Homework 1
Altera Lab Exercise 1

Hints

Part II
The directions for this part ask you to use eight copies of the one-line description of a one-bit two to one multiplexer to create an eight bit wide two to one multiplexer. Read these directions carefully. The one bit multiplexer is given by

```
assign m = (~s & x) | (s & y);
```

For your eight bit multiplexer you will have eight bits of X, eight bits of Y, and the output will be eight bits of M. So your eight bit multiplexer module is going to look like this (make X, Y, and M vectors):

```
module Mux8( S, X, Y, M );
  input S;           // mux select line in
  input  [7:0] X, Y; // mux inputs
  output [7:0] M;    // mux output

  // the mux:
  assign M[0] = (~S & X[0]) | (S & Y[0]);
  assign M[1] =
  assign M[2] =
  assign M[3] =
  assign M[4] =
  assign M[5] =
  assign M[6] =
  assign M[7] =

endmodule
```

Now you have a nice eight bit wide multiplexer module for general use, but what about all the switches and lights? For that you create a higher-level module that interfaces to the DE2 board and includes an instance of your multiplexer. It will look like this

```
module Part2( SW, LEDR, LEDG );
  input  [17:0] SW;    // toggle switches
  output [17:0] LEDR;  // red LEDs
  output [7:0]  LEDG;  // green LEDs
  wire S;              // the select line

  assign S = SW[17];   // clarify what SW[17] does
  assign LEDR = SW;    // show switch settings

  // instance of the 8-bit wide mux:
  Mux8 U1( S, SW[7:0], SW[15:8], LEDG[7:0] );

endmodule
```

I didn't need to define the wire S, but sometimes this helps clarify what various switches really do.

Now Part2() handles all the messy stuff of interfacing to the DE2 board and you can use your eight-bit (uncluttered) mux wherever you want.

We just built a circuit using a hierarchy of modules. The low-level module implements the details of the mux. The higher level module (Part2) contains an instance of the lower level module and interfaces to the DE2. We will try to create a hierarchy of modules for all the circuits we create for the course, with the highest level module interfacing with the DE2 board.

And that's all there is to Part II.

Part III

For this part we are going to create a circuit with more levels in the hierarchy. But the top level module (Part3) will take care of interfacing with all the switches and lights. So as we design the lower level modules we won't be worrying about that aspect.

Once again, carefully reading the directions, we are to build a three-bit wide, five to one multiplexer (Figure 5 in the lab document). Our plan is to take a hint from Part II and build a module that implements a one bit wide, five to one multiplexer (Figure 4c) by implementing the circuit in Figure 4a. Then we'll put three of these together in another module to create a three-bit wide, five to one mux.

When I look at Figure 4a, I think: If I had a one bit, two to one mux module I could create a higher level module with four instances of my one bit mux wired together correctly to create this circuit (Figure 4a). So that's what we'll do. Here's the simple (one bit) multiplexer (we went over these in class):

```
module Mux2_1( X, Y, S, F );
  input X, Y;  // input lines
  input S;     // select line
  output F;    // output

  assign F = (~S & X) | (S & Y);

endmodule
```

And here's how I would implement Figure 4a with them:

```
module Mux5_1( U, V, W, X, Y, S, M );
  input U, V, W, X, Y;   // the input lines
  input [2:0] S;         // the select lines
  output M;

  wire F1, F2, F3;

  // from Lab1, Figure 4a
  Mux2_1 Mux1( U,  V,  S[0], F1 );
  Mux2_1 Mux2( … );
  Mux2_1 Mux3( … );
  Mux2_1 Mux4( … );

endmodule
```

Notice I treat the select lines as a vector, not three individual bits.

Now that we have the one bit wide five to one multiplexer, we just need to group three of these together to form the three bit wide version:

```
module Mux3w_5to1( U, V, W, X, Y, S, M );
  input [2:0] U,V,W,X,Y;  // inputs
  input [2:0] S;          // select lines
  output [2:0] M;         // the output

  // the Mux:
  Mux5_1 Mux0( U[0], V[0], W[0], X[0], Y[0], S, M[0] );
  Mux5_1 Mux1( … );
  Mux5_1 Mux2( … );

endmodule
```

Notice I treat all inputs and outputs as vectors.

Now we just need that high-level module that contains our three bit wide mux and interfaces it to the DE2:

```
module Part3(SW, LEDR, LEDG );
  input  [17:0]SW;    // toggle switches (all inputs)
  output [17:0]LEDR;  // red LEDs (input indicators)
  output [2:0]LEDG;   // green LEDs (output indicators)

  wire [2:0]S;              // the select wires
  wire [2:0]U, V, W, X, Y;  // the inputs
  wire [2:0]M;              // the output

  // make it clear what the switches mean:
  assign S = SW[17:15];
  assign U = SW[2:0];
  assign V = SW[5:3];
  assign W = SW[8:6];
  assign X = SW[11:9];
  assign Y = SW[14:12];
  assign LEDR = SW;      // so we can see the inputs
  assign LEDG[2:0] = M;  // so we can see the outputs
  // the Mux:
  Mux3w_5to1 Mux0( U, V, W, X, Y, S, M );


endmodule
```

In this module I didn't really need to define those variables (wires) S, U, V, etc. For me it just made things clearer to define these variables. And the nice thing is, there is absolutely no penalty in term of the number of gates and so forth in the resulting circuit.

The hierarchy we just built looks something like this:

```
    Part3
      |
 Mux3w_5to1
      |
   Mux5_1
      |
   Mux2_1
```

Now we could have produced this circuit and interfaced it to the DE2 in one big messy module. But this way we have some lower level building blocks we can use again. And we'll definitely need Mux3w_5to1 in Part V.

Part IV
This is a fairly easy part, except that it's easy to fall into the trap of putting everything in one module. You would soon regret doing that because you'll need the decoder module several times over in the next two parts of the lab. So we'll build two modules for this part: a high-level module to interface our decoder to the switches and hex display and the decoder itself.

For the decoder you'll want to write out a big truth table that looks like this:

c2	c1	c0	Seg0	Seg1	…	Seg6
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
etc.						

Then just implement seven functions for seven segments:

$$Seg0 = f_0(c2, c1, c0)$$
$$Seg1 = f_1(c2, c1, c0)$$
$$\ldots$$

And so on.

For instance, for segment 0:

```
  assign Hex[0] = ~((~C[2]&~C[1]&C[0]) + (~C[2]&C[1]&C[0]));
```

There are a couple of things to note about using hex displays. For one thing, you turn a segment ON with a logic 0 (not a logic 1 as you would expect). So either put 0s in the truth table when you want the segment ON and 1s when you would want it off, or enter 1s when you want the segment ON and negate (in your Verilog code) each of the functions you come up with this way. This is what I did because it made more sense to me to write out the truth table with 1s when I wanted the segment on.

The other thing to keep in mind is that the hex displays were designed to be wired in with a big endian convention:

```
output [0:6] HEX0
```

Use this convention otherwise the csv pin assignment file won't work correctly for you.

My top-level module looks like this:

```
module Part4( SW, LEDR, HEX0 );
  input  [2:0]SW;    // toggle switches (inputs C[2:0])
  output [2:0]LEDR;  // red LEDs
  output [0:6]HEX0;  // seven-segment display

  assign LEDR = SW;  // displays the input switches
  HexHELO H( SW, HEX0 );  // hook up the hex display

endmodule
```

And I'll leave it up to you to figure out what's in the decoder module, HexHELO().

Part V
Carefully parse each sentence of Altera's lab write-up and you'll have a pretty good idea of how to lay out this part. Note in Figure 8 the module they name char_7seg is the same thing as my HexHELO module. And their mux_3bit_5to1 module is the same thing as my Mux3w_5to1 module, except I have reordered things so that the select lines are toward the end of the port list, not at the beginning.

Here's the start of extending the Part5 module as described in the write-up (using my module notation):

```
// the multiplexers:
Mux3w_5to1 Mu0( SW[14:12], SW[11:9], SW[8:6], SW[5:3], SW[2:0], SW[17:15], M0 ); // for HEX0
Mux3w_5to1 Mu1( SW[2:0], SW[14:12], SW[11:9], SW[8:6], SW[5:3], SW[17:15], M1 ); // for HEX1
…

// the decoders:
HexHELO H0( M0, HEX0);
HexHELO H1( M1, HEX1);
…
```

And so on.

Part VI

This part is very similar to Part V, except it uses all eight hex displays. So just expand Part V to include these additional displays. And, yes, you will need to build a three-wide eight to one multiplexer module to implement this circuit.