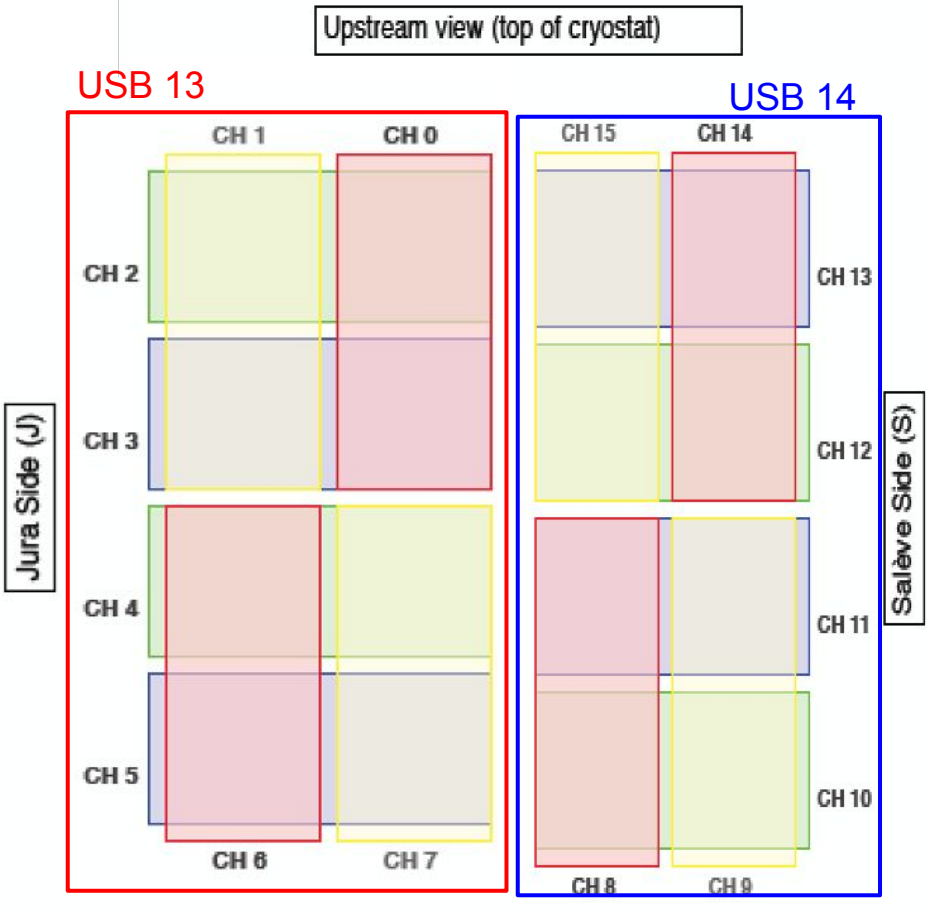
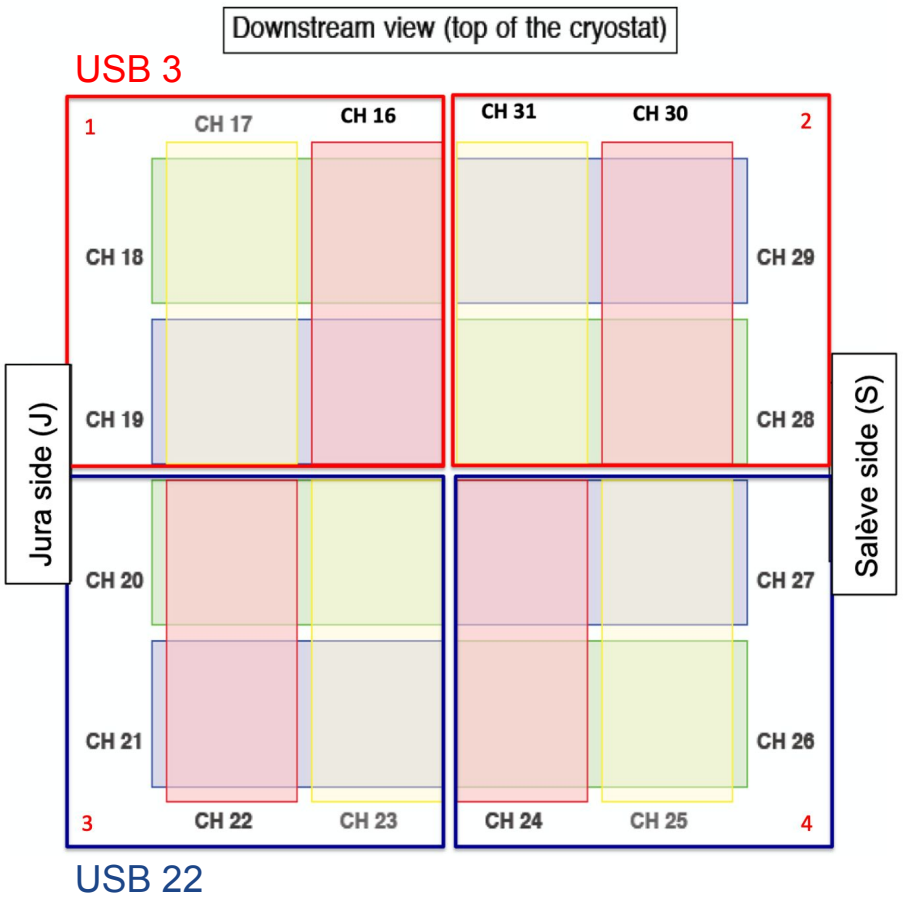
CRT Integration to DUNE DAQ for ProtoDUNE

Matt Murphy
29 February 2024

32 modules arranged into 8 super-modules and 4 individual USB daisy chains
Each module has one PMT with 64 channels, each reading out single scintillator strip





Running the backend

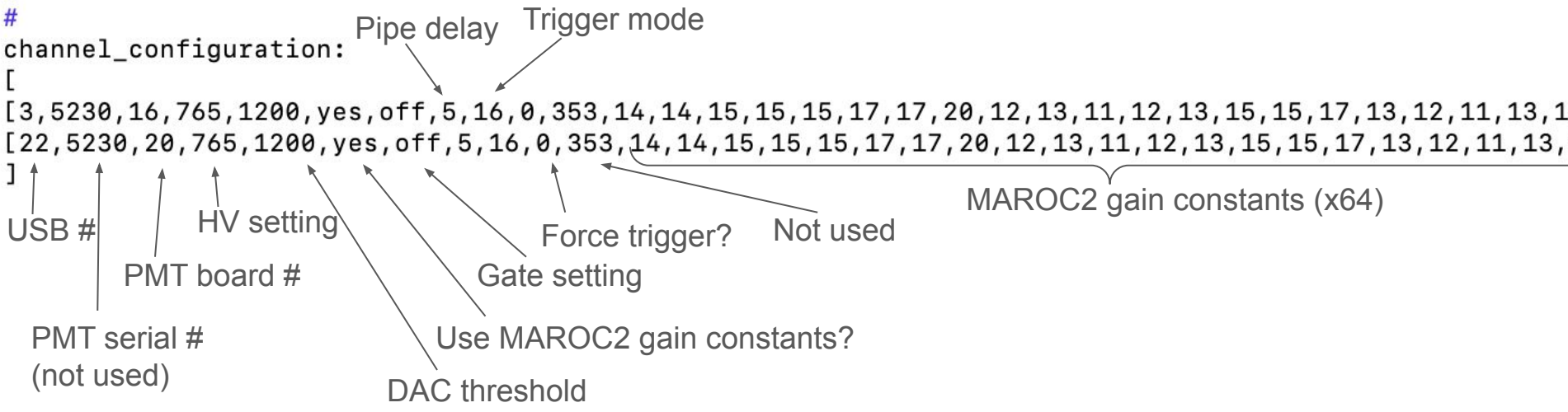
To start the CRT taking data, we invoke a backend process (compiled as a cc code using a makefile) such as `startallboards.cc`, which performs several operations:

- Starts the USB readout processes (4 in the case of the protoDUNE CRT), so data and controls will flow along the daisy-chain of PMT boards
- Loads configuration settings for each pmt board from a provided fcl file (example on next slide)
- configure and initialize all the pmt boards (takes baseline data, applies calibration, threshold and trigger settings)
- Calculates the baseline from data taken above and store them in a text file (used by the readout processor in DUNE DAQ)
- Starts taking data and store data into binary files (one per USB)

There is a parallel `stopallboards.cc` to shut down the readout and stop the run (and optionally provide various statistics and debugging plots)

fcl configuration file example

```
#USB_serial,PMT_Serial,board_number,HV,DAC_threshold,use_maroc2gain,gate,pipedelay,trigger_mode,force
14,gain15,gain16,gain17,gain18,gain19,gain20,gain21,gain22,gain23,gain24,gain25,gain26,gain27,gain28,
3,gain44,gain45,gain46,gain47,gain48,gain49,gain50,gain51,gain52,gain53,gain54,gain55,gain56,gain57,g
#
#
channel_configuration:
[
[3,5230,16,765,1200,yes,off,5,16,0,353,14,14,15,15,15,17,17,20,12,13,11,12,13,15,15,17,13,12,11,13,14
[22,5230,20,765,1200,yes,off,5,16,0,353,14,14,15,15,15,17,17,20,12,13,11,12,13,15,15,17,13,12,11,13,1
]
```



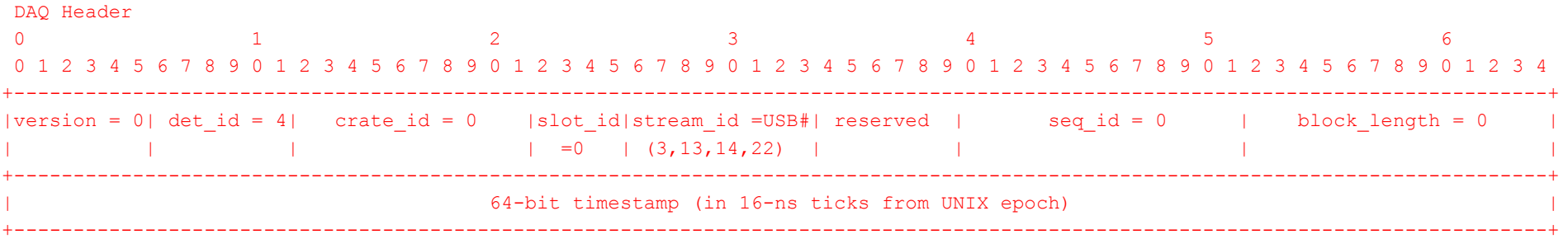
Timing

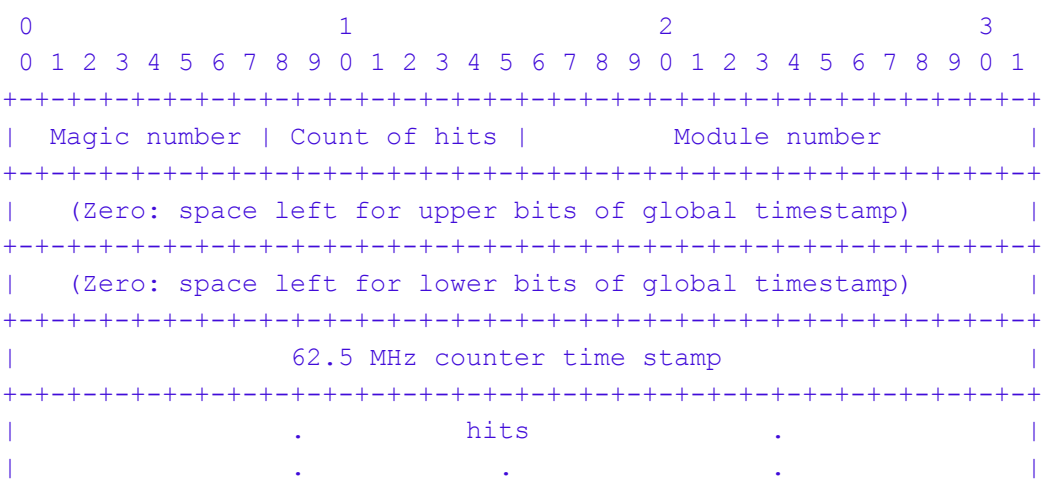
- Each board receives a 62.5 MHz clock signal which increments a 32-bit counter on each of the board - these values are reported with each hit as two separate 16 bits words
- Each board also receives a sync signal which resets the counter to 0 at a fixed interval - this is going to be provided by the DUNE timing board. We want to have the sync at fixed intervals >7s (sync with the timing board). We use a dual timer to veto all but every nth pulse, so the sync can be delivered every n seconds with n > 7 s
-The USB readout additionally injects in the data stream a packet with the time since UNIX epoch, in seconds (so-called t-packet)

Procedure to set the run start time:
1. Start the sync based on a global run start signal
2. Wait until we read the first hit where we have a UNIX time packet
3. The run start time is then: UNIX time - 32-bit timestamp - cable delays
From this point we have a consistent n-second sync that we can use to construct a full 64 bits timestamp

Reading the backend binary files

While the backend is running, the program `CRTInterface` watches the binary files, and the FillBuffer function returns, one at a time, CRT hits formatted as follows:





Each hit consists of:
Magic number `char` (= ‘H’)
Channel number `uint8_t` (= 0-63)
ADC value `int16_t`

To create a fixed-size format, this output is padded with zeros to the maximum possible size
(64 channels = 288 bytes)

Triggering

Each board sends a signal to the CTB for each hit it reads out;

These signals have all been verified and are currently active when the CRT runs.

The signal is formed as an OR of the 64 channels for each of the modules.

We have 32 of those signals used as input in the CTB.

Current progress on DUNE DAQ integration

Data format: `CRTFixedSizeFrame.hpp`*
Type adapter and frame processor

Added a fake readout mode in fdreadoutmodules (1 2)

This mode is able to take a test file generated by CRTInterface and successfully run the DAQ to produce an HDF5 output file.
A rough pass at a decoder seems to indicate that the HDF5 file contains the data in the format expected. However it seems to start reading from somewhere in the middle of the test file - maybe this is expected with the fake data reader mode?

All of my changes have been committed to a new `mmatt15/crt` branch in their respective repositories

*Initially I attempted both this fixed-sized mode as well as a variable-sized frame based on the PACMAN format - the fixed-size model has been easier for me to work with, and if it proves acceptable I can remove the other method and simplify the naming

Configuration files

crt_dro_map.json

```
[
  {
    "src_id": 100,
    "geo_id": {
      "det_id": 4,
      "crate_id": 0,
      "slot_id": 0,
      "stream_id": 39
    },
    "kind": "eth",
    "parameters": {
      "protocol": "udp",
      "mode": "fix_rate",
      "rx_iface": 0,
      "rx_host": "localhost",
      "rx_pcie_dev": "0000:00:00.0",
      "rx_mac": "00:00:00:00:00:00",
      "rx_ip": "0.0.0.0",
      "tx_host": "localhost",
      "tx_mac": "00:00:00:00:00:00",
      "tx_ip": "0.0.0.0"
    }
  }
]
```

daqconf.json

```
{
  "boot": {
    "use_connectivity_service": true,
    "start_connectivity_service": true,
    "connectivity_service_host": "localhost",
    "connectivity_service_port": 15432
  },
  "daq_common": {
    "data_rate_slowdown_factor": 1
  },
  "detector": {
    "clock_speed_hz": 62500000
  },
  "readout": {
    "use_fake_cards": true,
    "default_data_file": "/nfs/home/madmurph/testdata/testcrtdata_fixed.txt"
  },
  "trigger": {
    "trigger_window_before_ticks": 1000,
    "trigger_window_after_ticks": 1000
  },
  "hsi": {
    "random_trigger_rate_hz": 1.0
  }
}
```

To-Do: DAQ

-DUNE DAQ application module to start and stop the backend processes and run the CRTInterface

-Timestamp construction: could be implemented into the CRTInterface, or as a separate piece of the application module

-Run control and configuration

-Cleanup of backend binary files from the CRT computer

To-Do: Monitoring

Our initial idea is to use Grafana to provide monitoring of a few metrics:

-Data rate and average ADC value of each channel

-Timestamp checks (make sure each board receives the sync and that timestamps agree across all modules)