ECE 551
Homework #2
Due: Monday Feb 21st @ 11:55PM

Please Note: For this homework, you ***must***:

- Use informative module/signal/port names whenever at all reasonable to do so.
- Work individually

This problem will be started as an exercise on Weds Feb 9th

1) **(20pts)** Using Dataflow/RTL Verilog, implement a shifter capable of performing both arithmetic and logical right shifts on a 16-bit input (**src[15:0]**). The shift/rotate amount can be anywhere from 0 to 15-bits, and is specified by a 4-bit input (**amt[3:0]**). The result goes to a signal called **res[15:0]**. You are **not** allowed to use the >> or >>> operator. Write a testbench to verify its operation. This testbench does not need to be self checking, stimulus generation only (Please ensure reasonably comprehensive coverage with stimulus. i.e. cover multiple 16-bit quantities for both arithmetic and logical and for multiple shift amounts).

Signal Name:	Dir:	Description:
src[15:0]	in	Input vector to be left shifted
ars	in	If asserted then operation is arithmetic
amt[3:0]	in	Specifies the amount of the shift 0 to 15 bits
res[15:0]	out	Result of the shift/rotate

Submit to the dropbox:

a) The code used to describe the shifter (**shfter.sv**)
b) The code used for the testbench (**shifter_tb.sv**)
c) Proof (image of waveforms) that you bothered simulating

This problem will be started as an exercise on Fri Feb 11th

2) **(15 pts)** You have a 13-bit signed number (**incline[12:0]**) that tells you information about the incline your eBike is on. + means uphill, and – means downhill. This 13-bit number of course has a range of [-4096,4095] *(just ask a ECE252 student)*. However, for any incline beyond the range [-512,511] the eBike motor is going to be saturated at off or on full so the incline number could be saturated to a 10-bit signed number. You are to create a block that takes in a 13-bit signed number and outputs a 10-bit saturated signed number. It should be implemented in dataflow verilog. The file should be **incline_sat.v**

Signal:	Direction:	Description:
incline[12:0]	in	Incline from inertial senor (signed)
incline_sat[9:0]	out	10-bit saturated (signed) result

Create a testbench (**incline_sat_tb.sv**) test your DUT with several different input vectors. It should test cases of saturating positive, negative and not saturating. It should be **self-checking** *(compute what the right answer should be manually and use* ***if*** *statements and* ***$display*** *statements to form a self-check).* Submit both **incline_sat.sv** and **incline_sat_tb.sv** to the dropbox.

3) **(15 pts)**

You are on your own for this problem. No exercise covers it.

a. Below is the implementation of a D-latch.

```
module latch(d,clk,q);
  input d, clk;
  output reg q;

  always @(clk)
    if (clk)
      q <= d;

endmodule
```

Is this code correct? If so why does it correctly infer and model a latch? If not, what is wrong with it?

Submit to the dropbox a single file called **HW2_prob3.sv**

b. The comments should answer the questions about the latch posed above.

c. The file should contain the model of a D-FF with an active high synchronous **set** and an enable.

d. The file should contain the model of a D-FF with asynchronous active low reset and an active high enable.

e. The file should contain the model of a set/reset FF with active low synchronous reset. This means it has a synchronous set input (**set**) that sets it, a synchronous reset (**clr**) that clears it, and an active low aysnch reset (**rst_n**). Give priority to one of **set** or **clr** (your choice).

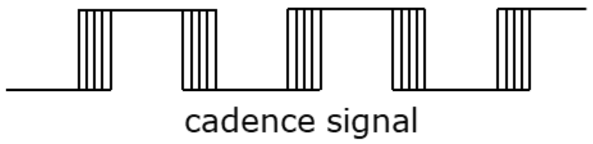
This problem will be started as an exercise on Fri Feb 11th & continued on the following Monday

4) **(30 pts)** This problem is covered in Exercise07 parts II & III. For those exercises you submitted whatever you had done at the end of the class period. It might not have been rock solid code yet. Polish up those design and submit **desiredDrive.sv** and **desiredDrive_tb.sv** and proof that you successfully ran **desiredDrive_tb**.

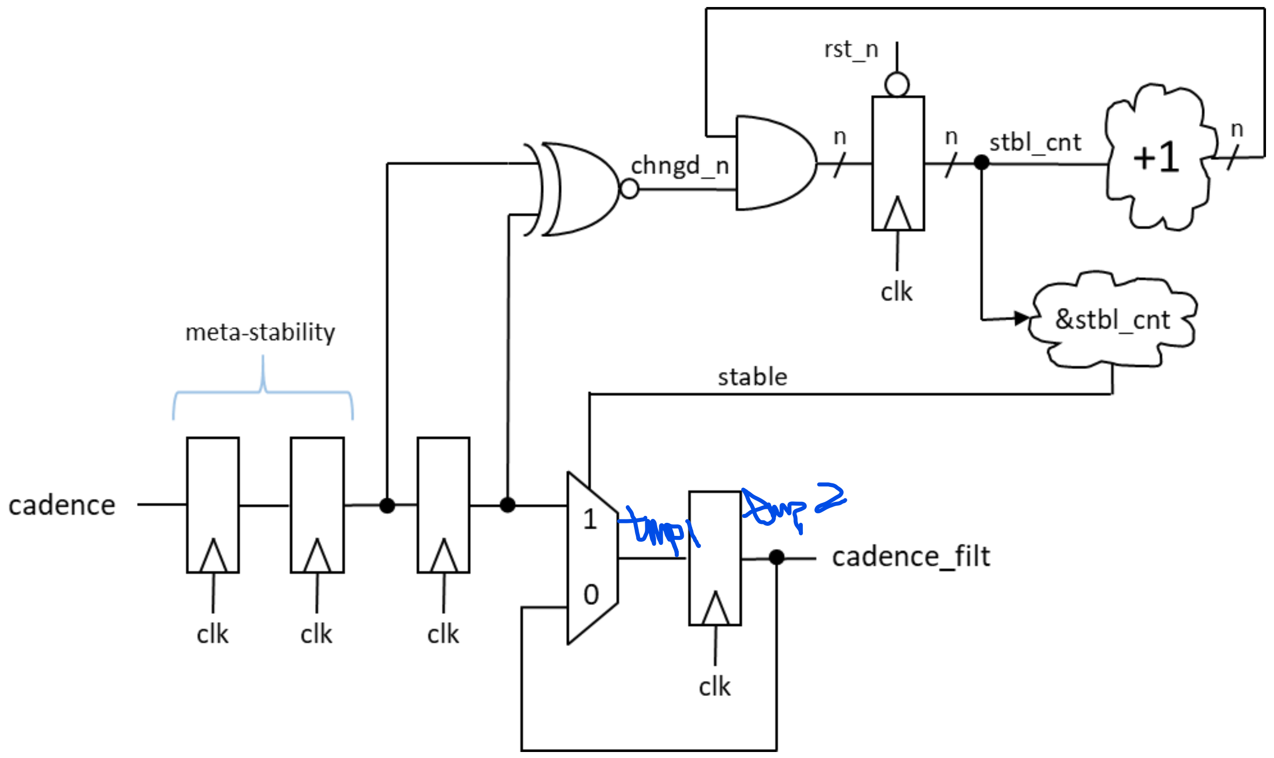
You are on your own for this problem. Don't start it until after you have watched Lecture04 videos.

5) (**20** pts) This problem will be a bit of a stretch of your current knowledge since it will be your first time inferring flops & counters. In this problem you are implementing a block needed for the project, so you want to do a good job. I suspect, however, later in the semester *(when you are more learned in the ways of Verilog)* you will look back at the code you wrote here and say: "yowzah that is garbage". Just give it your best "college try" for now.

The sensor that senses how fast the rider is pedaling is called a cadence sensor. It gives out 32 digital pulses per rotation of the cranks. The problem is the sensor is noisy as all heck. I think it is built with a rotating mechanical slip ring. All I know is it can give many false up and down pulses at the transition points, and that the output seems very low impedance and cannot be filtered via an analog low pass filter. We are going to have to build a digital filter to filter it.



Our filter is just going to look for greater than 1ms of stability in the signal before allowing the filtered version to be updated with the current signal. See the following diagram.



Of course, the incoming signal from the sensor is asynchronous to our clock domain so has to be double flopped to eliminate meta-stability. I am sure you would have thought of that, so sorry to mention it. An XNOR gate can look across the synchronized signal vs what the signal was 1 clock ago to see if it changed. If it did change our stability counter should be knocked down to zero. I will let you figure out how wide (**n**) the counter needs to be to measure at least 1ms of stability (our clock frequency will be 50MHz). If the stability counter gets up to a full count it means the signal has been stable for at least 1ms so the filtered version of the signal is allowed to take on the current version of the signal. Yes..this filter does introduce more than 1ms of delay to the signal and limits the frequency of what can come through. However, think of the application…does it matter? Even Greg LeMond never pedaled that fast. Code **cadence_filt.sv** with the following interface and submit it to the dropbox.

Signal:	**Dir:**	**Description:**
clk	In	50MHz clk
rst_n	In	Asynch active low reset
cadence	In	Raw input from cadence sensor
cadence_filt	Out	Filtered signal to be used elsewhere

A self checking testbench (**cadence_filt_tb.sv**) has been provided. Use it to check your design. **Submit** a capture of the transcript window showing it ran successfully.