

Summary of discussion on Run-Starts at LBNE DAQ workshop ****DRAFT****

Introduction

This document describes the steps needed to start a run in the 35t test to ensure that the data are synchronised between different components (the 16 RCEs, 8 SSPs and the Penn board). This technique relies on the individual counters and state machines in the front end electronics (cold boards and SSPs) to be synchronised at initialisation (at least after a power-up restart, or perhaps more frequently) and that is described in a companion note docdb-9xxx.

Principle

An LBNE *millislice* (the same thing as an artDAQ event) to collect all the digitised signals from each component between well defined start and end times and then let the artDAQ EventBuilder merge them. The millislice time interval is many (e.g. 10) drift-times and there is an overlap of at least one drift-time (1.3ms) to cope with physics events near the boundary. There is a concept of *microslice* which could be different for each sub-detector and is described in more detail in each section below. The millislice length and overlap length are programmable (e.g. when we run at low voltage, the drift time gets longer) but are both integer multiples of the microslice length.

[Please correct these guesses I have made of the artDAQ state names here: init-ready-running] The principle of operation is that the synchronisation described in docdb-9xxx is done as the board goes from the init to ready state (or it may be done less frequently, when entering the init state). Then, when RC requests to DAQINTERFACE to go from ready to running, the request includes the run number and a time in the near future at which the run should start. (It has not yet been decided whether RC or DAQINTERFACE should consult the NOvA timing unit to get the current time in order to add a configurable offset). When the boardreader receives the start run request, it asks the hardware to start sending data, and starts collecting the data into millislices. It throws away the millislices until it receives the first one which starts after the time in the run start message and from this millislice onwards, sends the millislices to artDAQ.

Run-stop is accomplished by sending an expected end time in a similar way in the end of run message. The boardreaders stop sending millislice events from the first one which occurs after the given time after which it begins to throw away the data and it instructs the hardware to stop sending data. artDAQ lets following events get processed through the system before declaring the run finished to RC.

Run pause/resume is implemented in a similar, and it is the decision of each boardreader whether to request the hardware stop sending data or to just throw it away during a pause.

The distance in the future that the start or stop is issued is a programmable parameter. It is possible that the boardreader discovers when it receives such a request that the time in the message may not be in the future, in which case it should assert an error.

The DAQ always starts, stops, pauses and resumes on the boundary of a complete millislice, it never chops the first or last millislice off.

RCE system

The microslice concept is most developed for the RCE system and will be described in more detail here. The RCE hardware sends the data to the boardreader in quanta of microslices. These have a length which is a power of two (likely 32, 64 or 128) of the 2MHz digitisation clock frequency, and start when the NOvA time stamp has its lowest bits set to zero (for a 32MHz LSB on the NOvA time stamp, the 2MHz clock ticks have time stamps where the 4 LSB are always zero. The time stamp of the start of a 128 tick long microslice also has the next 7 LSBs zero). [Check that the NOvA time has an LSB which counts at 32MHz, I think it could be 64MHz]. The board reader makes millislices out of microslices by simply concatenating them, the lengths in the microslice headers can be used to step through the microslices in the millislice. The boardreader does not

need to look in detail at the data in the microslices, it looks at the block counter in the header to detect missing blocks and looks at the time stamp to determine which blocks to drop just before the start and after the end of a run. It may be that a physical pulse spans across two microslices in which case the offline software must do the work of joining them up.

There are four modes of operation for the RCEs in order to limit the amount of data (the bandwidth is not enough to send all the data all the time), and the boardreader operates the same way for all four modes by concatenating the microslices to make millislices. The modes are: scope (send a few channels all the time), burst (send all the channels for some time), triggered (send all the channels when a trigger from the Penn board appears) and continuous (use zero suppression). It may be that a fifth mode is added which is triggered mode but with the zero suppression switched on.

What did we decide was the way of deciding which microslice to use to begin the millislice? I think it was either (1) The millislice starts with the first microslice which happens after the run start time (and if there is a pause/resume, the number of microslices thrown away in between is not necessarily a whole number of millislices) or (2) The millislice boundaries can be calculated by assuming the microslices have been combined into millislices this way all the time since the NOVA time was zero. [So if the millislice length is 128 2MHz ticks and the NOVA time has an LSB of 32MHz then we can define an 'absolute' microslice number since the beginning of the NOVA time epoch of $\text{AbsoluteMicroSlice} = \text{NovaTime} / (7+4)$, and if the length (excluding overlap) of the millislice is N microslices, then we could start the millislice on the first microslice after the run start time for which $(\text{AbsoluteMicroSlice}/N) * N == \text{AbsoluteMicroSlice}$. This uses integer divide. In this case if there is a pause/resume, the number of microslices discarded is a whole number of millislices (which doesn't matter, but is helpful in explaining the difference between these two). I think the advantage of (2) is it allows the microslice to be different in each subsystem (RCE, SSP, Penn board, any new board which comes in the future) while (1) would require care if they are different.

SSP system

The SSP delivers data consisting of headers (containing the NOVA time and pulse height of a SiPM pulse) followed by the waveform digits for a programmable length of time (maximum is about $8\mu\text{s}$). The programmable length of the waveform can be set to zero to just get the pulses and this is referred to as header mode. The SSP does not care about the mode in which the RCE runs (spill, burst, triggered, continuous). It may be possible it can do something with the trigger input from the Penn board, we will have to ask.

The SSP could collect headers and their corresponding waveforms in groups based on the start times in the headers into microslices corresponding to the microslices in the RCE and then combine microslices into millislices in the same way as the RCE as described above. It could also ignore the microslice concept and just make the millislices directly. It will have to do the same treatment as described above for the overlaps. A pulse that has a waveform which is right at the end of the millislice may have digitisations present which stretch beyond the millislice end time (including the overlap) which is OK, because the same pulse will also appear in the next millislice due to the overlapping.

It may be possible to run the SSP in a single mode to get all the information needed for the 35t physics programme, or it may be that separate runs for different studies are needed. This needs more discussions. One example of where it may not be possible to collect all the data in one go is as follows: For certain studies of the TPC data with the 35t, not only will the PDs be required to provide the t_0 of the triggered muon, but the drift window may contain parts of muons which appeared earlier, near the cathode, which are drifting towards the anode plane when the triggered muon occurs; or muons which came later, near the anode plane; the t_0 of these other muons will be required. This includes muons which occurred before the trigger time which makes it a bit tricky. The data can be collected by running in header mode and collecting the times of all the pulses which happen all the time. However, another study may be needed which requires the full digitisation of the pulse shapes from the SiPMs to study late light etc. This may need a separate run where the pulses are collected only close to the trigger time with a long waveform enabled. In this mode, the boardreader should

always include an empty millislice if no pulses arrive and should (if possible) indicate in the data stream which times the photon system was not being read out.

Penn board

The Penn board contains receive-amplifiers for the external counters, trigger logic to allow programmable combinations of counters to produce a trigger and issue it to the SSP and RCE. It also provides a readout stream which is clocked at a rate of (unknown, perhaps 2MHz) which contains the bit patterns of the triggers which were on during that tick. The Penn board contains a Zynq chip and so can provide data in a way similar to the RCE, i.e. in microslices which then get put together into millislices. It is thought that the data rate from the Penn board is small and can be read without zero suppression, but we should do the calculation when we find out the intended clock rate.

The start/stop run for the Penn board can follow that of the other modules. It is a choice whether the sending of triggers on the Penn board is enabled (a) at the same time as the Penn board data sending starts up (i.e. before the designated start time), in this case the other boards may throw away data from some triggers, but only before the designated start time or (b) the Penn board starts the run with triggers disabled, then starts the triggers. There seems to be no advantage/disadvantage of either of these approaches.

artDAQ software trigger

(Sussex group, please comment) Once artDAQ builds millislices, they are then sent to a trigger processing module that identifies interesting areas and decides which portions of which millislices to keep. It is probably easier at first to simply decide to keep or throw away the entire millislice. This will have to be fairly ruthless otherwise the data files will be enormous.

When the trigger is cutting out parts of millislices containing interesting events, it will append new art blocks to the millislice, one for each 'triggered event' and then drop the original information. If two 'triggered events' occur within one millislice, they will not be split up into separate artDAQ events, in this case, the art event will contain the two 'triggered events' as separate parts of one 'art' event'. The trigger is responsible for assigning event numbers to the events it triggers.