Experiment 2 - Sequential Synthesis and FPGA Device Programming

Parsa Sattari - 810199436 / Iman Rasouli - 810199425

Abstract: In this experiment, a Serial Transmitter was designed and synthesized in Quartus and implemented on an FPGA afterward.

Keywords: state machine, counter, simulation, synthesis, FPGA

I. EXPERIMENT

A.One Pulser

Regarding that FPGA's clock frequency is 50KHz, its clock period would be 20ns. It is evident this is much faster than the time needed to make and notice changes.

Therefore, we need a control 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.

Also, the Verilog description of the One Pulser can be seen below.

```
module One_Pulser (input clk, ClkPB, rst, output Clk_EN);
    reg [1:0] ns,ps;

    parameter[1:0]
    A = 0, B = 1, C = 2;

    always@(ps, ClkPB) begin
        ns = A;
        case(ps)
            A: ns = ClkPB ? B : A;
            B: ns = C;
            C: ns = ClkPB ? C : A;
        endcase
    end

    assign Clk_EN = (ps == B)? 1'b1:1'b0;

    always@(posedge clk, posedge rst) begin
        if(rst)
            ps <= A;
        else
            ps <= ns;
    end
endmodule
```

Fig 1. One Pulser Verilog description

B. Orthogonal State Machine

Orthogonal state machine contains a sequence detector and a 4-bit counter.

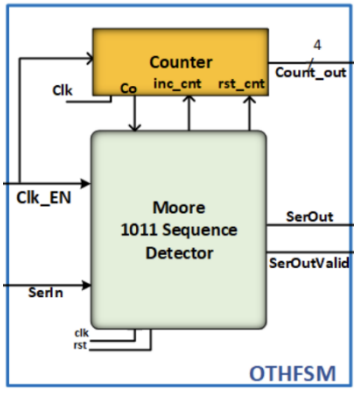

Fig 2. OTHFSM diagram [1]

1. Sequence Detector

Sequence detector will be a *Moore machine* which detects a 1011 sequence. When the correct sequence received the *serOutValid* should be asserted. Here is the state diagram:

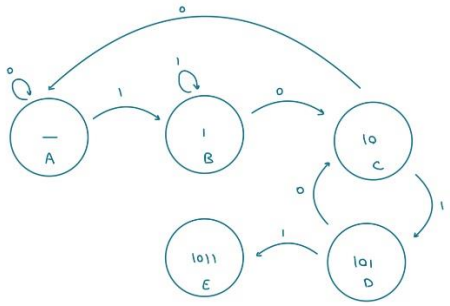

Fig 3. 1011 sequence detector SM

2. Counter

Whenever sequence detected, For the next 10 clock cycles *serIn* will be transmitted to *serOut*. To do this -instead of considering 10 empty states- we will use a 4-bit counter with a load signal (*rst_cnt*). Whenever this signal issued, Number 5 loads in counter's parallel input so it can count from 5 to 15 which is equal to 10 clock cycles. To have all together we need to know when counting is finished. Therefore we have a *Co* output signal from counter and as an input for the state machine.

After 10 clock cycles *serOutValid* will inserts and machine searches for next valid sequence.

Here is full state diagram of OTHFSM of serial transmitter:

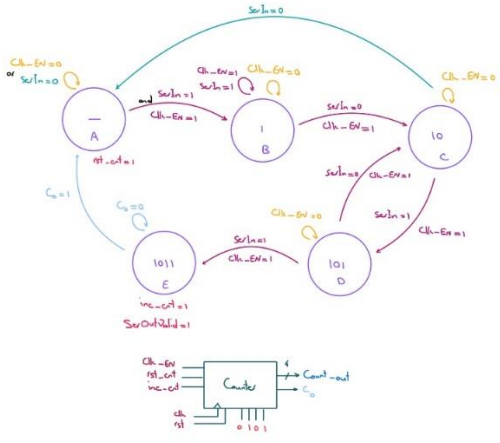

Fig 4. OTHFSM state diagram

```
module OTHFSM(input Clk_EN, SerIn, clk, rst, output reg SerOutValid, output SerOut, output reg [3:0] count
    reg [2:0] ps,ns;
    reg inc_cnt, rst_cnt;
    wire Co;

    parameter[2:0]
    A = 0, B = 1, C = 2, D = 3, E = 4;

    always@(ps, Clk_EN, SerIn, Co) begin
        ns = A;
        case(ps)
            A: ns = ~Clk_EN ? A : (SerIn ? B : A);
            B: ns = ~Clk_EN ? B : (SerIn ? B : C);
            C: ns = ~Clk_EN ? C : (SerIn ? D : A);
            D: ns = ~Clk_EN ? D : (SerIn ? E : C);
            E: ns = Co ? A : E;
        endcase
    end

    always@(ps) begin
        {rst_cnt, inc_cnt, SerOutValid} = 3'b000;
        case(ps)
            A: rst_cnt = 1;
            E: begin inc_cnt = 1; SerOutValid = 1; end
        endcase
    end

    always@(posedge clk, posedge rst) begin
        if (rst)
            ps <= A;
        else
            ps <= ns;
    end


    assign SerOut = (ps == E) ? SerIn : 1'bz;

    always@(posedge clk, posedge rst_cnt) begin
        if (rst_cnt)
            count <= 3'b101;
        else if (Clk_EN == 1 && inc_cnt == 1)
            count <= count + 1;
    end

    assign Co = &count;

endmodule
```

Fig 5. OTHFSM Verilog description

Note that this implementation only works when the *Clk_EN* signal has been activated.

C. Seven Segment Display

To be able to observe the counter's output, the seven-segment part of the FPGA can be used. But first, we need to assign the appropriate value to each of the seven outputs of the SSD. Accordingly, we make use of 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

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

Fig 6. SSD Verilog description

D. Serial Transmitter

Now, our Serial Transmitter will be completed by connecting the components described before. One Pulser's output should be connected to OTHFSM's and counter's *Clk_EN*. The Counter's output should be connected to SSD as well.

As it was said, we get instances of each of the three components and connect the appropriate wires to them.

```
module Serial_Transmitter(input ClkPB, clk, rst, SerIn, output [6:0] SSD, output SerOut, SerOutValid);
    wire Clk_EN;
    wire [3:0] count;

    One_Pulser B (clk, ClkPB, rst, Clk_EN);
    OTHFSM A (Clk_EN, SerIn, clk, rst, SerOutValid, SerOut, count);
    Seven_Segment C (count, 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.

```
`timescale 1ns/1ns

module TB();

    reg ClkPB, clk=0, rst=0, SerIn;
    wire [6:0] SSD;
    wire SerOut, SerOutValid;
    Serial_Transmitter CUT(ClkPB, clk, rst, SerIn, SSD, SerOut, SerOutValid);

    always #5 clk = ~clk;

    initial begin
    SerIn = 1'b0;
    #7 ClkPB = 1;
    #10 ClkPB = 0;

    #10 SerIn = 1'b1;
    #10 ClkPB = 1;
    #10 ClkPB = 0;

    #10 SerIn = 1'b0;
    #10 ClkPB = 1;
    #10 ClkPB = 0;

    #10 SerIn = 1'b1;
    #10 ClkPB = 1;
    #10 ClkPB = 0;

    #10 SerIn = 1'b1;
    #10 ClkPB = 1;
    #10 ClkPB = 0;

    #10 SerIn = 1'b1;
    #10 ClkPB = 1;
    #10 ClkPB = 0;

    repeat (15) begin
        #10 SerIn = $random;
        #10 ClkPB = 1;
        #10 ClkPB = 0;
    end
    #100 $stop;

    end

endmodule
```

Fig 8. Testbench

To simulate the effect of One_Pulser (pressing the push button on the FPGA) we alternate *ClkPB* signal exactly like the clock but with a lower frequency.

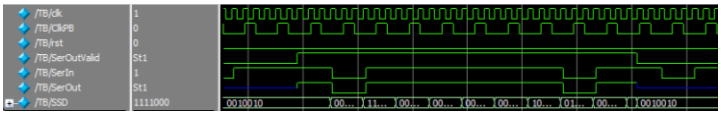

Fig 9. Serial transmitter simulation waveform

According to waveform when 1011 sequence detects *serOutValid* is asserted for 10 clock cycles (*ClkPB*) and during this time *serOut* follows *serIn*.

2.Synthesis

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

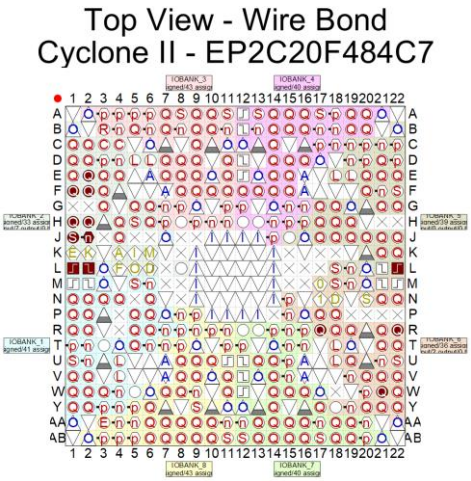

Fig 10. Pin assignment top view

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
ClkPB	Input	PIN_R22	6	B6_N0	PIN_R22	3.3-V LVTTL (default)		24mA (default)	
SSD[6]	Output	PIN_E2	2	B2_N1	PIN_E2	3.3-V LVTTL (default)		24mA (default)	
SSD[5]	Output	PIN_F1	2	B2_N1	PIN_F1	3.3-V LVTTL (default)		24mA (default)	
SSD[4]	Output	PIN_F2	2	B2_N1	PIN_F2	3.3-V LVTTL (default)		24mA (default)	
SSD[3]	Output	PIN_H1	2	B2_N1	PIN_H1	3.3-V LVTTL (default)		24mA (default)	
SSD[2]	Output	PIN_H2	2	B2_N1	PIN_H2	3.3-V LVTTL (default)		24mA (default)	
SSD[1]	Output	PIN_J1	2	B2_N1	PIN_J1	3.3-V LVTTL (default)		24mA (default)	
SSD[0]	Output	PIN_J2	2	B2_N1	PIN_J2	3.3-V LVTTL (default)		24mA (default)	
SerIn	Input	PIN_L2	2	B2_N1	PIN_L2	3.3-V LVTTL (default)		24mA (default)	
SerOut	Output	PIN_R17	6	B6_N1	PIN_R17	3.3-V LVTTL (default)		24mA (default)	
SerOutValid	Output	PIN_W21	6	B6_N1	PIN_W21	3.3-V LVTTL (default)		24mA (default)	
clk	Input	PIN_L1	2	B2_N1	PIN_L1	3.3-V LVTTL (default)		24mA (default)	
rst	Input	PIN_L22	5	B5_N1	PIN_L22	3.3-V LVTTL (default)		24mA (default)	

Fig 11. Pin assignment

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.

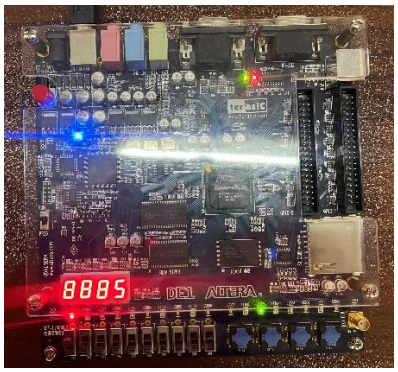

Fig 12. Serial transmitter circuit

In this case, Circuit detects the right sequence and as we see *serOutValid* is asserted (Green LED) and *serOut* (Red LED) follows *serIn* .Counter has been reset to “5” to count 10 cycles (First pin of seven segment shows counter output).

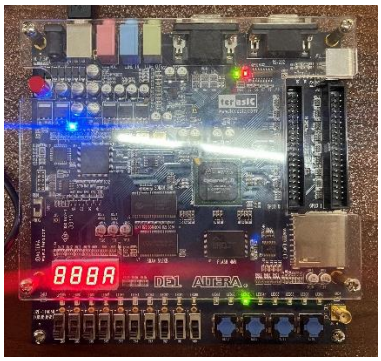

Fig 13. Serial transmitter circuit after 5 clock cycles

After 5 clock cycles, Counter output is “A” which is hexadecimal and equals to 10.

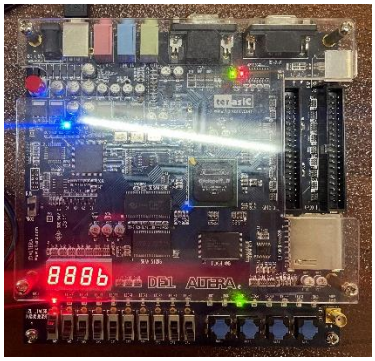

Fig 14. Serial transmitter circuit after another clock cycle

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.

III. References

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