ECE 551
HW3

- Due Mon Oct 13th @ 11:55PM
- Work Individually
- Remember What You Learned From the Cummings SNUG paper
- Use descriptive signal names and comment your code

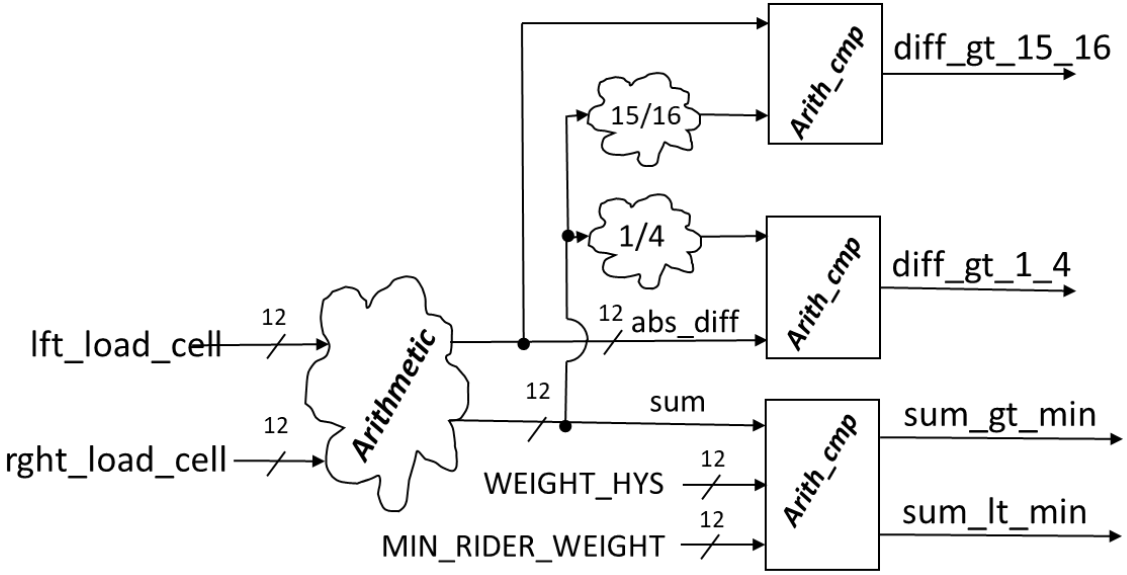
HW3 Problem 1 (**20pts)** SM Design

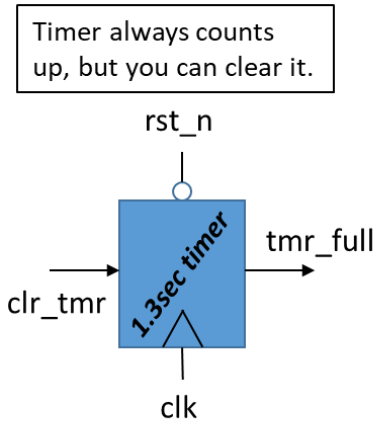
- Stepping onto a Segway is a bit disconcerting and we want the rider to be comfortably on and balanced before enabling steering.
- There are two load sensors in the floor of the Segway that measure force each foot is exerting on the floor.
- We will use measurements from these load sensors to determine when a rider is on, and reasonably balanced side to side.
- Your first task in this problem will be to draw a bubble diagram for the steering enable statemachine (***steer_en_SM***).
- The next two slides describe the desired behavior of the SM.

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 exit the initial state and enter a state looking for steady *(>1.3sec)* side to side balance. If **diff_gt_1_4** the rider is not balanced *(side to side)* and we clear the 1.3sec timer ***(clr_tmr)*** and remain in this state. If **diff_gt_1_4** has been low for 1.3 seconds then we know the rider is balanced and we enter the steering enabled state. We can leave the steering enabled state one of two ways. The rider suddenly gets knocked off the device ***(sum_lt_min** goes high),* 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 side to side balance for 1.3 sec. In this state there is a transition back to the initial state if **sum_lt_min** rises *(the rider completely stepped off).* If the user is knocked off we immediately return to the initial state from the steering enabled state. **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 behavior of the output signal **en_steer** was covered in the previous slide and should only be asserted when the rider is on the segway and has been balanced *(side to side)* for at least 1.3sec.

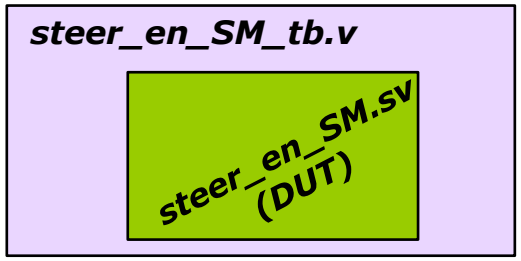
- The signal **rider_off** should be asserted whenever the rider weight is not known to exceed the minimum threshold *(i.e. when in the initial state or when in another state and **sum_lt_min**)*
  - There is a possibility the rider falls off the Segway all at once. Meaning there are two ways the state 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 (**sum_lt_min**). Your steering enabled state needs to check for both.
  - If the difference exceeds 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 return to the initial state waiting for **sum_gt_min.**
  - If they fell off all at once *(sum_lt_min asserted)* then you transition from steering enabled to the initial state. The check for "fell off" should have priority over the check for stepping off.

- **rider_off** goes to the authorization block that asserts **pwr_on**. It is used to prevent the power off of the segway while a rider is on. (Imagine you are riding your friends Segway and they took out their phone and commanded it to power off. Since it actively balances the rider, we cannot allow power off to occur unless we know the rider is off).

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

- The SM has the interface shown in this table 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 to **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** a picture of your bubble diagram, **steer_en_SM.sv** and proof that you're the testbench ran to completion on your implementation.

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



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

HW3 Problem 2 (**15pts)** PWM11

- Started as Exercise10 on Sept 29th
- See Exercise10 specification for details. Did you complete it well? Polish it up if not.
- **Submit**: **PWM11.sv**, **PWM11_tb.sv**, and **proof** (waveforms or other) that it ran correct in your testbench

HW3 Problem 3 (**20pts)** PID

- Started as Exercise12 on Oct 3rd
- See Exercise12 specification for details. Did you complete it well? Polish it up if not.
- **Submit**: **PID.sv**, and **proof** that is passed the provided self-checking testbench. I want to see **your name** in the image of the transcript window, so edit the provided testbench to include your name somewhere in the messages output to the transcript window.

HW3 Problem 4 (**15pts)** UART_TX

- Started as Exercise13 on Oct 6th
- See Exercise13 specification for details. Did you complete it well? Polish it up if not.
- **Submit**: **UART_TX.sv**

HW3 Problem 5 (**30pts)** UART_RX & Testing

- Started as Exercise14 on Oct 10th
- See Exercise14 specification for details. Did you complete it well? Polish it up if not.
- **Submit**: **UART_RX.sv, UART_tb.sv,** and **proof** this self-checking tb passed.