HW Exercise 3
(Altera Lab 7, modified)

Finite State Machines

This is an exercise in using finite state machines.

Part I

We wish to implement a finite state machine (FSM) that recognizes two specific sequences of applied input symbols, namely four consecutive 1s or four consecutive 0s. There is an input w and an output z. Whenever $w = 1$ or $w = 0$ for four consecutive clock pulses the value of z has to be 1; otherwise, $z = 0$. Overlapping sequences are allowed, so that if $w = 1$ for five consecutive clock pulses the output z will be equal to 1 after the fourth and fifth pulses. Figure 1 illustrates the required relationship between w and z.
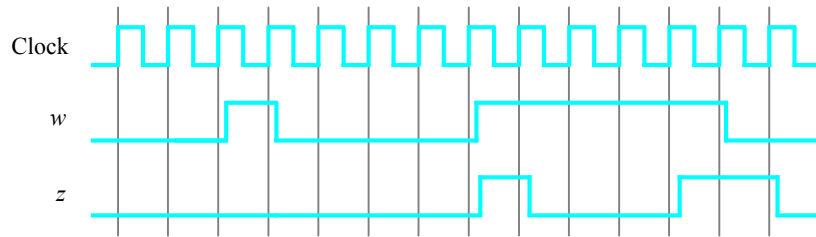

Figure 1. Required timing for the output z.

A state diagram for this FSM is shown in Figure 2. For this part you are to manually derive an FSM circuit that implements this state diagram, including the logic expressions that feed each of the state flip-flops. To implement the FSM use nine state flip-flops called $y_8, \ldots, y_0$ and the one-hot state assignment given in Table 1.

Name	State Code $y_8 y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0$
A	000000001
B	000000010
C	000000100
D	000001000
E	000010000
F	000100000
G	001000000
H	010000000
I	100000000

Table 1. One-hot codes for the FSM.
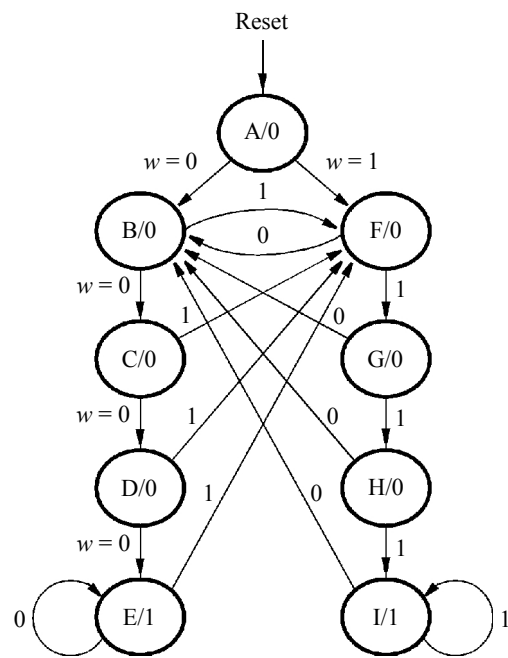
Figure 2. A state diagram for the FSM.

Nothing to turn in for Part 1.
Skip to Part 2.

Part II

For this part you are to write Verilog code for the FSM in Figure 2. Describe the state table for the FSM by using a Verilog **case** statement in an **always** block, and use another **always** block to instantiate the state flip-flops. You can use a third **always** block or simple assignment statements to specify the output z. To implement the FSM, use four state flip-flops $y_3, \ldots, y_0$ and binary codes, as shown in Table 3.

Name	State Code $y_3 y_2 y_1 y_0$
A	0000
B	0001
C	0010
D	0011
E	0100
F	0101
G	0110
H	0111
I	1000

Table 3. Binary codes for the FSM.

A suggested skeleton of the Verilog code is given in Figure 3.

```
   module Part2 ( ... );
   ... define input and output ports

   ... define signals
   reg [3:0] y_Q, Y_D;      // y_Q represents current state, Y_D represents next state
   localparam A = 4'b0000, B = 4'b0001, C = 4'b0010, D = 4'b0011, E =
     4'b0100, F = 4'b0101, G = 4'b0110, H = 4'b0111, I = 4'b1000;

   always @(w, y_Q)
   begin: state_table
        case (y_Q)
         A: if (!w) Y_D = B;
            else Y_D = F;
       ... remainder of state table
       default: Y_D = 4'bxxxx;
         endcase
   end // state_table

   always @(posedge Clock)
   begin: state_FFs
     ...
   end // state_FFS

   ... assignments for output z and the LEDs
 endmodule
```

Figure 3. Skeleton Verilog code for the FSM.

Implement your circuit as follows.

1. Create a new project for the FSM.

2. Include in the project your Verilog file that uses the style of code in Figure 3. Use the toggle switch SW_0 on the Altera DE2 board as an active-low synchronous reset input for the FSM, use SW_1 as the w input, and the pushbutton KEY_0 as the clock input which is applied manually. Use the green LED $LEDG_0$ as the output z, and assign the state flip-flop outputs to the red LEDs $LEDR_3$ to $LEDR_0$. Assign the pins on the FPGA to connect to the switches and the LEDs, as indicated in the User Manual for the DE2 board.

3. Before compiling your code it is necessary to explicitly tell the Synthesis tool in Quartus II that you wish to have the finite state machine implemented using the state assignment specified in your Verilog code. If you do not explicitly give this setting to Quartus II, the Synthesis tool will automatically use a state assignment of its own choosing, and it will ignore the state codes specified in your Verilog code. To make this setting, choose Assignments > Settings in Quartus II, and click on the Analysis and Synthesis item on the left side of the window, then click on the More Setting button. As indicated in Figure 4, change the parameter State Machine Processing to the setting User-Encoded.

4. To examine the circuit produced by Quartus II open the RTL Viewer tool. Double-click on the box shown in the circuit that represents the finite state machine, and determine whether the state diagram that it shows properly corresponds to the one in Figure 2. To see the state codes used for your FSM, open the Compilation Report, select the Analysis and Synthesis section of the report, and click on State Machines.

5. Simulate the behavior of your circuit using ModelSim. Replicate Figure 1 using ModelSim's wave output. Save this figure and turn it in as a jpg file in your Part2 folder.

6. Once you are confident that the circuit works properly as a result of your simulation, download the circuit into the FPGA chip. Test the functionality of your design by applying the input sequences and observing the output LEDs. Make sure that

the FSM properly transitions between states as displayed on the red LEDs, and that it produces the correct output values on $LEDG_0$.

7. In step 3 you instructed the Quartus II Synthesis tool to use the state assignment given in your Verilog code. To see the result of removing this setting, open again the Quartus II settings window by choosing Assignments > Settings, and click on the Analysis and Synthesis item, then click on the More Setting button. Change the setting for State Machine Processing from User-Encoded to One-Hot. Recompile the circuit and then open the report file, select the Analysis and Synthesis section of the report, and click on State Machines. Compare the state codes shown to those given in Table 2, and discuss any differences that you observe.
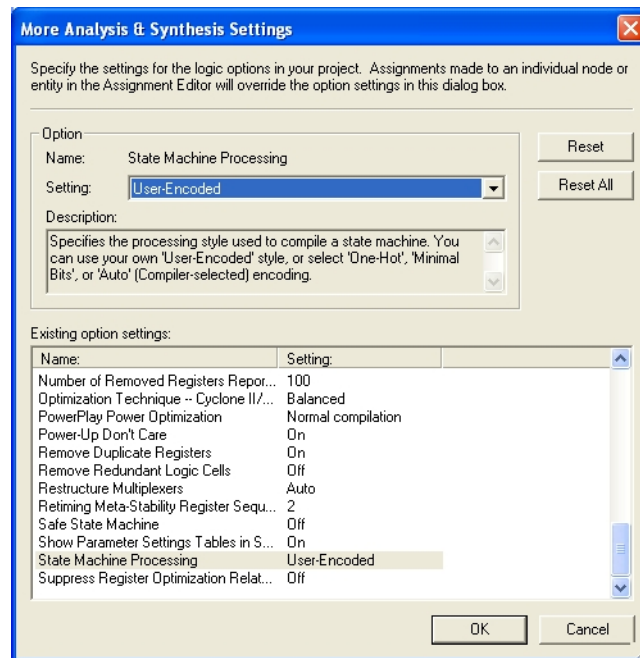

Figure 4. Specifying the state assignment method in Quartus II.

Part III

Skip this part.

Part IV

We want to design a modulo-10 counter-like circuit that behaves as follows. It is reset to 0 by the *Reset* input. It has two inputs, w_1 and w_0, which control its counting operation. If $w_1 w_0 = 00$, the count remains the same. If $w_1 w_0 = 01$, the count is incremented by 1. If $w_1 w_0 = 10$, the count is incremented by 2. If $w_1 w_0 = 11$, the count is decremented by 1. All changes take place on the active edge of a *Clock* input. Use toggle switches SW_2 and SW_1 for inputs w_1 and w_0. Use toggle switch SW_0 as an active-low synchronous reset, and use the pushbutton KEY_0 as a manual clock. Display the decimal contents of the counter on the 7-segment display *HEX0*.

1. Create a new project which will be used to implement the circuit on the DE2 board.

2. Write a Verilog file that defines the circuit. Use the style of code indicated in Figure 3 for your FSM (you will have different state names, etc.).

3. Include the Verilog file in your project and compile the circuit.

4. Simulate the behavior of your state machine using ModelSim. Use ModelSim's wave output. Save the image and turn in the jpg file in your Part4 folder.

5. Assign the pins on the FPGA to connect to the switches and the 7-segment display.

6. Recompile the circuit and download it into the FPGA chip.

7. Test the functionality of your design by applying some inputs and observing the output display.

Part V

For this part you are to design a circuit for the DE2 board that scrolls the word "HELLO" in ticker-tape fashion on the eight 7-segment displays *HEX*7 − 0. The letters should move from right to left each time you apply a manual clock pulse to the circuit. After the word "HELLO" scrolls off the left side of the displays it then starts again on the right side.

Note: this time don't use switches to set up values for the characters, instead define five localparams (i.e. use constants).

Design your circuit by using eight 4-bit registers connected in a queue-like fashion, such that the outputs of the first register feed the inputs of the second, the second feeds the third, and so on. This type of connection between registers is often called a *pipeline*. Each register's outputs should directly drive the seven segment decoder of one display. You are to design a finite state machine that controls the pipeline in two ways:

1. For the first eight clock pulses after the system is reset, the FSM inserts the correct characters (H,E,L,L,O, , ,) into the first of the 7-bit registers in the pipeline.

2. After step 1 is complete, the FSM configures the pipeline into a loop that connects the last register back to the first one, so that the letters continue to scroll indefinitely.

Write Verilog code for the ticker-tape circuit and create a Quartus II project for your design. Use KEY_0 on the DE2 board to clock the FSM and pipeline registers and use SW_0 as a synchronous active-low reset input. Write Verilog code in the style shown in Figure 3 for your finite state machine.
Compile your Verilog code, download it onto the DE2 board and test the circuit.

Part VI

For this part you are to modify your circuit from Part V so that it no longer requires manually-applied clock pulses. Your circuit should scroll the word "HELLO" such that the letters move from right to left in intervals of about one second. Scrolling should continue indefinitely; after the word "HELLO" scrolls off the left side of the displays it should start again on the right side.
Write Verilog code for the ticker-tape circuit and create a Quartus II project for your design. Use the 50-MHz clock signal, *CLOCK_50*, on the DE2 board to clock the FSM and pipeline registers and use KEY_0 as a synchronous active-low reset input. Write Verilog code in the style shown in Figure 3 for your finite state machine, and ensure that all flip-flops in your circuit are

clocked directly by the *CLOCK_50* input. Do not derive or use any other clock signals in your circuit.

Compile your Verilog code, download it onto the DE2 board and test the circuit.

Part VII

Skip this part.

For this assignment you will turn in Part2, Part4, Part5, and Part6.