**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

* Phase[[1]](#footnote-1): 0 to 360 degree, resolution is 32-bits (step 360/232).
* Gain: -1 ~ 1, resolution is 16-bits. (TODO: include GFP table)
* Frequency (fs is sampling rate of DAC, and fdds = fs/4):
  + DDS of each channel[[2]](#footnote-2): from - fdds/2 to fdds/2, resolution is 32-bits (step fdds/232­­­).
  + RFDC’s mixer (Fine mode)[[3]](#footnote-3): 0 up fs, resolution is 48-bits (step fs/248­­­).
  + DAC’s output: 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: no limit.
* 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.
* No envelope memory.

How to get started using it (zcu216)

The demo codes are here:

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

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*.

We design a simple experiment on zcu216 to test this ip core. The experiment is as follow: We use one DAC (DAC2) to send out four frequencies, and feed the four frequencies into one ADC (ADC0), following the ADC we use an axis\_pfb\_readout\_v2 ip that is also introduced in this thesis to demodulate the four frequencies, as shown in the figure below.

|  |
| --- |
|  |
| Use one DAC2 to send out four frequencies, and feed the four frequencies into one ADC0 |

Next, we let DAC2 play four pulses at the same time with different frequencies, and then let the ADC0 acquire the signal. Following shows the acquired pulses:

|  |
| --- |
|  |
|  |

We see that the pulse’s shape looks good.

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

IP core settings (double click on the ip):

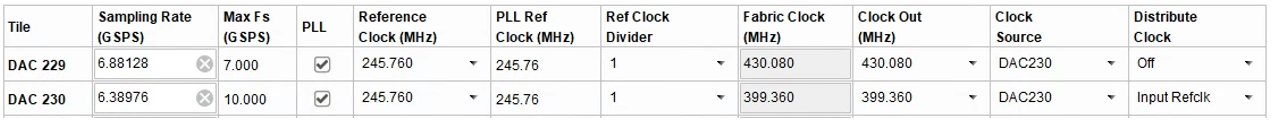
* “N\_DDS”=4

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:

The muxed sg ip is connected to a DAC in tile 229. Tile 230 clocking settings are also included below since it is referenced by tile 229.

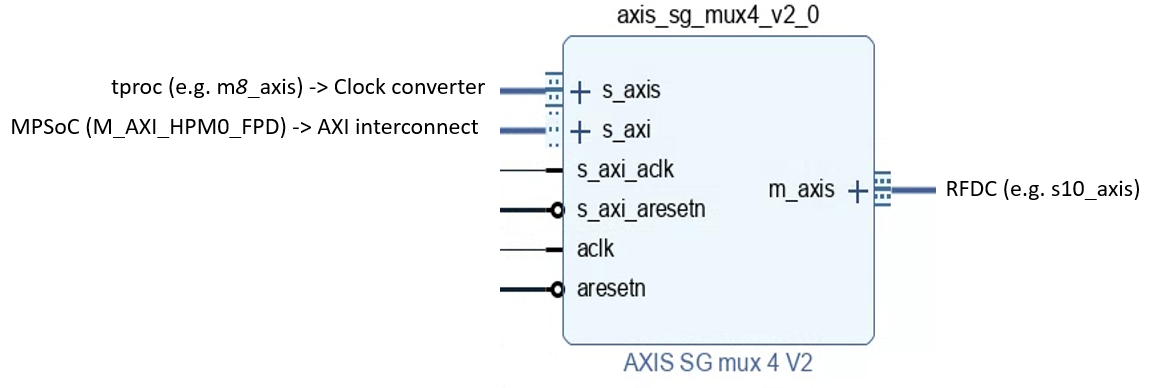


Wirings:

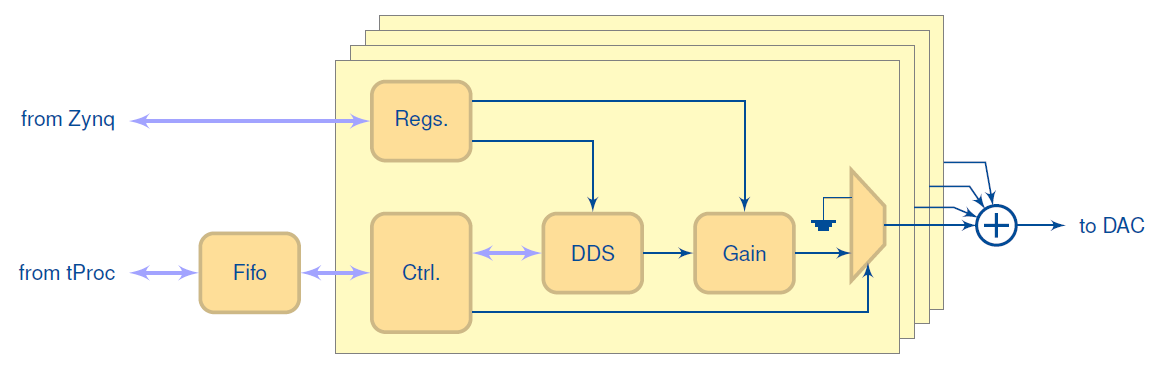
* The ”Clock converter” below is needed only if the clock speed of muxed sg (aclk) is different from that of tproc (aclk).
* For full details, you can re-create the vivado block design using the scripts (bd\_216 … .tcl, proj\_216 … .tcl) at:

<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*.



How it work (optional)



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)