ECE 352 AHW3

You may work with your assigned partner, or alone, as arranged by your “lab” section. If you work with a partner, we expect you to work together on the complete homework rather than sub-divide the tasks.

If you work with a partner, **only one** of you submit the requested files. The other should simply submit a file or comment stating who their partner was.

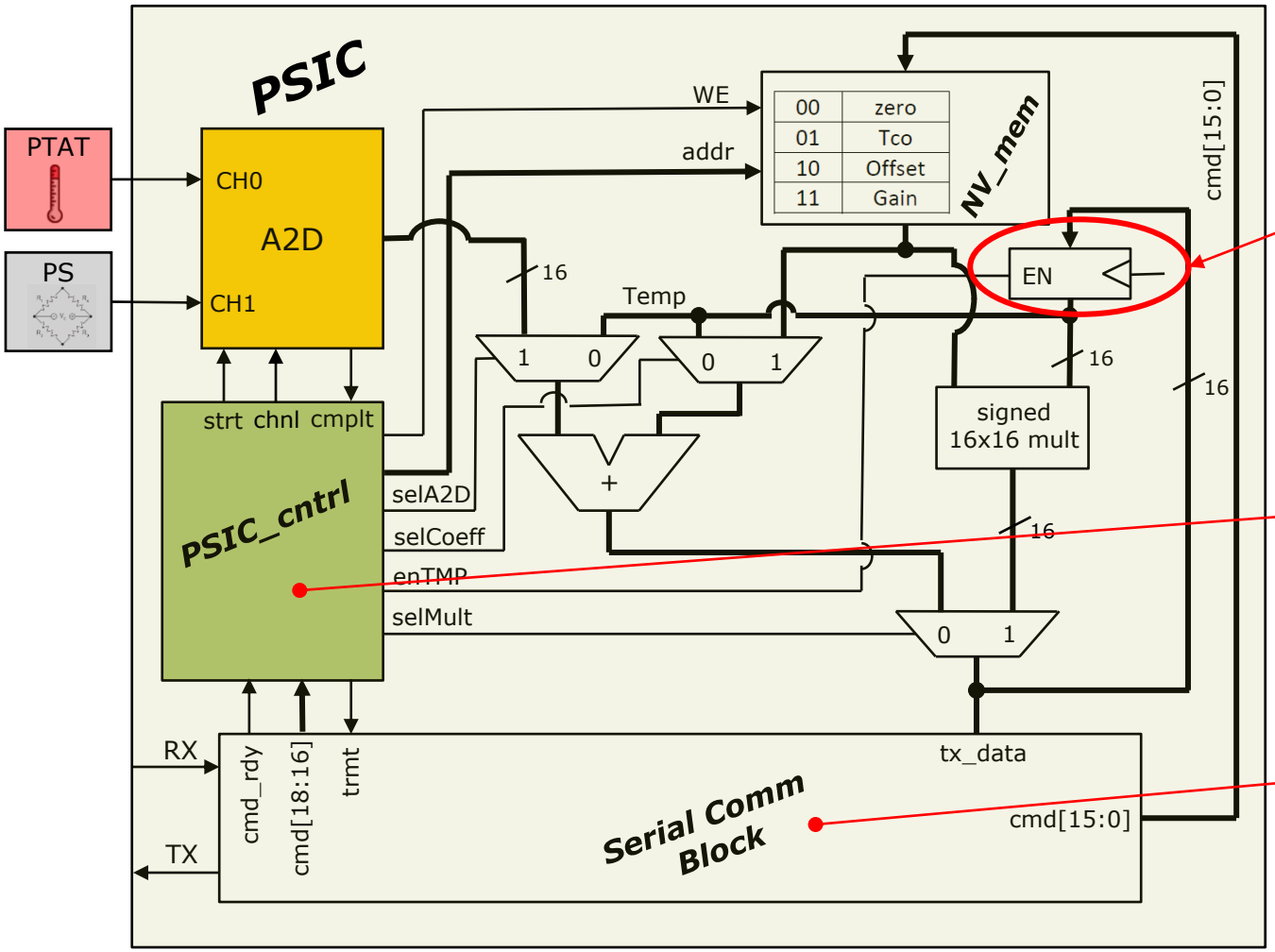
All our remaining AHW’s for this semester will work toward building up the components of a complex digital system. Our design this semester will be a:

Temperature Compensated

Pressure Sensor

Block Diagram of PSIC (**Where are our sequential elements?**)

In this applied HW we will take care of designing the sequential elements needed for the PSIC system. We will also finish off the datapath block.



The only sequential element you can readily see in the block diagram is a 16-bit wide temporary register (**Temp**)

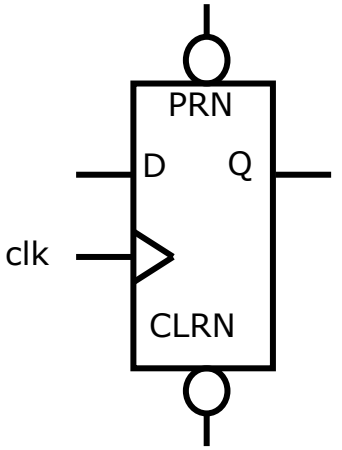
There are of course a few more sequential elements in the design embedded in the sub-modules.

The main control state machine has state flops

Serial_comm requires both state flops and some 8-bit buffering registers.

State flops (state6_reg)

- For ease of implementation when doing state machine designs by hand we use one-hot state machines. We will do that here as well. The main control statemachine (**PSIC_cntrl**) will require 6-states.
- The reset state of a 6-bit state register should be 6'b000001. Meaning the upper 5-bits are reset and the LSB is preset.
- The verilog for a single D flipflop with asynch active low preset (***PRN***) and asynch active low reset (***CLRN***) is provided (**d_ff.sv**).
- This should be implemented as a vectored instantiation of 6 **d_ff** cells (one line instantiation). To get the reset and preset signals correct you will have to make a couple of 6-bit reset/preset vectors.
- You will find a file called **state6_reg.sv** in the .zip. Flesh out the missing Verilog to implement the design.



single **d_ff** cell

State flops (state6_reg *continued* & state3_reg)

- There is a self checking test bench (**state6_reg_tb.sv**) available. Launch ModelSim, compile, and debug till you pass the test bench. Recall you will have to also add **d_ff.sv** to the project.
- Capture all the waves at the DUT level for the length of the simulation (**state6_reg_sim.jpg**). Also capture the "YAHOO! test passed" message from the transcript window (**state6_reg_yahoo.jpg**)
- We will also need a 3-bit state flop inside **serial_comm**. Copy your **state6_reg.sv** to **state3_reg.sv** and modify it to be a 3-bit state register.
- Modifying **state6_reg** to form **state3_reg** was simple enough and you made no mistakes…right? Its not that we don't trust you…its just that humans suck. Prove your **state3_reg.sv** works by running it through the provided **state3_reg_tb.sv**. Capture the "YAHOO! test passed" message (**state3_reg_yahoo.jpg**).

- Submit: **state6_reg.sv**, **state6_reg_sim.jpg**, **state6_reg_yahoo.jpg**, **state3_reg.sv**, & **state3_reg_yahoo** to the dropbox for AHW3.

State flops (state2_reg…you need this too)

- In addition to **state6_reg** and **state3_reg** you are also going to need a **state2_reg**.
- Producing and testing **state2_reg** will not be part of the assignment. However you should do that now so it is ready for AHW4. Copy your **state3_reg.sv** to **state2_reg.sv** and modify.
- See if you can understand what **state3_reg_tb.sv** does and copy that to **state2_reg_tb.sv** an modify and test.

D Flip-Flop with Enable.

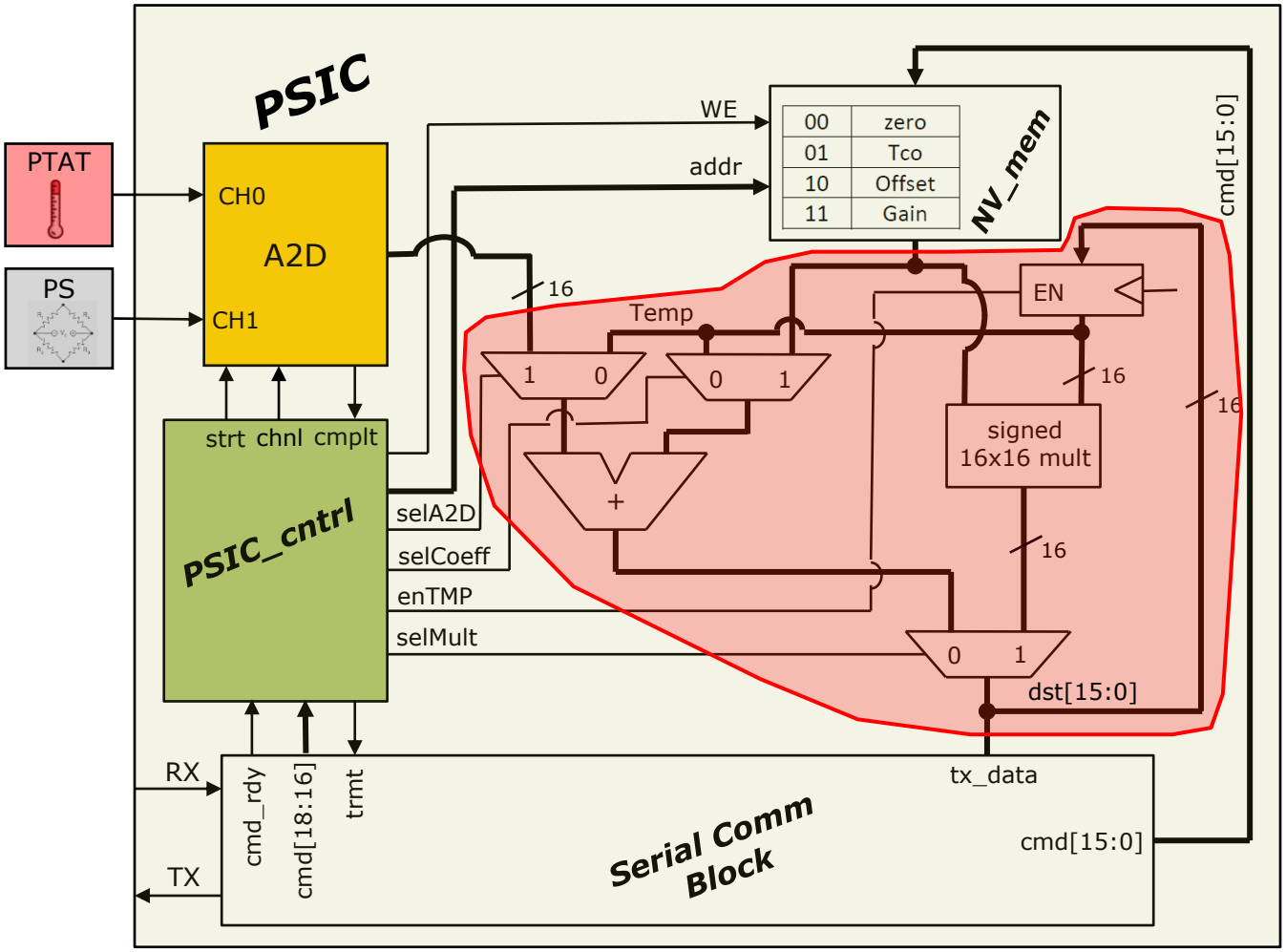
- A Verilog model of a basic D flip-flop has been provided (**d_ff.sv**). Your task is to add logic (that feeds the **D** input) to produce a D flip-flop with an enable (**EN**). When **EN** is asserted the next value of the **Q** output should be what is present on the **D** input. If **EN** is low the **Q** output should hold its current value.
- The active low preset of **d_ff** should be tied off inactive, but the active low reset (**CLRN**) of **d_ff** will still be used in our new resulting cell (**d_en_ff**).
- The last page of this document contains a K-map for you to fill out to determine the logic to feed the D input of the basic d_ff.sv. Fill it out and take a picture (**D_FF_Kmap.jpg**)
- A Verilog shell (**d_en_ff.sv**) is provided. It instantiates and connects a simple flop. Use structural Verilog (instances of Verilog primitive gates) to create the $\boldsymbol{D_{NEXT}}$ logic.
- A self-checking testbench (**d_en_ff_tb.sv**) is also provided. Using ModelSim compile and debug your design. When your design passes the testbench use the snipping tool to grab an image of the signals of the DUT for the entire length of the simulation (**d_en_ff_sim.jpg**). Also grab a screen image of the transcript window showing the "YAHOO! test passed" message (**d_en_ff_yahoo.jpg**).
- Submit: **D_FF_Kmap.jpg**, **d_en_ff.sv**, **d_en_ff_sim.jpg**, & **d_en_ff_yahoo.jpg**

en_reg8 and en_reg16

- The next part is quite simple. You are going to produce **en_reg8.sv** & **en_reg16.sv** by vectored instantiation of **d_en_ff**.
- A shell (**en_reg8.sv**) is provided. Add the missing code to implement an 8-bit wide enabled register. Since this is just a simple vectoring of the function of **d_en_ff** we are going to trust you did it right. There is no testbench.
- Now copy **en_reg8.sv** to a new file **en_reg16.sv**. Modify this to be a 16-bit wide enabled register.
- We will test this design as part of the context of the entire **datapath** of PSIC which is what you will do next.
- Submit **en_reg8.sv** and **en_reg16.sv**

Finishing off datapath.

The portion outlined in red below will become a sub-module called **datapath.sv**.



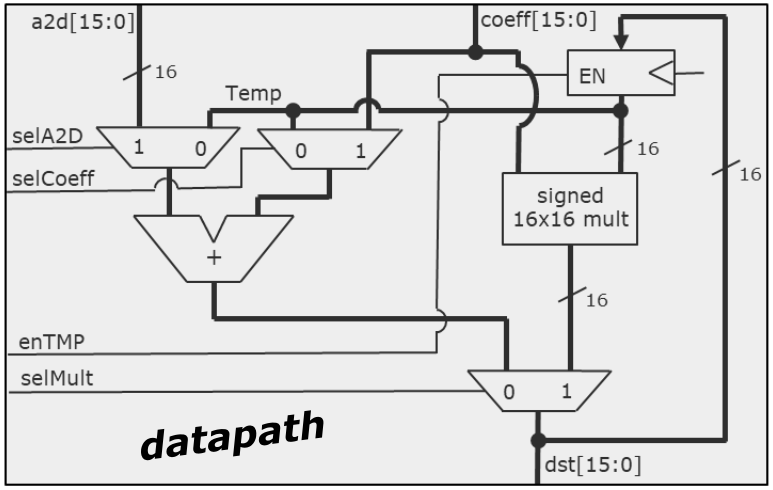
In AHW2 you produced the adder (**satAdd.sv**) and the multiplier (**satMult.sv**). Copy these blocks forward to your AHW3 folder.

Now that you have the implementation for the **Temp** register (**en_reg16**) you can complete **datapath.sv**

A shell for **datapath.sv** is provided.

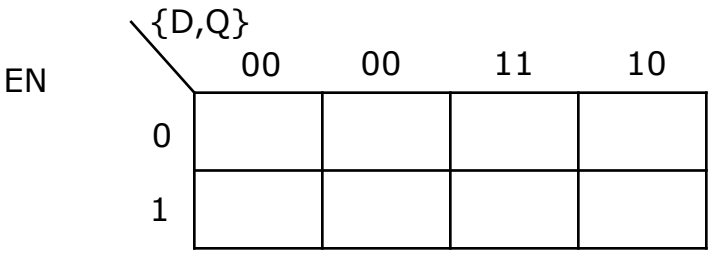
Completing and Testing datapath.sv

- Using the provided shell (**datapath.sv**) complete the code for by instantiating **en_reg16.sv** to form the **Temp** register.
- Then infer the two muxes feeding **satAdd** via dataflow statements.
- Instantiate both satAdd and satMult.



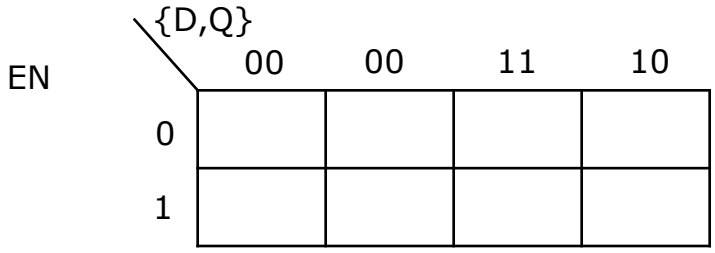
- Finally infer the **dst** mux selecting between add and mult via a dataflow statement.
- A self-checking testbench (**datapath_tb.sv**) is provided. Produce a ModelSim project and simulate/debug the design. This will also serve to test the functionality of **en_reg16** since it is part of the design.
- Capture the waveforms of datapath for the entire length of the simulation (**datapath_sim.jpg**). Also capture the "YAHOO test passed!" message (**datapath_yahoo.jpg**).
- Submit **datapath.sv**, **datapath_sim.jpg**, & **datapath_yahoo.jpg**

Kmaps (print, fillout, and submit an image **D_FF_Kmap.jpg**)



D_{NEXT} = _____

Another set below in case you messed up



D_{NEXT} = _____