CSC258H1: Pre-lab Exercise for Lab 2

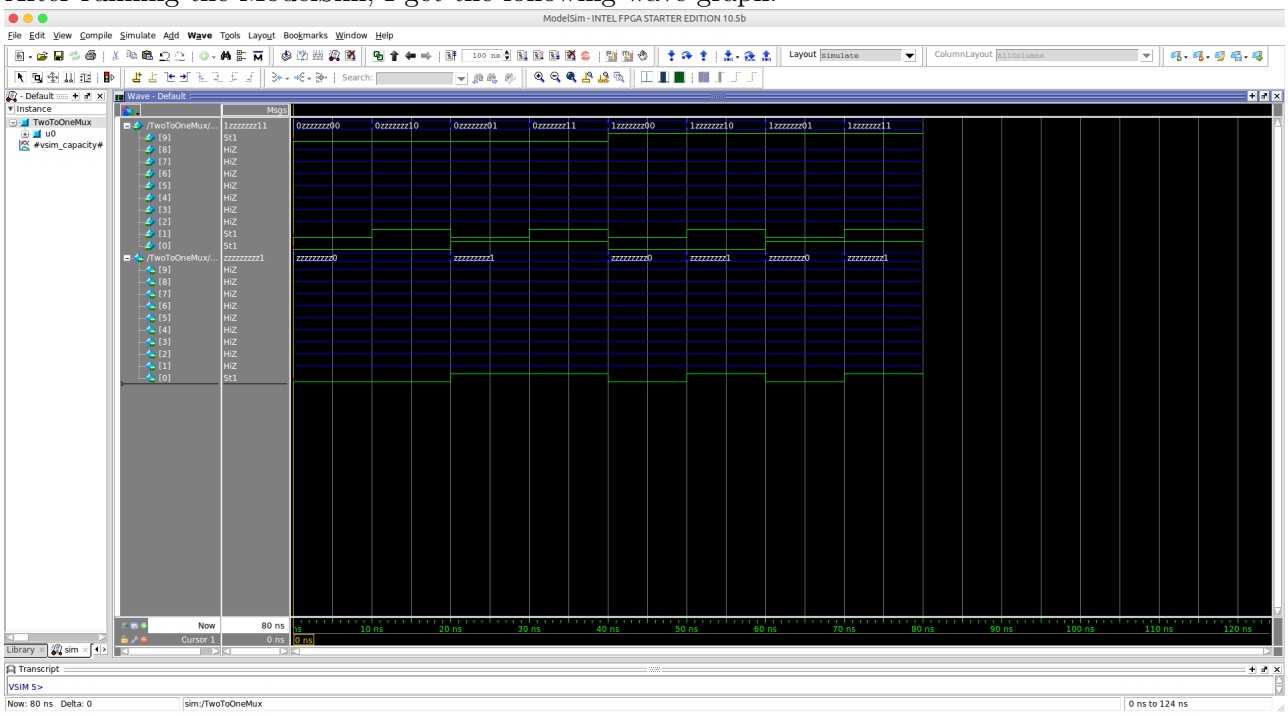
Wang, Weiqing

Due: January 14, 2018 before 12 a.m.

1 Part I

1. When you have familiarized yourself with the *.do* files, open ModelSim, and in the ModelSim's Transcript window (near the bottom) use the **cd** command to change to the directory where you place the *wave.do* and *mux.v* (or the file name you named your *.do* file). Look at the generated waveform, which is the simulation output, and verify that the provided test-cases work as expected.

Solution.
After running the ModelSim, I get the following wave graph:



The simulation output is consistent with my expectation.

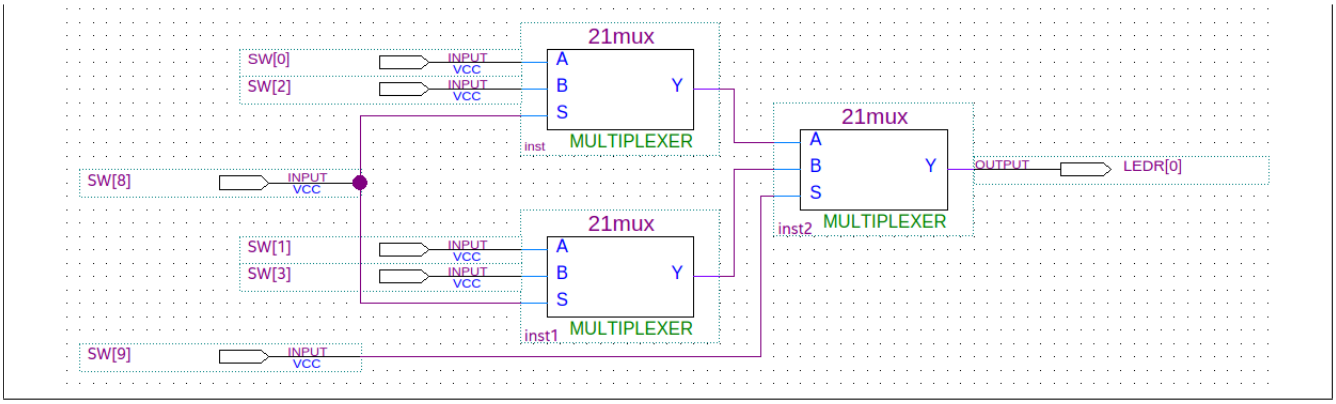
2 Part II

1. Answer the following question: If the truth table in Table 2 was given in full, how many rows would it have?

Solution.
There will be $2^6 = 64$ rows since there are 6 inputs in total.

2. Draw a schematic (not in Quartus) showing how you will connect the *mux2to1* modules to build the 4-to-1 multiplexer. Be prepared to explain it to the TA as a part of your prelab. The schematic should reflect how you are going to write your Verilog code.

Solution.
The schematic is as follows:



3. Create a new Quartus Prime project for your circuit and write the Verilog code.

Solution.
You may view the Verilog below, or see the attachment `FourToOneMux.v`.

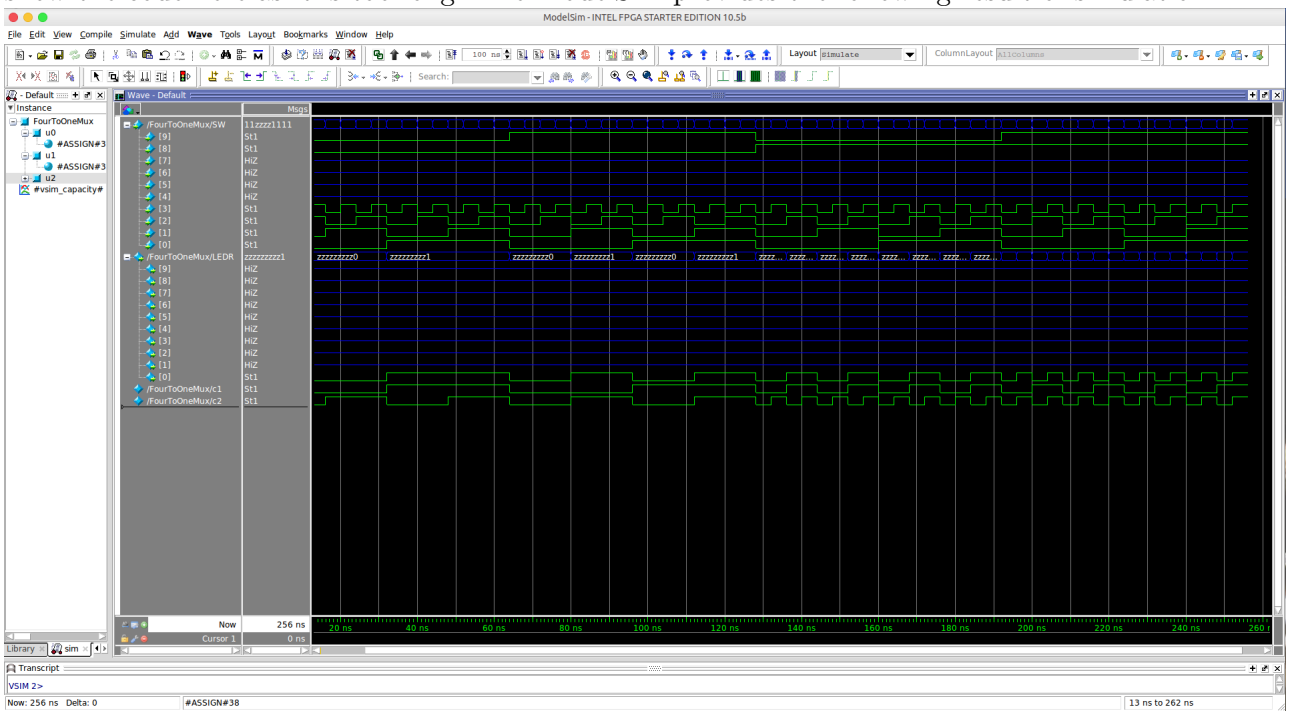
```
//SW[0:3] data inputs
//SW[9:8] select signal

//LEDR[0] output display

module FourToOneMux(LEDR, SW);
    input [9:0] SW;
    output [9:0] LEDR;
    mux2to1 u0(
        .x(SW[0]), // Input u
        .y(SW[2]), // Input w
        .s(SW[8]), // Input s0
        .m(c1)     // Wire c1
        );
     mux2to1 u1(
        .x(SW[1]), // Input v
        .y(SW[3]), // Input x
        .s(SW[8]), // Input s0
        .m(c2)     // Wire c2
        );
     mux2to1 u2(
        .x(c1),       // Output of u0
        .y(c2),       // Output of u1
        .s(SW[9]),    // Input s1
        .m(LEDR[0])   // Output m
        );
endmodule

module mux2to1(x, y, s, m);
    input x;  // First input
    input y;  // Second input
    input s;  // Switch
    output m; // Output
```

```
34
35      assign m = s ? y : x;
36  endmodule
```

A new Quartus Project was created, however, I am not going to give a screenshot here.

4. Simulate your circuit with ModelSim for different values of u, v, w and x. Do enough simulations to convince yourself that the circuit is working. You must show these to the TA as a part of the Pre-lab.

Solution.
I created a `FourToOneMux.do` file (which is attached) and ran it in ModelSim. I am not going to show the code here as it is too long. The ModelSim provides the following result of simulation:



3 Part III

1. Write the expression for seven Boolean functions, one for each segment of the 7 segment decoder. You must use Karnaugh maps for optimization. You should be able to explain to the TA how you generated these expressions by showing them the truth tables you wrote and Karnaugh maps you used to optimize your circuits

Solution. The truth table is as follows:

c_3	c_2	c_1	c_0	HEX[0]	HEX[1]	HEX[2]	HEX[3]	HEX[4]	HEX[5]	HEX[6]
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0

(Continued on Next Page)

(Continued)

c_3	c_2	c_1	c_0	HEX[0]	HEX[1]	HEX[2]	HEX[3]	HEX[4]	HEX[5]	HEX[6]
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0
1	0	1	0	0	0	0	1	0	0	0
1	0	1	1	1	1	0	0	0	0	0
1	1	0	0	0	1	1	0	0	0	1
1	1	0	1	1	0	0	0	0	1	0
1	1	1	0	0	1	1	0	0	0	0
1	1	1	1	0	1	1	1	0	0	0

Now I will present the table for individual characters:

0	$\overline{c_1}\overline{c_0}$	$\overline{c_1}c_0$	c_1c_0	$c_1\overline{c_0}$
$\overline{c_3}\overline{c_2}$	0	1	0	0
$\overline{c_3}c_2$	1	0	0	0
c_3c_2	0	1	0	0
$c_3\overline{c_2}$	0	0	1	0

Hence, $HEX[0] = \overline{c_3}\overline{c_2}\overline{c_1}c_0 + c_3\overline{c_2}c_1c_0 + c_3c_2\overline{c_1}c_0 + c_3\overline{c_2}c_1c_0$.

1	$\overline{c_1}\overline{c_0}$	$\overline{c_1}c_0$	c_1c_0	$c_1\overline{c_0}$
$\overline{c_3}\overline{c_2}$	0	0	0	0
$\overline{c_3}c_2$	0	1	0	**1**
c_3c_2	**1**	0	**1**	**1**
$c_3\overline{c_2}$	0	0	**1**	0

Hence, $HEX[1] = \overline{c_3}c_2\overline{c_1}c_0 + c_2c_1\overline{c_0} + c_3c_1c_0 + c_3c_2\overline{c_0}$.

2	$\overline{c_1}\overline{c_0}$	$\overline{c_1}c_0$	c_1c_0	$c_1\overline{c_0}$
$\overline{c_3}\overline{c_2}$	0	0	0	1
$\overline{c_3}c_2$	0	0	0	0
c_3c_2	**1**	0	**1**	**1**
$c_3\overline{c_2}$	0	0	0	0

Hence, $HEX[2] = c_3c_2c_1 + c_3c_2\overline{c_0} + \overline{c_3}c_2c_1\overline{c_0}$.

3	$\overline{c_1}\overline{c_0}$	$\overline{c_1}c_0$	c_1c_0	$c_1\overline{c_0}$
$\overline{c_3}\overline{c_2}$	0	1	0	1
$\overline{c_3}c_2$	1	0	**1**	0
c_3c_2	0	0	**1**	0
$c_3\overline{c_2}$	0	0	0	1

Hence, $HEX[3] = c_2c_1c_0 + \overline{c_3}c_2\overline{c_1}c_0 + \overline{c_3}c_2\overline{c_1}c_0 + \overline{c_3}c_2\overline{c_1}\overline{c_0} + c_3\overline{c_2}c_1\overline{c_0}$.

4	$\overline{c_1}\overline{c_0}$	$\overline{c_1}c_0$	c_1c_0	$c_1\overline{c_0}$
$\overline{c_3}\overline{c_2}$	0	**1**	**1**	0
$\overline{c_3}c_2$	**1**	**1**	**1**	0
c_3c_2	0	0	0	0
$c_3\overline{c_2}$	0	**1**	0	0

Hence, $HEX[4] = \overline{c_3}c_0 + \overline{c_3}c_2\overline{c_1} + \overline{c_2}c_1c_0$.

5	$\overline{c_1}\overline{c_0}$	$\overline{c_1}c_0$	c_1c_0	$c_1\overline{c_0}$
$\overline{c_3}\overline{c_2}$	0	**1**	**1**	**1**
$\overline{c_3}c_2$	0	0	**1**	0
c_3c_2	0	1	0	0
$c_3\overline{c_2}$	0	0	0	0

Hence, $HEX[5] = \overline{c_3}\overline{c_2}c_1 + \overline{c_3}\overline{c_2}c_0 + \overline{c_3}c_1c_0 + c_3c_2\overline{c_1}c_0$.

6	$\overline{c_1}\overline{c_0}$	$\overline{c_1}c_0$	c_1c_0	$c_1\overline{c_0}$
$\overline{c_3}\overline{c_2}$	**1**	**1**	0	0
$\overline{c_3}c_2$	0	0	1	0
c_3c_2	1	0	0	0
$c_3\overline{c_2}$	0	0	0	0

Hence, $HEX[6] = \overline{c_3}\overline{c_2}\overline{c_1} + \overline{c_3}c_2c_1c_0 + c_3c_2\overline{c_1}\overline{c_0}$.

2. Write a Verilog Module for the 7-segment decoder taking advantage of the aforementioned expressions.

Solution.
Please see the Verilog below, or you may view by opening the attached `HexDecoder.v` file.

```
//SW[0:3] data inputs
//HEX0[6:1] output display

module HexDecoder(HEX0, SW);
    input [3:0] SW;
    output [6:0] HEX0;
    hex0 u0(
        .x(SW[3]),    // INPUT C3
        .y(SW[2]),    // INPUT C2
        .z(SW[1]),    // INPUT C1
        .w(SW[0]),    // INPUT C0
        .m(HEX0[0])   // OUTPUT HEX0[0]
        );
    hex1 u1(
        .x(SW[3]),    // INPUT C3
        .y(SW[2]),    // INPUT C2
        .z(SW[1]),    // INPUT C1
```

```
        .w(SW[0]),    // INPUT C0
        .m(HEX0[1])   // OUTPUT HEX0[1]
        );
    hex2 u2(
        .x(SW[3]),    // INPUT C3
        .y(SW[2]),    // INPUT C2
        .z(SW[1]),    // INPUT C1
        .w(SW[0]),    // INPUT C0
        .m(HEX0[2])   // OUTPUT HEX0[2]
        );
    hex3 u3(
        .x(SW[3]),    // INPUT C3
        .y(SW[2]),    // INPUT C2
        .z(SW[1]),    // INPUT C1
        .w(SW[0]),    // INPUT C0
        .m(HEX0[3])   // OUTPUT HEX0[3]
        );
    hex4 u4(
        .x(SW[3]),    // INPUT C3
        .y(SW[2]),    // INPUT C2
        .z(SW[1]),    // INPUT C1
        .w(SW[0]),    // INPUT C0
        .m(HEX0[4])   // OUTPUT HEX0[4]
        );
    hex5 u5(
        .x(SW[3]),    // INPUT C3
        .y(SW[2]),    // INPUT C2
        .z(SW[1]),    // INPUT C1
        .w(SW[0]),    // INPUT C0
        .m(HEX0[5])   // OUTPUT HEX0[5]
        );
    hex6 u6(
        .x(SW[3]),    // INPUT C3
        .y(SW[2]),    // INPUT C2
        .z(SW[1]),    // INPUT C1
        .w(SW[0]),    // INPUT C0
        .m(HEX0[6])   // OUTPUT HEX0[6]
        );
endmodule

module hex0(x, y, z, w, m);
    input x;  // First input
    input y;  // Second input
    input z;  // Third input
    input w;  // Fourth input
    output m; // Output

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

```
67
68 endmodule
69
70 module hex1(x, y, z, w, m);
71     input x;  // First input
72     input y;  // Second input
73     input z;  // Third input
74     input w;  // Fourth input
75     output m; // Output
76
77     assign m = (~x & y & ~z & w) | (y & z & ~w) |
78                (x & z & w) | (x & y & ~w);
79
80 endmodule
81
82 module hex2(x, y, z, w, m);
83     input x;  // First input
84     input y;  // Second input
85     input z;  // Third input
86     input w;  // Fourth input
87     output m; // Output
88
89     assign m = (x & y & z) | (x & y & ~w) | (~x & ~y & z & ~w);
90
91 endmodule
92
93 module hex3(x, y, z, w, m);
94     input x;  // First input
95     input y;  // Second input
96     input z;  // Third input
97     input w;  // Fourth input
98     output m; // Output
99
100    assign m = (~x & ~y & ~z & w) | (y & z & w) | (~x & y & ~z & w) |
101               (~x & y & ~z & ~w) | (x & ~y & z & ~w);
102
103 endmodule
104
105 module hex4(x, y, z, w, m);
106     input x;  // First input
107     input y;  // Second input
108     input z;  // Third input
109     input w;  // Fourth input
110     output m; // Output
111
112     assign m = (~x & w) | (~x & y & ~z) | (~y & ~z & w);
113
114 endmodule
115
```

```
module hex5(x, y, z, w, m);
    input x;  // First input
    input y;  // Second input
    input z;  // Third input
    input w;  // Fourth input
    output m; // Output

    assign m = (~x & ~y & z) | (~x & ~y & w) | (~x & z & w) |
               (x & y & ~z & w);

endmodule

module hex6(x, y, z, w, m);
    input x;  // First input
    input y;  // Second input
    input z;  // Third input
    input w;  // Fourth input
    output m; // Output

    assign m = (~x & ~y & ~z) | (~x & y & z & w) | (x & y & ~z & ~w);

endmodule
```

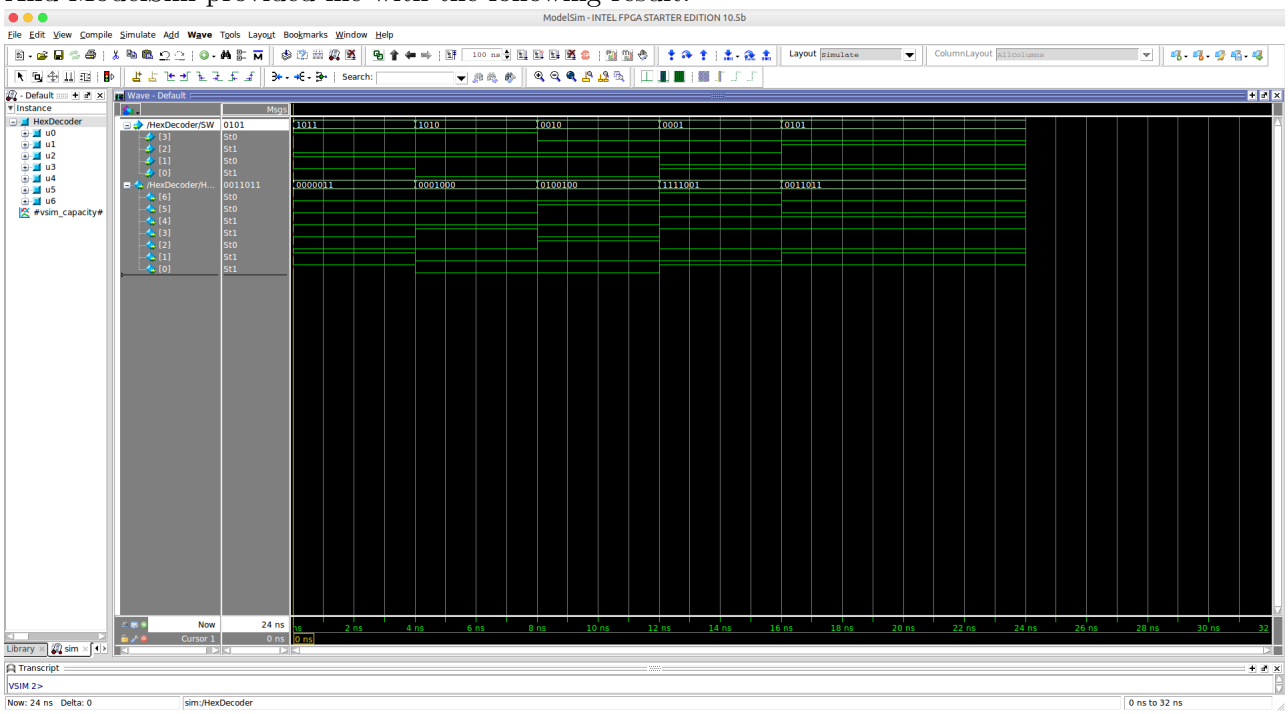
3. In order to be sure that your circuit is correct, you need to simulate your 7-segment decoder module with ModelSim, ensuring the output waveforms are correct. For this purpose of this lab, use the lab room as the test phase (e.g. BA2145, BA3155). Now you should provide this sequence of letters and numbers as the input to your module, and check the output. So for BA3145, provide input for B, then A, then 3, then 1, then 4 and finally 5. Yoiu must include simulation waveforms as part of your pre-lab.

I wrote the `HexDecoder.do` file for simulation, which is both attached to my submission and showed below:

```
vlib work
vlog -timescale 1ns/1ns HexDecoder.v
vsim HexDecoder
log {/*}
add wave {/*}

force {SW[0]} 1
force {SW[1]} 1
force {SW[2]} 0
force {SW[3]} 1
run 4ns

force {SW[0]} 0
force {SW[1]} 1
force {SW[2]} 0
force {SW[3]} 1
```

```
run 4ns

force {SW[0]} 0
force {SW[1]} 1
force {SW[2]} 0
force {SW[3]} 0
run 4ns

force {SW[0]} 1
force {SW[1]} 0
force {SW[2]} 0
force {SW[3]} 0
run 4ns

force {SW[0]} 1
force {SW[1]} 0
force {SW[2]} 1
force {SW[3]} 0
run 4ns

force {SW[0]} 1
force {SW[1]} 0
force {SW[2]} 1
force {SW[3]} 0
run 4ns
```

And ModelSim provided me with the following result:



The result is consistent with my expectation.