CSE 308: Digital System Design | Homework 3

HOMEWORK POLICY

- The purposes of homework include:
  - developing responsible study skills and work habits,
  - practicing a skill or process that students can do independently but not fluently,
  - previewing or preparing for new content, and reflecting or elaborating on information learned in class to deepen students' knowledge.
- Late submission of homework will result in zero score.
- Independently complete homework to the best of your ability.
- If you have any questions related to homework:
  - study with a friend,
  - form a study group in the class,
  - drop your query on Classroom's Stream.
- Collaboration is fine, but it should be you alone who writes up the answers.
- Academic dishonesty will not be tolerated and will result in at least an F for the homework. Your prospectus clearly states "suspension from the university is the expected penalty" for "plagiarism, cheating, and academic integrity issues" and this includes submitting the work of another person as your own or permitting another to submit yours as his/her own.
- Homework will be checked via the quiz policy. There will usually be one problem directly from the homework on the quiz/exam. If you have done all the homework, the quiz/exam should be automatic! If you do not do the homework, you are very unlikely to do well on the quiz/exam, and you should then expect to fail.

Problem 1: Draw the synthesized circuits described by the Verilog codes below.

(a)

```
module FA (a, b, cin, cout, sum);
        input a, b, cin; //inputs
        output cout, sum; //outputs
        wire w1, w2, w3, w4; // internal nets

        xor #(10) (w1, a, b); //delay time of 10 units
        xor #(10) (sum, w1, cin);
        and #(8) (w2, a, b); //delay time of 8 units
        and #(8) (w3, a, cin);
        and #(8) (w4, b, cin);
        or #(10)(cout, w2, w3, w4);
endmodule
```

(b)

```
module smpl_circuit (A, B, C, x, y);
        input A, B, C;
        output x, y;
        wire e;

        and #(10) g1(e, A, B);
        not #(5) g2(y, C);
        or #(15) g3(x, e, y);
endmodule
```

(c)

```
module D_latch (D, Clk, Q);
        input D, Clk;
        output Q;
        reg Q;

        always @(D or posedge Clk)
                if (Clk)
                        Q = D;
endmodule
```

Problem 2: Implement an 8-bit ripple carry adder, using as building block the 1-bit full adder from **1(a)**. Use the following module template for your top level design:

```
module adder8 (A, B, cin, S, cout);
        input[7:0] A, B;
        input cin;
        output[7:0] S;
        output cout;
        wire c1, c2, c3, c4, c5, c6, c7;

        //place code here

endmodule
```

Problem 3: Implement a 4-bit magnitude comparator using dataflow description (i.e. using continues assignment). Here's the skeleton of the module.

```
module mag_comp4 (A, B, ALB, AGB, AEB);
        input [3:0] A,B;
        output ALB, AGB, AEB;
        //ALB → A<B, AGB → A>B, AEB → A=B

        //place code here

endmodule
```

Problem 4: The following code is intended to implement a 1:2 de-MUX. The input is x and two outputs are y and z, and the switch control is s.

```
wire x, y;
reg z;
always @(x or y or z)
    begin
        if (!s) y=x;
        else z=x;
    end
```

(a) Find out all things wrong in this code.

(b) Does the code produce any latches? If no, state why. If yes, how many latches can be produced? And by which statement(s)?

(c) After all problems you find in **(a)** are fixed, draw the logic diagram of the synthesized hardware from the corrected code.

Problem 5: Below is a short snippet of Verilog code of a digital functional block. Draw the logic circuit diagram that is described by the Verilog code.

```
module M (Q, S, A, B, C);
        output Q;
        input S, A, B, C;
        reg Q;

        always @(posedge C)
                Q = (S) ? A : B ;
endmodule;
```

Problem 6: In this problem, design a 3-bit gray counter with positive reset. When reset, the count value becomes 000. Recall that a gray counter changes only one bit at a time. For example, a 2-bit gray counter has a count sequence 00, 01, 11, 10 corresponding to decimal count values of 0, 1, 2, and 3 respectively. In the gray count sequence, only one bit changes between adjacent count values, and the right-most bit is changed as long as it does not result in a code word that has been visited earlier.

The following are the ports of the module:

C	1-bit clock input, all actions must be on the positive edge
R	1-bit reset, causes reset on the positive edge
OUT	3-bit result

Problem 7: In this module, create a byte-wide 8-to-1 multiplexer. In this case, the value on the 3-bit select line will route 1 of 8 inputs to the output. This module is purely combinatorial.

The following are the ports of the module:

S	3-bit select line
I0, I1, I2, I3, I4, I5, I6, and I7	8-bit data inputs
O	8-bit output

Problem 8: For this design, combine gray counter (from **Problem 6**) with the multiplexer (from **Problem 7**) to create a circuit such that the output of the gray counter controls the select lines of the multiplexer.

The top-level design has the following port definitions:

C	1-bit clock
R	1-bit reset line
D0, D1, D2, D3, D4, D5, D6, and D7	8-bit data inputs
OUT	8-bit data output