Terascale detector workshop - Hands-On course: Exercises

1. Getting started

Connection to the mini-pcs
For the ten setups, there are five mini-PCs that connect to the FPGA boards.
The FPGA design software as well as a small GUI for reading out the FPGA are also installed on the small boxes.
Connection is made via ssh. X11 forwarding should be set up to show the waveforms and GUI for the FPGA development.

Connect to the Wi-Fi access point called “Terascale_HandsOn”,
password is the date of the event: 28022023
Note: This is an internal network without internet access.

Each mini-pc has a static IP assigned: 20.0.0.100-20.0.0.104
We will provide a sheet with hardware assignments to the individual groups, the IP addresses are also labeled on the PC.
User name for ssh is “user” ; password is “tera”

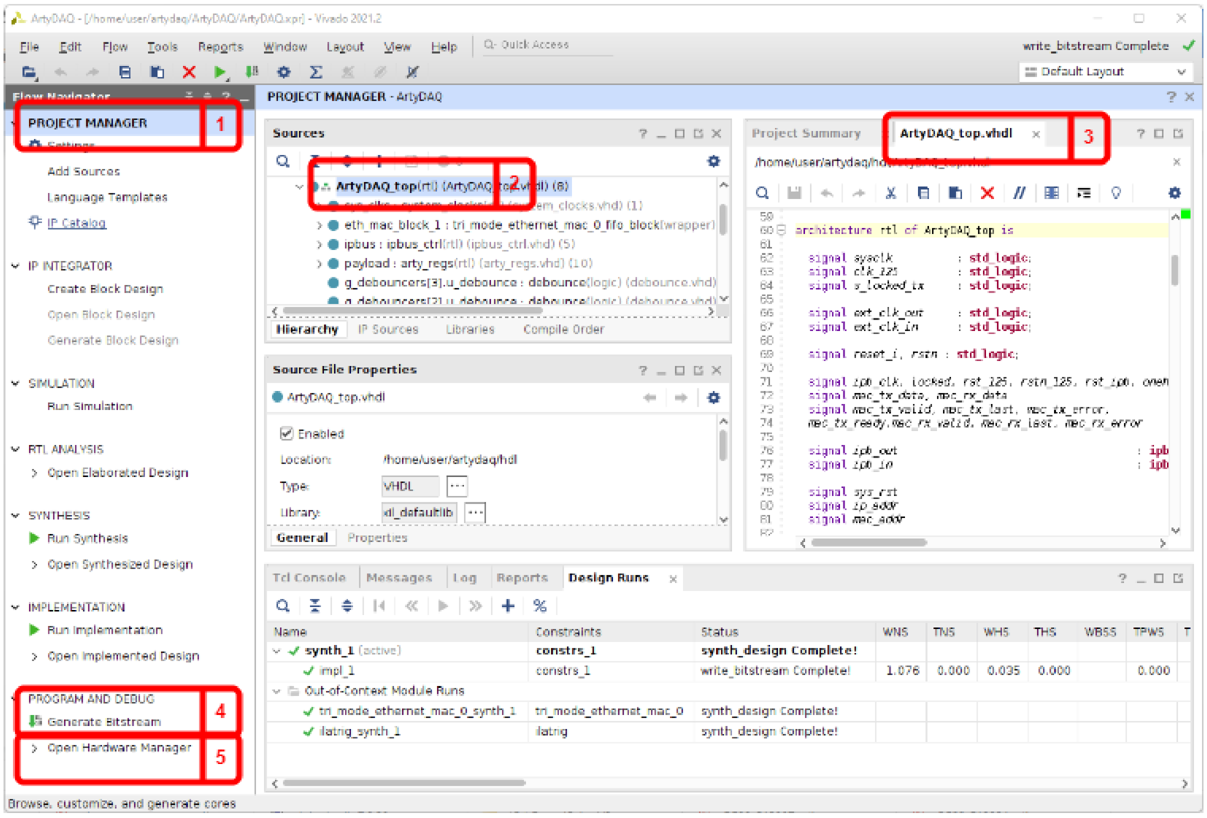
The XILINX Vivado GUI
Use ssh to connect to the mini-pc assigned to your group and go to the working directory.
Set up the environment sourcing a setup script. Start vivado and open the project:

```
ssh -Y user@20.0.0.10X
cd Terascale_GroupX/
source scripts/setup.sh
start_vivado
```

start_vivado is an alias we set up to automatically open the project and select the right FPGA.
The vivado GUI has the benefits of online syntax checking.
If you prefer to use a command line editor and no graphical interface, refer to the section “Generating a bitstream (command line)” at the end of the document.

Source files can be found in ~/Terascale_GroupX/hdl/ If you are getting stuck, you will find solutions in ~/Terascale_GroupX/hdl/solutions. The solutions are incremental, so you can use a solution to continue with the next exercise or to have a “clean start”.

In Vivado, clicking on the project manager **(1)** in the Flow navigator pane on the left, a tree representation of the current design will be opened. Double clicking on the top-level design (ArtyDAQ_top, **(2)**) opens the respective **vhdl source file** **(3)**.

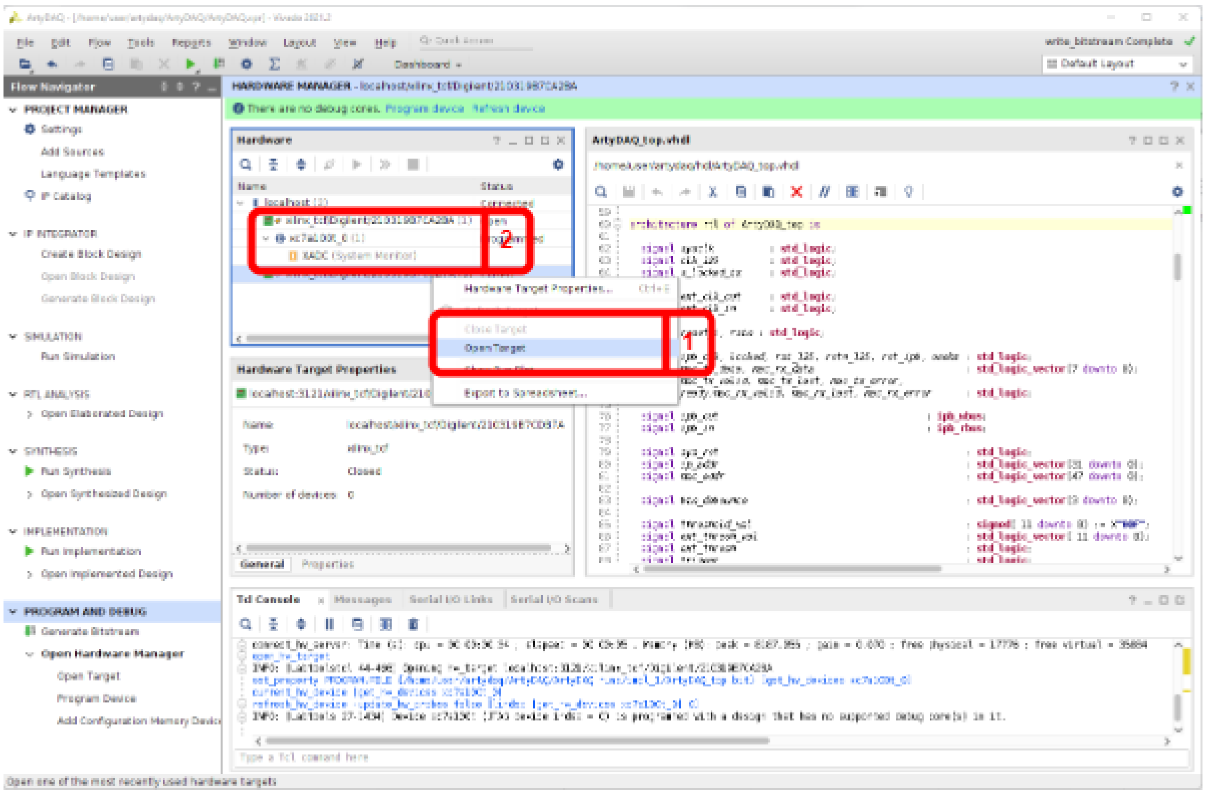


```
architecture rtl of ArtyDAQ_top is

  signal sysclk          : std_logic;
  signal clk_125         : std_logic;
  signal s_locked_tx     : std_logic;

  signal ext_clk_out     : std_logic;
  signal ext_clk_in      : std_logic;

  signal reset_i, rstn : std_logic;

  signal ipb_clk, locked, rst_125, rstn_125, rst_ipb, onen
  signal mac_tx_data, mac_rx_data
  signal mac_tx_valid, mac_tx_last, mac_tx_error,
  mac_tx_ready,mac_rx_valid, mac_rx_last, mac_rx_error

  signal ipb_out                                          : ipb
  signal ipb_in                                           : ipb

  signal sys_rst
  signal ip_addr
  signal mac_addr
```

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	T
✓ **synth_1** (active)	**constrs_1**	**synth_design Complete!**							
✓ impl_1	constrs_1	write_bitstream Complete!	1.076	0.000	0.035	0.000		0.000	
Out-of-Context Module Runs									
✓ tri_mode_ethernet_mac_0_synth_1	tri_mode_ethernet_mac_0	synth_design Complete!							
✓ ilatrig_synth_1	ilatrig	synth_design Complete!							

To **build** the firmware after modifying the hdl code, click on “Generate Bitstream” **(4)** on the bottom of the navigator panel. A full build will take about 2-4 minutes.
An initial version of the firmware has already been generated, building the firmware at this point is not needed.

To **upload** the firmware to the FPGA, open the “Hardware Manager” **(5)**, clicking at the respective line at the bottom left side of the vivado window. The connection to the FPGA should already be set up. If not, click on “Open target” -> “Auto Connect”.



```
connect_hw_server: Time (s): cpu = [illegible] ; elapsed = [illegible] . Memory (MB): peak = [illegible] ; gain = [illegible] ; free physical = 17776 ; free virtual = 35884
open_hw_target
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/[illegible]
set_property PROGRAM.FILE {/home/user/artydaq/ArtyDAQ/ArtyDAQ.runs/impl_1/ArtyDAQ_top.bit} [get_hw_devices xc7a100t_0]
current_hw_device [get_hw_devices xc7a100t_0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices xc7a100t_0] 0]
INFO: [Labtools 27-1434] Device xc7a100t (JTAG device index = 0) is programmed with a design that has no supported debug core(s) in it.
```

On each PC, there are two FPGA boards connected and visible in the hardware manager. Ensure the board of your group is opened (comparing its ID). If not, the FPGA board can be selected by right clicking on the respective board and clicking “Open” and accepting the following dialog **(1)**. To program the FPGA, select the small chip symbol with xc7a, **(2)**, select “Program device” in the drop down menu and click “Program” in the following dialog.

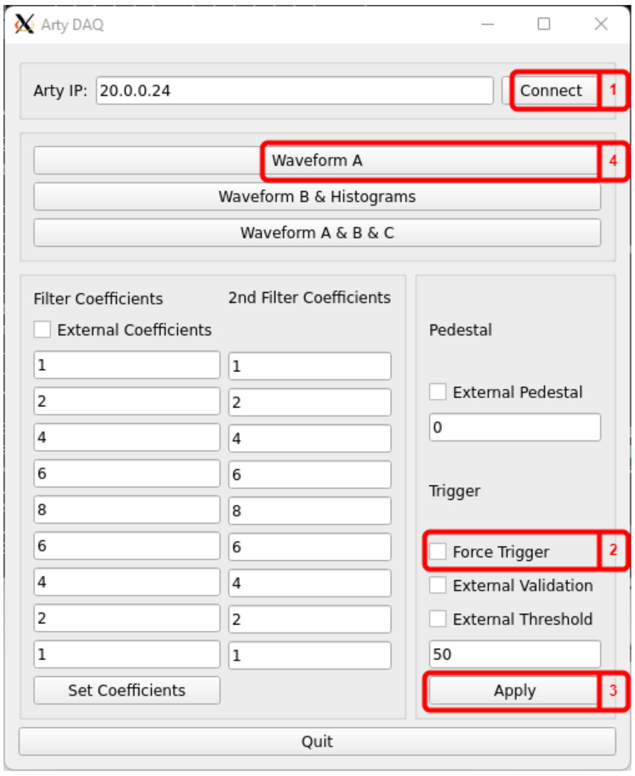
During firmware upload, the LEDs on the FPGA will turn off. After the firmware is uploaded, a LED7 on the FPGA board should start blinking again.

GUI for reading out the FPGA

In order to conveniently show and interpret the data from the FPGA, we prepared a small python-based GUI which allows to control registers in the FPGA and read back waveforms. The communication is established using the IPbus protocol (https://ipbus.web.cern.ch/).

It is time to open the GUI to read out the FPGA registers and start looking at the waveforms. Open a new terminal and connect via ssh to the mini-pc. As before, go to the working directory and source the setup script. The setup script places all required paths in your shell environment. Now start the readout GUI:

```
ssh -Y user@20.0.0.10X
cd Terascale_GroupX/
source scripts/setup.sh
ArtyGui.py
```



Arty IP: 20.0.0.24

Filter Coefficients	2nd Filter Coefficients
1	1
2	2
4	4
6	6
8	8
6	6
4	4
2	2
1	1

Pedestal: 0

External Threshold: 50

A small control window will appear. Start by clicking “Connect” **(1)**, then set the “Force Trigger” **(2)** checkbox and press Apply **(3)**. After that proceed to click on “Waveform A” **(4)**.
A new window appears showing the (currently empty) Waveform of the last triggered pulse.
At this point, the waveform will be constant 0 and continuously triggering.
We are going to change this in the next steps and exercises.
Close the GUI by clicking “Quit”.

2. <u>Sampling of ADC data</u>

In the firmware, the ADC data is not registered by the FPGA, so the waveform transferred to the PC is empty.
As a first firmware change, open the top level entity (ArtyDAQ_top) in the vivado Project Manager. In the process **"p_sample_adc"** starting at line 204, implement the necessary signal assignments to sample the adc data at every rising edge of the clock and generate a pipeline of 32 ADC values ("Waveform"):

- The data is stored in the array *adc_raw*, holding the last 32 samples from the ADC.
- At each rising edge of the clock, the ADC input data value *adc_data12* should be registered in the first (Index 0) sample of the output array.
- At each rising edge, the new state of the index *adc_raw*(N+1) should be assigned the current value of *adc_raw*(N) where N is an index [0..30]
  You can use a *for loop* in the process to do the assignment of the 31 old samples.

Generate a bitstream of the new design and upload it to the FPGA.
The readout GUI will stop if not closed when programming the FPGA.
After restarting the ArtyGUI, connect again, check "Force Trigger", press "Apply" and look at the waveform. You should be able to see a continuously updating waveform and some noise, eventually some noise triggers from the Silicon Photomultiplier and also larger pulses from cosmics.
Quit the ArtyGUI.

3. <u>Leading edge discrimination</u>

Currently the trigger is always set to one, i.e. waveforms are stored instantaneously whenever the FPGA is not busy. Now we also want a low level trigger based on the raw ADC data by implementing a leading edge discrimination.
We want to later use other sources than the raw ADC data, thus a different signal *trigger_val* is used for the comparison.
At this point, *trigger_val* = *adc_raw*(24). We choose an older sample (24) to allow the full pulse to fit into the waveform. To do this, select the right statement in the vhdl source around line 200 below the header "*Trigger source selection*" and comment out the old trigger assignment.

In the process "**p_zero_suppression"** around line 278, assign the trigger signal following below principles:

- if either the condition *reset* = '1' or the condition *trigger_clear*='1' is true, assign the *trigger* signal a value of '0' in the next clock cycle.
- If the value of *trigger_val* is greater than the value of a signal *threshold_val* - both interpreted as signed integer numbers - assign the *trigger* a value of '1' in the next clock cycle.
- To enable the above comparison with the correct representation, use the statement *to_integer(some_signal)* > *to_integer(some_other_signal)* for the comparison.

Generate a bitstream of the new design and upload it to the FPGA.
After restarting the ArtyGUI, connect again and look at the waveform.
Now, we can try different values of the external threshold and see the effect of the discriminator. For that, you need to check the "External Threshold" and set different values by clicking "Apply" (keep in mind the pedestal values are negative).

4. <u>Pedestal Subtraction</u>

Implement the subtraction of a pedestal value from the ADC data.
This will later enable us to multiply the ADC samples with filter coefficients without adding a larger offset that would limit the dynamic range of the filter output.
In the "*Trigger source selection*" please select the right trigger source and comment out the old trigger assignment.

The pedestal subtraction can be done most efficiently directly after sampling the ADC data.

- In the VHDL process ***"*p_sample_adc*"***, add a statement that will assign the signal *adc_noped(0)* the raw adc input data (*adc_data32*) minus a pedestal (*pedestal_selected).*
- Generate a "Waveform" for this, by assigning a time series adc_noped(0..31) of pedestal subtracted ADC samples as in the first assignment.

Generate a bitstream of the new design and upload it to the FPGA.
The readout GUI will stop if not closed when programming the FPGA.
After restarting the ArtyGUI, connect again and look at the waveform by clicking (Waveform A&B&C). You will see three waveforms. You can now adjust the pedestal to be subtracted by choosing an appropriate value in the "pedestal" Field of the GUI. Make sure the "Pedestal Selection" is set to "0" and click apply.
For waveform B, you can adjust the scale, using the "four arrows button" in the top bar.
You can set the external pedestal to different values. Adjust the external threshold in order to see the larger pulses.

5. Pedestal Substraction 2 - IIR Filter

Instead of setting a fixed offset to be subtracted, it is also possible to calculate an average pedestal value and subtract this value from the raw ADC data.
Calculating the Pedestal as an ADC average corresponds to implementing a low pass filter. If this value is then subtracted from the ADC, the pedestal subtracted output corresponds to a high pass Filter, i.e. also low frequency noises are filtered out.

As discussed in the presentation, to calculate an average over many samples, FIR filters have their limitations due to the limited historical information they keep. Therefore we will use an IIR filter to calculate an average pedestal.

- In the process *"***p_pede_ma***"*, implement a lossy integrator as a 1-Tap IIR filter to calculate a new average pedestal in each clock cycle. Use 64 as a division factor.
- Generate again the bitstream and configure the FPGA.
- Check the output of the pedestal subtraction: In the GUI, select "1" as the pedestal source and compare the filtered waveform with the raw ADC data.
- Observe the pulse, especially also at the falling edge and baseline.
- Change the integration time constants by changing the division factor in the firmware. What happens at the falling edge of the pulse as the integration times get longer or shorter?

6. <u>Low pass using a finite impulse response filter</u>

To get a better estimation of the pulse energy, we want to further filter the noise by applying a filter with low pass frequency behavior.
Implement a FIR filter with an order of nine, using the previously pedestal-subtracted ADC values.

- Implement the filter output *adc_filt[0]* using the filter coefficients *coeff[n]* and the input data *adc_noped[n] with n=[0..8].*
- Use the transposed implementation with 2 summands per step.
  Intermediate results are stored in the array *filter_taps[n]*
- In the HDL description, you can again use a for-loop to implement the summation over the products of the filter coefficients and input samples.
- As we want again a time series of the filter output to be able to compare waveforms, also assign 31 additional "historic samples" *adc_flt[1..31]* as done before.

7. Measurement of pulse energy & Cross validation

Summing over the Bins of the filtered waveform (*adc_filt[n]*), define an energy *filt_e_out.* In the GUI, select Waveform B & Histogram. For every pulse, the result of the scalar energy output is filled into the histogram (the window is only updated every 30 pulses). There is a second histogram filled less frequently. This histogram is only filled when the second system of your partner group also saw a trigger at the same time, with a window of ten clock cycles (i.e. 250ns). Thus, with reasonable threshold settings, this histogram only shows events where a particle passed both scintillators. We will discuss the implementation of this function, you do not need to implement it.

Additional information

Generating a bitstream & FPGA programming from command line.
Use ssh to connect to the mini-pc assigned to your group and go to the working directory. Set up the environment sourcing a setup script. Then we are ready to build the firmware and eventually upload it to the FPGA:

```
cd Terascale_GroupX/
source scripts/setup.sh

# build the firmware
# if there is a error in the vhdl file the build will stop quite fast.
# a full build can take 2-3 minutes to finish
vivado -mode batch -source scripts/generate_bitstream.tcl ArtyDAQ/ArtyDAQ.xpr

# once building the bitfile succeeds, upload it to the FPGA
vivado -mode batch -source scripts/program_fpga.tcl ArtyDAQ/ArtyDAQ.xpr
```

Synchronization of the combined systems (*)
In order to have a common timestamp, the two systems need to be synchronized
This is done by
1) sending the clock signal from one board to the other and
2) Resetting the time stamp counter synchronously

To synchronize the counters:
1) Press and hold BTN1 on both boards. The firmware is in synchronization mode.
2) Press BTN0 on one of the boards. This will reset the counter on both boards with a well defined time difference.
3) Release BTN1 on both boards. Both FPGAs are now ready again to accept triggers from the ADC waveform processing.

As a first verification of the synchronization, observe the last LED on the FPGA board. If they are blinking with the same phase, the synchronization was successful.

Windows, SSH and X11…
- We recommend using powershell instead of wsl2, we found that the network address resolution works without delay in this case.
  WSL sometimes has longer timeouts during connection via SSH.
- If using xming, use xlaunch or configure it disabling access control
- Before connecting via SSH, set up your display in powershell:
  ```
  PS C:\Users\konra> $env:DISPLAY="127.0.0.1:0.0"
  PS C:\Users\konra> ssh user@20.0.0.100 -Y
  ```

Assignment of Hardware

Group	PC	FPGA#	Switches (3,0)	IP	ADC	Box	HV Supply	Bias voltage
1	103	A2BA	0,0,0,0	20.0.0.16		3	Keithley	53V
2	103	DB7A	0,0,0,1	20.0.0.17		4	^^	^^
3	100	F6BA	0,0,1,0	20.0.0.18		5	Keithley	53V
4	100	A21A	0,0,1,1	20.0.0.19		6	^^	^^
5	102	A16A	0,1,0,0	20.0.0.20		8	Agilent	65V
6	102	A4AA	0,1,0,1	20.0.0.21		7	Agilent	70V
7	104	A22A	0,1,1,0	20.0.0.22		1	Keithley	51.5V
8	104	DB6A	0,1,1,1	20.0.0.23		2	^^	51.5V
9	101	99EA	1,0,0,0	20.0.0.24		9	Agilent	73V
10	101	CFAA	1,0,0,1	20.0.0.25		10	Agilent	71.5

Layout of the room:

	Blackboard		
		Group 2	Group 1
		Group 4	Group 3
		Group 6	Group 5
		Group 8	Group 7
		Group 10	Group 9