ECE 551
HW3

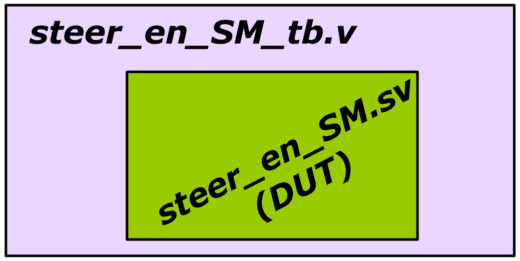
- Due Wed Oct 17th @ class (9:55AM)
- Work Individually
- Remember What You Learned From the Cummings SNUG paper
- Use descriptive signal names and comment your code

HW3 Problem 1 (**20pts)** SM Design

- In HW1 you made the bubble diagram for the SM that determined when to enable steering. The SM has the interface shown here →
- You will now be implementing that SM in Verilog and testing it via a provided testbench.
- Download **steer_en_SM_shell.sv** and **steer_en_SM_tb.v**
- Flush out **steer_en_SM_shell.sv** to implement the statemachine
- Rename the file **steer_en_SM.sv**
- Using the provided testbench (**steer_en_SM_tb.v**) test it in ModelSim and debug any errors the testbench points out.
- Submit both **steer_en_SM.sv** and proof that you're the testbench ran to completion on your implementation.
- The next two slides repeat information to clarify the function of this SM.

Signal:	**Direction:**
clk, rst_n	in
tmr_full	in
sum_gt_min	in
diff_gt_1_4	in
diff_gt_15_16	in
clr_tmr	out
en_steer	out
rider_off	out

OK..the **rider_off** signal did not exist in HW1, but it is simple to add.

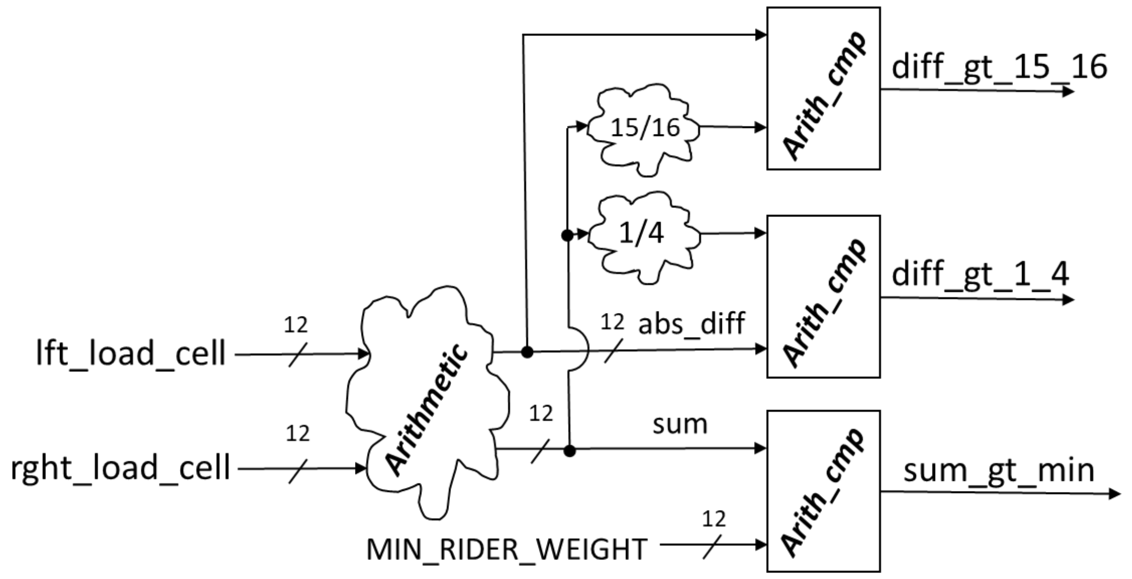


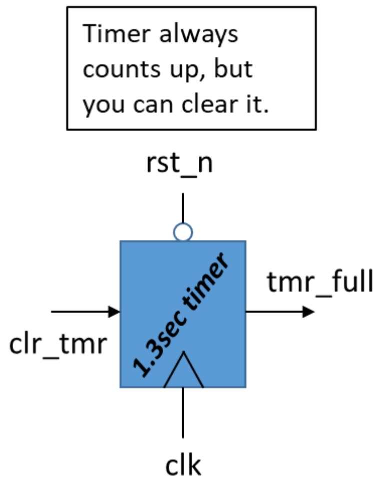
You flush out and rename **steer_en_SM_shell.sv** and test and debug it using the provided self checking test bench

Not started as an in class exercise. Do this one on your own.

HW3 Problem 1 **(Clarifying Materials)**

Assume the hardware elements shown here are available to you. You are figuring out the SM bubble diagram





First we look to see that the sum of the two load cells exceeds the minimum rider weight. If that condition is met (**sum_gt_min**) we next look at **diff_gt_1_4** to see that the absolute value of the difference between the left and right load cells does not exceed 1/4 of the total weight for 1.3 seconds. If the difference has been consistently below 1/4 of the sum for 1.3 seconds then steering is enabled (hold **en_steer** high). We can leave the steering enabled state one of two ways. The rider suddenly gets knocked off the device (**sum_gt_min** goes low), or the difference between the load cells exceeds 15/16 of the sum (rider is stepping off). Under both conditions we exit the steering enabled state. If the user is stepping off we return to the state for waiting for balance for 1.3 sec. If the user is knocked off we return to the initial state. If the user is stepping off and we are in the wait state checking for balance we have to look for them to completely step off (**sum_gt_min falls**) or to regain balance. **Draw the bubble diagram** for this SM. Take a picture with your phone and submit it. It must be legible!

Steering Enable (**Clarifying Materials continued**)

- The output **en_steer** should be asserted whenever the state machine determines we are in normal mode of operation with steering enabled. This signal will go to **balance_cntrl** block and the piezo buzzer control.

- The signal **rider_off** should be asserted whenever the rider weight does not exceed the minimum threshold (i.e. when (lft_ld + rght_ld)<MIN_RIDER_WEIGHT).
  - There is a possibility the rider falls off the Segway all at once. Meaning there are two ways the state machine can exit normal mode. If the load cell difference exceeds 15/16 of the sum, or if all of a sudden the load cell sum is less than MIN_RIDER_WEIGHT (rider_off). Your normal mode case needs to check for both.
  - If the difference exceeded 15/16 you return to a state where you are waiting for the rider to achieve balance for 1.3sec or to completely step off. If they do step off you assert **rider_off.**
  - If they fell off all at once then you transition from normal mode to the initial state (waiting for sum to exceed MIN_RIDER_WEIGHT)

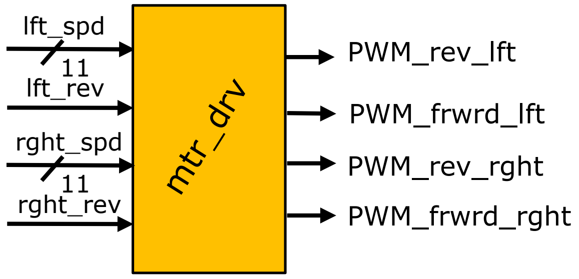
- **rider_off** goes to the balance controller to clear the integral term of the PID loop. (Imagine you were riding along on your Segway and the integral term was at a high value. Then you got "clotheslined" off your Seqway. You would want your device to stop, not keep going and run away from you).

HW3 Problem 2 (**25pts)** PWM11 & mtr_drv

In exercises 8,9, and 10 you completed an 11-bit PWM module. You will now be using that module to complete the **mtr_drv** block.

mtr_drv has the following interface:

Signal:	Dir:	Description:
clk, rst_n	in	50MHz clock, and active low asynch reset
lft_spd[10:0]	in	Left motor duty cycle
lft_rev	in	If high left motor should be driven in reverse
PWM_rev_lft PWM_frwrd_lft	out	11-bit PWM signal (2048 divisions/period) go to H-Bridge controller chip
rght_spd[10:0]	in	Left motor duty cycle
rght_rev	in	If high left motor should be driven in reverse
PWM_rev_rght PWM_frwrd_rght	out	11-bit PWM signal (2048 divisions/period) go to H-Bridge controller chip

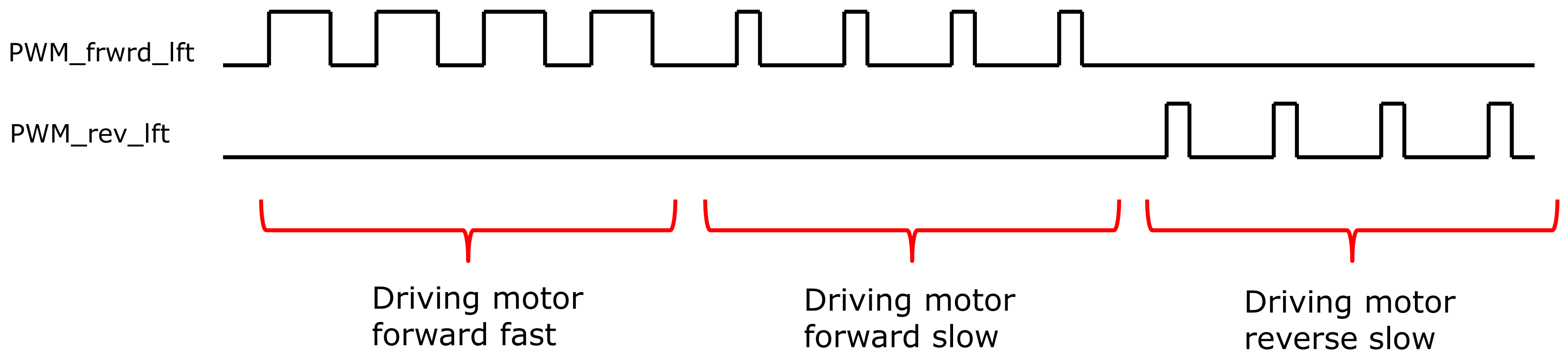


- The digital core will produce a motor drive magnitude and direction for both the left and right motors.
- The magnitude of drive will be an 11-bit unsigned number.

Exercises 8,9, and 10 will have us doing much of this problem.

HW3 Problem 2 **(25pts)** PWM11 & mtr_drv

- The DC motors are driven through an H-bridge. An H-bridge uses a PWM signal and can drive current in either direction through a load proportional to the duty cycle of the incoming PWM signal. The example waveforms below illustrate how the H-bridge driver chip we are using works:


PWM_frwrd_lft
PWM_rev_lft
Driving motor forward fast
Driving motor forward slow
Driving motor reverse slow

- You are just using two copies of that (one for left and one for right) along with some simple ANDing logic to make this unit.

Submit the following files:

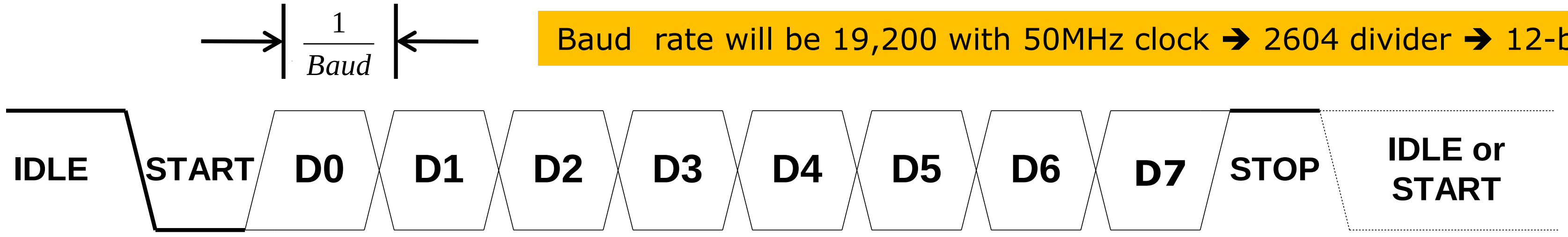
PWM11.sv PWM11_tb.sv

mtr_drv.sv mtr_drv_tb.sv ➔ test bench drives both directions for each left/right

mtr_drv_tb.jpg ➔ proof you ran your motor drive testbench

What is UART (RS-232)

- RS-232 signal phases
  - Idle
  - Start bit
  - Data (8-data for our project)
  - Parity (no parity for our project)
  - Stop bit – channel returns to idle condition
  - Idle or Start next frame

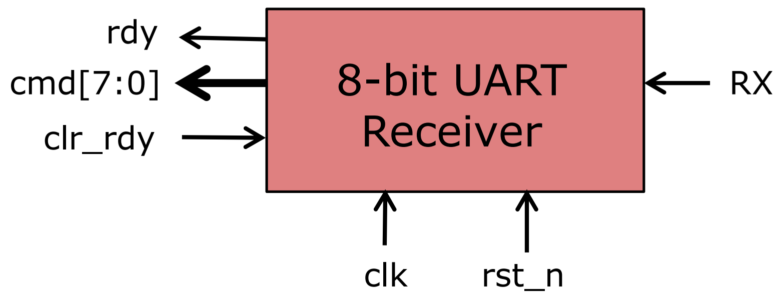


Baud rate will be 19,200 with 50MHz clock ➔ 2604 divider ➔ 12-bit

- Receiver monitors for falling edge of Start bit. Counts off 1.5 bit times and starts shifting (right shifting since LSB is first) data into a register.

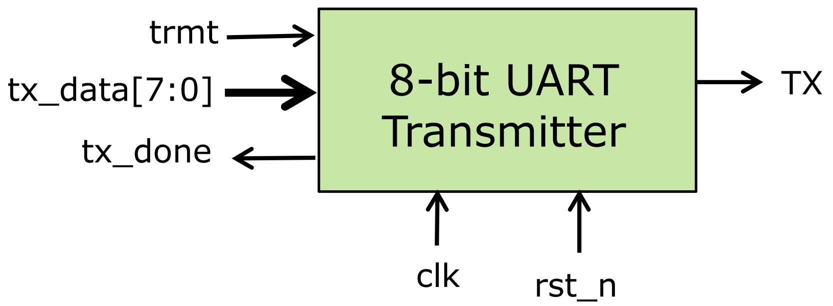
- Transmitter sits idle till told to transmit. Then will shift out a 9-bit (start bit appended) register at the baud rate interval.

UART Receiver/Transmitter



A host computer will send commands to the Logic Analyzer via a UART serial peripheral

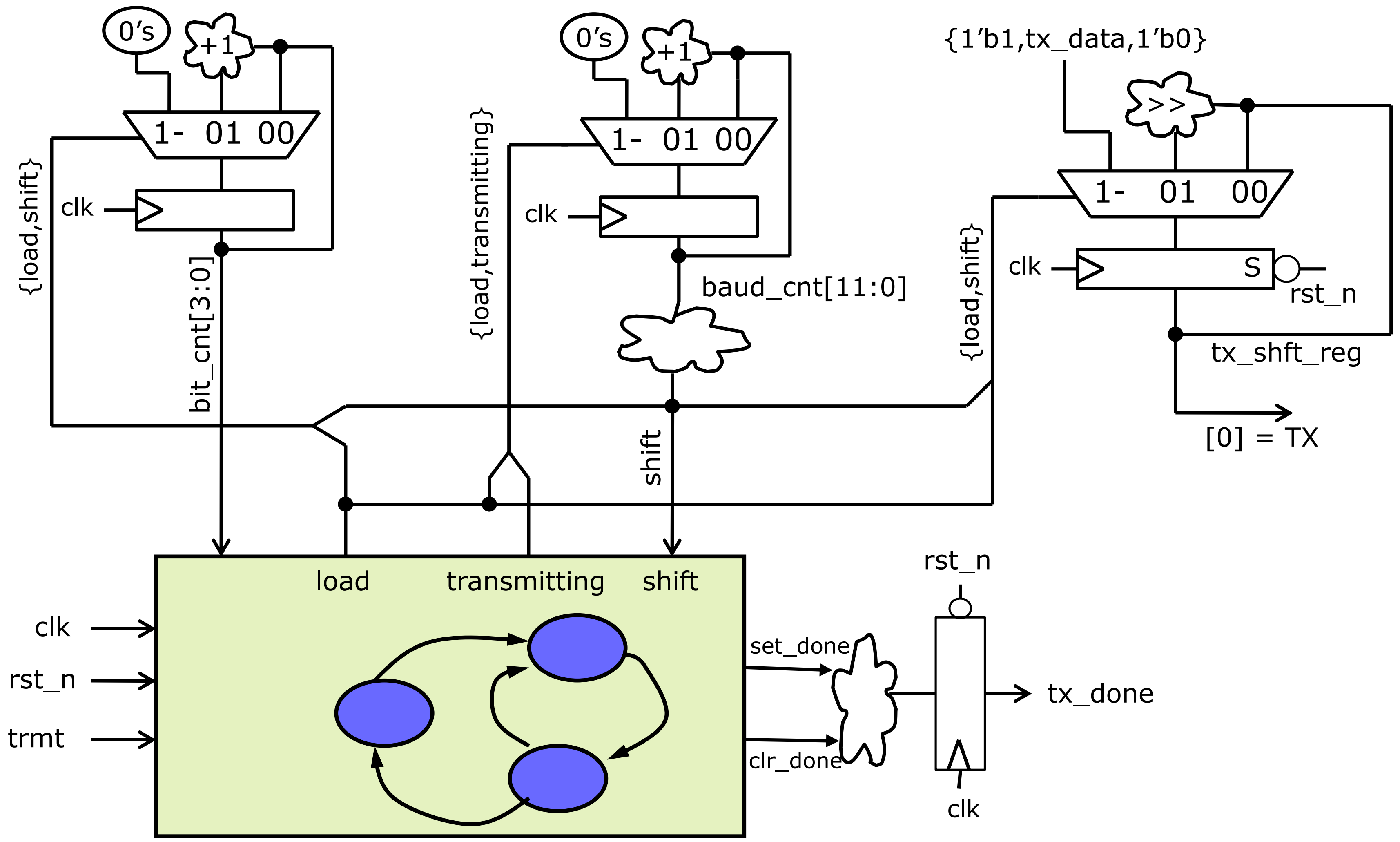
Signal:	Dir:	Description
clk,rst_n	in	100MHz system clock & active low reset
RX	in	Serial data carrying command from host computer
rdy	out	Asserted when a byte has been received. Falls when new start bit comes, or when ***clr_rdy*** knocks it down.
cmd[7:0]	out	Byte received (serves as command to LA)
clr_rdy	in	Asserted to knock down the **rdy** signal.



The follower sends responses back to the host computer. These reponses are sent via a UART serial peripheral.

Signal:	Dir:	Description
clk,rst_n	in	100MHz system clock & active low reset
TX	out	Serial data output back to host
trmt	in	Asserted for 1 clock to initiate transmission
tx_data[7:0]	in	Byte to transmit (response from LA)
tx_done	out	Asserted when byte is done transmitting. Stays high till next byte transmitted.

Possible Topology of UART_tx



HW3 Problem 3 (**20pts)** UART Transmitter

Implement a the UART Transmitter (**UART_tx.sv**).

Make a simple test bench for it. This is one instance in which I would not spend too much time on the test bench. You can just instantiate your transmitter and send a few bytes. Verify the correct functionality (including baud rate) by staring at the green waveforms. You will make a more comprehensive test bench in the next problem.

Submit **UART_tx.sv** to the dropbox for HW3.

We will start this in Exercise 11…you will have to complete on your own.

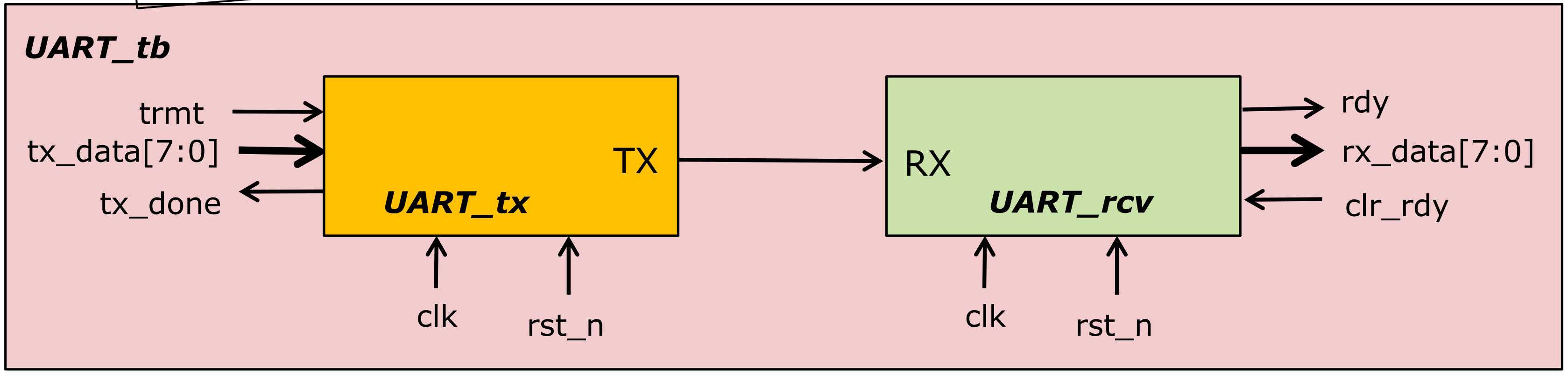
HW3 Problem 4 (**25pts + 10pts**) UART Receiver

Implement a the UART Receiver (**UART_rcv.sv**).

Since you have a transmitter too, it is now easy to make a self checking test bench. Architect the test bench as shown. Does the 8-bit value you transmit match the value you receive when the transmission completes?

Submit **UART_rcv.sv** and your test bench to the dropbox for HW3.

We will start this in Exercise 12…you will have to complete on your own.



UART_tb
trmt
tx_data[7:0]
tx_done
TX
UART_tx
clk
rst_n
RX
UART_rcv
rdy
rx_data[7:0]
clr_rdy
clk
rst_n