ANTS2 Acquisition GUI

Functions and scripts sequence

1. Button “Connect TRB3 DAQ”:

The ANTS2 script function **prepare_DAQ()** is called when this button pressed. It has three steps: **1)** send a command to startup TRB3 DAQ, **2)** wait for the confirmation that the DAQ system is prepared and **3)** send the initial trigger threshold values (specified in the GUI).

These three steps are explained in detail in what follows.

1.1 Called local system script:
/home/daq/DAQ_Software/TRB_DAQ_using_DABC/**startDAQ_remotely.sh:**

```
#!/bin/bash
ssh odroid@10.0.0.1 'DISPLAY=:0 nohup /home/odroid/trbsoft/userscripts/trb49/startup_from_remote.sh'
```

So, in the Odroid machine the script **startup_from_remote.sh** is executed:

```
#!/bin/bash

xterm -e /home/odroid/trbsoft/userscripts/trb49/startup_TRB49_remotely.sh ;
export DAQOPSERVER="localhost:1" ;

# ';" means that the commands are executed in sequence. The next will only start when the previous finishes, JM.

xterm -e /home/odroid/trbsoft/userscripts/trb49/launch_cts.sh  &

# Load CTS configuration
cd /home/odroid/trbsoft/userscripts/trb49/
sh ./cts-dump-remotely.sh
```

As can be seen, the above script launches by its own the two following scripts from “xterm” terminals (in the controller machine - Odroid):

‘startup_TRB49_remotely.sh’ and ‘launch_cts.sh’:

In the end, a third script is executed to load the CTS configurations: cts-dump-remotely.sh

1.2 Called local script /home/daq/DAQ_Software/TRB_DAQ_using_DABC/Aux_scripts/ **open_url2.sh**

This script waits for the end of TRB3 DAQ startup. At that time, the CTS web controller is available through any browser. So, the way that is being used to check if the TRB3 DAQ startup was successfuly performed, is to listen to the answer of a “curl” command that checks the avaibility of the *url* of the CTS web controller (e.g.: 10.0.0.1:1234/cts/cts.htm). The interrogation is made each 3 seconds and in case the status of the *url* is “OK” the script finishes. In this case, the next procedures of the main function are called (see the next function called in 1.3, below).

The **open_url2.sh** script is called with a timeout (value in milliseconds, always ~20% above the average time that TRB3 DAQ takes to startup). If the timeout is overcame, a message is shown in the GUI, asking the user to confirm that the remote controller is switched ON and that there is connection with it (e.g. sending a “ping” command).

The **open_url2.sh** script code is shown below:

```
#!/bin/bash
url=$1
while true
do
        STATUS=$(curl -s -o /dev/null -w '%{http_code}' http://$url)
        if [ $STATUS -eq 200 ]; then
                echo "Got 200! TRB3 DAQ started up. CTS web control available. Acquisitions can be
done."
                break
         else
                echo "...Still waiting. Got $STATUS : TRB3 DAQ NOT prepared yet. CTS webC control NOT
available yet..."
         fi
        sleep 3
done
```

1.3 Called ANTS2 script function change_CTS_trigger_thresholds()

This function is only called if TRB3 DAQ is prepared (see 1.2).

This function takes the triggering thresholds from the respective GUI text fields and calls a local script to write (using the *sed* system application) that values as the arguments of a command sent to the remote controller (Odroid) by other script.

/home/jsm/Documents/TRB3/TRB_DAQ_using_DABC/**set_Thresholds_CTS_Trigger_channels_9_and_10.sh**:

```
#!/bin/bash

THRESHOLD_CH9_NEW=$1      # Takes the 1st argument

THRESHOLD_CH10_NEW=$2     # Takes the 2nd argument

DIR_remote_change_thr_script="/home/jsm/Documents/TRB3/TRB_DAQ_using_DABC/"

sed -e "s|THRESHOLD_CH9|$THRESHOLD_CH9_NEW|g" -e "s|THRESHOLD_CH10|$THRESHOLD_CH10_NEW|
g" "$DIR_remote_change_thr_script/remote_change_CTS_trigger_thresholds_base.sh" >
"$DIR_remote_change_thr_script/remote_change_CTS_trigger_thresholds.sh"
```

As can be seen in the code of the scipt above, the *sed* command will set the threshold values (passed by argument) in the script file remote_change_CTS_trigger_thresholds.sh substituting the words ‘THRESHOLD_CH9’ and ‘THRESHOLD_CH10’ in a “base” file called remote_change_CTS_trigger_thresholds_**base**.sh.

Then, the script **remote_change_CTS_trigger_thresholds.sh** is executed locally. This script (see below) executes other script (called change_CTS_thresholds.sh) in the remote controller (odroid@10.0.0.1):

```
#!/bin/bash

ssh odroid@10.0.0.1 'DISPLAY=:0 nohup /home/odroid/trbsoft/userscripts/trb49/change_CTS_thresholds.sh
THRESHOLD_CH9 THRESHOLD_CH10'
```

The code in the script **change_CTS_thresholds.sh** is the following:

```
#!/bin/bash

DIRBINtrbcmd2="/home/odroid/trbsoft/trbnettools/bin"

cd /home/odroid/trbsoft/userscripts/trb49

DAQ_TOOLS_PATH=~/trbsoft/daqtools
export PATH=$PATH:$DAQ_TOOLS_PATH
export PATH=$PATH:$DAQ_TOOLS_PATH/tools

export DAQOPSERVER=localhost:1

export TRB3_SERVER=trb049:26000               #change to the respective host name (the same than
/etc/hosts)
export TRBNETDPID=$(pgrep -f "trbnetd -i 1")
#echo "- trbnetd pid: $TRBNETDPID"

THRESHOLD_CH9=$1
#echo $THRESHOLD_CH9
THRESHOLD_CH10=$2
#echo $THRESHOLD_CH10

#Convert thresholds to hexadecimal
THRESHOLD_CH9_HEX=$(printf '%x\n' $THRESHOLD_CH9)
THRESHOLD_CH10_HEX=$(printf '%x\n' $THRESHOLD_CH10)

#Switch ON the invert checkbox
$DIRBINtrbcmd2/trbcmd w 0xc310 0xa013 0x000100$THRESHOLD_CH9_HEX
$DIRBINtrbcmd2/trbcmd w 0xc311 0xa013 0x000100$THRESHOLD_CH10_HEX
sleep 1s

#Switch OFF the invert checkbox
$DIRBINtrbcmd2/trbcmd w 0xc310 0xa013 0x000000$THRESHOLD_CH9_HEX
$DIRBINtrbcmd2/trbcmd w 0xc311 0xa013 0x000000$THRESHOLD_CH10_HEX
```

2. Button "**START Acquisition**"

2.1 Called ANTS2 script function **start_simple_acquisition()**:

This function is divided in **four steps**:

2.1.1 "Preparation": Calls the function

create_hlds_dir_and_write_into_xml_hlds_dir_and_size()

This function calls the local bash script "/home/daq/DAQ_Software/TRB_DAQ_using_DABC/" **create_hlds_dir_and_substitute_dir_and_size_in_EventBuilder_xml.sh**:

- Creates the directory chosen by the user (text field)
- Writes in the EventBuilder_TRB49.xml file the directory of .HLD files and the file size in MB.
- Writes in the script ‘start_remote_acquisition.sh’ (to be executed remotley) the number of seconds of the acquisition.

Code of the script **start_simple_acquisition():**

```
#!/bin/bash

DIR_HLDS_NEW=$1
SIZE_HLDS_NEW=$2
ACQUISITION_TIME_NEW=$3

DIR_EventBuilder_xml="/home/daq/DAQ_Software/TRB_DAQ_using_DABC/"

# Create the directory DIR_HLDS_NEW:
mkdir -p $DIR_HLDS_NEW

# Replace the path of the HLDs files (passed as the 1st argument) and the size of the HLDs files (2nd argument)
sed -e "s|SIZE_HLD|$SIZE_HLDS_NEW|g" -e "s|DIR_HLDS|$DIR_HLDS_NEW|g"
"$DIR_EventBuilder_xml/EventBuilder_TRB49_base.xml" >
"$DIR_EventBuilder_xml/EventBuilder_TRB49.xml"

# Replace the acquisition time in the script that sends a ssh command to execute other script in the remote controller of TRB (Odroid-C2 mini-PC)
sed -e "s|ACQUISITION_TIME|$ACQUISITION_TIME_NEW|g"
"$DIR_EventBuilder_xml/start_remote_acquisition_base.sh" >
"$DIR_EventBuilder_xml/start_remote_acquisition.sh"
```

Note for the reader not familiar with bash scripting:

The $1, $2 and $3 are the first, second and third arguments of the script respectively.
The “sed” command is a system command for changing expressions of a file. The file with the substituted expressions can have a different name, as it is being done in the script start_simple_acquisition().

2.1.2 Launch the script **startDABCnoSleep.sh**

/home/daq/DAQ_Software/TRB_DAQ_using_DABC/**startDABCnoSleep.sh**

Note that this script executes the command below (‘dabc_exe’ with a file as argument), which needs that the configuration file called ‘**EventBuilder_TRB49.xml**’ exists. It is in the same directory of the script.

Command executed by the startDABCnoSleep.sh script:

```
| dabc_exe /home/daq/DAQ_Software/TRB_DAQ_using_DABC/EventBuilder_TRB49.xml'
```

This .xml config. file, among other settings, have the directory and size (MB) of the .HLD files that will be sent by the TRB3 board (throught the Gbit Ethernet connection). The line in the file where that settings are defined has the structure below:

| <OutputPort **name=**"Output1" **url=**"hld://path_of_hlds_files_directory/dabc.hld?maxsize=size"/>

Example for a directory in the external disk called 'B16_Archive' and files of size 100 MB:

<OutputPort name="Output1" url="hld:///**mnt/B16_Archive/hlds/2019-01-17/Acquisition1**/dabc.hld?maxsize=**100**"/>

2.1.3 Call the ANTS2 script **change_CTS_trigger_thresholds()**

This script sets the trigger thresholds (taken from the respective GUI textfields). Please see the step 1.2 above for the function details.

2.1.4 Executes locally the bash script **start_remote_acquisition.sh** to START REMOTE ACQUISITION

Switches ON the *Ch9* and *Ch10* in TRB CTS web controller (the respective *checkboxes*).

Note that the script calls a script from Odroid (controller machine) passing the time of acquisition (in seconds) as argument. This way the switch OFF of the *Ch9* and *Ch10* is made in the Odroid.

Code of the script **start_remote_acquisition.sh**:

```
#!/bin/bash

#Example for an acquisition of 240 seconds
ssh odroid@10.0.0.1 'DISPLAY=:0 nohup /home/odroid/trbsoft/userscripts/trb49/start_acquisition.sh 240'
```

Attention: (to be turned around with a system timer)
An ANTS2 scripting sleep() function is called, to wait the selected "acquisition_time" ms (GUI text field) before calling a command to kill the opened xterm consoles:

core.sleep(acquisition_time * 1000 * 2.2) // At this time sleep() is faster then the real time (maybe when multi-core is being used). It works fine for simple tests...

3. Button **"Process .HLD files"**

The function **processHLDsAndSendDATFiles_2()** is called.

It processes the .hld files present in the acquisition directory (specified in the respective GUI text field). The processed files are saved with the same name and the extention .DAT.
The processment of the files follows the configuration in the TRBReader application.

Typically, the signal **extraction method** is “Integrate waveforms” (from sample #11 to sample #28, when digitizing 30 samples per waveform). The pedestals (baseline) are not dynamically subtracted, but rather subtracted when loading .DAT file in ANTS2. This option must be activated in ANTS2 if pedestals were previoulsy calculated and loaded. Optionally, the convertion factor from electronic channels into photoelectrons can be also loaded for each sensor.

Essentially, the function searches for .HLD files and calls a native script function from TRBReader “hld” unit:

```
hld.ProcessFile(hld_file_name, dat_file_name)
```

4. Pedestals

4.1 Button “**1) Start Pedestals acquisition**”

Selection of the directory and size of .HLD files
Selection of acquisition time (in seconds)

The ANTS2 script **function start_pedestals_acquisition()** is called.

It consists of three main steps: **1) Preparation:** create and write the directory for the .HLD files as well as the file size and acquisition time into the respective bash scripts, **2) Launch statDABCnoSleep.sh** script to prepare the destiny (of the .HLD files) machine for the acquisition, **3)** Call a script to **start remotely the acquisition**.

4.1.1 Preparation:

Call
create_pedestals_hlds_dir_and_write_into_xml_pedestals_hlds_dir_size_and_acq_time() function.

This function takes the parameters (.HLD files directory and size; acquisition time) from the respective GUI text-fields and pass them as argument of the the bash script **create_hlds_dir_and_substitute_dir_and_size_in_EventBuilder_xml.sh.** The task of this script is explained in 2.1.1.

4.1.2 Launch the scipt **statDABCnoSleep.sh.**

This script prepared the machine defined as destiny machine (for where .HLD files will be transfered from TRB3 board). Please, see 2.1.2 for details.

4.1.3 Executes locally the bash script **start_remote_acquisition_of_pedestals.sh**

to start remote acquisition of pedestals. The random generator is switched ON in the controller machine (Odroid-C2).

Switches ON the *Random 0* channel in TRB CTS web controller (it is a *checkbox*).
The acquisition rate can be selected from the CTS webpage (10.0.0.0:1234/cts/cts.htm).

Note that the script calls a script from Odroid (controller machine) passing the time of acquisition (in seconds) as argument. This way the switch OFF of the *Random 0* channel is made in the Odroid.

Code of the script **start_remote_acquisition_of_pedestals.sh:**

```
#!/bin/bash

# Example for an acquisition time of 90 seconds
ssh odroid@10.0.0.1 'DISPLAY=:0 nohup /home/odroid/trbsoft/userscripts/trb49/start_acquisition_of_pedestals.sh 90'
```

Note that the value of the acuisition time (in seconds) is written in the preparation step (4.1.1).

4.2 Button **“2) Process pedestals .HLD files”**

The procedure is the same as in 3 (above).

4.3 Button **“3) Calculate pedestals”**

When the button “Calculate Pedestals” is pressed a custom ANTS2 function is called: calculatePedestals(). See the code in appendix.

This function, for each sensor, finds the first peak in the histogram of electronic channels. This peak corresponds to the signal magnitude in the measurement system (measured in number of electronic channels) that is always present, there is or there isn’t photons impinging the sensors. The first peak can be assumed to be the baseline value (pedestal).

5. REAL-TIME

5.1 Real-time acquisition configuration parameters

- Directory to save the .HLD sent from TRB3 board (throught Ethernet protocol)
- Size of the .HLD files
- Real-time buffer size (in number of events)
- Number of events to be stored (check if it is being used – the purpose is to have more events for posterior analysis, if needed)

- Optional processes during acquisition

  a) Activate Chi2 monitoring
  b) Delete .HLDs files after processing.

5.2 Functions of the real-time chain when “Stat Acquire in Real-time” is pressed.

Flowchart

5.3 Chi2 monitoring option

If the GUI checkbox “Activate Chi2 MONITORING” is checked the average Chi2 of each bunch of events (events present in the .HLD file being processed) will be compared with a reference Chi2 value. An alarm is triggered (message box) notifing the user. In that case it is recommended to stop the real-time acquisition

When the checkbox “Activate Chi2 MONITORING” is checked, three parameters are allowes to be specified:

- Chi2 manually set reference value
- Number of events to control Chi2
- Goodness margin factor (value between 0.0 and 1.0).

If the Chi2 monitoring is selected, the first step is to set the selected Chi2 reference value. For this purpose a C++ function was implemented to be called from ANTS2 scripts:

```
custom.setIsReferenceChi2Calculated(bool, chi2_reference_value)
```

The funtion above besides setting the Chi2 reference value, sets initializes a boolean variable which is always true in this phase of the real-time chain implementation. It can be used later, in a version of the system that calculates the Chi2 reference value based on the first bunch of events. In this case, the Chi2 monitoring can only start when the reference value is “calculated”. This is the reason for the boolean value.

The function for the Chi2 monitoring was also implemented in C++ (to be called in ANTS scripting mode). It averages the chi2 of a bunch of events and compares with the reference value using the percentual margin passed as argument. The calling of the function from script is as follows:

```
trigger_alarm = custom.chi2Control2(num_events_to_control_chi2, chi2_goodness_margin_factor) // ARGUMENTS - 1st: unsigned long num_events_to_control, 2nd: float margin_factor and 3rd: float percentage_of_bads
```

For the beginning of the acquisition, a minimum number of events is set to averaged its Chi2 value. If the events are less than that limit, the chi2 monitoring do not start yet.

5.4 Extra features

5.4.1 RESET button

When RESET is pressed, the data structure that holds the events (EventDataHub) is cleared. The .HDLs and .DAT files in the current real-time directory are also deleted.

6. Light response functions (LRFs) calculation

The directory with the .DAT files of the flood acquisition to be used for the LRFs calculation must be selected in the correspondent GUI text-field.

The LRFs parameters specified in the GUI are loaded into ANTS2 and used when the button “Calculate new LRFs” is pressed.

Appendices of scripts and automation

A. Some custom writen **ANTS2 script functions**:

A.1 function prepare_DAQ()

```
{
	var timeout = 45000 // Please, check if this value is ~20% higher than the average time (in milliseconds) that TRB3 takes to start up (reference value: ~25000 ms)
	var elapsed_time = 0

core.StartExternalProcess( "/home/jsm/Documents/TRB3/TRB_DAQ_using_DABC/startDAQ_remotely.sh", [], true, 2000)


	custom.startTimer2()

core.StartExternalProcess( "/home/jsm/Documents/TRB3/TRB_DAQ_using_DABC/Aux_scripts/open_url2.sh", ["10.0.0.1:1234"], true, timeout)

	elapsed_time = custom.getElapsedTimeTimer2()
	core.print("elapsed_time: " + elapsed_time)

	if(elapsed_time < timeout)
	{
		change_CTS_trigger_thresholds()

		gui.setEnabled("STARTSimpleAcquisition", true )
		gui.setEnabled("label_acquisition_time_seconds", true )
		gui.setEnabled("edit_acquisition_time_seconds", true )

		gui.textAppendPlainText("text_field", "Connection established with TRB3 controller machine.\nTRB3 DAQ started up. CTS web control available. Acquisitions can be done.")
	}
	else
		gui.textAppendPlainText("text_field", "TRB3 DAQ NOT prepared yet. CTS webC control NOT available.\nPlease check if the controller machine is switched on and that you can ping it.")


}
```

A.2 function calculatePedestals()

```
{
//Load pedestals to an array of 64 elements


//var dir = "/data/GAGGCam/hlds/2018-04-10/ColdCamera/Pedestals_Bias_27_0V_1kHz_SiPMPatchsAssembledAgain_4"
//var dir = "/home/jsm/Documents/Real-Time-Dat-PEDESTALS-Files-from-TRBReader"
var dir = gui.editGetText("edit_pedestals_hlds_files_dir")


// 1) PROCESS PEDESTALs .hdl files -> integrate the signals -> generate .dat files:
// var s_hlds_pedestals_files_dir = gui.editGetText("edit_pedestals_hlds_files_dir")
// processHLDsAndSendDATFiles(dir, "pedestals")


// 2) CALCULATE PEDESTALS
var fileNamePattern = "*.dat"
var files = core.SetNewFileFinder( dir, fileNamePattern )
```

```
events.ClearEvents( )
config.Replace( "DetectorConfig.LoadExpDataConfig.Preprocessing" , false )


for(var f = 0; f < files.length; f++)
{
var namefile = dir + "/" + files[f]
void events.LoadEventsAscii(namefile, true )
}


hist.DeleteAllHist()


var numEvents = events.GetNumEvents()
var numPms = pms.CountPM()


var pedestals = []
for (var i=0; i < numPms; i++)
{
pedestals[i] = 0
}

var electronic_channel = 0
var prefix_Hist_name = "Peaks_PM_"
var iPM_Hist_name =""
var iPM = 0


// Create one Histogram for each PM

//var num_bins = numEvents/37

for(var iPM = 0; iPM < numPms; iPM++)
{
iPM_Hist_name = prefix_Hist_name + iPM
//hist.NewHist(iPM_Hist_name, 135, -18470, -18200)
hist.NewHist(iPM_Hist_name, 135, 0, 0)
//hist.NewHist(iPM_Hist_name, num_bins, 0, 0)
//hist.NewHist(iPM_Hist_name, 120, 0, 0)
hist.SetTitles(iPM_Hist_name, "ElectronicChannel", "Counts")
}



// Fill the Histograms
for (var iEvent=0; iEvent<numEvents; iEvent++)
{
for(var iPM = 0; iPM < numPms; iPM++)
{
iPM_Hist_name = prefix_Hist_name + iPM
electronic_channel = events.GetPMsignal(iEvent, iPM)
hist.Fill(iPM_Hist_name, electronic_channel, 1)
}
}



// Find the fist peak, which is the pedestal and save in the array 'pedestals'
```

```
for(var iPM = 0; iPM < numPms; iPM++)
{
iPM_Hist_name = prefix_Hist_name + iPM
var peaks = hist.FindPeaks(iPM_Hist_name, 4, 0.2)
pedestals[iPM] = peaks[iPM,0]
core.print("Peaks for PM# " + iPM + " are at: " + peaks)
hist.SetLineProperties(iPM_Hist_name, 3, 1, 1)
hist.Draw(iPM_Hist_name, "")
//grwin.AddToBasket(iPM_Hist_name)


for (var i=0; i<peaks.length; i++)
grwin.AddLine( peaks[i], 0, peaks[i], grwin.GetAxis().maxY, 2, 2, 2)


grwin.AddToBasket(iPM_Hist_name)
var pedestals_image_name = dir + "/" + iPM_Hist_name + ".png"
grwin.SaveImage( pedestals_image_name )
}


// Write 'pedestals' array to a file
// var currentDir_name = core.getCurrentDir( )
// var workDir_name = core.GetWorkDir( )
var pedestals_filename = dir + "/pedestals_file.dat"
core.print(pedestals_filename)
core.createFile( pedestals_filename, false )
core.saveArray( pedestals_filename, pedestals )
}
// End of CALCULATE Pedestals
```

B. trb3 scripts (commands to TRB3 board)

B.1 startup_TRB49_remotely.sh

```
#!/bin/bash

#set -x; export PS4='LINENO >'  # Uncomment to see what is being executed in each line. JM

cd /home/odroid/trbsoft/userscripts/trb49


# to run the script from the crontab:
if [[ -x ~/.bashrc ]]; then
    chmod a+x ~/.bashrc
fi
PS1='$ '; source ~/.bashrc


DAQ_TOOLS_PATH=~/trbsoft/daqtools
export PATH=$PATH:$DAQ_TOOLS_PATH
export PATH=$PATH:$DAQ_TOOLS_PATH/tools

export DAQOPSERVER=localhost:1

export TRB3_SERVER=trb049:26000                #change to the respective host name (the same than
/etc/hosts)
export TRBNETDPID=$(pgrep -f "trbnetd -i 1")
echo "- trbnetd pid: $TRBNETDPID"

if [[ -z "$TRBNETDPID" ]]
then
```

```
    ~/trbsoft/trbnettools/bin/trbnetd -i 1      #starts trbnetd which allows connection with the TRB3; trbnetd
-i 1 opens a trbnetd with th$
fi

./check_ping.pl                              #ping the TRB, the script must be changed accordingly with the
respective TRB name

echo "reset"
./trbreset_loop.pl                                  #the # of endpoints must be changed accordingly (5
endpoints for 1 TRB)
sleep 1;

#add Jan 2019 – because when called remotley this sctipt does not know the bin directory of the 'trbcmd'
application
DIRBINtrbcmd="/home/odroid/trbsoft/trbnettools/bin"

$DIRBINtrbcmd/trbcmd i 0xffff

$DIRBINtrbcmd/trbcmd s 0x6a000006e95d3528 0x00 0xC310 #ADC
$DIRBINtrbcmd/trbcmd s 0xef000006e95d3228 0x01 0xC311 #ADC
$DIRBINtrbcmd/trbcmd s 0xcf000006e95d1b28 0x02 0xC312 #not used
$DIRBINtrbcmd/trbcmd s 0xb1000006e9545028 0x03 0xC313 #not used

$DIRBINtrbcmd/trbcmd s 0x96000006e95d1828 0x05 0x8000 #CTS TRB49

$DIRBINtrbcmd/trbcmd i 0xffff

echo "GbE settings"
loadregisterdb.pl db/register_configgbe.db          #must be changed accordingly (ID of the
CTS_FPGA as defined above)
loadregisterdb.pl db/register_configgbe_ip.db       #must be changed accordingly (DAQ_MAC + IP +
Dest Port (used in EventBuilder_TRB186.xml))

#echo "TDC settings"
loadregisterdb.pl db/register_configtdc.db          #must be changed accordingly (ID of the TDC_FPGAs as
defined above)
#echo "TDC settings end"

#echo "pulser"
# pulser #0 to 10 kHz
# trbcmd w 0xc001 0xa156 0x4e20       # 0x4268 = 17000 -> 170000 ns = 0.170ms -> 5.88 kHz (1ADC
+ 1 TDC -> 87MB/s, 8% of deadtime)

#echo "pulser enable"
# pulser enable
# trbcmd setbit 0xc001 0xa101 0x2

# JM commented the three commands below, from Alberto/João Saraiva group
#echo "start trigger"
#trbcmd w 0xc001 0xa101 0xffff4000  #enable ch14 (coincidence module 0) trg_channel_mask:
edge=1111 1111 1111 1111, mask=0000 0000 0010 0000

#echo "start coincidences with detector 1&3"
#trbcmd w 0xc001 0xa13e 0x30005              #trg_coin_config0: coin_mask=0000 0101,
inhibit_mask=0000 0000; window=3 = 30 ns

#echo "limit cts_throttle"
#trbcmd w 0xc001 0xa00c 0x00000401  # cts_throttle: enable=true, stop=false, threshold=1


#Threshold Settings - Below some settings of Alberto/João Saraiva group
#~/trbsoft/daqtools/tools/dac_program.pl ~/trbsoft/userscripts/trb12/thresholds/configFile_a001_40mv
#~/trbsoft/daqtools/tools/dac_program.pl ~/trbsoft/userscripts/trb12/thresholds/configFile_a002_40mv
#~/trbsoft/daqtools/tools/dac_program.pl ~/trbsoft/userscripts/trb12/thresholds/configFile_a003_40mv

echo "Before ADCs setup"

#setup ADC addon
```

```
Samples=30
SamplesAfterTrigger=60
Threshold_Ch9=80
Threshold_Ch100=220

FPGA="0xC310"
~/trbsoft/daqtools/tools/adc.pl $FPGA init

$DIRBINtrbcmd/trbcmd w $FPGA 0xa010 $Samples                    # Buffer depth
$DIRBINtrbcmd/trbcmd w $FPGA 0xa011 $SamplesAfterTrigger     # Samples after trigger
$DIRBINtrbcmd/trbcmd w $FPGA 0xa012 1                           # Process blocks
$DIRBINtrbcmd/trbcmd w $FPGA 0xa013 $Threshold_Ch9          # Trigger offset, invert
$DIRBINtrbcmd/trbcmd w $FPGA 0xa014 0                           # Readout offset
$DIRBINtrbcmd/trbcmd w $FPGA 0xa015 1                           # Downsampling
$DIRBINtrbcmd/trbcmd w $FPGA 0xa016 8                           # Baseline
#trbcmd w $FPGA 0xa017 0x30000000                               # Trigger Enable ch31-00
#trbcmd w $FPGA 0xa018 0x0000                                  # Trigger Enable ch47-32
$DIRBINtrbcmd/trbcmd w $FPGA 0xa017 0x30000       #Trigger Enable ch31-00   JM: Hardware channels
16 and 17
$DIRBINtrbcmd/trbcmd w $FPGA 0xa018 0x0     #Trigger Enable ch47-32
$DIRBINtrbcmd/trbcmd w $FPGA 0xa01a 0x00000000   #Channel disable ch31-00, all channels except
ch0
$DIRBINtrbcmd/trbcmd w $FPGA 0xa01b 0x0000        #Channel disable ch47-32
$DIRBINtrbcmd/trbcmd w $FPGA 0xa01c 0                #Processing mode 0=BlockMode, 1=PSA, 2=CFD

# ignore CFD stuff
#trbcmd w $FPGA 0xa01d 0x340      #CFD delay is 3, CFD window 64=0x40

$DIRBINtrbcmd/trbcmd w $FPGA 0xa020 1         #Sum values
$DIRBINtrbcmd/trbcmd w $FPGA 0xa021 1         #Sum values
$DIRBINtrbcmd/trbcmd w $FPGA 0xa022 1         #Sum values
$DIRBINtrbcmd/trbcmd w $FPGA 0xa023 1         #Sum values
$DIRBINtrbcmd/trbcmd w $FPGA 0xa024 $Samples    #word count
$DIRBINtrbcmd/trbcmd w $FPGA 0xa025 0         #word count
$DIRBINtrbcmd/trbcmd w $FPGA 0xa026 0         #word count
$DIRBINtrbcmd/trbcmd w $FPGA 0xa027 0         #word count

$DIRBINtrbcmd/trbcmd w $FPGA 0xa000 0x100      #Reset Baseline

# Second ADC
FPGA1="0xC311"
~/trbsoft/daqtools/tools/adc.pl $FPGA1 init

$DIRBINtrbcmd/trbcmd w $FPGA1 0xa010 $Samples                    #Buffer depth
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa011 $SamplesAfterTrigger        #Samples after trigger
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa012 1                           #Process blocks
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa013 $Threshold_Ch100            #Trigger offset, invert
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa014 0         #Readout offset
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa015 1         #Downsampling
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa016 8         #Baseline
#trbcmd w $FPGA1 0xa017 0x30000000                  #Trigger Enable ch31-00
#trbcmd w $FPGA1 0xa018 0x0000                      #Trigger Enable ch47-32
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa017 0x30000     #Trigger Enable ch31-00  JM: Hardware channels
64 and 65
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa018 0x0         #Trigger Enable ch47-32
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa01a 0x00000000  #Channel disable ch31-00, all channels except
ch0
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa01b 0x0000     #Channel disable ch47-32
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa01c 0         #Processing mode 0=BlockMode, 1=PSA, 2=CFD
# ignore CFD stuff
#trbcmd w $FPGA 0xa01d 0x340      #CFD delay is 3, CFD window 64=0x40

$DIRBINtrbcmd/trbcmd w $FPGA1 0xa020 1         #Sum values
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa021 1         #Sum values
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa022 1         #Sum values
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa023 1         #Sum values
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa024 $Samples    #word count
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa025 0         #word count
```

```
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa026 0          #word count
$DIRBINtrbcmd/trbcmd w $FPGA1 0xa027 0          #word count

$DIRBINtrbcmd/trbcmd w $FPGA1 0xa000 0x100      #Reset Baseline
# Second ADC Done

echo "ADCs configured"
```

B.2 start_acquisition.sh

```
#!/bin/bash

DIRBINtrbcmd2="/home/odroid/trbsoft/trbnettools/bin"

#test
cd /home/odroid/trbsoft/userscripts/trb49

DAQ_TOOLS_PATH=~/trbsoft/daqtools
export PATH=$PATH:$DAQ_TOOLS_PATH
export PATH=$PATH:$DAQ_TOOLS_PATH/tools

export DAQOPSERVER=localhost:1

export TRB3_SERVER=trb049:26000                #change to the respective host name (the same than
/etc/hosts)
export TRBNETDPID=$(pgrep -f "trbnetd -i 1")
echo "- trbnetd pid: $TRBNETDPID"

#if [[ -z "$TRBNETDPID" ]]
#then
#    ~/trbsoft/trbnettools/bin/trbnetd -i 1      #starts trbnetd which allows connection with the TRB3;
trbnetd -i 1 opens a trbnetd with th$
#fi

THRESHOLD_CH9=$2 #use $2 to get the 1st argument
THRESHOLD_CH10=$3 #use $3 to get the 2nd argument

#Convert thresholds do hexadecimal
THRESHOLD_CH9_HEX=$(printf '%x\n' $THRESHOLD_CH9)
THRESHOLD_CH10_HEX=$(printf '%x\n' $THRESHOLD_CH10)

$DIRBINtrbcmd2/trbcmd w 0xc310 0xa013 0x000100$THRESHOLD_CH9_HEX
$DIRBINtrbcmd2/trbcmd w 0xc311 0xa013 0x000100$THRESHOLD_CH10_HEX
sleep 1s
$DIRBINtrbcmd2/trbcmd w 0xc310 0xa013 0x000000$THRESHOLD_CH9_HEX
$DIRBINtrbcmd2/trbcmd w 0xc311 0xa013 0x000000$THRESHOLD_CH10_HEX

sleep 3s # check if it is needed

$DIRBINtrbcmd2/trbcmd w 0x8000 0xa101 0xffff0600
echo "acquire on"
sleep $1s #20s #pass as argument the time of acquisition (10 seconds in this case. For 15 minutes use:
15m)
$DIRBINtrbcmd2/trbcmd w 0x8000 0xa101 0xffff0000
echo "acquire off"
```

C. Perl scripts (from TRB3 developers)

C.1 Script **check_ping.pl**

```
#!/usr/bin/perl

use warnings;
use strict;
```

```
use Parallel::ForkManager;
use Net::Ping;
use Getopt::Long;
use Data::Dumper;

my $power;
my $reboot;
my $help;

my $result = GetOptions (
                                "h|help"          => \$help,
                                "p|powercycle"    => \$power,
                                "r|reboot"    => \$reboot
                               );


my $map = {
0 => { trb => 49,  addr => "0x8000", sys => "CTS"},
#0 => { trb => 12,  addr => "0xc001", sys => "CTS"},
#1 => { trb =>  12, addr => "0xc002", sys => "GbE"},
# 2 => { trb => 113, addr => "0x8113", sys => "TOF"},
# 3 => { trb => 158, addr => "0x8158", sys => "TOF"},

};
my $MAX_PROCESSES=50;
my $pm = Parallel::ForkManager->new($MAX_PROCESSES);
my $maximal_reboot_counter = 4;
my $number_of_reboots_done = 0;

my $rh_unsuccessful = {};

$pm->run_on_finish(
                        sub { my ($pid, $exit_code, $ident,  $exit_signal, $core_dump,
$data_structure_reference) = @_;
                                #print "** $ident just got out of the pool ".
                                #    "with PID $pid and exit code: $exit_code\n";
                                #print Dumper ($pid, $exit_code, $ident,  $exit_signal, $core_dump,
$data_structure_reference);
                                if ($exit_code == 0) {
                                  $rh_unsuccessful->{$ident} = $$data_structure_reference;
                                }
                            }
                      );

#my $p = Net::Ping->new();

my $first_iteration = 1;

#print Dumper keys %$rh_unsuccessful;

while ( (($first_iteration == 1) || keys %$rh_unsuccessful) &&
          ($number_of_reboots_done < $maximal_reboot_counter) ) {
  #print Dumper $rh_unsuccessful;
  #print Dumper keys %$rh_unsuccessful;

  $rh_unsuccessful = {};
  $first_iteration = 0;
  foreach my $ct (keys %$map) {
    my $success = 0;
    #my $num = sprintf "%3.3d", $ct;
    my $trbnum= $map->{$ct}->{trb};
    my $num = sprintf "%3.3d", $trbnum;
    my $host= "trb" . $num;
    #my $host= "trb" . $num ."b";
    my $system = $map->{$ct}->{sys};
    #print "192.168.0.$ct   $host.gsi.de $host\n";
    #my $r = $p->ping($host,1);
    my $c= "ping -W1 -c2 $host";
```

```
    #    my $sysnum = sprintf "0x80%.2x", $ct;
    my $sysnum = $map->{$ct}->{addr};
    #$sysnum = "0x7999" if $ct == -1;

    my $pid = $pm->start("$sysnum") and next;

    #my $p = Net::Ping->new("udp", 1);
    #my $r = $p->ping("192.168.0.56");
    #$p->close();
    #print "result: $r\n";

    my $r = qx($c);
    #printf "$sysnum, system: %-8s, trb: $host ", $system;
    printf "$sysnum  $host  %-8s ", $system;
    if (grep /64 bytes/, $r) {
      print "is alive.\n";
      $success = $trbnum;
    } else {
      print "is not alive.\n";
    }

    my $str = "jhhj";
    $pm->finish($success, \$host); # Terminates the child process
  }

  $pm->wait_all_children;

  #$rh_unsuccessful = { "0x8007"=>"hh", "0x8001"=>"jjhj"} ;

  if ($reboot && ($number_of_reboots_done < $maximal_reboot_counter) && keys %$rh_unsuccessful) {
    #print Dumper $rh_unsuccessful;
    print "have to reboot FPGAs, first make a reset and reassign the addresses.\n";
    my $cmd = 'trbcmd reset; sleep 2;  ~/trbsoft/daqtools/merge_serial_address.pl
$DAQ_TOOLS_PATH/base/serials_trb3.db $USER_DIR/db/addresses_trb3.db';
    qx($cmd);
    sleep 3;
    # test trbnet:
    my $error_str = "ERROR: read_uid failed: Termination Status Error";
    $cmd = "trbcmd i 0xffff 2>&1";
    my $r = qx($cmd);
    if ($r=~/$error_str/) {
      print "could not access trbnet, so have to reboot all FPGAs.\n";
      $rh_unsuccessful = { "0xffff"=>"all"} ;
    }

    my $ctsnum = $map->{0}->{addr};
    if ($rh_unsuccessful->{ $ctsnum } || (scalar keys %$rh_unsuccessful) > 5) {
      print "many TRBs (or CTS: ". $ctsnum . ") are not alive, so let us make a reload of all FPGAs.\n";
      $rh_unsuccessful = { "0xffff"=>"all"} ;
    }

    foreach my $cur (keys %$rh_unsuccessful) {
      my $host = $rh_unsuccessful->{$cur};
      #my $cmd = "trbcmd reload " . $cur;
      $cmd = "trbcmd reload $cur";
      print "rebooting: $cur\n";
      #print "$cmd\n";
      qx($cmd);
      #print "number of reboots done: $number_of_reboots_done\n";
    }
    print "wait 9 seconds\n";
    sleep 9;
    $number_of_reboots_done++;
  }

}
```

```
exit 1 if(scalar keys %$rh_unsuccessful > 0);
#$p→close();
```

C.2 Script **trbreset_loop.pl**

```
#!/usr/bin/perl

use strict;
use warnings;

### Change THIS! endpoint == FPGA; (1 TRB = 5 FPGAs) -> JPS
my $required_endpoints = 5;

my $max_counter = 10;
my $counter = 0;
my $number = 0;


#js while (($number != 65) || ($counter > $max_counter)) {
#while (($number < $required_endpoints) || ($counter > $max_counter)) {
while (($number < $required_endpoints) ) {
    my $c; my $res;

    $counter++;
    $c= "/home/odroid/trbsoft/trbnettools/bin/trbcmd reset";
   # $c= "trbcmd reset";
    $res = qx($c);

    $c = "/home/odroid/trbsoft/trbnettools/bin/trbcmd i 0xffff | wc -l";
   # $c = "trbcmd i 0xffff | wc -l";
    $res = qx($c),
    print "- number of trb endpoints in the system: $res";
    ($number) = $res =~ /(\d+)/;
    print "number of enpoints is not equal to the required enpoints $required_endpoints, so try next
reset!\n" if ($number !=$required_endpoints);
}
```