**axis\_sg\_mux4\_v2**

Introduction

This ip is often used to play multiple readout pulses to multiple resonators to readout the states of multiple qubits at the same time, all using one DAC. This ip is often used in conjunction with axis\_pfb\_readout\_v2 (also introduced in this thesis) to demodulate the pulses coming back from resonator.

Specs

To know how to program values for the following quantities from python, see the section *qick python library*.

* ~~Phase[[1]](#footnote-1): 0 to 360 degree, resolution is 32-bits (step 360/2~~~~32~~~~).~~
* Gain: -1 ~ 1, resolution is 16-bits. (TODO: include GFP table)
* DDS Frequency[[2]](#footnote-2): from - fdds/2 to fdds/2, resolution is 32-bits (step fdds/232­­­), where fs is sampling rate of DAC, and fdds = fs/4.
* RFDC mixer frequency (Fine mode)[[3]](#footnote-3): 0 up fs, resolution is 48-bits (step fs/248­­­).

(Note: DAC’s actual output frequency is RFDC’s mixer frequency + DDS frequencies, together with their images (see *Sampling & re-construction* section).)

* Nyquist zone: 1 or 2.
* Available waveform style: const (square wave).
* Waveform length:
  + const: no limit.
* No envelope memory.

How to get started using it (zcu216, vivado2020.2)

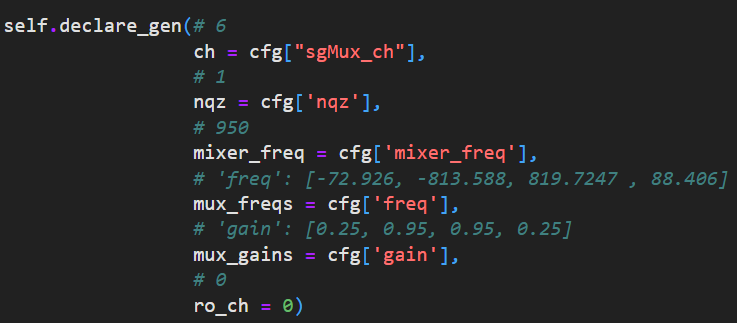
The demo codes are here at the link below (see the section *qick python library* for more details about the code structure in general). The bitstream files are also included. If you don’t know how to use the bitstream files, see the section *generate bitstream & load with pynq*.

<https://github.com/Ri-chard-Wu/thesis/tree/master/codes/axis_sg_mux4_v2-demo-216>

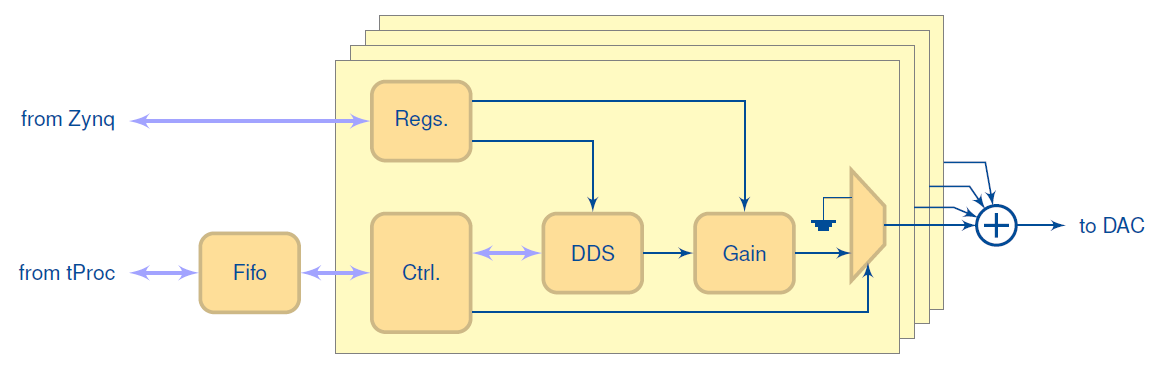
The wiring is a simple loopback from a DAC to a ADC. The DAC first sends out four frequencies to ADC. We then use an axis\_pfb\_readout\_v2 ip (also introduced in this thesis) to demodulate the four frequencies. The results is shown in figure below.

|  |  |
| --- | --- |
|  |  |

The Gain, DDS frequency, RFDC mixer frequency, and Nyquist zone mentioned in *Spec* are used this way:



How it work



* Four channels. Their outputs are summed and output through a DAC.
  + Frequency, phases and gains of the four channels can be independently controlled.
  + Play time of the four pulses cannot be independently controlled.

how to include it in firmware (zcu216, vivado2020.2)

IP core settings (double click on the ip):

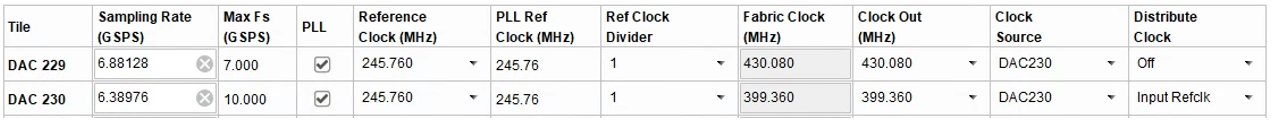
* “N\_DDS”=4

RFDC (Zynq Ultrascale+ RF Data Converter) DAC settings:

|  |  |
| --- | --- |
|  | * Interpolation Mode: 4x * Samples per AXI4-Stream Cycle: 8 * Datapath Mode: DUC 0 to Fs/2 * Mixer Type: Fine * Mixer Mode: I/Q->Real |
|  |  |

RFDC DAC tile clocking settings:

* For details about the fields see the section *rfdc settings*.
* For demo purposes, just select *Clock Source* to be the tile itself, and select *Distribution Clock* to be *off*. For details, see the section *rfdc settings*.

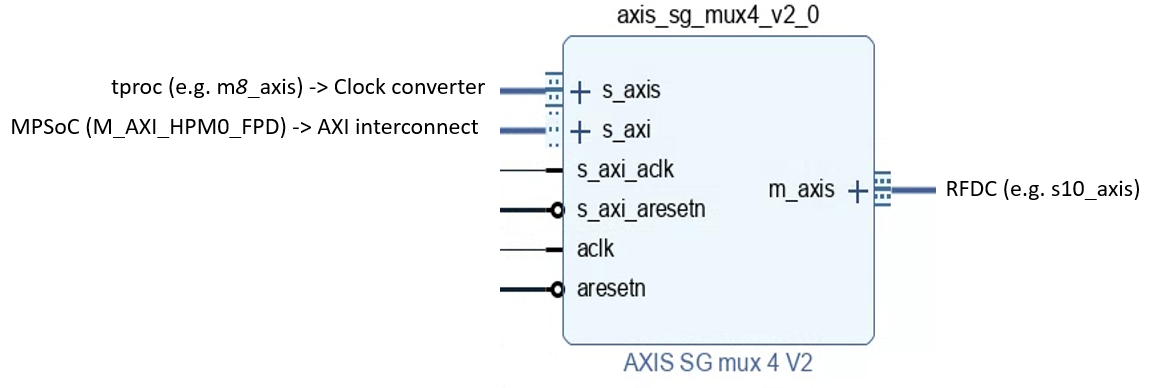


Wirings:

* The ”Clock converter” below is needed only if the clock speed of muxed sg (aclk) is different from that of tproc (aclk).
* For full details, you can re-create the vivado block design using the scripts (bd\_216 … .tcl, proj\_216 … .tcl) at:

<https://github.com/Ri-chard-Wu/thesis/tree/master/codes/axis_sg_mux4_v2-test-216>

If you don’t know how to use the scripts, see the section *export & re-create vivado block design*.



1. relevant qick functions: QickConfig::deg2reg(). [↑](#footnote-ref-1)
2. relevant qick functions: QickConfig::freq2reg(), QickConfig::freq2int(), QickSoc::set\_mux\_freqs(), AxisSgMux4V2::set\_freq(). Relevant parameters: gencfg['b\_dds'] and gencfg['f\_dds'] in QickConfig::freq2reg(). [↑](#footnote-ref-2)
3. relavant qick functions: AbsSignalGen::set\_mixer\_freq(), RFDC::set\_mixer\_freq(). Relavant codes: “fstep = fs/2\*\*48” in RFDC::set\_mixer\_freq(). [↑](#footnote-ref-3)