DAQ Syncing

Adam West

March 29, 2017

1 Gen. 1

In Generation 1 we had one computer which controlled the state of the experiment and acquired data from the PMTs. A 50 Hz trigger was continuously generated by an SRS DG645 pulse generator, which was referenced against a 200 kHz clock that controlled the polarisation switching. Data was being continuously acquired at 50 Hz from the PMTs by the old PXI modules.

The switching of the state of the experiment ($\tilde{\mathcal{N}}$,$\tilde{\mathcal{E}}$,$\tilde{\theta}$,$\tilde{\mathcal{B}}$,$\tilde{\mathcal{P}}$,$\tilde{\mathcal{G}}$,$\tilde{\mathcal{R}}$) is controlled by the DAQ computer as follows. Labview generates the switch structure for the run (including block and superblock switches) ahead of time. This is then read as the run progresses so that the next state of the experiment can be prepared. Depending on which switches have to be switched the DAQ computer allows a variable amount of 'settling time' for these switches to be carried out. Once this settling time has elapsed the DAQ computer considers the experiment to be in a ready state and sends a signal via Labview to say that the data which is being continuously acquired should be saved to disk.

2 Gen. 2

There are a number of options available to us:

1. Scheme 1:
   Same as Gen. 1 — everything is kept on the same computer and the signal for when the new FPGA-based DAQ acquires and saves data is handled through Labview. Brendon (or I) will probably try putting the new VIs for controlling the experiment on the new DAQ computer to see if everything is happy running at the same time. This is prompted partly by recent observations of certain aspects of experimental control slowing down on the DAQ computer (recall this is also handling logging).

2. Scheme 2:
   Control of the experimental state is handled by the old DAQ computer which then sends a signal to a separate computer which is handling data acquisition from the PMTs. This signal could be either a 'hardware' signal, e.g. a TTL pulse, or could be over the network between computers. I have been working with the former recently.

3. Scheme 3:
   Total hardware control. There is a piece of hardware that generates a purely digital signal to control the experimental state and to trigger the PMT data acquisition. Brendon's recent email pointed out a few problems with this, including the fact that numerous pieces of hardware (e.g. waveplate rotaters, electric field power supplies) are communicated with over RS232, and that there may be some associated loss in duty cycle.

2.1 Scheme 1

2.2 Scheme 2

This is the scheme that I have been trying to set up recently. A little background information: the way the FPGA DAQ currently works is that it is set to continuously acquire data whenever it receives a trigger. After it has acquired 25 traces it averages them together and then saves them. 64 such traces are saved into a single 'block' file. Because we need to allow 'switching time' when the state of the experiment is changed, the 50 Hz trigger that is sent to the FPGA DAQ needs to be modulated by whether the experiment is 'ready' or not.

An outline of how to modulate the trigger sent to the FPGA DAQ is outlined in Figure 1. The basic idea is to

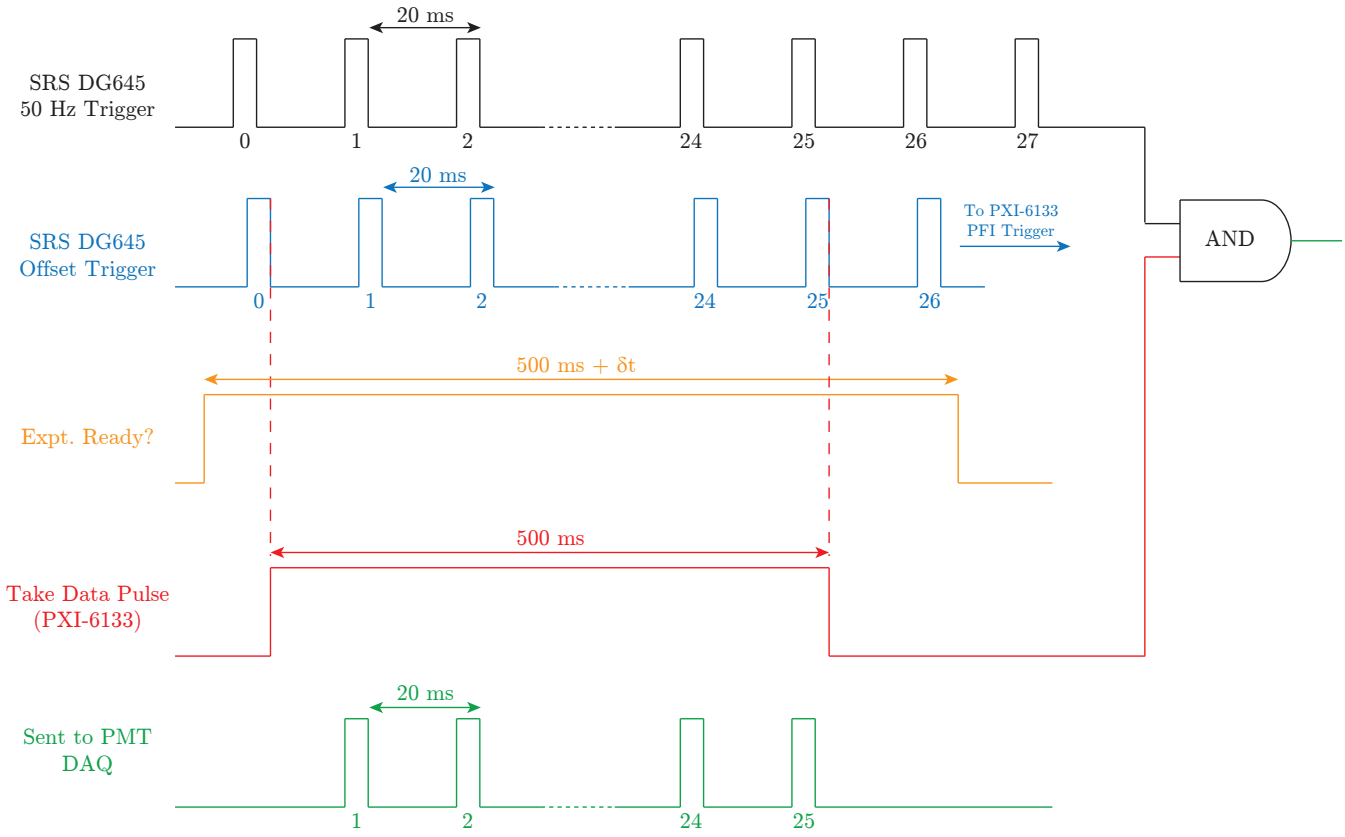

Figure 1: Schematic of proposed timing for synchronising the 'main' DAQ computer with the PMT DAQ computer.

use an AND gate to modulate the constant 50 Hz trigger with a separate TTL signal that represents whether the experiment is 'ready' or not. This pulse should be 500 ms long to a decent precision ($\approx$ 20 ms) to ensure that 25 traces are acquired for each state of the experiment. One could envision an alternative where the Labview running on the PMT DAQ computer only ever takes 25 traces for each state of the experiment, but you would still need some communication to tell it when there is a new 'state'.

The 500 ms pulse is generated by a PXI-6133 multifunction DAQ module. I had some difficulties in getting this to work well — because there is no dedicated clock for the digital part of the device one has to 'borrow' a clock from the Analogue part of the device. After some playing around I eventually got it to work well. The block diagram for the VI is shown in Fig. 2. The timing resolution is at least as good as 1 µs — I didn't investigate further than this, and the VI can be called repeatedly with around 20 ms of dead time between calls. This is faster than any switch in the experiment.

The 500 ms pulse together with the constant 50 Hz trigger results in 25 pulses. There is a small caveat, which is that the 500 ms pulse and the 50 Hz trigger will have varying relative phase, due to the switching of the experimental state. Thus if the rising edge of the 500 ms pulse falls in the middle of one of the trigger pulses you will end up with an extra trigger, giving 26 records acquired, with the first one being mistimed. The solution to this is to use a second output from the DG645 which is an offset 50 Hz pulse train. By using a falling edge of this pulse train to trigger the 500 ms pulse you can guarantee repeatability of only ever generating 25 triggers. The Figure should explain this a lot more clearly than described here.

I have tested this scheme with the FPGA module and found that it works acceptably, i.e. I have generated 64 500 ms pulses, giving 64 sets of 25 triggers, corresponding to an entire block of data. The FPGA module sees the correct number of triggers, and acquires the correct amount of data.

Remaining things to test are to ensure that we can integrate this into the main experimental control VI such that it can effectively perform the necessary switches and correctly create these 500 ms pulses.

2.3 Scheme 3

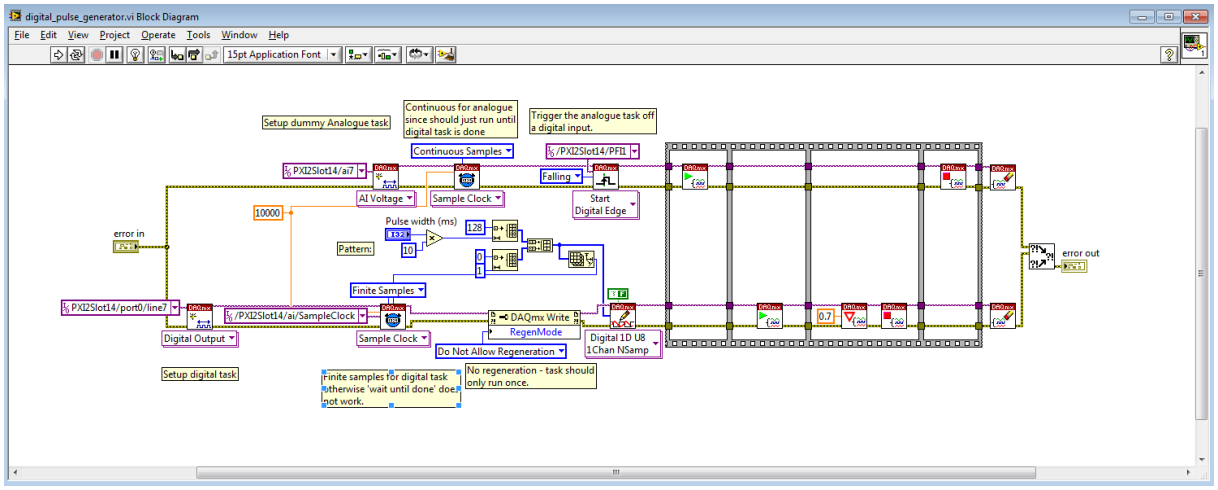

Figure 2: Block diagram of the VI used to generate a digital pulse using the PXI-6133 module.