

Do not copy and paste the module declarations directly from this document! Doing so will add Word formatting that the text editor cannot parse.

Problem 1 (10 points)

- a. Write a structural Verilog description of the BCD-to-Excess-3 Code Converter depicted in Figure 3-2 on page 101 of your textbook. Use the code shown in Figure 4-20 of your textbook (p. 188) as a framework for representing primitive logic gates. (You may represent gates having more than two inputs in your Verilog model.)

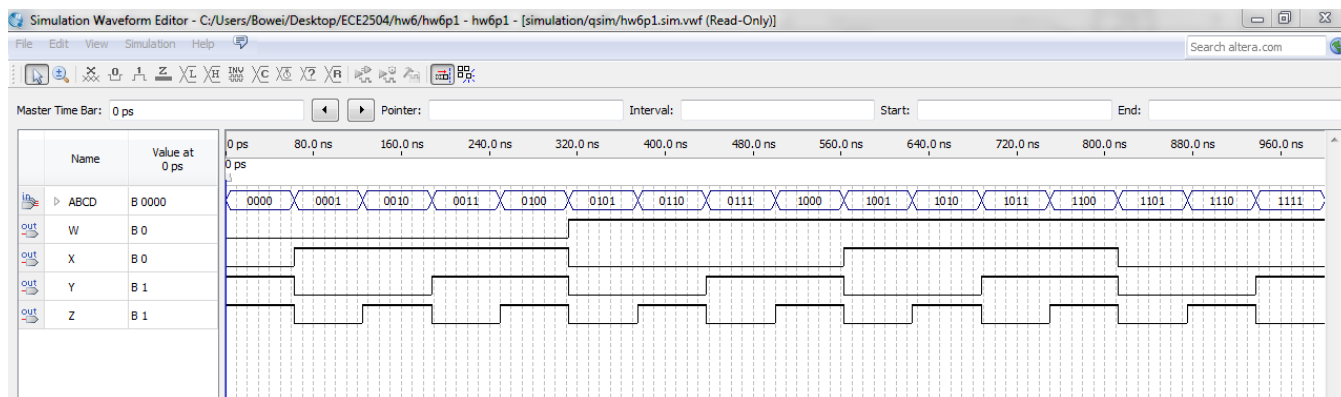
Use the following module declaration to define your circuit:

```
module code_converter_structural(A, B, C, D, W, X, Y, Z);  
    input A, B, C, D;  
    output W, X, Y, Z;
```

Add wires as needed by the structure of the circuit you are describing in Verilog.

```
1 // ECE 2504 - Homework 6  
2 // Problem 1  
3 // Bowei Zhao /bowei  
4 // 28 March 2015  
5 // This is a structural model for the circuit diagram depicted on Page 101 of the  
6 // textbook. The circuit is a BCD-to-Excess-3 converter.  
7  
8 module hw6p1(A, B, C, D, W, X, Y, Z);  
9     input A, B, C, D;  
10    output W, X, Y, Z;  
11    wire n1, n2, n3, n4, n5, n6, n7, n8;  
12  
13    or GATE1(W, A, n1);  
14    and GATE2(n1, B, n2);  
15    or GATE3(n2, C, D);  
16    and GATE4(n3, B, n4);  
17    and GATE5(n7, n5, n2);  
18    or GATE6(X, n7, n3);  
19    and GATE7(n8, C, D);  
20    or GATE8(Y, n4, n8);  
21  
22  
23  
24  
25    not INV1(n4, n2); // inverse for the gate 4  
26    not INV2(n5, B); // inverse for gate 5  
27    not INV3(Z, D); // inverse for Z  
28  
29    endmodule
```

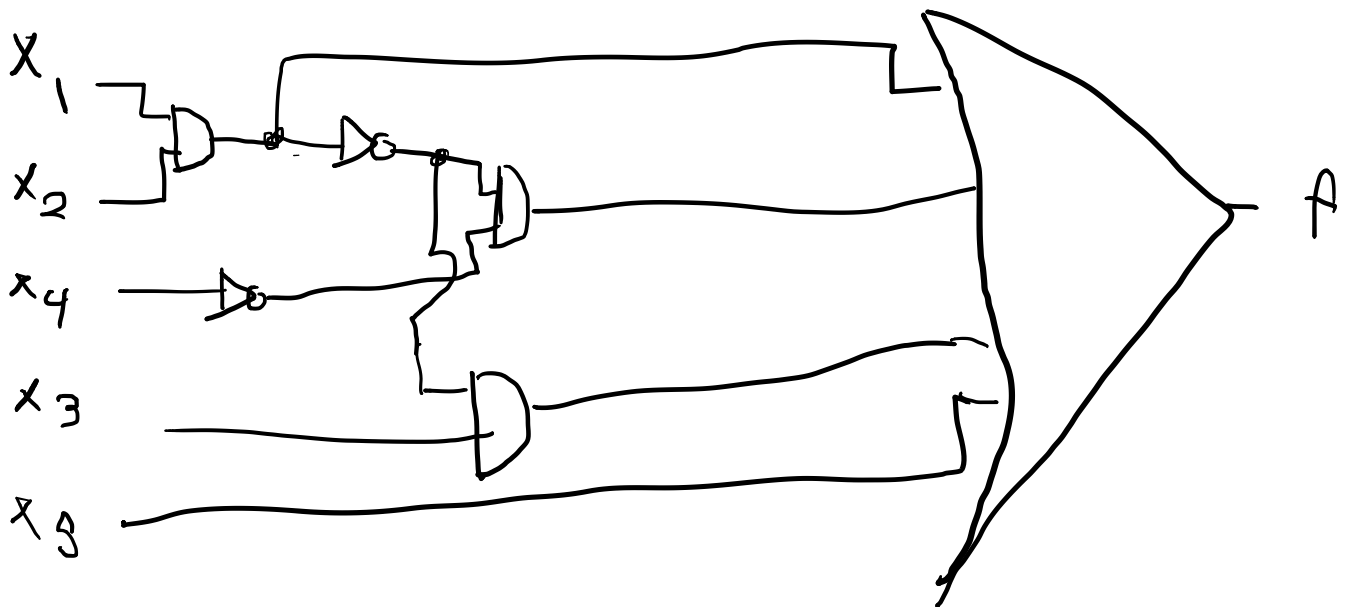
- b. Compile and simulate the circuit in Quartus. Submit the waveform showing all inputs and outputs. **When following the instructions on creating Verilog files for simulation in Quartus, remember that you must name your Verilog files in the same way that you named the module in the file.**



Problem 2 (5 points)

Draw the logic diagram that corresponds to the following structural Verilog description.

```
// Combinational Circuit 1: Structural Verilog Description
module problem2(x1, x2, x3, x4, x5, f);           // 1
// 2
    input x1, x2, x3, x4, x5;                 // 3
    output f;                                 // 4
// 5
    wire n1, n2, n3, n4, n5;                 // 6
// 7
    not not1(n4, n1), not2(n5, x4);           // 8
    and and1(n1, x1, x2), and2(n2, n4, x3), and3(n3, n4, n5); // 8
    or or1(f, n1, n2, n3, x5);               // 9
// 10
endmodule                                     // 11
```



Problem 3 (10 points)

- a. Using the equations that represent a 1-bit full adder, create a *structural* Verilog description for a 4-bit ripple carry adder. You should be able to create a single *structural* model for a 1-bit full adder, and then instantiate the full adder properly for each bit of the ripple carry adder. Use the following model as the framework for your module.

```
module adder4bit (A, B, Cin, S, Cout);
    input [3:0] A,B;          // Two 4-bit addends.
    input Cin;                // 1-bit carry-in to the LSB.
    output [3:0] S;           // The 4-bit sum.
    output Cout;              // 1-bit carry-out from the MSB.

    wire C1, C2, C3;

    adder1bit inst1(A[0], B[0], Cin, S[0], C1 );
    adder1bit inst1(A[1], B[1], C1, S[1], C2 );
    adder1bit inst1(A[2], B[2], C2, S[2], C3 );
    adder1bit inst1(A[3], B[3], C3, S[3], Cout );

endmodule

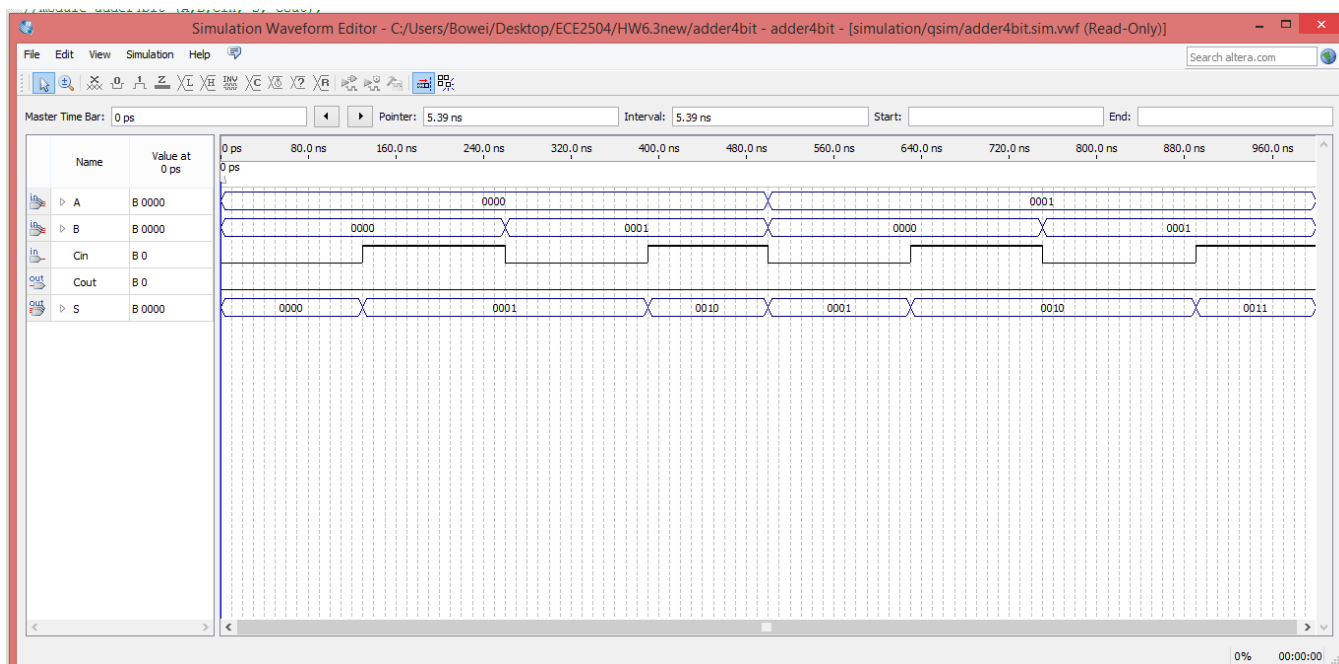
module adder1bit (A, B, Cin, S, Cout);
    input A,B;                // Two 1-bit addends.
    input Cin;                 // 1-bit carry-in.
    output S;                  // The 1-bit sum.
    output Cout;               // 1-bit carry-out from the MSB.

    wire n1, n2, n3;
    and gate1(n1, A,B);
    and gate2(n2,A,Cin);
    and gate3(n3, B, Cin);
    or gate4(Cout, n1, n2, n3);
    xor gate5(S,A,B,Cin);

endmodule
```

You may describe both modules in the same file. Name your file consistently with the top-level module, which in this case is the 4-bit adder.

- b. Compile and simulate your description in Quartus. Apply the combinations that check the least significant 1-bit full adder. There should be eight such input combinations that check all possible combinations of A[0], B[0], and Cin. Since all of the 1-bit full adders are the same, checking the least-significant adder should also serve as a check for the other 1-bit full adders.



Problem 4 (10 points)

- a. Using Figure 4-23 of your textbook (p. 192) as a framework, write a dataflow Verilog description of the circuit from Problem 1 using logic operators.

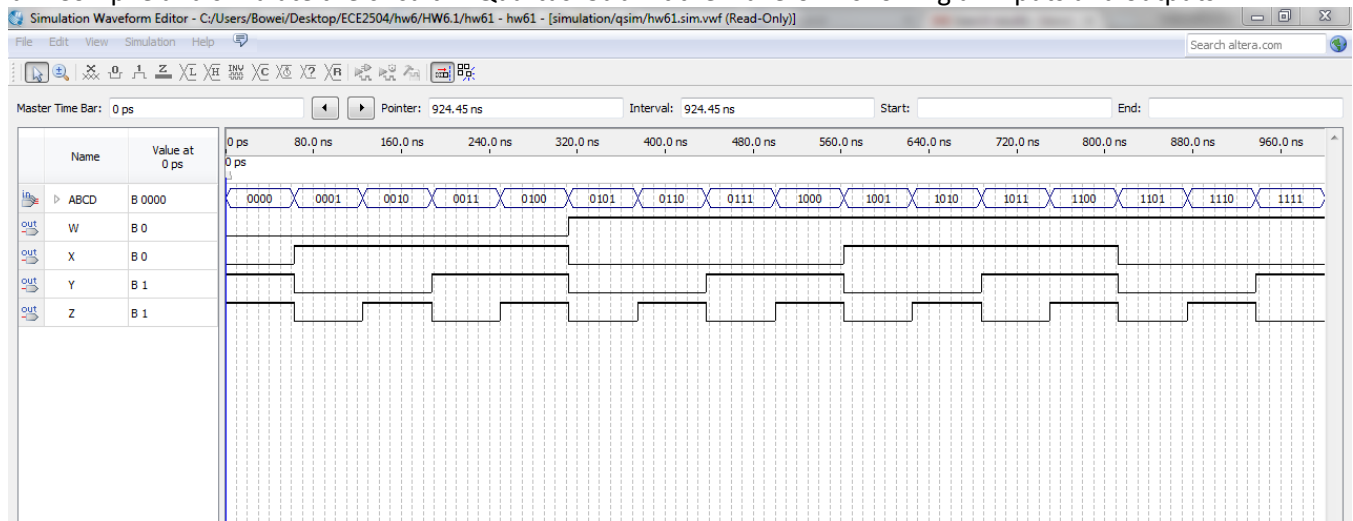
Use the following module declaration to define your circuit:

```
module code_converter_dataflow(A, B, C, D, W, X, Y, Z);  
    input A, B, C, D;  
    output W, X, Y, Z;
```

You may add wires if you need them, but you should be able to use single assign statements to describe each of the outputs.

```
1 // ECE 2504 - Homework 6  
2 // Problem 4  
3 //Bowe Zhao  
4 // 25 March 2015  
5 // This is a structural verilog description model for the circuit diagram depicted on Page 101 of the  
6 // textbook. The circuit is a BCD-to-Excess-3 converter.  
7  
8 module hw61(A, B, C, D, W, X, Y, Z);  
9     input A, B, C, D;  
10    output W, X, Y, Z;  
11  
12    assign W = (((C|D) & B) | A);  
13    assign X = ((~B&(C|D)) | (B&~(C|D)));  
14    assign Y = (~(C|D) | (C&D));  
15    assign Z = (~D);  
16  
17 endmodule
```

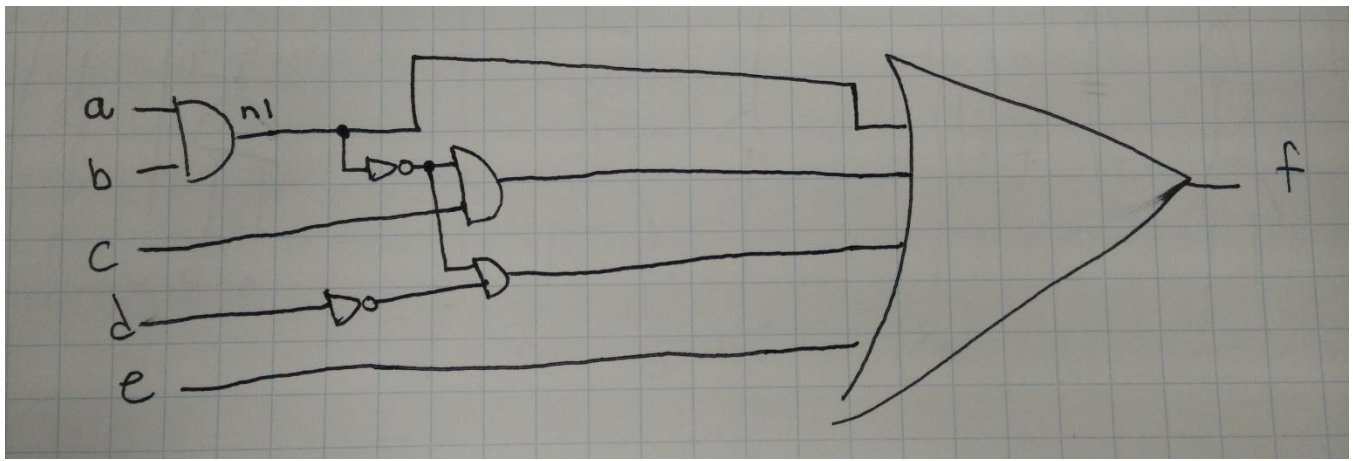
- b. Compile and simulate the circuit in Quartus. Submit the waveform showing all inputs and outputs.



Problem 5 (5 points)

Draw the logic diagram that represents the circuit needed to implement the following Verilog dataflow description.

```
// Combinational Circuit 2: Dataflow Verilog Description
module problem5(f, a, b, c, d, e);                                // 1
                                                                // 2
    input a, b, c, d, e;                                        // 3
    output f;                                                  // 4
    wire n1;                                                  // 5
                                                                // 6
    assign n1 = a & b;                                          // 7
    assign f = n1 | (~n1 & c) | (~n1 & ~d) | e;                // 8
                                                                // 9
endmodule                                                       // 10
```



Problem 6 (10 points)

- a. Using the dataflow concept from Figure 4-24 of your textbook (p. 193), write a Verilog dataflow description for a 16-to-1 multiplexer using the conditional operator. Use the following model as the framework for your module.

```
module multiplexer16to1(sel, A, F);
    input [3:0] sel;        // The 4-bit select value.
    input [15:0] A;         // The 16 1-bit multiplexer inputs.
                             // Using a vector shortens the number of inputs.
    output F;               // The multiplexer output.

    // INSERT THE CODE FOR YOUR CONDITIONAL MULTIPLEXER HERE.

endmodule
```

So, for example, if sel = 0111, then F should equal A[7]. If sel = 1110, then F should equal A[14].

```
1 // ECE 2504 - Homework 6
2 // Problem 6
3 // Bowei Zhao
4 // 25 March 2015
5 // This is a structural model for the circuit diagram depicted on Page 101 of the
6 // textbook. The circuit is a BCD-to-Excess-3 converter.
7
8 module hw62(sel, A, F);
9     input [3:0] sel;        // The 4-bit select value.
10    input [15:0] A;         // The 16 1-bit multiplexer inputs.
11    output F;               // The multiplexer output.
12
13
14    assign F = (sel == 4'b0000) ? A[0] :
15              (sel == 4'b0001) ? A[1] :
16              (sel == 4'b0010) ? A[2] :
17              (sel == 4'b0011) ? A[3] :
18              (sel == 4'b0100) ? A[4] :
19              (sel == 4'b0101) ? A[5] :
20              (sel == 4'b0110) ? A[6] :
21              (sel == 4'b0111) ? A[7] :
22              (sel == 4'b1000) ? A[8] :
23              (sel == 4'b1001) ? A[9] :
24              (sel == 4'b1010) ? A[10] :
25              (sel == 4'b1011) ? A[11] :
26              (sel == 4'b1100) ? A[12] :
27              (sel == 4'b1101) ? A[13] :
28              (sel == 4'b1110) ? A[14] :
29              (sel == 4'b1111) ? A[15] : 1'bx;
30
31 endmodule
32
```


- b. Compile and simulate your circuit in Quartus. Use a set of inputs that are a good test for the selection function it performs. Submit the waveform you generate.

