ECE 551
HW5

- Due Fri Apr 19th by class time
- Work Individually on Problem 1
  - This is submitted to the dropbox
- Rest of HW can be done as a team
  - These problems are submitted via dropbox
  - **Please**…only **one** person submit all problems for the team!

HW5 Problem 1 (This problem in **individual** basis (rest is team))

1. (25pts) Post Synthesis Simulation of **I2S_Slave.vg**

Posted on the class webpage (under the tutorials section) is a file about Post Synthesis Validation (simulation). Read it carefully.

Individual problem and not covered by exercise

As part of HW4 (and Ex14) you created a synthesis script to synthesize **SPI_mstr**. Adapt that script to synthesize your **I2S_Slave.sv** and produce **I2S_Slave.vg**.. Now incorporate that gate level netlist into the test bench for I2S_Slave prove your post synthesis netlist works.

Submit to the dropbox:

a) Your **I2S_Slave.vg** gate level netlist that you simulated
b) Simulation results proving success. Waveforms or results from the self-checking testbench. Yes…you could turn in something here that fooled me into thinking you did this when you actually didn't. However, you would be hurting yourself, because you will need post synthesis simulation to work for the class project, and you can't fool me for that.

(**Note:** you may want to try post synthesis simulation of a few of your other blocks: PDM,SPI_mstr, … to ensure they are ready for the project)

HW5 Problem 2

This problem will be started as Ex18 on Weds Apr 10th

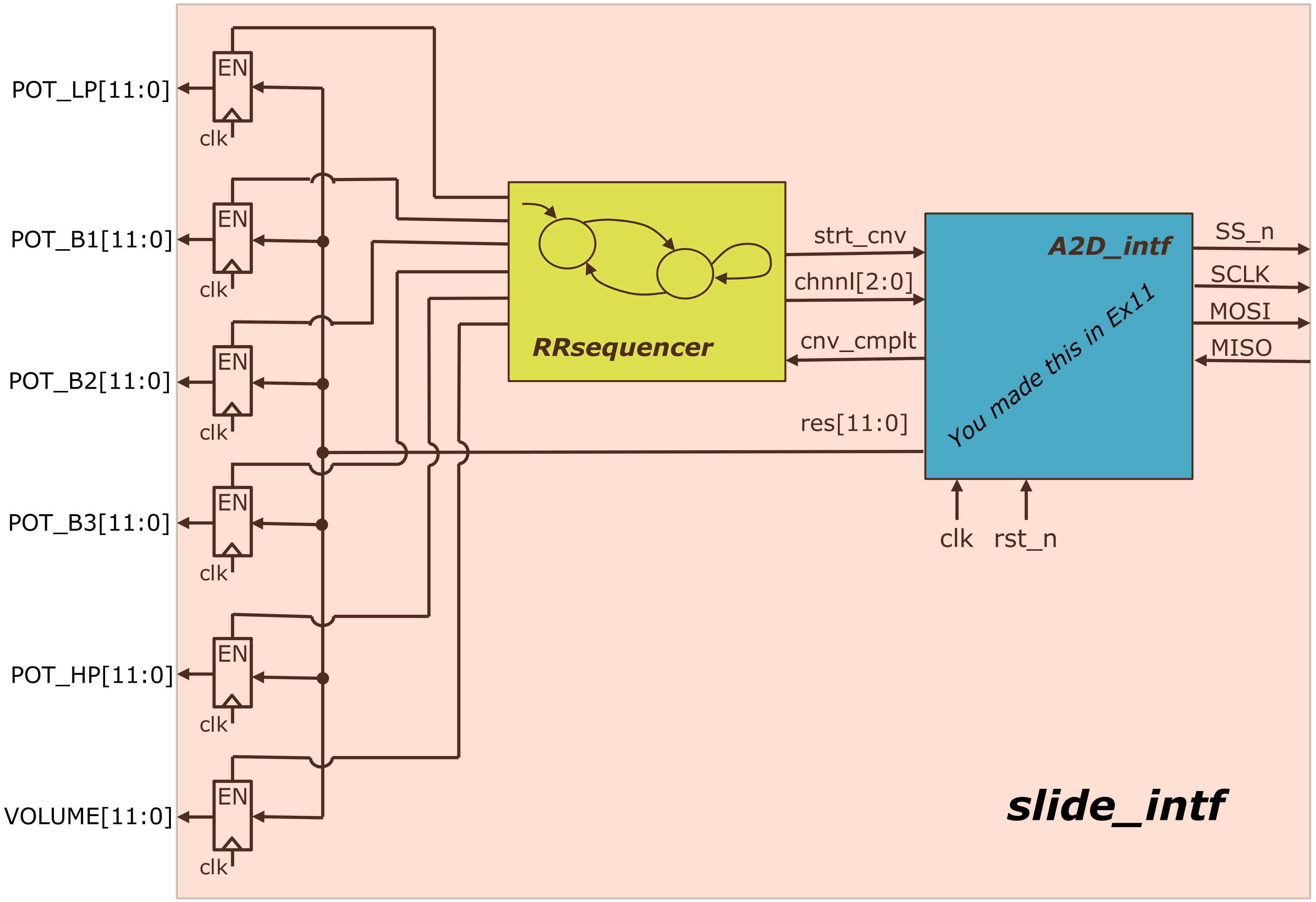
2. (20pts) Slide Potentiometer Interface

You have built and tested (on the DE0) **a2d_intf.sv** You will now use this block to build a higher level block (**slide_intf.sv**) that will simply provide the various potentiometer setting levels to your digital core.

This module will perform "round robin" conversions on the various A2D channels that are connected to the slide pots, and provide the potentiometer values as 12-bit digital numbers to the core.

A2D Channel:	Slide Pot:
001	Gain LP (gain of bass) → POT_LP[11:0]
000	Gain B1 (gain of 64Hz to 256Hz) → POT_B1[11:0]
100	Gain B2 (gain of 256 to 1kHz) → POT_B2[11:0]
010	Gain B3 (gain of 1k to 4kHz) → POT_B3[11:0]
011	Gain HP (gain of treble) → POT_HP[11:0]
111	Volume (used to scale over all left/right audio output) → VOLUME[11:0]

HW5 Problem 2 (continued) make it, test it, submit code for DUT and TB



HW5 Problem 3

This problem is started in Ex16 on Apr 8th

3.) (15pts) Speaker Drive

As part of Ex16 you will build **spkr_drv.sv**

To check off for Ex17 your team had to prove your **spkr_drv.sv** worked.

So…if your team checked off for Ex17 then you are done with this problem.

HW5 Problem 4 (Circular Queues)

Not covered by exercise, vital to project

4.) (40pts) Study the slides of project spec starting with the slide titled “What We Want to Build” through all the slides that discuss implementing the circular queues.

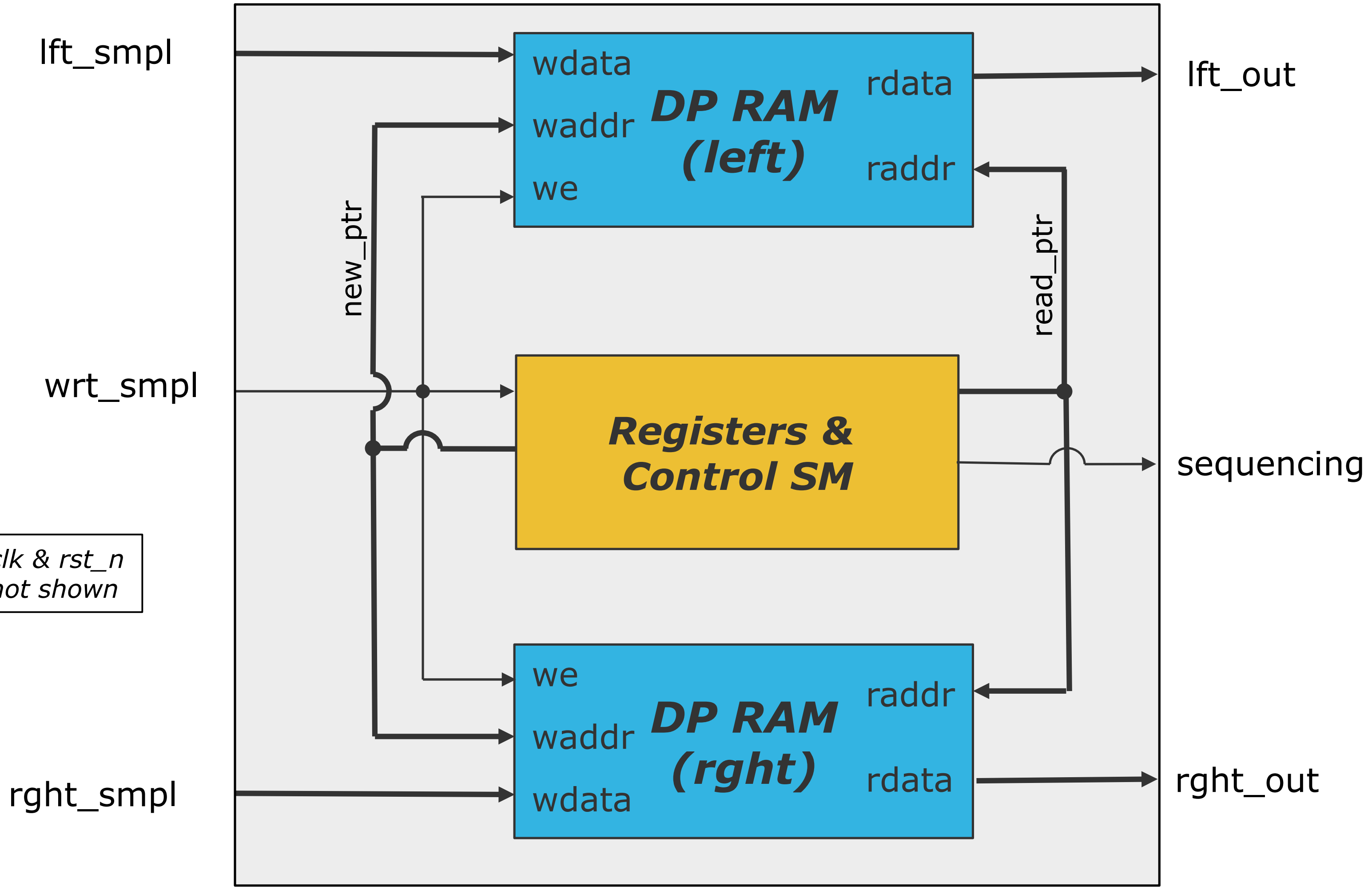
Understanding the need for the circular queues and their requirements. The need for two different queue sampling rates (44.1kHz & 22.05kHz), and the need for extra size of the 44.1kHz queue is one of the trickier things to understand.

There are two dual port memory models that are provided for use in building the circular queues. Two copies (left/right) of **dualPort1024x16.v** should be instantiated in your low frequency queue (used for bass & B1). Two copies (left/right) of **dualPort1536x16.v** should be instantiated in your high frequency queue (used for B2, B3 & treble).

We are at the point in the project development where your team has to start making micro-architectural decisions about the implementation of the equalizer digital core. However, I will present a few ideas about how I implemented the queues in my version of the project.

I added just a bit of “intelligence” into the queues themselves. As the I2S_Slave has audio samples ready (signified by the ***vld*** signal) they are written to the queues. When the queues become “full” they will start to automatically read out their contents from oldest sample to the (oldest + 1020) sample (1021 samples in total). While reading out these samples the FIR filters would be computing.

HW5 Problem 4 (Circular Queues)



clk & rst_n not shown

General Queue Structure

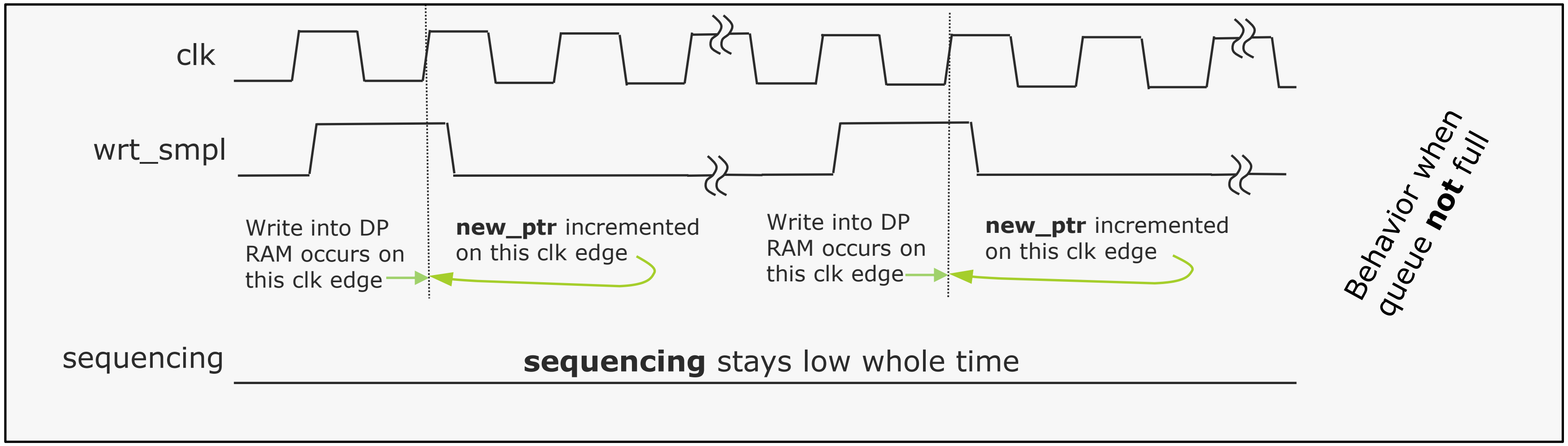
HW5 Problem 3 (Circular Queues..Hints)

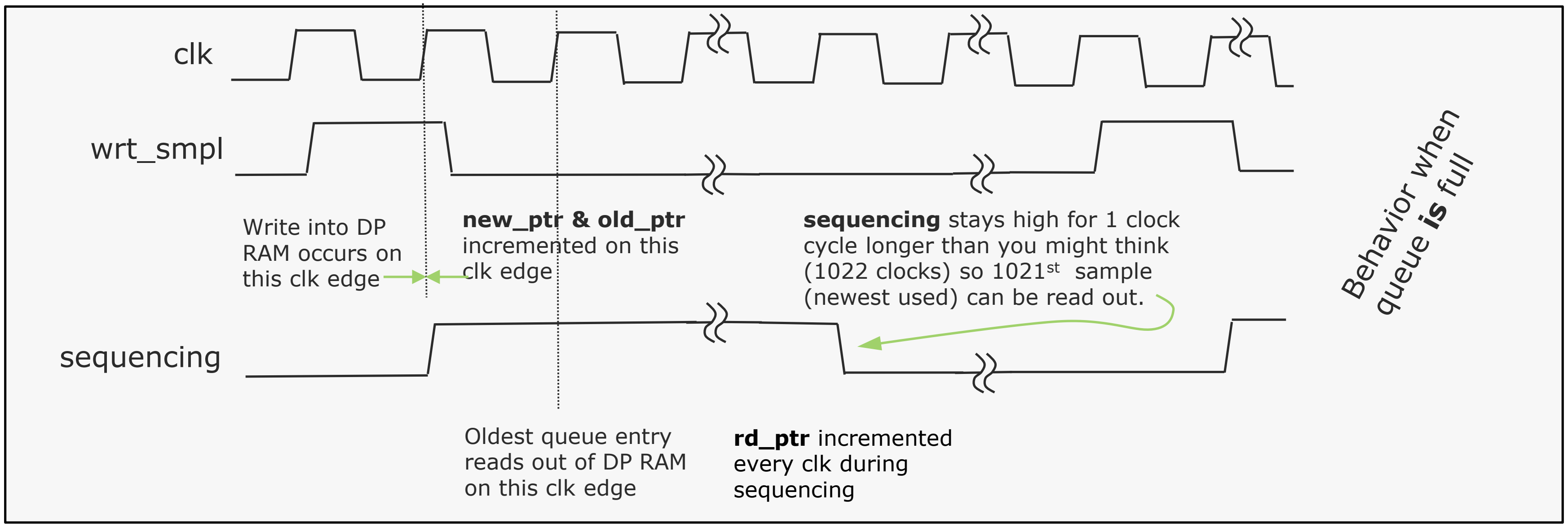
Signal:	Dir:	Description:
clk,rst_n	in	clk & reset
lft_smpl[15:0] rght_smpl[15:0]	in	Newest sample from I2S_slave to be written into queue
wrt_smpl	in	If high we are writing a sample and then starting a readout of 1021 samples from the oldest to the (oldest+1020). For high freq queue this is connected to a rising edge detector on ***vld*** (from I2S_Slave). For low freq queue this is every other rise edge of ***vld.***
lft_out[15:0] rght_out[15:0]	out	Once the queues are full a readout of 1021 samples (starting with the oldest samples) will be initiated every new **wrt_smpl**. **lft/rght_out**[15:0] is the data being read out.
sequencing	out	This signal is high the whole time the 1021 samples are being read out from the queue. Essentially this is the time the FIR filters would be calculating. NOTE: The dualport RAMs and the Coefficient ROMs read out on the next rising edge of clock from when the address to read changed.

NOTE: Once a queue is full the *old_ptr* and the *new_ptr* will both increment with every new sample written to the queue. One must be careful to ensure the pointers wrap around properly. This happens automagically for the low freq queue since the dual port RAM has a power of 2 number of entries. However, for the high freq queue you have to manage the pointers more carefully.

When is a queue full? For the low freq queue it is full when you have written the 1021st sample (address 1020). For the high freq queue it is full when you have written the 1531st sample (address 1530).

HW5 Problem 3 (Circular Queues..Hints)


clk
wrt_smpl
Write into DP RAM occurs on this clk edge
new_ptr incremented on this clk edge
Write into DP RAM occurs on this clk edge
new_ptr incremented on this clk edge
sequencing
sequencing stays low whole time
Behavior when queue **not** full


clk
wrt_smpl
Write into DP RAM occurs on this clk edge
new_ptr & old_ptr incremented on this clk edge
sequencing stays high for 1 clock cycle longer than you might think (1022 clocks) so 1021st sample (newest used) can be read out.
sequencing
Oldest queue entry reads out of DP RAM on this clk edge
rd_ptr incremented every clk during sequencing
Behavior when queue **is** full

HW5 Problem 3 (Circular Queues..Hints)

When reading out a queue's entries (during "**sequencing**") you will run a *read_ptr* from *old_ptr* to *old_ptr* + 1020. Keep in mind this might wrap around a boundary of the dualport RAM address space.

One can setup a pointer (lets call it *end_ptr*) to be *old_ptr* + 1020.

assign *end_ptr* = *old_ptr* + 1020;

Now *read_ptr* can be run from *old_ptr* up to and including *end_ptr* during "sequencing". This works just fine for the low freq queue since *old_ptr* + 1020 will wrap perfectly since there are a power of 2 number of entries in the dualport ram. However, for the high freq queue one has to be more careful about how *end_ptr* is constructed so it works out when it crosses the 1535 to 0 boundary.

Problem 3 (40pts)

Code and test your low and high frequency circular queues that will hold the audio samples. *The project will one if each of these queues. One holding left/right data to service LP & B1 FIRs, and one holding left/right data to service B2,B3,HP FIRs.*

The test benches should be self-checking if possible.

Submit the Verilog for the circular queues and the Verilog for the test benches to the dropbox.

Expected files:
high_freq_queue.sv, low_freq_queue.sv, high_freq_queue_tb.sv, low_freq_queue_tb.sv