Alexandria University
Faculty of Engineering
Electrical Engineering Department

EE432 VLSI Modeling and Design
Lab#6: System-Level Design of Finite State Machines

Objectives

The purpose of this exercise is to to build Finite State Machines (FSMs). The designed circuits are to be implemented on an Altera FPGA board. Upon the completion of this Lab, you should be able to:

1. Design FSMs using the behavioral description style.
2. Use Quartus II software to capture, synthesis, and implement digital FPGA systems and verify your design with Modelsim.
3. Download the design bitstream and test your implementation on the CIC-310 CPLD-FPGA kit.

Requirements

Lab 6 will expose students to the design of Finite State FSMs. A sequence detection FSM will be developed in Verilog. You are required to write testbenches in Verilog to verify the functionality of the developed circuits. Use Modelsim to verify the functionality of the circuits and Quartus to configure the Altera FPGA. Test the implemented circuit using a set of on/off switches, pushbuttons, LEDs, and 7-segment displays offered by the CIC-310 CPLD-FPGA kit.

FSM Description:

Part I:

We wish to implement an 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.

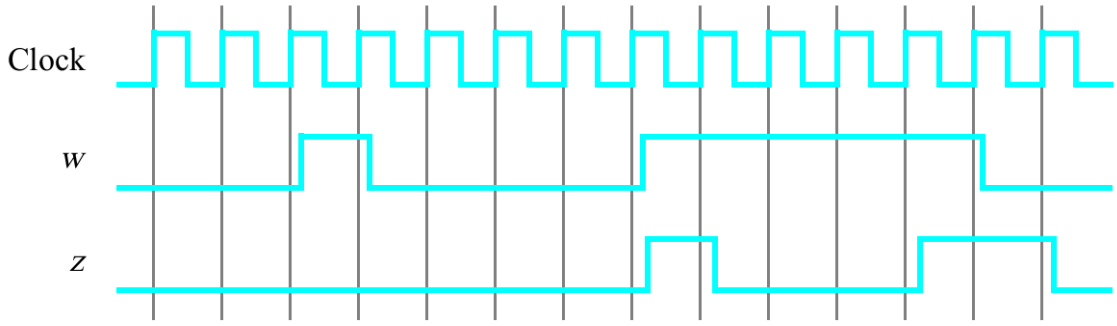

Figure 1: Required timing for the output z

A state diagram for this FSM is shown in Figure 2. For this part you are to manually derive an FSMcircuit 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.

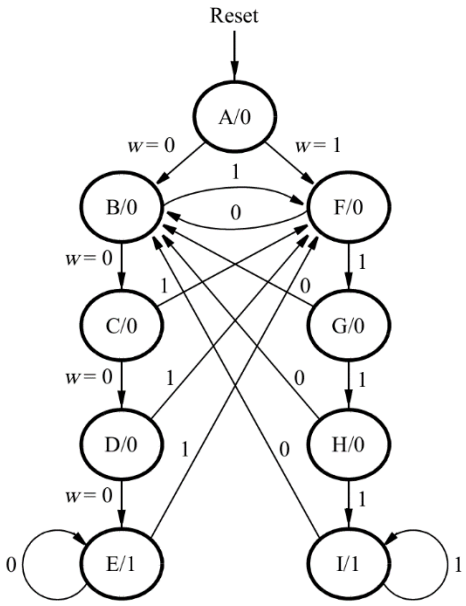

Table1: One-hot code for the FSM

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

Table2: Modified one-hot code for the FSM

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

Figure 2: A state diagram for the FSM

Part II:

We can 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; ... ; y_0 and binary codes, as shown in Table 3.

Table3: Binary codes for the FSM

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

Part III:

The sequence detector can be implemented in a straightforward manner 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*.

Procedures:

Part I:

1. Create a new Quartus II project for the FSM circuit. Select the appropriate target chip that matches the FPGA chip on the Altera CIC-310 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.
3. Use a bushbutton on the board as an active-low synchronous reset input for the FSM, use a switch as the *w* input, and a bushbutton as the clock input which is applied manually. Use s LED as the output *z*, and assign the state flip-flop outputs to nine LEDs.
4. 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 FPGA board. Compile the circuit.

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 flip-flop LEDs, and that it produces the correct output values on the output LED.
7. 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, which can be conveniently used to realize the reset state, but the flip-flops often do not include a set input.
8. 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 state assignment.) Compile your new circuit and test it both through simulation and by downloading it onto the FPGA board.

Part II:

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 a bushbutton on the board as an active-low synchronous reset input for the FSM, use a switch as the *w* input, and a pushbutton as the clock input which is applied manually. Use a LED as the output *z*, and assign the state flip-flop outputs to the red lights LEDR3 to LEDR0. Assign the pins on the FPGA to connect to the switches and the LEDs, as indicated in the User Manual for the CIC-310 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 FSMproperly transitions between states as displayed on the flip-flop LEDs, and that it produces the correct output values on the output LED.
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.

Part III:

1. Make a Quartus II project for your design and implement the circuit on the DE2-series 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.

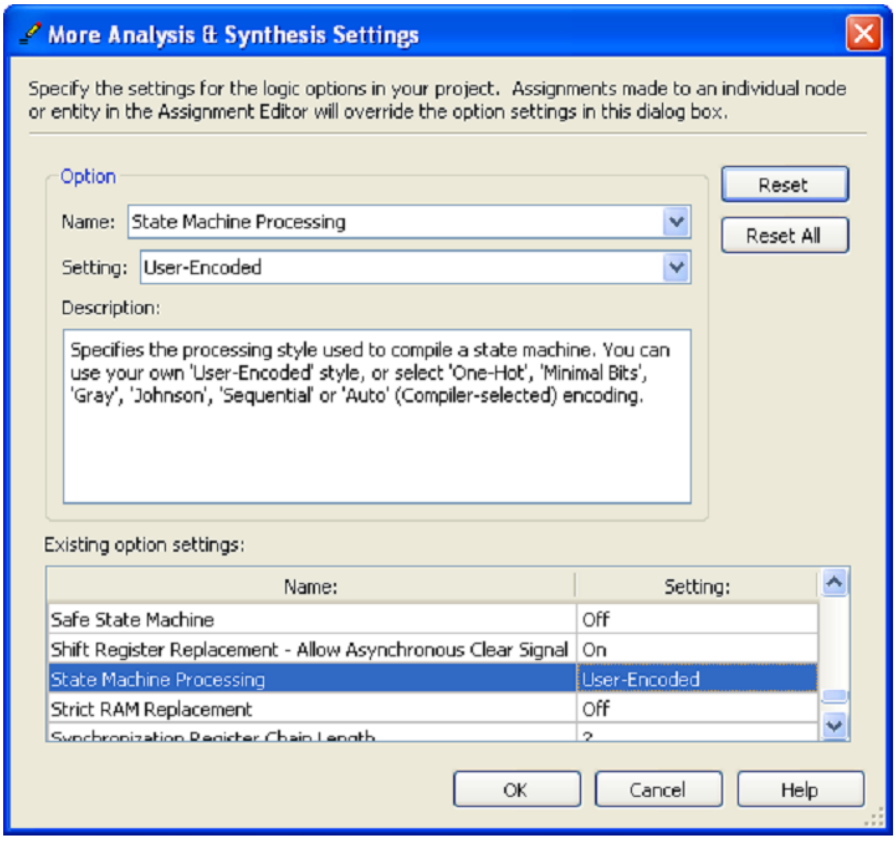

Figure 3: Specifying the state assignment method in Quartus II