Elettronica dei Sistemi Digitali
Digital Systems Electronics

Lab#5

FSM

This lab aims at investigating the operation of Finite State Machines. You will need to implement different FSM by starting from different design hypotheses. Hence, you will learn how to design an FSM by using the VHDL description language. **For each of the following designs you have to provide a hand-written block scheme of what you will describe in VHDL.**

Contents:
1. One-Hot Finite state machine
2. Modified One-Hot FSM
2. Two-process FSM
3. "HELLO" FSM

Abbreviations and acronyms:
FSM   – Finite State Machine
IC   – Integrated Circuit
LED   – Light Emitting Diode
MUX   – Multiplexer
RTL   – Register Transfer Level
VHDL – Very high speed integrated circuits Hardware Description Language

1 – One-Hot Finite state machine

We want to implement a Finite State Machine (FSM) that recognizes two specific sequences of input symbols, namely four consecutive 1s or four consecutive 0s. Summarizing we have an input w and the FSM gives 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.
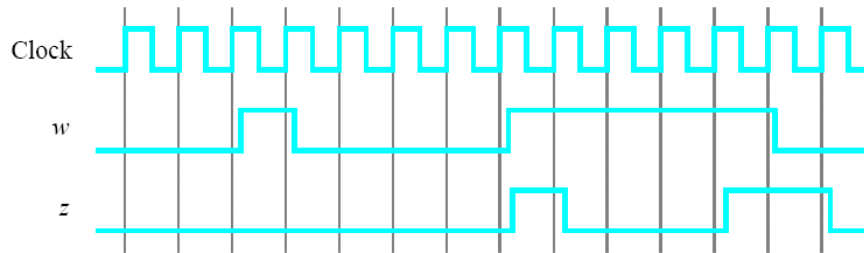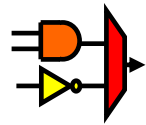
Figure 1 - Required timing for the output z.

A state diagram for this FSM is shown in Figure 2. For this part you need 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 of the FSM.

Design and implement your circuit on the DE1 board as follows:

1. **Create a new Quartus Prime project** for the FSM circuit. Select as the target chip the Cyclone V 5CSEMA5F31C6, which is the FPGA chip on the Altera DE1 board.
2. **Write a VHDL file that instantiates the 9 flip-flops** and specifies the logic expressions that drive the flip-flop input ports. Use only simple assignment statements in your VHDL 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 DE1 board as an active-low synchronous reset input for the FSM, use SW_1 as the w input, and the push button KEY_0 as the clock input which is applied manually. Use the red LED $LEDR_0$ as the output z.
3. **Include the VHDL 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 DE1 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 generates the correct output values on $LEDR_0$.
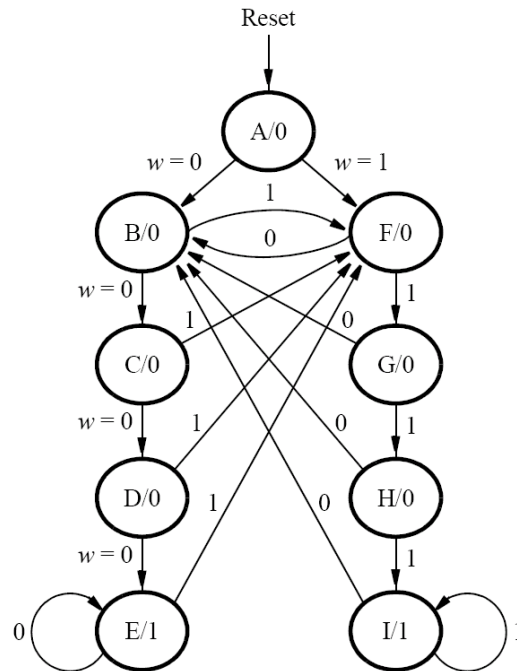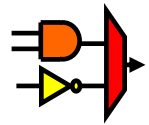
Figure 2 - A state diagram of the FSM.
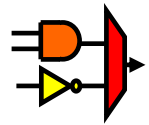
2 – Modified One-Hot FSM

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 VHDL 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 DE1 board. *Keep the same pin assignment.*

3 – Two-process FSM

For this part you need to write another style of VHDL code for the FSM in Figure 2. In this version 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 VHDL CASE statement in a PROCESS block, and use another PROCESS block to instantiate the state flip-flops. You can use a third PROCESS block or simple assignment statements to specify the output *z*.

A VHDL code template is given in Figure 3. Observe that the present and next state vectors of the FSM are defined as an enumerated type with possible values given by the symbols *A* to *I*. The VHDL compiler determines how many state flip-flops the circuit requires and it automatically chooses the state assignment.

Name	State Code $y_8 y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0$
A	000000000
B	000000011
C	000000101
D	000001001
E	000010001
F	000100001
G	001000001
H	010000001
I	100000001

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

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY part2 IS
    PORT ( . . . define input and output ports
          . . .);
END part2;

ARCHITECTURE Behavior OF part2 IS
    . . . declare signals
    TYPE State_type IS (A, B, C, D, E, F, G, H, I);
    SIGNAL y_Q, Y_D : State_type;  -- y_Q is present state, y_D is next
state
BEGIN
. . .
PROCESS (w, y_Q) -- state table
BEGIN
    CASE y_Q IS
        WHEN A =>   IF (w = '0') THEN Y_D <= B;
                    ELSE Y_D <= F;
                    END IF;
        . . . other states
    END CASE;
END PROCESS; -- state table

PROCESS (Clock) -- state flip-flops
BEGIN
. . .
END PROCESS;

. . . assignments for output z and the LEDs

END Behavior;
```
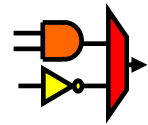
Figure 3 - VHDL code template of the FSM.

Implement your circuit as follows:

1. **Create a new project** for the FSM. Select as the target chip the Cyclone V 5CSEMA5F31C6N.
2. **Include in the project your VHDL file that has the style of code in Figure 3**. Use the toggle switch SW_0 on the Altera DE1 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 red LED $LEDR_0$ as the output z. Assign the pins on the FPGA to connect to the switches and the LED, as indicated in the User Manual for the DE1 board.
3. Before compiling your code it is possible to tell the Synthesis tool in Quartus Prime what **style of state assignment it should use**. Choose Assignments > Settings in Quartus Prime, and then click on the Compiler Settings item on the left side of the window. Then click on Advanced Settings (Synthesis)… then look for State Machine Processing and set Minimal Bits, as indicated in Figure 4.
4. To examine the circuit produced by Quartus Prime 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.
5. **Simulate 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.
7. In step 3 you instructed the Quartus Prime Synthesis tool to use the state assignment given in your VHDL code. To see the result of changing this setting, open again the Quartus Prime settings window by choosing Assignments > Settings, and click on the Compiler Settings item. Change the setting for State Machine Processing from Minimal Bits 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.
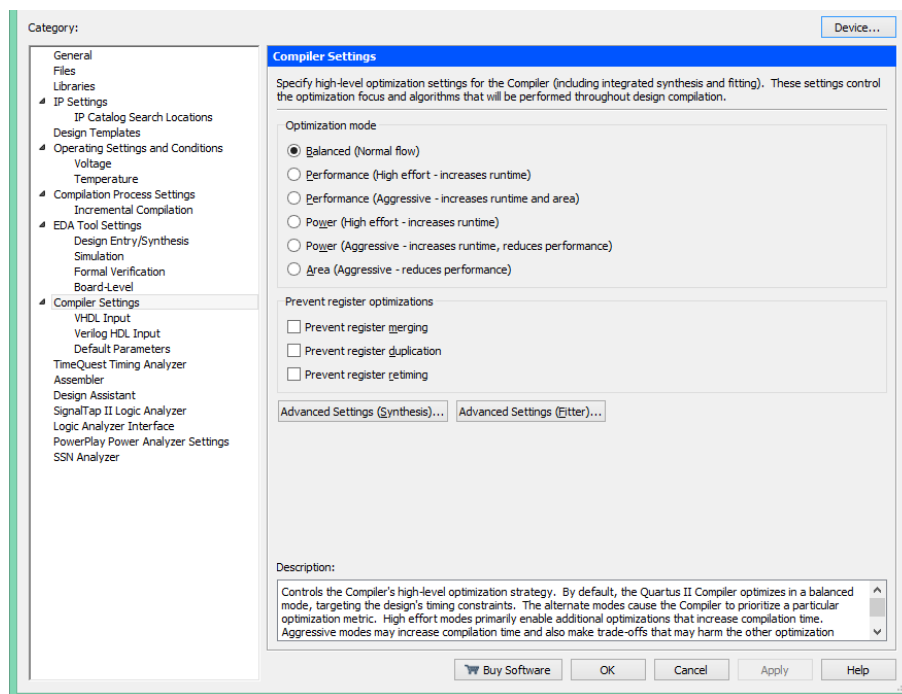
(a) – Settings Window
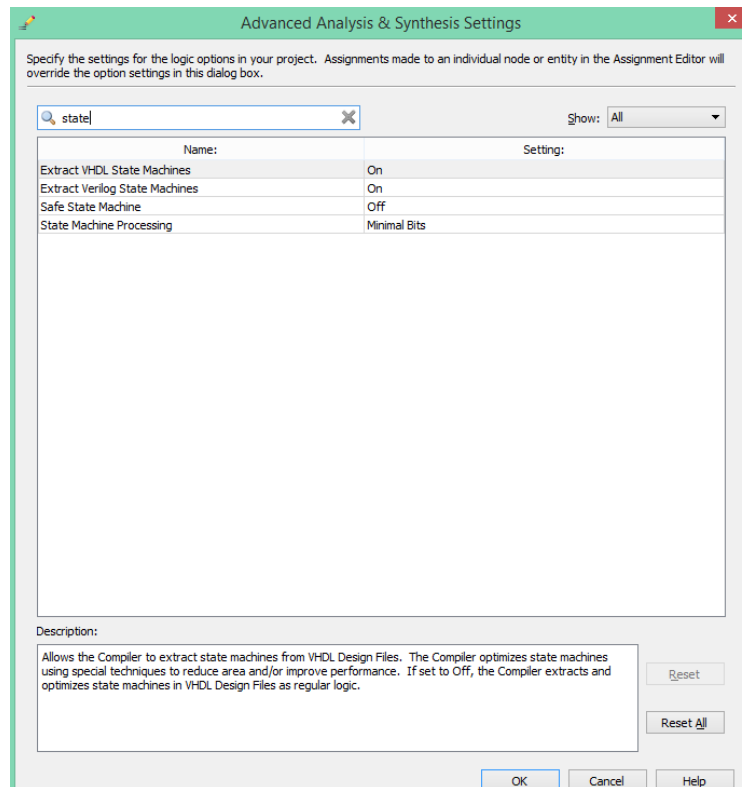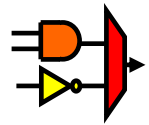
(b) – State Machine Processing ComboBox

Figure 4 - Specifying the state assignment method in Quartus Prime.

4 – "HELLO" FSM

Here you need to design a circuit for the DE1 board that scrolls the word "HELLO" in ticker-tape fashion on the six 7-segment displays $HEX5 - 0$. The letters should move from right to left in intervals of about one second. After the word "HELLO" scrolls off the left side of the displays, it should then start again on the right side.

Design your circuit by using six 7-bit registers connected in a queue-like fashion, in such a way that the outputs of the first register feed the inputs of the second one, the second feeds the third one, and so on (pipeline). Each register's outputs should directly drive the seven segments of one display.

You need to design a finite state machine that controls the pipeline in two ways:

1. For the first six seconds after the system has been 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 every second. Your circuit should scroll the word "HELLO" such that the letters move from right to left. Scrolling should continue indefinitely; after the word "HELLO" scrolls off the left side of the displays it should start again on the right side.
3. Write your VHDL code for the ticker-tape circuit and create a Quartus Prime project for your design. Use the 50-MHz clock signal, *CLOCK_50*, on the DE1 board to clock the FSM and pipeline registers and use *KEY*0 as a synchronous active-low reset input. Write VHDL 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.
4. Compile your VHDL code, download it onto the DE1 board and test the circuit.