CSE341 Computer Organization, Spring 2018 Project#2 Due: 05/4/2018 6pm

This project is due Friday May 4h by 6pm. There is no grade for Problem 0, but you are strongly encouraged to finish it.

You need to work individually for this project. For Problems 1 and 2, you must use STRUCTURAL Verilog which means you develop the code using low-level building blocks such as logic gates and flip-flops. Your code will be something similar to Problem 0. For Problem 3, you can use BEHAVIORAL Verilog which means you develop code from behavioral specifications using arithmetic expressions, high level procedural assignments and control flow structures as in high level languages.

Problem 0 (0 points): Verilog Familiarization Problem

Run the following program and familiarize yourself with Verilog.
(If you are going to do a cut and paste from here, make sure that long lines are copied correctly.
The vi editor may break long lines into multiple lines.)

Sample Program

```
/* A 4 bit ripple carry adder is implemented using structural Verilog HDL */
/* code. a and b are 4 bit inputs and s and c_out are outputs. s is a 4 bit
*/
/* sum output and c_out is a 1 bit carry output. */
module adder ();
reg[3:0] a, b; /* declare data types of inputs a and b */
wire[3:0] s; /* declare data type of output s */
wire c_out; /* declare data type of output c_out */
/* Instantiate the 1 bit full adder module defined below to form the */
/* 4 blocks of ripple carry adder. The input carry is assumed to be zero. */
fulladder FA0(c1, s[0], a[0], b[0], 1'b0);
fulladder FA1(c2, s[1], a[1], b[1], c1);
fulladder FA2(c3, s[2], a[2], b[2], c2);
fulladder FA3(c_out, s[3], a[3], b[3], c3);
/* Testbench to give the inputs and check the output values. The time */
/* variable, inputs and outputs are displayed. */
initial begin /* Beginning of initial block */
/* The monitor statement monitors the value of variables at all instants */
/* and displays the result whenever there is any change. */
$monitor ($time, "a=%b, b=%b, s=%b, cout=%b, c1=%b, c2=%b,
c3=%b ", a, b, s, c_out, c1, c2, c3);
a=0; b=0; /* give specific input values */
#100 $display ($time); /* display the time variable */
#900 a=15; b=15;
#100 $display ($time);
#900 a=0; b=15;
#100 $display ($time);
#900 a=15; b=1;
#100 $display ($time);
#900 a=5; b=7;
#100 $display ($time);
end /* end of the initial block
*/
```

```
endmodule /* end of the adder module */
/* 1 Bit Full adder module */
module fulladder (cout, si, ai, bi, cin);
parameter delay2=1, delay3=2, delay4=3;
/* variables defined as parameter */
input ai, bi, cin; /* declaring inputs */
output cout, si; /* declaring outputs */
and #delay3 (si1, ~ai, ~bi, cin), /* si1=~ai.~bi.cin */
(si2, ~ai, bi, ~cin), /* si2=~ai.bi.~cin */
(si3, ai, ~bi, ~cin), /* si3=ai.~bi.~cin */
(si4, ai, bi, cin); /* si4=ai.bi.cin */
or #delay3 (si, si1, si2, si3, si4); /* si=si1+si2+si3+si4 */
and #delay2 (ci1, ai, bi), /* ci1=ai.bi */
(ci2, ai, cin), /* ci2=ai.cin */
(ci3, bi, cin); /* ci3=bi.cin */
or #delay3 (cout, ci1, ci2, ci3); /* cout=ci1+ci2+ci3 */
endmodule
```

Compare the output of your simulation with the following result
and interpret your findings.

```
OUTPUT OF SIMULATION
===================
Top-level modules:
adder
0a=0000, b=0000, s=xxxx, cout=x, c1=x, c2=x, c3=x
3a=0000, b=0000, s=xxxx, cout=0, c1=0, c2=0, c3=0
4a=0000, b=0000, s=xxx0, cout=0, c1=0, c2=0, c3=0
7a=0000, b=0000, s=0000, cout=0, c1=0, c2=0, c3=0
100
1000a=1111, b=1111, s=0000, cout=0, c1=0, c2=0, c3=0
1003a=1111, b=1111, s=0000, cout=1, c1=1, c2=1, c3=1
1007a=1111, b=1111, s=1110, cout=1, c1=1, c2=1, c3=1
1100
2000a=0000, b=1111, s=1110, cout=1, c1=1, c2=1, c3=1
2003a=0000, b=1111, s=1110, cout=1, c1=0, c2=1, c3=1
2004a=0000, b=1111, s=0001, cout=1, c1=0, c2=1, c3=1
2006a=0000, b=1111, s=0001, cout=1, c1=0, c2=0, c3=1
2007a=0000, b=1111, s=0011, cout=1, c1=0, c2=0, c3=1
2009a=0000, b=1111, s=0011, cout=1, c1=0, c2=0, c3=0
2010a=0000, b=1111, s=0111, cout=1, c1=0, c2=0, c3=0
2012a=0000, b=1111, s=0111, cout=0, c1=0, c2=0, c3=0
2013a=0000, b=1111, s=1111, cout=0, c1=0, c2=0, c3=0
2100
3000a=1111, b=0001, s=1111, cout=0, c1=0, c2=0, c3=0
3003a=1111, b=0001, s=1111, cout=0, c1=1, c2=0, c3=0
3004a=1111, b=0001, s=1110, cout=0, c1=1, c2=0, c3=0
3006a=1111, b=0001, s=1110, cout=0, c1=1, c2=1, c3=0
3007a=1111, b=0001, s=1100, cout=0, c1=1, c2=1, c3=0
3009a=1111, b=0001, s=1100, cout=0, c1=1, c2=1, c3=1
3010a=1111, b=0001, s=1000, cout=0, c1=1, c2=1, c3=1
3012a=1111, b=0001, s=1000, cout=1, c1=1, c2=1, c3=1
3013a=1111, b=0001, s=0000, cout=1, c1=1, c2=1, c3=1
```

```
3100
4000a=0101, b=0111, s=0000, cout=1, c1=1, c2=1, c3=1
4003a=0101, b=0111, s=0000, cout=0, c1=1, c2=1, c3=1
4004a=0101, b=0111, s=1100, cout=0, c1=1, c2=1, c3=1
```

In the following problems, assume all gates have unit propagation delay. You need to submit your Verilog code in the final submission. In your project report, you must provide a brief description of the code (with suitable comments) along with the screenshots of inputs and outputs.

Problem 1: Ripple-Carry Adder Design
Design, simulate and verify a 16-bit ripple-carry adder by performing the following additions/subtractions (the values of a and b are given in decimal): (i) (-10) + (100), (ii) (63) - (-127), (iii) (15) + (95), (iv) (-32) + (79), (v) (-59) + (-16), (vi) (1000) + (2001), (vii) (-3210) + (15).

Problem 2: Design of a 4-bit Magnitude Comparator

Design, simulate and verify a 4-bit magnitude comparator in Verilog. Use a variety of inputs to test your design and include the results in your report. You may approach this problem in a hierarchical fashion. If you do so, first design a single-bit comparator and then wire four copies of the same comparator appropriately to produce a 4-bit comparator.

A single bit comparator circuit has two data inputs, three control inputs and three compare outputs. The three control inputs provide a mechanism for generation of multi-bit comparators by cascading several 1-bit comparators.

This is how the single bit comparator logic works. The A>B output is 1 if "the A input is greater than the B input (AB is 10)" or if "A is equal to B and the > input is 1." The A=B output is 1 if "A is equal to B and the = input is 1." The A < B output is the opposite of the A>B output. This line becomes 1 if "A input is less than B output (AB is 01)" or if "A is equal to B and the < input is 1." Based on this functional description of the 1-bit comparator, Karnaugh maps for its three outputs can be extracted easily.

Since the logic is simple, you really don't need to refer to any literature other than the Verilog manual for its implementation. However, if you need any help, you could look at any elementary books on Logic Design.

If you have any difficulty in understanding this problem, you may first consider a few simple numbers and make comparisons starting at the lsb moving toward the msb. If you like to take a "flat design" approach instead of the hierarchical design paradigm (that is, consider the 4-bit comparator as a single module), such a solution is also acceptable. But in practice, you generally follow a modular/hierarchical design approach.

Problem 3: Design of a Carry Lookahead Adder

Repeat Problem 1 where you now design a 4-bit carry lookahead adder instead of the ripple carry adder.
(YOU MAY USE BEHAVIORAL VERILOG FOR THIS PART).