ECE 352 AHW4

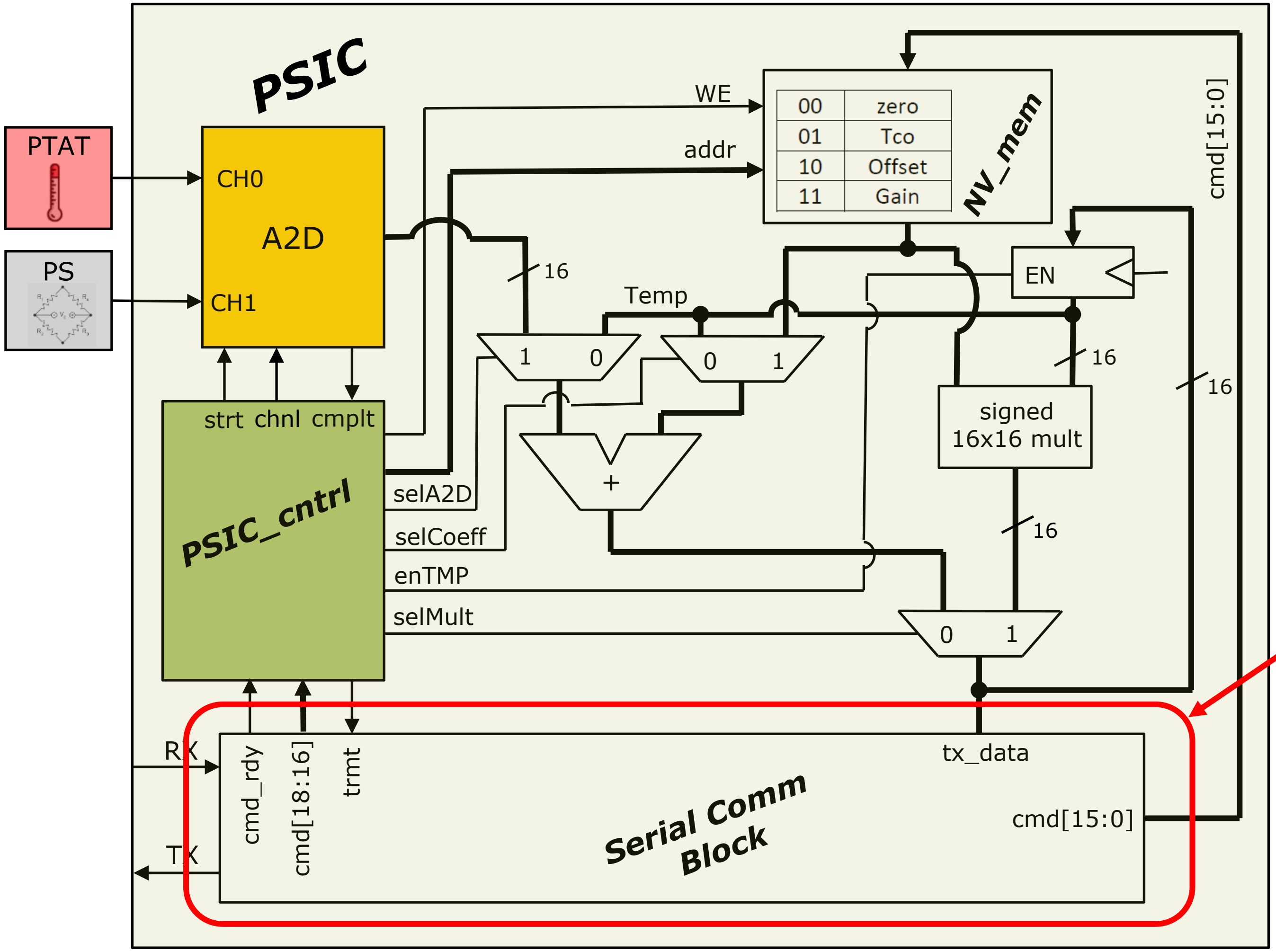
You may work with your assigned partner, or alone, as arranged by your "lab" section. If you work with a partner, we expect you to work together on the complete homework rather than sub-divide the tasks.

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

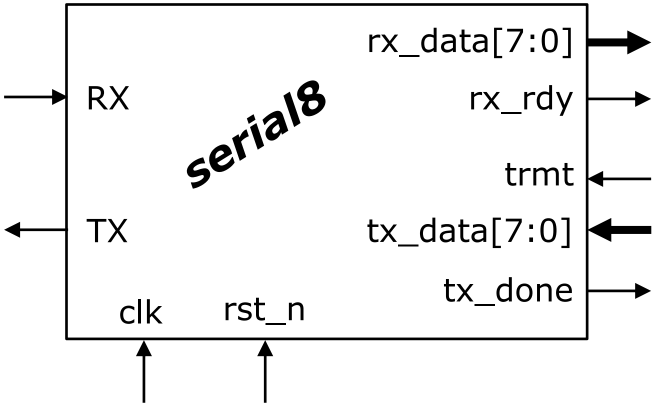
Block Diagram of PSIC (**This AHW focusses on serial_comm**)

In this applied HW we will take care of designing the state machines that enable serial_comm to receive 24-bit cmds and sent 16-bit data packets.



We will be working on some SM's inside the **serial_comm** block.

serial8 (one-byte serial transceiver)



serial8 is a provided block. It is a single byte serial transmitter/receiver (transceiver). The underlying serial protocol is not important for you to understand. You just need to understand the functionality at a higher level so you can interface to it to make the ***serial_comm*** block.

serial8 receives a stream of bits serially over the **RX** line and packages these bits into a byte (**rx_data**[7:0]). When it has a byte ready it assertes **rx_rdy** for a single clock period.

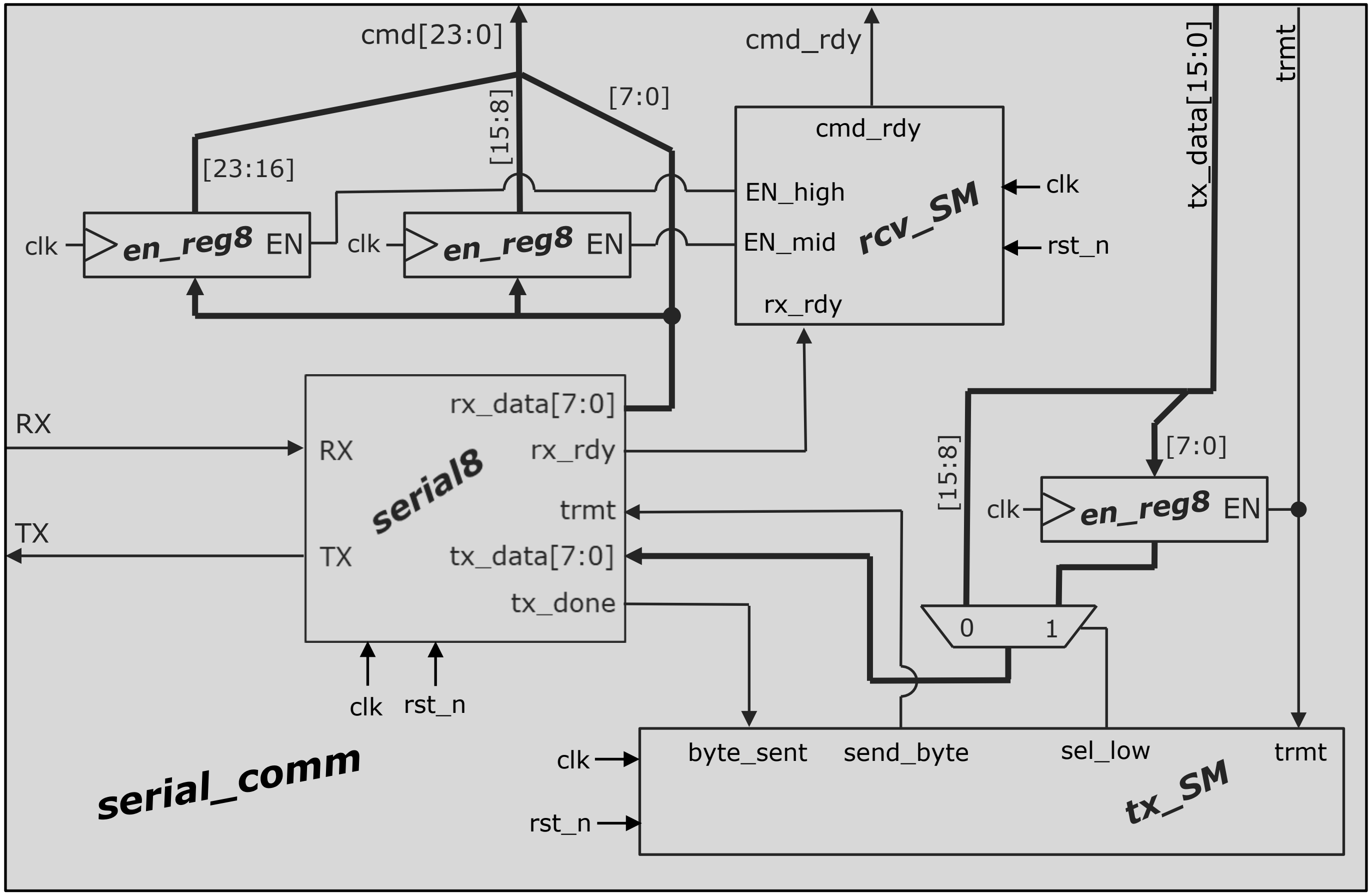
Signal:	Dir:	Description:
clk,rst_n	in	Clock and asynch active low reset.
RX	in	Serial data input line (1 bit stream coming in serially)
rx_data[7:0]	out	Byte that was received serially.
rx_rdy	in	A 1 clock cycle wide high pulse indicates **rx_data**[7:0] is valid and can be "consumed".
tx_data[7:0]	in	Byte to transmit serially
trmt	in	Pulse this signal high for 1 clock to initiate a transmission.
tx_done	Out	Indicates ***serial8*** is done transmitting the byte.
TX	Out	Serial output line (serial transmission of **tx_data**[7:0])

Transmission is the opposite process. The byte to be transmitted is applied to **tx_data**[7:0]. The **trmt** signal is then asserted for a single clock cycle. The byte will be transmitted a bit at a time over the **TX** line. When the entire byte is done being transmitted ***serial8*** will let you know by asserting **tx_done** for a single clock period.

Our job in this AWH is to package this byte based transceiver with some supporting datapath and make a unit (***serial_comm***) that is capable of receiving 24-bit commands (3-bytes) and sending 16-bit data (2-bytes).

serial_comm (24-bit receive, 16-bit transmit)

cmd[23:0] will be formed by 3-bytes received with the MSByte first. When the first byte is ready (**rx_rdy**) the ***rcv_SM*** will enable the highest (**EN_high**) holding register that will hold bits [23:16] of **cmd**. When the middle byte of **cmd** is received ***rcv_SM*** will enable the holding register (**EN_mid**) that holds bits [15:8] of **cmd**. When the lowest byte of **cmd** is received it does not need to be buffered in a holding register because it exists and is held as **rx_data** within ***serial8***.



The high byte of **tx_data**[15:0] is transmitted first. When **trmt** is asserted the low byte of **tx_data** is captured (saved) so it can be transmitted later when the high byte has completed transmitting (**tx_done**). When the ***tx_SM*** selects and transmits the low byte it can assume its job is done and return to its "IDLE" state (it does not have to look for a 2nd pulse on **tx_done**.

tx_SM (statemachine in charge of transmission)

Study the context and role (purpose) of the ***tx_SM*** in the preceding block diagram of ***serial_comm***

Signal:	Dir:	Description:
clk,rst_n	in	Clock and asynch active low reset
trmt	in	Signal initiating a transmission of two bytes. High byte goes out first
byte_sent	in	From **tx_done** of ***serial8***. Indicates a byte has been sent.
sel_low	out	Selects the low byte of **tx_data**[15:0] to be transmitted next
send_byte	out	To **trmt** of ***serial8***. Initiates transmission of a byte.

Your first task is to draw a bubble diagram of this SM using proper state diagram notation.

This should be done as a mealy machine *(as should every state machine you ever make in your life)*

This machine can be accomplished in 2-states.

Draw a state diagram for ***tx_SM*** and take a clear picture (**tx_SM_bubble.jpg**).

Submit **tx_SM_bubble.jpg** to the dropbox for AHW4

tx_SM (statemachine in charge of transmission)

- Copy over **state3_reg.sv**, **state2_reg.sv**, **d_en_ff.sv**, **en_reg8.sv** from your AHW3 folder. *(Did you make a **state2_reg.sv** ? It was a non-graded portion of that AHW).*
- **tx_SM.sv** is a shell that is provided. Using your bubble diagram complete the code for ***tx_SM***.
- A testbench (**tx_SM_tb.sv**) is provided. Build a modelSim project and test your ***tx_SM***. When it is bug free capture the waveforms of the DUT for the length of the simulation (**tx_SM_waves.jpg**). Also capture the “YAHOO test passed!” message from the transcript window (**tx_SM_yahoo.jpg**).
- Submit **tx_SM.sv**, **tx_SM_waves.jpg**, and **tx_SM_yahoo.jpg** to the dropbox for AHW4

rcv_SM (statemachine that handles reception)

Study the context and role (purpose) of the ***rcv_SM*** in the preceding block diagram of ***serial_comm***

Signal:	Dir:	Description:
clk,rst_n	in	Clock and asynch active low reset
rx_rdy	in	Indicates that a new byte has been received from ***serial8***
EN_high	out	Used to capture high byte into holding register
EN_mid	out	Used to capture middle byte into holding register
cmd_rdy	out	Asserted when 3rd byte of **cmd** has been received.

Draw a bubble diagram of this SM using proper state diagram notation.

This should be done as a mealy machine *(as should every state machine you ever make in your life)*

This machine can be accomplished in 3-states.

Draw a state diagram for ***rcv_SM*** and take a clear picture (**rcv_SM_bubble.jpg**).

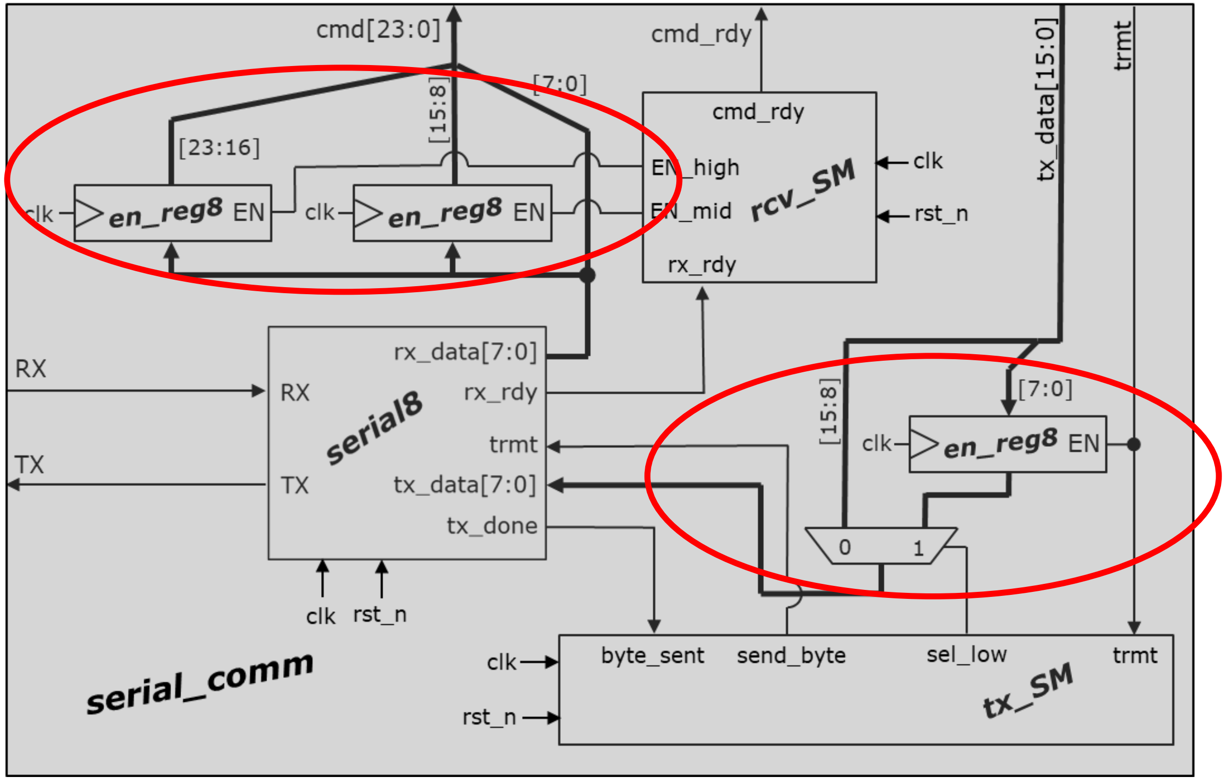
Submit **rcv_SM_bubble.jpg** to the dropbox for AHW4

rcv_SM (statemachine that handles reception)

- **rcv_SM.sv** is a shell that is provided. Using your bubble diagram complete the code for ***rcv_SM***.
- A testbench (**rcv_SM_tb.sv**) is provided. Build a modelSim project and test your ***rcv_SM***. When it is bug free capture the waveforms of the DUT for the length of the simulation (**rcv_SM_waves.jpg**). Also capture the "YAHOO test passed!" message from the transcript window (**rcv_SM_yahoo.jpg**).
- Submit **rcv_SM.sv**, **rcv_SM_waves.jpg**, and **rcv_SM_yahoo.jpg** to the dropbox for AHW4

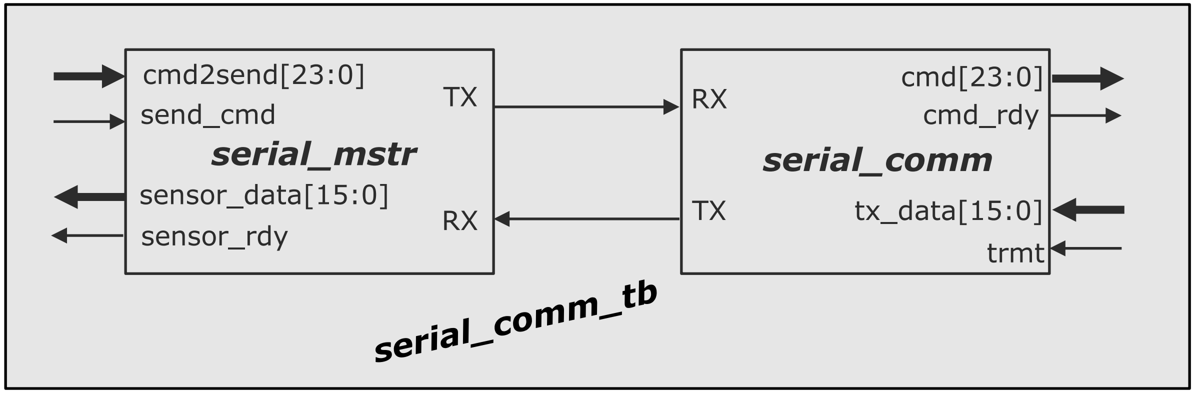
serial_comm (completing the datapath)

- **serial_comm.sv** is provided and instantiates ***serial8***, ***tx_SM***, and ***rcv_SM***. But the datapath elements are for you to fill in. You will be using your ***en_reg8*** so copy that over from AHW3.
- Fill in the missing code to complete the datapath as shown circled in red below.



- See the next page regarding testing your completed **serial_comm.sv.**

serial_comm (testing)



Typically the easiest way to test a serial block is to link it up with another serial block that uses the same protocol.

serial_mstr.sv is provided and is the "conjugate" of ***serial_comm***. It sends 24-bit data and received 16-bit data.

A toplevel testbench for serial_comm (**serial_comm_tb.sv**) is provided. It transmits some 24-bit data and checks that your DUT (**serial_comm**) receives the same command as was transmitted.

It similarly will initiate your DUT (***serial_comm***) to transmit 16-bit data and check that the received (**sensor_data**) is same as was transmitted.

Build a ModelSim project using ***serial_mstr***, ***serial_comm_tb***, and ***serial_comm***. Debug until the self-checking testbench shows no errors. Capture the waveforms of the DUT for the duration of the simulation (**serial_comm_waves.jpg**). Also capture the "YAHOO test passed!" message from the transcript window (**serial_comm_yahoo.jpg**).

Submit **serial_comm.sv**, **serial_comm_waves.jpg**, and **serial_comm_yahoo.jpg** to the dropbox for AHW4.