**experiment program coding basics**

At the time of writing there are three types of experiment program:

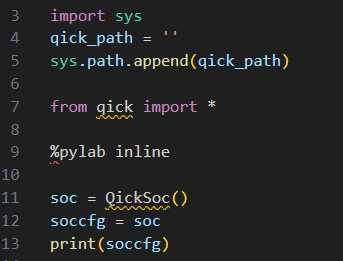
|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Main use** |
| 1 | AveragerProgram + acquire\_decimated() | Debugging, tof calibrate, etc. |
| 2 | AveragerProgram + acquire() | Measurement, with few variables to sweep. |
| 3 | RAveragerProgram + acquire() | Measurement, with several variables to sweep. |

Since only type1 and type2 are used more commonly in the qubit measurement to be introduced in this thesis, following I will only detail on type1 and type2. To know how to use type3, you can refer to “qick\_demos /02\_Sweeping\_variables.ipynb” on qick’s official github repository.

Type1: AveragerProgram + acquire\_decimated()

I will mostly follow the demo notebook “qick\_demos /00\_Send\_receive\_pulse.ipynb” on qick’s official github repository as an example, which uses the *axis\_signal\_gen\_v6* as sg (signal generator), and *axis\_readout\_v2* as sr (signal reader). Examples of using other sg and sr can be found in the section *ip*.

We first import packages & load bitstream:



**Line4**: qick\_path is the directory where you place the downloaded qick repository.

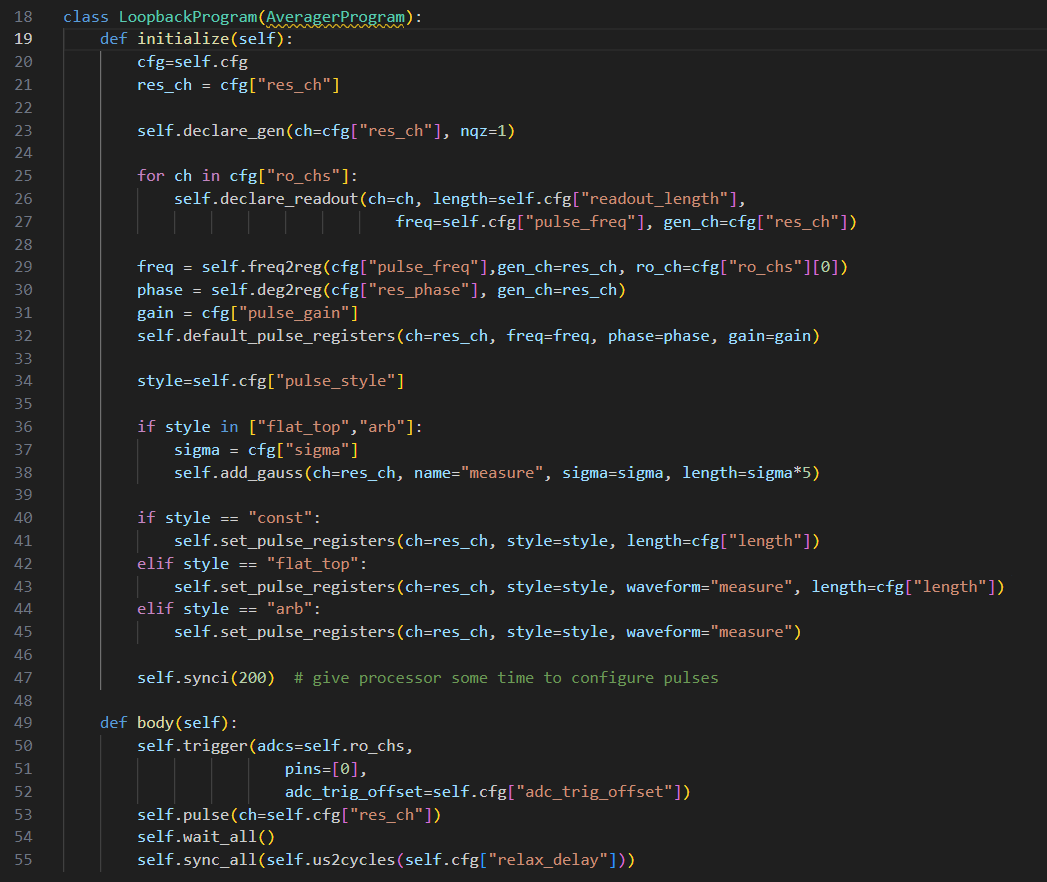
**Line11**: create an QickSoc object and load default bitstream distributed with qick’s repo. If you want to use other bitstream, you can replace this line with:

Soc = QickSoc(bitfile= bitfile\_path)

Where bitfile\_path is the path to the directory containing the bitstream file. Also notice that an accompanying .hwh file also need to be placed in the same directory (see the section *generate bitstream & load with pynq*).

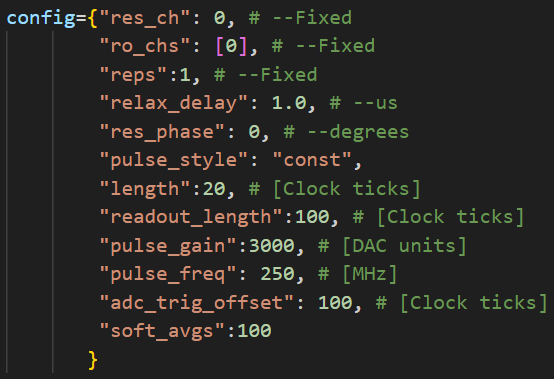
**Line13**: Display firmware info, including dac, adc port mappings, clock speeds, buffer sizes, etc.

Next, define the program class as shown below.

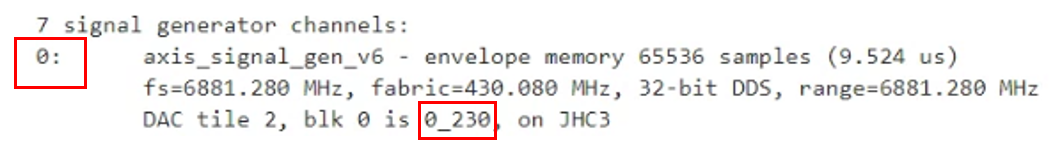


**Line19**: the contents inside initialize() will be converted to binary instructions as parts of a tproc program, and will be executed only once at the beginning by tproc to initialize stuffs each time the tproc program runs. The tproc program will be run cfg[‘soft\_avg’] number of times when you call acquire() (introduced later).

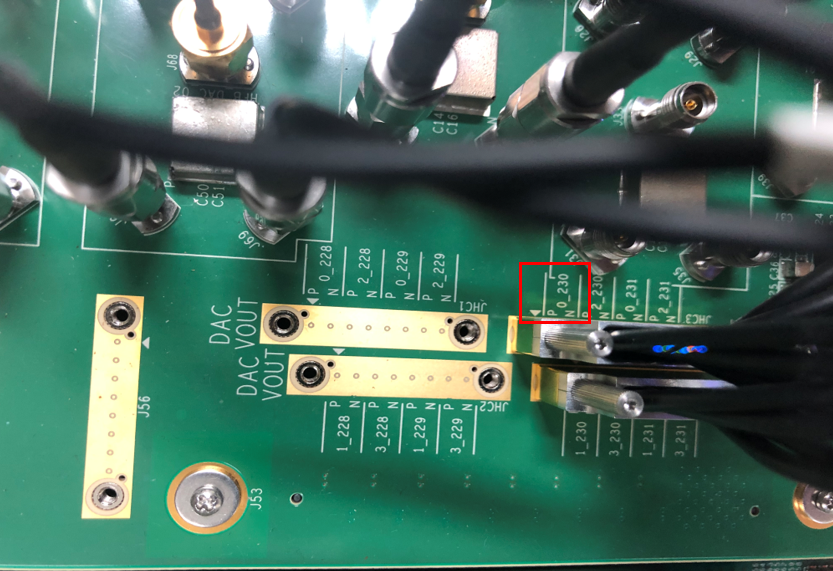
**Line20**: the cfg is as shown below, each quantity will be explained later.



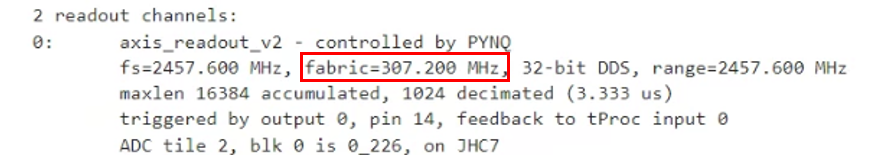
**Line23**: declare to use a signal generator. Need to pass it channel number (res\_ch), and the nyquist zone of its output (see the section *sampling & re-construction*). The channel number info can be obtained from the firmware info (see Line13 explanation above) as follow:



The ‘0’ is the virtual channel number and is filled in res\_ch, and the ‘0\_230’ is the physical channel number that you want to use to send RF signal and can be found on xm655 add-on card as shown below.



**Line25, 26, 27**: declare to use signal reader. The ro\_chs can be similarly obtained as in Line23 explanation above. readout\_length is length of signal to read in signal reader’s fabric clock cycle, which can be found as follow:



For example, if readout\_length is 100, the length of the signal that will be readed is 100 / (ffabric \* 106) = 325.5 ns. pulse\_freq is the demodulation frequency. We also pass the channel of the signal generator whose signal this signal reader is trying to read to ensure frequency match. If there is a frequency mismatch, you won’t see consistent phase between acquisitions.

**Line29**: Convert pulse\_freq (originally in MHz) to register value, which the logic circuit in fpga can process. We also need to pass signal generator and readers channel, res\_ch and ro\_chs, to ensure frequency match. pulse\_freq is the carrier signal’s frequency at the output (of DAC or XM655). See the section *ip* for the valid pulse\_freq range and resolution.

**Line30**: convert res\_phase (originally in degree) to register value. res\_phase is the phase offset relative to 0. See the section *ip* for the valid res\_phase range and resolution.

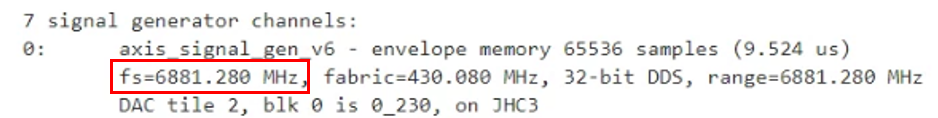
**Line31**: pulse\_gain is a multiplicative factor to the signal just before coming out from the sg. See the section *ip* for the valid pulse\_gain range and resolution.

**Line32**: see the doc for axis\_signal\_gen\_v6 in the section *ip* for details about *pulse registers*.

**Line36 ~ 45**: There are three build-in pulse styles: square (const), gaussian square (flat\_top), gaussian (arb) as shown below.

|  |  |  |
| --- | --- | --- |
| const | flat\_top | gauss |
|  |  |  |

Note that the time between two successive data points is 1/fs, where fs is the DAC sampling freq:



Note also that the arb can be used to implement arbitrary envelope and not just gaussian, and the way you do this is by setting pulse\_style to be “arb” and replacing line36 ~ 38 with:

self.add\_envelope(ch, name, idata = my\_data)

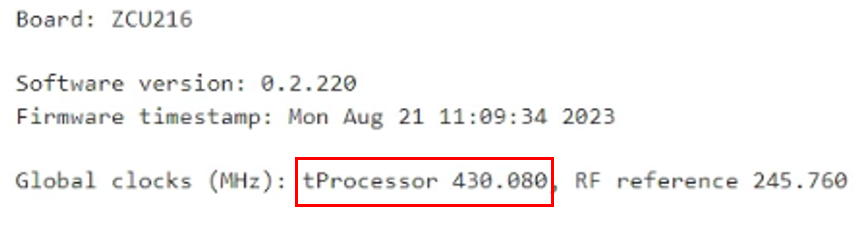
where my\_data is your arbitrary waveform data and it is a 1D array.

At line38, we add a gaussian envelope (as shown below) into table memory in sg (see the doc for axis\_signal\_gen\_v6 in the section *ip*). The quantities sigma, length are all in fabric clock ticks of the sg. *name* can be arbitrarily but need to match the *waveform* used in set\_pulse\_registers() to be introduced later. At line40 ~ 45 we write some quantities to pulse registers (see the doc for axis\_signal\_gen\_v6 in the section *ip*).

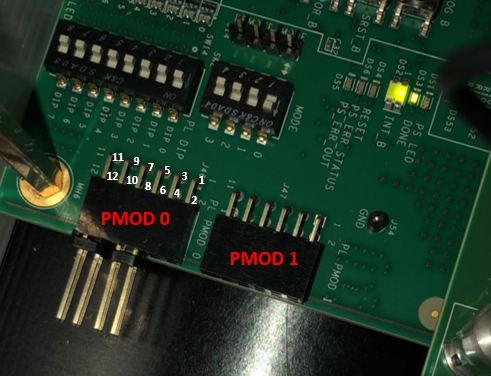
**Line47**: offset the reference time forward by 200, tnew = told + 200, and the specified timings for all the operations that follows (t in self.pulse(), adc\_trig\_offset in self.trigger(), etc) will be with respect to tnew.

**Line49**: the content in body() will be converted to binary instructions as parts of a tproc program, and will be executed by tproc by cfg[‘reps’] number of times continuously in fpga without software intervention (therefore is fast) each time the tproc program runs, and then the tproc program will end. The tproc program will be run cfg[‘soft\_avg’] number of times when you call acquire() (introduced later).

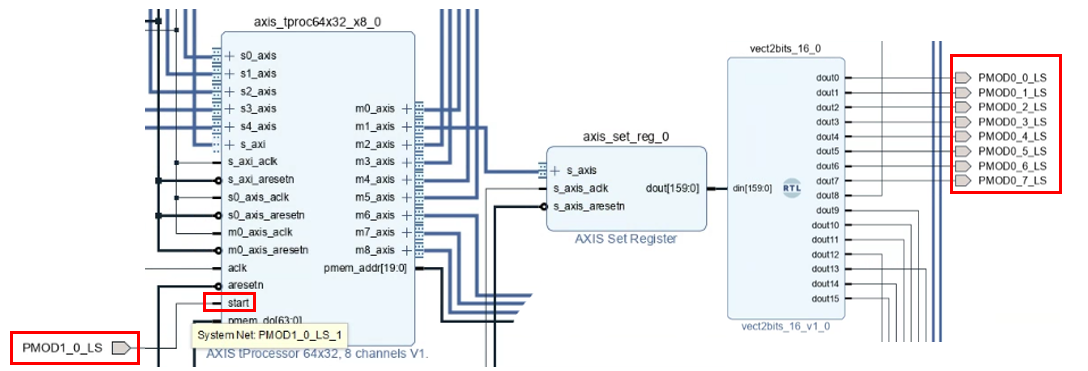
**Line50~52**: Command the tproc to fire square trigger pulse at time tnew + adc\_trig\_offset, both in tproc clock ticks, which can be found in firmware info as shown below. The tnew is introduce earlier in Line47 explanation.



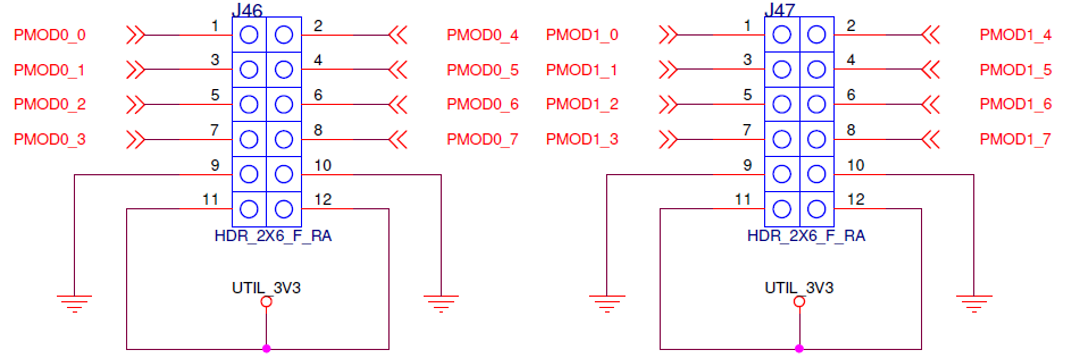
Two square trigger pulses will be fired, one is internal to fpga from tproc ip directly go to readout buffer ip (adcs=self.ro\_chs), the other can go to other external instruments through pin 0 of PMOD on zcu216 (pins=[0]). There are two PMOD’s on zcu216:



PMOD 1 is used to allow us to trigger tproc to start running program from outside, PMOD is used for tpoc to send trigger to outside. This info can be obtained from bd (vivado block design) as shown below. To know how to re-create an bd, see the section *export & re-create vivado block design*.



The pin mapping from virtual to physical can be found in XTP577 - ZCU216 Schematics:

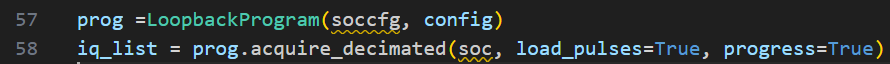


**Line53**: play a pulse at the channel res\_ch at time t = tnew + 0, where 0 is the default pulse time. If you want to play pulse at other time, say t1, then replace this line with self.pulse(t=t1, ch=self.cfg["res\_ch"]). The tnew is introduce earlier in Line47 explanation.

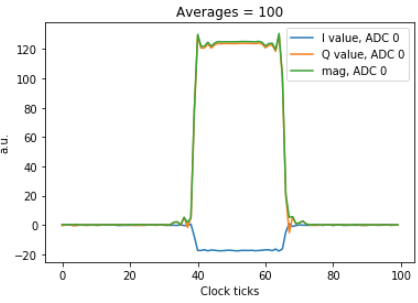
**Line54**: pause the tproc until all the readout operations that are started before self.wait\_all() is called are finished. In our case, the tproc will be paused until t = tnew + adc\_trig\_offset + readout\_length, where the tnew is introduce earlier in Line47 explanation.

**Line55**: Similar to Line47. Let tend\_max be the end time of the operation that ends the latest among all operations that are stared before self.sync\_all() is called. The new reference time will be t’new = tnew  + tend\_max + self.us2cycles(self.cfg["relax\_delay"]). In our case, tend\_max = adc\_trig\_offset + readout\_length. The tnew is introduce earlier in Line47 explanation.

Finally, we will generate tproc program, load into fpag, and start the tproc. When an object of LoopbackProgram is created, the contents in initialize() and body() will be converted into tproc program, which consists of an array of 64-bits instructions. When you call acquire() or acquire\_decimated(), the envelope data and the tproc program are loaded to fpga, and the tproc program will be executed by config[‘soft\_avg’] number of times.



The acquired data iq\_list is (down-sampled) raw waveform data.



Type2: AveragerProgram + acquire()

Everything is the same as in Type1 above, except we replace Line58 by:

avgi, avgq = prog.acquire(soc, load\_pulses=True, progress=False)

both avgi and avgq contain only a single number, which is the average of entire i or q components of the (down-sampled) raw waveform data (an array) from type1 above.