Experiment 2 - Sequential Synthesis and FPGA Device Programming

Mohammad Amin Alinejad- 810199463 / Amirali Shahriary- 810100173

Abstract: In this experiment, we introduced to the concepts of state machines, designed a Serial Transmitter and synthesized in Quartus and implement it on an FPGA device afterward.

Keywords: serial transmitter, sequence detector, state machine, counter, simulation, synthesis, FPGA

I. EXPERIMENT

A. One Pulser

Due to FPGA's clock frequency that is 50KHz, its clock period would be 20ns. And it is much faster than what we need so we have to make a device that solve this.

So, we use one pulser as a solution, One pulser module provides a clock-enable input for the counter and one for the sequence detector. We use its signal which enables the OTHFSM for one clock when required. So, we use one of the push buttons on the FPGA to enable the Sequence Detector and the Counter for one clock after it is pushed.

Verilog description of the one pulser module can be seen below:

```
`timescale 1ns/1ns
module one_pulser(input clk_pb,clk,rst,output reg clk_en);
reg[1:0] ns,ps;
always@(ps, clk_pb) begin

                ns = 2'b00;
                case(ps)
                        2'b00 : ns = clk_pb ? 2'b00 : 2'b01;
                        2'b01 : ns = 2'b10;
                        2'b10 : ns = clk_pb ? 2'b00 : 2'b10;
                        default ns =  2'b00;
                endcase
end
assign clk_en = (ps == 2'b01) ? 1'b1 : 1'b0;
 always@(posedge clk , posedge rst) begin
        if(rst)
          ps <=2'b0;
        else
          ps <=ns;
        end
endmodule

```

Fig 1. One Pulser module Verilog description

B. Orthogonal State Machine

Orthogonal FSM consists of two part: Moore state machine and a 4-bit counter.

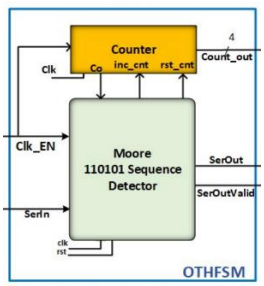


Fig 2. OTHFSM diagram [1]

1. Sequence Detector

Sequence detector is Moore machine that when the clock Enable push-button is pushed start to detect a 110101 sequence. It waits for the sequence on the (serIn), when the correct sequence received the (serOutValid) should be 1. The state diagram and the Verilog description of the detector:

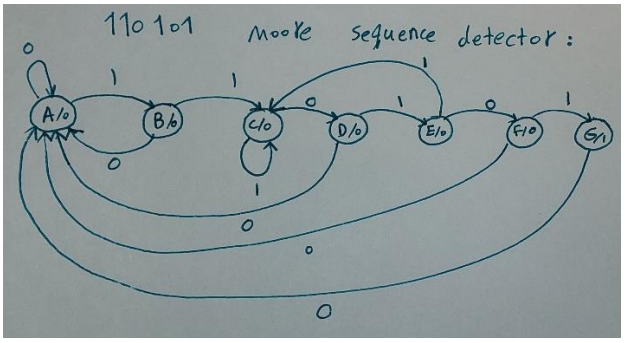


Fig 3. 110101 sequence detector SM

```
module Moore_detector(input clk_en, clk, rst, Ser_In, Co, output inc_cnt, rst_cnt, SerOutValid, Ser_Out);
        reg [2:0]ns , ps ;
        always @(ps,Ser_In)begin
                ns = 3'b000 ;
                case (ps)
                        3'b000 : ns = (Ser_In) ? 3'b001 : 3'b000 ;
                        3'b001 : ns = (Ser_In) ? 3'b010 : 3'b000 ;
                        3'b010 : ns = (Ser_In) ? 3'b010 : 3'b011 ;
                        3'b011 : ns = (Ser_In) ? 3'b100 : 3'b000 ;
                        3'b100 : ns = (Ser_In) ? 3'b010 : 3'b101 ;
                        3'b101 : ns = (Ser_In) ? 3'b110 : 3'b000 ;
                        3'b110 : ns = Co ? 3'b000 : 3'b110;
                        default ns = 3'b000;
                endcase
        end
        assign SerOutValid = (ps == 3'b110) ? 1'b1 : 1'b0 ;
        assign inc_cnt = (ps == 3'b110) ? 1'b1 : 1'b0 ;
        assign rst_cnt = (ps == 3'b101) ? 1'b1: 1'b0 ;
        assign Ser_Out = (Co==1'b0 && rst ==1'b0) ? Ser_In :1'b0;
        always @(posedge clk , posedge rst)begin
                if (rst)
                        ps <= 3'b000 ;
                else if(clk_en)
                        ps <= ns;
        end
endmodule
```

Fig 4. 110101 sequence detector Verilog description

2. Counter

The detector waits for the sequence and whenever the 110101 sequence is detected, For the next 10 clock cycles (serIn) will be transmitted to (serOut). To do this, instead of considering 10 empty state- we will need a 4-bit counter with a load signal (rst_cnt). After that, the counter will count for the next 10 consecutive clock cycles. We need to know when counting is finished. Therefore we have a Co output signal from the counter and as an input for the state machine. Note that during all these times the (serOutvalid) remains one. After 10 clock cycles (*serOutValid)* will insert and machine searches for next valid sequence.

```
module Counter(input clk, clk_en, rst_cnt, inc_cnt, output reg [3:0] cnt_out, output Co);
        always@(posedge clk, posedge rst_cnt) begin
                if(rst_cnt) cnt_out <= 3'b110;
                else if(clk_en) cnt_out <= (inc_cnt == 1) ? cnt_out + 1 : cnt_out;

        end
        assign Co = &{cnt_out, inc_cnt};

endmodule
```

Fig 5. Counter Verilog description

C. Seven Segment Display

To watch the counter's output, we use the seven-segment part of the FPGA. But first, we need to assign the appropriate value to each of the seven outputs of the SSD. Each seven segment receives a 4-bit input and displays the HEX value on its 7-bit output. So we use the table below.

Inputs				Segments							
A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	For display 0
0	0	0	1	0	1	1	0	0	0	0	For display 1
0	0	1	0	1	1	0	1	1	0	1	For display 2
0	0	1	1	1	1	1	1	0	0	1	For display 3
0	1	0	0	0	1	1	0	0	1	1	For display 4
0	1	0	1	1	0	1	1	0	1	1	For display 5
0	1	1	0	1	0	1	1	1	1	1	For display 6
0	1	1	1	1	1	1	0	0	0	0	For display 7
1	0	0	0	1	1	1	1	1	1	1	For display 8
1	0	0	1	1	1	1	1	0	1	1	For display 9
1	0	1	0	1	1	1	0	1	1	1	For display A
1	0	1	1	0	0	1	1	1	1	1	For display b
1	1	0	0	1	0	0	1	1	1	0	For display C
1	1	0	1	0	1	1	1	1	0	1	For display d
1	1	1	0	1	0	0	1	1	1	1	For display E
1	1	1	1	1	0	0	0	1	1	1	For display F

Table 1. Seven Segment corresponding outputs

```
`timescale 1ns/1ns
module Hex_display(input [3:0] bin_data, output reg [6:0] hex_data);
always@(bin_data) begin
    hex_data = 7'b0;
    case(bin_data)
        4'b0000: hex_data = 7'b1000000;
        4'b0001: hex_data = 7'b1111001;
        4'b0010: hex_data = 7'b0100100;
        4'b0011: hex_data = 7'b0110000;
        4'b0100: hex_data = 7'b0011001;
        4'b0101: hex_data = 7'b0010010;
        4'b0110: hex_data = 7'b0000010;
        4'b0111: hex_data = 7'b1111000;
        4'b1000: hex_data = 7'b0000000;
        4'b1001: hex_data = 7'b0010000;
        4'b1010: hex_data = 7'b0001000;
        4'b1011: hex_data = 7'b0000011;
        4'b1100: hex_data = 7'b1000110;
        4'b1101: hex_data = 7'b0100001;
        4'b1110: hex_data = 7'b0000110;
        4'b1111: hex_data = 7'b0001110;
    endcase
end
endmodule
```

Fig 6. HEX display Verilog description

D. Serial Transmitter

Now, to complete our Serial Transmitter, we have to connect the components that described before. The circuit will be completed by connecting the components described before. One Pulser's output should be connected to OTHFSM's and counter's (clk_en) and the counter's output should be connected to SSD as well.

As it was said, now we connect these three components via appropriate wires.

```
`timescale 1ns/1ns

module fpga(input clk, rst, clk_pb, Ser_In, output[6:0] SSD, output Ser_Out, SerOutValid);
wire clk_en, rst_cnt , inc_cnt, Co;
wire [3:0] cnt_out;
//reg [6:0] hex_data;
Counter cnt_inst(clk, clk_en, rst_cnt, inc_cnt,cnt_out, Co);
one_pulser one_inst(clk_pb,clk,rst,clk_en);
Moore_detector moore_inst(clk_en, clk, rst, Ser_In, Co, inc_cnt, rst_cnt, SerOutValid, Ser_Out);
Hex_display hex_inst(cnt_out, SSD);
endmodule
```

Fig 7. Serial Transmitter Verilog description

1.Simulation

To test the circuit, we provide a testbench. At first give a valid sequence to check detection and then random sequences. Because of the length of test bench code, we just can see a part of the code.

```
module fpga_tb();

        reg clk = 1'b0, rst = 1'b0, pb = 1'b0, se_in = 1'b0;

        wire [6:0] hex_out;

        wire ser_out,ser_out_val;

        fpga fpga_t(clk, rst, pb, se_in, hex_out, ser_out, ser_out_val);

        always #10 clk=~clk;

        always #50 pb=~pb;

        initial begin

                clk=0;

                #10;

                rst=1;

                #10;

                rst=0;

                se_in=1'b1;

                #50;

                #50;

                se_in=1'b1;

                #50;
```

Fig 8. Testbench

To simulate the effect of One Pulser (pressing the push button on the FPGA) we alternate pb signal exactly like the clk but with a lower frequency.

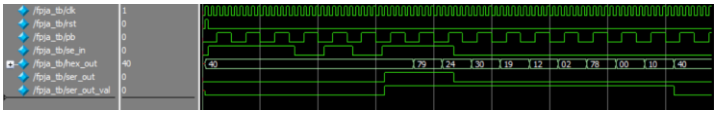


Fig 9. Serial transmitter simulation waveform

According to waveform when 110101 sequence is detected, *serOutValid* is asserted for 10 clock cycles (*pb*) and during this time *serOut* is same as *serIn*.

2.Synthesis

Synthesize the module in Quartus II. The FPGA code will be the top module. To map the circuit on device we need pin assignment using *Cyclone EP2C20F484C7* manual.

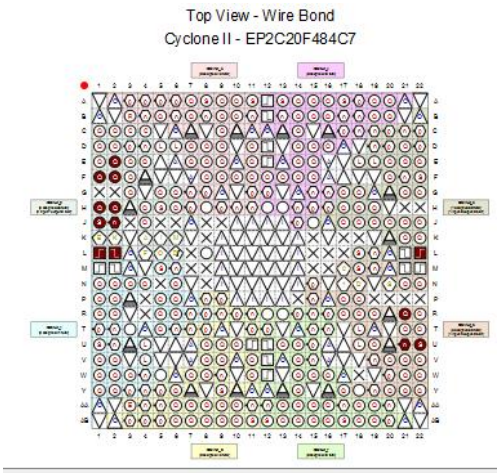
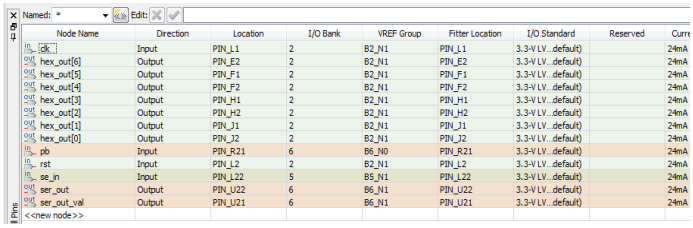


Fig 10. Pin assignment top view



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Curre
clk	Input	PIN_L1	2	B2_N1	PIN_L1	3.3-V LV...default)		24mA
hex_out[6]	Output	PIN_E2	2	B2_N1	PIN_E2	3.3-V LV...default)		24mA
hex_out[5]	Output	PIN_F1	2	B2_N1	PIN_F1	3.3-V LV...default)		24mA
hex_out[4]	Output	PIN_F2	2	B2_N1	PIN_F2	3.3-V LV...default)		24mA
hex_out[3]	Output	PIN_H1	2	B2_N1	PIN_H1	3.3-V LV...default)		24mA
hex_out[2]	Output	PIN_H2	2	B2_N1	PIN_H2	3.3-V LV...default)		24mA
hex_out[1]	Output	PIN_J1	2	B2_N1	PIN_J1	3.3-V LV...default)		24mA
hex_out[0]	Output	PIN_J2	2	B2_N1	PIN_J2	3.3-V LV...default)		24mA
pb	Input	PIN_R21	6	B6_N0	PIN_R21	3.3-V LV...default)		24mA
rst	Input	PIN_L2	2	B2_N1	PIN_L2	3.3-V LV...default)		24mA
se_in	Input	PIN_L22	5	B5_N1	PIN_L22	3.3-V LV...default)		24mA
ser_out	Output	PIN_U22	6	B6_N1	PIN_U22	3.3-V LV...default)		24mA
ser_out_val	Output	PIN_U21	6	B6_N1	PIN_U21	3.3-V LV...default)		24mA
<<new node>>								

Fig 11. Pin assignment

We use DE1_Manual to find each port and output location code and assign them by that manner.

After assigning the pins, we compile the project and run it on the FPGA. In the following pertaining images of the implementation and testing can be seen.

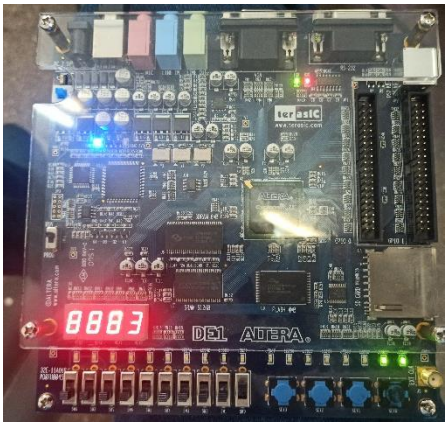


Fig 12. Serial transmitter circuit

In this case, Circuit detects the right sequence and count 3 times after that. As we can see *serOutValid* and *serOut* are asserted (second and first green LED respectively).in the next picture we can easily see that *serOut* follows *serIn*. Key 1 is our pulse clk and the first switch is our *serIn*. at last the first pin of seven segment shows counter output.

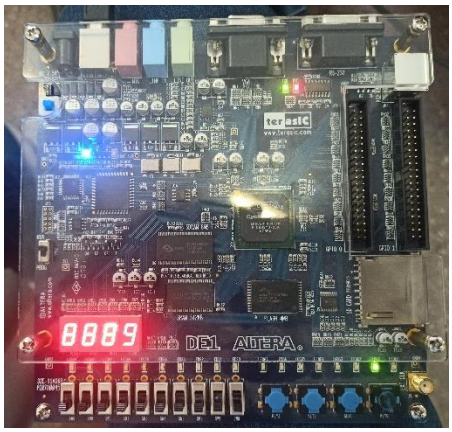


Fig 13. Serial transmitter circuit

After 6 clocks, we can see the 9 on seven segment output.

On the other hand, you can see that *serOut* light is off due to the input switch is 0.

2.Synthesis Results

In this part we can see some timing and planning summary and data analyses according to FPGA implementing.

	Name	ps.Wait	ps.F	ps.E	ps.D	ps.C	ps.B	ps.A
1	ps.A	0	0	0	0	0	0	0
2	ps.B	0	0	0	0	0	1	1
3	ps.C	0	0	0	0	1	0	1
4	ps.D	0	0	0	1	0	0	1
5	ps.E	0	0	1	0	0	0	1
6	ps.F	0	1	0	0	0	0	1
7	ps.Wait	1	0	0	0	0	0	1

Flow Summary	
Flow Status	Successful - Wed Mar 29 11:44:28 2023
Quartus II 32-bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	fpga
Top-level Entity Name	fpga
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	25 / 18,752 (< 1 %)
Total combinational functions	25 / 18,752 (< 1 %)
Dedicated logic registers	13 / 18,752 (< 1 %)
Total registers	13
Total pins	13 / 315 (4 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

Fig 14. Quartus Synthesis data results

II. CONCLUSIONS

In this experiment, we learned how to design a state machine by Hoffman model, convert it to Verilog code and implement it on an FPGA by testing a hex counter.

III. REFERENCES

[1] Katayoon Basharkhah and Zahra Jahanpeim and Zain Navabi, *Digital Logic Laboratory*, University of Tehran, Fall 1401.