ECE 551
HW5

- Due Monday April 20th by class time
- Work Individually

HW5 Problem 1

You are on your own for this problem

1. (25pts) Post Synthesis Simulation of your **SPI_RX.sv**

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

As part HW4 you created **SPI_RX.sv** and tested in in a self checking test bench. As part of HW4 you also wrote a synthesis script (for your UART). Adapt the synthesis script to apply to synthesizing you **SPI_RX** module and produce **SPI_RX.vg**. Then simulate the gate level netlist.

Submit to the dropbox:

a) Your **SPI_RX.vg** gate level netlist that you simulated
b) Simulation results proving success. 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 to ensure they are ready for the project)

HW5 Problem 2 (Note…this problem is worth 0pts)

This problem requires a reasonably high bandwidth low latency connection to the best-tux machines (using GUI interface). Since I can't count on you having that the points are now zero. This was intended to teach you what APR really is.

You are on your own for this problem

2. (0pts) APR of **UART.vg**

Posted on the class webpage (under the tutorials section) is a video on IC Compiler (APR how to video). There is also an accompanying .pdf with slides. Watch the video and follow along in the .pdf

Perform APR of your **UART.vg** netlist with an aspect ratio of 2:1 (twice as wide as high). **Submit a video** of the screen of your linux terminal with the result of your finished APR block. Include a head shot of yourself(s) in the video.

HW5 Problem 3

This problem will be started as Ex20 on Mon Apr 13th

3. (50pts) Command Interface

There is a slide in the project spec entitled "Digital Core Functionality (command processing)". It outlines the command/response portion of the Logic Analyzer. Look it over.

There is also a slide entitled "Register Set" that outlines all the registers in Logic Analyzer (their addresses, and their reset values). Look it over.

There is a slide entitled "Command Set (commands received over UART interface)". Look it over.

It makes sense (in my opinion) to have a module that processes the commands (received from host PC via **UART_wrapper**) and contains all the control/config registers.

You will be coding that block(**cmd_cfg**) here. (see next page)

HW5 Problem 3 (continued) (Interface of cmd_cfg block)

```
parameter ENTRIES = 384,   // defaults to 384 for simulation, use 12288 for DE-0
          LOG2 = 9;        // Log base 2 of number of entries

input clk,rst_n;
input [15:0] cmd;          // 16-bit command from UART (host) to be executed
input cmd_rdy;             // indicates command is valid
input resp_sent;           // indicates transmission of resp[7:0] to host is complete
input rd_done;             // indicates last byte of sample data has been read
input set_capture_done;    // from the capture module (sets capture done bit)
input [7:0] rdataCH1,rdataCH2;
input [7:0] rdataCH3,rdataCH4;
input [7:0] rdataCH5;

output [7:0] resp;             // data to send to host as response
output send_resp;              // used to initiate transmission to host (via UART)
output clr_cmd_rdy;            // when finished processing command use this to knock down cmd_rdy
output strt_rd;                // asserted to fire off a read of channel RAMs
output [LOG2-1:0] trig_pos;    // how many sample after trigger to capture
output reg [3:0] decimator;    // goes to clk_rst_smpl block
output reg [7:0] maskL,maskH;               // to trigger logic for protocol triggering
output reg [7:0] matchL,matchH;             // to trigger logic for protocol triggering
output reg [7:0] baud_cntL,baud_cntH;       // to trigger logic for UART triggering
output reg [5:0] TrigCfg;                   // some bits to trigger logic, others to capture unit
output reg [4:0] CH1TrigCfg,CH2TrigCfg;     // to channel trigger logic
output reg [4:0] CH3TrigCfg,CH4TrigCfg;     // to channel trigger logic
output reg [4:0] CH5TrigCfg;                // to channel trigger logic
output reg [7:0] VIH,VIL;                   // to dual_PWM to set thresholds
```

HW5 Problem 3 (continued) (Handling dump channel)

The commands to read and write the control/config registers are pretty straight forward to implement. Still pay attention to getting the posAck negAck stuff to work, and that you are resetting the registers to the proper value.

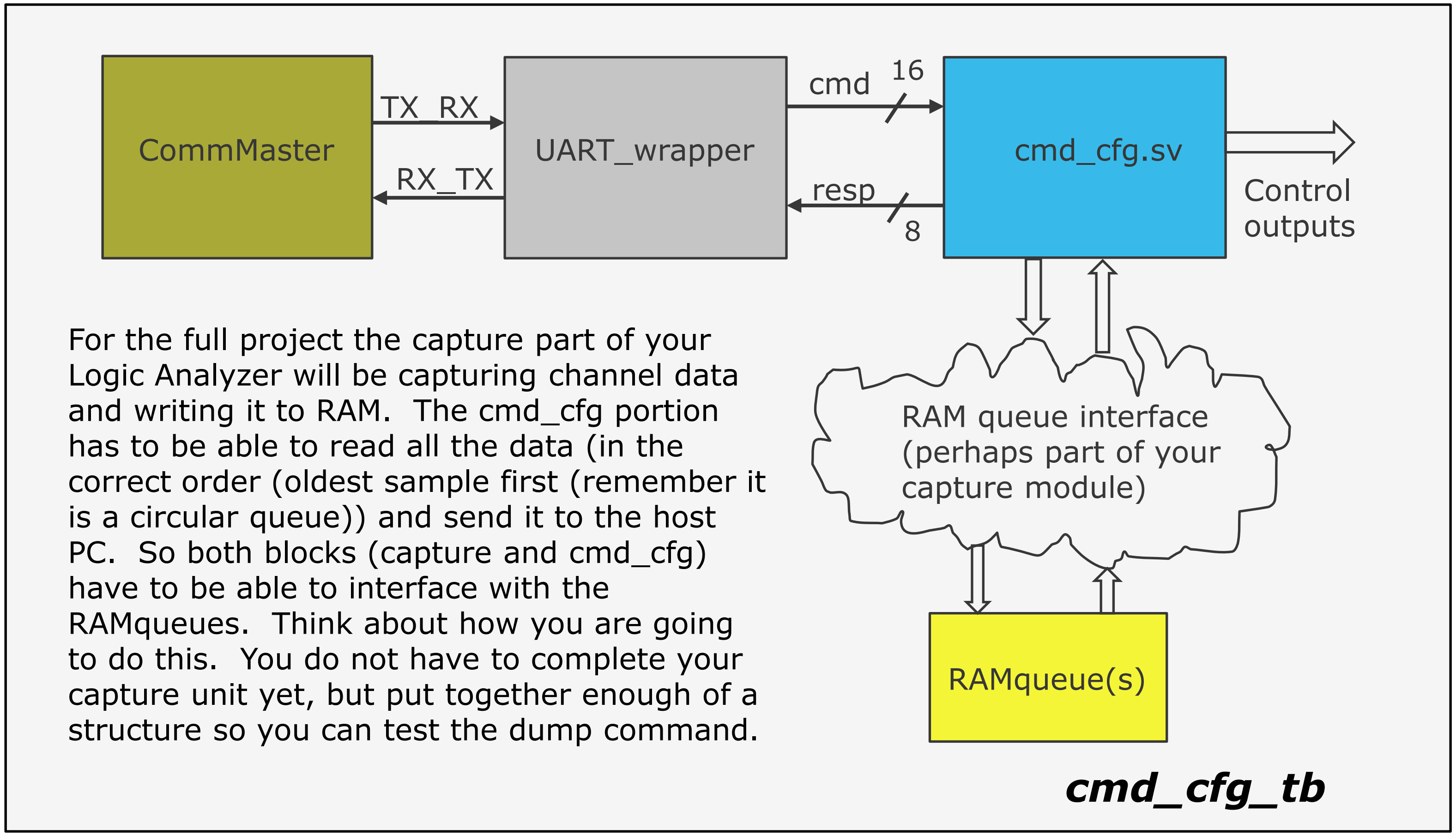
The trickier command is the channel dump. Dumping involves reading all the locations from one of five RAMqueues and writing the bytes as responses to the UART_wrapper.

This is interesting because writing to the RAMs is really controlled by the "capture" part of the logic analyzer (which we have not implemented yet). So which block controls the address to the RAMs?

Look at the slide "Digital Core Functionality (Channel Capture).

Decide as a team how you are going to interface to the RAMqueue blocks. Partition your interface for cmd_cfg so the channel dump command will work.

HW5 Problem 3 (continued) (Testing cmd_cfg)



For the full project the capture part of your Logic Analyzer will be capturing channel data and writing it to RAM. The cmd_cfg portion has to be able to read all the data (in the correct order (oldest sample first (remember it is a circular queue)) and send it to the host PC. So both blocks (capture and cmd_cfg) have to be able to interface with the RAMqueues. Think about how you are going to do this. You do not have to complete your capture unit yet, but put together enough of a structure so you can test the dump command.

Submit to the dropbox: Your code for **cmd_cfg.sv**. Your self checking testbench **cmd_cfg_tb.sv** and any supporting files. Also submit proof that your testbench ran.

HW5 Problem 4 (Channel Sample Logic)

This problem will be started as Ex19 on Weds Apr 8th

4. (25pts) Channel Sample Logic

There is a slide in the project spec entitled “Capture Logic (50MHz to 400MHz to 100MHz)”. It outlines the clocking of frontend which operates at 400Mhz. Make an effort to understand what is going on with this logic.

The next slide is entitled “Channel Sample Logic” it shows a partitioning of logic for sampling channel data. This logic would be repeated 5 times in your design for the 5 channels. Study this slide and understand what it is you need to make.

Finally there is a slide entitled “clk_rst_sample Block”. This is a block that is provided to you. Look at the .zip in Exercise19 area of Canvas.

There are other files provided as well to aid in testing.

HW5 Problem 3 (Channel Sample Logic) (continued)

4. Create **channel_sample.sv** that encodes the functionality of the channel sampling logic. The interface should be similar to that shown in the table below:

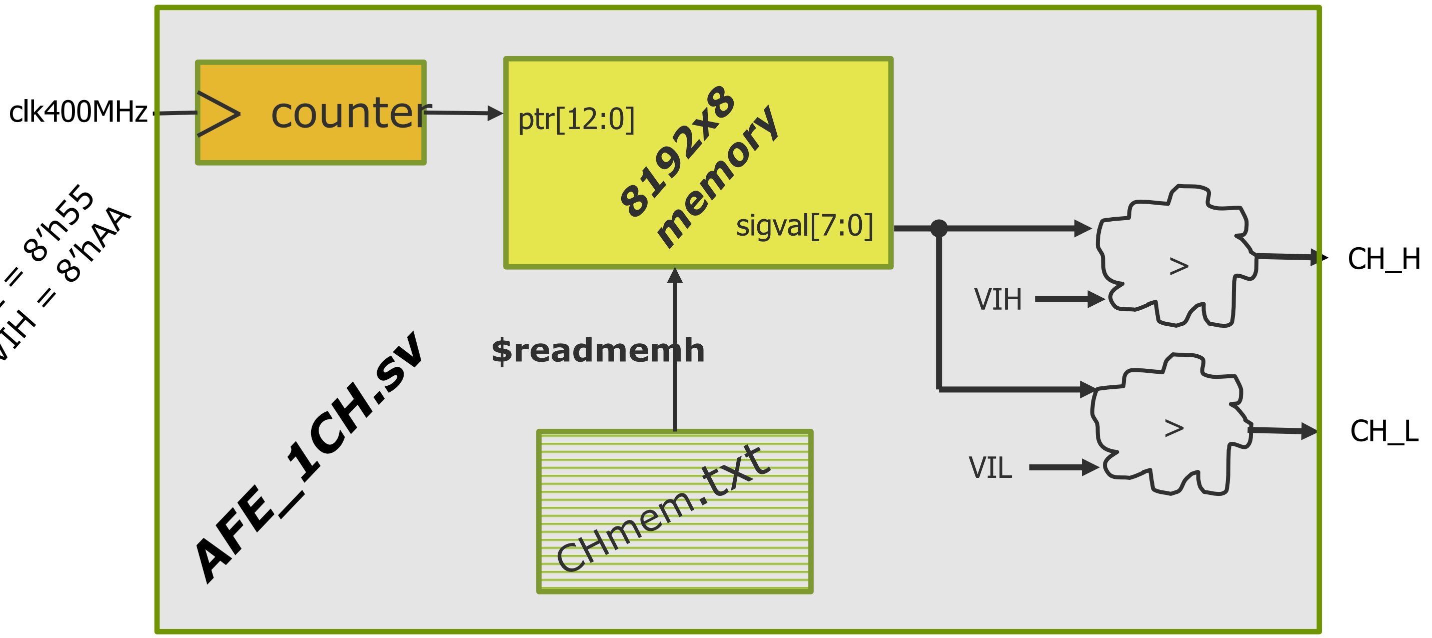
Signal:	Dir:	Description:
smpl_clk	In	This is the decimated clock from clk_rst_smpl. Samples are captured on the negative edge of this clock
clk	In	100MHz system clock. This is used to flop the 8-bit accumulation of 4 2-bit samples.
CH_H, CH_L	In	These are the unsynchronized channel samples from the comparators comparing against VIH, VIL
CH_Hff5 CH_Lff5	Out	These are the 5th flopped versions of the channels. They go on to the trigger logic
smpl[7:0]	Out	This is the 8-bit sample that will get written to the RAMqueue for that channel. It is a collection of 4 2-bit samples

The bigger job will be testing **channel_sample.sv**

HW5 Problem 3 (Channel Sample Logic) (AFE_1CH)

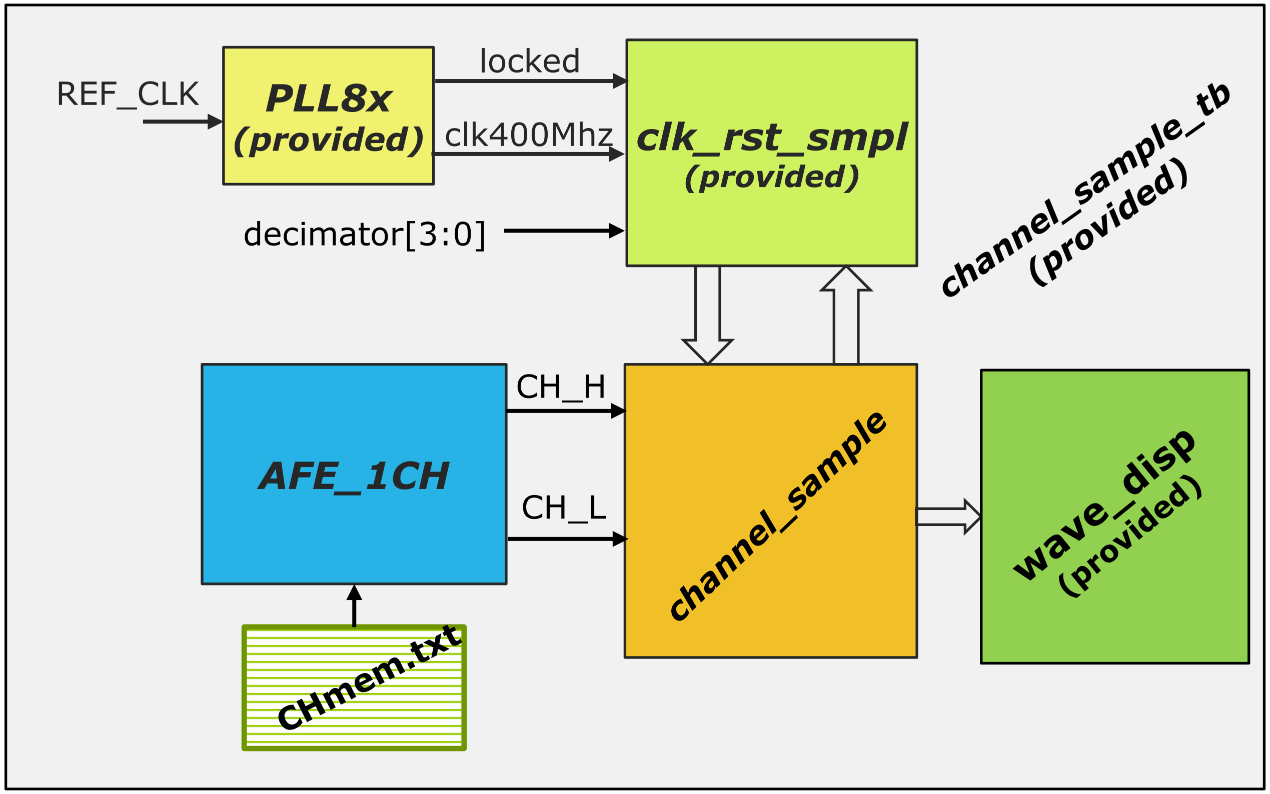
- You will next build a model of the **A**nalog **F**ront **E**nd for a single channel (**AFE_1CH.sv**).
- This block models both the signal the logic analyzer is connected to plus the comparators that compare this signal to VIH & VIL.
- How do we model the signal the logic analyzer is connected to?
- Sometimes it makes sense to read data from a file. **$readmemh**

NOTE:
localparam VIL = 8'h55
localparam VIH = 8'hAA



HW5 Problem 3 (Channel Sample Logic) (testing)

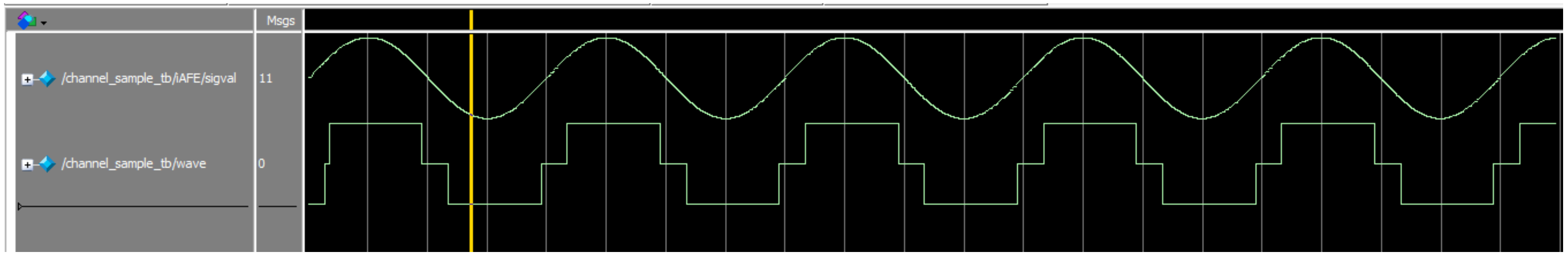
- A test bench and all the needed collateral files is provided.
- This testbench is not self-checking, however, you can display both the "analog wave" coming out of your memory inside AFE_1CH to the wave reconstructed by wave_disp, and get a quick visual confirmation the unit is working.
- See the next slides for how to display waves as analog within ModelSim
- Play with VIH and VIL settings. Play with decimator settings. Do results change as expected.



HW5 Problem 3 (Channel Sample Logic) (testing)

- There are a number of signals you might want to plot in the debug process. In the end, however, you will plot **sigval** (output from memory in ***AFE_1CH***) and **wave** (the reconstructed wave from ***wave_disp***) as analog.

- The image below shows my results.



Submit to the dropbox:

channel_sample.sv
AFE_1CH.sv
Some proof that you ran your test bench.