ECE 551
Homework #2
Due: Mon Oct 2nd @ 11:55PM

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

- Use informative module/signal/port names

No exercise for this problem…do on your own

1) **(15pts)** Dataflow (RTL) warmup…a 16-bit left shifter/rotator. Using dataflow/RTL verilog implement a shifter/rotator (**lshift_rot.sv**) capable of performing left shifts or rotates on a 16-bit input (**src[15:0]**). The shift 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 (**lshift_rot_tb.sv**) to verify its operation. This testbench should be self-checking. (Please ensure reasonably comprehensive coverage with stimulus. i.e. cover multiple 16-bit quantities for both shifts and rotates and for multiple shift amounts).

See bottom of this document for hints on implementing a logarithmic shifter

Signal Name:	Dir:	Description:
src[15:0]	in	Input vector to be shifted/rotated
rot	in	If asserted rotate instead of shift
amt[3:0]	in	Specifies shift amount 0 to 15
res[15:0]	out	Result of shift

Submit:

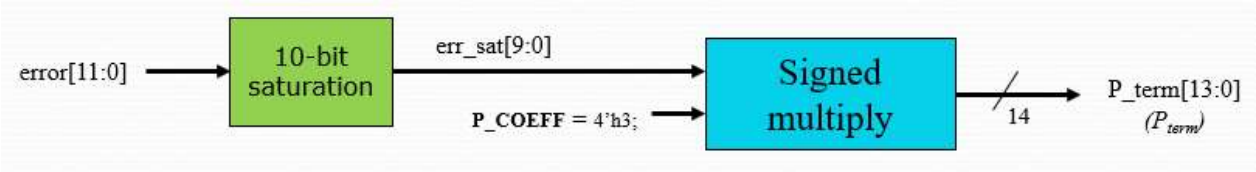
lshift_rot.sv → verilog of shifter
lshift__rot_tb.sv → testbench
lshift_waves.jpg → capture of waveforms from your testbench proving you ran it.

2) **(20pts)** See exercise06 for more detailed description.

Signal:	Dir:	Description:
error[11:0]	in	12-bit signed error term MEMs gyro.
P_term[13:0]	out	14-bit signed P component of PID controller

This problem started as in class exercise (Ex06) on Weds Sept 20th

The name of the file should be ***P_term.sv.*** The P_COEFF should be declared as a **localparam** and its value should be set to 4'h3. Keep in mind creating a signed multiply in verilog does not happen by default. To make a multiply signed both signals being multiplied have to be of type signed, and the result the product is assigned into has to be of type signed.



Write a test bench (***P_term_tb.v***) and instantiate the module. Apply several values to the inputs that would saturate both positive and negative and not saturate. The testbench should be self-checking.

Submit to the dropbox

a) P_term.sv
b) P_term_tb.sv
c) **P_term_waves.png** (or JPG) that shows you simulated.

3) **(15 pts)** See exercise07 for description:

This problem started as in class exercise (Ex07) on Fri Sept 22nd

Submit to the dropbox

a) IR_Math.sv
b) Image of transcript window showing it passed provided testbench

4) **(20 pts)**

No exercise for this problem…do on your own

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**

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

b. The file should contain the model of a D-FF with an active low synchronous reset.

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

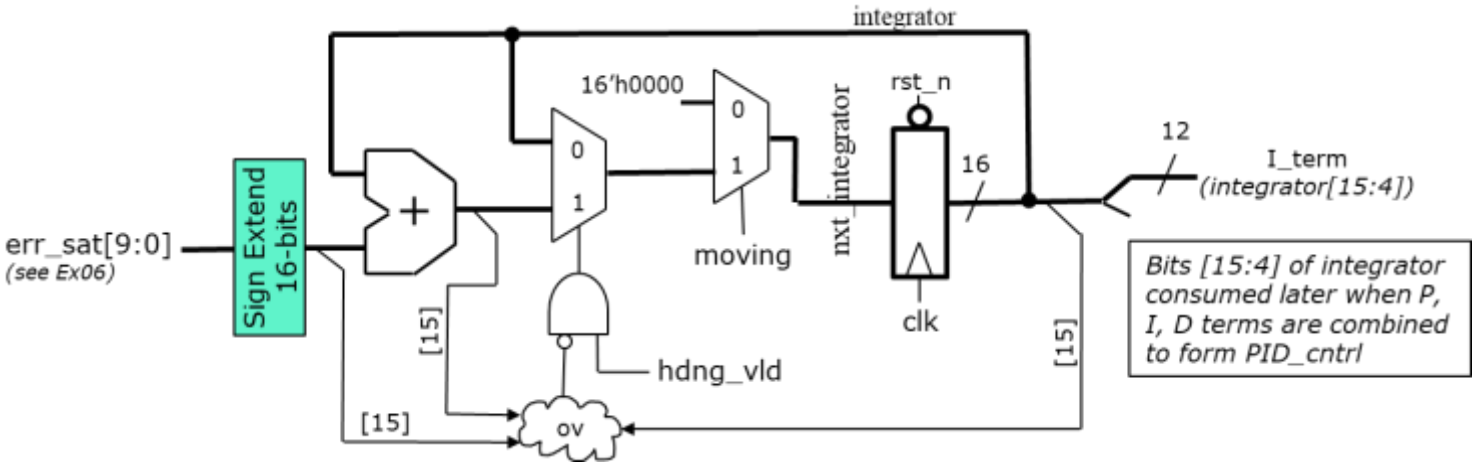
d. The file should contain the model of a SR FF with **active low asynchronous** reset. SR meaning it has a **S** input that will set the flop, and a **R** input that will reset the flop, and it maintains state if neither **S** or **R** are high. Both **S** & **R** are synchronous inputs, and this flop still has a clk. This is not some crazy form of cross coupled NOR or NAND gate that is frequently shown in textbooks. Which has priority (**S** or **R**) if both are high? That will never happen so just pick one. It also has active low async reset. This is a handy style flop that we will use frequently.

e. I would like you to use the **always_ff** construct of System Verilog. The file should contain (as comments) the answer to this question: Does the use of the **always_ff** construct ensure the logic will infer a flop? Looking for a little more than a simple yes/no answer

5) (25 pts) I_term_DP.sv

This problem started as in class exercise (Ex09) on Mon Sept 25th

See Exercise09 for more detailed description.



Signal:	Dir:	Description:
clk, rst_n	in	System clock and asynch active low reset
hdng_vld	in	A new **error** signal is valid and should be accumulated into I_term
moving	in	The MazeRunner is moving so PID should be active
err_sat[9:0]	in	The error of the course (heading – desired_heading)
I_term[11:0]	out	The I_term for eventual use in PID control

Submit to the dropbox

a) I_term.sv
b) I_term_tb.sv
c) **I_term_waves.png** (or JPG) that shows you simulated.

Hints on building a shifter

This topology is referred to as a logarithmic shifter.

This is showing a logical right shift, but it is not hard to think about how to change it for left shift. To extend it to perform rotate or shift you can imagine another set of 4 muxes that choose between 0's or the MSBs of previous stage coming in as the new LSBs