Laboratory Exercise 7

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*.

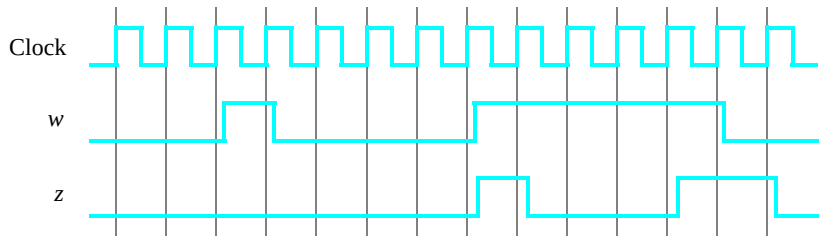

Figure 1. Required timing for the output *z*.

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 $y_8y_7y_6y_5y_4y_3y_2y_1y_0$
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.

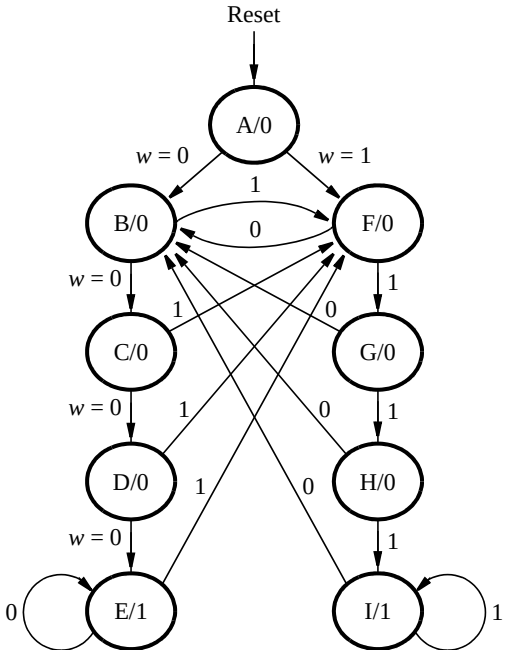

Figure 2. A state diagram for the FSM.

Design and implement your circuit on the DE2 board as follows.

1. Create a new Quartus II project for the FSM circuit. Select as the target chip the Cyclone II EP2C35F672C6, which is the FPGA chip on the Altera DE2 board.

2. Write a Verilog file that instantiates the nine flip-flops in the circuit and which specifies the logic expressions that drive the flip-flop input ports. Use only simple **assign** statements in your Verilog code to specify the logic feeding the flip-flops. Note that the one-hot code enables you to derive these expressions by inspection.

   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 $LEDR_8$ to $LEDR_0$.

3. Include the Verilog file in your project, and assign the pins on the FPGA to connect to the switches and the LEDs, as indicated in the User Manual for the DE2 board. Compile the circuit.

4. Simulate the behavior of your circuit.

5. 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 $LEDG_0$.

6. Finally, consider a modification of the one-hot code given in Table 1. When an FSM is going to be implemented in an FPGA, the circuit can often be simplified if all flip-flop outputs are 0 when the FSM is in the reset state. This approach is preferable because the FPGA's flip-flops usually include a *clear* input port, which can be conveniently used to realize the reset state, but the flip-flops often do not include a *set* input port.

Table 2 shows a modified one-hot state assignment in which the reset state, *A*, uses all 0s. This is accomplished by inverting the state variable y_0. Create a modified version of your Verilog code that implements this state assignment. (*Hint*: you should need to make very few changes to the logic expressions in your circuit to implement the modified codes.) Compile your new circuit and test it both through simulation and by downloading it onto the DE2 board.

Name	State Code $y_8y_7y_6y_5y_4y_3y_2y_1y_0$
A	000000000
B	000000011
C	000000101
D	000001001
E	000010001
F	000100001
G	001000001
H	010000001
I	100000001

Table 2. Modified one-hot codes for the FSM.

Part II

For this part you are to write another style of Verilog code for the FSM in Figure 2. In this version of the code you should not manually derive the logic expressions needed for each state flip-flop. Instead, 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 $y_3y_2y_1y_0$
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
    parameter 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. Select as the target chip the Cyclone II EP2C35F672C6.
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.
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.

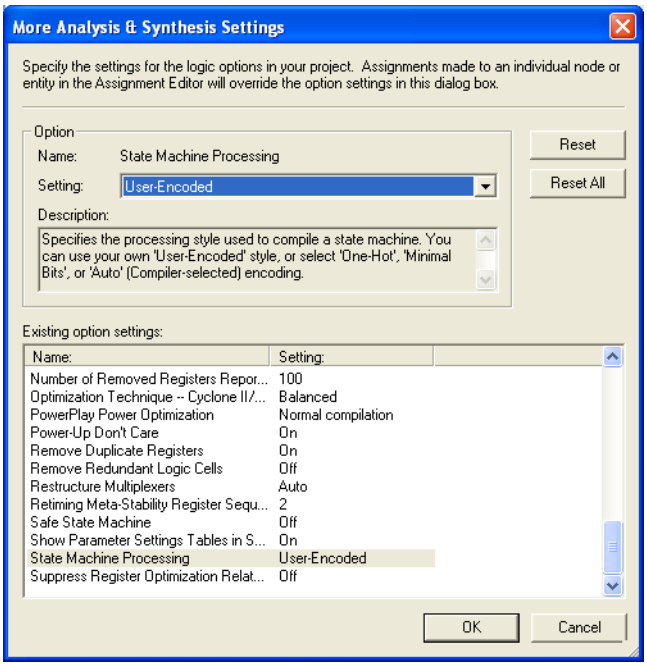

Figure 4. Specifying the state assignment method in Quartus II.

Part III

For this part you are to implement the sequence-detector FSM by using shift registers, instead of using the more formal approach described above. Create Verilog code that instantiates two 4-bit shift registers; one is for recognizing a sequence of four 0s, and the other for four 1s. Include the appropriate logic expressions in your design to produce the output z. Make a Quartus II project for your design and implement the circuit on the DE2 board. Use the switches and LEDs on the board in a similar way as you did for Parts I and II and observe the behavior of your shift registers and the output z. Answer the following question: could you use just one 4-bit shift register, rather than two? Explain your answer.

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 $w_1w_0 = 00$, the count remains the same. If $w_1w_0 = 01$, the count is incremented by 1. If $w_1w_0 = 10$, the count is incremented by 2. If $w_1w_0 = 11$, 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.
3. Include the Verilog file in your project and compile the circuit.
4. Simulate the behavior of your circuit.
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 $HEX7 - 0$. 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.

Design your circuit by using eight 7-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 segments 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,0, , ,) 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

Augment your design from Part VI so that under the control of pushbuttons KEY_2 and KEY_1 the rate at which the letters move from right to left can be changed. If KEY_1 is pressed, the letters should move twice as fast. If KEY_2 is pressed, the rate has to be reduced by a factor of 2.

Note that the KEY_2 and KEY_1 switches are debounced and will produce exactly one low pulse when pressed. However, there is no way of knowing how long a switch may remain depressed, which means that the pulse duration can be arbitrarily long. A good approach for designing this circuit is to include a second FSM in your Verilog code that properly responds to the pressed keys. The outputs of this FSM can change appropriately when a key is pressed, and the FSM can wait for each key press to end before continuing. The outputs produced by this second FSM can be used as part of the scheme for creating a variable time interval in your circuit. Note that KEY_2 and KEY_1 are asynchronous inputs to your circuit, so be sure to synchronize them to the clock signal before using these signals as inputs to your finite state machine.

The ticker tape should operate as follows. When the circuit is reset, scrolling occurs at about one second intervals. Pressing KEY_1 repeatedly causes the scrolling speed to double to a maximum of four letters per second. Pressing KEY_2 repeatedly causes the scrolling speed to slow down to a minimum of one letter every four seconds.

Implement your circuit on the DE2 board and demonstrate that it works properly.