ECE 551
Homework #2
Due: Wednesday Feb 20th @ Class

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 13th

1) **(20pts)** Using Dataflow/RTL Verilog, implement a shifter capable of performing right shifts (logical or arithmetic) 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 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 multiple shift amounts).

Signal Name:	Dir:	Description:
src[15:0]	in	Input vector to be logically or arithmetically right shifted.
arith	in	If **arith** is asserted then the right shift should be arithmetic.
amt[3:0]	in	Specifies the amount of the shift 0 to 15 bits
res[15:0]	out	Result of the shift

Submit to the dropbox:

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

2) **(20 pts)** Using dataflow RTL implement a 4-bit wide adder that has a the following interface:

Signal:	Direction:	Description:
A[3:0]	in	Operand 1
B[3:0]	in	Operand 2
cin	in	Carry in
Sum[3:0]	out	Sum of operand 1 & 2
co	out	Carry out from addition of operands

Create a testbench that **exhaustively** tests all combinations of inputs. The testbench should also contain a signal called ***fail***. A behavioral implementation of the adder in the test bench should be used to compare with

our dataflow version. If the two versions differ the ***fail*** signal should be asserted. Submit to the dropbox

a. verilog for the DUT (**adder.v**) **(5pts)**
b. Turn in your verilog for the testbench (**adder_tb.v**)
c. Proof that you ran it to completion successfully.

3) **(20 pts)**

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.v

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 reset 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 J-K FF with active high synchronous reset.

This problem will be started as an exercise on Weds Feb 15th & continued on the 18th

4) **(40 pts)** There are three slides in the project specification that speak about scaling of the different frequency bands and volume. Two of the slides are titled: "Scaling of Bands and Volume" and the third slide is titled: "Band Scaling". Read these slides over carefully.

Create Verilog for a module called **band_scale.v**. This module should implement the functionality outlined on the 3rd slide. It should have the following interface:

Signal:	**Direction:**	**Description:**
POT[11:0]	In	Represents A2D reading from the slide potentiometer used to set the gain for the frequency band of the equalizer. This should be regarded as an unsigned number.
audio[15:0]	In	Represents the signed 16-bit audio signal coming from a FIR filter for the particular band. This number is scaled by the potentiometer reading squared to produce the scaled result.
scaled[15:0]	Out	Signed 16-bit output that is the scaled result of the input audio by the slide potentiometer reading.

Submit a file called **band_scale.v** or **band_scale.sv** to the dropbox for HW2.

Write a testbench for this unit. Submit a file called **band_scale_tb.v** to the dropbox for HW2.

Some things to note:

1. Pay attention to what operands are considered signed and what ones are not.
2. To perform a signed multiply the source and destination signals need to be defined as ***signed***. (i.e. wire signed [14:0] op1,op2;)
3. The saturation detection using bits [28:25] is important.
4. You will need this unit for your project, so do a good job on this.