DEEP UNDERGROUND NEUTRINO EXPERIMENT

Ciemat

CERN

OpHitFinder bug

P. Barham Alzás 27/11

The problem

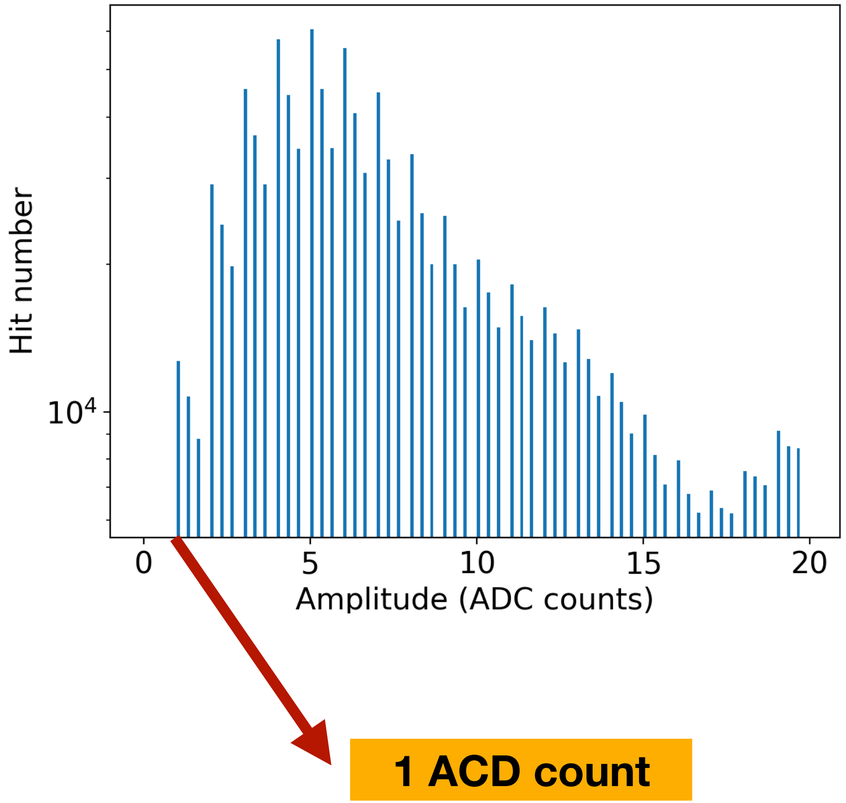
Step 1: Simulate light propagation for any number of events/backgrounds (Generation -> G4 -> DetSim).

Step 2: Run the standard `reco` fhicl (e.g `standard_reco_dunevd10kt_1x8x14_3view_30deg_light.fcl` for the FD-VD) setting an ADC threshold for the hit finder algorithm:

```
physics.producers.ophit10ppm.HitAlgoPset.ADCThreshold: 15 # 15 is "standard"
physics.producers.ophit10ppm.SPEArea: 130
```

Step 3: Look at the hit amplitudes…

We should only have hits above 15 ADC!

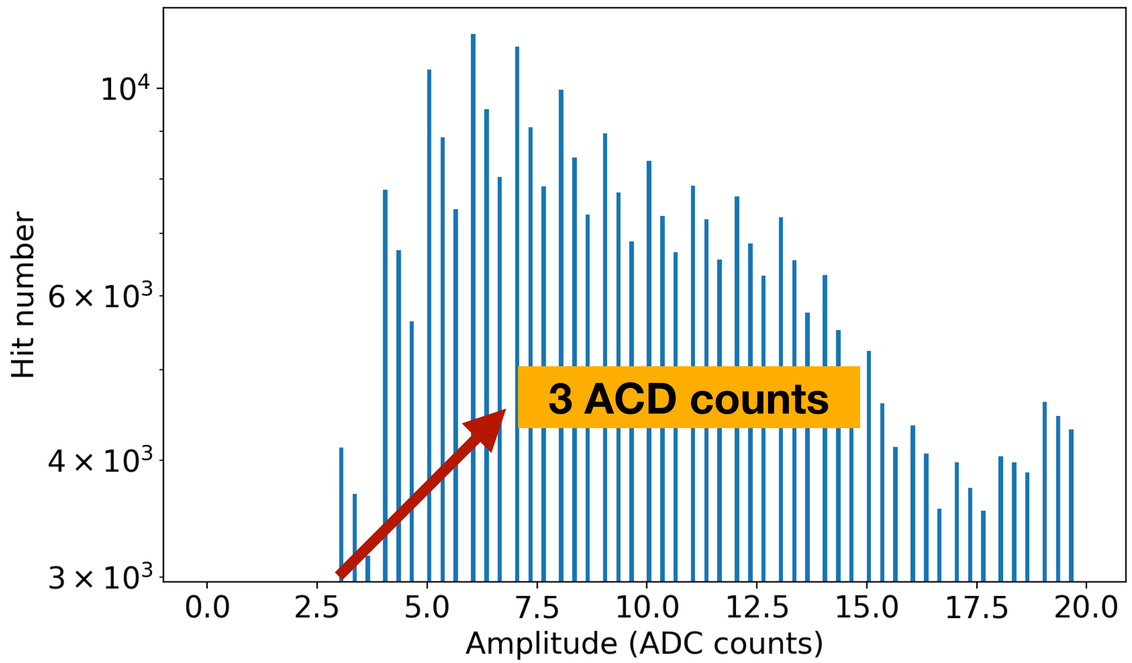


The problem

```
ophit10ppm: {
   AreaToPE: true
   ChannelMasks: []
   GenModule: "generator"
   HitAlgoPset: {
      ADCThreshold: 15
      MinWidth: 60
      Name: "SiPM"
      Pedestal: 1500
      SecondThreshold: 1
   }
   HitThreshold: 15
   InputLabels: [
      ""
   ]
   InputModule: "opdigi10ppm"
   PedAlgoPset: {
      Method: 0
      Name: "Edges"
      NumSampleFront: 3
      NumSampleTail: 3
   }
   SPEArea: 130
   SPEShift: 4.3e-1
   UseCalibrator: false
   module_type: "OpHitFinder"
   reco_man: {
      module_type: "PulseRecoManager"
   }
}
```

fhicl-dump of the reco file

If we change the value of “SecondThreshold”, eg:
SecondThreshold: 3



Hits are being recorded with amplitudes above “SecondThreshold”, and not above “ADCThreshold” as they should.

Code organisation

larana/larana/OpticalDetector/
OpHitFinder_module.cc

```
pmtana::PulseRecoManager fPulseRecoMgr;
std::unique_ptr<pmtana::PMTPulseRecoBase> const fThreshAlg;
std::unique_ptr<pmtana::PMTPedestalBase> const fPedAlg;
```

```
namespace opdet {

  //-----------------------------------------------------------------------
  // Constructor
  OpHitFinder::OpHitFinder(const fhicl::ParameterSet& pset)
    : EDProducer{pset}
    , fPulseRecoMgr()
    , fThreshAlg{art::make_tool<opdet::IHitAlgoMakerTool>(makeHitAlgoToolConfig(pset))->makeAlgo()}
    , fPedAlg{art::make_tool<opdet::IPedAlgoMakerTool>(makePedAlgoToolConfig(pset))->makeAlgo()}
  {
```

```
fhicl::ParameterSet toolConfig = makeAlgoToolConfig(baseConfig, "HitAlgoPset", "Algo");
```

Code organisation

larana/larana/OpticalDetector/
OpHitFinder_module.cc

```
//-----------------------------------------------------
void OpHitFinder::produce(art::Event& evt)
{
```

```
      RunHitFinder(WaveformVector,
                   *HitPtr,
                   fPulseRecoMgr,
                   *fThreshAlg,
                   geometry,
                   fHitThreshold,
                   clock_data,
                   calibrator,
                   fUseStartTime);
    }
    // Store results into the event
    evt.put(std::move(HitPtr));
  }
```

larana/larana/OpticalDetector/
OpHitFinder/OpHitAlg.cxx

```
//-----------------------------------------------------------------------
void RunHitFinder(std::vector<raw::OpDetWaveform> const& opDetWaveformVector,
                  std::vector<recob::OpHit>& hitVector,
```

```
      pulseRecoMgr.Reconstruct(waveform);

      // Get the result
      auto const& pulses = threshAlg.GetPulses();

      const double timeStamp = waveform.TimeStamp();

      for (auto const& pulse : pulses)
        ConstructHit(hitThreshold,
                     channel,
                     timeStamp,
                     pulse,
                     hitVector,
                     clocksData,
                     calibrator,
                     use_start_time);
    }
```

Code organisation

larana/larana/OpticalDetector/
OpHitFinder/PMTPulseRecoManager.cxx

```
//***************************************************************************
bool PulseRecoManager::Reconstruct(const pmtana::Waveform_t& wf) const
//***************************************************************************
{
```

larana/larana/OpticalDetector/
OpHitFinder/PMTPulseRecoBase.cxx

```
//***************************************************************************
bool PMTPulseRecoBase::Reconstruct(const Waveform_t& wf,
                                   const PedestalMean_t& mean_v,
                                   const PedestalSigma_t& sigma_v)
//***************************************************************************
{
  _status = this->RecoPulse(wf, mean_v, sigma_v);
  return _status;
}
```

larana/larana/OpticalDetector/
OpHitFinder/AlgoSiPM.cxx

The bug origin

larana/larana/OpticalDetector/OpHitFinder/AlgoSiPM.cxx

```
bool AlgoSiPM::RecoPulse(const pmtana::Waveform_t& wf,
                         const pmtana::PedestalMean_t& ped_mean,
                         const pmtana::PedestalSigma_t& ped_rms)
{
```

```
_adc_thres = pset.get<float>("ADCThreshold");
_min_width = pset.get<float>("MinWidth");
_2nd_thres = pset.get<float>("SecondThreshold");
_pedestal = pset.get<float>("Pedestal");
```

Unused

```
for (short const& value : wf) {

  if (!fire && (double(value) >= pre_threshold)) {

    // Found a new pulse
    fire = true;
    first_found = false;
    record_hit = false;
    _pulse.t_start = counter;
  }

  if (fire && (double(value) < pre_threshold)) {

    // Found the end of a pulse
    fire = false;
```

```
double threshold = _adc_thres;
threshold += pedestal;
double pre_threshold = _2nd_thres;
pre_threshold += pedestal;
```

```
if (fire) {

  // We want to record the hit only if _adc_thres is reached
  if (!record_hit && (double(value) >= threshold)) record_hit = true;
```

The bug origin

`larana/larana/OpticalDetector/OpHitFinder/AlgoSiPM.cxx`

```
bool AlgoSiPM::RecoPulse(const pmtana::Waveform_t& wf,
                         const pmtana::PedestalMean_t& ped_mean,
                         const pmtana::PedestalSigma_t& ped_rms)
{
```

```
  if (fire) {

    // We want to record the hit only if _adc_thres is reached
    if (!record_hit && (double(value) >= threshold)) record_hit = true;
```

```
if (!first_found && (_pulse.peak < (double(value) - double(pedestal)))) {

  // Found a new maximum
  _pulse.peak = (double(value) - double(pedestal));
  _pulse.t_max = counter;
}
else if (!first_found)
  // Found the first peak
  first_found = true;
```

Only the first peak of each pulse is recorded.

If we have a higher peak later going above `threshold`, it will set `record_hit` to “`true`” but not change the value of `_pulse.peak`, recording the hit with an incorrect amplitude/peak time.

This leads many recorded hits to appear below threshold.

Does this affect you?

This **does not** affect you if:

- You have only been “counting hits” (i.e., how many hits I get per ARAPUCA/event, etc).

This **does** affect you if:

- You have been using hit properties for classification/analysis (hit amplitude, peak time). The area and PEs of the hit seem correctly calculated (if you are using the area to compute the PEs).
- You have done any posterior cut on the hit amplitudes (either using the `physics.producers.ophit10ppm.HitThreshold` parameter in the `reco` fhicl or “by hand” in the analysis).

Easy fix

If we don’t care about the value of the first peak, we can just remove the `first_found` condition.

```
if (!first_found && (_pulse.peak < (double(value) - double(pedestal)))) {

  // Found a new maximum
  _pulse.peak = (double(value) - double(pedestal));
  _pulse.t_max = counter;
}
else if (!first_found)
  // Found the first peak
  first_found = true;
```

```
if (_pulse.peak < (double(value) - double(pedestal))) {

  // Found a new maximum
  _pulse.peak = (double(value) - double(pedestal));
  _pulse.t_max = counter;
}
```

(If we do and want to keep it, we just add an extra condition)