ECE 551
Homework #2
Due: Thurs Feb 19th @ Class

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

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

You are on your own for this problem…not covered by in class exercise

1) **(20pts)** Look at the project spec. There is a slide titled "Protocol Trigger Logic" (around slide 8). It talks about the general idea that the logic analyzer can be triggered by a match on data received from a serial peripheral (either SPI or UART). Some more details about this topic is also given in a slide titled: "UART Module for Protocol Triggering" (around slide 24) and a slide titled: "SPI Unit for Protocol Triggering" (around slide 27). You can see that there is a ***match*** input that specifies the data you are looking for a match on to generate a trigger. You can also see that there is a ***mask*** input specifying which bits of ***match*** can be considered to be don't cares. Of course there is also the data received from the serial peripheral (call that ***serial_data***). There would also be a signal that would specify when the serial data is valid (call that ***serial_vld***). You are to write a module using **dataflow** Verilog that implements the protocol trigger comparison logic.

Signal:	**Dir:**	**Description:**
serial_data[7:0]	in	Data received from the serial protocol (either UART or SPI) this would come from a receive shift register in the given peripheral. For this HW just consider it a primary input signal.
serial_vld	in	A signal that would be asserted only when ***serial_data*** was valid.
mask[7:0]	In	A set bit indicates the corresponding bit of ***match*** is to be treated as a don't care for purposes of comparison to generate the trigger signal
match[7:0]	In	Specifies the data this unit is looking to match to generate a protocol trigger
prot_trig	out	Asserted if there is a protocol data match.

Submit to the dropbox:

a) Dataflow Verilog of the above described module (**data_comp.v**)
b) A testbench for the above module that tests a number of conditions of both data and mask bits (**data_comp_tb.v**)
c) Proof (image of waveforms) that you bothered simulating

You are on your own for this problem…not covered by in class exercise

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 instantiate or implement a behavioral implementation of the adder to be used as the "golden reference". The testbench should be self checking and should error out and stop if there is a miscompare, or finish with a happy message if all goes well.

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)** You are on your own for this problem…not covered by in class exercise

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 high synchronous reset and an enable.

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 J-K FF with active high synchronous reset.

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.

You are on your own for this problem…not covered by in class exercise

4) **(10 pts)** Research PWM. What is it? List at least 4 distinct applications in which it is used. Draw a block diagram (using high level blocks like counters and comparators and perhaps a flop or two) to show how you would implement a PWM module.

Submit to the dropbox:

a) PWM.txt that contains answers to the above questions.
b) A graphical file of some sort (could be a scan of a hand drawn diagram) showing your block diagram of a PWM module

Covered by an in class exercise started on Friday Feb 14th (because who doesn't love RAM Queues)

5) **(30 pts)** Look at the project spec. There are a slide entitled: "RAM Blocks…(around side 12). Read over this and the subsequent slide. Now create a Verilog model of the dual port RAM you will use for the project. Also create a test bench to simulate and verify its functionality.

Submit to the drop box:

a) The Verilog model of the dual port RAM (**RAMqueue.v**)
b) A test bench exercising its functionality (**RAMqueue_tb.v**)
c) Proof (waveforms or otherwise) that you ran the testbench.

NOTE: The number of entries in the RAM queue (**ENTRIES**), and the width of the required address vectors (**LOG2**) should be parameters that default to 384 and 9 respectively. Please use the names (**ENTRIES** & **LOG2**) for these parameters.