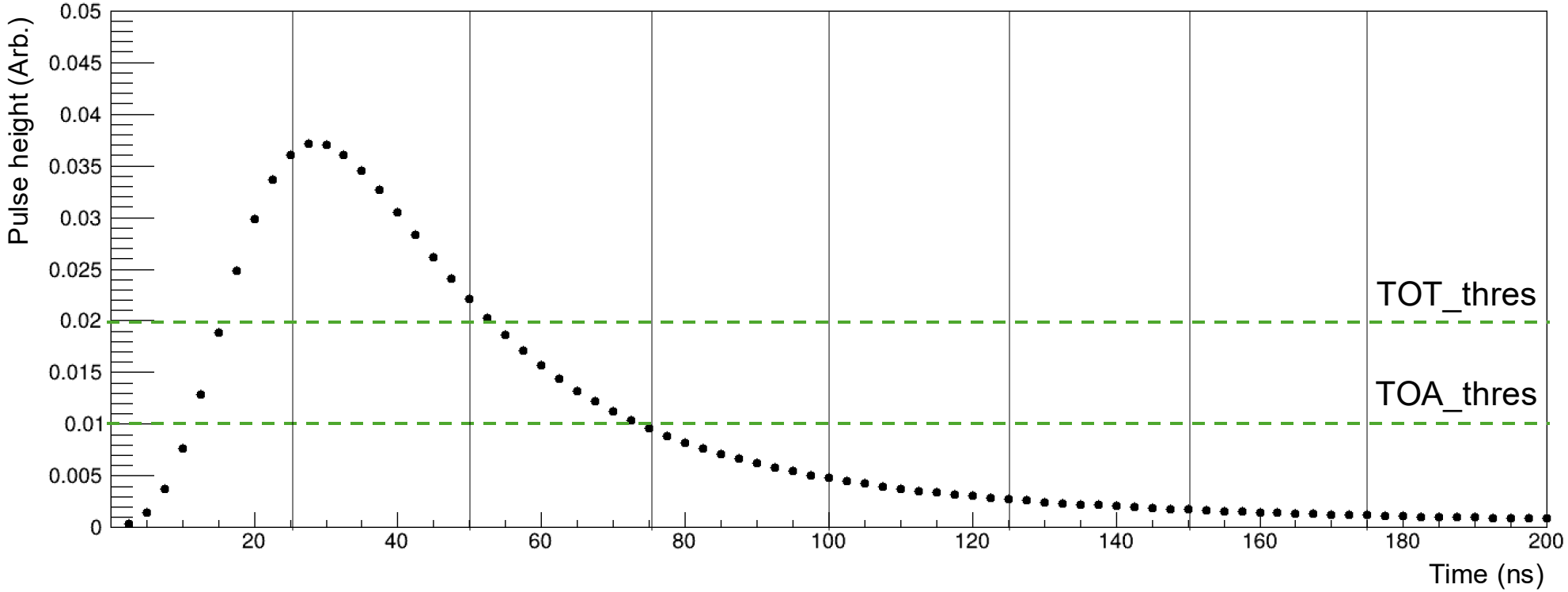
Discussion on the digitization algorithm and PRs that haven’t been merged yet

Minho Kim
Argonne National Laboratory

BIC Simulation Meeting
October 7, 2025

Argonne NATIONAL LABORATORY

HGCROC measurement



ADC	X	Nonzero	Nonzero	Nonzero	Nonzero	Nonzero	Nonzero	Nonzero
TOA	Nonzero	0	0	0	0	0	0	0
TOT	Nonzero	0	0	0	0	0	0	0

- TOA and TOT are stored in the same sample. In this sample, we can't trust the ADC sample.
- In each sample, ADC is measured with the same phase space. → Can't fill the HGCROCsample in real time. → A class was defined to hold the precursor values to ADC, TOA, and TOT.

HGCROCRawSample class

```
class HGCROCRawSample {
public:
  HGCROCRawSample(std::size_t n_meas)
      : meas_types(n_meas, 0), amplitudes(n_meas, 0), TOAs(n_meas, 0), TOTs(n_meas, 0) {}

  void setAmplitude(std::size_t idx, double amp) { amplitudes[idx] = amp; }
  void setTOA(std::size_t idx, double toa) {
    meas_types[idx] = 1;
    TOAs[idx]       = toa;
  }
  void setTOT(std::size_t idx, double tot) {
    meas_types[idx] = 2;
    TOTs[idx]       = tot;
  }

  const std::vector<uint8_t>& getMeasTypes() const { return meas_types; }
  const std::vector<double>& getAmplitudes() const { return amplitudes; }
  const std::vector<double>& getTOAs() const { return TOAs; }
  const std::vector<double>& getTOTs() const { return TOTs; }

private:
  // meas_type 1: Pulse crossed below the threshold.
  // meas_type 2: Pulse didn't cross the threshold.
  // meas_type 0: Neither of the above cases.
  std::vector<uint8_t> meas_types;
  std::vector<double> amplitudes;
  std::vector<double> TOAs;
  std::vector<double> TOTs;
};
```

Initialized with number of samples.

getters

setters

member variables

- Might need more optimization regarding the member variables.

Digitization algorithm

```
for (const auto& pulse : *in_pulses) {
  double pulse_t     = pulse.getTime();
  double pulse_dt    = pulse.getInterval();
  std::size_t n_amps = pulse.getAmplitude().size();

  // Estimate the number of samples.
  const std::size_t timeIdx_begin =
      static_cast<std::size_t>(std::floor(pulse_t / m_cfg.time_window));
  const std::size_t timeIdx_end = static_cast<std::size_t>(
      std::floor((pulse_t + (n_amps - 1) * pulse_dt) / m_cfg.time_window));

  HGCROCRawSample raw_sample(timeIdx_end - timeIdx_begin + 1);

  int sample_tick = std::llround(m_cfg.sample_period / pulse_dt);
  int adc_counter =
      std::llround((timeIdx_begin * m_cfg.sample_period + m_cfg.adc_phase - pulse_t) / pulse_dt);

  bool tot_progress  = false;
  bool tot_complete  = false;
  std::size_t toaIdx = 0;

  for (std::size_t i = 0; i < n_amps; i++) {
    double t = pulse_t + i * pulse_dt;
    const std::size_t sampleIdx =
        static_cast<std::size_t>(std::floor(t / m_cfg.sample_period)) - timeIdx_begin;
```

Calculate the number of samples

Construct an HGCROCRawSample object.

Declare adc_counter to measure the amplitude every 25 ns.

Declare toaIdx to write the TOT to the sample where the TOA is stored.

Digitization algorithm

```
      adc_counter++;
      // Measure amplitudes
      if (adc_counter == sample_tick) {
        raw_sample.setAmplitude(sampleIdx, pulse.getAmplitude()[i]);
        adc_counter = 0;
      }

      // Measure crossing point for TOA
      if (!tot_progress && pulse.getAmplitude()[i] > m_cfg.toa_thres) {
        toaIdx = sampleIdx;
        raw_sample.setTOA(sampleIdx,
                          get_crossing_time(m_cfg.threshold, t, pulse_dt, pulse.getAmplitude()[i],
                                            pulse.getAmplitude()[i - 1]));
        tot_progress = true;
        tot_complete = false;
      }

      // Measure crossing point for TOT
      if (tot_progress && !tot_complete && pulse.getAmplitude()[i] < m_cfg.threshold) {
        raw_sample.setTOT(toaIdx,
                          get_crossing_time(m_cfg.threshold, t, pulse_dt, pulse.getAmplitude()[i],
                                            pulse.getAmplitude()[i - 1]));
        tot_complete = true;
        tot_progess  = false;
      }
    }
  }
} // PulseDigi:process
```

Measure amplitude every 25 ns

sampleIdx of the TOA is assigned.

TOA measurement

sampleIdx of the TOA is used.

TOT measurement

- To fill the actual ADC, TOA, and TOT values, dynamic ranges of them should be studied. → They will be studied using October campaign sample and the digitization algorithm will be complete.

Unmerged PRs

Plugging in Pulsecombiner and Pulsenoise for BIC #2076

Open mhkim-anl wants to merge 11 commits into `main` from `2075-plugging-in-pulsecombiner-and-pulsenoise-for-bic`

- It would be great if the EICrecon PR #2076 could be merged before the October campaign to study dynamic ranges of ADC, TOA, and TOT values.

Add LightGuide and EndOfSectorBox lengths #956

Open mhkim-anl wants to merge 3 commits into `main` from `954-add-lengths-for-bic`

- A new PR that uses the new light guide positions for attenuation will be submitted right after the ePIC PR #956 is merged.