ECE 2504: Introduction to Computer Engineering
Homework Assignment 6 (50 points)

Bowei Zhao

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.)

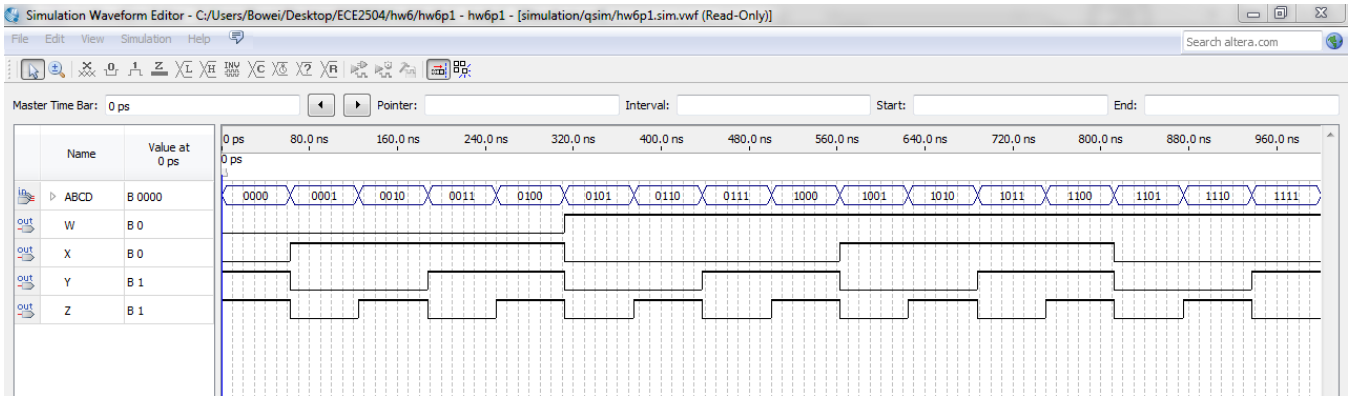
Use the following module declaration to define your circuit:

```
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.

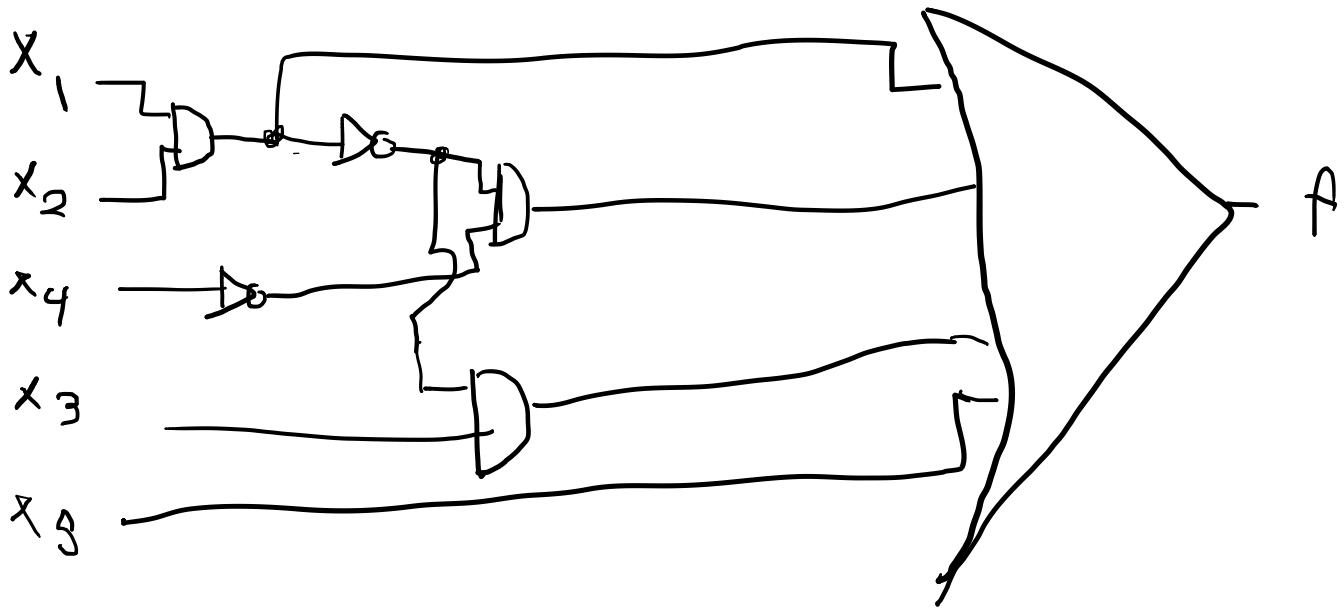
```
1  // ECE 2504 - Homework 6
2  // Problem 1
3  // Bowei Zhao /bowei
4  // 28 March 2015
5  // This is a structural model for the circuit diagram depicted on Page 101 of the
6  // textbook. The circuit is a BCD-to-Excess-3 converter.
7
8  module code_converter_structural(A, B, C, D, W, X, Y, Z);
9     // all the inputs and outputs and wires after being defined in the port
10    input A, B, C, D;
11    output W, X, Y, Z;
12    wire n1, n2, n3, n4, n5, n6 , n7, n8;
13
14    // pretty much just going through doing gate logics and finding outputs for W, X, Y and Z
15 or GATE1(W, A, n1);
16 and GATE2(n1,B,n2);
17 or GATE3(n2,C,D);
18 and GATE4(n3,B,n4);
19 and GATE5(n7, n5, n2);
20 or GATE6(X, n7, n3);
21 and GATE7(n8, C,D);
22 or GATE8(Y, n4, n8);
23
24
25
26 // defining inverters to use with above logic gates
27 not INV1(n4, n2); // inverse for the gate 4
28 not INV2(n5,B); // inverse for gate 5
29 not INV3(Z,D); // inverse for Z
30
31 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.**


Simulation Waveform Editor - C:/Users/Bowei/Desktop/ECE2504/hw6/hw6p1 - hw6p1 - [simulation/qsim/hw6p1.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer:
Interval:
Start:
End:
Name
Value at 0 ps
ABCD
B 0000
W
B 0
X
B 0
Y
B 1
Z
B 1
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

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
                                                                           // 8
    not not1(n4, n1), not2(n5, x4);                                        // 7
    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)

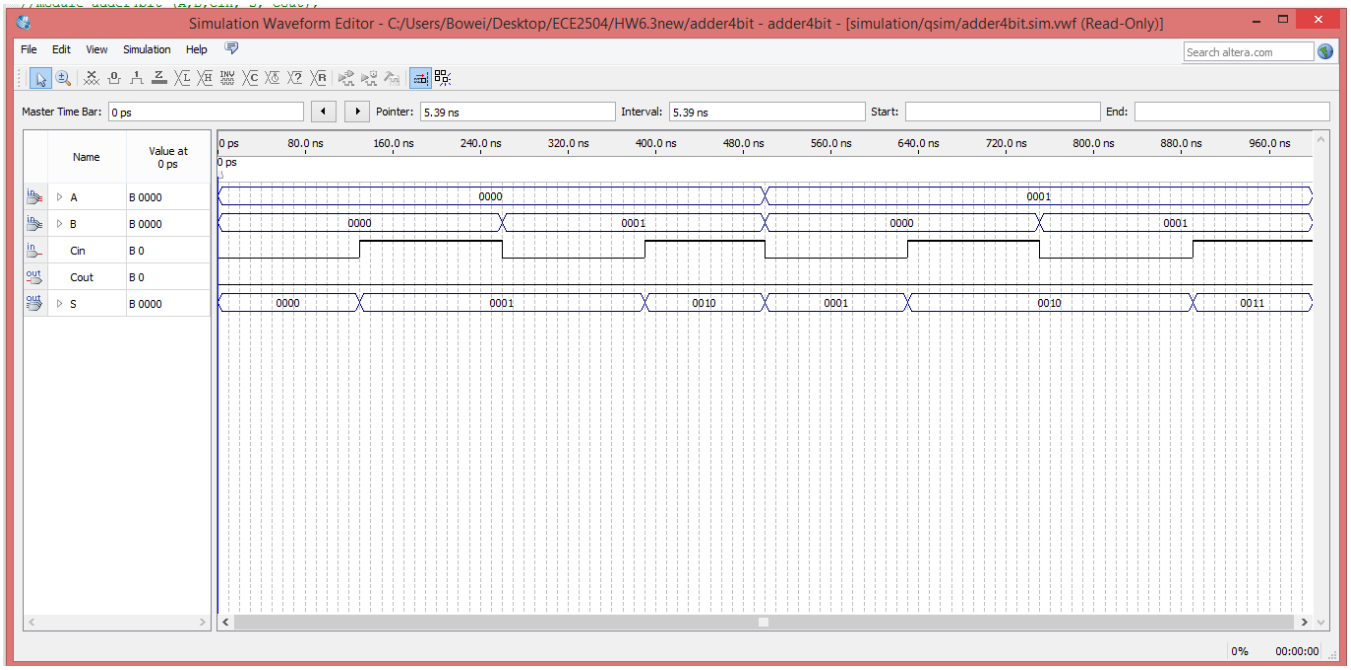
a. Using the equations that represent a 1-bit full adder, create a *structural* Verilog description for a 4-bit ripple carry adder. You should be able to create a single *structural* model for a 1-bit full adder, and then instantiate the full adder properly for each bit of the ripple carry adder. Use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

```
1  // ECE 2504 - Homework 6
2  // Problem 3
3  // Bowei Zhao
4  // 25 March 2015
5  // This is a 4 bit rippel adder
6  // With gates taken from the textbook.
7
8
9  module adder4bit (A,B,Cin, S, Cout);
10    input [3:0]A,B; // inputs for the 4 bit adder
11    input Cin;
12    output [3:0] S;
13    output Cout;
14
15    wire C1, C2, C3;
16
17    // ripple adder is such that it takes in a Cin and A and B initial but then sends it
18    // a left over which in this case will be fed into each other to give an output
19    adder1bit inst1(A[0], B[0], Cin, S[0], C1);
20    adder1bit inst2(A[1], B[1], C1, S[1], C2);
21    adder1bit inst3(A[2], B[2], C2, S[2], C3);
22    adder1bit inst4(A[3], B[3], C3, S[3], Cout);
23
24 endmodule
25
26
27 module adder1bit (A,B,Cin, S, Cout);
28
29    input A, B;
30    input Cin; // carry in
31    output S; // output for select line
32    output Cout;
33
34    wire n1, n2, n3;
35
36    // logic gate representation
37    and gate1(n1, A,B);
38    and gate2(n2,A,Cin);
39    and gate3(n3, B, Cin);
40    or gate4(Cout, n1, n2, n3);
41    xor gate5(S,A,B,Cin);
42
43 endmodule
44
45
46
```

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.


Simulation Waveform Editor - C:/Users/Bowei/Desktop/ECE2504/HW6.3new/adder4bit - adder4bit - [simulation/qsim/adder4bit.sim.vwf (Read-Only)]
File
Edit
View
Simulation
Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 5.39 ns
Interval: 5.39 ns
Start:
End:
Name
Value at 0 ps
A
B 0000
B
B 0000
Cin
B 0
Cout
B 0
S
B 0000
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
0000
0001
0000
0001
0000
0001
0000
0001
0010
0001
0010
0011
0%
00:00:00

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.

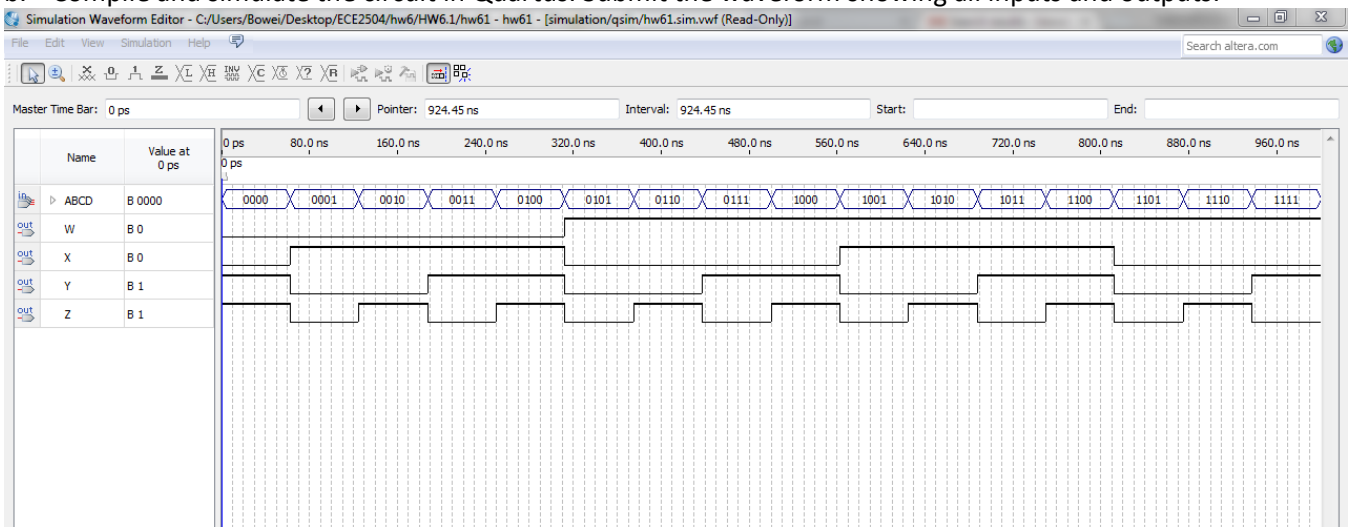
Use the following module declaration to define your circuit:

```
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  //Bowei Zhao
4  // 25 March 2015
5  // This is a structural verilog dataflow from p192 of textbook.
6
7  module code_converter_dataflow(A, B, C, D, W, X, Y, Z);
8     input A, B, C, D;
9     output W, X, Y, Z;
10    // set inputs and outputs depending on ports.
11
12       // using assign operators and boolean arithmetic operation symbols to do work
13    assign W =(((C|D) & B) | A);
14    assign X =((~B&(C|D))| (B&~(C|D)));
15    assign Y = (~(C|D)| (C&D));
16    assign Z = (~D);
17    // outputs for each gate are indepenedent assigns.
18 endmodule
```

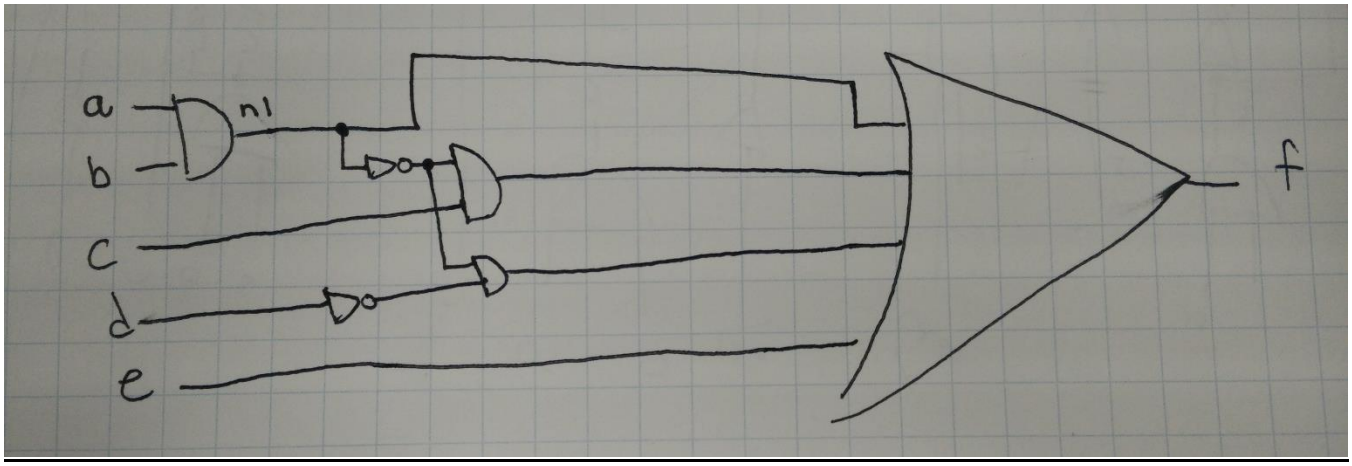
b. Compile and simulate the circuit in Quartus. Submit the waveform showing all inputs and outputs.



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)

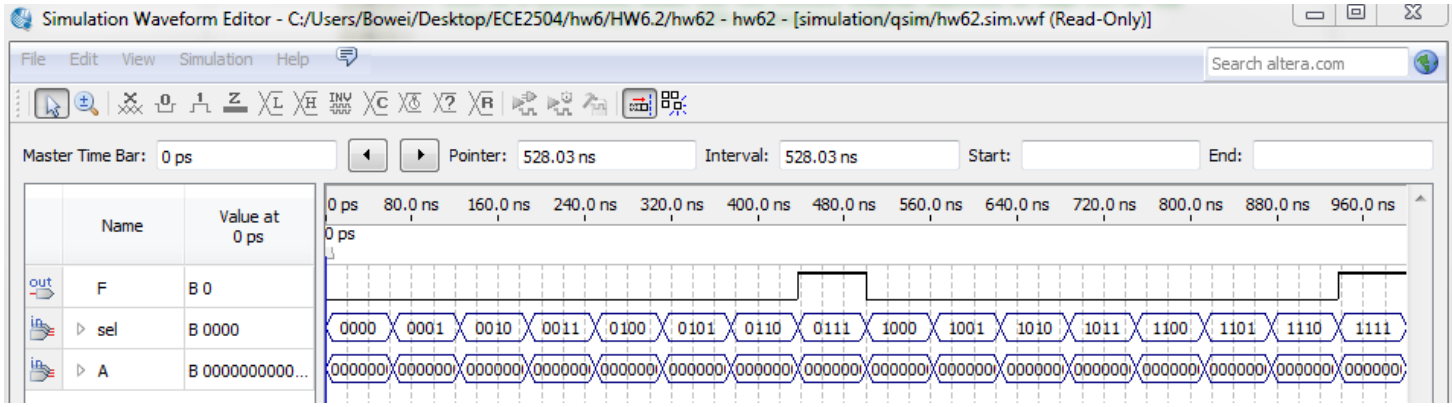
a. Using the dataflow concept from Figure 4-24 of your textbook (p. 193), write a Verilog dataflow description for a 16-to-1 multiplexer using the conditional operator. Use the following model as the framework for your module.

```
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 a 16 to 1 multiplexer
6
7  module hw62(sel, A, F);
8     input [3:0] sel;     // The 4-bit select value.
9     input [15:0] A;      // The 16 1-bit multiplexer inputs.
10    output F;         // The multiplexer output.
11
12
13 assign F =  (sel == 4'b0000) ? A[0] : // where if the select tests binary code from 0000 (0) to 1111 (15) it will thus have gone through
14             (sel == 4'b0001) ? A[1] : // the 16 bits and tested them with the true false statement or else outputting a false of 1'bx
15             (sel == 4'b0010) ? A[2] : // if it was untrue.
16             (sel == 4'b0011) ? A[3] :
17             (sel == 4'b0100) ? A[4] :
18             (sel == 4'b0101) ? A[5] :
19             (sel == 4'b0110) ? A[6] :
20             (sel == 4'b0111) ? A[7] :
21             (sel == 4'b1000) ? A[8] :
22             (sel == 4'b1001) ? A[9] :
23             (sel == 4'b1010) ? A[10] :
24             (sel == 4'b1011) ? A[11] :
25             (sel == 4'b1100) ? A[12] :
26             (sel == 4'b1101) ? A[13] :
27             (sel == 4'b1110) ? A[14] :
28             (sel == 4'b1111) ? A[15] : 1'bx;
29
30 endmodule
31
```

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.


Simulation Waveform Editor - C:/Users/Bowei/Desktop/ECE2504/hw6/HW6.2/hw62 - hw62 - [simulation/qsim/hw62.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 528.03 ns
Interval: 528.03 ns
Start:
End:
Name
Value at 0 ps
F
B 0
sel
B 0000
A
B 0000000000...
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111