 |I|  - uDTC_8CBC3_LEDtst.bin
```

You will only see the files that have been loaded on-to our sd-card…
So expect to see only GoldemImage.bin if this is a freshly configured card!!

Connecting the software to your front-end object

… *load the correct firmware for your flavour of test system*

- Download the image for your test system from either the website https://udtc-ot-firmware.web.cern.ch or the school indico page if the above link is un-available :
    - 2cbc3_dio5 : for reading out a 2CBC3 hybrid/module
    - 8cbc3 : for reading out an 8CBC3 hybrid
    - dio5_opto_2h_cic1 : for reading out a 2S skeleton/module with CIC1
    - dio5_opto_2h_cic2 : for reading out a 2S skeleton/module with CIC2
    - cic1_2s_none : for reading out a 2S FEH prototype [with CIC1] using a 2S SEH test card v0
    - cic2_2s_none : for reading out a 2S FEH prototype [with CIC2] using a 2S SEH test card v0
    - cic1_2s_crate : for reading out a 2S FEH prototype [with CIC1] in a multi-hybrid test crate
    - cic2_2s_crate : for reading out a 2S FEH prototype [with CIC2] in a multi-hybrid test crate

- Use the FPGA configuration binary provided by Ph2ACF to load the image file (either .bin or .bit) onto the SD-card [fpgaconfig -c $hardware_description_xml -f $image_file -i $image_name]
- Use the FPGA configuration binary provided by Ph2ACF to confirm that the image file has been loaded on-to the SD-card [fpgaconfig -c $hardware_description_xml -l]
- Use the FPGA configuration binary provided by Ph2ACF to select the image file (either .bin or .bit) from the SD-card and load it onto the FPGA [fpgaconfig -c $hardware_description_xml -i $image_name]

Connecting the software to your front-end object

… *check that you have the correct objects listed in your hardware description file*

- Make sure that the hardware description file contains the correct description for your set-up :

settings/D19CDescription.xml

Example hardware description files for 8CBC3 hybrid

```xml
<OpticalGroup Id="0" FMCId="0" >

    <Hybrid Id="0" Status="1">
        <Global>
            <Settings threshold="500" latency="29"/>
            <TestPulse enable="0" polarity="0" amplitude="0xFA" c
            <ClusterStub clusterwidth="4" ptwidth="4" layerswap="
            <Misc analogmux="0b00011" pipelogic="0" stublogic="0"
            <ChannelMask disable=""/>
        </Global>

        <CBC_Files path="./settings/CbcFiles/" />
        <CBC Id="0" configfile="CBC3_default.txt" />
        <CBC Id="1" configfile="CBC3_default.txt" />
        <CBC Id="2" configfile="CBC3_default.txt" />
        <CBC Id="3" configfile="CBC3_default.txt" />
        <CBC Id="4" configfile="CBC3_default.txt" />
        <CBC Id="5" configfile="CBC3_default.txt" />
        <CBC Id="6" configfile="CBC3_default.txt" />
        <CBC Id="7" configfile="CBC3_default.txt" />
    </Hybrid>
</OpticalGroup>
```

if using a 2CBC3 hybrid/module remember to comment out chips 2-7!

settings/D19CDescription_Cic2.xml

Example hardware description files for a 2S module

```xml
<!-- RHS -->
<OpticalGroup Id="0" FMCId="0" >
  <GBT phaseTap="7" enable="1"/>
    <Hybrid Id="0" Status="1" LinkId="0" >
        <Global>
            <Settings threshold="550" latency="74"/>
            <TestPulse enable="1" polarity="0" amplitude="0x3F" channelgroup="0" delay="0" groundothers="0"/>
            <ClusterStub clusterwidth="4" ptwidth="14" layerswap="0" off1="0" off2="0" off3="0" off4="0"/>
            <Misc analogmux="0b00000" pipelogic="0" stublogic="0" or254="1" tpgclock="1" testclock="0" dll="0"/>
            <ChannelMask disable=""/>
            <CIC2 enableBend="1" enableLastLine="0" enableSparsification="0" clockFrequency="320"/>
        </Global>

        <CBC_Files path="${PH2ACF_BASE_DIR}/settings/CbcFiles/" />
        <CBC Id="0" configfile="CBC3_default.txt" />
        <CBC Id="1" configfile="CBC3_default.txt" />
        <CBC Id="2" configfile="CBC3_default.txt" />
        <CBC Id="3" configfile="CBC3_default.txt" />
        <CBC Id="4" configfile="CBC3_default.txt" />
        <CBC Id="5" configfile="CBC3_default.txt" />
        <CBC Id="6" configfile="CBC3_default.txt" />
        <CBC Id="7" configfile="CBC3_default.txt" />

        <CIC_Files path="${PH2ACF_BASE_DIR}/settings/CicFiles/" />
        <CIC2 Id="8" configfile="CIC2_default.txt" />
    </Hybrid>

    <!-- LHS -->
    <Hybrid Id="1" Status="1" LinkId="0">
        <Global>
            <Settings threshold="550" latency="19"/>
            <TestPulse enable="0" polarity="0" amplitude="0xFF" channelgroup="0" delay="0" groundothers="1"/>
            <ClusterStub clusterwidth="4" ptwidth="14" layerswap="0" off1="0" off2="0" off3="0" off4="0"/>
            <Misc analogmux="0b00000" pipelogic="0" stublogic="0" or254="1" tpgclock="1" testclock="1" dll="1"/>
            <ChannelMask disable=""/>
            <CIC2 enableBend="1" enableLastLine="0" enableSparsification="0" clockFrequency="320"/>
        </Global>

        <CBC_Files path="${PH2ACF_BASE_DIR}/settings/CbcFiles/" />
        <CBC Id="0" configfile="CBC3_default.txt" />
        <CBC Id="1" configfile="CBC3_default.txt" />
        <CBC Id="2" configfile="CBC3_default.txt" />
        <CBC Id="3" configfile="CBC3_default.txt" />
        <CBC Id="4" configfile="CBC3_default.txt" />
        <CBC Id="5" configfile="CBC3_default.txt" />
        <CBC Id="6" configfile="CBC3_default.txt" />
        <CBC Id="7" configfile="CBC3_default.txt" />

        <CIC_Files path="${PH2ACF_BASE_DIR}/settings/CicFiles/" />
        <CIC2 Id="8" configfile="CIC2_default.txt" />
    </Hybrid>

</OpticalGroup>
```

if using a single 2S FEH prototype remember to :

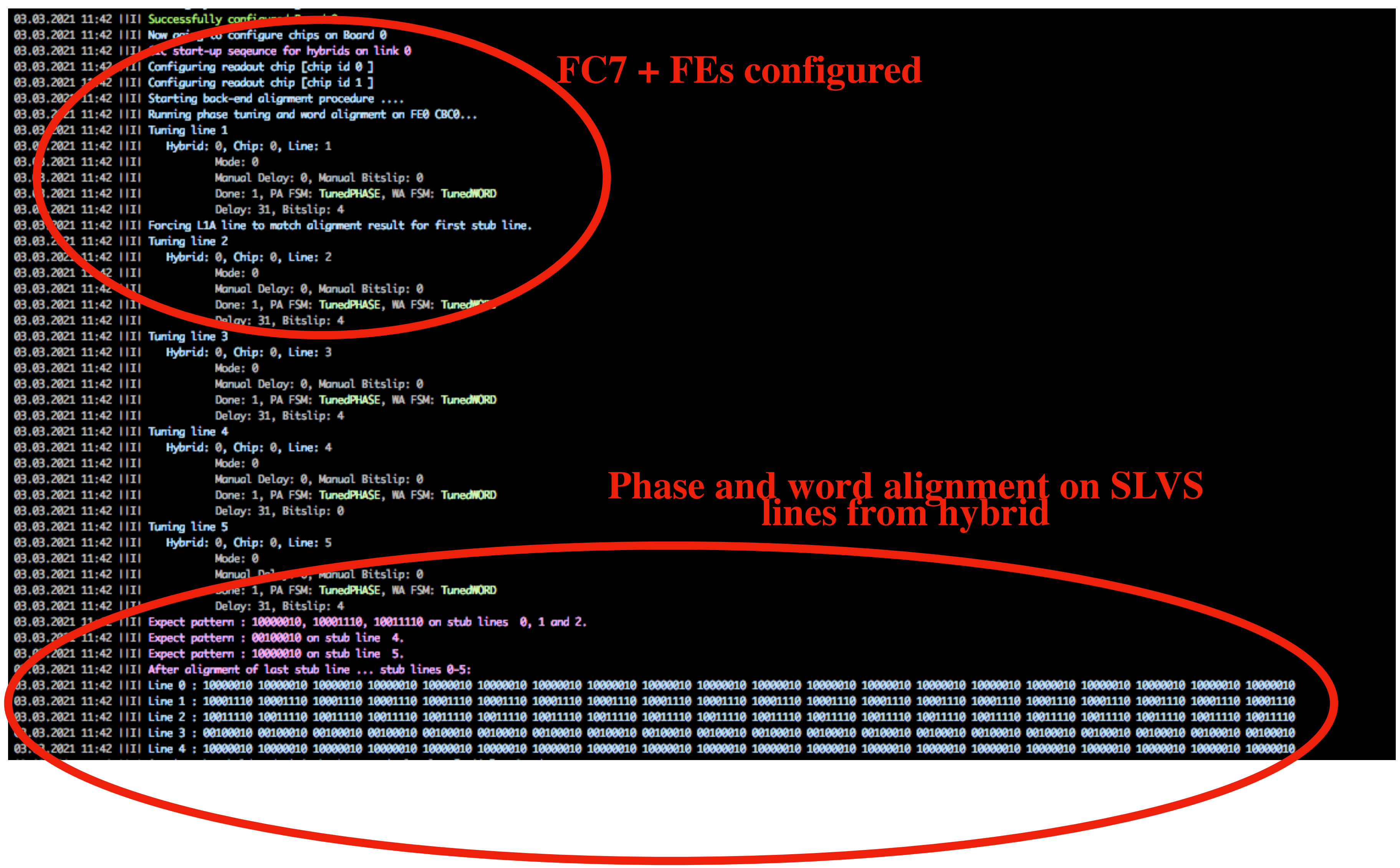
- *comment out the LHS hybrid !!*
- *disable the optical link [GBT, enable = 0] !!*
- *select the CIC version that corresponds to your hardware!*

Connecting the software to your front-end object

… *check that you can communicate with the front-end objects [electrical readout]*

- Use the 2S front-end hybrid test binary to verify that you can communicate with the front-end object [feh_2s_test -f $hardware_description_xml -b] :
  - Common (Tool) tools ConfigureHw() → reset FC7, configure FC7 with settings from xml, hard-reset to all front-end ASICs, configure all front-end ASICs based on hardware description xml
  - Back-end alignment (BackendAlignment) tool → prepare back-end for data taking (optimize sampling point fo electrical readout, align received data packets with 320 MHz clock boundaries, etc.)

Example of running on n 8CBC3 hybrid

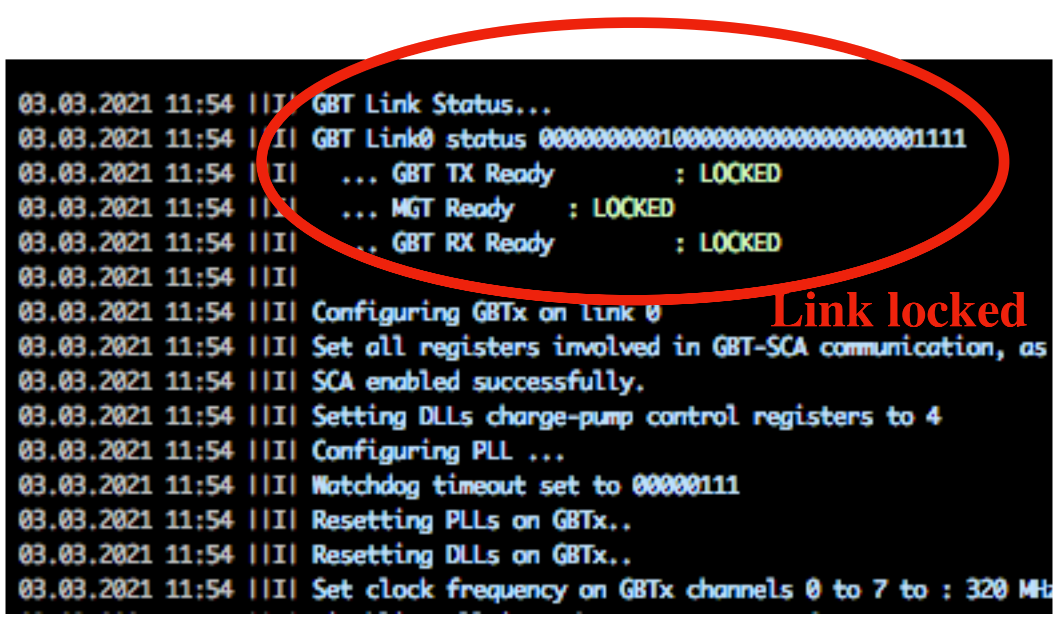


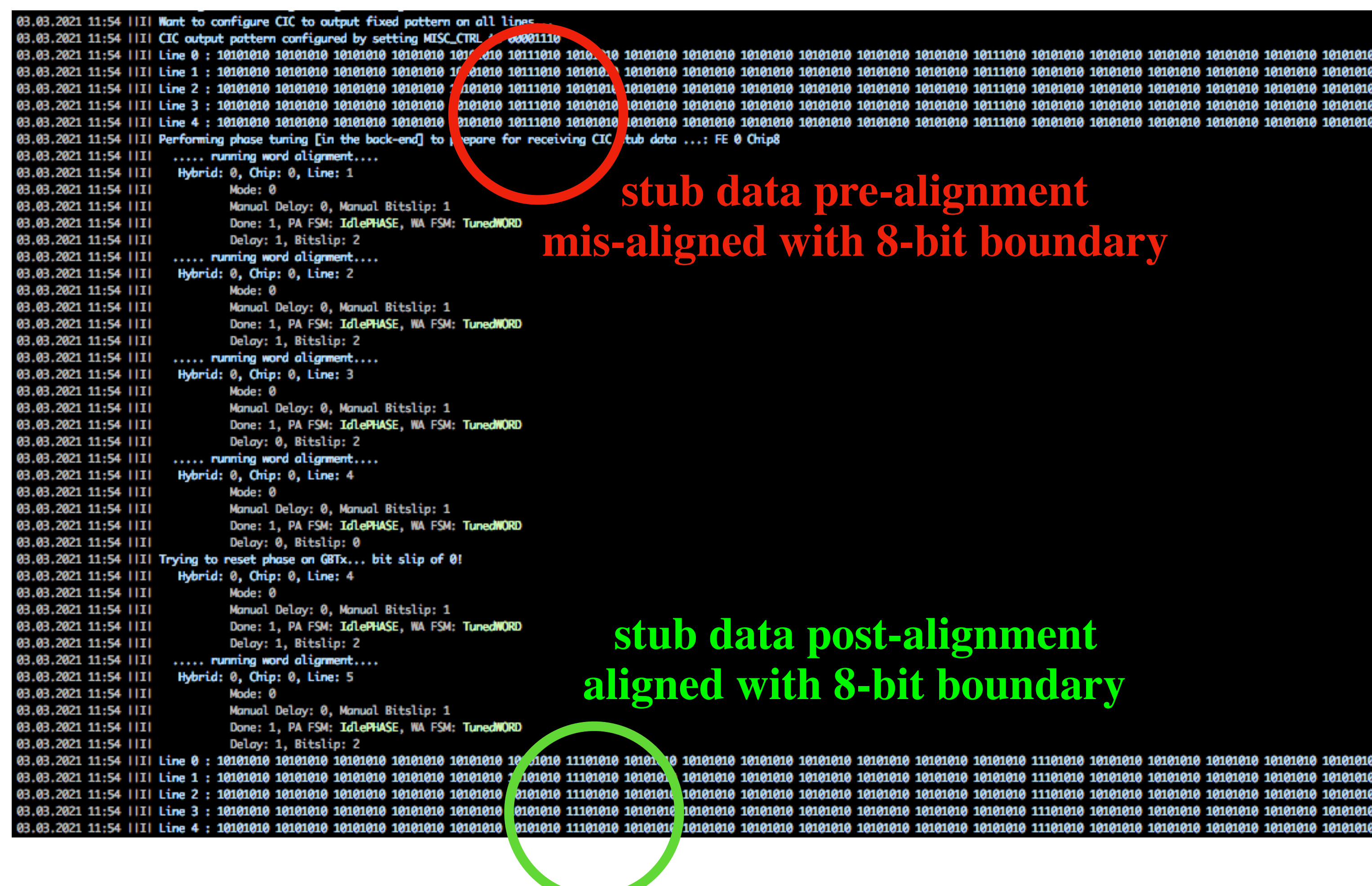
Connecting the software to your front-end object

… check that you can communicate with the front-end objects [optical readout]

- Use the 2S front-end hybrid test binary to verify that you can communicate with the front-end object [feh_2s_test -f $hardware_description_xml —withCIC—b] :
  - Common (Tool) tools ConfigureHw() ➝ reset FC7, configure FC7 with settings from xml, hard-reset to all front-end ASICs, configure all front-end ASICs based on hardware description xml
  - Configure ASICs on service hybrids, reset optical link in the back-end, check link lock
  - Back-end alignment (BackendAlignment) tool → prepare back-end for data taking (align received data packets with 320 MHz clock boundaries)
  - Cic alignment (CicAlignment) tool → prepare CIC for data taking (optimize sampling point for data from CBCs/MPAs, align received stub data with 320 MHz clock boundaries)

Example of running on a 2S skeleton





Running your first calibration

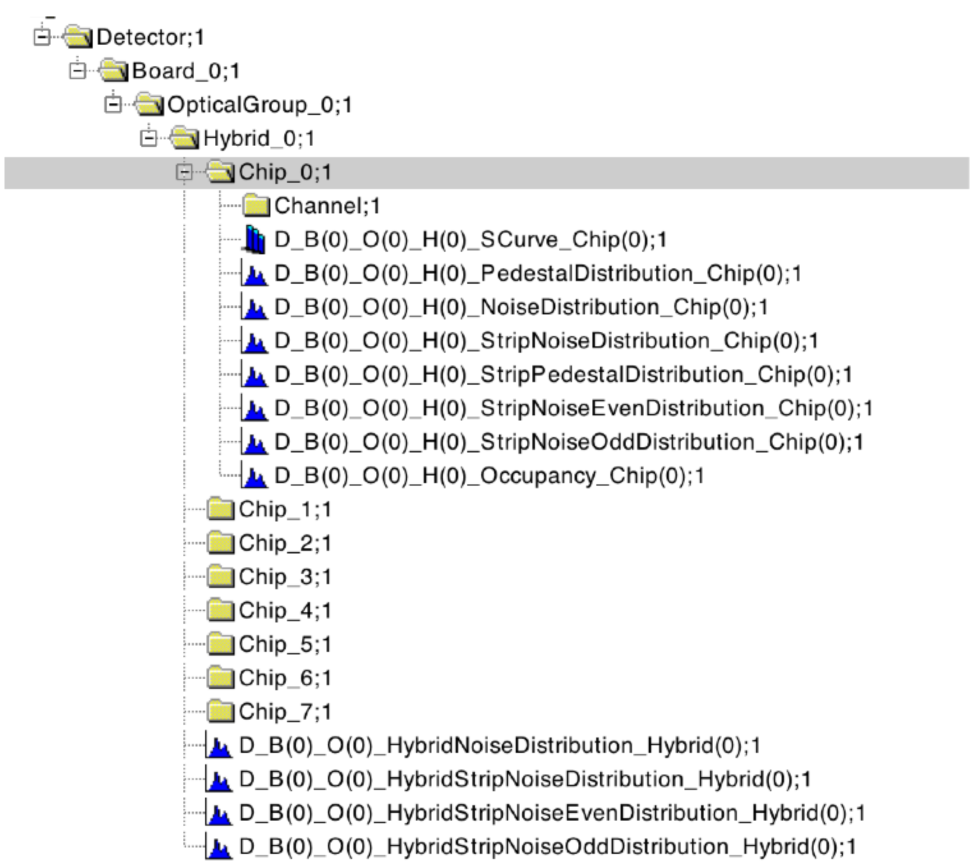
Connecting the software to your front-end object

… *check that you can equalize the threshold response of all channels on a hybrid*

- First,, use the 2S front-end hybrid test binary to simultaneously record S-curves (noise hit occupancy vs. threshold) curves for all channels on your test system [feh_2s_test -f $hardware_description_xml —withCIC* -a -m -b]
    - look at the last root file saved to Results : [root -l Results/$(ls -t Results | head -n1)/Hybrid.root]

<u>Directory structure of root file</u>

Board → Optical Group(s) → Hybrid(s) → (Readout) Chip(s)



- For each readout ASIC on a hybrid :
    1. noise hit occupancy vs. threshold
    2. distribution of pedestals [in threshold units] extracted from above
    3. noise vs. channel number on the readout ASIC
    4. pedestal vs. channel number on the readout ASIC
    5. noise vs. (nth) even channel on the readout ASIC
    6. noise vs. (nth) odd channel on the readout ASIC
    7. noise hit occupancy vs. channel number on the readout ASIC

- Summary plots for each hybrid in an optical group
    1. distribution of noise [in threshold units] for all readout channels on this readout hybrid
    2. noise vs. strip number (top + bottom) on this readout hybrid
    3. noise vs. (nth) bottom strip sensor connected to this readout hybrid
    4. noise vs. (nth) top strip sensor connected to this readout hybrid

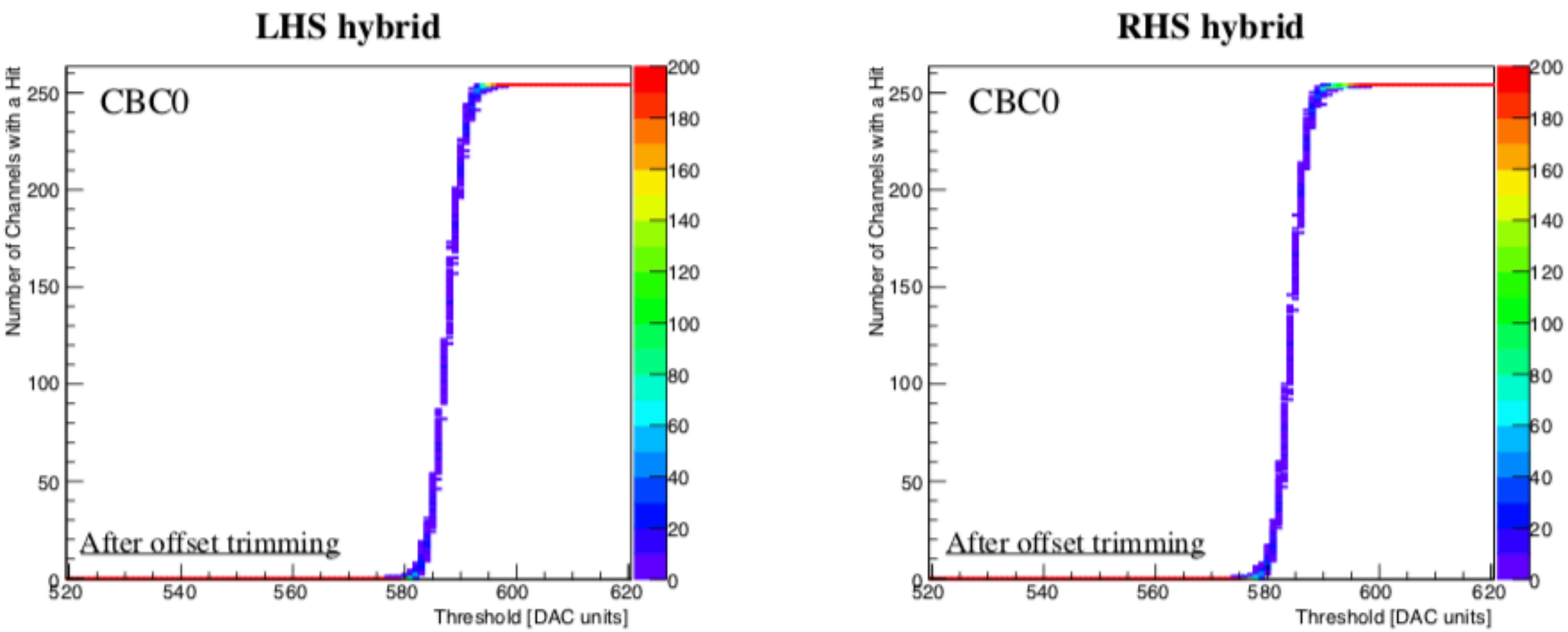
REMOVE IF USING A HYBRID WITH NO CIC

Connecting the software to your front-end object

… check that you can equalize the threshold response of all channels on a hybrid

- Definition of pedestal and noise in 2S modules :

  - Measure noise occupancy by sending 200 triggers, at a rate of 10kHz, to the assembly while scanning the threshold [Vcth] :



  - The difference in the number of hits between every threshold and the next is used characterize the noise per channel :

$$w_i = \frac{N_{\mathrm{hits}}(\mathrm{Vcth}_{i+1}) - N_{\mathrm{hits}}(\mathrm{Vcth}_i)}{\mathrm{Vcth}_{i+1} - \mathrm{Vcth}_i}$$

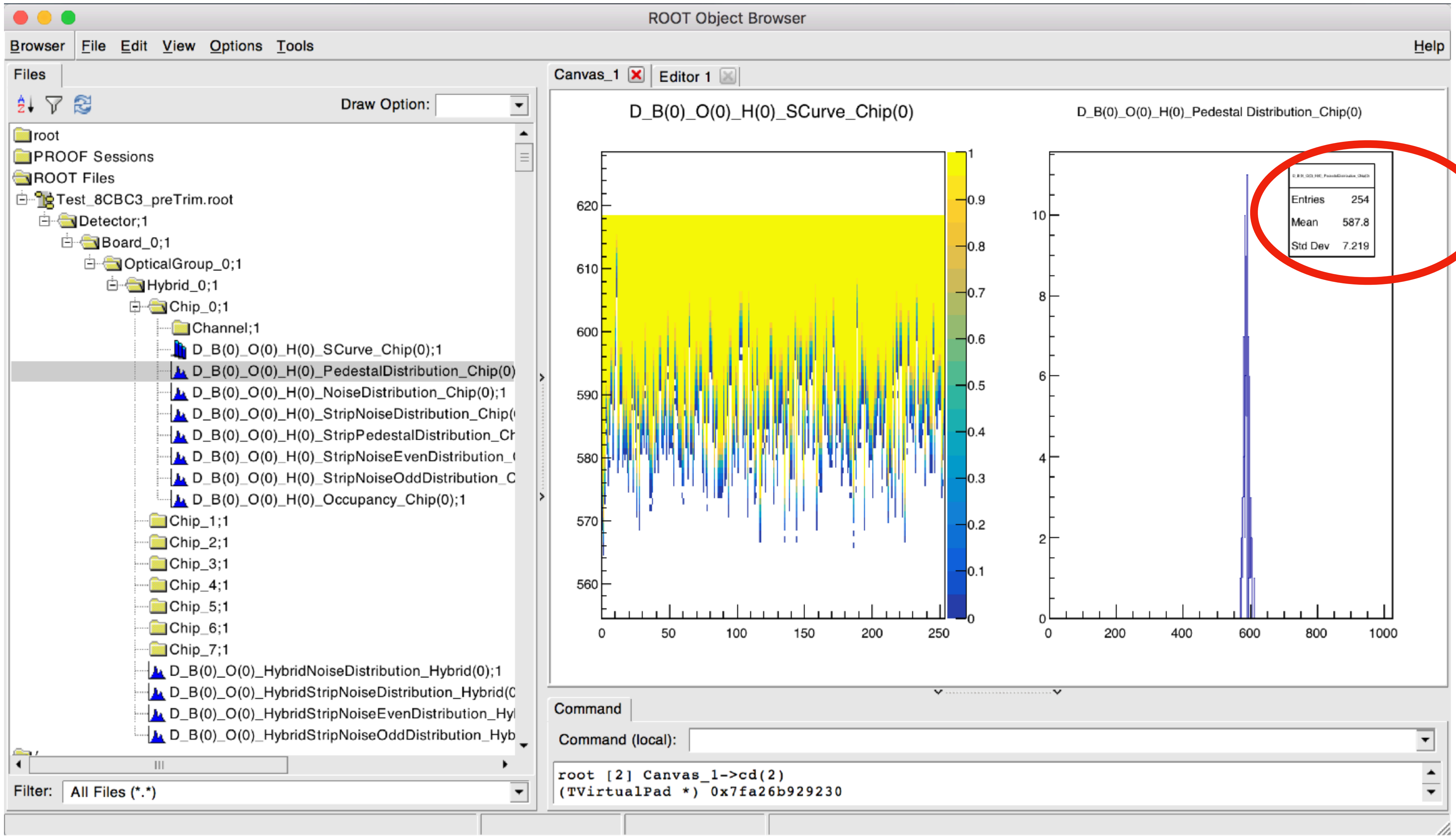
    - Pedestal [μ] :threshold at which the probability of seeing a hit is 50% $\mu = \frac{\sum w_i \mathrm{Vcth}_i}{\sum w_i}$
    - width of the noise distribution [σ] $\sigma^2 = \frac{\sum w_i (V_{\mathrm{cth}i} - \mu)^2}{\sum w_i}$

Connecting the software to your front-end object

… *check that you can equalize the threshold response of all channels on a hybrid*

- First,, use the 2S front-end hybrid test binary to simultaneously record S-curves (noise hit occupancy vs. threshold) curves for all channels on your test system [feh_2s_test -f $hardware_description_xml —withCIC* -a -m -b]
  - look at the last root file saved to Results : [root -l Results/$(ls -t Results | head -n1)/Hybrid.root]

S-curves and extracted pedestals pre-threshold equalization



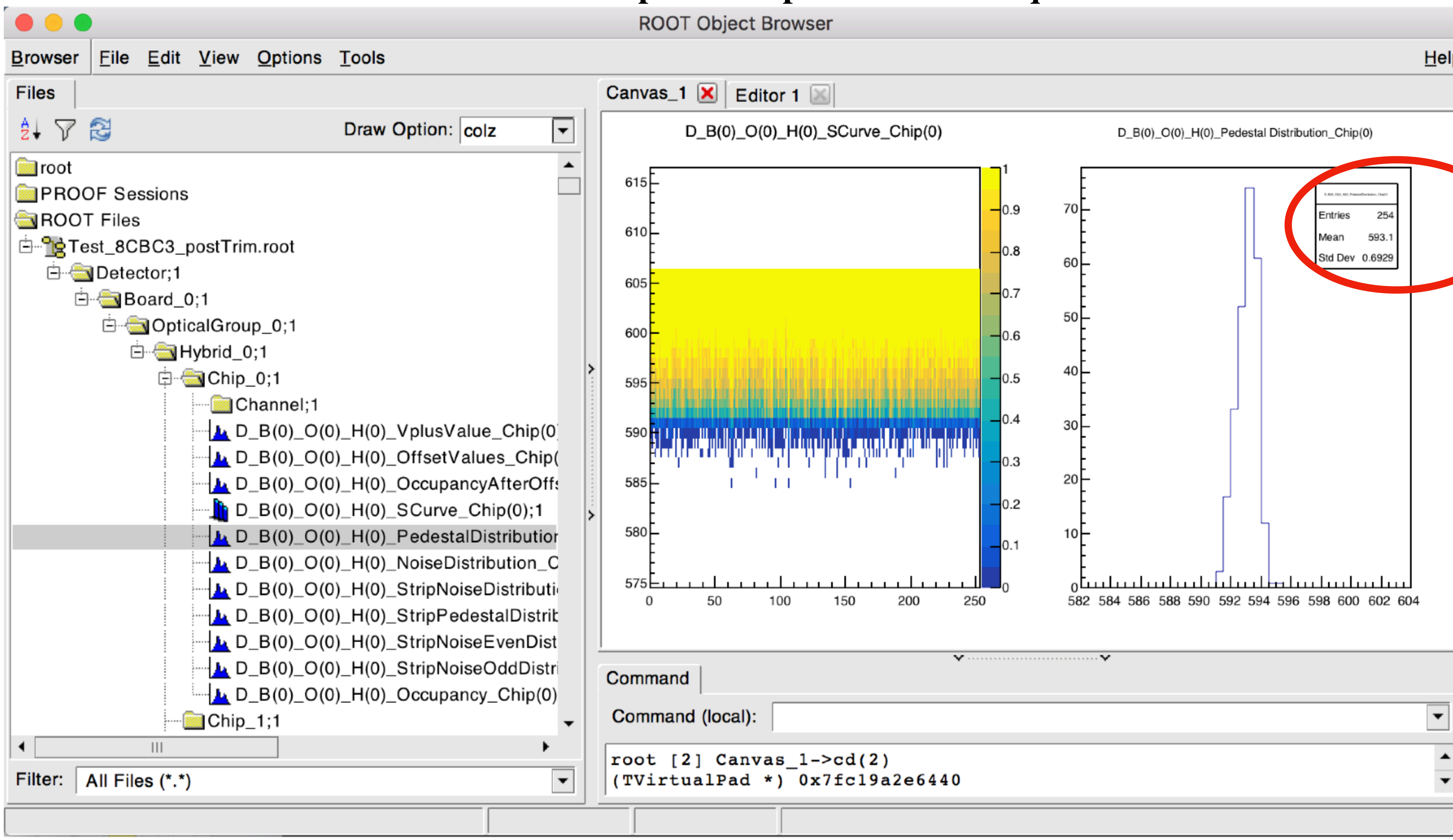
REMOVE IF USING A HYBRID WITH NO CIC

Connecting the software to your front-end object

… *check that you can equalize the threshold response of all channels on a hybrid*

- First, use the 2S front-end hybrid test binary to simultaneously equalize the threshold response of all channels, and then record S-curves (noise hit occupancy vs. threshold) curves [feh_2s_test -f $hardware_description_xml —withCIC* -a -t -m -b]
  - look at the last root file saved to Results : [root -l Results/$(ls -t Results | head -n1)/Hybrid.root]

S-curves and extracted pedestals post-threshold equalization



REMOVE IF USING A HYBRID WITH NO CIC

Optimizing the calibration

… how does the threshold dispersion depend on the number of points used in the equalization algorithm?

- Go to the 'threshold equalization' sheet in the google doc [1] and select a cell, i.e. a combination of values, that is free and mark it with your name

```xml
<Settings>
    <!--PedestalEqualization-->
    <Setting name="Nevents">100</Setting>
    <Setting name="TestPulsePotentiometer">0x00</Setting>
    <Setting name="HoleMode">0</Setting>
    <Setting name="VerificationLoop">1</Setting>
    <Setting name="MaskChannelsFromOtherGroups">0</Setting>
    <Setting name="FitSCurves">0</Setting>
    <Setting name="PlotSCurves">1</Setting>
    <!--Pedestal and Noise measurement-->
    <Setting name="PedeNoisePulseAmplitude">0</Setting>
    <Setting name="PedeNoiseMinBreakCount">5</Setting>
    <Setting name="PedeNoiseStepSize">3</Setting>
```

- Modify the xml to allow you to fill in the google doc[1] with the results of the parameter scan :
    - Nevents [points per step] : number of events collected for each point of bitwise threshold trimming scan

[1] https://docs.google.com/spreadsheets/d/1tFlyPXvcE3-NtkVmwrtrYP56YTXCbVBjkNnBSKYrzJc/edit#gid=0

Optimizing the calibration

… *how do the pedestal and noise depend on the way the data is collected?*

- First, pull the latest commit from the school's branch : [git fetch; git pull origin daqSchool2021]
- Recompile and set-up the environment [make -C build -j4; source setup.sh]
- The following settings have been made configurable in the xml :

```xml
<Settings>
    <!--PedestalEqualization-->
    <Setting name="Nevents">100</Setting>
    <Setting name="TestPulsePotentiometer">0x00</Setting>
    <Setting name="HoleMode">0</Setting>
    <Setting name="VerificationLoop">1</Setting>
    <Setting name="MaskChannelsFromOtherGroups">0</Setting>
    <Setting name="FitSCurves">0</Setting>
    <Setting name="PlotSCurves">1</Setting>
    <!--Pedestal and Noise measurement-->
    <Setting name="PedeNoisePulseAmplitude">0</Setting>
    <Setting name="PedeNoiseMinBreakCount">5</Setting>
    <Setting name="PedeNoiseStepSize">3</Setting>
```

- Go to the 'pedestal/threshold' measurement sheet in the google doc [1] and select a cell, i.e. a combination of values, that is free and mark it with your name
- Modify the xml to allow you to fill in the google doc[1] with the results of the parameter scan :
  - Nevents [points per step] : number of events collected for each point of bitwise threshold trimming scan
  - PedeNoiseStepSize [size of the step] : step size in DAC units

[1] https://docs.google.com/spreadsheets/d/1tFlyPXvcE3-NtkVmwrtrYP56YTXCbVBjkNnBSKYrzJc/edit#gid=0

Writing your first calibration

Writing a calibration

… measure occupancy for a fixed range of thresholds , plot data, extract pedestal

- Skeleton code provided in the daqSchool2021 branch of the repository :
  - complete top level source provided - but have a look at the code and take note of the steps required before executing *any* calibrations
  - skeleton tool provided (up-to you to fill in the blanks) : measure noise hit occupancy for threshold going from 550 to 650
  - skeleton DQM plotter utility provided (up-to you to fill in the blanks) : plot a 2D histogram (x = channel #, y = threshold, z = occupancy) and use it to estimate the pedestal

src/run_simple_scurve.cc

```
Tool cTool;
std::stringstream outp;
cTool.InitializeHw ( cHWFile, outp );
cTool.InitializeSettings ( cHWFile, outp );
LOG (INFO) << outp.str();
outp.str ("");
cTool.ConfigureHw ();
cTool.CreateResultDirectory ( cDirectory );
cTool.InitResultFile ( "SimpleSCurveResult" );

Timer t;
t.start();

// align back-end
BackEndAlignment cBackEndAligner;
cBackEndAligner.Inherit(&cTool);
cBackEndAligner.Start(0);
cBackEndAligner.waitForRunToBeCompleted();
// reset all chip and board registers
// to what they were before this tool was called
cBackEndAligner.Reset();

// if CIC is enabled then align CIC first
if(cWithCIC)
{
    CicFEAlignment cCicAligner;
    cCicAligner.Inherit(&cTool);
    cCicAligner.Start(0);
    cCicAligner.waitForRunToBeCompleted();
    // reset all chip and board registers
    // to what they were before this tool was called
    cCicAligner.Reset();
    cCicAligner.dumpConfigFiles();
}

// EDIT - begin //
// Create here your calibration, inherit from Tool run the calibration
SimpleSCurve theSimpleSCurve;
theSimpleSCurve.Inherit (&cTool);
theSimpleSCurve.Initialise ();
theSimpleSCurve.runSimpleSCurve();
theSimpleSCurve.writeObjects();
// EDIT - end //

//Tool old style command (some of them will vanish/merged)
cTool.dumpConfigFiles();
cTool.resetPointers();
cTool.SaveResults();
cTool.WriteRootFile();
cTool.CloseResultFile();
cTool.Destroy();
t.stop();
t.show ( "Time to Run Calibration example" );
```

top level source to produce executable

tools/SimpleSCurve.*

```
using namespace Ph2_HwDescription;
using namespace Ph2_HwInterface;
using namespace Ph2_System;

class SimpleSCurve : public Tool
{

  public:
    SimpleSCurve();
    ~SimpleSCurve();

    void Initialise             (void);
    void runSimpleSCurve(void);
    void writeObjects           (void);

    //State machine
    void Start                  (int currentRun) override;
    void Stop                   (void) override;

  private:
    uint32_t fEventsPerPoint;

    #ifdef __USE_ROOT__
      //Calibration is not running on the SoC: Histogrammer
      DQMHistogramSimpleSCurve fDQMHistogramSimpleSCurve;
    #endif
};

#endif
```

user specific tool

DQMUtils/DQMHistogramSimpleSCurve.*

```
class DQMHistogramSimpleSCurve : public DQMHistogramBase
{

  public:
    /*!
     * constructor
     */
    DQMHistogramSimpleSCurve ();

    /*!
     * destructor
     */
    ~DQMHistogramSimpleSCurve();
```

DQM plotter tool