^{6}He CRES DAQ Design

Brent Graner

September 13, 2018

Abstract

This document outlines the organization of the data acqusition (DAQ) software for the ^{6}He CRES experiment. Beginning from the design principles, consideration is given to the simplest means of overcoming the ambiguity introduced into Doppler-shifted signals, as well as the most efficient way to organize data processing without loss of frequency/energy resolution. The ROACH2 bitcode is introduced and discussed, along with installation tips for getting up to speed on development work.

1 Doppler shift

The axial motion of a radiating electron in a magnetic trap will cause the signal recieved by an on-axis observer to be modulated in frequency each time the electron reverses direction. The magnitude of the Doppler shift (proportional to the electron's maximum axial velocity) and the frequency of axial oscillation together define the *modulation index*, denoted here as h:

$$h = \frac{2\pi z_m f_c}{v_p} \tag{1}$$

where f_c is the cyclotron frequency, z_m is the maximum axial displacement of the electron from the center of the trap, and v_p is the (frequency-dependent) phase velocity of the waveguide mode of interest (in our case, the TE_{10} mode).

The sidebands induced in a Doppler-shifted wavetrain at high modulation index make it difficult to unambiguously identify the cyclotron frequency. The effect of frequency modulation due to the axial motion of the trapped electron is illustrated in Figure 1. At a modulation index of 2.41, the carrier disappears and all received power is shifted into a large number of sidebands on either side. If the SNR is poor in the ^{6}He data, fewer sidebands may stand out above the noise floor, making unambiguous identification of the carrier frequency nearly impossible. The modulation index may be suppressed by decreasing the trap depth, at the cost of a substantial decrease in the rate of detected events. To accept a non-negligible fraction of electrons, the trap must be deep enough to include particles with a modulation index greater than 2.

Because a dual-receiver experiment is feasible, the simplest way to eliminate the Doppler shift as a source of uncertainty is to digitize the signals from both ends of the waveguide. Then one amplifier will output a red-shifted wave, while

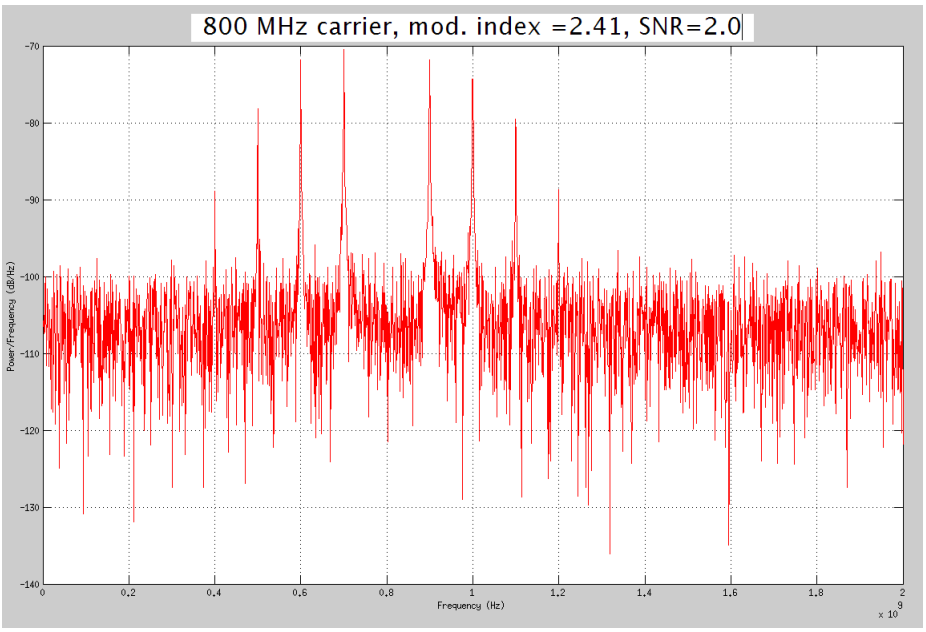

Figure 1: Plot of simulated power versus frequency for an 800 MHz electron with a signal/noise power ratio of 2.0, an axial trap oscillation frequency of 100 MHz, and a modulation index of 2.41.

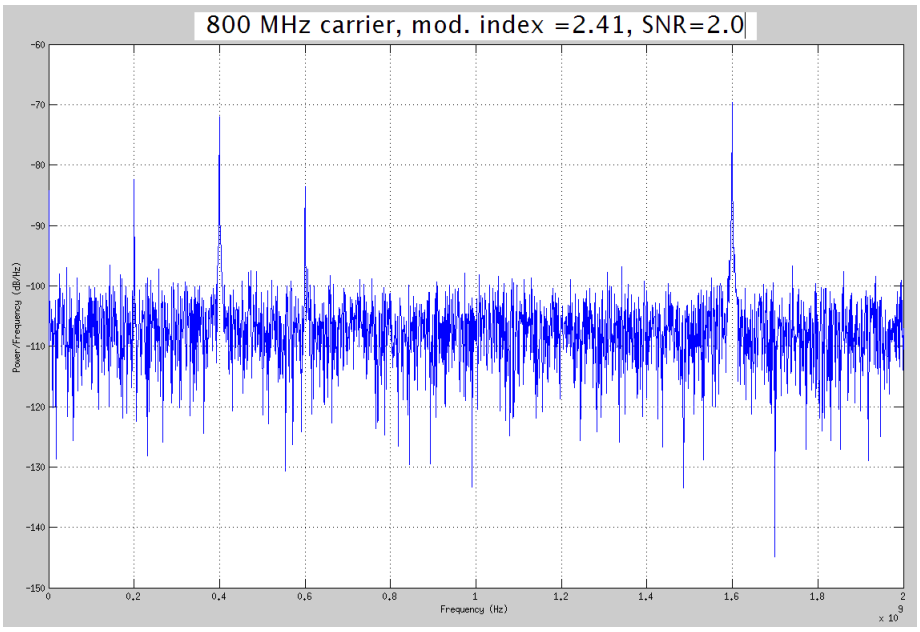

Figure 2: Plot of simulated power versus frequency for the same 800 MHz electron with red-shifted and blue-shifted wavetrains multiplied together following digitazation with 2 8-bit 5Gs/sec ADC cards.

the other amplifier will output a blue-shifted one[1]. The downmixed signals can be digitally multiplied and Fourier transformed, which will yield peaks at the sum and difference frequencies. The sum signal of the red-shifted and blue-shifted wave will be at twice the cyclotron frequency, giving an unambiguous way to identify the carrier from any set of sidebands.

The drawback to this method is the limitations imposed by the Nyquist theorem and a finite sampling rate. Instead of being limited to features less than half the frequency of the sampling, we will be limited to seeing cyclotron radiation at less than 1/4 of the ADC sampling frequency. In practice, this means that we will be able to see only cyclotron radiation less than 1.25 GHz above the frequency of the analog downmixing LO. Given that the ambiguity in the Doppler shifted waveforms is a considerable technical challenge, it seems that this is a viable tradeoff. We can put a bandpass filter on the analog RF output from 0.1 to 1.1 GHz, and the minimum cyclotron frequency can be placed in the baseband by tuning the value of the magnetic field B_0.

2 Introduction to the CASPER and ROACH2 systems

The DAQ system for the He6CRES experiment is based on a Reconfigurable Open Architecture Computing Hardware v.2 (ROACH2) Field Programmable Gate Array (FPGA) system designed by the Collaboration for Astronomical Signal Processing and Electronics Research (CASPER collaboration). The FPGA is programmed through a graphical interface (similar in style to National Instruments LabVIEW) that divides the FPGA chip into various 'blocks' with dedicated input and output pins. However, information does not flow through the FPGA in the manner of a traditional processor, where the software is reducible to a list of operations to be executed sequentially on data that is accessed and stored in memory. In an ideal FPGA design, each block operates on each input and output pin on each clock cycle, as if all the functions in a typical software program were called simultaneously on a continuous stream of inputs. This design enables enormous throughput, but is complicated and unintuitive for a traditional programmer to conceptualize.

The FPGA was provided to us from Xilinx, and is programmed using the Xilinx Simulink software package with ISE. The following sections give an explanation of the DAQ system, beginning with a description of the software stack necessary to program and compile bitcode. The remaining sections are devoted to explaining the `he6_cres_correlator.slx` model file in further detail, as well as the low-level libraries available for interacting with the ROACH2 once it has been programmed.

[1] There will be a path-length difference of order 10 cm between the decay volume and either end of the waveguides as they exit the magnet bore, but the low frequency of axial oscillation implies the red-shifted and blue-shifted wavetrains will be several meters long, so 2 amplifers situated at the same value of z will almost always see waves with opposite Doppler shift. If necessary, the residual effect can be compensated by extending the waveguide arm without a 180-degree bend farther away in z from the decay volume.

3 CASPER Library and Xilinx ISE Installation

This section details the installation of the software needed to compile FPGA designs into bitcode that can be loaded directly onto the ROACH2 and is meant to help people through the inevitable compatibility issues encountered in the installation process. Readers who are not interested in installing the software to develop bitcode themselves should skip to the next section.

I achieved a semi-stable configuration using the following list as described at https://casper.berkeley.edu/wiki/MSSGE_Setup_with_Xilinx_14.x_and_Matlab_2012b

1. Ubuntu 16.04
2. MATLAB R2012b + valid license file
3. Xilinx ISE version 14.7 system edition with System Generator
4. CASPER libraries mlib_devel (switched to the ROACH2 branch)
5. A valid Xilinx license file
6. Linked versions of make and gmake

1) Ubuntu 16.04: This is what I took as a starting point. If you have total freedom, CASPER recommends Ubuntu version 12.04 or 14.04. Xilinx ISE 14.7 and MATLAB will work on Windows 7, but I don't know how the CASPER libraries could be installed without /bash.

2) MATLAB R2012b can be downloaded from the Mathworks website, although you will need to manually select and download a substantial number of packages. Per the instructions from Mathworks, just put all the .zip files in a directory, unzip the installer, execute it in a terminal, and go through the steps in the GUI.

   pitfall: Mathworks licensing authentication servers require an internet connection configured as 'eth0'. If your machine (like mine) has no physical ethernet jack, your wireless connection will be named something like 'wlan0' by default.

   workaround: (as described at https://askubuntu.com/questions/767786/changing-network-interfaces-name-ubuntu-16-04): run `ifconfig` to check whether or not your machine has an eth0 interface. If not, navigate to (or create) the directory eth/udev/rules.d and edit (or create) the file called 70-persistent-net.rules (you may also have to change the directory permissions using `chmod` or `chown`). Add the following line to 70-persistent-net.rules, replacing the 12 "X" characters with the digits of your machine's MAC address:

   SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTRaddress=="XX:XX:XX:XX:XX:XX", ATTRdev_id=="0x0", ATTRtype=="1", NAME="eth0"

   Reboot and run `ifconfig` again to verify that your changes have taken effect.

3) Xilinx ISE version 14.7 system edition with System Generator: Version 14.7 is the latest iteration that can be used to program the Xilinx Vertex 6, and by extension, the ROACH2. Packages can be downloaded from https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools/v2012_4—14_7.html

   Instructions can be found at https://github.com/casper-astro/mlib_devel/wiki/How-to-install-Xilinx-ISE

   pitfall: Install the "System" edition, as opposed to the "WEB PACK" or "embedded" development kit. Be sure to edit the LD_LIBRARY_PATH variable in ALL THREE of the following .settings64.sh files:

   Xilinx/14.7/ISE_DS/common/.settings64.sh
   Xilinx/14.7/ISE_DS/EDK/.settings64.sh
   Xilinx/14.7/ISE_DS/ISE/.settings64.sh

   If you don't have all three of the above directories, you need to change the edition of ISE design suite you installed.

4) CASPER libraries mlib_devel (switched to the ROACH2 branch): Clone the ROACH2 branch of the libraries by running `git clone -b roach2 https://github.com/casper-astro/mlib_devel/tree/roach2`

5) License file: Xilinx donates licenses through its university program; it takes a few days to be approved and each license is 'node-locked' so that it can be run on one and only one machine. If you don't want to do the paperwork or wait, I have some extra licenses–ask and ye shall receive. Whoever you get it from, run the License Configuration Manager `xlcm` and navigate to the folder where you keep your license file to install it.

6) Another potential pitfall at compile time involves the use of gmake. Ubuntu has both make and gmake installed by default, but they may need to be linked. Use the command `sudo ln -s /usr/bin/make /usr/bin/gmake` to create a symbolic link to gmake.

4 Input signal digitization and normalization

The current design of the he6_cres_correlator includes 2 ADC1x5000-8 (aka ASIAA 5Gsps ADC) cards. Our models use the DMUX 1:1 multiplexing configuration (set in hardware), which outputs two 8-bit samples per ADC clock cycle[2]. The ADC is run off an external clock input with a maximum rate of 2.5GHz, which equates to 5 GS/sec. However, Andre Young has advised us that the ADC response becomes distorted at the highest possible frequencies, so the target sampling rate is set to 4 GHz (or a 2000 MHz ADC clock). The corresponding CASPER software 'yellow block' features 16 parallel outputs, two of which are filled on each ADC clock cycle. The FPGA should therefore be clocked at 1/8 times the rate of the ADC clock. The design of the he6_cres_correlator

[2]The other available option is DMUX 1:2, which outputs four 4-bit streams per ADC clock cycle

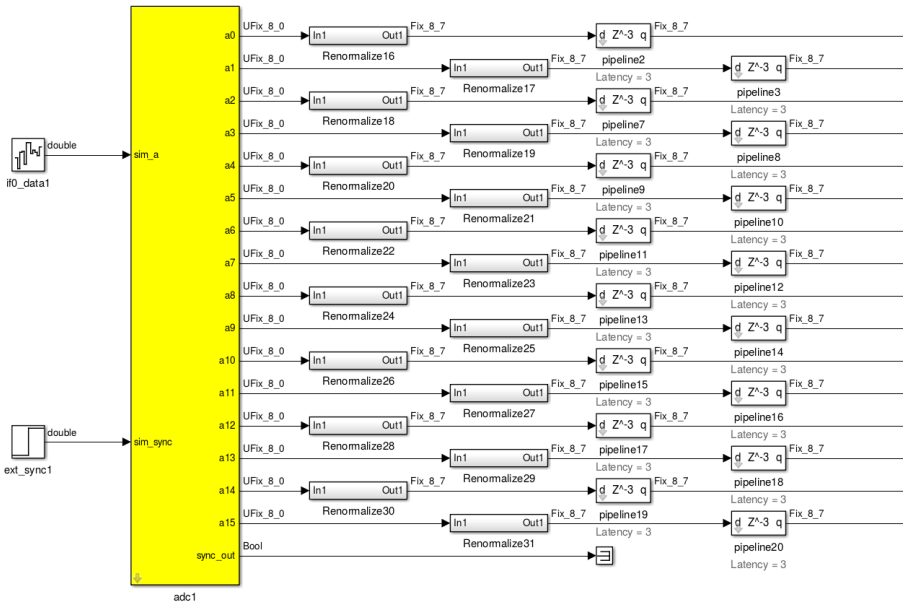

Figure 3: The ADC portion of the block diagram. White noise is input in simulation, while the ADC yellow block outputs 16 8-bit binary numbers (UFix_8_0) on each FPGA clock cycle. The renormalize blocks use two's complement representation to convert each UFix_8_0 to signed Fix_8_7, giving the normalized signal a numerical range of ± 1.

is set to derive the FPGA clock from the ADC clock inputs[3].

The ADC has a differential analog input range from 0 to 500 mV, and the values output from the yellow asiaa_adc5g yellow blocks are encoded as fixed point 8-bit binary numbers (i.e., from 0 to 255). However, the digital filtering and Fourier transform algorithms work best with signals that have zero DC component. Therfore, each ADC yellow block output is fed to a *renormalize* block, which slices the most significant bit, inverts its value, and concatenates it with the remaining 7 (unchanged) bits. The new bit string is then 'reinterpreted' by the code as a two's complement number with 8_7-bits, meaning that the most sigificant bit (MSB) inidicates the sign, while the 'binary point' is understood to come between the MSB and the 7 LSB. This proceedure is equivalent to subtracting 128, then dividing by 128. The output signal should therefore have a numerical range of $\pm$ 1. For further explanation of the two's complement scheme and nomenclature, see https://casper.berkeley.edu/wiki/Wideband_Spectrometer

[3]The ADC can be also be operated in dual-channel mode, putting two different analog signals into a single ADC card, with one sample per SMA channel per ADC clock cycle. In single-channel mode, one SMA input is connected, and the ADC card gives 2 samples per clock cycle. We will use the single channel mode, so the clock frequency will be half that of the sampling frequency (cf. https://www.mail-archive.com/casper@lists.berkeley.edu/msg06356.html)

5 Fixed-point multiplication

The re-normalized outputs are sent to the *multipliers* subsystem, where each pair of simultaneous voltage samples from the two ADCs is multiplied. Because there are 7 bits representing the magnitude of the ADC voltage sample, there will be $2^7 \cdot 2^7$ possible magnitudes for the product. Including the sign bit, the products can be stored as 15_14 bit fixed-point values without any loss of resolution. However, it is more convenient to work in powers of 2, so the outputs of the mulitpliers are 16_14 two's complement fixed point values (or Fix_16_14, for short)

6 Polyphase filter bank

The output of the multiplier section is entered into a polyphase filter bank block which applies finite impulse response (FIR) digital filters to reduce flaws in the discrete Fourier transform algorithms including spectral leakage, scalloping loss, etc. The filters are implemented by digitally multiplying input data points with sinc function windows. This is an important topic and is treated extensively in the book by Smith, which may be downloaded free of charge at http://www.dspguide.com/.

The input bitwidth is set to 16 bits to match the multiplier output. The PFB takes 2^{13} time samples as input, which must match the FFT block number of input samples. Samples are output as two's complement 18_17 bit numbers (Fix_18_17), and divided by 4 using shift registers before being input to the FFT block itself.

7 Fourier transform

The FFT algorithm of choice is a 13-stage real-input butterfly biplex Fourier transform. The most noticeable feature of the green FFT block is that it has twice the number of inputs as outputs. The standard FFT algorithm will output the same number of frequency channels as it takes time-domain inputs, but the negative frequency components of a real-input FFT will end up being a mirror-image version of the positive frequency components. Therefore, the FFT block only outputs the positive half-spectrum, so the block has half the number of outputs as it has inputs. In our case, the green FFT block takes 2^{13} real values as input (16 input values per FPGA clock cycle) and gives 2^{12} complex values as output (8 per clock cycle). One thing to note about the CASPER designs is the apparent convention for representing complex numbers as fixed-point quantities. Many of the CASPER blocks incoporate a 'BitWidth' setting, and operate using "A complex number whose higher BitWidth bits are its real part and lower BitWidth bits are its imaginary part". This is only relevant to the FFT block because it's the only block in the program that takes real (time-domain) samples as input, and returns complex (frequency domain) points as output. The real and imaginary pats of a given fequency bin are output simultaneously on each clock cycle for a given output 'pin'.

Since we are interested in a spectrogram of power vs. frequency, it is more appropriate to work with the mod squared of each FFT channel. Each power block slices a complex-valued FFT output into two 18_17-bit samples (in two's

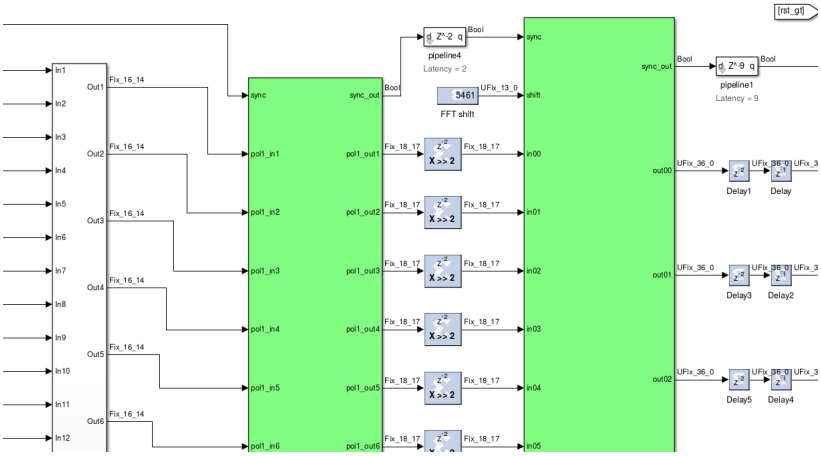

Figure 4: Section of the polyphase filter bank (PFB) and ADC portion of the block diagram. Six of the outputs from the multiplier subsection are visible at the left, which feed into the PFB green block at the center. The PFB outputs a Boolean sync pulse one clock cycle before the output data are valid, while each 18_17 bit fixed point output is divided by 4 using shift registers before being input to the FFT block on the right. The FFT takes a continuous stream of time-domain inputs and outputs the positive half-spectrum of frequency points, interleaved across the 8 output 'pins'. The FFT is set for 2^{13} time domain inputs, so the positive half-spectrum repeats with a period of $2^{13}/(2 \cdot 2^3) = 512$ clock cycles. For more on the sync pulse and the vector character of the FFT outputs, see the CASPER sync memo.

complement binary representation), then squares both real and imaginary parts (returning them to 36_34 bits) and adds them. Slice blocks then take the 16 most significant bits of the output. The mod squared values of the FFT are non-negative by definition, so we can get the best resolution out of 16 bits by dropping the two's complement representation convention at this point and treating the outputs as unsigned binary numbers.

Finally, the FFT shift register must be carefully set to maximize resolution without encountering overflow problems. An N-stage FFT block nust have an N-bit shift register, where each bit indicates whether or not the output of each successive stage is to be divided by 2. Andre Young recommends a shift register of 1101010101010 for the Project8 DAQ to prevent bit overflow, but it is not clear from first principles what the 'best' shift register value is for a given DAQ system. It is therefore left as a programmable variable in the bitcode.

8 Sync pulsing

The PFB and FFT blocks require a periodic sync pulse to keep them functioning properly on continuous inputs. The CASPER sync memo (incuded in the project documentation) details the behavior and requirements for a properly-functioning sync signal. As defined in the memo, the minimum sync period for a wideband FFT block is

$$LCM(\text{reorder orders}) \cdot \frac{\text{FFT size}}{\#\text{ of FFT inputs}} \tag{2}$$

where LCM refers to the least common multiple of the set of orders of the reordering/unscrambling blocks. The FFT block in the he6_cres_correlator has 2 'reorder' blocks, both of order 2, as well as an order 3 'unscrambler' block. The least common multiple of 2 and 3 is 6, so the minimum sync period is $6 \cdot 2^{13}/2^4 = 3072$. The 4 taps in the PFB block adds a factor of 4, and averaging 8 consecutive spectra adds a factor of 8. The sync period used is therefore 98304 FPGA clock cycles, and the sync pulse is implemeted with a 17-bit free running counter block and a comparator block. The counter reset and the comparator each have an internal latency of 1 clock cycle, so the comparison value is 98302.

9 Output data rate and BRAM vector accumulators

The design speed of the FPGA clock is 250 MHz, and the output of the FFT and power blocks will consist of 8 16-bit power spectrum amplitudes each clock cycle. This gives a 4000 MB/s output data stream, which could be output via the quad-SFP+ 10GbE ROACH interface, but is much too fast to be written to a typical computer drive. Hard disk drives (HDDs) can record data at about 125 MB/s, while more expensive solid-state drives (SDD) can record at about 550 MB/s. Because we are outputing frequency-domain data alone, we can limit the amount of data to be transfered and written to disk by implementing a power spectrum averaging window. Instead of taking each spectrum calculated by the FPGA and outputing it into the network immediately, a handful of power spectra are

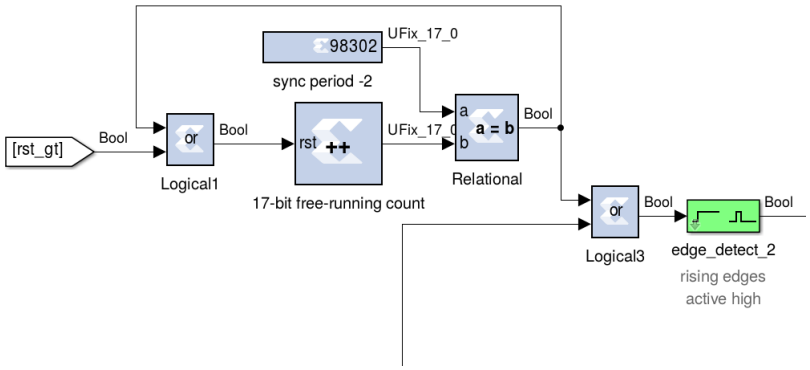

Figure 5: The sync portion of the block diagram. A pulse on the reset goto (rst_gt) can manually reset the system. On reset, the 17-bit counter starts at 0 and resets itself after the output value reaches 98302. The extra latency of the counter reset gives the system a pulse after every 98304 FPGA clock cycles.

added together in a set of vector accumulator block RAMs (or BRAMs) on the ROACH2 and output as a single composite spectrum. Choosing 8 spectra as an averaging parameter decreases the output to a manageable 500 MB/s, low enough to put the ROACH2 on a single blade server 10GbE network interface, where psyllid can be running and wrting egg file outputs to a 'hot storage' SSD. The 'hot' drive can then be manually transfered to a network-attached storage (NAS) unit once full. For a 1 TB SSD, 500 MB/s should fill the drive in 2000 seconds, or just over 20 minutes of live time.

The BRAM vector accumulators are connected to each FFT/power block/slice block output. The vector accumulators are designed to add their input value on each clock cycle to the value retrieved from memory a fixed number of clock cycles in the past. The FFT block outputs its 4096 frequency channels in canonical order spread across all the simultaneous outputs. For example, the first clock cycle with output will output the DC signal component (or 'zeroth' frequency channel) on out00, the first frequency channel on out01, etc. On the following clock cycle, the eighth frequency channel will be output on out00, the ninth on out01, and so on. It will therefore take 512 clock cycles to output all 4096 frequency channels with 8 simultaneous outputs, so to add successive spectra, the BRAM vector accumulators are set to sum each output value to the stored value from 512 clock cycles in the past. The BRAM vector accumulators will continue to add up the FFT outputs from successive spectra until they receive a 'new_acc' pulse, which is dervied from the MSB of a free-running 12-bit counter.

Finally, we should consider the inpact of averaging spectra on the time resolution of our waterfall plots. With 2^{13} points in each FFT and a sampling frequency of 4 GHz, each power spectrum will take 4.1μs to capture. This is about a factor of 7 shorter than the time resolution of the Project 8 plots, which are 30μs long. This means that our choice of 8 spectra as an averaging parameter decreases the output data rate to 500 MB/s without adversely impacting the time resolution of events.

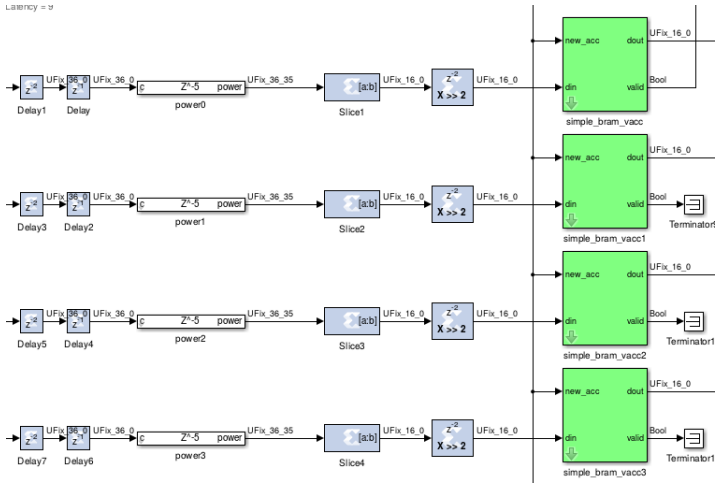

Figure 6: The post-FFT section of the ROACH code. The power blocks convert from 18-bit complex-valued FFT outputs to unsigned 36-bit binary numbers (UFix_36_35). The slice blocks round the output to 16 bits, and the block RAM vector accumulators add the data points to the appropriate place in a vector of 512 points stored in memory. The accumulated vector is output when the block receives a pulse on the new_acc input pin.

10 Power spectrum packetzation and resolution

The simplest and most direct way to packetize the data would be to output one power spectrum per packet. The typical packets used by Project 8 have a payload of 8192 bytes each. The packetization block (and the overall packet structure) are lifted from the Project 8 design alternately known as r2daq or ArtooDaq. We retain the 8192-byte payload and 64-byte header, which includes metadata such as timestamps, packet number, analog channel, and some extra space reserved for future use.

To see if it is feasible to write a full spectrum in one such packet, consider the resolution attainable in writing a 2 GHz frequency region of interest (noting that we are interested in the sum frequency of two inputs each ≤ 1 GHz to remove the Doppler shift and still be below half the ADC sampling frequency). With 2048 frequency channels output from the FFT, we can fit all the frequency channels in a single packet with 16-bit (i.e. double-precision) resolution. Assuming the typical cyclotron frequency is approximately 19 GHz, the frequency resolution is

$$\frac{2 \text{ GHz}}{19 \text{ GHz}} * \frac{1}{2048} = 5.12 * 10^{-5} \tag{3}$$

or about 50 ppm resolution of the average cyclotron frequency, well within acceptable limits.

The *packetization* block is used to assemble the header and output the frequency data to the 10GbE network interface. The bus block concatenates the 8 16-bit BRAM vector accumulator outputs into a single 128-bit word, which is passed to the packetization block. The data are delayed 3 clock cycles relative to the sync pulse to enable the packetization block to assemble and output the

256-bit header (one clock cycle for each 128 bit header word, and 1 cycle for counter latency).

11 Phasmid library and installation tips

The Project 8 software module called phasmid is located at https://github.com/project8/phasmid. This module contains the lowest-level library r2daq.py for interacting with the ROACH2, programming the FPGA with some precompiled bitcode, getting and setting values from the software registers, changing the FFT shift vector, as well as calibrating offset, gain, and phase parameters of individual ADC cores. A similar version, He6DAQ.py, is unique to the He6CRES ROACH2 setup and is currently being developed.

To install phasmid:

1) Run `git clone https://github.com/project8/phasmid`

2) Run `git clone https://github.com/sma-wideband/adc_tests`

3) Change into directory containing lowest-level-ROACH2 interface library (r2daq.py or He6DAQ.py)

4) Copy adc_5g folder from adc_tests into same directory as r2daq.py or He6DAQ.py

5) Run `sudo pip install corr==0.7.3`

6) If scipy is not already installed, run `sudo pip install scipy`

7) If netifaces is not already installed, run `sudo pip install netifaces`

8) Run `sudo pip install construct==2.5.5-reupload`

If the installation is successful, you should be able to navigate to the folder with the DAQ library, boot up an interactive python session (run `ipython`), and run `import r2daq` or `import He6DAQ` without any error messages.