# Experiment 1 - Sequential Synthesis and FPGA Programming

Shahriar Attar,
*810100186*,
attarshahriar@gmail.com

Elahe K. Nadrabadi,
*810100132*,
elahekhodavrdi@gmail.com

***Abstract*— This document is a report for experiment #2 of Digital Logic Design Laboratory at ECE department, University of Tehran. The purpose of this experiment is to introduce the concepts of state machines and also getting familiar with FPGA devices and implementation .**

***Keywords*— Serial Transmitter, State Machine, One-Pulser, Orthogonal Finite State Machine, Seven Segment Display**

## I. Introduction

This experiment mostly tries to introduce the concepts of state machines and how to implement orthogonal finite state machines. Then we saw a practical use of that OTHFSM and other components that make that experiment doable by us, such as one-pulser and seven segment display.

## II. Serial Transmitter

We will search for 110101 sequence on the *serIn* input and when we detect it the *serOutValid* is asserted and we begin transmitting its *serIn* on its *serOut* for the next 10 clock cycles. After that the circuit returns to the state in which we search for the sequence. The overall design is shown in Fig. 1.

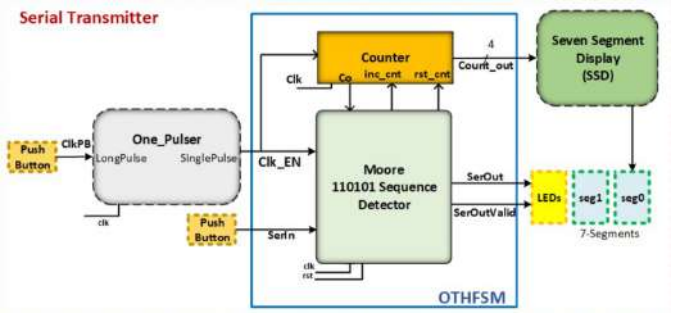


Fig. 1 overall design of serial transmitter

### A. One-Pulser

This module provides a *Clk_EN* input for the counter and the sequence detector part. This common input *Clk_EN* is used for controlling the clock when the circuit is implemented on an FPGA board. The one-pulser connects to a push-button on your board (*ClkPB*) and when pressed it creates a single pulse that is synchronized with the system clock. Its state diagram is shown in Fig. 2. The main idea behind this module is that since humans are not as fast to change input in every clock toggle (since they usually have a duration of about few nanoseconds) we need a module to determine when we want to enable the clock, and since we don't want to interfere with timing, it has to be synchronized with the system clock. In Fig. 2 the *lp* represents the *ClkPB* input in overall design, and the sp is the *Clk_EN* output which we will assert once.

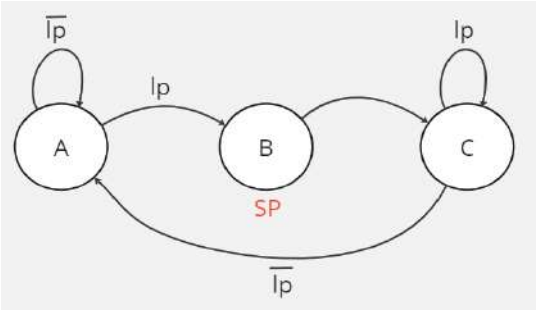


Fig. 2 finite state machine diagram of one-pulser

1) *Verilog Description:* : First we implement the module in Verilog. You can see the code in Fig. 3.

```
`define A 2'b00
`define B 2'b01
`define C 2'b10

module one_pulser(clk, rst, lp, sp);

    input clk, rst, lp;
    output sp;

    reg [1:0] ps, ns;

    always @(ps or lp) begin
        case (ps)
            `A: ns = lp ? `B : `A;
            `B: ns = `C;
            `C: ns = lp ? `C : `A;
            default: ns = `A;
        endcase
    end

    assign sp = (ps == `B);

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

endmodule
```

Fig. 3 verilog implementation of one-pulser

2) *Testing the design*: Then we wrote a test bench to check our module. testbench code is shown in Fig. 4 and ModelSim output is shown in Fig. 5.

```
module onepulser_tb();
    reg clk, rst, clkPB;
    wire Clk_EN;

    one_pulser pulser(
        .clk(clk), .rst(rst), .lp(clkPB), .sp(Clk_EN)
    );

    always #5 clk = ~clk;
    always #50 clkPB = ~clkPB;

    initial begin
        clk = 1'b0;
        clkPB = 1'b0;
        rst = 1'b1;
        #10 rst = 1'b0;
    end
endmodule
```

Fig. 4 testbench code for one-pulser module

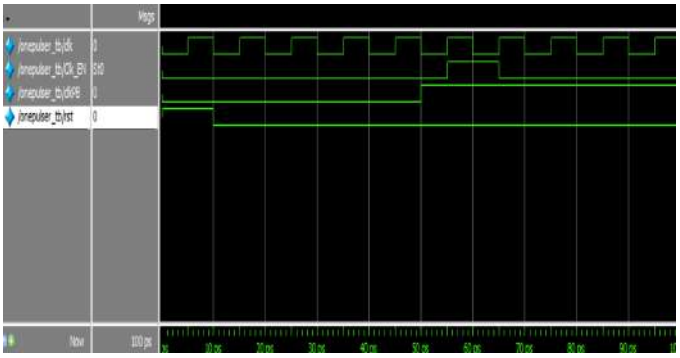

Fig. 5 ModelSim output for one-pulser testbench

## B. Orthogonal Finite State Machine

It consists of a Moore state machine and a counter. The Moore state machine is a 110101 detector whose state diagram can be seen in Fig. 6, and the counter is 4-bit but the initial value for it is 6 so instead of 16 it counts 10 times and then the Co signal is issued. When *Clk_EN* is pushed the Moore sequence detector checks its input and decides which state to go to. The sequence detector waits for the sequence of 110101 on its *serIn* input and when this is received, the *serOutValid* output becomes one and we transmit every input on *serIn* for the next 10 consecutive to the *serOut*, during which *serOutValid* should remain one. Note that when *serOutValid* is zero we put z on *serOut* to avoid any misunderstanding.

1) *State diagram*: You can see the state diagram of 110101 sequence Moore machine. All transitions are based on *serIn* other than the ones explicitly written.

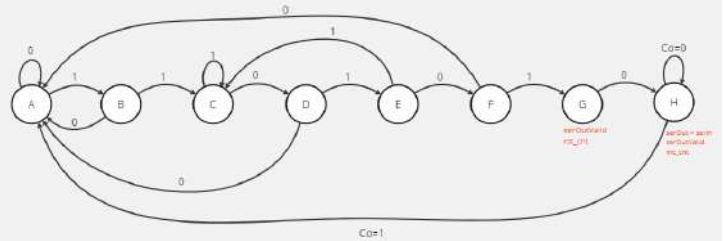


Fig .6 state diagram of 110101 sequence Moore machine

2) *Write top-level module for OTHFSM*: It consists of two main parts that are detector and counter, that each Verilog description is shown in Fig. 7 and Fig. 8 respectively. The top-level module implementation is also available on Fig. 9.

```
module detector(clk, Clk_EN, rst, serIn, Co,
                serOut, serOutValid, inc_cnt, rst_cnt);

    input clk, Clk_EN, rst, serIn, Co;
    output reg serOut, serOutValid, inc_cnt, rst_cnt;

    parameter A = 3'b000, B = 3'b001,
              C = 3'b010, D = 3'b011, E = 3'b100,
              F = 3'b101, G = 3'b110, H = 3'b111;

    reg [2:0] pstate = A;
    reg [2:0] nstate = A;

    always @(serIn or Co or Clk_EN) begin

        case (pstate)
            A : nstate <= serIn ? B : A;
            B : nstate <= serIn ? C : A;
            C : nstate <= ~serIn ? D : C;
            D : nstate <= ~serIn ? A : E;
            E : nstate <= ~serIn ?  F : C;
            F : nstate <= serIn ? G : A;
            G : nstate <= H;
            H : nstate <= Co ? A : H;
        endcase
    end

    always @(pstate) begin
        {serOutValid, inc_cnt, rst_cnt} <= 3'b000;
        case (pstate)
            G : {serOutValid, rst_cnt} = 2'b11;
            H : {serOutValid, inc_cnt} = 2'b1;
        endcase
    end

    always @(posedge clk, posedge rst) begin
        if (rst)
            pstate <= A;
        else if (Clk_EN)
            pstate <= nstate;
    end

    assign serOut = (pstate == H)? serIn : 1'bz;

endmodule
```

Fig.7 sequence detector implementation in Verilog

```verilog
module counter (clk, clk_en, rst_cnt,
                inc_cnt, co, cnt_out);

    input clk, clk_en, rst_cnt, inc_cnt;

    output co;
    output reg [3:0] cnt_out;

    always @(posedge clk) begin
        if (rst_cnt)
            cnt_out <= 4'd6;
        else if(clk_en && inc_cnt)
            cnt_out = cnt_out + 1;
    end

    assign co = &cnt_out;

endmodule
```

Fig. 8 counter implementation in Verilog

```verilog
module OTHFSM(clk, rst, clk_en, serIn,
              serOut, serOutValid, cnt_out);

    input clk, rst, clk_en, serIn;
    output serOut, serOutValid;
    output [3:0] cnt_out;

    wire co, inc_cnt, rst_cnt;

    detector sequence_detector(
        clk, rst, clk_en, serIn, co,
        serOut, serOutValid, inc_cnt, rst_cnt
    );

    counter cntr(
        clk, clk_en, rst_cnt, inc_cnt, co, cnt_out
    );

endmodule
```

Fig. 9 top-level module in Verilog

3) *Testing the design:* The designed testbench as depicted in Fig. 11 will show the correctness of the design. ModelSim output is demonstrated in Fig. 10.

As you can see when we have detected the sequence for 10 clock cycles *serOutValid* has become one and we transmit data from *serIn* to *serOut.*

After that the *serOutValid* has gone back to zero and the value on *serOut* has been changed to the high-impedance (i.e. z value).

During each clock, counter will increase one time till we reach 15 in which our *Co* will become one as it is the result of logical-and of all bits of *cnt_out*.

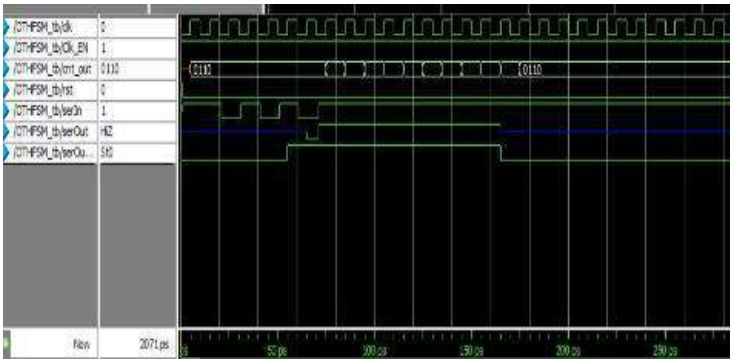

Fig. 10 ModelSim output for OTHFSM testbench

```verilog
module OTHFSM_tb();
    reg clk, Clk_EN, serIn, rst;
    wire serOut, serOutValid;
    wire [3:0] cnt_out;

    OTHFSM OTHFSM_instance(
        .clk(clk), .rst(rst), .clk_en(Clk_EN),
        .serIn(serIn), .serOut(serOut),
        .serOutValid(serOutValid), .cnt_out(cnt_out)
    );

    always #5 clk = ~clk;

    initial begin
        clk = 1'b0;
        rst = 1'b1;
        Clk_EN = 1'b1;
        #1 rst = 1'b0;
        serIn = 1'b1;
        #10 serIn = 1'b1;
        #10 serIn = 1'b0;
        #10 serIn = 1'b1;
        #10 serIn = 1'b0;
        #10 serIn = 1'b1;
        #10 serIn = 1'b0;
        #10 serIn = 1'b1;
        #2000 $stop;
    end
endmodule
```

Fig. 11 OTHFSM testbench written in Verilog

## C. *Seven Segment Display*

It's a simple module used for displaying the counter output on the seven segments of your FPGA board. Each seven segment receives a 4-bit input and displays the HEX value on its 7-bit output. Verilog code is attached in Fig. 12.

```verilog
`define ZERO 7'b100_0000
`define ONE 7'b111_1001
`define TWO 7'b010_0100
`define THREE 7'b011_0000
`define FOUR 7'b001_1001
`define FIVE 7'b001_0110
`define SIX 7'b000_0010
`define SEVEN 7'b111_1000
`define EIGHT 7'b000_0000
`define NINE 7'b001_0000
`define TEN 7'b000_1000
`define ELEVEN 7'b000_0011
`define TWELEVE 7'b100_0110
`define THIRTEEN 7'b010_0001
`define FOURTEEN 7'b000_0110
`define FIFTEEN 7'b000_1110

module hex_display(d_in, d_out);
    input [3:0] d_in;
    output reg [6:0] d_out;

    always @(d_in) begin
        case (d_in)
            4'd0:    d_out = `ZERO;
            4'd1:    d_out = `ONE;
            4'd2:    d_out = `TWO;
            4'd3:    d_out = `THREE;
            4'd4:    d_out = `FOUR;
            4'd5:    d_out = `FIVE;
            4'd6:    d_out = `SIX;
            4'd7:    d_out =  `SEVEN;
            4'd8:    d_out = `EIGHT;
            4'd9:    d_out = `NINE;
            4'd10:   d_out = `TEN;
            4'd11:   d_out = `ELEVEN;
            4'd12:   d_out = `TWELEVE;
            4'd13:   d_out = `THIRTEEN;
            4'd14:   d_out = `FOURTEEN;
            4'd15:   d_out = `FIFTEEN;
            default: d_out = `ZERO;
        endcase
    end
endmodule
```

Fig. 12 seven segment display code in Verilog

## III. Design Synthesis and FPGA Programming

A. *Serial Transmitter Implementation:* In this part we will use DE1 development board for our design implementation. The top-level Verilog module of the whole serial transmitter is shown in Fig. 13. Then you can follow the step by step guide to synthesise the design.

```verilog
module serial_transmitter(clk, rst, lp, serIn,
                          serOut, serOutValid, hex_out);

    input clk, rst, lp, serIn;
    output serOut, serOutValid;
    output [6:0] hex_out;
    wire clk_en;

    wire [3:0] cnt_out;

    one_pulser op (
        clk, rst, lp, clk_en
    );
    OTHFSM seqwc (
        clk, rst, clk_en, serIn,
        serOut, serOutValid, cnt_out
    );
    hex_display hd (
        cnt_out, hex_out
    );
endmodule
```

Fig. 13 top-level module implementation in Verilog language

1) *Make a Quartus Project.*
2) *Add the top-level Verilog codes to your project.*
3) *Connect the main FPGA clock to the clock input of your circuit:* for this you can check the DE1 manual and choose your own desired clock but we chose 50MHz that according to the manual is PIN_L1.
4) *Connect a push-button to the serIn of the serial transmitter circuit, another push-button to the input of the one-pulser circuit:* you can use any of the following list for inputs: [PIN_L22, PIN_L21, PIN_M22, PIN_V12, PIN_W12, PIN_U12, PIN_U11, PIN_M2, PIN_M1, PIN_L2]. But obviously the buttons will differ when you use different inputs. The given list is the order from right to left. The general design of DE1 development board is shown below in Fig. 14 and Fig. 15.

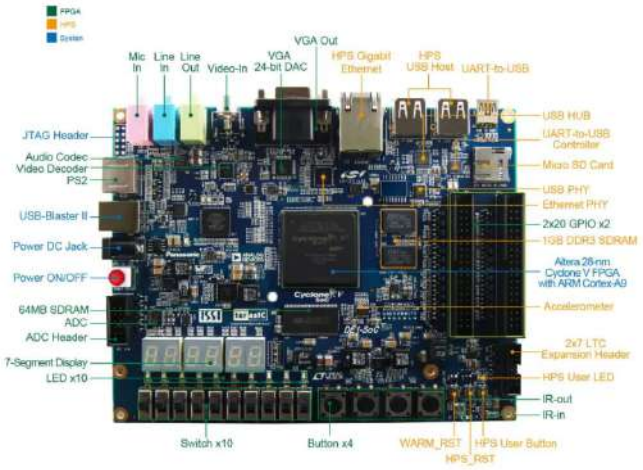


Fig. 14 general design of DE1 development board

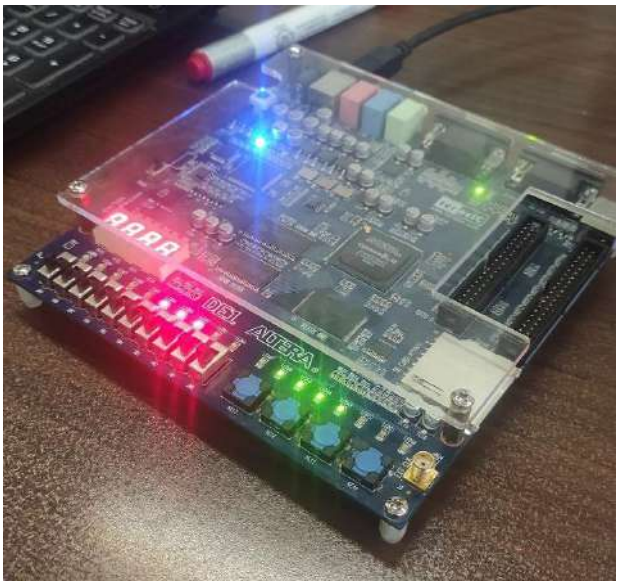

Fig. 15 general design of DE1 development board

5) *Perform the synthesis of your design.*
6) *Program the Cyclone II device.*
7) *For the output display, use an output LED for displaying the serOutValid and one LED for serOut. Use one seven segment for displaying the counter output:* You have two types of LED, namely RED and GREEN. And also you can arbitrarily choose one of the seven segments for displaying counter output in hexadecimal.
8) *Applying a serial input*: push the push-button corresponding to the serIn according to the value you want on the input, and then press the ClkPB push button once.

## IV. Conclusions

The general ideas discussed in this experiment were:

A. Finite State Machines and Orthogonal FSMs
B. One-Pulser
C. Seven Segment Display
D. Design Synthesis and FPGA Programming


## Acknowledgment

This report was prepared by Elahe K. Nadrabadi and Shahriar Attar for 3,4 sessions of DLD Laboratory at ECE department in 1401-1402 spring semester.


## References


[1] Introduction to the finite state machines (FSM) [Online]. Available: https://www.inf.ed.ac.uk/teaching/courses/inf1/cl/notes/Comp1.pdf

[2] Altera DE1 board [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=83