Laboratory Exercise 6 – ECE241 Fall 2014

Finite State Machines

This is an exercise in using finite state machines.

Preparation

You are required to write the Verilog code for Parts I, II and IV (no Verilog is required for Part III). For marking by the teaching assistants, you need to bring with you (pasted into your lab book) your Verilog code for Parts I, II and IV, the simulation output for your circuit for Part I, a print-out from the RTL Viewer and State Machine Viewer of your compiled circuit for Part II, a hand-drawn schematic for Part III, and simulation output from Part IV. Label your simulation output with the specific behaviours being exercised at various points of the simulation.

In-lab Work

You are required to implement and test all of Parts I, II and IV of the lab. But you only need to demonstrate to the teaching assisants Parts II and IV. Your mark will be based on these two parts of the lab.

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 the sequence 1101. There is an input w and an output z. Whenever $w = 1$ for four consecutive clock pulses, or when the sequence 1101 appears on w across 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.

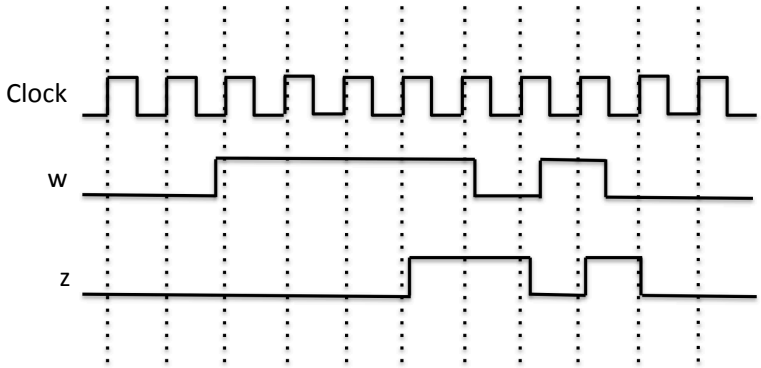

Figure 1: Required timing for the output z.

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 seven state flip-flops called $y_6, \ldots, y_0$ and the one-hot state assignment given in Table 1.

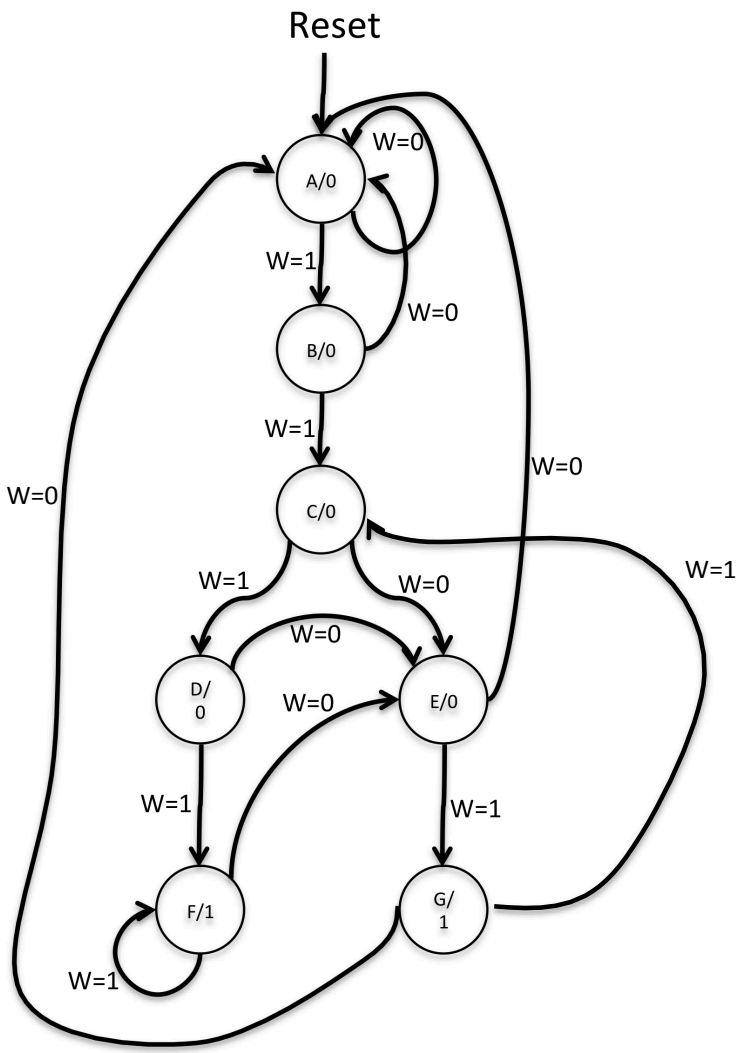

Figure 2: A state diagram for the FSM.

Name	State Code $y_6y_5y_4y_3y_2y_1y_0$
A	0000001
B	0000010
C	0000100
D	0001000
E	0010000
F	0100000
G	1000000

Table 1: One-hot codes for the FSM.

Design and implement your circuit on the DE2-series board as follows:

1. Create a new Quartus II project for the FSM circuit. Select the appropriate target chip that matches the

FPGA chip on the Altera DE2.

2. Write a Verilog file that instantiates the seven 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 DE2-series 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 light $LEDG_0$ as the output z, and assign the state flip-flop outputs to the red lights $LEDR_6$ 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 on the DE2. 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 $LEDG_0$.

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 next-state logic 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. The FSM should use three state flip-flops $y_2, \ldots, y_0$ and the binary state codes, as shown in Table 2.

It is extremely important that you keep the combinational logic and the state flip-flops in separate always blocks. If you mix the two, you will most likely encounter strange behaviour and difficult debugging. The recommended style makes the structure and behaviour of the FSM very clear to the tool and you will have a better chance to get the circuit that you want.

Name	State Code $y_2y_1y_0$
A	000
B	001
C	010
D	011
E	100
F	101
G	110

Table 2: 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 [2:0] y_Q, Y_D;     // y_Q represents current state, Y_D represents next state
    parameter A = 3'b000, B = 3'b001, C = 3'b010, D = 3'b011, E = 3'b100,
        F = 3'b101, G = 3'b110;

    always @(w, y_Q)
    begin: state_table
        case (y_Q)
            A: if (!w) Y_D = A;
                else Y_D = B;
            . . . remainder of state table
            default: Y_D = 3'bxxx;
        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 DE2-series 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 light $LEDG_0$ as the output z, and assign the state flip-flop outputs to the red lights $LEDR_2$ to $LEDR_0$. Assign the pins on the FPGA to connect to the switches and the LEDs on the DE2.
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 Settings button. 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 $LEDG_0$.

Part III

The sequence detector can be implemented in a straightforward manner using a 4-bit shift register, instead of using the more formal FSM design approach. Draw a shift register-based circuit for the sequence detector in your lab book and show it to the TAs as pre-work. No Verilog is required for this step.

Part IV

In this part of the exercise you are to implement a Morse code encoder using an FSM. The Morse code uses patterns of short and long pulses to represent a message. Each letter is represented as a sequence of dots (a short pulse), and dashes (a long pulse). For example, starting from A, eight letters of the alphabet have the following representation:

A	• —
B	— • • •
C	— • — •
D	— • •
E	•
F	• • — •
G	— — •
H	• • • •

Design and implement a Morse code encoder circuit using an FSM. Your circuit should take as input one of the eight letters of the alphabet starting from A (as in the table above) and display the Morse code for it on a red LED, $LEDR_0$. Use switches SW_{2-0} and pushbuttons KEY_{1-0} as inputs. When a user presses KEY_1, the circuit should display the Morse code for a letter specified by SW_{2-0} (000 for A, 001 for B, etc.), using 0.5-second pulses to represent dots, and 1.5-second pulses to represent dashes. The time between pulses is 0.5 seconds. Pushbutton KEY_0 should function as an asynchronous reset.
You are free to design the circuit however you wish. To get started here are some suggested functional blocks to use:

- A counter to keep track of how many symbols will be output. This counter is loaded according to the letter selected. For example, an "E" only has one symbol, while "B" has four symbols;
- A shift register that is parallel loaded with the letter symbol according to the letter selected. A dash is a "1" and a dot is a "0";
- A timer, or maybe more, to time the size of the pulses;
- An FSM to sequence the operations: read the switches to figure out the letter, load the symbol counter and the shift register, read a bit from the shift register, turn on the LED, start the appropriate timer delay and wait for it to expire, turn off the LED, wait 0.5 seconds for the time between symbols, decrement the number of symbols, repeat until the number of symbols remaining is zero.

Determine the resource usage of your circuit in logic elements. Bragging rights to whoever does the circuit with the lowest resource requirements!