‘Timing in’ simulation and data

Wesley Ketchum (FNAL)

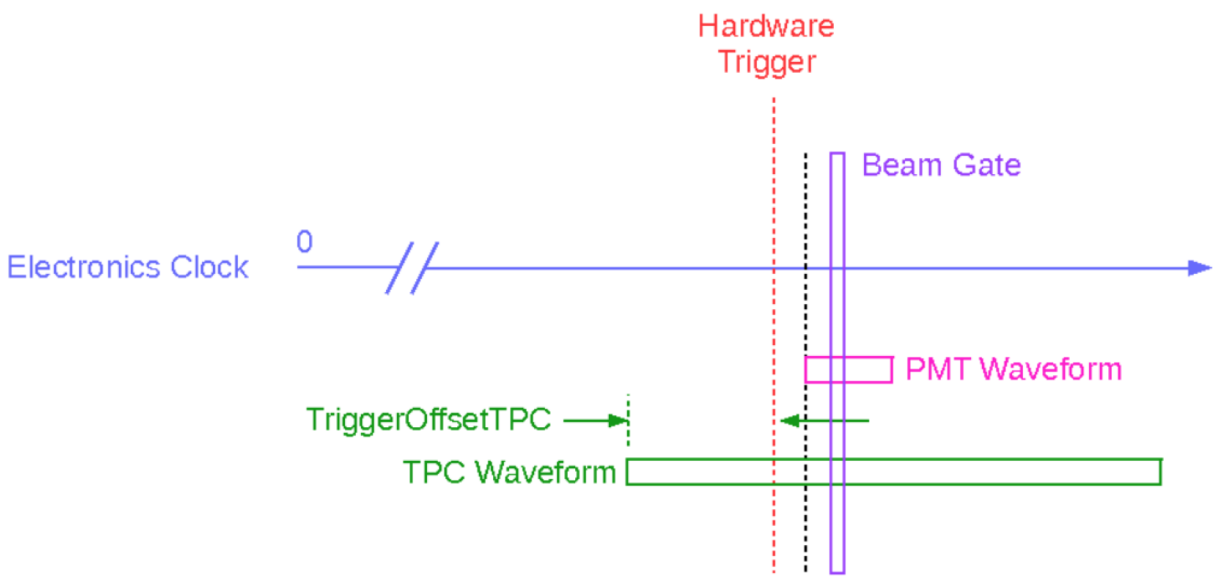
The problem (at least in MicroBooNE…)

- Generally we have two separate and usually self-consistent worlds to consider for overlays:
  - Data
  - Simulation

- When it comes to timing:
  - Data times typically center around real times or counters referenced against the beginning of a run or PPS or …
  - Simulation times typically the same for every event (except maybe a generation time), with known/consistent offsets event-to-event

MicroBooNE data timing diagram

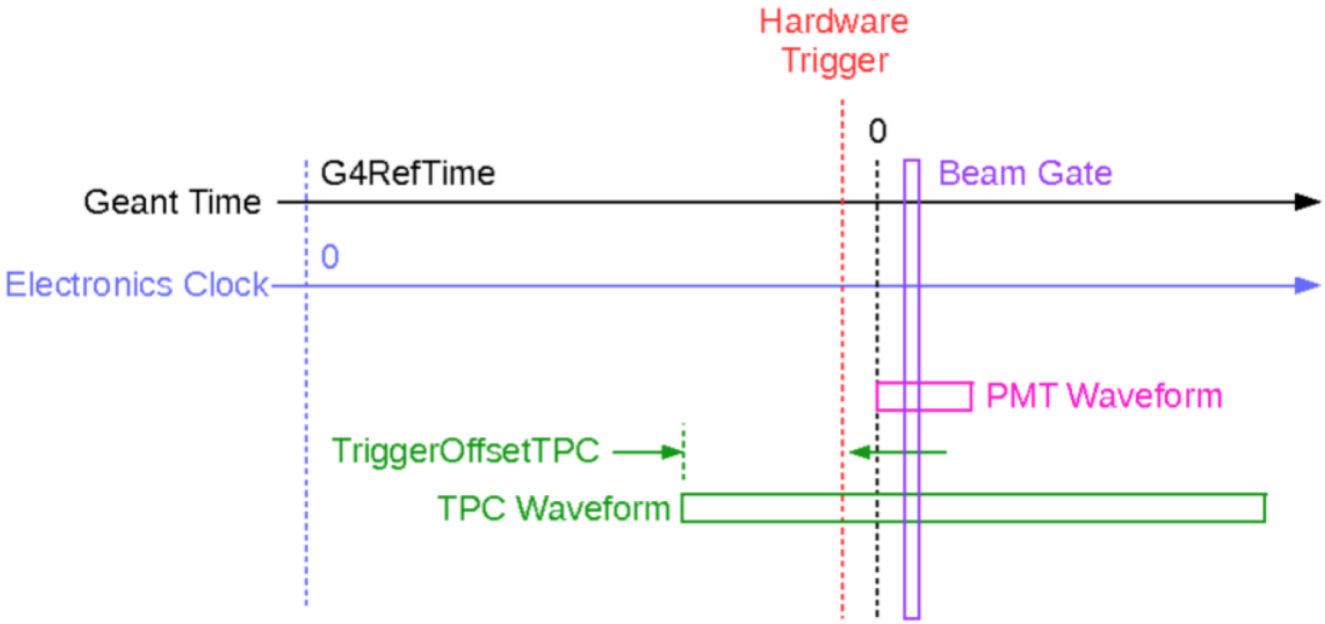
- Every event assigned the real time at which it occurred
- Electronics clock starts from ‘0’ at the beginning of a run
- A readout trigger arrives at some point later
  - TPC readout induced (including readout of buffered data)
  - Some ‘unbiased’ PMT waveform readout occurs a fixed (short) time after that trigger [which is unfortunately slightly different for on-beam and off-beam]



H. Greenlee, MicroBooNE DocDB 12290

MicroBooNE simulation timing diagram

- Every event assigned the time at which it was generated
- Truth particle time zero-point aligned with the beginning of PMT waveform
- Electronics clock 0 at constant offset relative to that time
  - Same for every event
- Otherwise, timing all consistent with data



H. Greenlee, MicroBooNE DocDB 12290

Symmetry in overlays

- When overlaying data, we have a primary (source) event onto which we add a secondary event

- Will generally follow the timing conventions for whichever is the primary, and adjust the secondary to match
  - And, not, other ‘art’ conventions, since we cannot modify the art::Event

- Two ways this can typically go for overlays for which we have to make a choice
  - Simulation as source, data as secondary
  - Data a source, simulation as secondary

Symmetry breaking in MicroBooNE

- In MicroBooNE we eventually chose to use data as primary and simulation as ‘secondary’

- Why?
  - Production reasons: it’s easy to run simulation on top of a data event and then add waveforms together later (no need for EventMixing apparatus to read a secondary file)
  - Conditions/calibration reasons: we have time-dependent detector conditions and want to ensure the same apparatus for accessing/applying conditions data

- Note that we still have the code to use the EventMixing framework in art to read in data from a secondary file, but it’s not used

(Quick aside on this choice)

- Using evt.isRealData() is great …

- Except when it's not
  - If we simulate on top of a "real data" art::Event, then we inherit this as 'true' throughout
  - If we use this to flag whether or not to try to compare to truth → doesn't work for overlays

- Code now peppered with 'OverrideRealData' flags in fcl configurable to account for this
  - This may be better achieved with something in the art::Event itself?

Once deciding that …

- Need to ensure everything in simulation gets updated to match timing of data

- (1) Updating (creating new…) trigger in the data object so it looks like neutrinos (https://cdcvs.fnal.gov/redmine/projects/ubevt/repository/revisions/v08_00_00_br/entry/ubevt/DataOverlay/DataOverlayMixer/OverlayRawDataMicroBooNE_module.cc#L307)

```
306
307 bool mix::OverlayRawDataMicroBooNE::MixTriggerData( const art::Event& event, std::vector<raw::Trigger> & output) {
308
309   output.clear();
310
311   art::Handle< std::vector<raw::Trigger> > mcTriggerHandle;
312   event.getByLabel(fTriggerMCModuleLabel, mcTriggerHandle);
313
314   art::Handle< std::vector<raw::Trigger> > dataTriggerHandle;
315   event.getByLabel(fTriggerDataModuleLabel, dataTriggerHandle);
316
317   unsigned int trig_num; double trig_time,gate_time; unsigned int trig_bits;
318   trig_num  = dataTriggerHandle->at(0).TriggerNumber();
319   trig_time = dataTriggerHandle->at(0).TriggerTime();
320   gate_time = dataTriggerHandle->at(0).BeamGateTime();
321   trig_bits = dataTriggerHandle->at(0).TriggerBits() | mcTriggerHandle->at(0).TriggerBits();
322   output.emplace_back(trig_num,trig_time,gate_time,trig_bits);
323
324   return true;
325 }
```

And once doing that?

- Generally, all of the timing pieces are handled via the DetectorClocksService

- Generally, times are all referenced against a trigger time, with appropriate offsets tuned via fcl parameters
  - Where the trigger time is read in from a trigger object on the event, whose module label is fcl configurable

- Ideally then, everything should ‘just work’ with the proper configurations…

Living in an unideal world

- (2) Ideally, there's no other differences between data object timing with respect to a trigger, but if there are it must be accounted for

- In MicroBooNE, as I mentioned before, has this problem
  - The start-time of the unbiased PMT waveform, relative to the trigger signal, is a bit different between on-beam (what we want to mimic) and off-beam (the data upon which we overlay) data
  - We handle these additional offsets in fcl configuration of optical reconstruction and trigger conditions

Comparison back to simulation

- (3) Last step is to ensure timing offsets to simulation are properly accounted for

- There's two major issues MicroBooNE has encountered here:
  - Working with the (TPC) BackTracker
  - Direct comparisons of MCParticle time

  - (Note, we don't use in detail the OpDetBackTracker, so there could be additional issues there…)

Let's start with MCParticle time

- Remember, for MicroBooNE
  - In simulation, TriggerTime is a fixed time: it's the same for every event, even if the art::Event's time is not the same
  - In data, TriggerTime is a time since the start of a run
    - Which is not the same for every event!

- And, to match TPC objects to MCParticles, we use

```
/// Given G4 time returns electronics clock count [tdc]
virtual double TPCG4Time2Tick(double g4time) const override
{ return (G4ToElecTime(g4time) - doTPCTime()) /
fTPCClock.TickPeriod(); }
```

The problem, in specific

- That function tries to take G4 particles and find where they should be in TPC ticks

- Underneath…
  - `G4ToElecTime(g4time) = g4_time * 1.e-3 - fG4RefTime`
  - `doTPCTime() = TriggerTime() + TriggerOffsetTPC();`

- In uboone simulation, G4RefTime (coming from fcl) and TriggerTime() (coming from a trigger object created in a "triggersim" module) are tuned to work correctly together

- This breaks down if we inherit the trigger time in another way
  - Which we do from data in the overlays

Modifying G4RefTime

- To work with this, we've allowed ability to modify the G4RefTime on an event-by-event basis
  - Luckily, we have a good example for a mechanism to adjust DetectorClocks event-by-event: the trigger time
  - New function added:
    - setDetectorClocksStandardG4RefTimeCorrectionFromEvent (and the like)

```
virtual void RebaseG4RefTime(double sim_trig_time)
{ fG4RefTime = fG4RefTimeDefault - TriggerTime() +
sim_trig_time; }
```

  - Note, we keep the fcl as the "G4RefTimeDefault" so we can always recalculate against that
  - TriggerTime() here is from our data trigger module
  - sim_trig_time is a second trigger time read in from a new G4RefCorrTrigModuleName fcl parameter
    - By default, if not set or not there, will not call the rebase function
- So, with proper fcl configuration, this function will work…

Now, to the BackTracker issues

- The BackTracker relies on matching from the ‘tick’ times:

```
  const std::vector< sim::TrackIDE > BackTracker::ChannelToTrackIDEs(raw::ChannelID_t channel, const
double hit_start_time, const double hit_end_time) const{
    std::vector< sim::TrackIDE > trackIDEs;
    double totalE=0.;
    try{
      art::Ptr<sim::SimChannel> schannel = this->FindSimChannel(channel);

      // loop over the electrons in the channel and grab those that are in time
      // with the identified hit start and stop times
      int start_tdc = fDetClocks->TPCTick2TDC( hit_start_time );
      int end_tdc   = fDetClocks->TPCTick2TDC( hit_end_time   );
      if(start_tdc<0) start_tdc = 0;
      if(end_tdc<0) end_tdc = 0;
      std::vector<sim::IDE> simides = schannel->TrackIDsAndEnergies(start_tdc, end_tdc);
```

  - `doTPCTime() = TriggerTime() + TriggerOffsetTPC();`

- No G4RefTime here → our trick for the MCParticles won’t work

How to handle this?

- MicroBooNE already does its BackTracker just once, in production jobs…
  - Stores Hit ←→MCParticle associations, and drops the SimChannels
  - We did this to save space in our files/reduce data I/O times for production

- So, we can hack things by running BackTracking in its own step
  - BackTracking code (basically) works out of the box using trigger object defined from simulation
  - →Special "reco" stage dedicated just to BackTracking (see next talk on production chain)

At this point you may be thinking…

Discussion: how do we improve things for the future?

- New experiments should think early about consistent definitions of timing between data and simulation
  - Expect you will need to merge these!

- This likely points to better definitions of ‘TriggerTime()’ and of setting of the MCParticle time in G4
  - The latter may be handled well and stably with the additional G4RefTime correction?
  - Does there need to be an associated correction to TriggerOffsetTPC() to handle BackTracking case?
    - And does that propagate through to *all* electronics clocks

- I don’t have an explicit proposal for this, but it seems to me there should be a better way to handle this within the clocks service

- IDEAS WELCOME