ECE 352 AHW2

You may work with your assigned partner, or alone, as arranged by your "lab" section TA after AHW1 grading. 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

Motivation & High Level Description:

- Interfacing digital systems to the “real world” requires sensors.

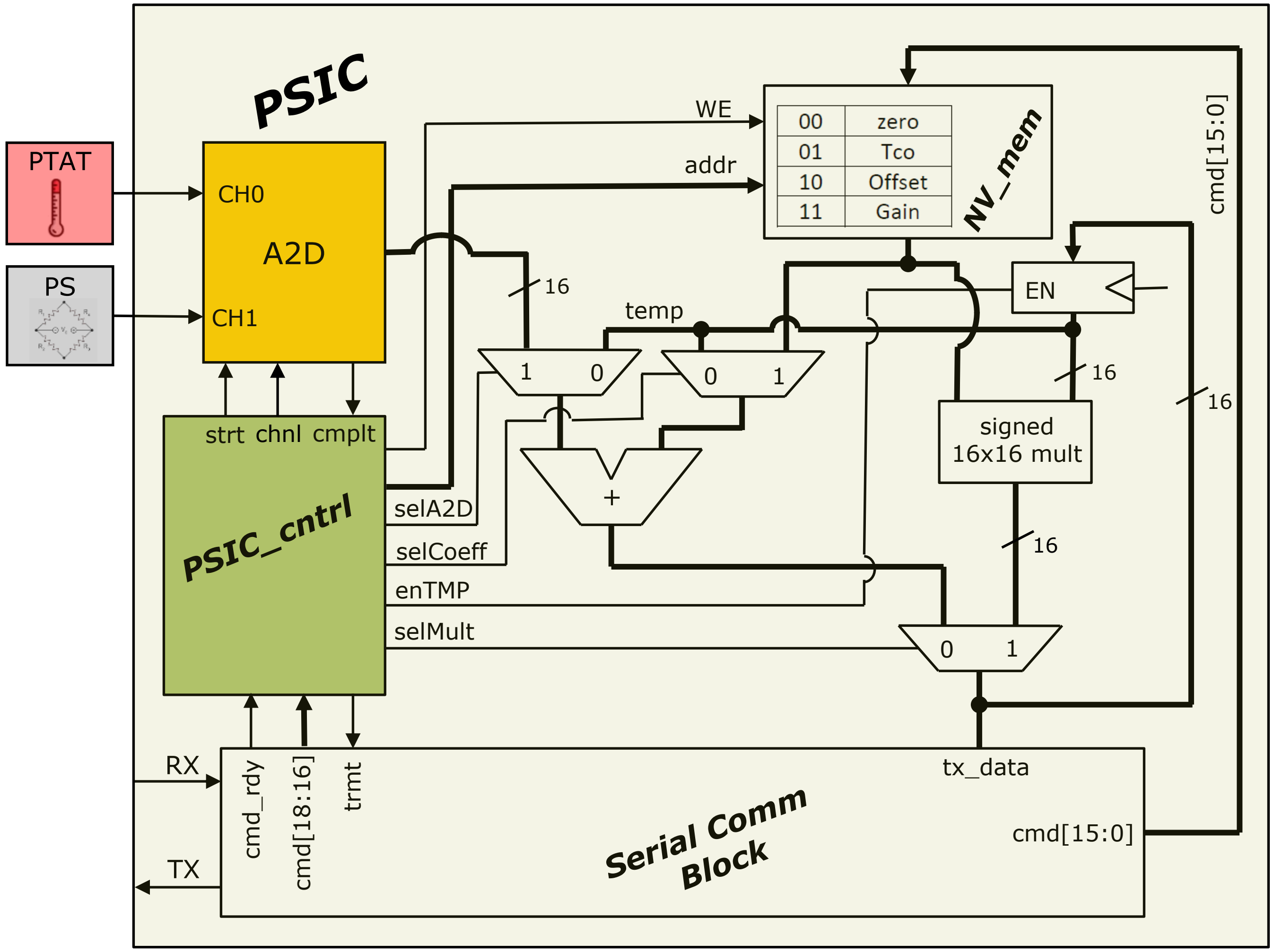
- Sensors are never perfect, and certainly not uniform.
  - Variation in offset of sensed parameter from unit to unit
  - Variation in sensitivity of sensor from unit to unit

- Sensors require calibration.
  - Offset compensation
  - Gain (sensitivity) compensation
  - Temperature compensation (very common for the offset of a sensor to shift with temperature)

- We will create a system to tackle these issues for a pressure sensor, but the general idea of sensor signal conditioning is common to many types of sensors.

Block Diagram of PSIC (**P**ressure **S**ensor **I**ntegrated **C**ircuit)

The pressure sensor (PS) will need to be calibrated for offset and gain. It will also need temperature compensation to adjust for offset variation.



A 16-bit **A**nalog to **D**igital (A2D) converter will provide readings of both the pressure sensor and a signal **p**roportional **t**o **a**bsolute **t**emperature (PTAT).

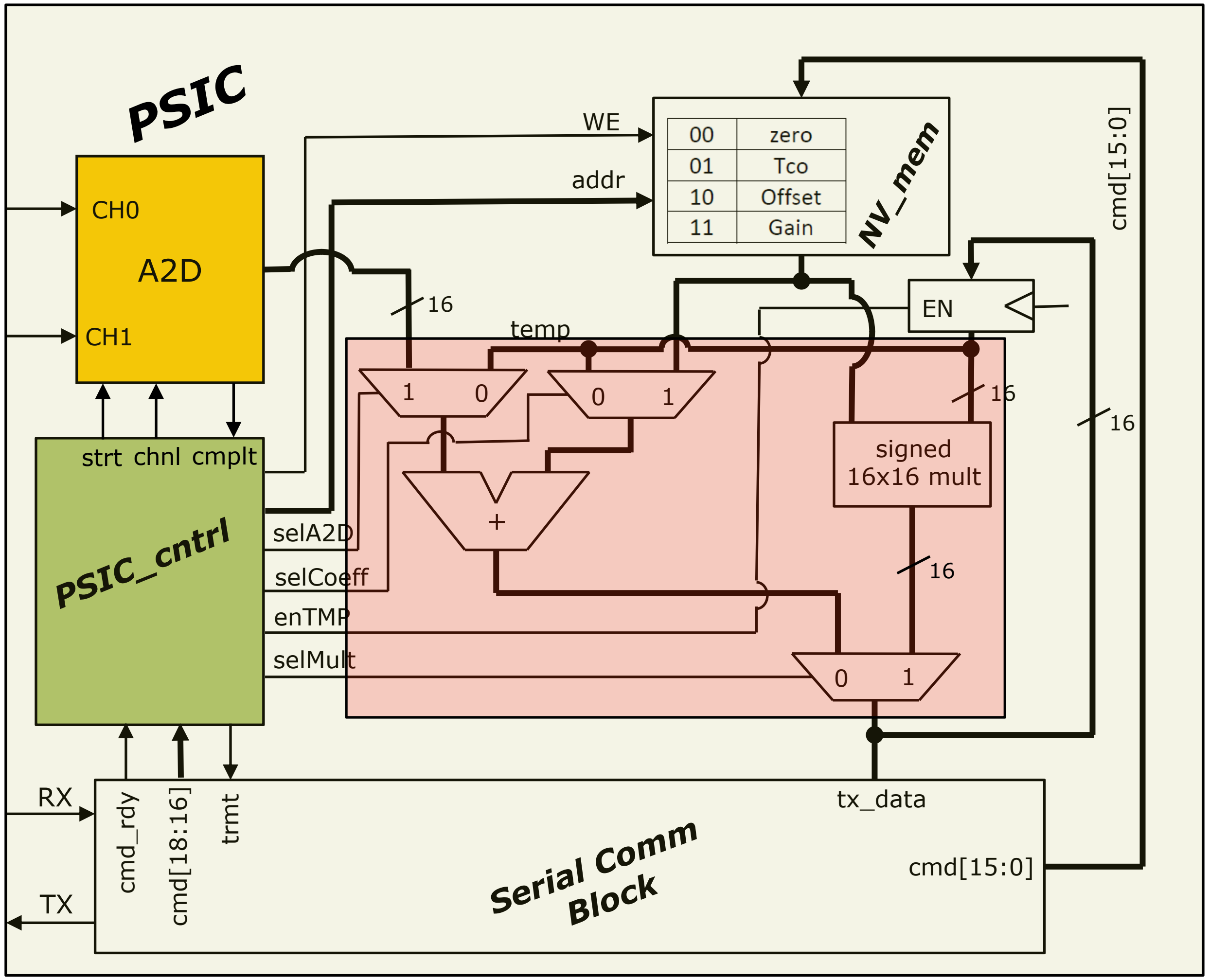
Calibration coefficients are stored in a **N**on-**V**olatile memory. The PSIC will correct the raw pressure sensor reading.

Pressure = **(**PS_{raw} + Offset + T_{CO}*PTAT**)***Gain

The PSIC receives commands/data to write the coefficients via a serial comm block. It also sends the corrected pressure via the serial comm block.

PSIC Datapath

The contents of the red block below is the datapath (computes the math). These are the blocks we will work on for AHW2. We will work on other parts of this system in later AHW's.

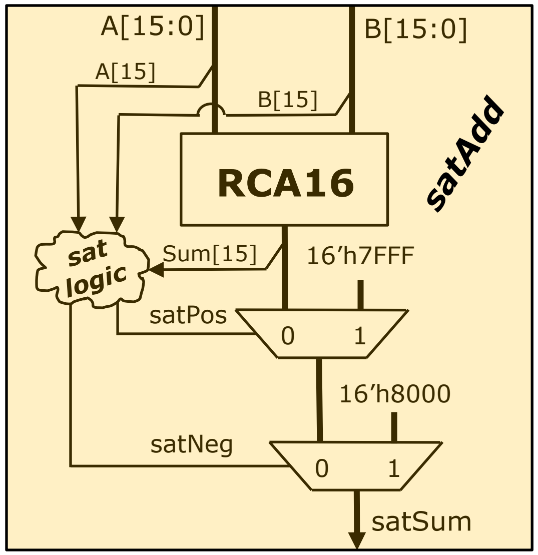


The datapath blocks are a little more complex than they seem. What happens when you have +overflow in addition? It wraps around and the result looks negative. Similar but opposite for underflow (result that would be more negative than we can represent would look +).

For a sensor signal conditioner you can't have this. You must have saturating arithmetic. So if the result would overflow+ you saturate to most positive value (0x7FFF). Similarly if it would underflow negative we saturate to the most negative number (0x8000).

The multiplier will also have some saturation cases we need to handle.

Saturating Adder



Your first task is to use your 16-bit adder (**RCA16**) from AHW1 to build a 16-bit saturating adder.

Using the MSB of each operand to be added and the MSB of the resulting Sum you should be able derive the logic for both positive and negative overflow (**satPos** & **satNeg**).

The 2nd to last page of this file has K-maps for you to fill out to determine the logic. Print that page, fill out the K-map by hand and take a picture to submit (**sat_logic_Kmap.jpg**).

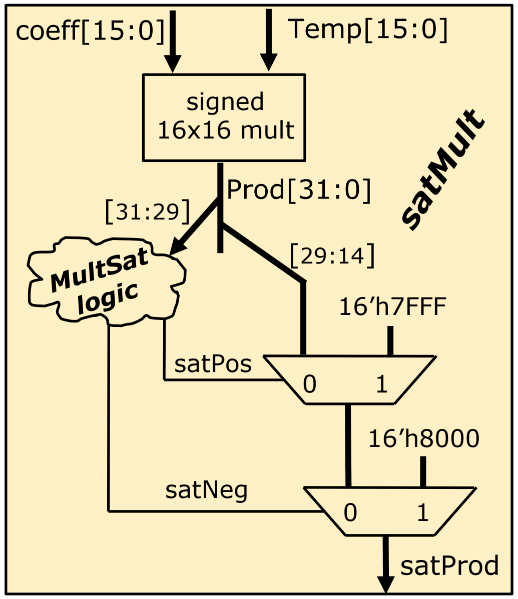
A shell is provided (**satAdd.sv**) for the saturating 16-bit adder. Complete the verilog:

1. Copy over your **RCA16.sv** and **FA.sv** files from AHW1
2. Complete the instantiation of **RCA16** to produce ***SUM,*** note we are not using ***Cin*** of **RCA16**
3. Using your solution from the K-maps instantiate verilog primitives to form ***satPos*** & ***satNeg***
4. Using dataflow verilog (assign statements) infer the two muxes shown that perform the saturation.

A testbench (**satAdd_tb.sv**) is provided. When you have completed **satAdd.sv** simulate and debug it using the provided testbench. When it is passing capture the image of the waveforms of the DUT for the full length of the simulation (**satAdd_waves.jpg**). Also capture the "Yahoo Test Passed" message in the transcript window (**satAdd_yahoo.jpg**).

Submit the following files: **sat_logic_Kmap.jpg**, **satAdd.sv**, **satAdd_waves.jpg, and satAdd_yahoo.jpg**

Saturating Multiplier



Floating point arithmetic units are really expensive (a lot of transistors required). For something like scaling the offset corrected pressure signal by a gain term we will use fixed point math.

The product of two signed 16-bit numbers is a 32-bit number. We only want a 16-bit result. We have a fixed divide by 2^{14} baked into the math by using bits [29:14] of the product as our 16-bit result.

Consider the sensor requires a gain factor of 1.125. Since we throw away the lower 14-bits of the result, we are effectively dividing by 2^{14} so our gain coefficient would be $1.125*2^{14}$ = 18432 = 16'h4800.

However, we still need to examine the upper 3-bits of the product [31:29] to determine if we should saturate the result positive, negative, or if we can just use bits [29:14] as the result.

K-maps to determine the MultSat_logic are provided on the last page. Print, fillout and take a picture (**MultSat_logic_Kmap.jpg**).

A shell is provided (**satMult.sv**) for the saturating Multiplier. Complete the verilog:

1. The signed 16x16 multiplier producing ***Prod[31:0]*** is already inferred for you.
2. Using your solution from the K-maps instantiate verilog primitives to form ***satPos*** & ***satNeg***
3. Using dataflow verilog (assign statements) infer the two muxes shown that perform the saturation.

A testbench (**satMult_tb.sv**) is provided. Simulate and debug. When it is passing capture the image of the waveforms of the DUT for the full length of the simulation (**satMult_waves.jpg**). Also capture the “Yahoo Test Passed” message in the transcript window (**satMult_yahoo.jpg**).

Saturating Multiplier

Submit the following files: **MultSat_logic_Kmap.jpg**, **satMult.sv**, **satMult_waves.jpg,** and **satMult_yahoo.jpg**

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

Sum[15] \ {A[15],B[15]}	00	01	11	10
0				
1				

satNeg

Sum[15] \ {A[15],B[15]}	00	01	11	10
0				
1				

satPos

satNeg = ______________________________

satPos = ______________________________

Another set below in case you messed up

Sum[15] \ {A[15],B[15]}	00	01	11	10
0				
1				

satNeg

Sum[15] \ {A[15],B[15]}	00	01	11	10
0				
1				

satPos

satNeg = ______________________________

satPos = ______________________________

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

Prod[31] \ Prod[30:29]	00	01	11	10
0				
1				

satNeg

Prod[31] \ Prod[30:29]	00	01	11	10
0				
1				

satPos

satNeg = ________________________________ satPos = ________________________________

Another set below in case you messed up

Prod[31] \ Prod[30:29]	00	01	11	10
0				
1				

satNeg

Prod[31] \ Prod[30:29]	00	01	11	10
0				
1				

satPos

satNeg = ________________________________ satPos = ________________________________