**Instruction for using the 4πβ(PC)-γ counter using the Radionuclide Activities using Digital Instrumentation and Coincidence/Anticoincidence Logic (RADICAL) System for data acquisition.**

1. Start-up.
   1. It is important that the PC is started up after the PXI chassis is started up, if you are getting errors when you try to run the acquisition that indicate that the computer can’t find the FPGA target (alternatively, if you look in NI’s MAX program and the FPGA isn’t there), you’ll need to restart the computer with the PXI already started.
   2. Do a measurement of the β-channel plateau (the counting can be done using the RADICAL system or a NIM counter).
   3. Set the gamma channel voltage(s) to the recommended voltage.
   4. Turn on the P-10 gas so that there is something like 1 bubble per second (the detector needs some time to purge the ambient air upon start-up or after opening the door).
2. Set up the analogue electronics chain.
   1. Choose an appropriate gain/integration time on both amplifiers.
   2. Set the SCA levels:
      1. Measure the pulse height spectrum and identify where the discrimination level(s) should be.
      2. Connect a pulser to the input of the shaping amplifier.
      3. Adjust the pulse height of a pulser output to match the pulse height associated with the discrimination level(s).
      4. Observe the SCA output and adjust the discrimination level(s) so that the amplified pulser pulse is (just) discriminated.
   3. Connect the SCA outputs to Ch1 and Ch2 of RADICAL.
3. Acquire data.
   1. Open the data acquisition program, CoincCountFPGA\_Host\_EncodedOutput.vi, located within the LabVIEW project CoincCountFPGA.lvproj. This project is located within a folder on the desktop entitled Coincidence Counting FPGA.
      1. It is important that LabVIEW be run with administrator privileges (to work around some limitations associated with ANSTO’s IT security). IT has given us a local administrator account, hepsu (password=Becquerel123). You can run as hepsu when logged in as hepvis (if you log in as hepsu you can’t access the network). To do this, right click on LabVIEW and select ‘run as…’. Choose the bottom option, and select [*computer name]*\hepsu, and enter the password.
      2. If you ever need to recompile the FPGA code (you won’t need to unless you modify the program to, say, include another input), the compilation will give an error as the compile worker is by default opened as hepvis. You can get around this by:
         1. Attempt the compile (it will fail).
         2. Open Windows’ task manager.
         3. In processes, there will be ‘compileworker.exe’ (or similar) running as hepvis.
         4. Force stop this process and open the compile worker, making sure to run as hepsu (you can get to it from the start menu within the National Instruments->FPGA Module subfolder).
   2. Fill out the parameters on the front panel and acquire data.
      1. The only tricky one is the number of buffer events to read out per iteration; there are some notes on the front panel that give some idea regarding what this should be if you don’t know what to set it as.
         1. Essentially for higher count rates this should be high, for lower count rates this should be low. If you get a buffer overflow error increase this number, if you get a timeout error reduce this number.
4. Analyse data.
   1. Open Matlab and set the working directory to be the folder in which the acquired data were saved.
   2. Any data that you actually want to use in Matlab needs to be imported and decoded, I’ve written a function to do this.
      1. The function is of the form:  
         *OutputMatrix* = DecodeData(‘*InputFile*’);
      2. In the above, italicised words indicate parameters that you need to input (I’ll use the same convention below).
      3. *OutputMatrix* is a matrix with 4 columns and a number of rows that corresponds to the number of events that were acquired. Column 1 is channel 1, column 2 is channel 2, column 3 is the number of clock ticks since the start of acquisition, and column 4 gives a 1 if the previous event timed out on the FPGA write to the FIFO (0 otherwise).
   3. At this point, it really is up to you how you analyse the data, but I’ll give some general instructions about how you may want to do it.
   4. To get the number of counts received in channel 1, do:  
      *NumCounts* = sum(*OutputMatrix*(1,:));
   5. I’ve written a function to analyse the data. The function itself is quite heavily commented so hopefully it is pretty readable. You can do coincidence or anti-coincidence analysis with it, extending or non-extending dead time, with a gamma delay, dead time and coincidence resolving time of your choice. The functional form is as follows:  
      [*Ch1 Ch2 UncorrCh1 UncorrCh2 Coincidence Tlive Treal*] =  
      GetCoincidences(*OutputMatrix*, *GammaDelay*, *DeadTime*, *CoincResolvingTime*, *IsAntiCoinc*, *IsExtendingDT*);
      1. The above parameters should all be obvious except for the final two: these are Boolean (true/false) parameters that determine whether anti-coincidence is used (true for yes, false for no), and whether the dead time is extending (true for yes, false for no).
      2. All times are given in **nanoseconds**.
      3. Note that although the channel and timing data are output to screen, they are also saved to the workspace.
   6. NB: all of the functions that I’ve listed are scriptable; it should be very easy to write a script to import a bunch of different acquisitions, then analyse them with different resolving times/dead times, using coincidence/anti-coincidence and extending/non-extending dead time, then to plot all of the results or do further analysis on the outputs.
      1. For example, you might write a script that takes all of the raw data from an extrapolation, then fits the extrapolation and plots the result, and outputs the activity with uncertainty.