

Laboratory 2:

FINITE STATE MACHINES

OBJECTIVES

- Getting to know how to describe finite state machine (FSM) using variety styles of System Verilog code (logic expressions/ behavioral expressions/ shift registers).
- Design and implement digital circuits using FSM.
- Download the circuit into the FPGA chip and test its functionality.

PREPARATION FOR LAB 2

- Finish Pre Lab 2 at home.
- Students have to simulate all the exercises in Pre Lab 2 at home. All results (codes, waveform, RTL viewer, ...) have to be captured and submitted to instructors prior to the lab session.
If not, students will not participate in the lab and be considered absent this session.

REFERENCE

1. Intel FPGA training



Laboratory 2:

FINITE STATE MACHINES

EXPERIMENT 1

Objective: Know how to implement a FSM circuit and download the circuit into the FPGA chip.

Requirement: Implement a 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 . And the state diagram for this FSM is shown in Figure 2.

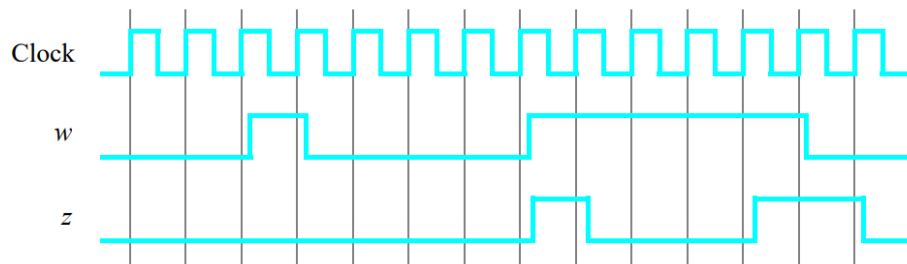


Figure 1: Required timing for the output z .

Instruction:

Students derive an FSM circuit that implements this state diagram, including the logic expressions that feed each of the state flip-flops. Using 9 state flip-flops called y_8, \dots, y_0 and the one-hot state assignment given in Table 1.



Laboratory 2: FINITE STATE MACHINES

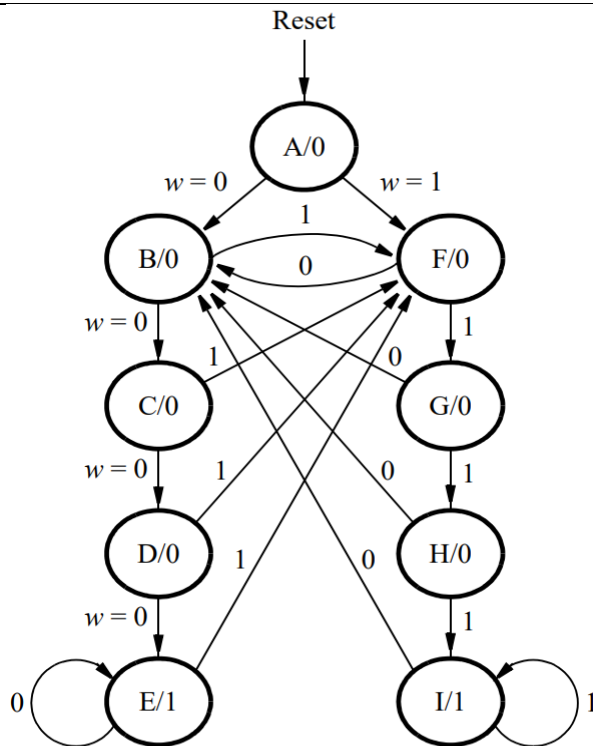


Figure 2: A state diagram for the FSM

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

Table 1: One-hot codes for the FSM

1. Create a new Quartus project for your circuit.
2. Follow FSM circuit designed in exercise - Pre-Lab 2, write a System Verilog file that instantiates the nine flip-flops in the circuit and which specifies the logic expressions that drive the flip-flops input ports.
3. Use the toggle switch SW0 as an active-low synchronous reset input for the FSM, use SW1 as the w input, and the pushbutton KEY0 as the clock input which is applied manually. Use the red light LEDR9 as the output z , and assign the state flip-flop outputs to the red lights LEDR8 to LEDR0.
4. Compile the project, and then download the resulting 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 LEDR9.



Laboratory 2:

FINITE STATE MACHINES

5. It is often desirable to set all flip-flop outputs to the value 0 in the reset state. Table 2 shows a modified one-hot state assignment in which the reset state, A, uses all 0s. Create a modified version of your System Verilog code that implements this state assignment. Compile your new circuit and test it.

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

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.



Laboratory 2:

FINITE STATE MACHINES

EXPERIMENT 2

Objective: Know how to implement a FSM circuit using System Verilog behavioral expressions and download the circuit into the FPGA chip..

Requirement: Implement the FSM given in experiment 1, using another style of System Verilog code. Use a Verilog CASE statement in a 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, \dots, 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

Instruction:

1. Create a new Quartus project for your circuit.
2. Use the same switches, pushbuttons, and lights that were used in previous experiments.
3. It is necessary to explicitly tell the Synthesis tool in Quartus 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, 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, and click on the **Compiler Settings** item on the left side of the window, then click on the **Advanced**



Laboratory 2:

FINITE STATE MACHINES

Settings (Synthesis) button. As indicated in Figure 3, change the parameter **State Machine Processing** to the setting **User-Encoded**.

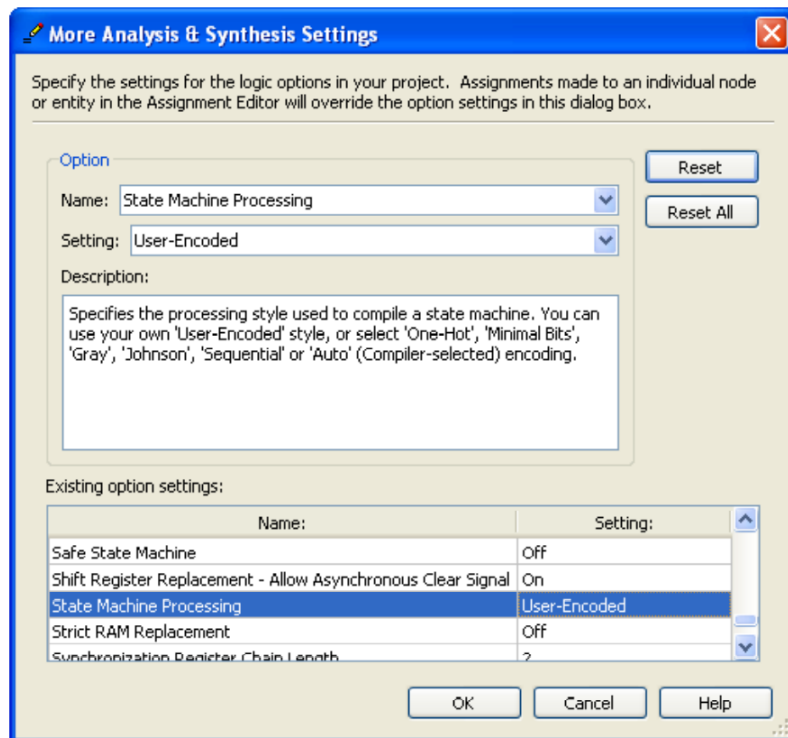


Figure 3: Specifying the state assignment method in Quartus.

4. Compile your project.

Examine the circuit produced by Quartus 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** → **Analysis and Synthesis** section → **State Machines**.

5. Download the circuit into the FPGA chip and test its functionality.

6. 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**



Laboratory 2:

FINITE STATE MACHINES

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.

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.



Laboratory 2:

FINITE STATE MACHINES

EXPERIMENT 3

Objective: Know how to implement sequence detector using shift registers.

Requirement: Create System 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.

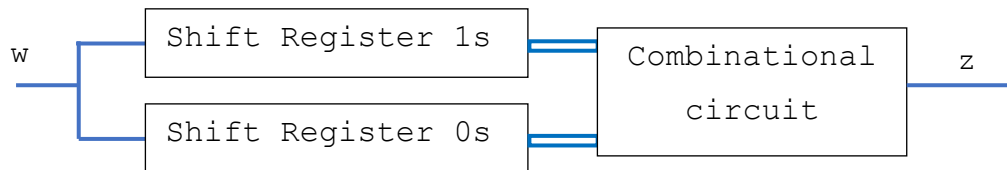


Figure 4: Sequence detector using shift registers.

Instruction:

1. Create a new Quartus project for your circuit.
 2. Use the same switches, pushbuttons, and lights that were used in previous experiments.
 3. Compile your project. Download the circuit into the FPGA chip and test its functionality.
Observe the behavior of your shift registers and the output z.
- ❖ Could you use just one 4-bit shift shift register, rather than two? Explain your answer

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.



Laboratory 2:

FINITE STATE MACHINES

EXPERIMENT 4

Objective: Know how to implement a digital circuit using an FSM.

Requirement: 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, the first 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. The circuit take as input one of the first eight letters of the alphabet and display the Morse code for it on LEDs.

A high-level schematic diagram of a possible circuit for the Morse-code encoder is shown in Figure 5.

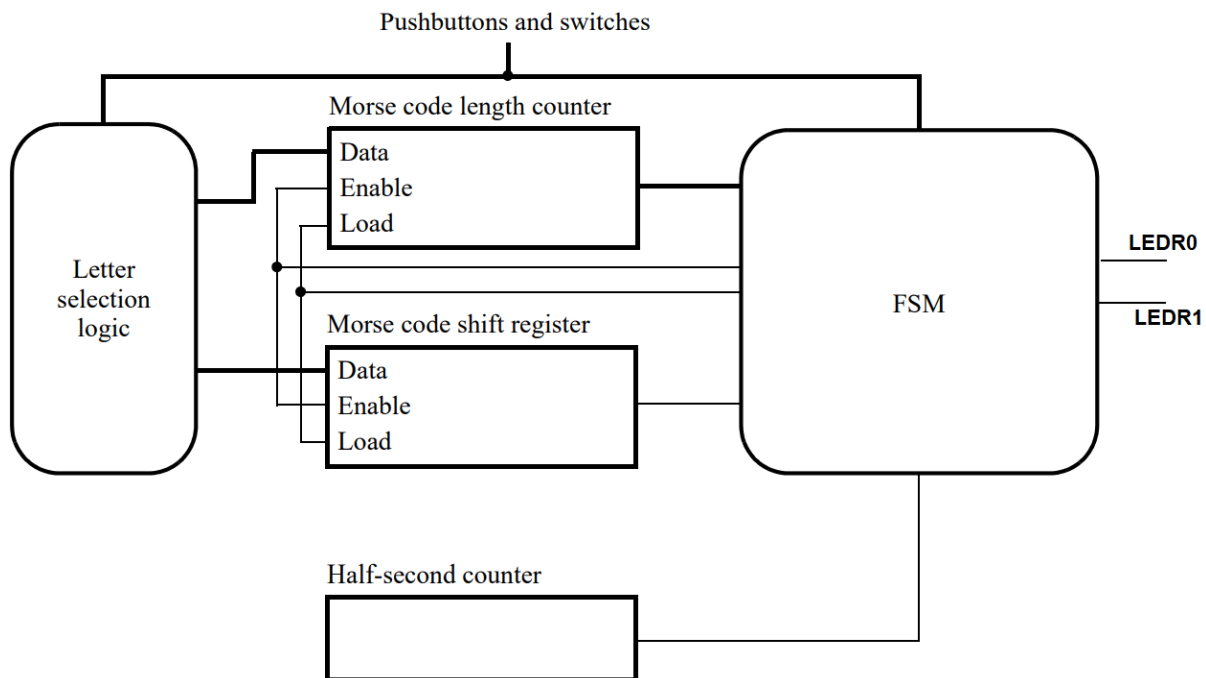


Figure 5: High-level schematic diagram for the Morse-code encoder.



Laboratory 2:

FINITE STATE MACHINES

Instruction:

1. Filling this suggested skeleton of the VHDL code:

```
module part4 (SW, CLOCK_50, KEY, LEDR);

    /*** DECLARATIONS ***/

    /** FSM State Table
    always @(...)
    begin: state_table
        ...
    end // state_table

    /** FSM State flip-flops

    always @(posedge Clock)
        ...
    /** FSM outputs
    // turn on the Morse code light in the states below
    assign light_on = ... ;
    // specify when to load the Morse code into the shift register, and length into the counter
    assign load_regs = ...;
    // specify when to shift the Morse code bits and decrement the length counter
    assign shift_and_count = ...;

    /* Create an enable signal that is asserted once every second. */
    modulo_counter ...;

    /* Letter selection */
    always @(*)
    case (SW)
        A_SW: begin morse_code = ... ; morse_length = ... ; end
        ...
        ...

    endcase

    /* Store the Morse code to be sent in a shift register, and its length in a counter */
    always@(posedge CLOCK_50)
    begin
        /* if Reset = 0 then data = size = 0; otherwise, if load = 1 then data = morse_code and size =
        morse_length; if shift = 1 then data[2:0]= data[3:1]) & data[3] = 1'b0 and size = size - 1'b1 */
    endmodule

module modulo_counter(...);
    ...
endmodule
```



Laboratory 2:

FINITE STATE MACHINES

2. Create a new Quartus project for your circuit.
3. Use switches SW2 – SW0 and pushbuttons KEY1 – KEY0 as inputs. When a user presses KEY1, the circuit should display the Morse code for a letter specified by SW₂₋₀, using a LEDR0 to represent dots, and LEDR1 to represent dashes. Each LED will be on for half of a second. Pushbutton KEY0 should function as an asynchronous reset.
4. Compile your project. Download the circuit into the FPGA chip and test its functionality.

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.

