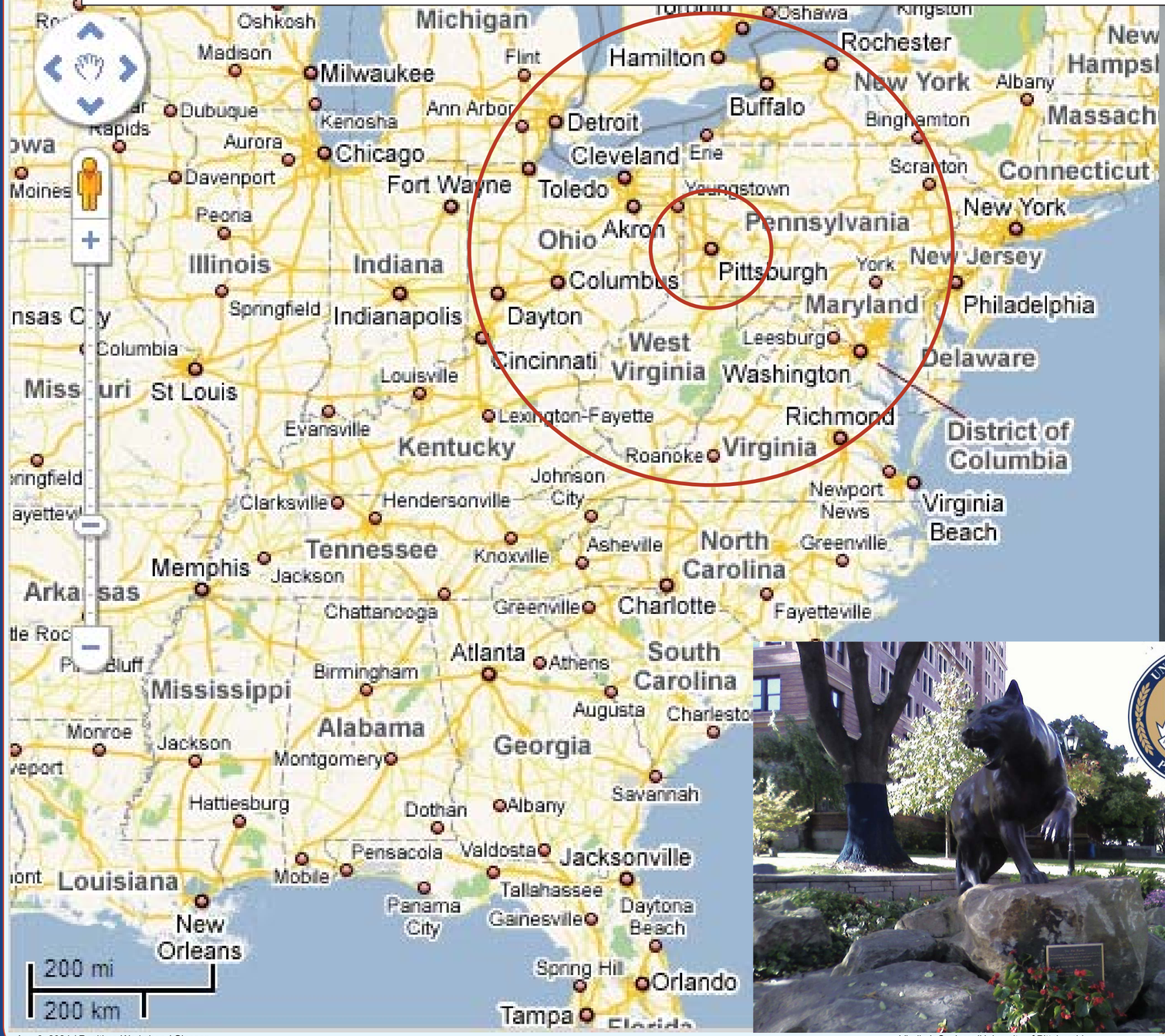
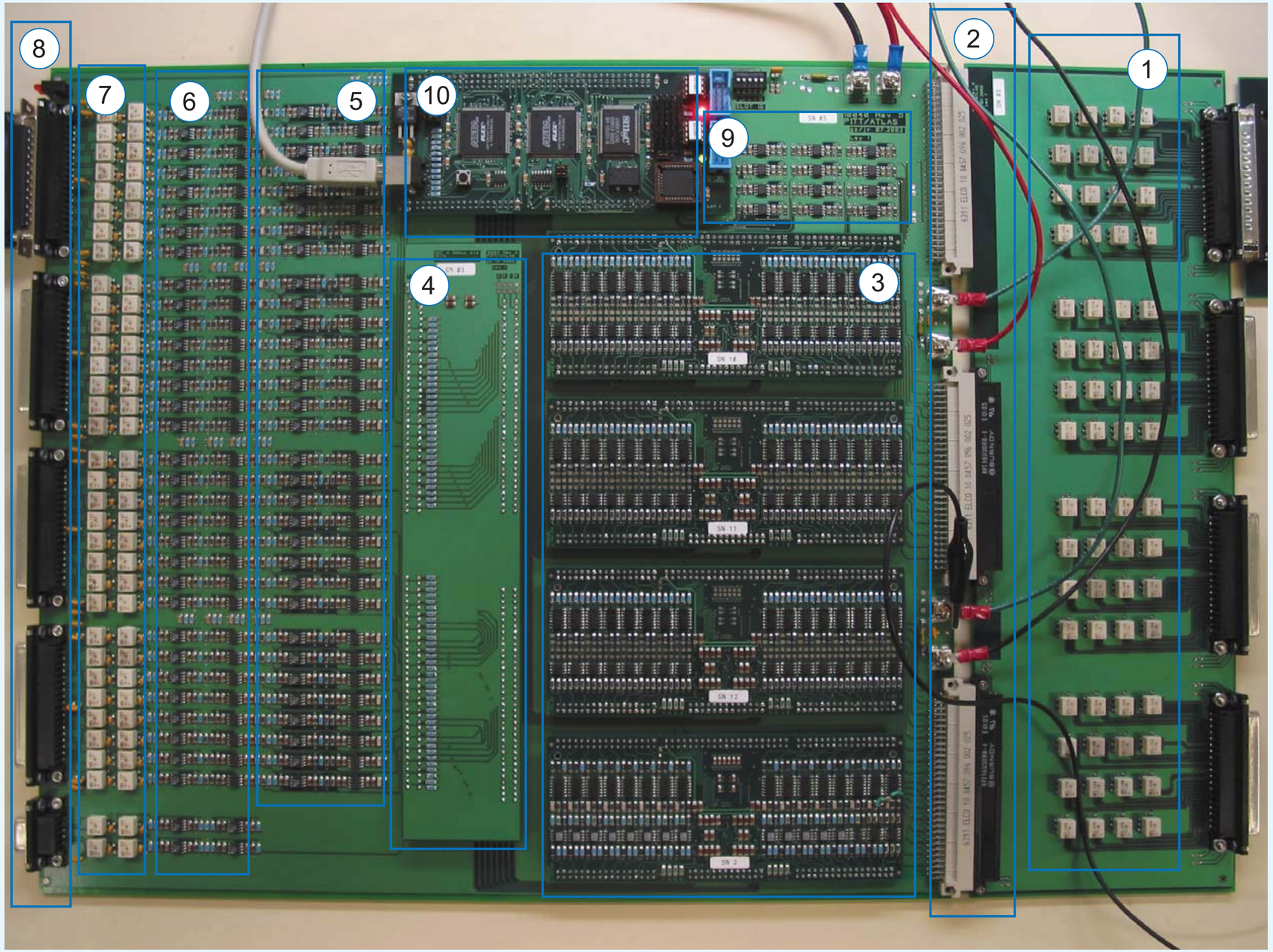
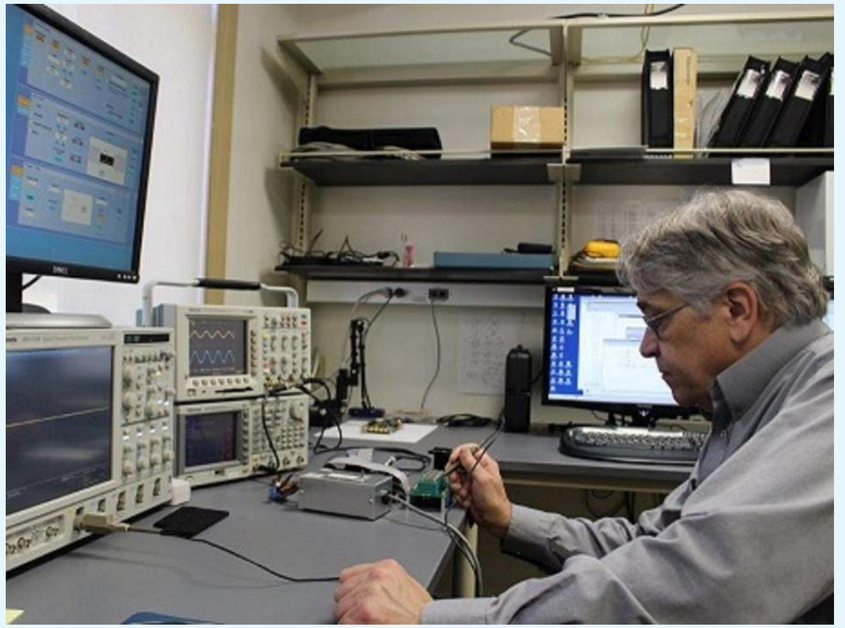
Our (more recent) involvement with XILINX/AMD FPGAs and SoCs at the University of Pittsburgh



The past: analog interface between ATLAS calorimeters and L1 trigger / FPGA work by engineers



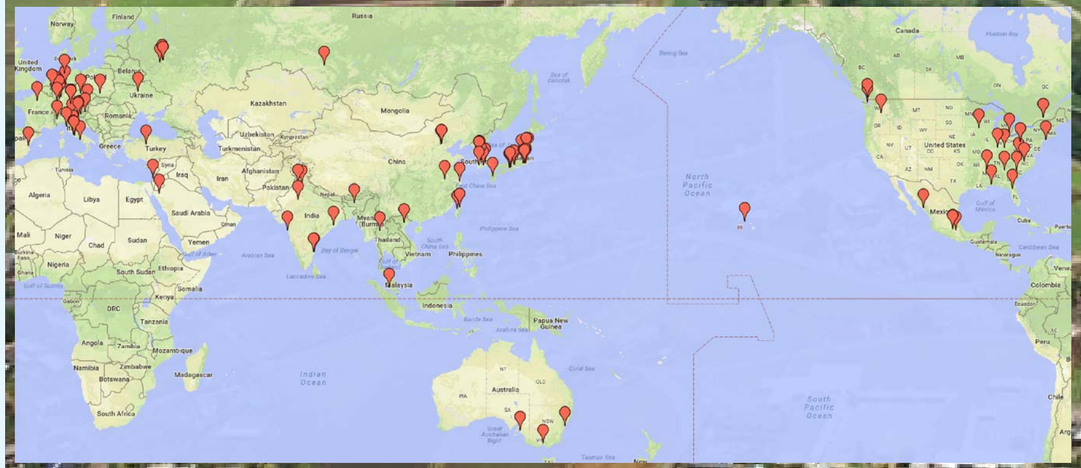


Belle II @ Super-KEKB

Intensity frontier B-factory experiment, successor to Belle @ KEKB (1999-2010)

7 GeV e^-, 4 GeV e^+

E_{CM} = 10.58 GeV



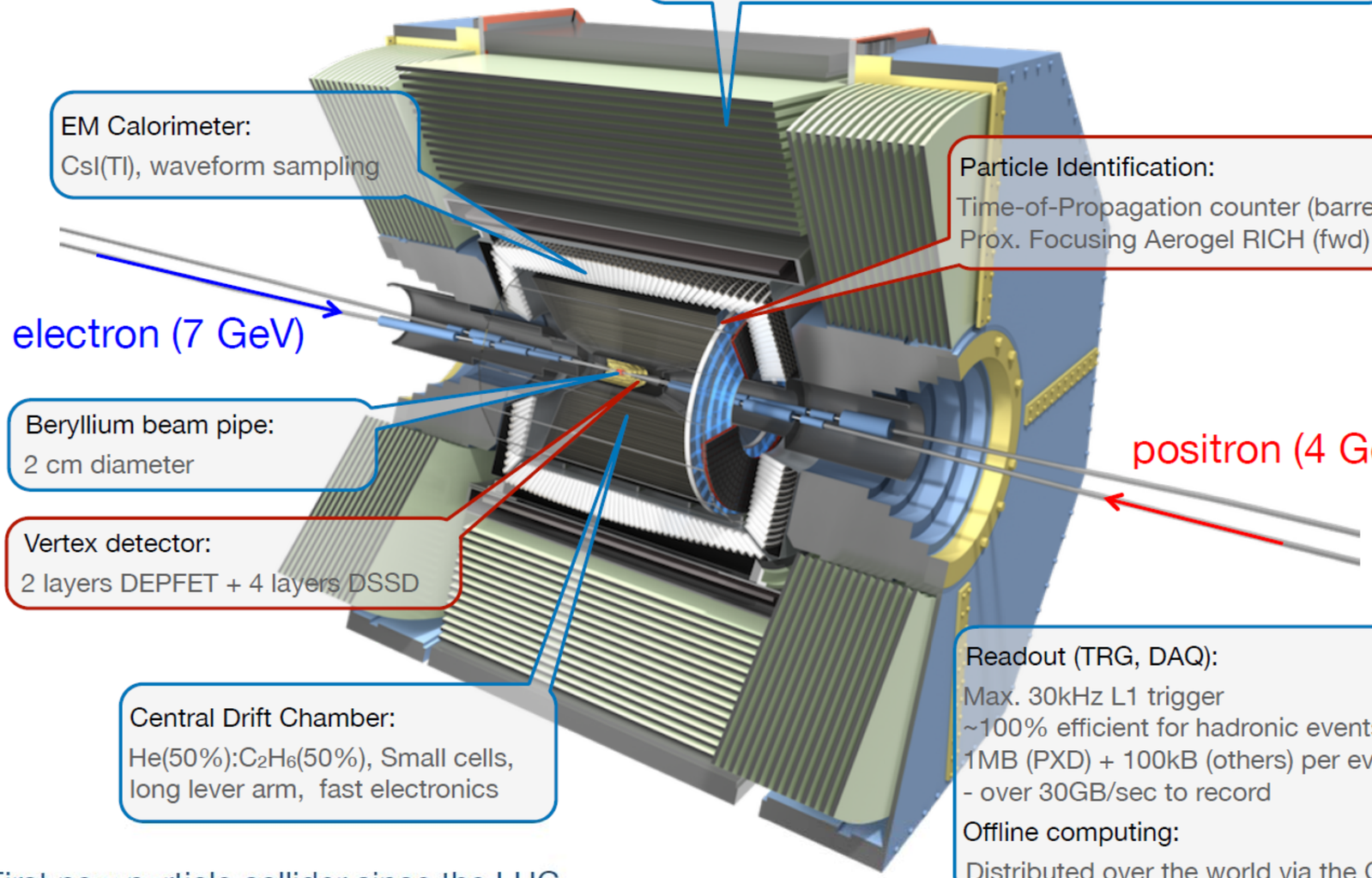
A collaboration of ~800 scientists from 25 countries

The Belle II detector

K_L and muon detector:
Resistive Plate Counter (barrel outer layers)
Scintillator + WLSF + MPPC (end-caps, inner 2 barrel layers)

EM Calorimeter:
CsI(Tl), waveform sampling

Particle Identification:
Time-of-Propagation counter (barrel)
Prox. Focusing Aerogel RICH (fwd)



Beryllium beam pipe:
2 cm diameter

Vertex detector:
2 layers DEPFET + 4 layers DSSD

Central Drift Chamber:
He(50%):C_2H_6(50%), Small cells, long lever arm, fast electronics

Readout (TRG, DAQ):
Max. 30kHz L1 trigger
~100% efficient for hadronic events.
1MB (PXD) + 100kB (others) per event
- over 30GB/sec to record

Offline computing:
Distributed over the world via the GRID

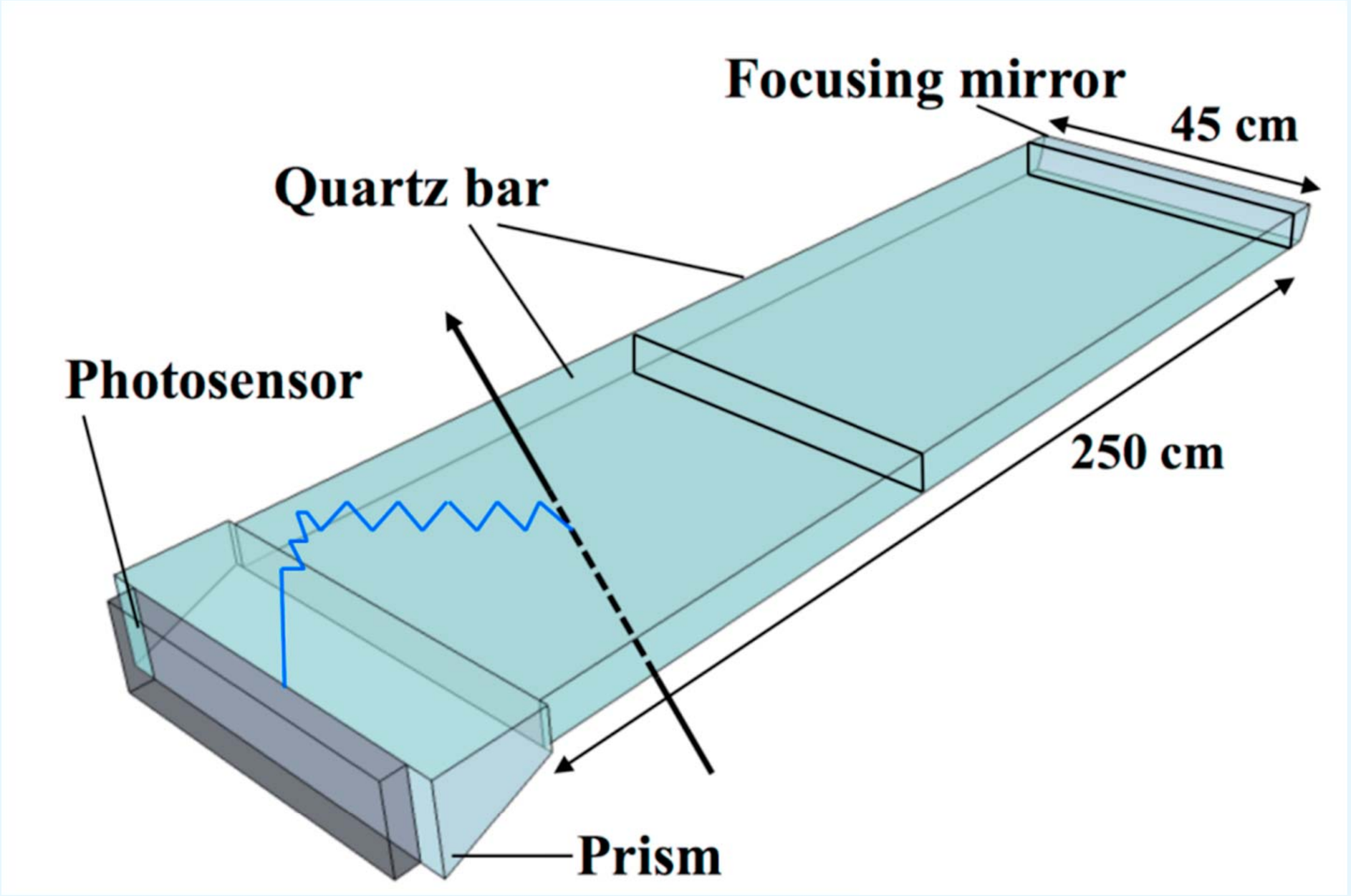
First new particle collider since the LHC
(intensity rather than energy frontier; e^+e^- rather than pp)

arXiv:1011.0352 [physics.ins-det]

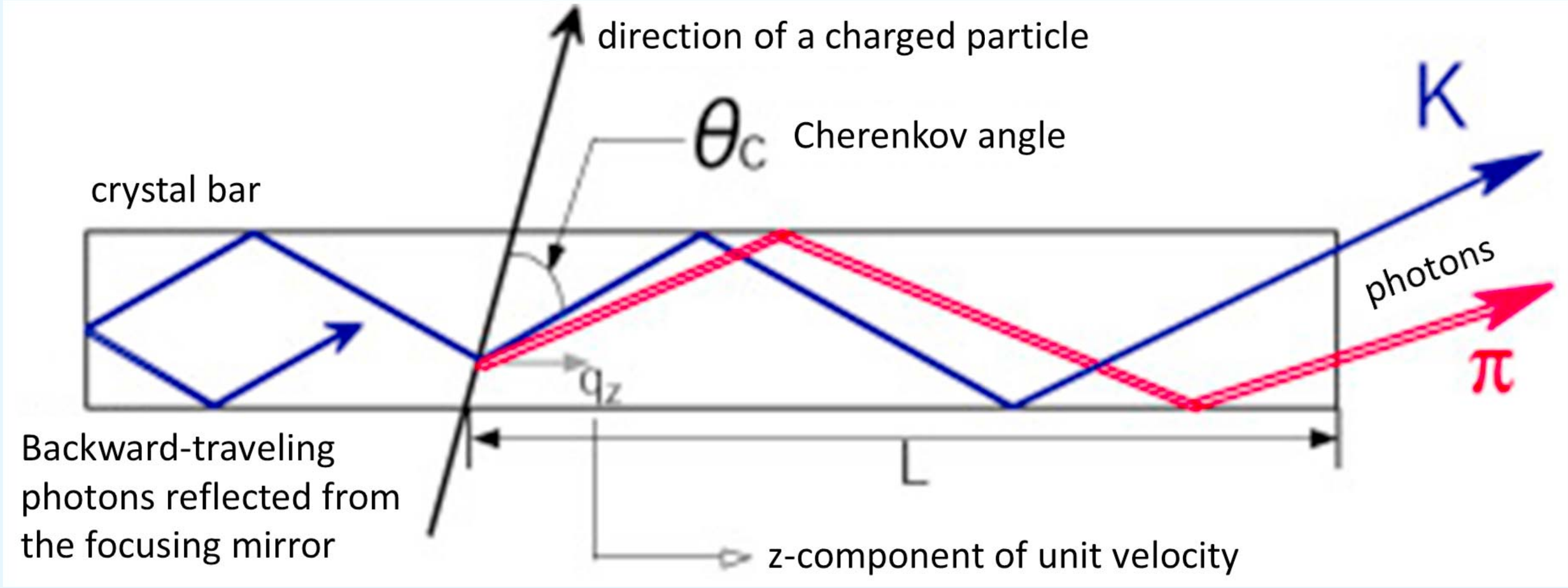
~8m wide
~8m tall

~1,400 tons

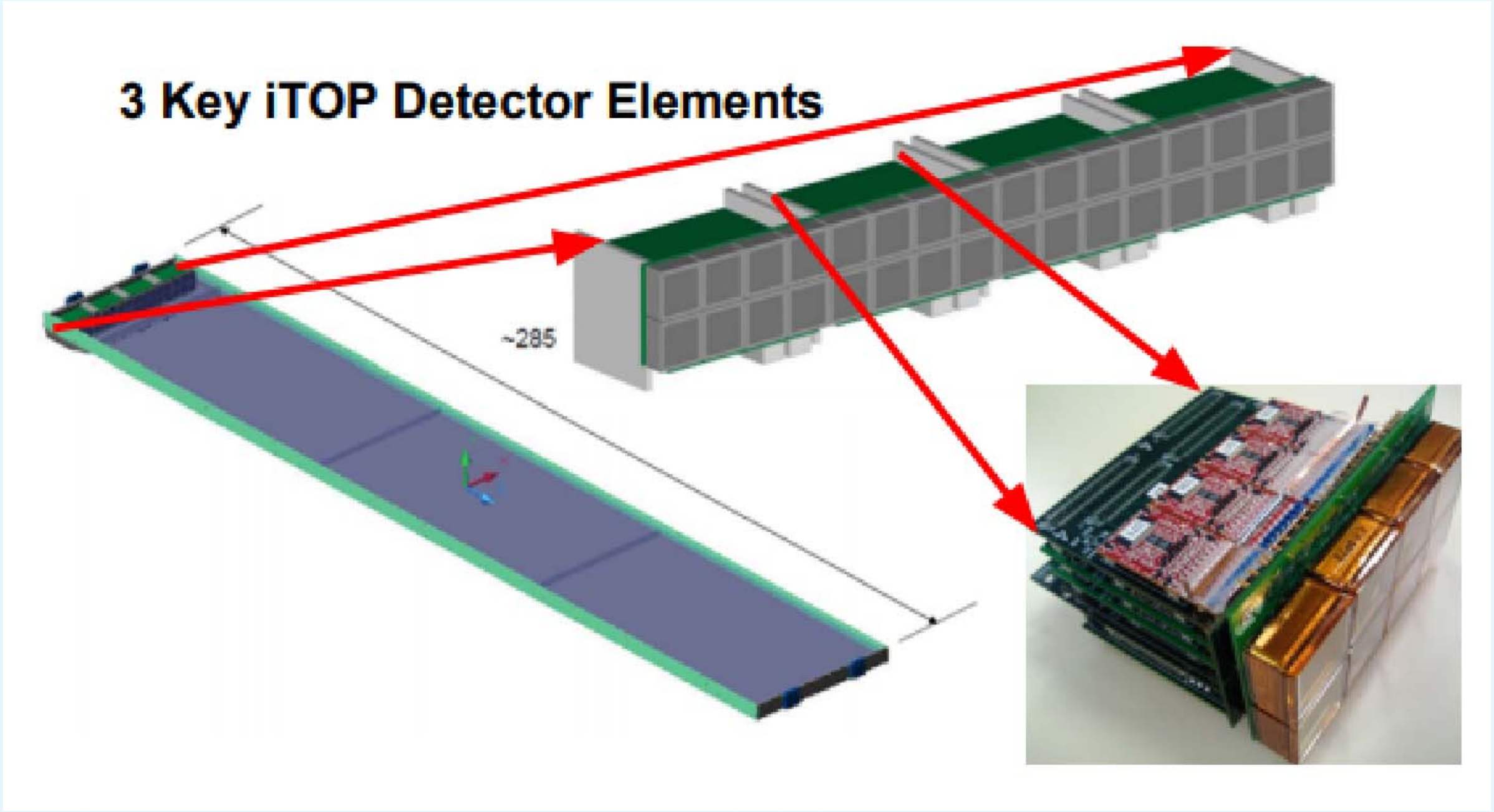
More recent work: TOP-based evaluation of beam collision time for L1 trigger / TOP FEE QC/QA



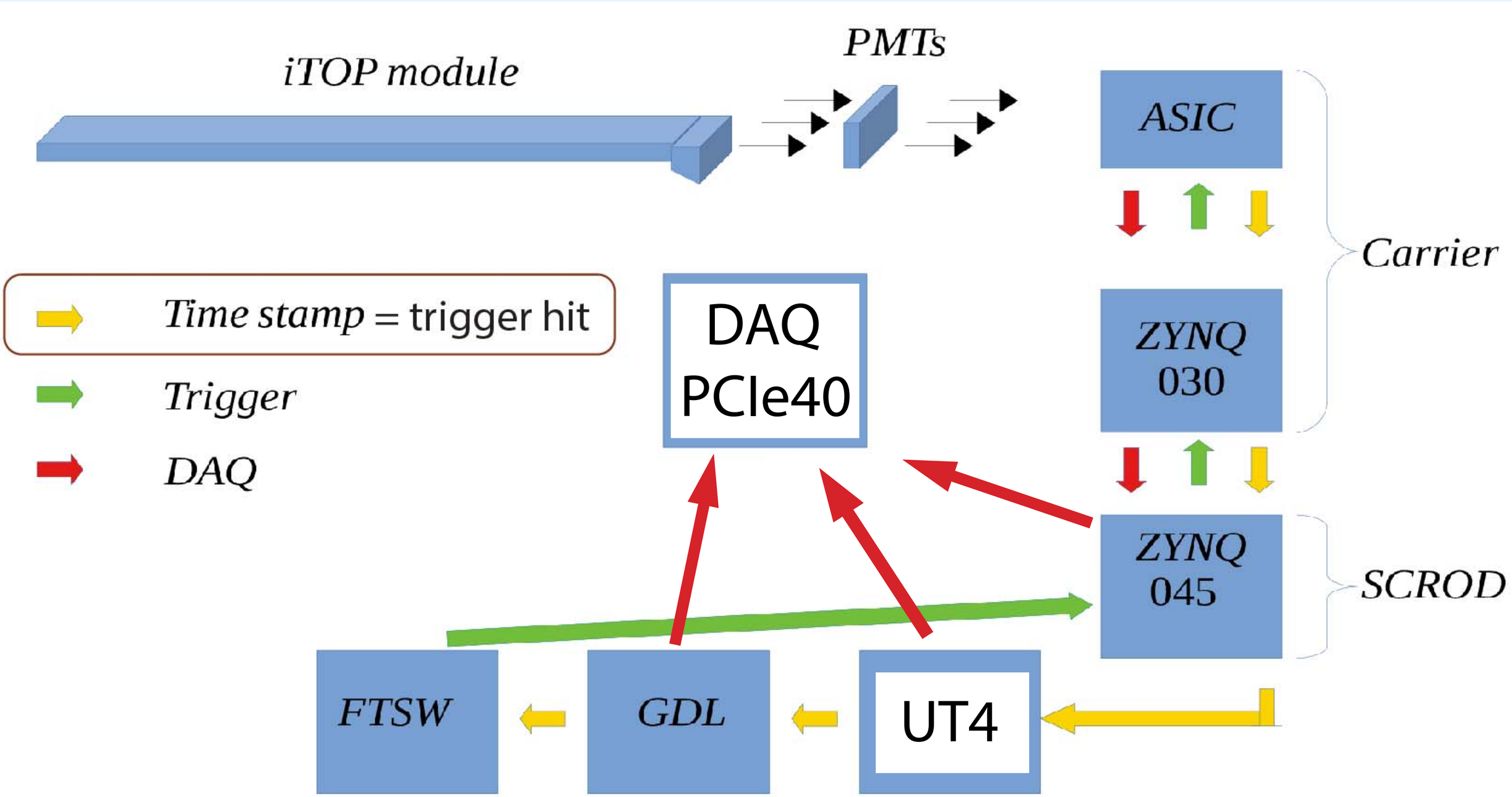
More recent work: TOP-based evaluation of beam collision time for L1 trigger / TOP FEE QC/QA



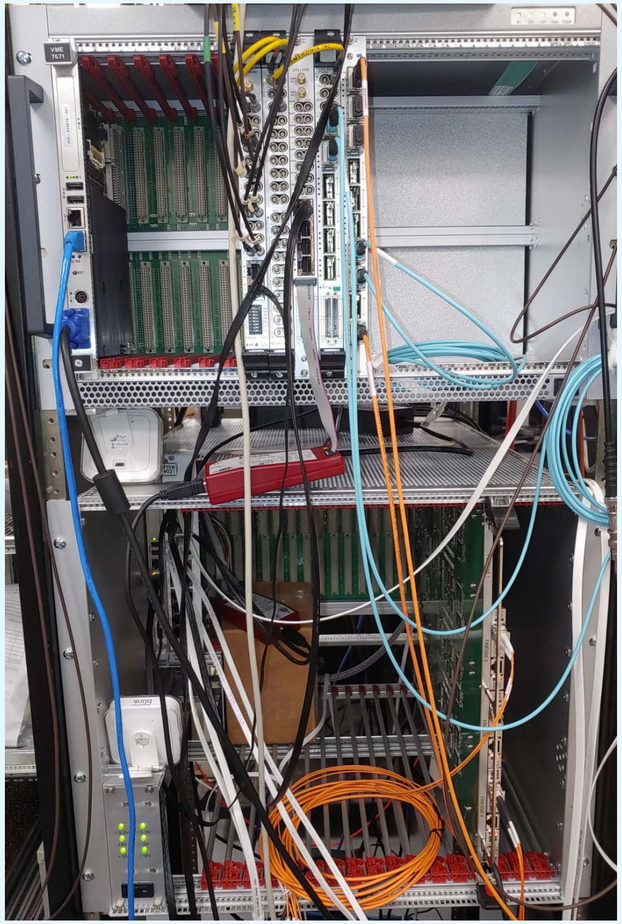
More recent work: TOP-based evaluation of beam collision time for L1 trigger / TOP FEE QC/QA

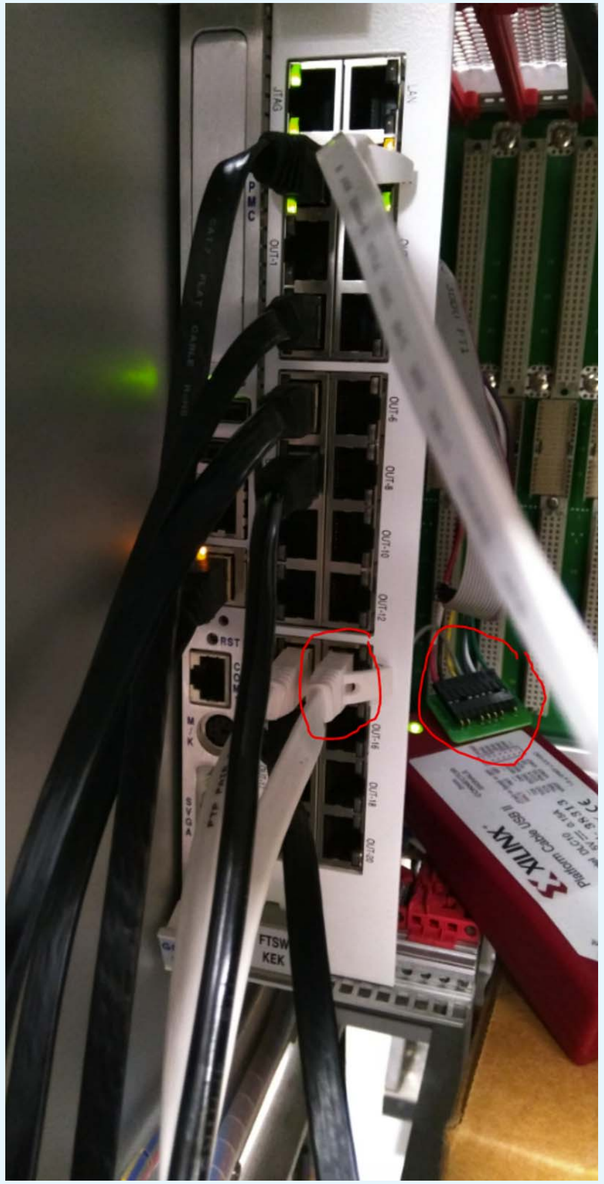


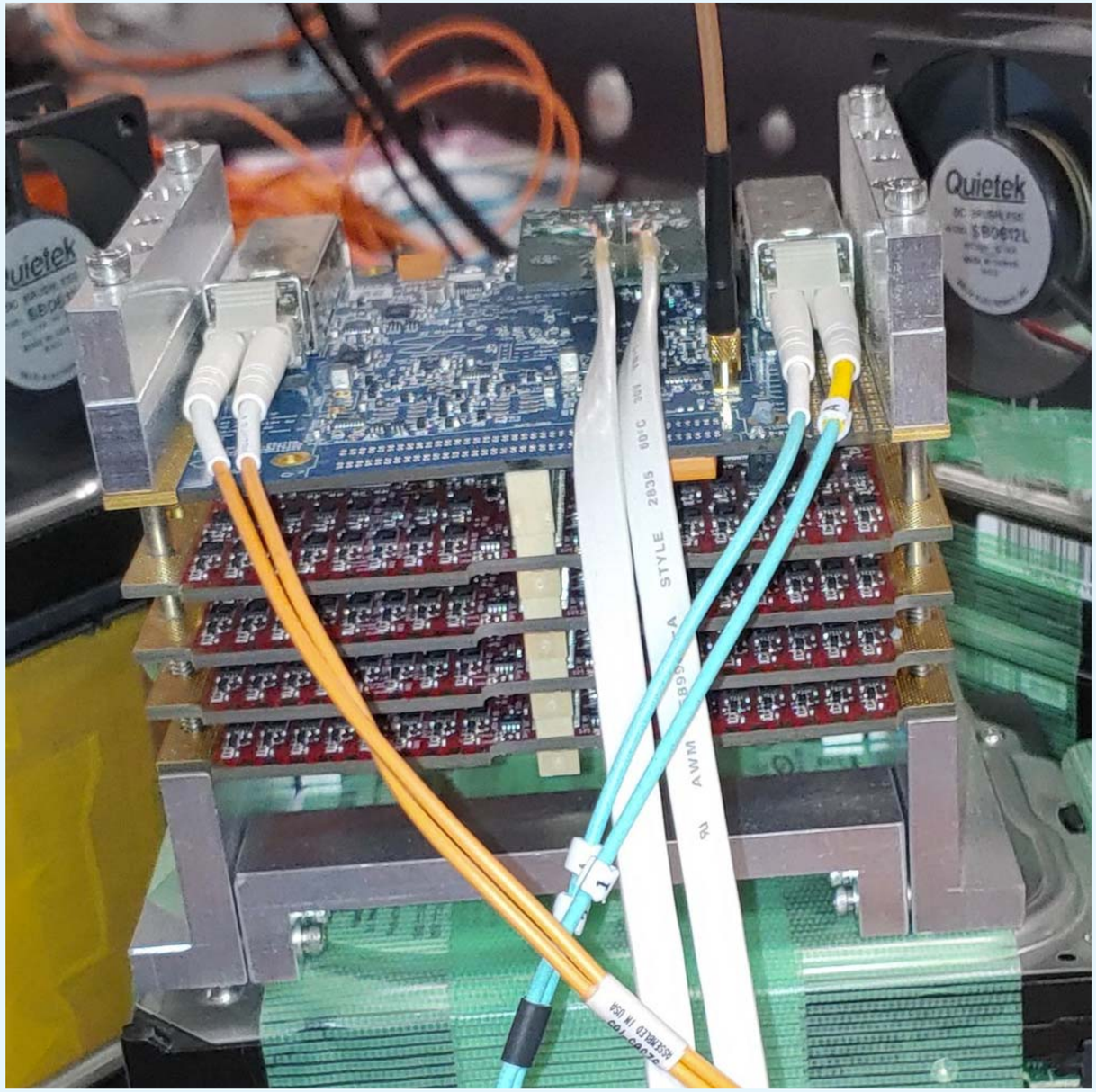
More recent work: TOP-based evaluation of beam collision time for L1 trigger / TOP FEE QC/QA



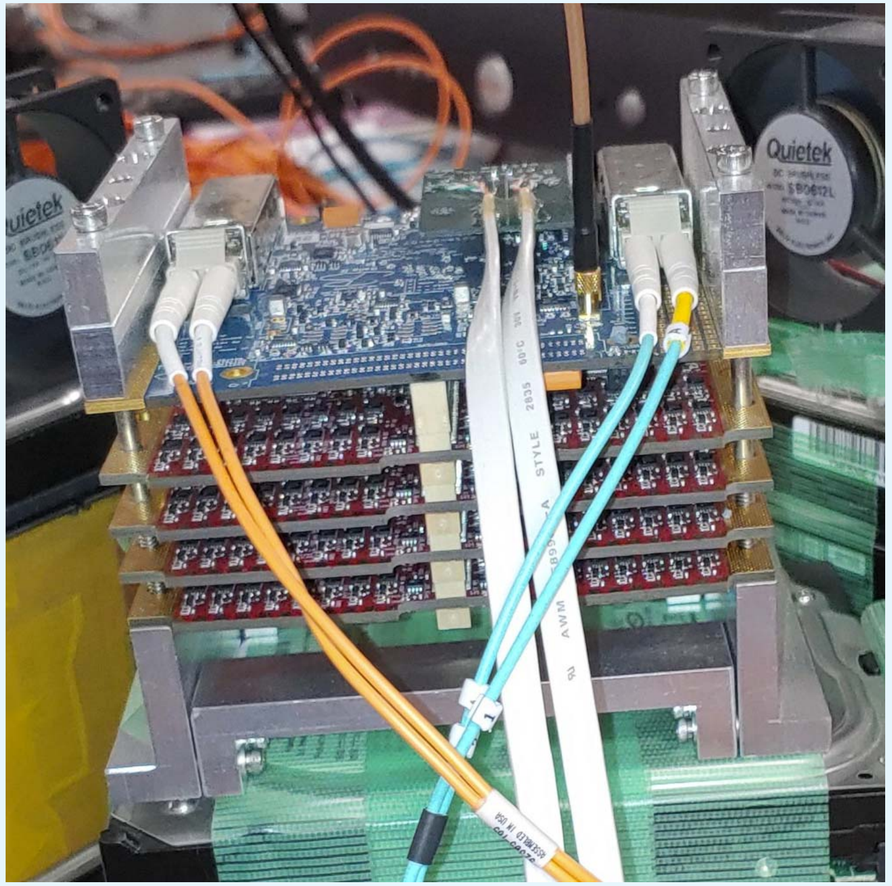
Some of the TOP FEE, trigger and DAQ infrastructure in the lab

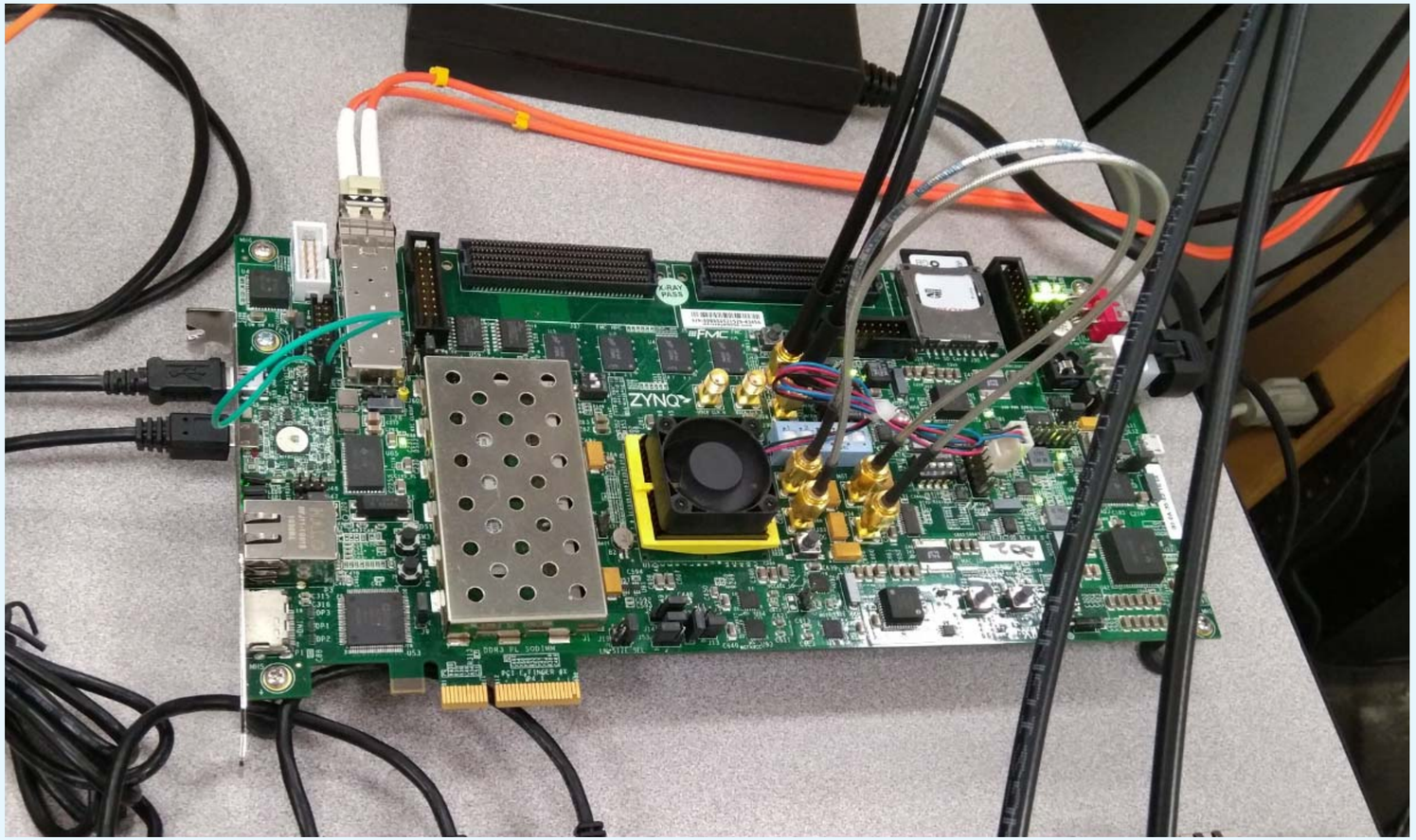






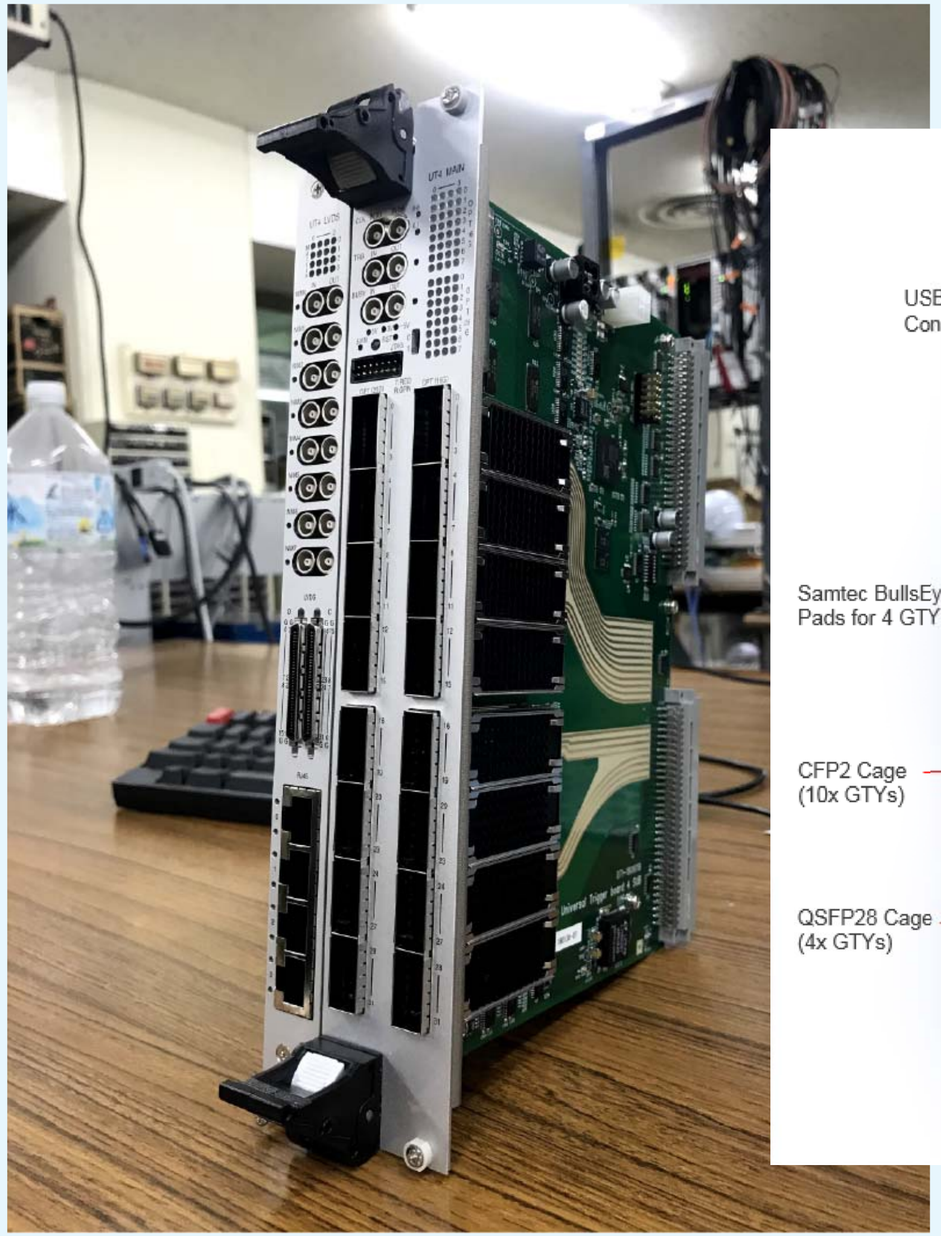
To understand SoC/FPGA ZYNQ-045 we also used ZC706 evaluation board

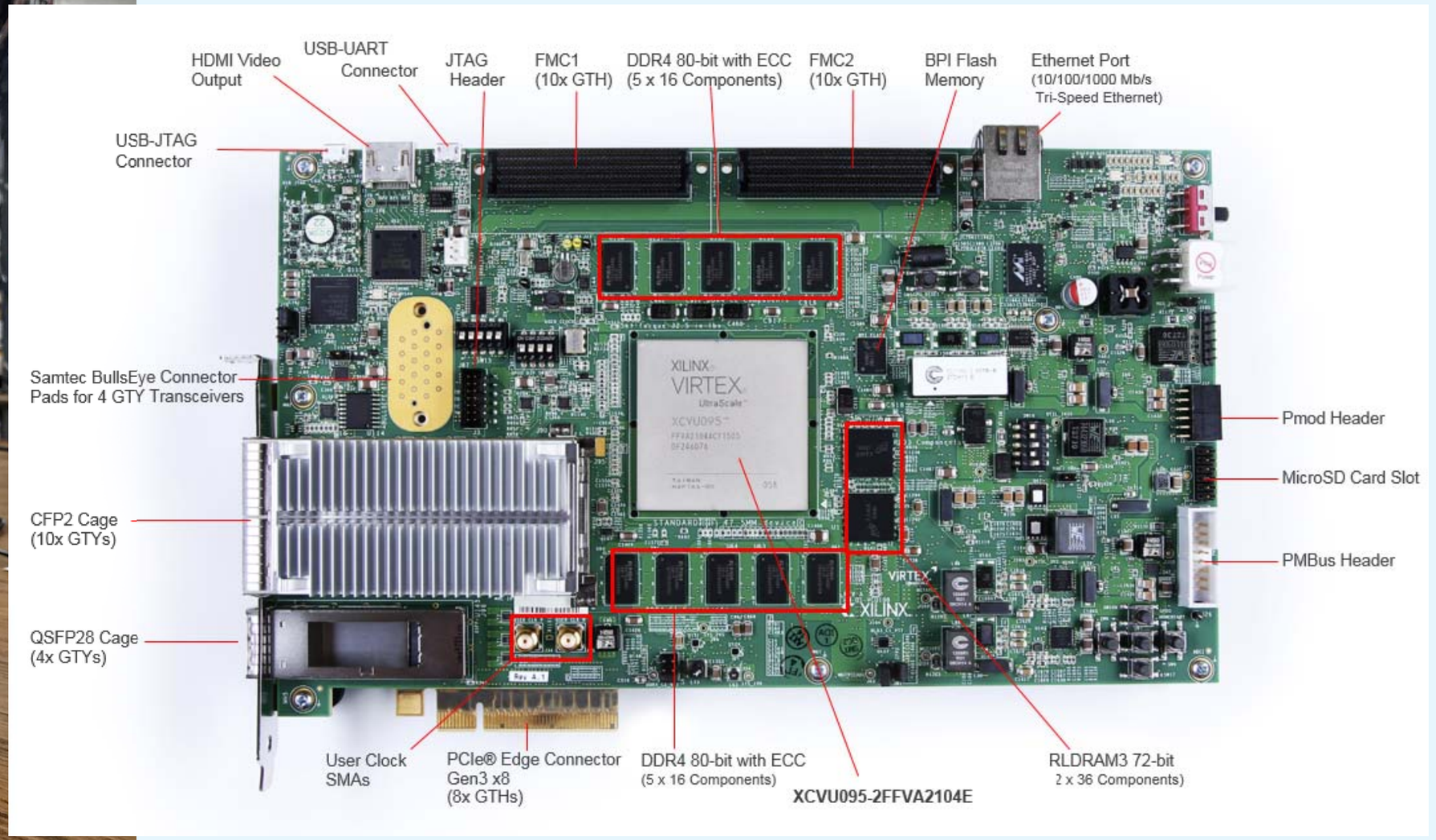




Actual GTY bandwidth, AURORA performance, stability etc have been investigated with realistic reference and system clocks

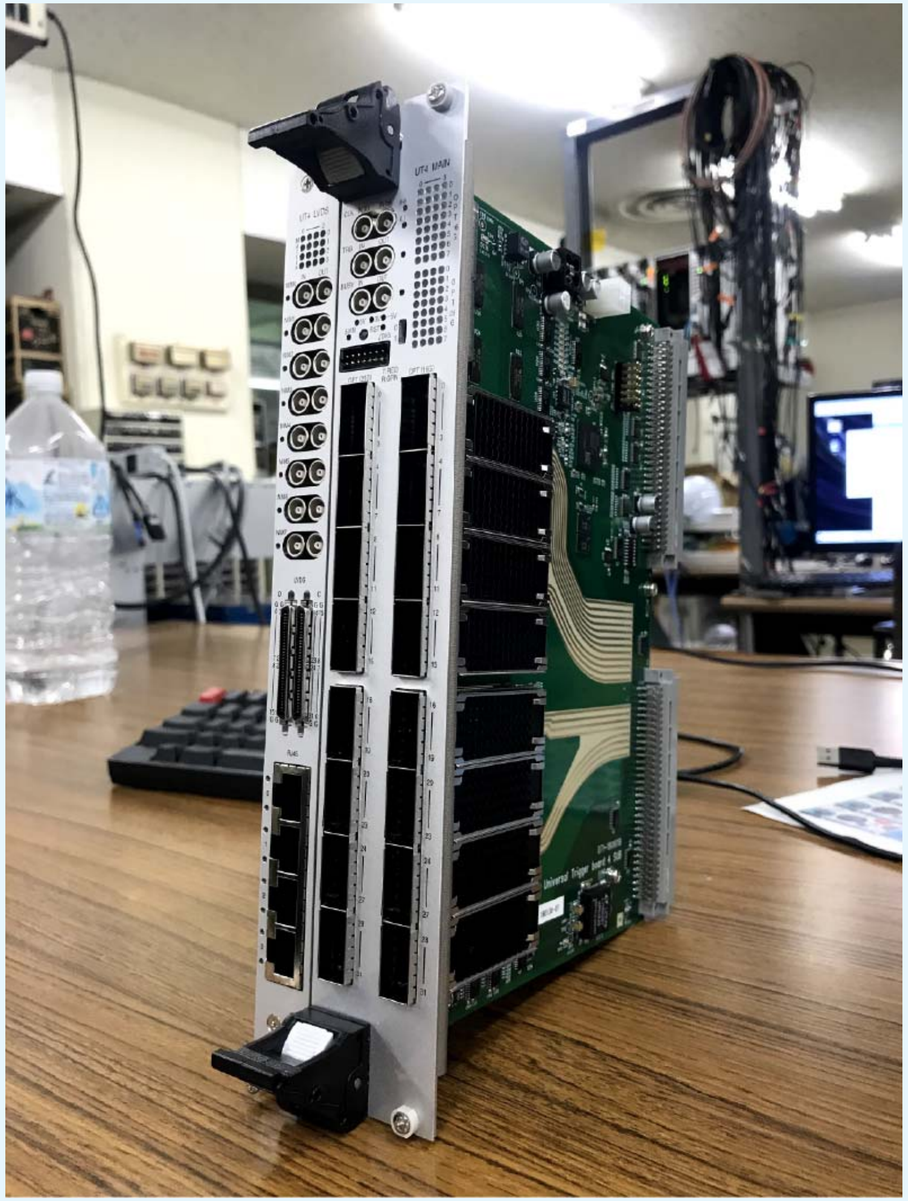
When the time came to upgrade Trigger and DAQ, we used evaluation board VCU108

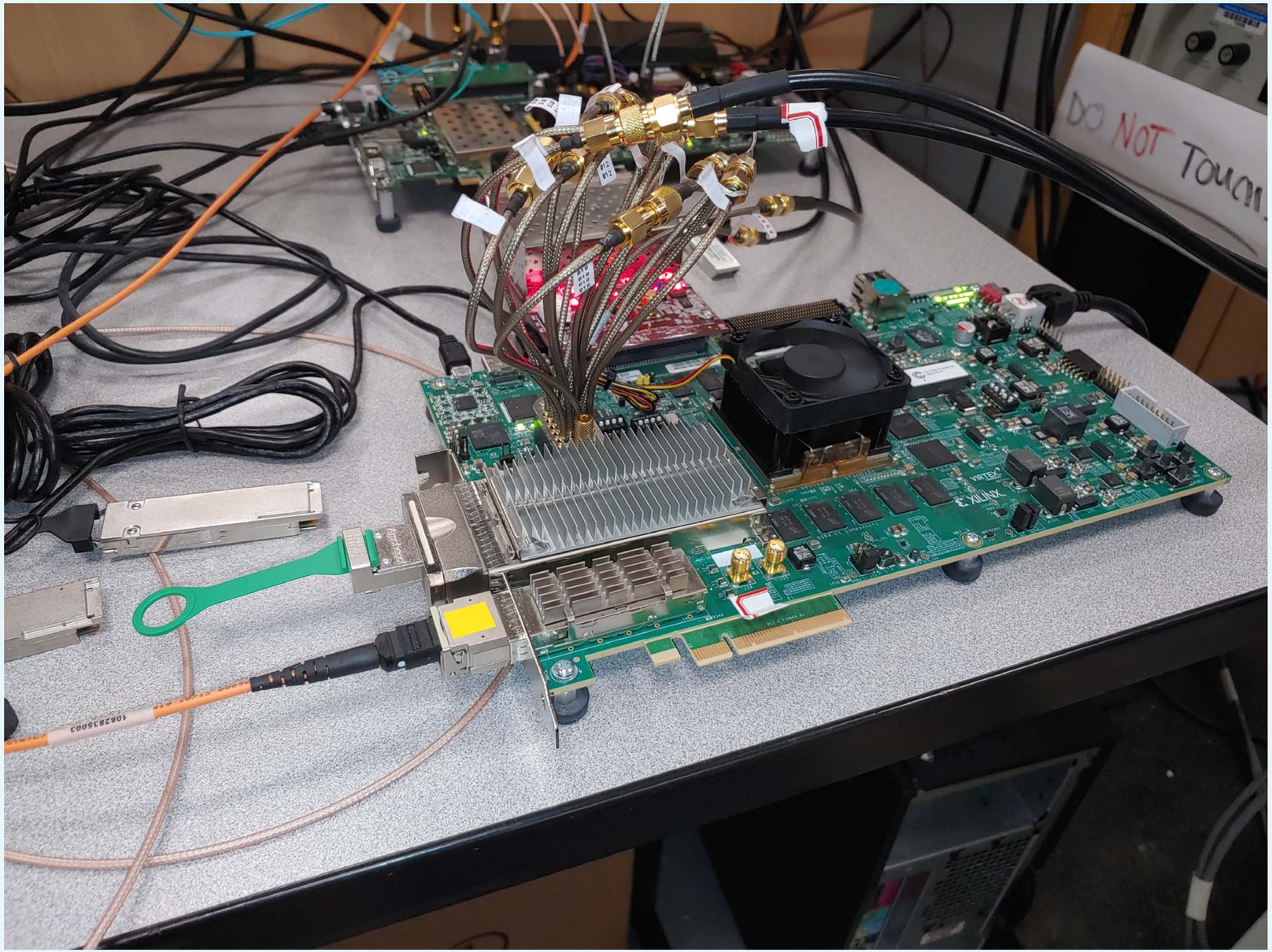




Everything we had to learn about custom boards UT4 (used at KEK for trigger), we learned locally using VCU108 (UltraScale)

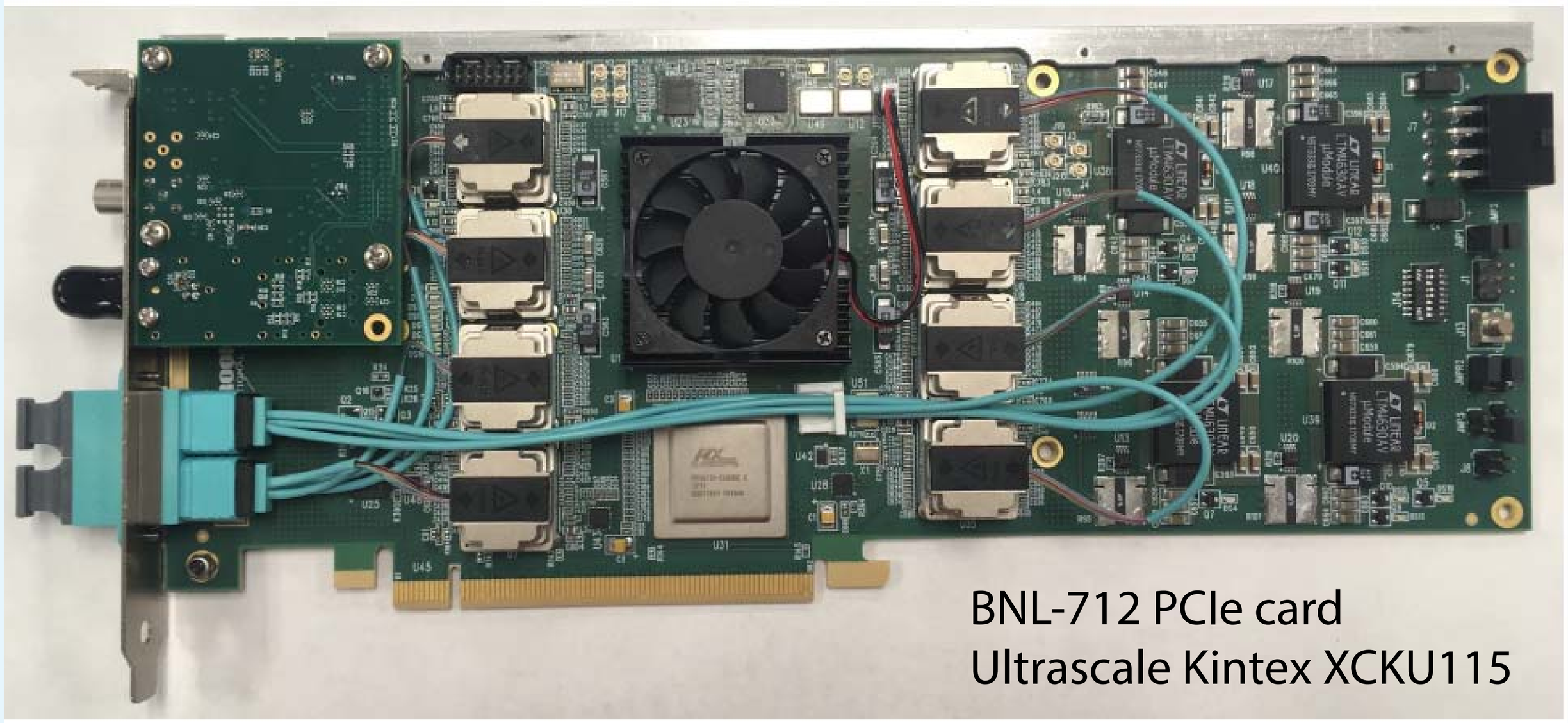
When the time came to upgrade Trigger and DAQ, we used evaluation board VCU108





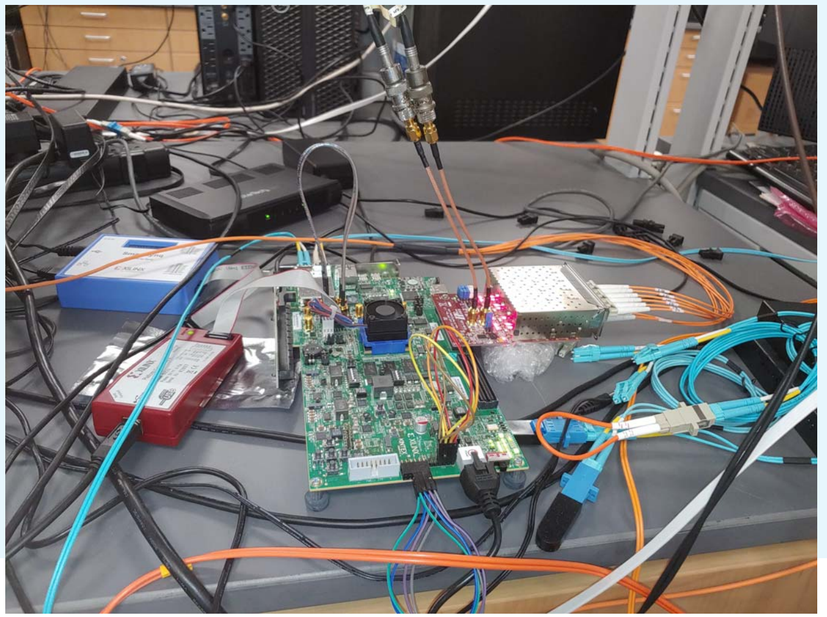
Everything we had to learn about custom boards UT4 (used at KEK for trigger), we learned locally using VCU108 (UltraScale)

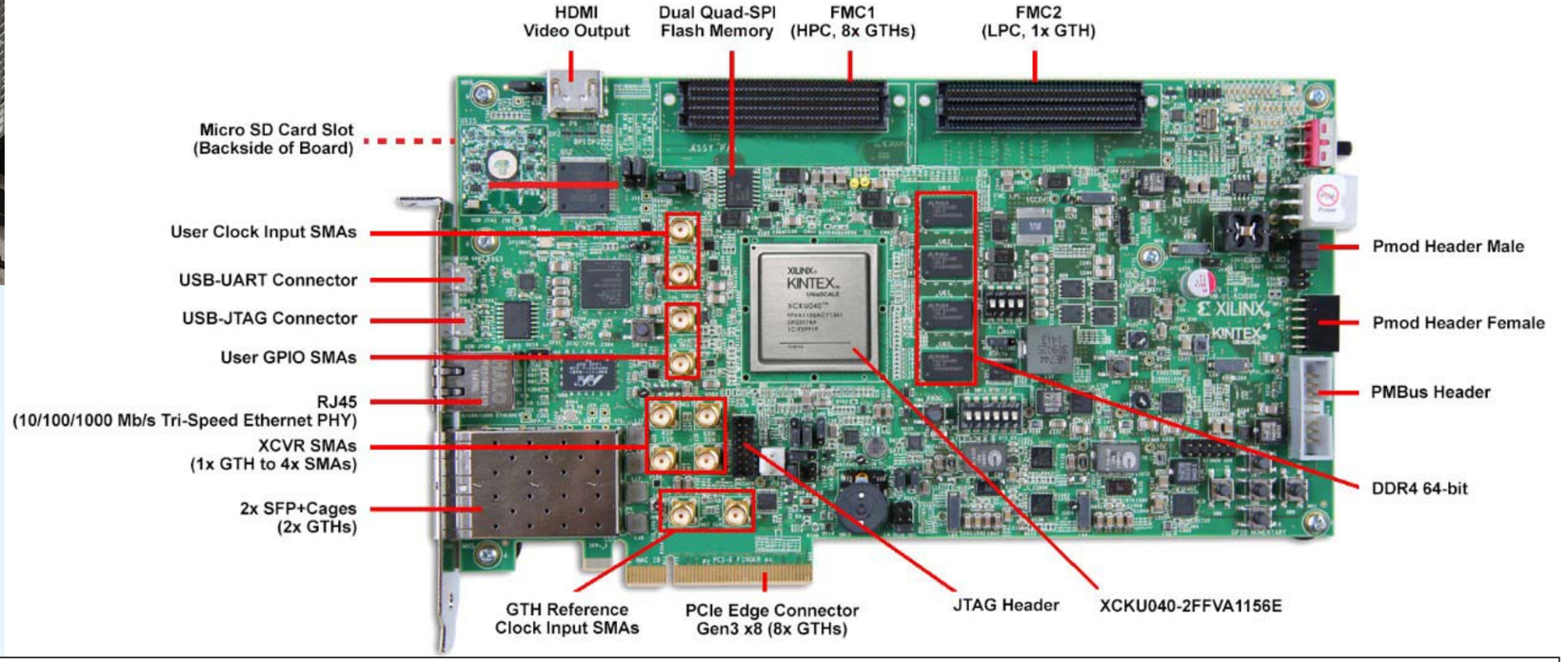
Before PCIe40 (see Dmytro Levit's talk) was chosen for Belle II DAQ upgrade, we played with FELIX



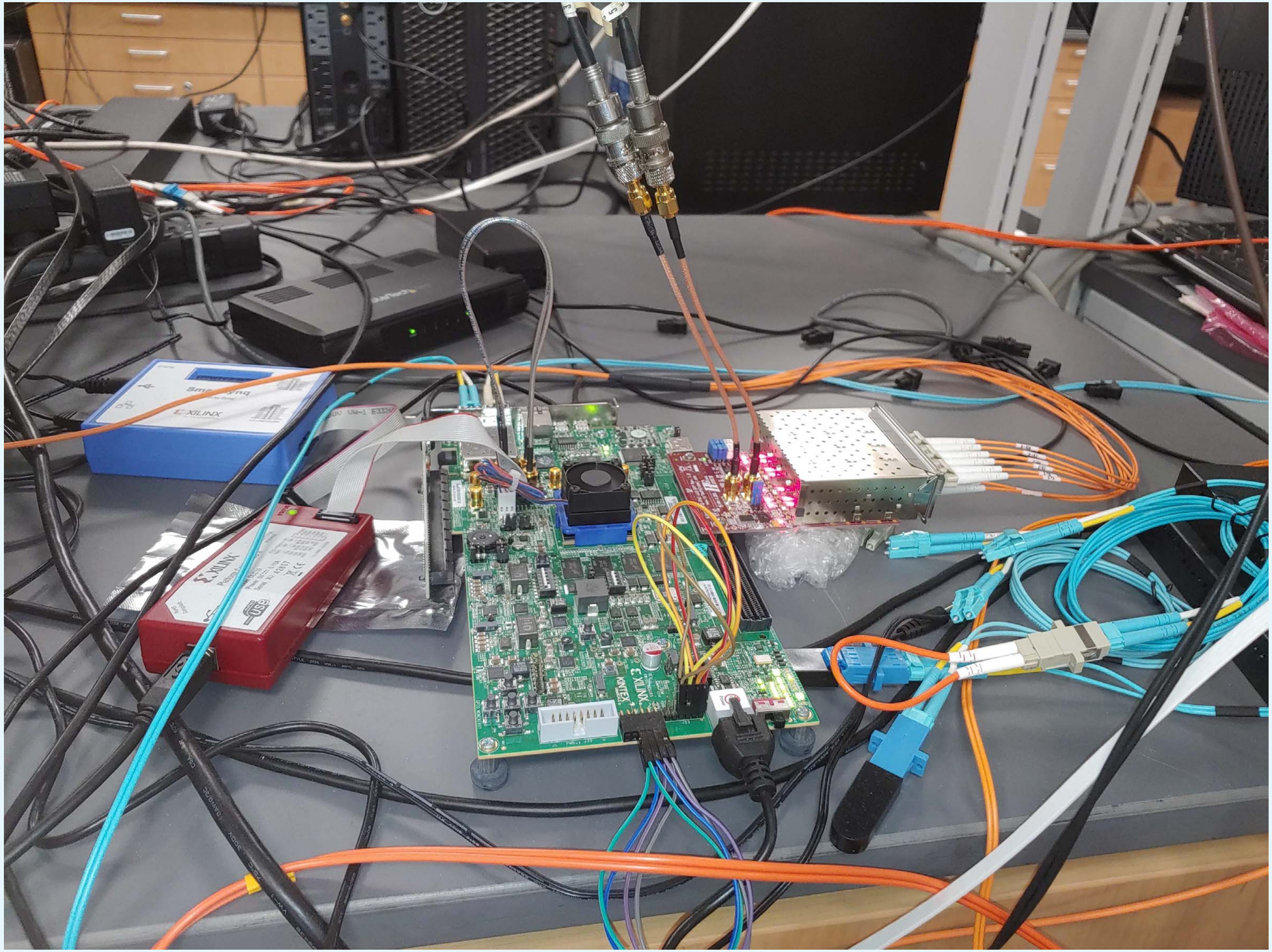
In the course of working on various projects we accumulated an arsenal of FPGA / SoC boards and built up the infrastructure

BNL FELIX platform got us into Kintex and KCU105

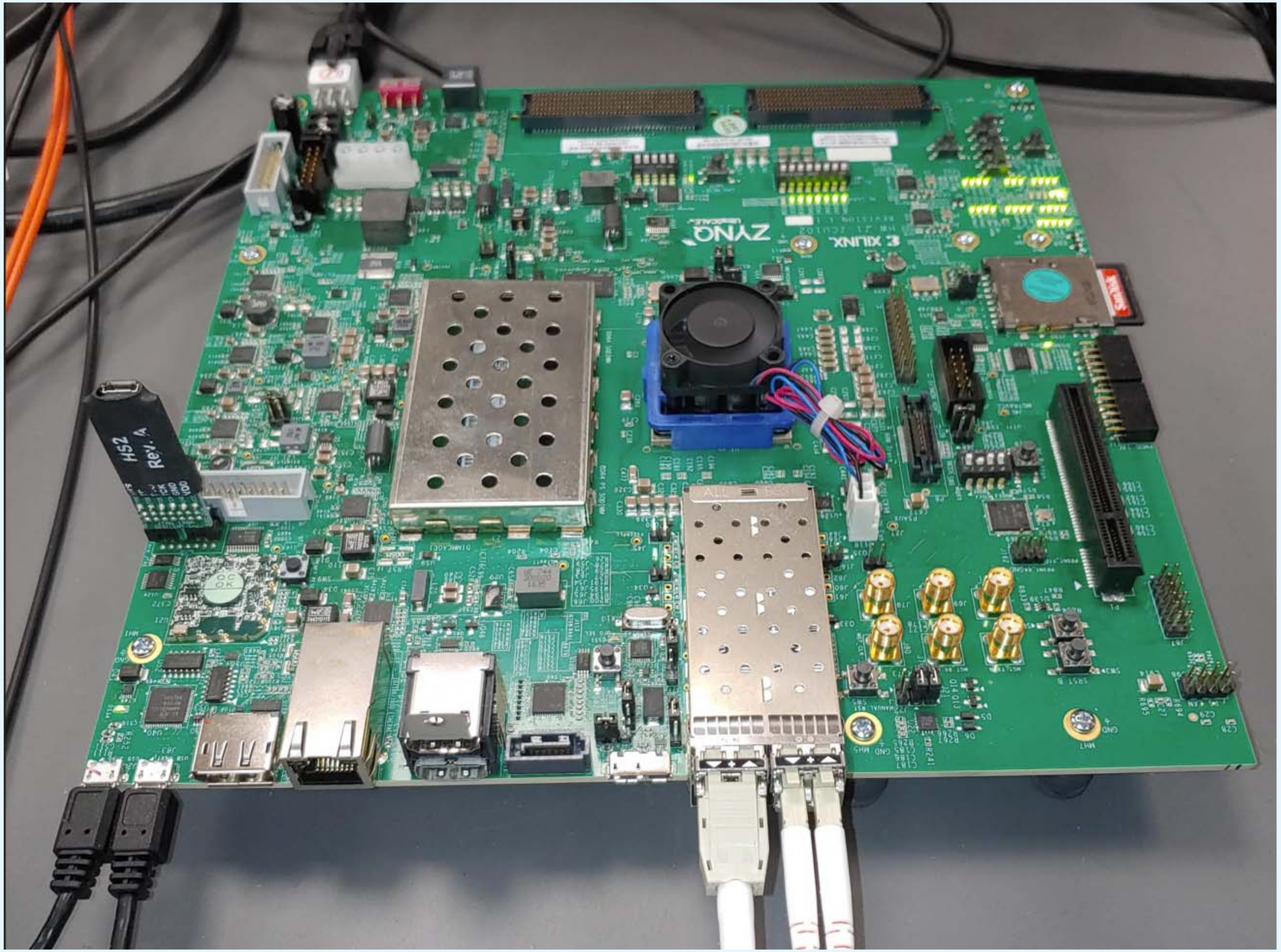




This investment didn't pay off in terms of Belle II DAQ upgrade, however, we got a change to play with Kintex UltraScale

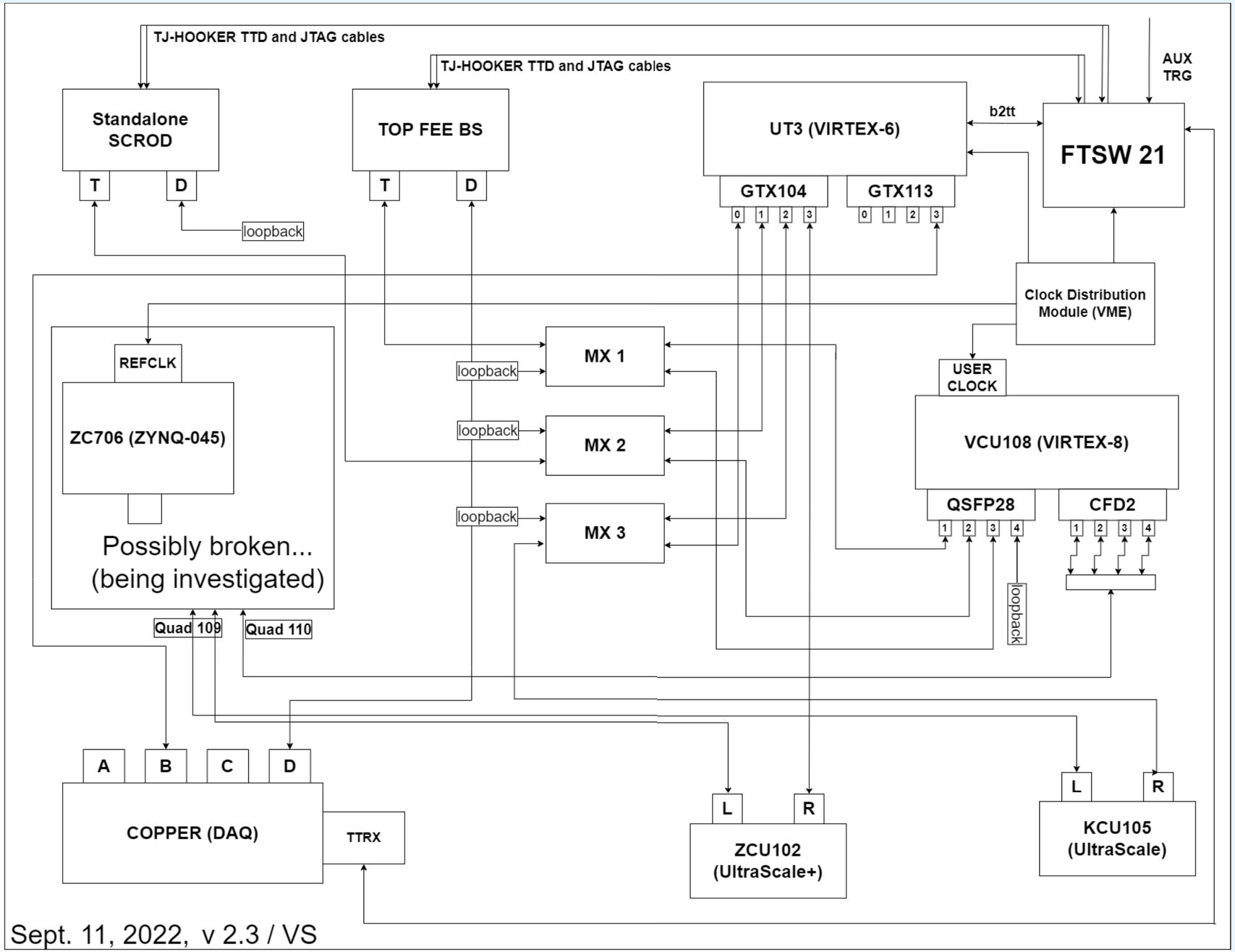

XILINX

When an opportunity unexpectedly became available we used a chance to get an UltraScale+ board



ASIC characterization / connection to NALU Scientific / new sampling and digitizing electronics

Now we use this ecosystem to train new students in the area of FPGA FW and related matters



How we train new students / these are not professionals / usually no prior experience with FPGAs

1) Figure everything out by yourself / free reign to experiment with the boards

2) First project is usually “make an evaluation board do something for you”

3) After it works take a look at Methodology Report / prepare to be shocked

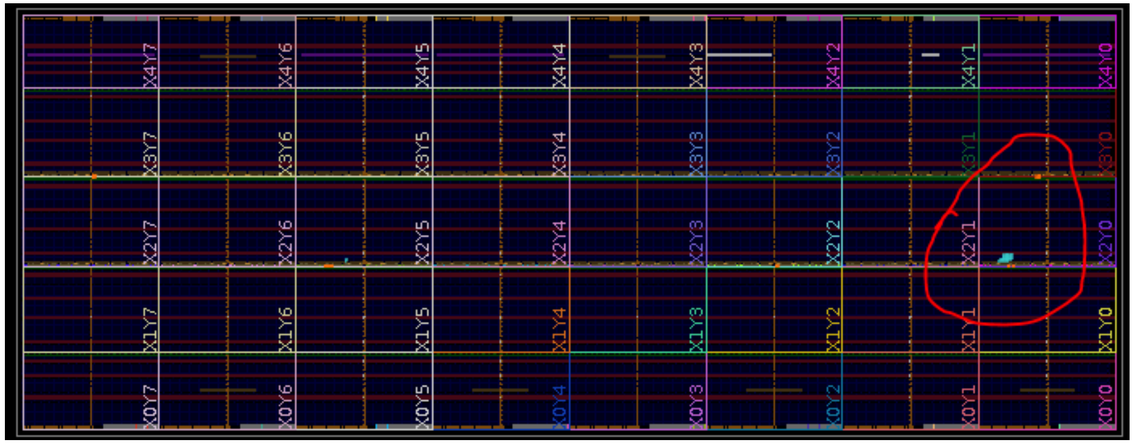
4) Explain repeatedly how HDL is not (exactly) a programming language

5) Ask to prepare IBERTs AND investigate the sources prepared by Core Gen

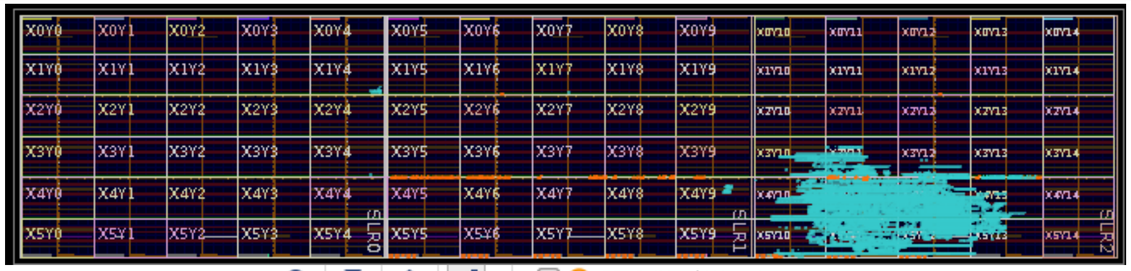
6) Discuss timing reports / domains / constraints / ways to deal with these

7) Learn as much as possible from Transceiver Wizard and such + XILINX docs

Some examples of how "good" of FW designers / how "knowledgeable" we are...

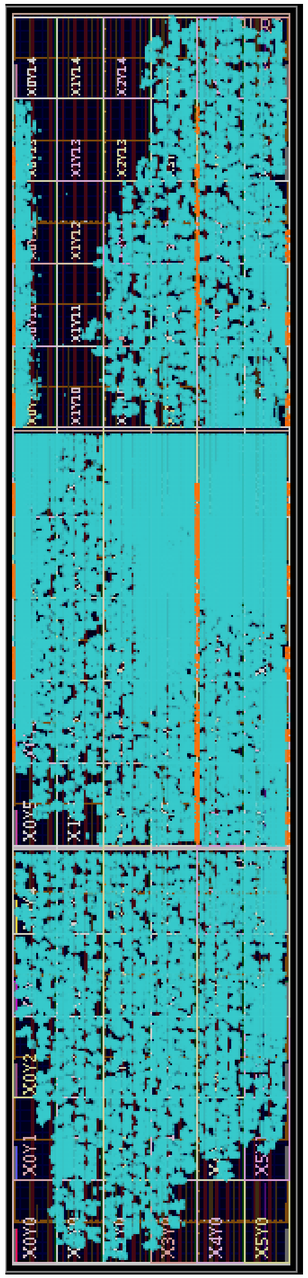


All Violations (67)
Timing (67)
Bad Practice (67)
TIMING-17 (67)



31 Critical Warnings

Name	Severity	De
All Violations (76)		
Timing (46)		
Bad Practice (46)		
Netlist (11)		
XDC (15)		
Clocking (4)		
Bad Practice (4)		



14 Critical Warnings

Name	Severity	Deta
All Violations (214)		
Timing (108)		
Bad Practice (108)		
Netlist (92)		
XDC (8)		
Bad Practice (1)		
XDCB-5 (1)		
Hold (5)		
XDCH-2 (5)		
Constraints Coverage (2)		
XDCV-2 (2)		
Clocking (6)		
Bad Practice (6)		
MMCME3_ADV (6)		
CLKC-25 (1)		

Selected examples of what we discovered (likely about our ignorance) in the course of our work

1) HSLB boards (receivers for FEE readout) weren't stable with our clocks in our environment.
   A student found a way to modify board's firmware to make it work for us.
   Lesson learned: show some initiative and don't rely on someone else to fix the problem for you.
   This student continued his education in a graduate school in EE, designs space-bourne instruments.

2) QC/QA of SCRODs (Standard Control and Read-Out Data boards) revealed that calibration of RAM was very difficult. If I recall correctly, it was traced to some issues in memory controller on the chip.
   Information was obtained in timely manner to make the right decision about using RAM in ops.
   A better calibration scheme was developed (via running it on embedded core of ZYNQ-045).
   Lesson learned: should allow more time between the last design cycle and production stage.

3) Verilog modules instantiation in VIVADO does not always produce the desired result.
   Lesson learned: don't rely just on Chipscope, sometimes RTL design is worth looking at.

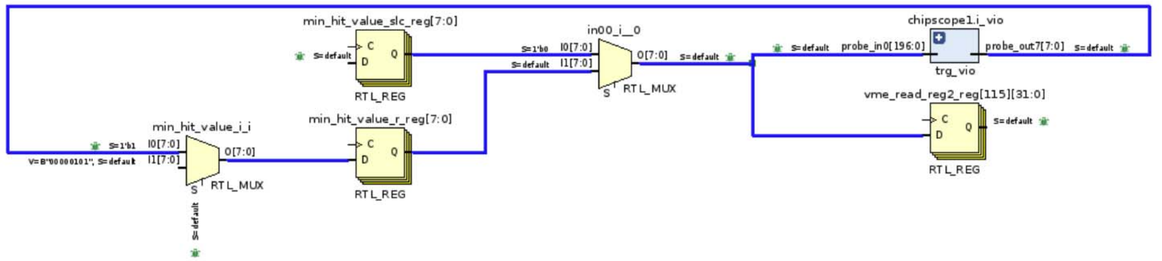
4) While transitioning to newest VIVADO (2023.2.2) discovered that its default optimization changed.
   Lesson learned: be careful with version changes, don't expect the tools to be smarter than you.

5) Discovered an "interesting" difference in how VIVADO grabs all programming cables.

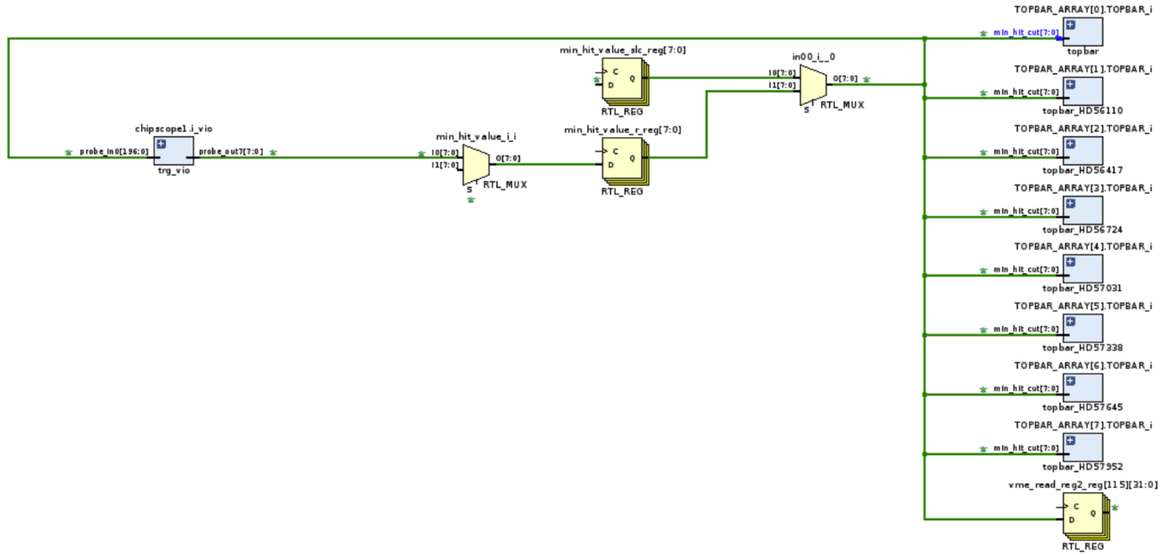
Verilog modules port instantiation surprise when generate block is used

```
// from toptrig.v (top-level module for TOP TRG FW)

***

genvar i;
generate
        for(i=0; i<8; i=i+1) begin: TOPBAR_ARRAY
                topbar TOPBAR_i (
                // User I/O
                .RST_AURORA(rst_aurora_quad[i]),
***
//              .min_hit_cut(min_hit_value),    // BAD!
                .min_hit_cut(min_hit_value_w),  // GOOD!
***
        end
endgenerate

always @ (posedge clk_127)
begin
        min_hit_value_r <= min_hit_value_i;
end

***

reg [7:0]  min_hit_value_slc;

***

min_hit_value_slc <= vme_write_reg2[3][7:0];

wire [7:0]  min_hit_value_local;
assign min_hit_value_local = min_hit_value_r;

wire [7:0] min_hit_value;
assign min_hit_value = (control_mode==1'b0)?(min_hit_value_slc):(min_hit_value_local);

***

// vs / March 7, 2024: critical for routing into 8 instantiated copied of topbar!
wire [7:0] min_hit_value_w;
assign min_hit_value_w = min_hit_value;

***
```

before introducing an additional wire as an input to instantiated/generated copies of TOPBAR:



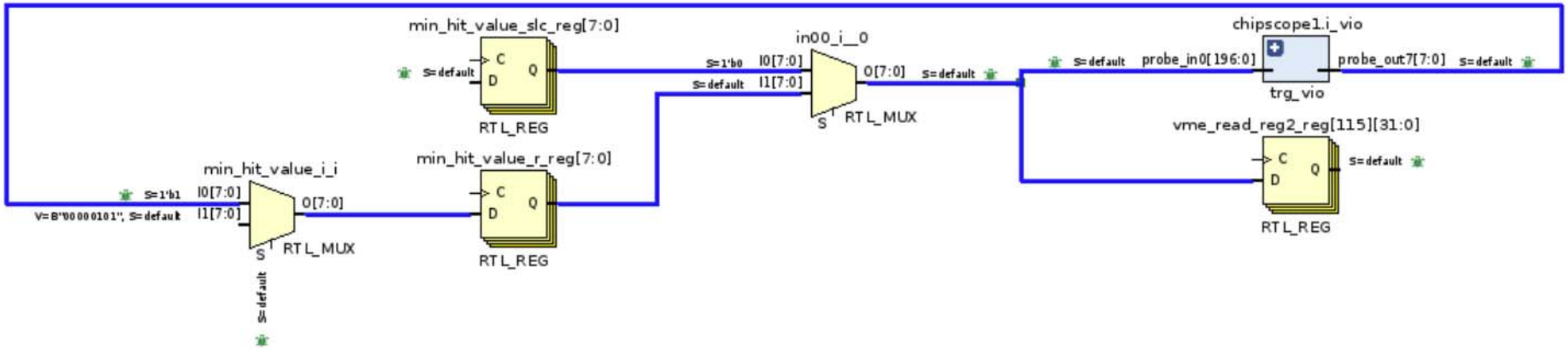
after an additional wire was introduced:



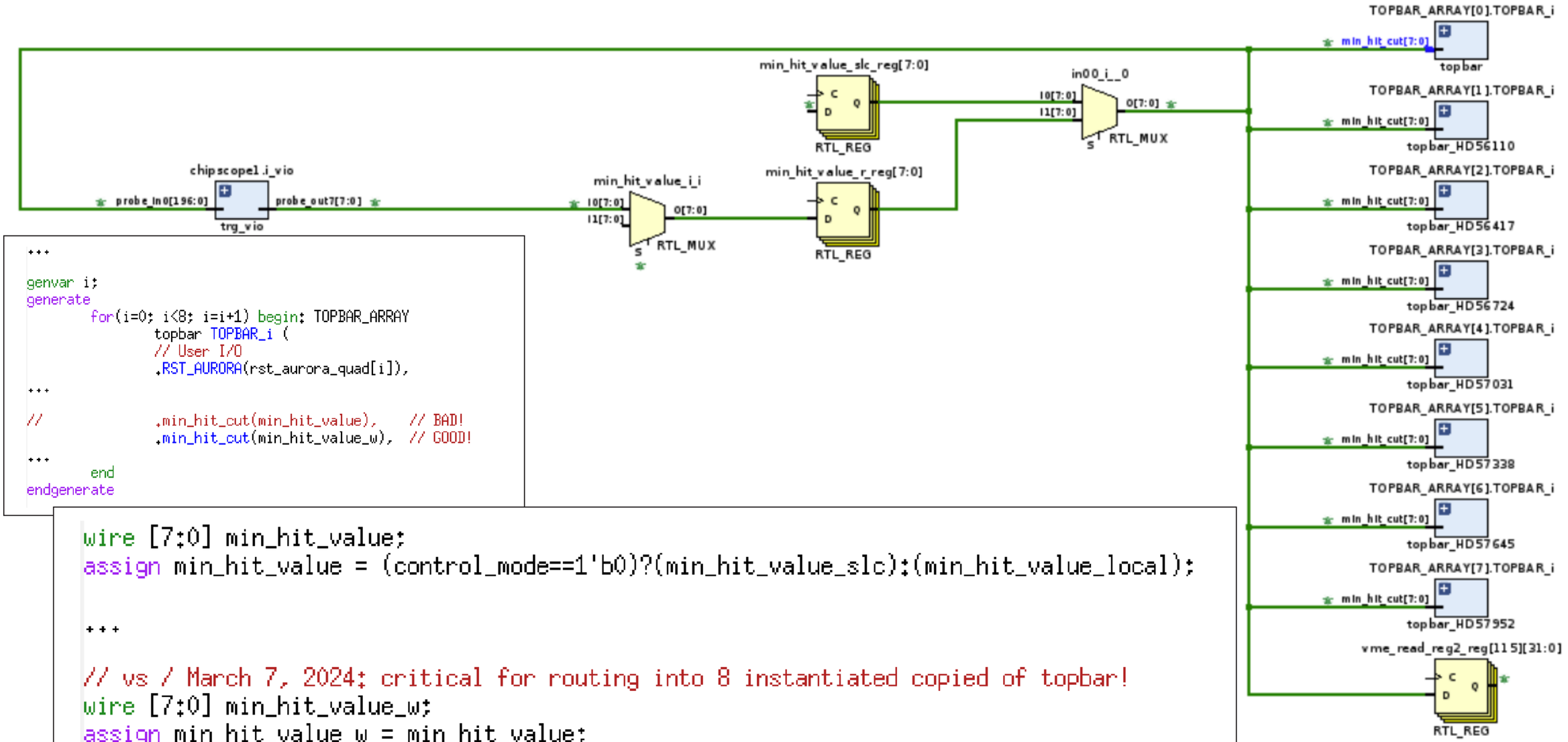
same result (before / after) if an additional register (instead of a multiplexer) is used in a synchronous process

Before an extra redundant wire is introduced, RTL design shows no connection to instantiated modules

before introducing an additional wire as an input to instantiated/generated copies of TOPBAR:



After an extra redundant wire is introduced, RTL design confirms connection to instantiated modules




```
...

genvar i;
generate
        for(i=0; i<8; i=i+1) begin: TOPBAR_ARRAY
                topbar TOPBAR_i (
                // User I/O
                .RST_AURORA(rst_aurora_quad[i]),

...

//              .min_hit_cut(min_hit_value),    // BAD!
                .min_hit_cut(min_hit_value_w),  // GOOD!

...
        end
endgenerate
```

```
wire [7:0] min_hit_value;
assign min_hit_value = (control_mode==1'b0)?(min_hit_value_slc):(min_hit_value_local);


...

// vs / March 7, 2024: critical for routing into 8 instantiated copied of topbar!
wire [7:0] min_hit_value_w;
assign min_hit_value_w = min_hit_value;
```

same result (before / after) if an additional register (instead of a multiplexer) is used in a synchronous process

An interesting example of many instances of a “better default optimization” in newest VIVADO

VIVADO 2020.2

->

VIVADO 2023.2.2

Many reports of multiple drivers

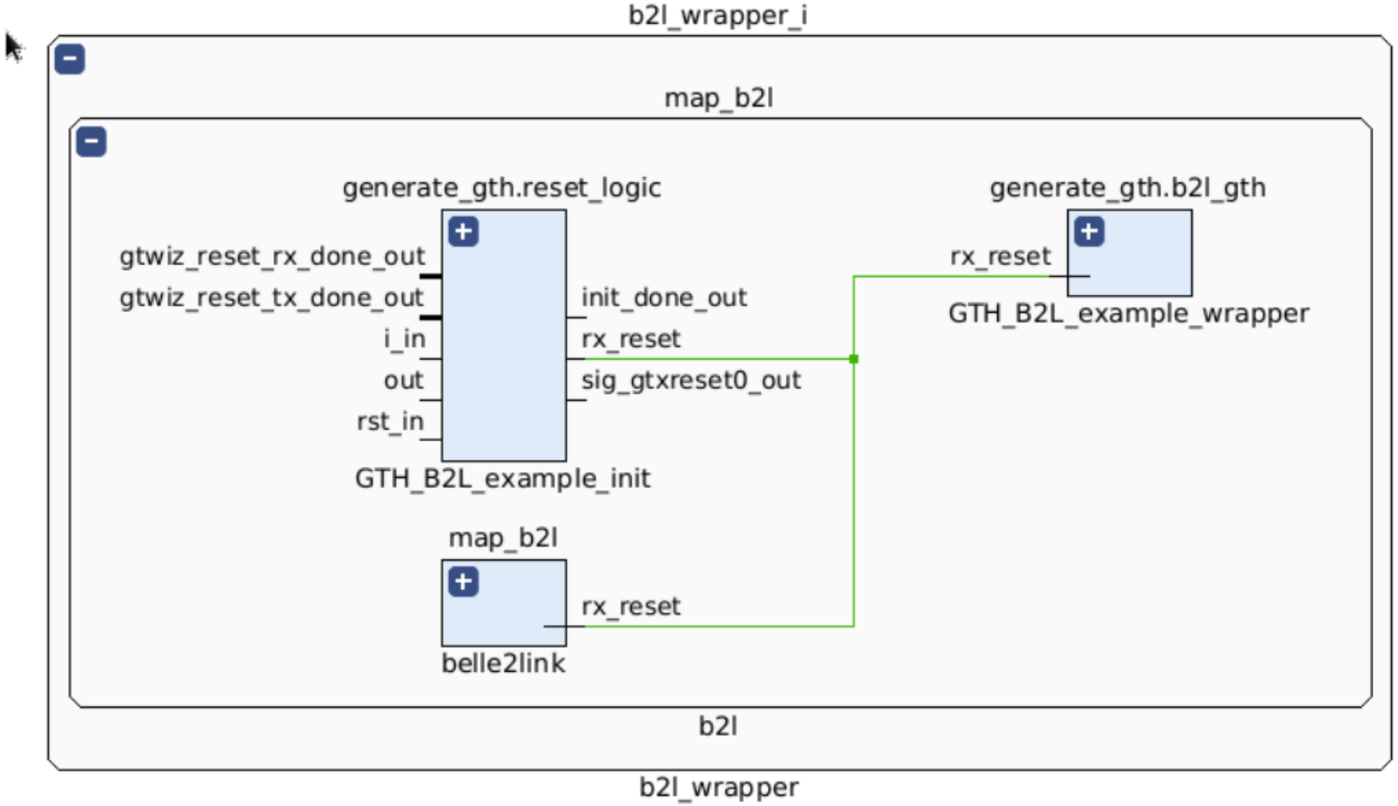
such error was not reported previously!

this is due to the following line in b2l.vhd:

gt_reset_rx_datapath <= gt_reset_rx_datapath_i or rx_reset;

where two resets come from two different places, but note an "or"!

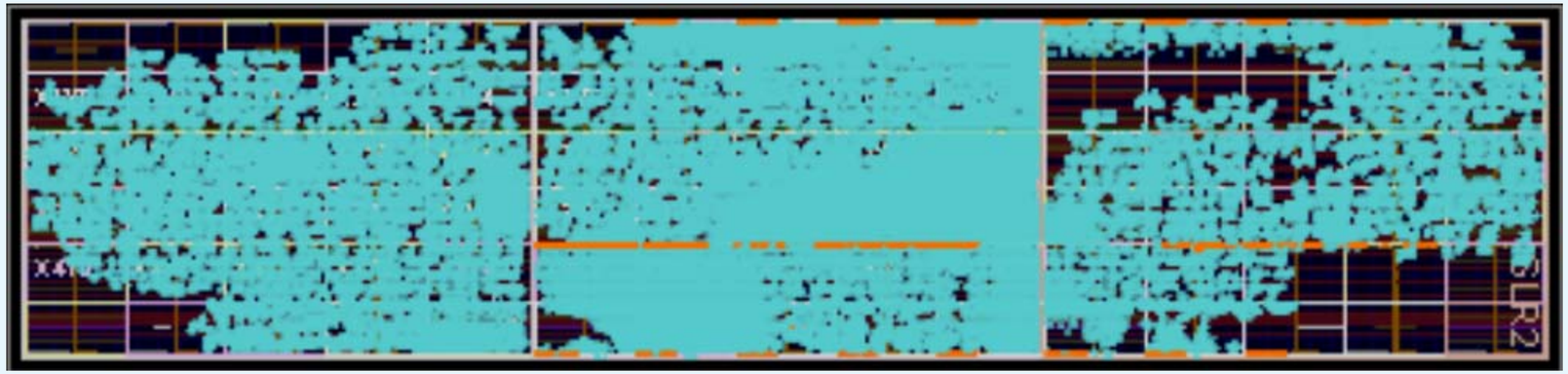
interestingly, this "or" was optimized away because rx_reset is always 1'b0 but then it was reported as an error:



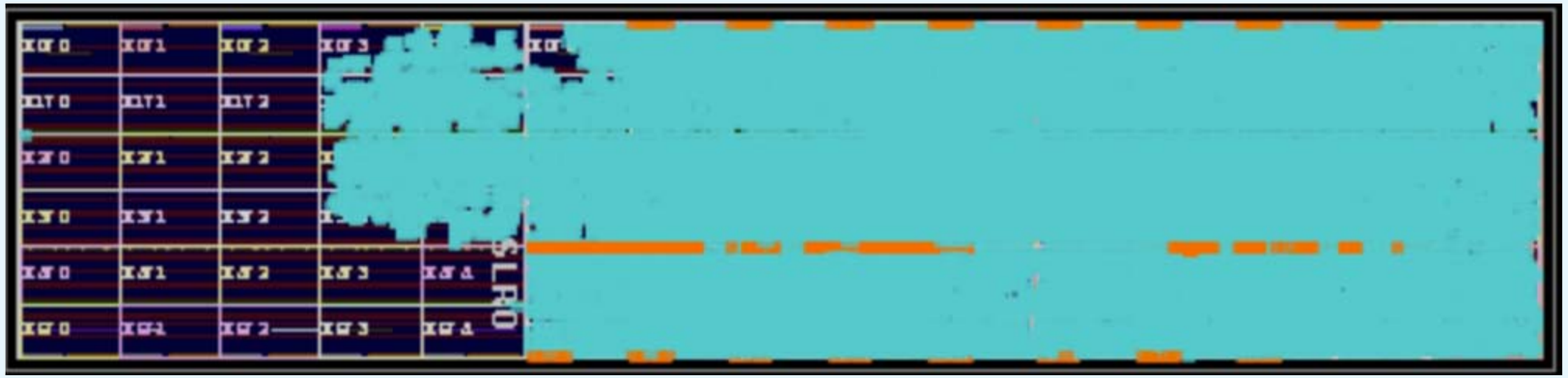
other instances identified also, similar situation, such signals commented out

On the other hand, perhaps the newest VIVADO (with its defaults) is actually much better?

VIVADO 2020.2



VIVADO 2023.2.2



Problem with several instances of VIVADO in a multiuser environment with several devices

The setup / how to experience this problem:

There are several instances of VIVADO running on your system
Several devices are connected to the system
Each device uses its own programming cable
Each instance of VIVADO is expected to use its own exclusive device

The statement of the problem:

Devices could not be programmed / debugged simultaneously
This is because all instances of VIVADO are using the same hardware manager

Solution which does not work:

Make each instance of VIVADO use its own dedicated hardware manager

This does not help because hardware manager which starts first grabs all programming cables

No such problem in IMPACT/ISE because IMPACT could be told which programming cable to use!

A working solution to the problem of simultaneous programming / debugging in VIVADO

Disable all USB devices (for programing cables)
Enable programming cable (USB device) #1
Start hardware manager instance #1
Start VIVADO instance #1
Connect VIVADO instance #1 to hardware manager #1
Disable all programming cables
Enable programming cable (USB device) #2
Start hardware manager instance #2
Start VIVADO instance #2
Connect VIVADO instance #2 to hardware manager #2
Disable all programming cables
...
Enable all programmming cables

Of course, do all this from a script 

```
# a working hack for two instances of VIVADO

disable_all_redboxes
enable_one_redbox 1
start_servers_1
vivado -source ~/bin/connect_to_redbox_1 &
sleep 10

disable_all_redboxes
enable_one_redbox 2
start_servers_2
vivado -source ~/bin/connect_to_redbox_2 &
sleep 10

enable_all_redboxes
```

Of course, all this is of questionable value and quality, but works well for our purposes

```
#!/bin/bash
# disable all redboxes
echo "Xilinx programming cables found:"
lsusb | grep -i "xilinx" | grep -v "grep" | grep -v "show_all_redboxes"
list=`lsusb | grep -i "xilinx" | grep -v "grep" | grep -v "show_all_redboxes" | replace " " "#" -- | replace ":" "" --`
it=1
for box in ${list}
do
#  echo "found a programmer: ${box}"
  bus=`echo ${box} | awk -F# '{print($2)}'`
  device=`echo ${box} | awk -F# '{print($4)}'`
#  echo "Bus: $bus, device: $device"
  devfull="/dev/bus/usb/${bus}/${device}"
  echo "Before disabling programmer $it:"
  ls -qasl ${devfull}
  sudo chmod 0 ${devfull}
  echo "After disabling programmer $it:"
  ls -qasl ${devfull}
  it=$((it+1))
done
```

```
#!/bin/bash
# enable one redbox
itenable=$1
if test "$itenable" != "1" && test "$itenable" != "2"
then
  echo "There is no programmer $itenable (?). Your choices are 1 or 2"
  exit
fi
list=`lsusb | grep -i "xilinx" | grep -v "grep" | grep -v "show_all_redboxes" | replace " " "#" -- | replace ":" "" --`
it=1
for box in ${list}
do
  if test "${itenable}" == "$it"
  then
    bus=`echo ${box} | awk -F# '{print($2)}'`
    device=`echo ${box} | awk -F# '{print($4)}'`
    devfull="/dev/bus/usb/${bus}/${device}"
    echo "Before enabling programmer $it:"
    ls -qasl ${devfull}
    sudo chmod 666 ${devfull}
    echo "After enabling programmer $it:"
    ls -qasl ${devfull}
  fi
  it=$((it+1))
done
```