

TARGETC readout boards

Salvador Ventura

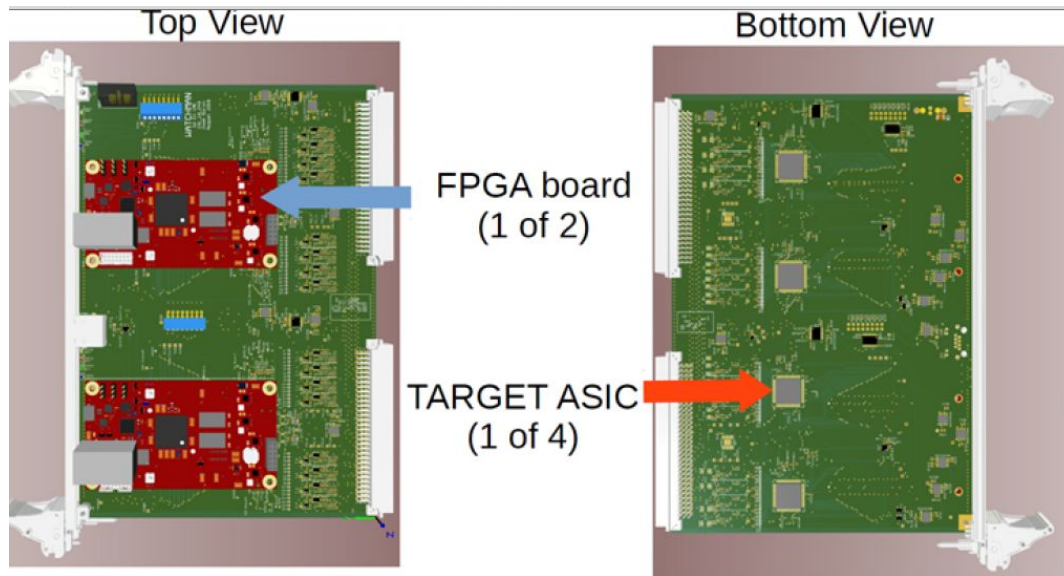
Jose Duron, Gary Varner, Kurtis Nishimura

HMB kick-off meeting

08-21-2020

16U VME Card

- 4 TARGETCs
- 16 PMT inputs.
- Gain stages for each PMT input: $\times 10$, $\times 1$, $\times 1/10$ and $\times 1/100$.
- 2 trigger mechanisms to improve sensitivity and minimize false triggers.
- 2 Microzed Boards, Zynq Z020

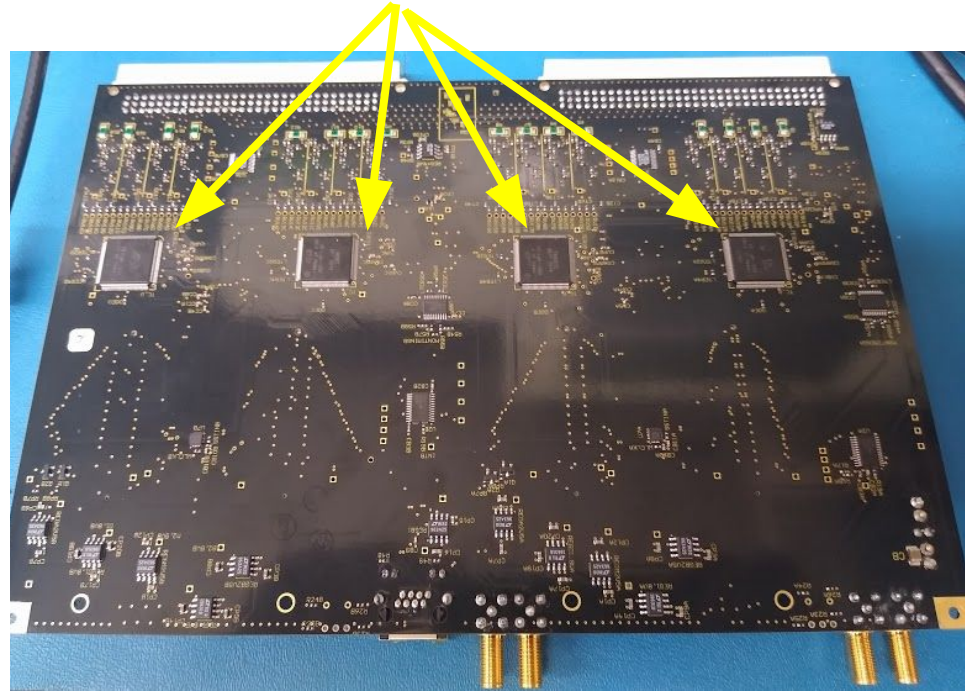
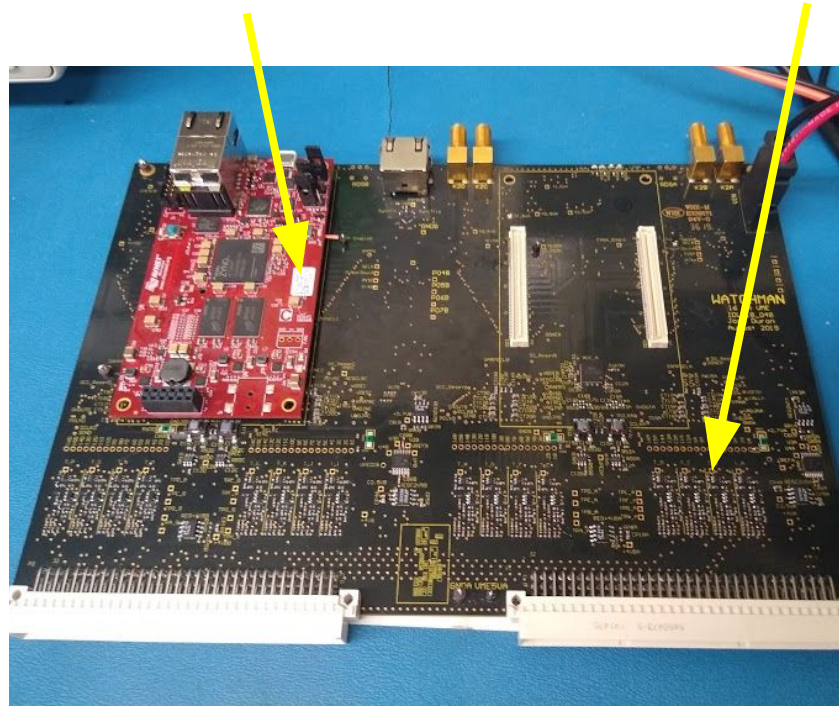


TARGETC Readout for WATCHMAN

Microzed board

Gain stages

TARGETCs



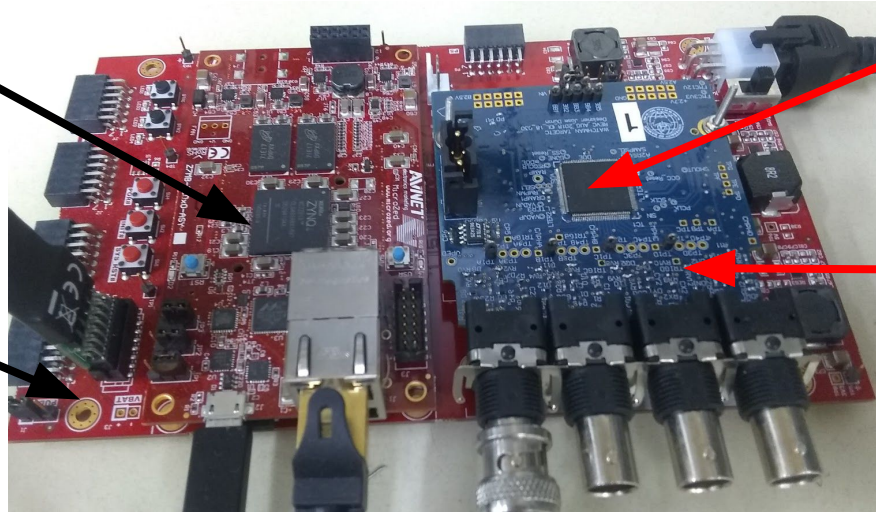
TARGETC Readout for WATCHMAN

FMC Board

- **TARGETC**. A 1GSa/s, 12-bit, 16-channel Wilkinson ADC, which was designed and fabricated by Gary Varner's ASIC group at UH-Manoa.
- A **ZYNQ SoC** by Xilinx with an FPGA and a dual-core ARM processor.

MicroZed Board
Zynq-7000 SOC
FPGA+ ARM Proc.

FMC carrier board

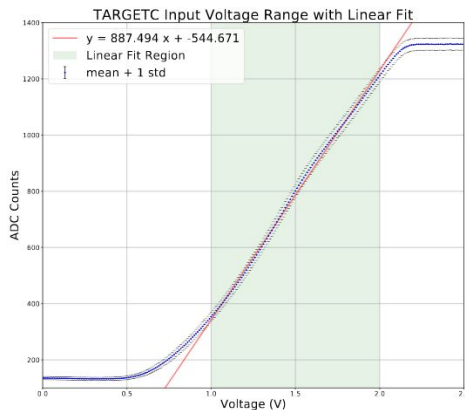


**x1, x10, x0.1
and x0.01
gain stages**

Done:

Firmware including:

- Interface with TARGETC
- Auto trigger algorithm: sel storage array address to write and windows to digitize from trigger signal.
- Data transfer

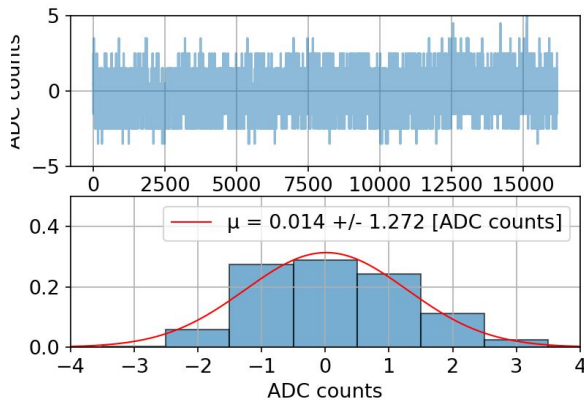


Dynamic range

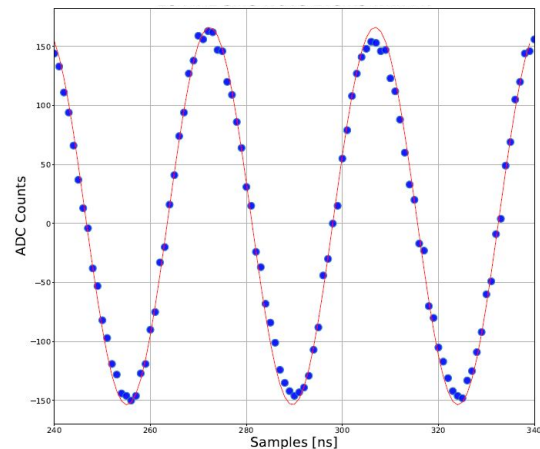
4 PMT-channel FMC board characterization

Current status:

- Migration from FMC board to VME board

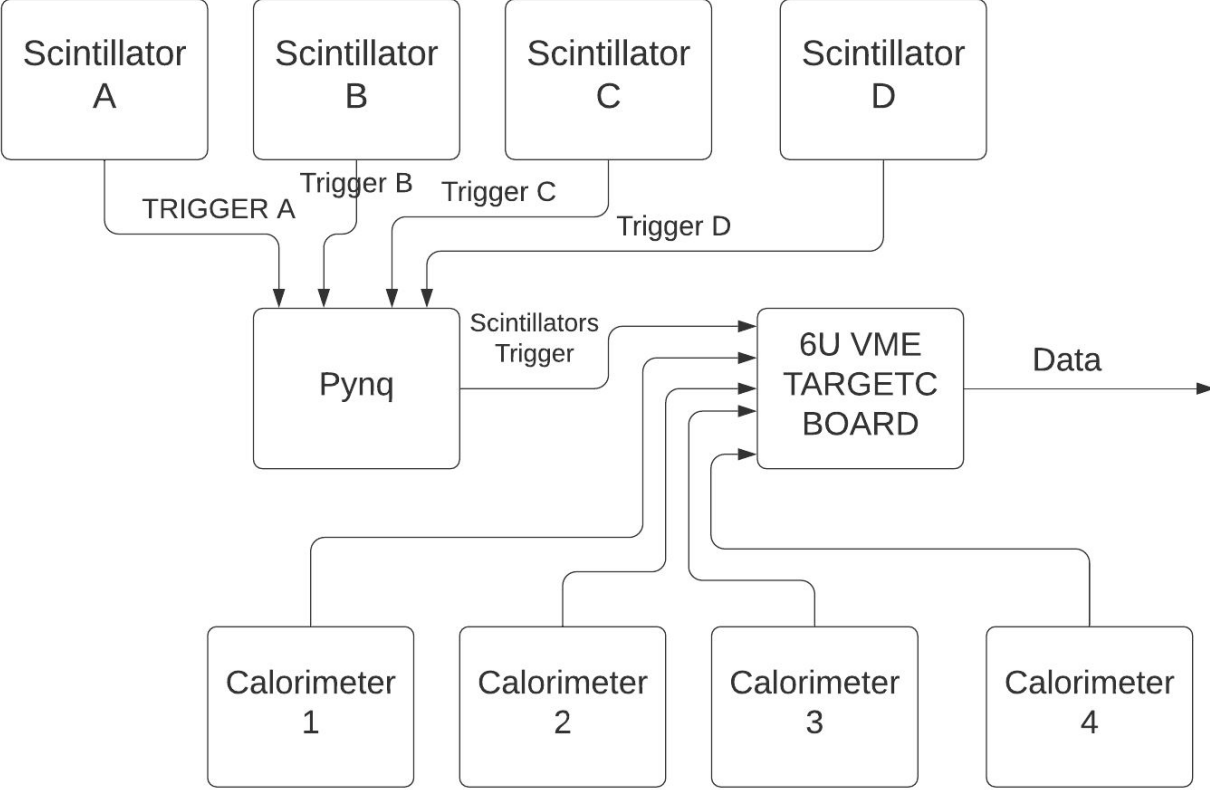


Noise

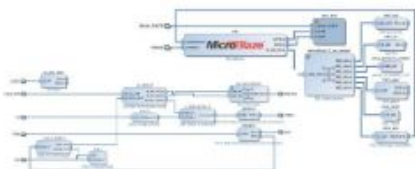


Sine wave fitting

Hawaii Muon Beamline



FPGA overlays – hardware libraries



Step 1:
Create an FPGA design for a class
of related applications

```
void set_invert_bits()
{
  write(0x00000000, INVERT_BITS_REG);
}

void set_invert_bits_enable()
{
  write(0x00000001, INVERT_BITS_REG);
}

int main()
{
  // set invert bits
  set_invert_bits();

  // set invert bits enable
  set_invert_bits_enable();

  // initialization
  init_board();
}
```

Step 3:
Wrap the C API to create a Python
library

```
void main()
{
  // initialize
  init_board();
  init_board();
  init_board();
  // set invert bits to 1
  set_invert_bits(1);
  // set invert bits enable to 1
  set_invert_bits_enable(1);
  // read data from the device
  int data[100];
  read_data(data, 100);
  for (int i = 0; i < 100; i++)
  {
    printf("Data %d: %d\n", i, data[i]);
  }
}
```

Step 2:
Export the bitstream and a C API
for programming the design

```
from time import sleep
from sys import argv
from pynq import Overlay, PYNQ_ADC, PYNQ_DAC

o1 = Overlay('bone.bit')
o1.download()
# setting values from user to 2.0v with step 0.1v.
dac_id = int(argv[2]) # Type in the PYNQ ID of the DAC (0 - 3)
adc_id = int(argv[3]) # Type in the PYNQ ID of the ADC (0 - 3)

dac = PYNQ_DAC(dac_id)
adc = PYNQ_ADC(adc_id)

for j in range(10):
    value = 0.1 * j
    dac.write(value)
    sleep(0.5)
    # reading analog read from the ADC
    # x1=rawinput[0]
    print("voltage read by DAC is: (1.4F) volts, format: raw[1,0][0]")
```

Step 4:
Import the bitstream and the library
in your Python scripts and program