ECE 352 AHW5

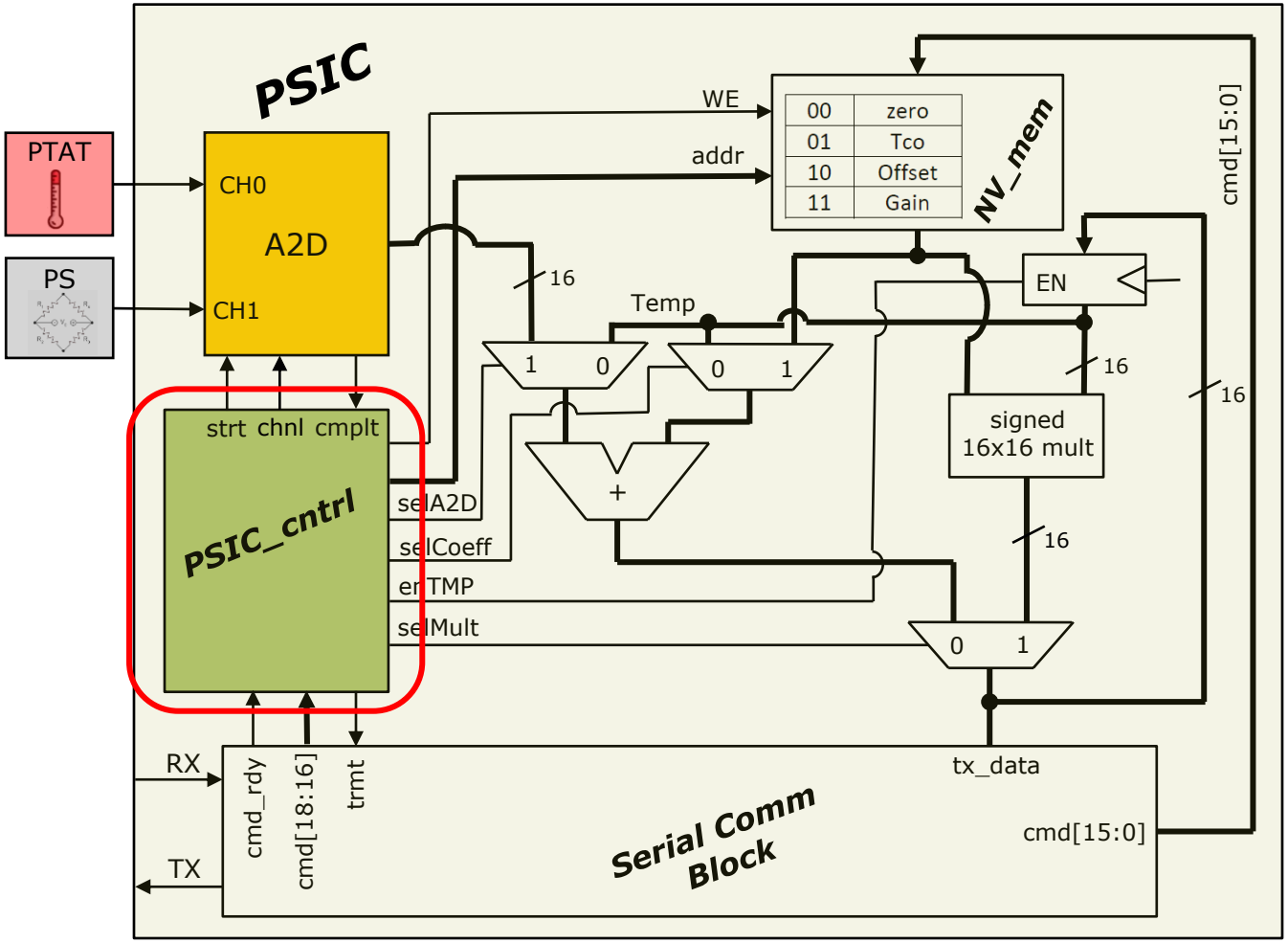
You may work with your assigned partner, or alone, as arranged by your “lab” section. If you work with a partner, we expect you to work together on the complete homework rather than sub-divide the tasks.

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.

Temperature Compensated Pressure Sensor

Block Diagram of PSIC (**This AHW focusses on PSIC_cntrl**)

In this applied HW we will implement ***PSIC_cntrl*** which interprets and executes the incoming command (**cmd**[18:16]). If the command is read pressure it orchestrates the computation.



The calculation performed to compensate the raw **P**ressure **S**ensor reading is:

P_{comp} = (PS_{raw} + Offset + T_{co}*PTAT)*Gain

Where:

PS_{raw} = Raw pressure sensor reading from A2D

Offset = Offset coefficient stored in NV_mem

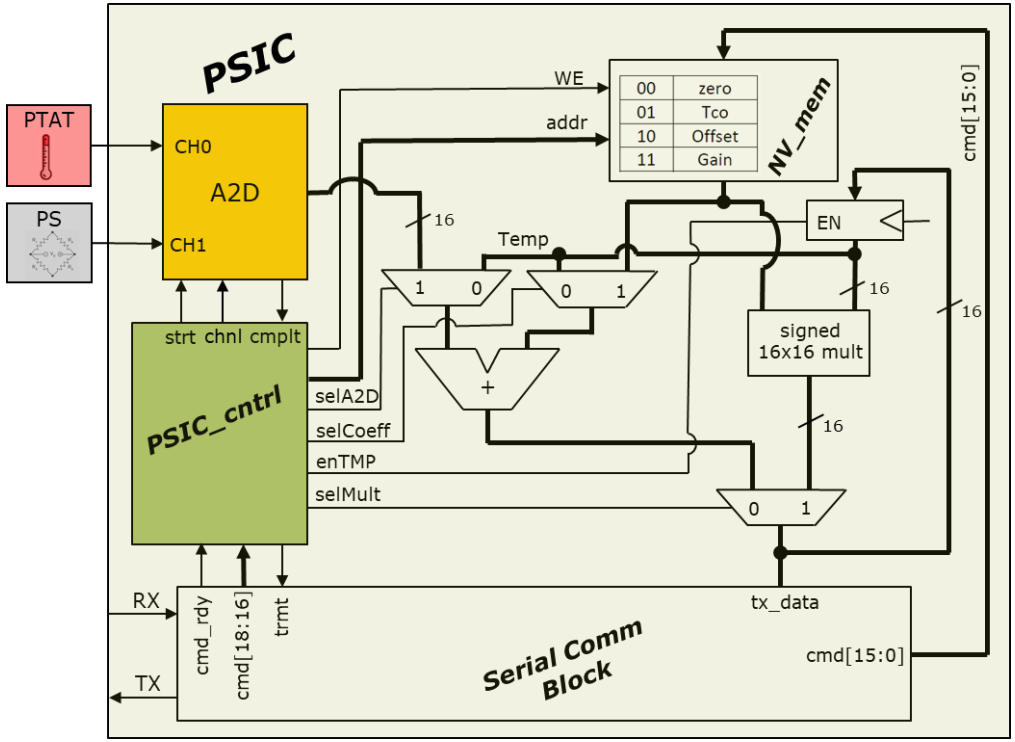
T_{co} = Temperature offset coefficient stored in NV_mem

PTAT = Temperature reading from A2D *(Proportional To Absolute Temperature)*

Gain = Gain (span) coefficient stored in NV_mem

Order of Operation of Calculations

$P_{comp} = (PS_{raw} + Offset + T_{co}*PTAT)*Gain$



Study the datapath we have.

Note a single temporary register (**Temp**) used to store intermediate calculations.

Note the connections from the A2D, Temp register, and NV_mem.

Note there is a hardcoded zero value in NV_mem that can be accessed

Given this datapath determine the order of operations needed to perform the compensation calculation.

There is a table at the end of this document. Print it and fill it out (legibly) for what operations need to be performed in what order and what control signals are pertinent to that step.

There might be more lines in the table than you need. There is not a single correct answer (however there are a very limited number of ways that work). This is more a thinking tool for you.

Take a picture of your completed table (**op_table.jpg**) and submit it to the dropbox.

Commands PSIC_cntrl can execute

cmd[18]	cmd[17:16]	cmd[15:0]	Description:
0	XX	0xXXXX	Perform fresh A2D conversions on both PTAT and PS and perform correction equation and write result to ***serial_comm*** (**tx_data**).
1	00	0xDATA	Writes the 16-bit value present on cmd[15:0] to the 'zero' address of NV_mem. *This particular command is unlikely to be used since address 00 of NV_mem initializes to zero and would typically be kept at that value in normal operation.*
1	01	0xDATA	Writes the 16-bit value present on cmd[15:0] to Tco coefficient
1	10	0xDATA	Writes the 16-bit value present on cmd[15:0] to Offset coefficient
1	11	0xDATA	Writes the 16-bit value present on cmd[15:0] to Gain coefficient

Although the command is 24-bits wide, bits [23:19] are ignored. When **cmd_rdy** is asserted ***PSIC_cntrl*** will inspect **cmd[18]:**

- If it is zero it performs fresh A2D conversions on PTAT and PS and computes a new compensated pressure reading for output via ***serial_comm***.
- If it is one a write to NV_mem is being performed. The data to be written is present in cmd[15:0] and the address to write is formed by cmd[17:16]. To write to ***NV_mem*** ***PSIC_cntrl*** simply has to ensure the proper address and assert WE for 1 clock cycle. It also needs to send a response via ***serial_comm***. The data it sends does not matter, but a response does need to be initiated.

Using the above description of commands and your table outlining computation steps draw a state diagram (bubble diagram) for ***PSIC_cntrl***. Use proper state diagram notation and make it neat.

Take a picture of your bubble diagram (**PSIC_bubble.jpg**) and submit it to the dropbox for AHW5.

You should be able to name that tune in 6 states, however, you are free to use more.

PSIC_cntrl verilog

A shell for ***PSIC_cntrl*** is provided (**PSIC_cntrl.sv**). Flesh out the operations of the main SM. *If your machine will use more than 6-states you also need to widen the state flops accordingly.*

Once you have **PSIC_cntrl.sv** fleshed out and compiling in ModelSim you are ready to test. Instead of testing ***PSIC_cntrl*** in a dedicated testbench we will test it at the full system level. In addition to the provided files in the AHW5_Files.zip you will need to copy many files forward from AHW2, AHW3, & AHW4. Some of these files include: **d_en_ff.sv**, **datapath.sv**, **en_reg8.sv**, **en_reg16.sv**, **FA.sv**, **RCA16.sv**, **rcv_SM.sv**, **satAdd.sv**, **satMult,sv**, **serial_comm.sv**, **state2_reg.sv**, **…**

There is a toplevel file (**PSIC.sv**) provided that wraps all the elements of the design together.

There is an **incomplete** testbench (**PSIC_tb.sv**) that performs a few basic tests of the top level.

Signal:	Dir:	NOTE:
clk, rst_n	in	clk & reset
cmd_rdy	in	Pulsed high 1 clock
cmd[2:0]	in	Represents cmd[18:16]
trmt	Out	Asserted for 1 clock to initiate transmission of tx_data
WE	out	To NV_mem to initiate write 1 clock wide
addr[1:0]	Out	Address to write in NV_mem
selA2D	Out	Selects A2D result over Temp
selCoeff	Out	Selects NV_mem over Temp
selMult	Out	Selects Mult as result of DP
enTmp	Out	Enable write of Temp reg
strt_cnv	out	1 clock wide pulse initiates conversion of A2D
chnl	Out	0=PTAT, 1=Pressure
cnv_cmplt	in	1 clock wide, A2D has data

NOTE: the provided testbench is said to be incomplete because it only performs 3 very basic tests. These tests do not include non-zero values for **Offset** or **Tco**. They do not include any tests of saturation of addition or multiplication.

Study how the first 3 tests are done. Then using copy/paste/modify techniques add many more tests to test the full functionality of the PSIC. *(including ensuring saturation cases work).*

When your entire design is passing your augmented testbench capture an image of your transcript window showing it passes all your tests. (**PSIC_cntrl_yahoo.jpg**). Don't try to fool us because we will be dropping your **PSIC_cntrl.sv** into our testbench to ensure it works.

Submit: **PSIC_cntrl.sv**, **PSIC_tb.sv**, and **PSIC_cntrl_yahoo.jpg** to the dropbox for AHW5

Order of Operation of Calculations

Step:	Operation Description:	Control/Status Signals to Consider:
1		
2		
3		
4		
5		
6		
7		

Adding new tests.

The values for raw A2D readings (both ***PTAT*** and ***PRESSURE***) can be found and modified in the file **A2D.sv**. Look at that file now.

The first test in the provided testbench was a write of a unity gain coefficient to the NV_memory. It therefore did not do any A2D conversions and did not "consume" any of the values in the **A2D.sv** file.

The second test in the provided testbench requested a corrected pressure reading, but at the time ***OFFSET*** and ***TCO*** coefficients were both zero and the *GAIN* was unity so the corrected value was simply the same as the raw pressure reading. If you look at **A2D.sv** you see ***PRESSURE[0]*** = 16'h3456. So that is the expected result.

The third test in the provided testbench writes the ***GAIN*** coefficient to 1.25 and then requests a conversion again. Now the raw A2D reading for pressure will ***be PRESSURE[1]*** = 16'3ABC. If you multiply 0x3ABC by 1.25 you would get 0x496B. So that is the expected result.

Lets talk about a hypothetical 4th test that you will add. You will test with a small positive ***OFFSET***. You will also edit **A2D.sv** and change ***PRESSURE[2]*** to be 16'h5678. (You are allowed to edit the values in **A2D.sv**). We will use an offset of 16'h000C.

First you would send a command to write the ***OFFSET*** value. **cmd2send** = 24'h06_000C;
Then you would wait for the response (while !sensor_rdy line from testbench).
Then you would send the command to get a corrected pressure reading. **cmd2send** = 24'h00_5000
What result do you expect? (remember ***GAIN*** was last set to 1.25, and ***TCO*** is still zero)
Expected = (16'h5678 + 16'h000C)*1.25 = 16'h6C25

From there you can introduce a non-zero ***TCO*** and check those results.

Then there is a check of saturation cases.