Experiment 2 - Sequential Synthesis and FPGA Programming

Student Name : Shahzad Momayez , Mohammad Amanlou
Student ID : 810100272 , 810100084

***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 get familiar with FPGA programming and program it to detect 110101.**

***Keywords*— One pulser, FPGA, Synthesis, FSM, Seven Segment display, Serial Transmitter.**

I. Introduction

The first goal of this experiment is to introduce the concepts of state machines that are mostly used for controllers. The second goal is to get familiar with FPGA devices and implementation.

II. Design

A. Serial Transmitter:
serial transmitter circuit searches on its serIn input for a start sequence of 110101 to begin transmitting its serIn on its serOut. When the start sequence is received, the serOutvalid is asserted and for the next 10 clock cycles whatever appears on serIn will be transmitted on serOut. After the entire 10 bits are transmitted, the circuit returns to the state where search for the start sequence begins again. The initial state of this circuit is where it searches for the start sequence.

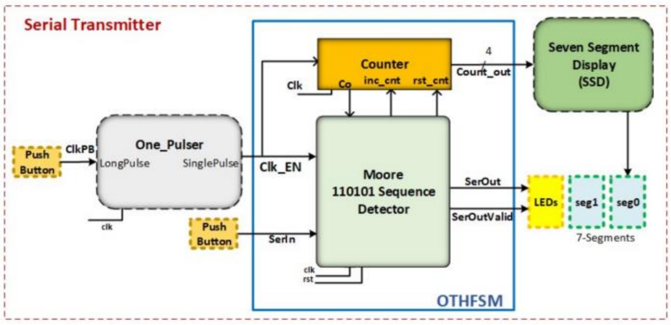


Fig1.serial transmitter

From the definition of serial transmitter, we know that The transmitter will insert sequentially the **D0**, **D1**, **D2** and **D3** values on the line **SER**, after the transmission of the start bit. One stop bit is then generated, simply maintaining the **SER** line at **0** for the duration of a clock cycle. The output **RDY** is activated only when the transmitter is in the idle state (i.e, when it is not transmitting, waiting for a low to high transition on the input **GO**).

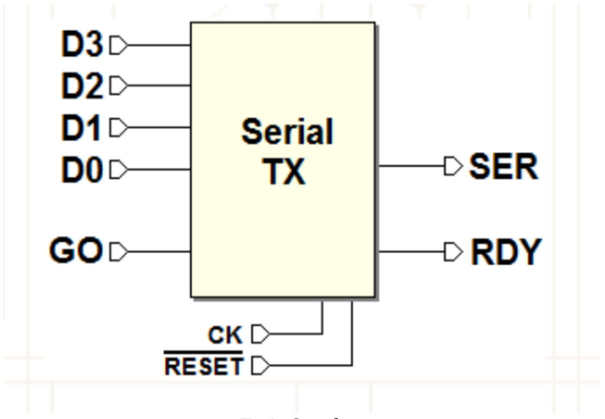


Fig2. Serial transmitter

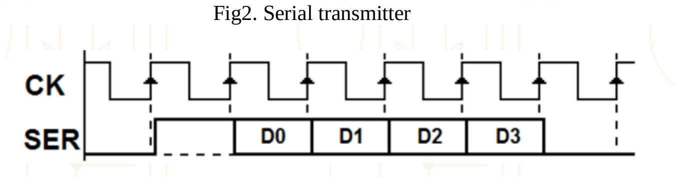
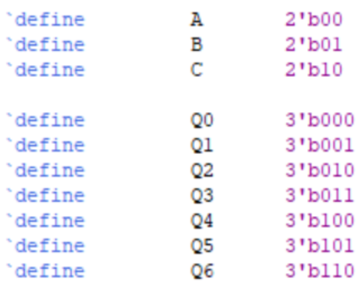


Fig3. Serial transmitter clock cycle

In order to see the result on our FPGA, we need to add an extra signal to our Sequential detector; a CLK_EN signal. This signal is used in along our CLK signal for our components when changing state. CLK_EN is generated from a circuit we'll be naming "One_Pulser". Our overall design consists of 4 components:

+ Sequence Detector
+ One_Pulser
+Counter
+Hex_Display

First of all we define some of the states and we use them in the following sections of the project.



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

`define         Q0      3'b000
`define         Q1      3'b001
`define         Q2      3'b010
`define         Q3      3'b011
`define         Q4      3'b100
`define         Q5      3'b101
`define         Q6      3'b110
```

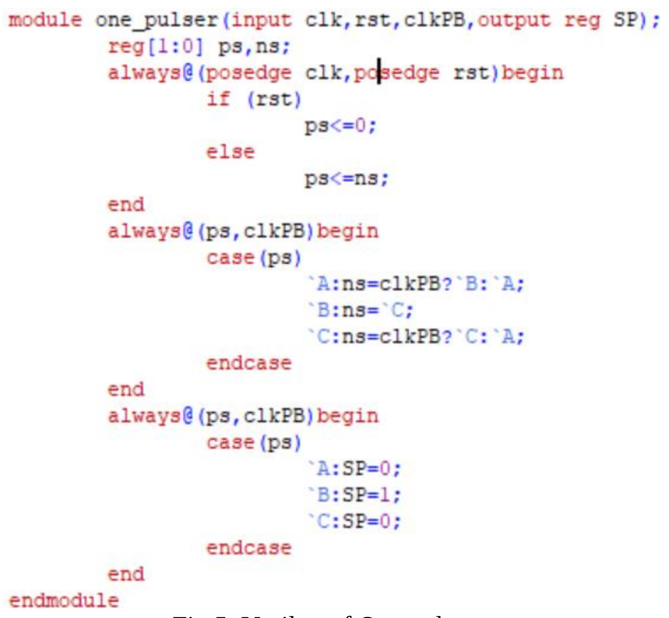
Fig4. Definition of states

B. One_pulser:

The overall design of this experiment is shown in figure 1. As shown three main components are needed for this design. The Onepulser module provides a clock-enable input for the counter and one for the sequence detector part. This common input (clkEn) 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. The output of this circuit connects to the clock enable input (clkEn) of the sequence detector and the counter.

One pulser works like clock enable to our circuit. We use one pulser due to the fact that clock is too fast. We use one pulser so we can control that the clock is enable or not.

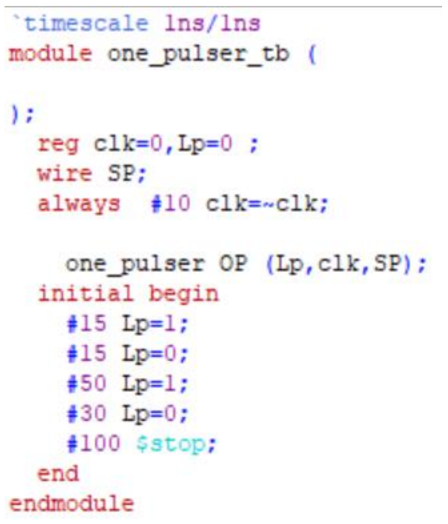
1)



```
module one_pulser(input clk,rst,clkPB,output reg SP);
        reg[1:0] ps,ns;
        always@(posedge clk,posedge rst)begin
                if (rst)
                        ps<=0;
                else
                        ps<=ns;
        end
        always@(ps,clkPB)begin
                case(ps)
                        `A:ns=clkPB?`B:`A;
                        `B:ns=`C;
                        `C:ns=clkPB?`C:`A;
                endcase
        end
        always@(ps,clkPB)begin
                case(ps)
                        `A:SP=0;
                        `B:SP=1;
                        `C:SP=0;
                endcase
        end
endmodule
```

Fig 5. Verilog of One pulser

2)



```
`timescale 1ns/1ns
module one_pulser_tb (

);
  reg clk=0,Lp=0 ;
  wire SP;
  always  #10 clk=~clk;

    one_pulser OP (Lp,clk,SP);
  initial begin
    #15 Lp=1;
    #15 Lp=0;
    #50 Lp=1;
    #30 Lp=0;
    #100 $stop;
  end
endmodule
```

Fig 6. testbench of One pulser

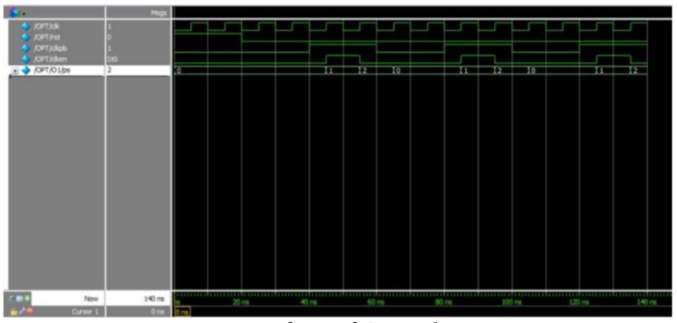


Fig 7. Waveform of One pulser

C.Orthogonal FSM:

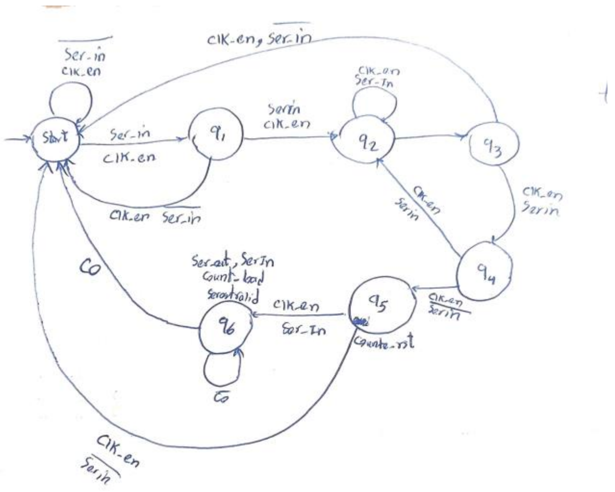
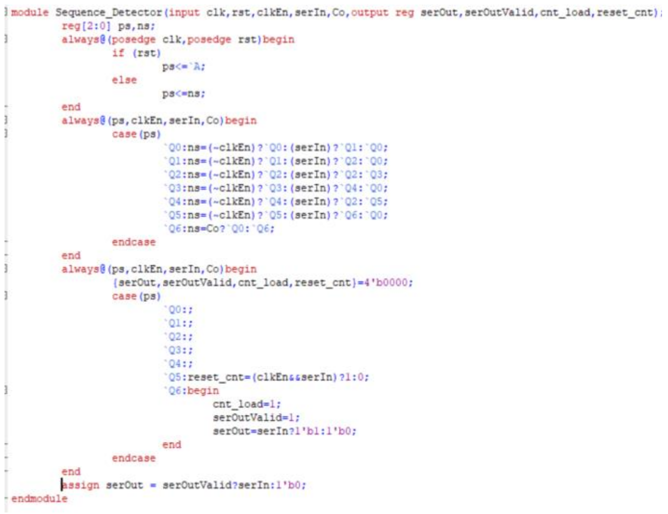


Fig 8. State diagram of sequence detector

As shown in the fig 8. This state machine detects 110101 on its SerIn and then the output stays 1 for 10 clock period.

D. Sequence Detector

This Sequence Detector circuit does 2 different things:
first of all, it keeps reading input from SerIn port until a "110101" sequence is detected.
second of all, it activates SerOut_valid signal, initializes the 4-bit counter with
"0110"(decimal of it = 6), activates to load signal for the counter and shows the value of SerIn on an output port called SerOut until the carry-out of the counter is equal to 1 (counter counts10 times). Then it returns to the initial state. Note that this circuit looks at CLK_EN alongside CLK as discussed.
previously. Fig. 8 shows the state diagram for Sequence Detector and Fig. 9 shows the Verilog implementation.

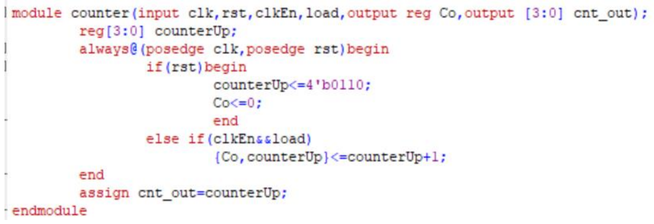


```
module Sequence_Detector(input clk,rst,clkEn,serIn,Co,output reg serOut,serOutValid,cnt_load,reset_cnt);
        reg[2:0] ps,ns;
        always@(posedge clk,posedge rst)begin
                if (rst)
                        ps<=`A;
                else
                        ps<=ns;
        end
        always@(ps,clkEn,serIn,Co)begin
                case(ps)
                        `Q0:ns=(~clkEn)?`Q0:(serIn)?`Q1:`Q0;
                        `Q1:ns=(~clkEn)?`Q1:(serIn)?`Q2:`Q0;
                        `Q2:ns=(~clkEn)?`Q2:(serIn)?`Q2:`Q3;
                        `Q3:ns=(~clkEn)?`Q3:(serIn)?`Q4:`Q0;
                        `Q4:ns=(~clkEn)?`Q4:(serIn)?`Q2:`Q5;
                        `Q5:ns=(~clkEn)?`Q5:(serIn)?`Q6:`Q0;
                        `Q6:ns=Co?`Q0:`Q6;
                endcase
        end
        always@(ps,clkEn,serIn,Co)begin
                {serOut,serOutValid,cnt_load,reset_cnt}=4'b0000;
                case(ps)
                        `Q0:;
                        `Q1:;
                        `Q2:;
                        `Q3:;
                        `Q4:;
                        `Q5:reset_cnt=(clkEn&&serIn)?1:0;
                        `Q6:begin
                                cnt_load=1;
                                serOutValid=1;
                                serOut=serIn?1'b1:1'b0;
                        end
                endcase
        end
        assign serOut = serOutValid?serIn:1'b0;
endmodule
```

Fig. 9 Verlog implementation of Sequence Detector

E. Counter

A simple 4-bit counter with 2 small changes; first, this counter need CLK_EN alongside CLK for each increment; second, the RST signal initializes the counter to "0110" (6 in
decimal). Fig. 10 shows the Verlog implementation of Counter.



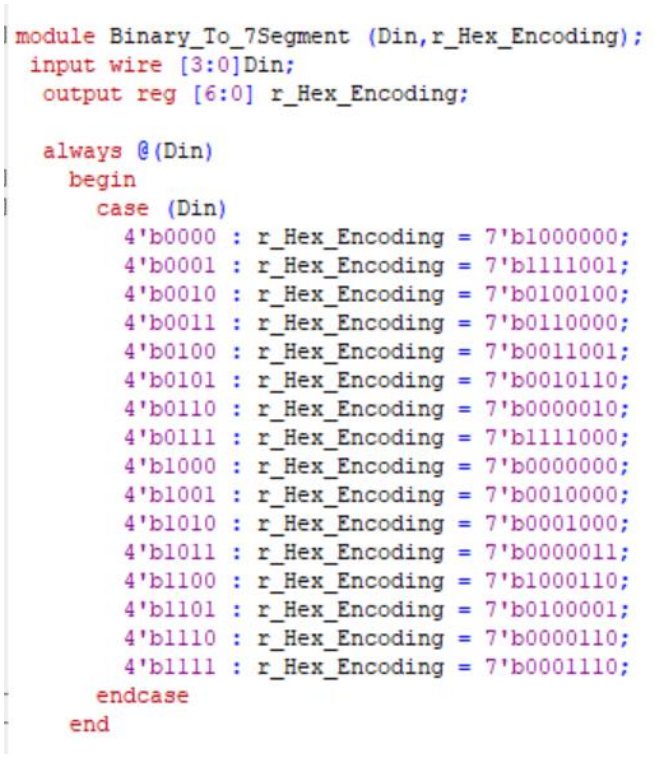
```
module counter(input clk,rst,clkEn,load,output reg Co,output [3:0] cnt_out);
        reg[3:0] counterUp;
        always@(posedge clk,posedge rst)begin
                if(rst)begin
                        counterUp<=4'b0110;
                        Co<=0;
                        end
                else if(clkEn&&load)
                        {Co,counterUp}<=counterUp+1;
        end
        assign cnt_out=counterUp;
endmodule
```

Fig 10 . Verilog of counter

The next xxpart of this experiment is called an Orthogonal FSM. It consists of a Moore state machine and a counter. When the clock Enable push-button is pushed the Moore sequence detector checks its input and decides which state to go. The sequence detector waits for the sequence of 110101 on its serIn input and when this is received, the serOutvalid output becomes one. The push button is pressed for every input that the state machine receives. After that, the counter will be enabled and will count for the next 10 consecutive clock cycles. During all these time the serOutvalid remains one.

F. Hex_Display

In order to see the counter's output on our FPGA, we use a Hex_Display module which takes the output of our counter and has a 7-bit output corresponding to a 7-segmented display.
Now we put everything together to create our Top-Level module and test it. Fig. 9 is out Top-Level module in Verilog. Fig. 14 is our test-bench and Fig.12 is the waveform results of
our test-bench.



```
module Binary_To_7Segment (Din,r_Hex_Encoding);
 input wire [3:0]Din;
  output reg [6:0] r_Hex_Encoding;

  always @(Din)
    begin
      case (Din)
        4'b0000 : r_Hex_Encoding = 7'b1000000;
        4'b0001 : r_Hex_Encoding = 7'b1111001;
        4'b0010 : r_Hex_Encoding = 7'b0100100;
        4'b0011 : r_Hex_Encoding = 7'b0110000;
        4'b0100 : r_Hex_Encoding = 7'b0011001;
        4'b0101 : r_Hex_Encoding = 7'b0010110;
        4'b0110 : r_Hex_Encoding = 7'b0000010;
        4'b0111 : r_Hex_Encoding = 7'b1111000;
        4'b1000 : r_Hex_Encoding = 7'b0000000;
        4'b1001 : r_Hex_Encoding = 7'b0010000;
        4'b1010 : r_Hex_Encoding = 7'b0001000;
        4'b1011 : r_Hex_Encoding = 7'b0000011;
        4'b1100 : r_Hex_Encoding = 7'b1000110;
        4'b1101 : r_Hex_Encoding = 7'b0100001;
        4'b1110 : r_Hex_Encoding = 7'b0000110;
        4'b1111 : r_Hex_Encoding = 7'b0001110;
      endcase
    end
```

Fig 11 . verlog of hex display

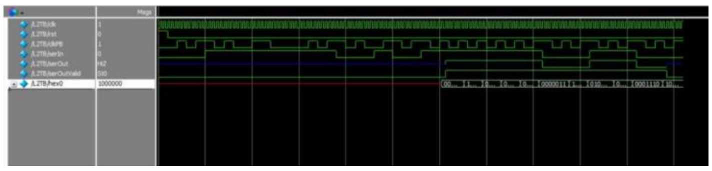
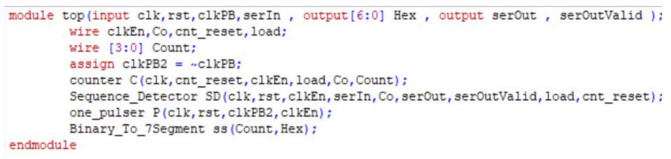


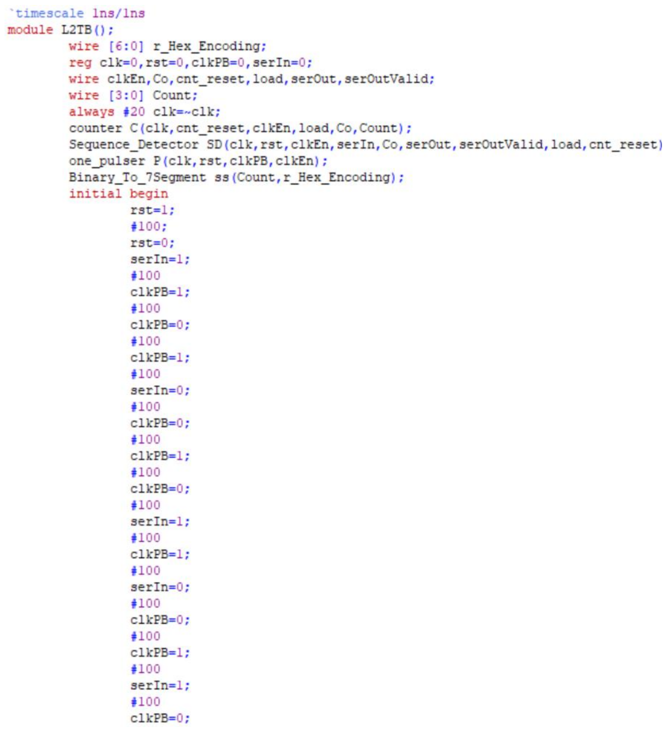
Fig 12. waveform of hex display

G. Top level Module

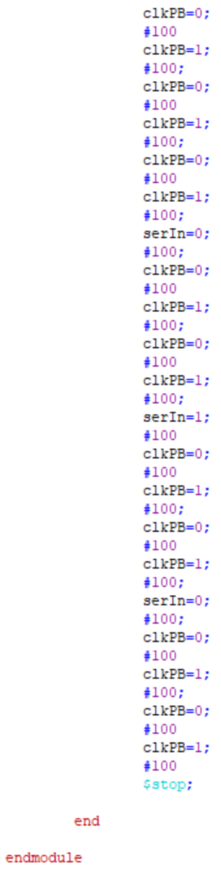


```
module top(input clk,rst,clkPB,serIn , output[6:0] Hex , output serOut , serOutValid );
        wire clkEn,Co,cnt_reset,load;
        wire [3:0] Count;
        assign clkPB2 = ~clkPB;
        counter C(clk,cnt_reset,clkEn,load,Co,Count);
        Sequence_Detector SD(clk,rst,clkEn,serIn,Co,serOut,serOutValid,load,cnt_reset);
        one_pulser P(clk,rst,clkPB2,clkEn);
        Binary_To_7Segment ss(Count,Hex);
endmodule
```

Fig 13. waveform of hex display



```
`timescale 1ns/1ns
module L2TB();
        wire [6:0] r_Hex_Encoding;
        reg clk=0,rst=0,clkPB=0,serIn=0;
        wire clkEn,Co,cnt_reset,load,serOut,serOutValid;
        wire [3:0] Count;
        always #20 clk=~clk;
        counter C(clk,cnt_reset,clkEn,load,Co,Count);
        Sequence_Detector SD(clk,rst,clkEn,serIn,Co,serOut,serOutValid,load,cnt_reset)
        one_pulser P(clk,rst,clkPB,clkEn);
        Binary_To_7Segment ss(Count,r_Hex_Encoding);
        initial begin
                rst=1;
                #100;
                rst=0;
                serIn=1;
                #100
                clkPB=1;
                #100
                clkPB=0;
                #100
                clkPB=1;
                #100
                serIn=0;
                #100
                clkPB=0;
                #100
                clkPB=1;
                #100
                clkPB=0;
                #100
                serIn=1;
                #100
                clkPB=1;
                #100
                serIn=0;
                #100
                clkPB=0;
                #100
                clkPB=1;
                #100
                serIn=1;
                #100
                clkPB=0;
```



```
                clkPB=0;
                #100
                clkPB=1;
                #100;
                clkPB=0;
                #100
                clkPB=1;
                #100;
                clkPB=0;
                #100
                clkPB=1;
                #100;
                serIn=0;
                #100;
                clkPB=0;
                #100
                clkPB=1;
                #100;
                clkPB=0;
                #100
                clkPB=1;
                #100;
                serIn=1;
                #100
                clkPB=0;
                #100
                clkPB=1;
                #100;
                clkPB=0;
                #100
                clkPB=1;
                #100;
                serIn=0;
                #100;
                clkPB=0;
                #100
                clkPB=1;
                #100;
                clkPB=0;
                #100
                clkPB=1;
                #100
                $stop;

        end

endmodule
```

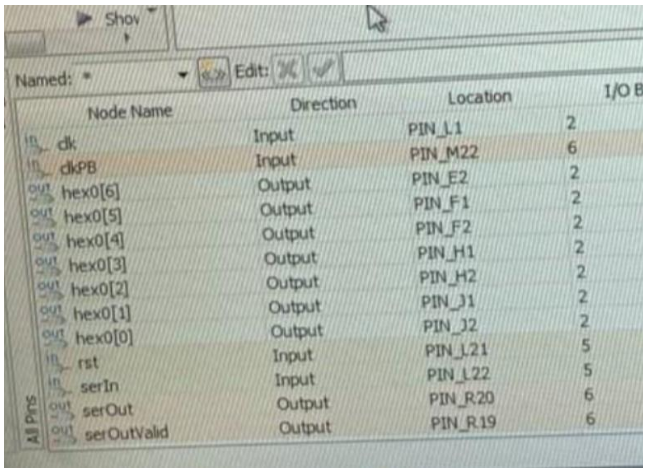
Fig 14 . test bench of all project

III. SYNTHESIS AND FPGA PROGRAMMING

Our next step is to Synthesis our design for our FPGA. In order to achieve this we use Quartus Prime software. First we create a new project and select our FPGA(Cyclone EP2C20F484C7) we add the top-level code to it.
II FPGA on DE1 board
. Then, add our Verilog files to the project.
The last thing we need to do is assign FPGA pins to inputs and outputs of Top-Leve module (PIN numbers can be found in DE1 device documentation). Here's the list of inputs and outputs and assigned pins:

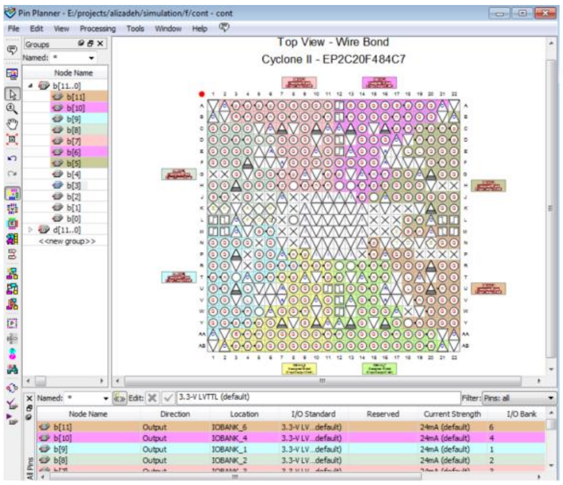
- CLK is assigned to boards 50 MHZ clock
- Hex_Display is assigned to hex0
- SerIn is assigned to SW0
- RST is assigned to SW1
- ClkPB is assigned to SW2
- SerOut is assigned to LEDR0
- SerOutValid is assigned to LEDR1

You can see the assignments inside Quartus Prime in Fig. 15



Node Name	Direction	Location	I/O B
clk	Input	PIN_L1	2
clkPB	Input	PIN_M22	6
hex0[6]	Output	PIN_E2	2
hex0[5]	Output	PIN_F1	2
hex0[4]	Output	PIN_F2	2
hex0[3]	Output	PIN_H1	2
hex0[2]	Output	PIN_H2	2
hex0[1]	Output	PIN_J1	2
hex0[0]	Output	PIN_J2	2
rst	Input	PIN_L21	5
serIn	Input	PIN_L22	5
serOut	Output	PIN_R20	6
serOutValid	Output	PIN_R19	6

Fig 15. Pin planner



Pin Planner - E:/projects/alizadeh/simulation/f/cont - cont

Top View - Wire Bond
Cyclone II - EP2C20F484C7

Named: * Edit: 3.3-V LVTTL (default) Filter: Pins: all

Node Name	Direction	Location	I/O Standard	Reserved	Current Strength	I/O Bank
b[11]	Output	IOBANK_6	3.3-V LV...default)		24mA (default)	6
b[10]	Output	IOBANK_4	3.3-V LV...default)		24mA (default)	4
b[9]	Output	IOBANK_1	3.3-V LV...default)		24mA (default)	1
b[8]	Output	IOBANK_2	3.3-V LV...default)		24mA (default)	2

Fig 16. Pin planner sample

And at least we can test our Serial Transmitter on our FPGA! Below you can see pictures of final tests.
Then we program our design on the board. We detect 110101 sequence of ser_in. Then counting to 10 starts. (From6 to F)

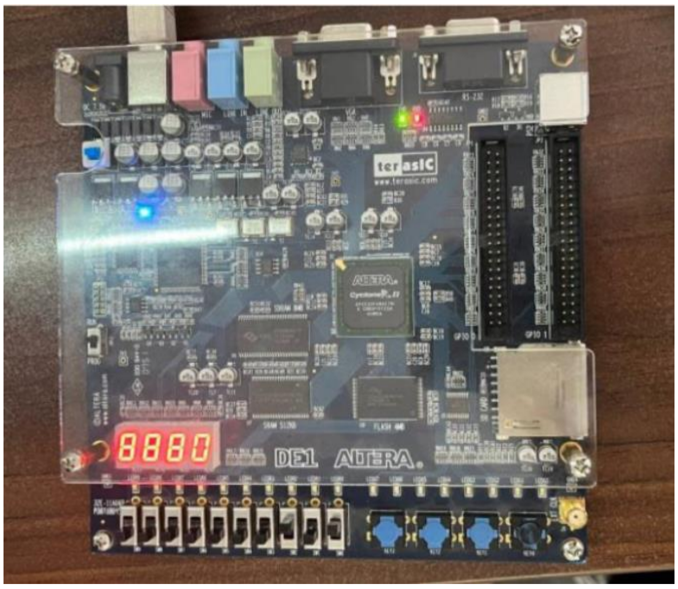


Fig. 12 Initial state of FPGA

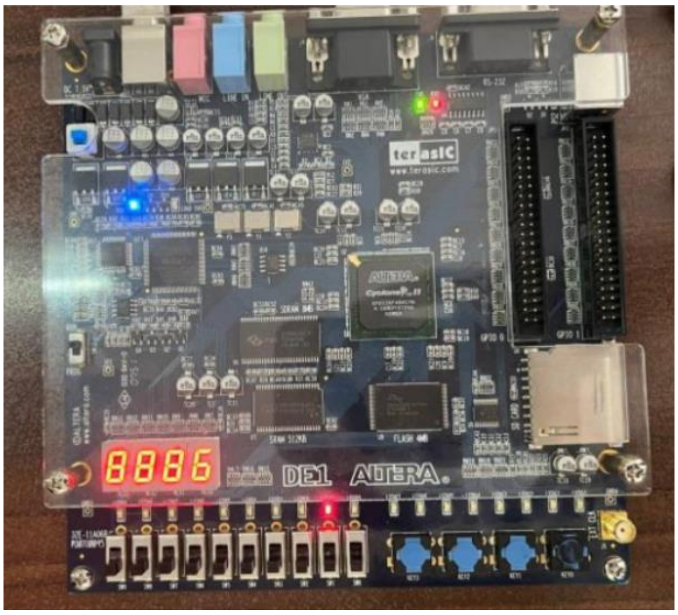


Fig. 17 SerOut and SerIn are equal

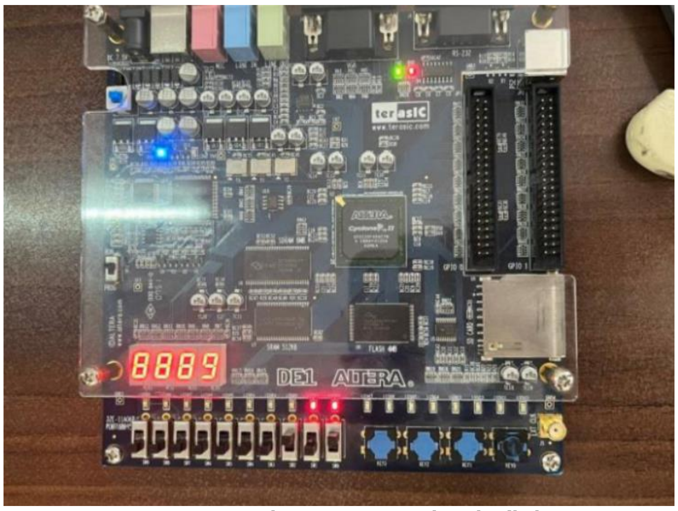


Fig. 18 Counter keeps counting with each ClkPb

IV.CONCLUSIONS

To wrap it up, our design is behaving as expected similar
to our simulations using ModelSim.
We designed the Serial Transmitter , we implemented its hardware design by coding on ModelSim, after that we transmit our code to Quartus and fix the errors. After all the output on the hardware is what we expected.

ACKNOWLEDGMENT

This report was prepared and developed by Shahzad Momayez(SID:810100272) and Mohammad Amanlou(SID:810100084) ,bachelor students of Computer engineering at University of Tehran, under the supervision of Professor Zain Navabi.