Abu Dhabi University

CEN - 466 Advanced Digital Design

Lab Report 3
Sequential Circuits: FSMs

Author:
Muhammad Obaidullah 1030313
Mohammed Farooq 1007778
Mehdi Ismail 1005689

Supervisor:
Dr. Mohammed Assad Ghazal

Section 1

December 8, 2012

Abstract

In this lab we were taught how to use VHDL to solve any problem using Finite State Machine method. Finite state machine is sequential type of implementation. In this Lab we wrote a simple FSM code to jump from one state to another by the use of Clock.

1 Introduction

Some times some problems cannot be simplified to be solved easily by using the traditional combinational logic. Therefore, we use another approach to the method and that is using the clock we break down the problem into sequences of steps / instructions / commands. These instructions are followed in a particular sequence depending upon the clock signal given to them as counter of program execution progress.

Finite State machine method involves solving a problem by thinking of the problem from the view point of a bunch of inputs, outputs and states. In this way we can solve any type of problem regardless of its difficulty. In this lab we tried to make VHDL standard FSM code for these kinds of problems.

2 Experiment Set-up

The Experiment was set up by opening and setting up the VHDL coding Integrated Development Environment Quartus II. The code was written in the Quartus IDE and then we debugged it. After debugging, We assigned the outputs to the pins on the ALTERA board. These outputs can by anything ranging from LEDs to Buzzers. Additionally, there were some inputs to be assigned to or how else could we get the input from the user. These all pins were assigned by referring to the Datasheet of the ALTERA Cyclone II and were assigned by the Pin Planner inside the Quartus.

3 List of Equipment used

- Power cable for Altera board.
- USB cable.
- A PC running Quartus II.
- Altera board.

4 Procedure

- Open Quartus.
- Click on Create a new project.
- Click next.
- Create a folder on a directory of your choice and select it as a working directory for you project.
- Name your project and click next.

- Now choose the board to be “EP2C35F672C6”.
- Click next and finish.
- Now go to File ¿ New ¿ VHDL File.
- Start writing the code.

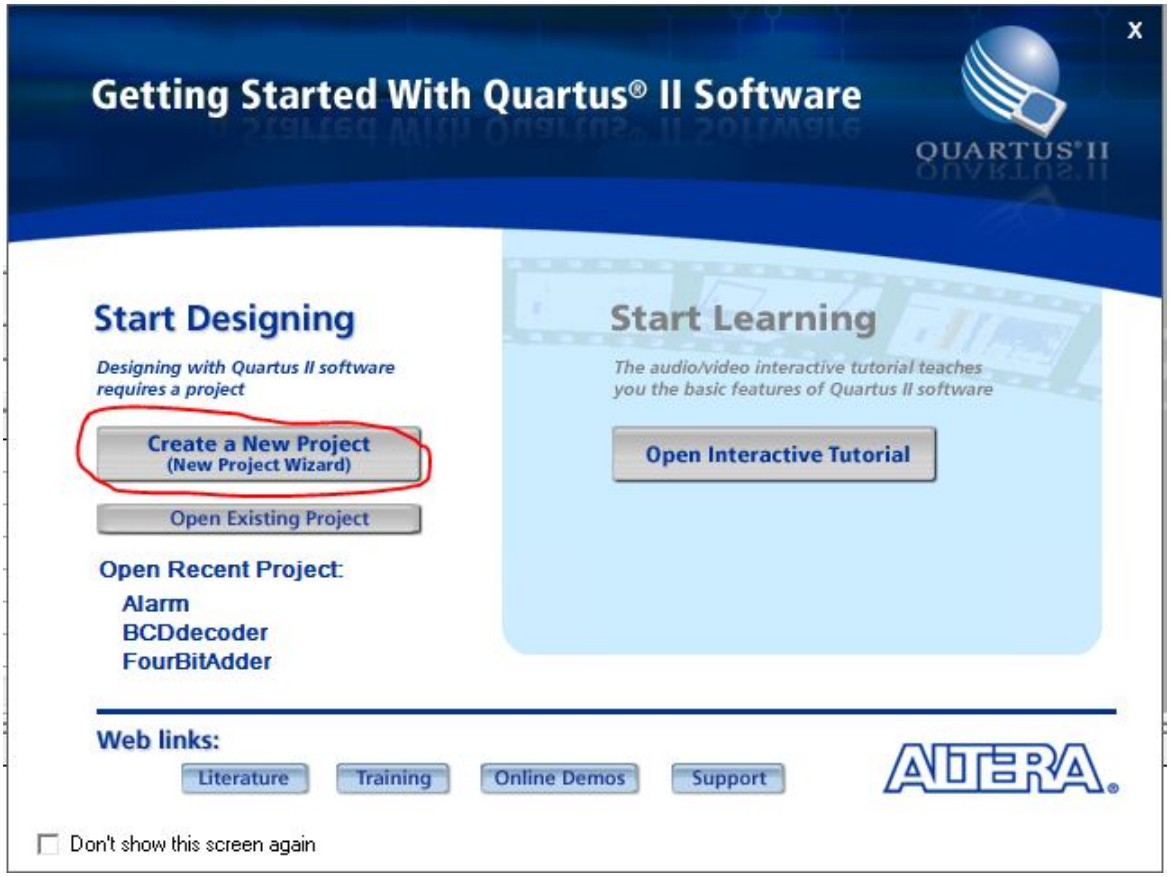

Figure 1: Click on “Create a New Project”.

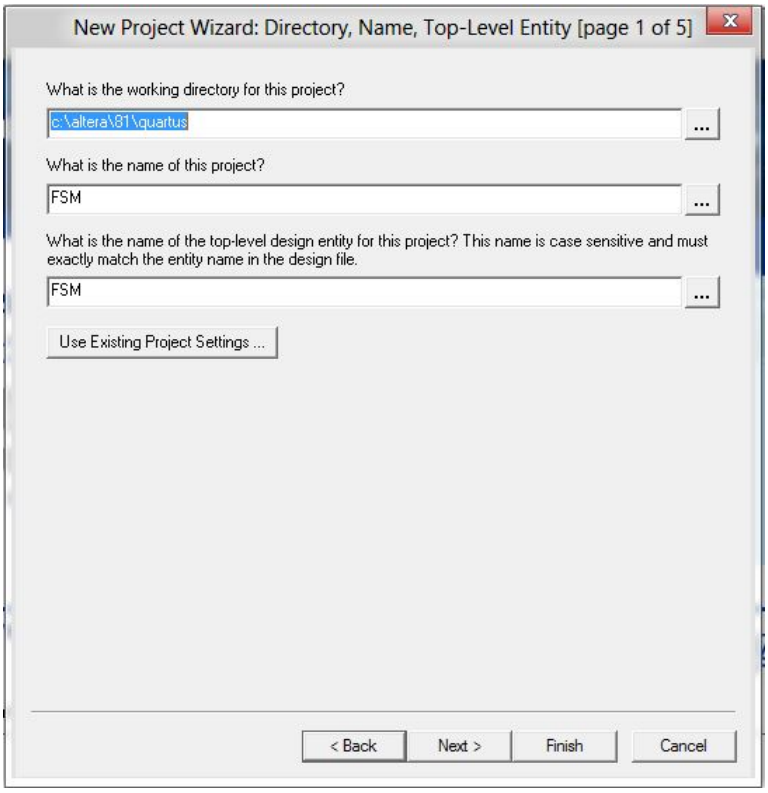

Figure 2: Choose the directory and name it without spaces and special characters.

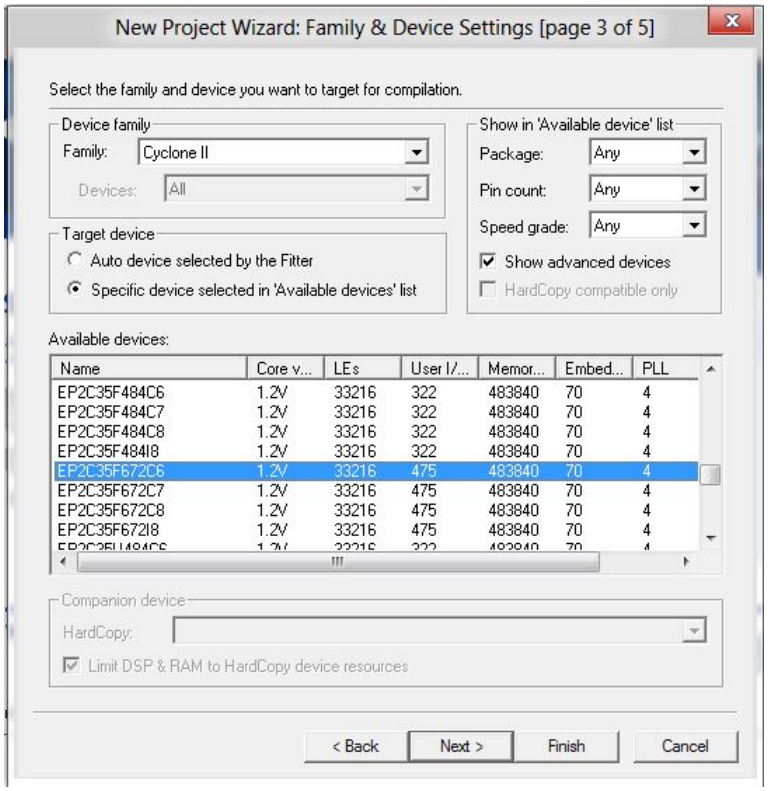

Figure 3: Choose the highlighted board and the Family should be “Cyclone II”.

5 Main Entity "FSM" Code

```
-- Basic libraries to include --
library ieee ;
use ieee.std_logic_1164.all;

-- The main Entity --
entity FSM is
port( n:    in std_logic;
  clock: in std_logic;
  reset: in std_logic;
  y:    out std_logic_vector(3 downto 0)
);
end FSM;

-- The architecture of FSM Entity --
architecture FSM_arch of FSM is
component clk_div IS
PORT( clock_50Mhz  : IN STD_LOGIC; -- The input to the clock divider is from the pin
     assignment and takes 50 MHz input
  clock_1MHz  : OUT STD_LOGIC; -- 50 MHz divided to give 1 MHz Output
  clock_100KHz : OUT STD_LOGIC; -- 50 MHz divided to give 100 KHz Output
  clock_10KHz  : OUT STD_LOGIC; -- 50 MHz divided to give 10 KHz Output
  clock_1KHz  : OUT STD_LOGIC; -- 50 MHz divided to give 1 KHz Output
  clock_100Hz  : OUT STD_LOGIC; -- 50 MHz divided to give 100 Hz Output
  clock_10Hz  : OUT STD_LOGIC; -- 50 MHz divided to give 10 Hz Output
  clock_1Hz  : OUT STD_LOGIC); -- 50 MHz divided to give 1 Hz Output, which is
      basically one cycle per second.
END component;
    type state_type is (idle, washing, soaping, cleaning, drying);  --Declaring all
       the 5 states
    signal next_state, current: state_type; --Declaring two wires to store the
       current and the next state
  signal clk1hz: std_lOGIC;
begin
  gate1: clk_div port map (clock_50Mhz=>clock, clock_1Hz=>clk1hz); -- Taking the 1Hz
      output so that every one second the state is changed
        state_reg: process(clk1hz, reset)
    begin
 if (reset='1') then -- If Reset is one, then go immediately to Idle State
            current <= idle;
 elsif (clk1hz'event and clk1hz='1') then
     current <= next_state;
 end if;
 end process;
  state_machine: process(current, n)
    begin

 case current is
   when idle => y <= "0000"; -- All outputs are OFF in Idle State
   if n='0' then
       next_state <= idle;
   elsif n ='1' then
       next_state <= washing;
   end if;
     when washing => y <= "1000"; --First LED Lights up to show that it is in the
        Washing State
   if n='0' then
```

```
51          next_state<=washing;
    elsif n='1' then
53          next_state <= soaping;
    end if;
55        when soaping => y <= "0100"; --Second LED Lights up to show that it is in the
              Soaping State
    if n='0' then
57          next_state <= soaping;
    elsif n='1' then
59          next_state <= cleaning;
    end if;
61        when cleaning => y <= "0010"; --Third LED Lights up to show that it is in the
              Cleaning State
    if n='0' then
63    next_state <= cleaning;
    elsif n='1' then
65   next_state <= drying;
    end if;
67
    when drying => y <= "0001"; --Fourth LED Lights up to show that it is in the
        Drying State
69    if n='0' then
          next_state <= drying;
71    elsif n='1' then
          next_state <= idle; --Go Back to Idle state Again
73    end if;
        when others =>
75    y <= "0000";
      next_state <= idle;
77  end case;
      end process;
79 end FSM_arch;
```

6 Entity "CLKDIV" Code

```
1  LIBRARY IEEE;
   USE  IEEE.STD_LOGIC_1164.all;
3  USE  IEEE.STD_LOGIC_ARITH.all;
   USE  IEEE.STD_LOGIC_UNSIGNED.all;
5  ENTITY clk_div IS
   PORT( clock_50Mhz  : IN STD_LOGIC;
7    clock_1MHz   : OUT STD_LOGIC;
     clock_100KHz : OUT STD_LOGIC;
9    clock_10KHz  : OUT STD_LOGIC;
     clock_1KHz   : OUT STD_LOGIC;
11   clock_100Hz  : OUT STD_LOGIC;
     clock_10Hz   : OUT STD_LOGIC;
13   clock_1Hz    : OUT STD_LOGIC);
   END clk_div;
15 ARCHITECTURE Behavior OF clk_div IS
    SIGNAL count_1Mhz          : STD_LOGIC_VECTOR(5 DOWNTO 0);
17  SIGNAL count_100Khz, count_10Khz, count_1Khz   : STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL count_100hz, count_10hz, count_1hz     : STD_LOGIC_VECTOR(2 DOWNTO 0);
19  SIGNAL  clock_1Mhz_int, clock_100Khz_int   : STD_LOGIC;
    SIGNAL clock_10Khz_int, clock_1Khz_int     : STD_LOGIC;
```

```
 SIGNAL clock_100hz_int, clock_10Hz_int    : STD_LOGIC;
 SIGNAL clock_1Hz_int         : STD_LOGIC;
BEGIN
 PROCESS
 BEGIN
-- Divide by 50
  WAIT UNTIL clock_50Mhz'EVENT and clock_50Mhz = '1';
   IF count_1Mhz < 49 THEN
    count_1Mhz <= count_1Mhz + 1;
   ELSE
    count_1Mhz <= "000000";
   END IF;
   IF count_1Mhz < 24 THEN
    clock_1Mhz_int <= '0';
   ELSE
    clock_1Mhz_int <= '1';
   END IF;
-- Ripple clocks are used in this code to save prescalar hardware
-- Sync all clock prescalar outputs back to master clock signal
   clock_1Mhz   <= clock_1Mhz_int;
   clock_100Khz  <= clock_100Khz_int;
   clock_10Khz  <= clock_10Khz_int;
   clock_1Khz   <= clock_1Khz_int;
   clock_100hz  <= clock_100hz_int;
   clock_10hz   <= clock_10hz_int;
   clock_1hz    <= clock_1hz_int;
 END PROCESS;
 -- Divide by 10
 PROCESS
 BEGIN
  WAIT UNTIL clock_1Mhz_int'EVENT and clock_1Mhz_int = '1';
   IF count_100Khz /= 4 THEN
    count_100Khz <= count_100Khz + 1;
   ELSE
    count_100khz <= "000";
    clock_100Khz_int <= NOT clock_100Khz_int;
   END IF;
 END PROCESS;
 -- Divide by 10
 PROCESS
 BEGIN
  WAIT UNTIL clock_100Khz_int'EVENT and clock_100Khz_int = '1';
   IF count_10Khz /= 4 THEN
    count_10Khz <= count_10Khz + 1;
   ELSE
    count_10khz <= "000";
    clock_10Khz_int <= NOT clock_10Khz_int;
   END IF;
 END PROCESS;
 -- Divide by 10
 PROCESS
 BEGIN
  WAIT UNTIL clock_10Khz_int'EVENT and clock_10Khz_int = '1';
   IF count_1Khz /= 4 THEN
    count_1Khz <= count_1Khz + 1;
   ELSE
    count_1khz <= "000";
    clock_1Khz_int <= NOT clock_1Khz_int;
   END IF;
```

```
 END PROCESS;
81 -- Divide by 10
 PROCESS
83 BEGIN
  WAIT UNTIL clock_1Khz_int'EVENT and clock_1Khz_int = '1';
85   IF count_100hz /= 4 THEN
      count_100hz <= count_100hz + 1;
87   ELSE
      count_100hz <= "000";
89    clock_100hz_int <= NOT clock_100hz_int;
    END IF;
91 END PROCESS;
 -- Divide by 10
93 PROCESS
 BEGIN
95  WAIT UNTIL clock_100hz_int'EVENT and clock_100hz_int = '1';
    IF count_10hz /= 4 THEN
97    count_10hz <= count_10hz + 1;
    ELSE
99    count_10hz <= "000";
      clock_10hz_int <= NOT clock_10hz_int;
101   END IF;
 END PROCESS;
103 -- Divide by 10
 PROCESS
105 BEGIN
  WAIT UNTIL clock_10hz_int'EVENT and clock_10hz_int = '1';
107   IF count_1hz /= 4 THEN
      count_1hz <= count_1hz + 1;
109   ELSE
      count_1hz <= "000";
111   clock_1hz_int <= NOT clock_1hz_int;
    END IF;
113 END PROCESS;
END Behavior;
```

7 Results and Discussions

- After compiling the code successful uploading and running of the Altera Board was achieved.
- Whenever the clock button was pressed, the state was changed to the next one and the next two LEDs lit up.

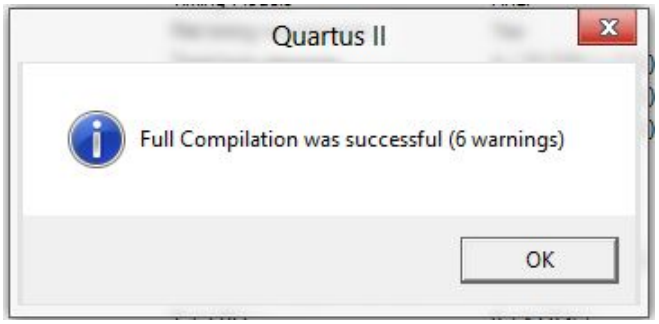

Figure 4: After writing the code, a successful compilation of the code was achieved.

Figure 5: LED for the first state is turned ON after 1 second.

Figure 6: LED for the second state is turned ON after 1 second

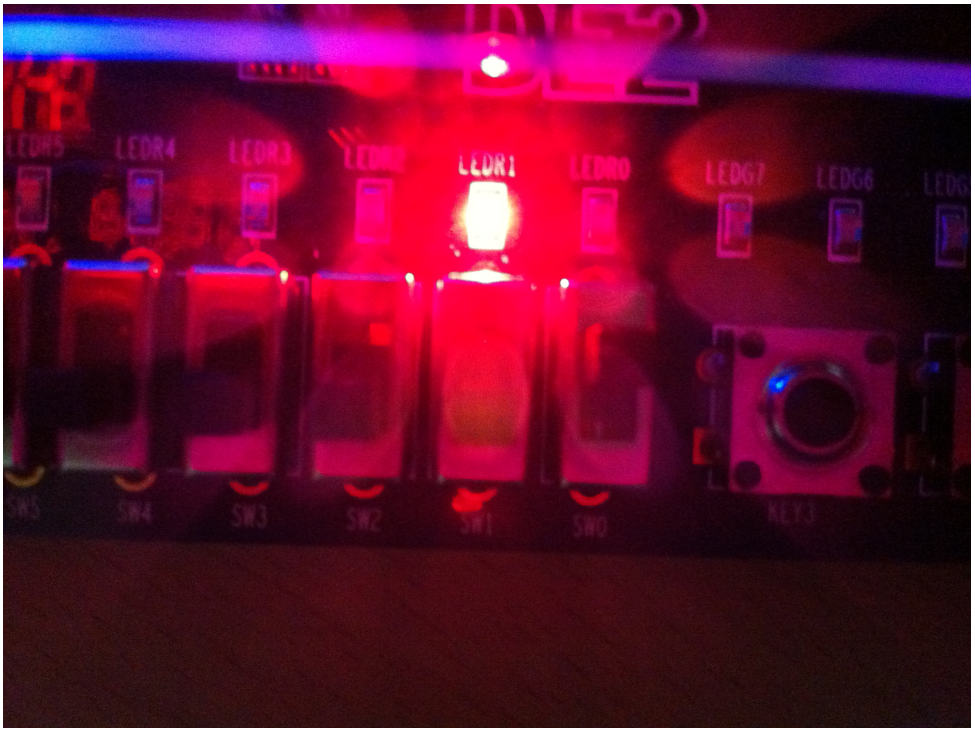

Figure 7: LED for the Third state is turned ON after 1 second.

Figure 8: LED for the Fourth state is turned ON after 1 second.

Figure 9: When the "RESET" button is pressed, the whole system comes to a halt and no LED is lit up as the system is in idle state where no LEDS should be ON.

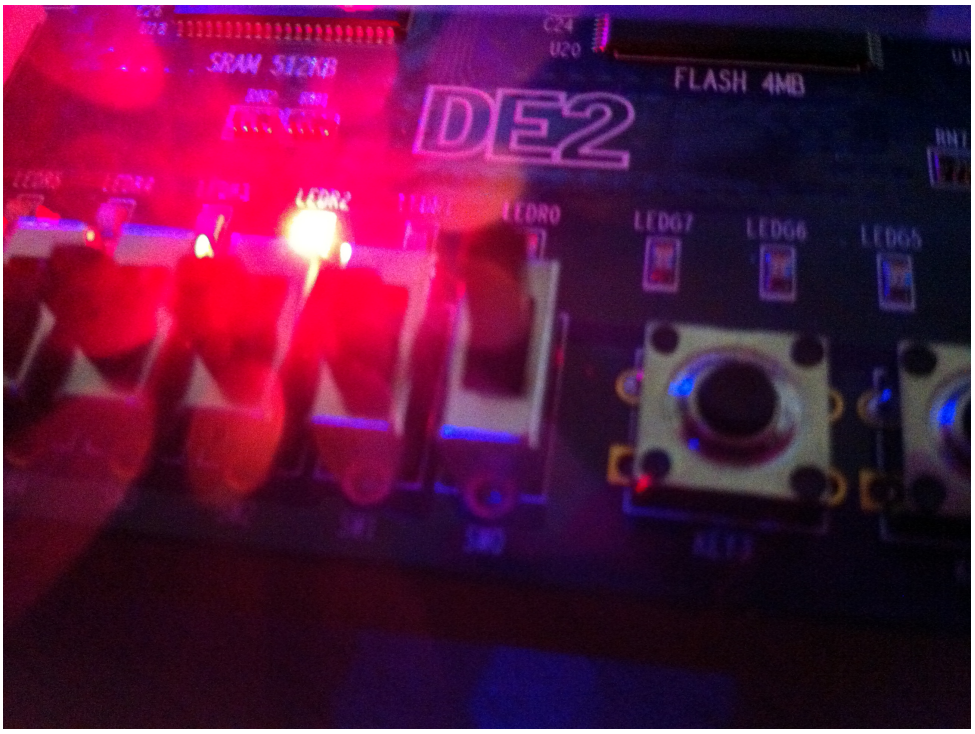

Figure 10: When the "SET" button is switched ON, the system continues to work with normal functionality.

8 Conclusion

- This was just a basic FSM, and it can be changed later to add or subtract states and make it a more complex design.
- Solving a problem using FSM is much easier than doing it combinationally.
- FSM allows us to think of the outputs and inputs and build them into a number of stages.

9 Team Dynamics

Report/Member	Weight/Grade	Obaidullah	Farooq	Mehdi Ismail
Abstract	20%	65%	15%	15%
Introduction	10%	0%	50%	50%
Procedure Part 1	10%	100%	0%	0%
Procedure Part 2	10%	0%	100%	0%
Procedure Part 3	10%	0%	0%	100%
Results Part 1	10%	100%	0%	0%
Results Part 2	10%	0%	100%	0%
Results Part 3	10%	0%	0%	100%
Conclusion	10%	0%	50%	50%
Claimed Contribution		**33%**	**33%**	**33%**
Contribution Validation Penalty		**0%**	**0%**	**0%**
Overall Contribution		**33%**	**33%**	**33%**
Overall Grade with Quality	**100%**	**100.0%**	**100.0%**	**100.0%**