Experiment 2 - Sequential Synthesis and FPGA Device Programming

Ali Padyav, 810199388
Bita Nasiri, 810199504

***Abstract*— This document is a report about designing Seiral Transmitter and implementing on FPGA for the 2nd experiment of the DLD Lab (ECE 045) at University of Tehran, Department of Electrical and Computer Engineering.**

***Keywords*— Serial Transmitter, Sequence Detector, FPGA Programming, Altera Cyclone II, Quartus, Seven Segment Display**

I. RTL Design

A. Datapath

Fig. 1 is whole circuit's datapath.

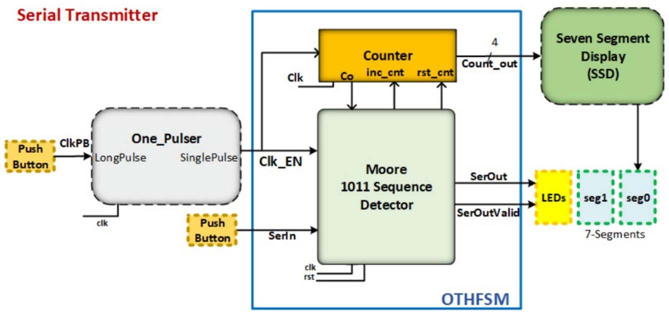


Fig. 1 Datapath design

B. Onepulser

Onepulser is used for asserting ClkEn whe we need to get input or count. Because we cant work with board's clock frequency.

1) Controller

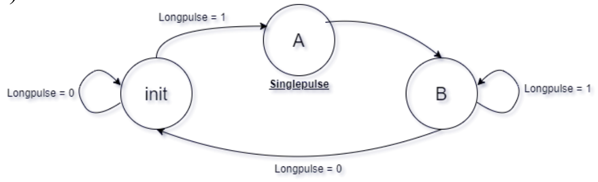
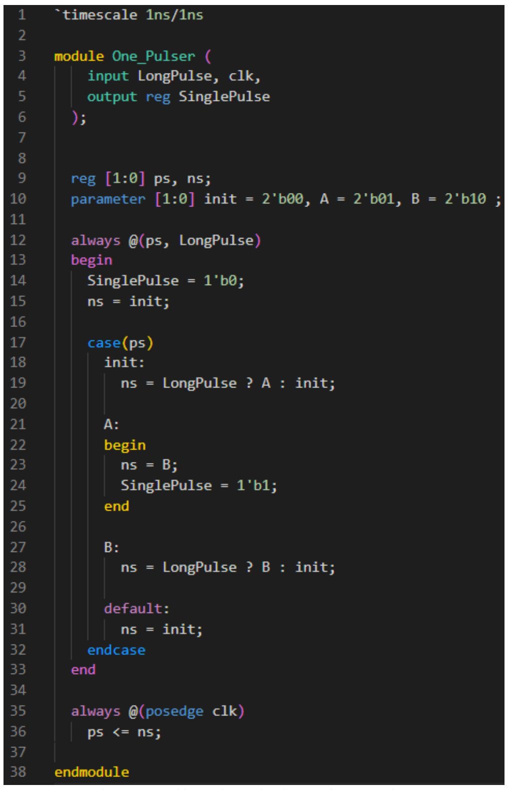


Fig. 2 Onepulser controller

2) Verilog Description



```
`timescale 1ns/1ns

module One_Pulser (
    input LongPulse, clk,
    output reg SinglePulse
  );


  reg [1:0] ps, ns;
  parameter [1:0] init = 2'b00, A = 2'b01, B = 2'b10 ;

  always @(ps, LongPulse)
  begin
    SinglePulse = 1'b0;
    ns = init;

    case(ps)
      init:
        ns = LongPulse ? A : init;

      A:
      begin
        ns = B;
        SinglePulse = 1'b1;
      end

      B:
        ns = LongPulse ? B : init;

      default:
        ns = init;
    endcase
  end

  always @(posedge clk)
    ps <= ns;

endmodule
```

Fig. 3 Verilog description of onepulser

3) Testbench

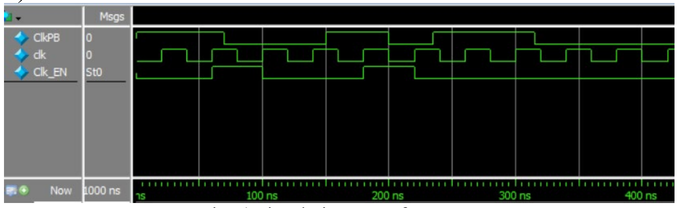


Fig. 4 Simulation waveform

C. Orthogonal Finite State Machine

1) State diagram of the sequence detector

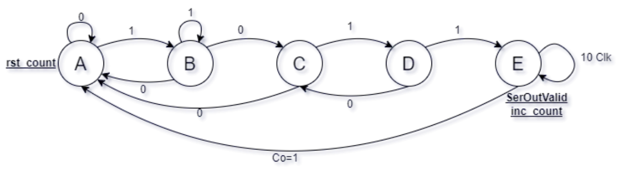
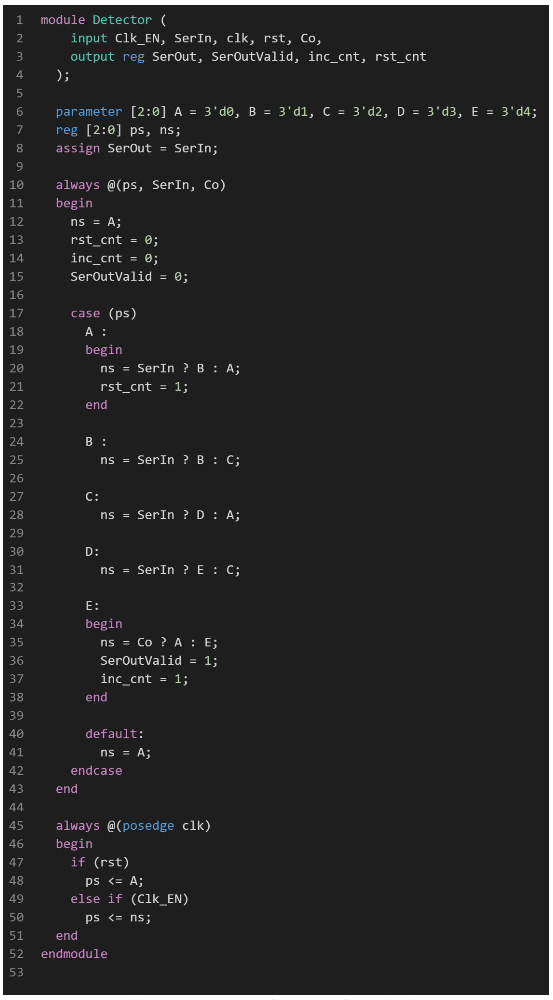


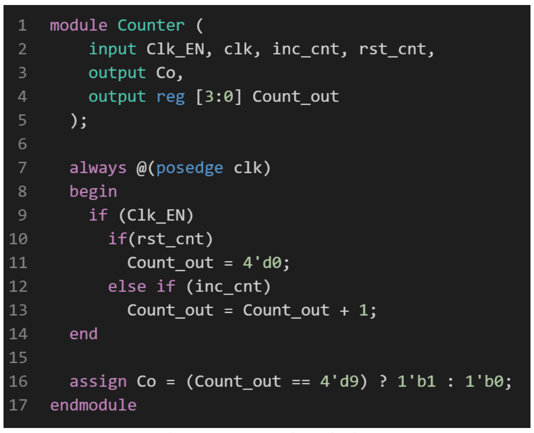
Fig. 5 Moore machine of 1011 detector

2) Verilog Description



```verilog
module Detector (
    input Clk_EN, SerIn, clk, rst, Co,
    output reg SerOut, SerOutValid, inc_cnt, rst_cnt
  );

  parameter [2:0] A = 3'd0, B = 3'd1, C = 3'd2, D = 3'd3, E = 3'd4;
  reg [2:0] ps, ns;
  assign SerOut = SerIn;

  always @(ps, SerIn, Co)
  begin
    ns = A;
    rst_cnt = 0;
    inc_cnt = 0;
    SerOutValid = 0;

    case (ps)
      A :
      begin
        ns = SerIn ? B : A;
        rst_cnt = 1;
      end

      B :
        ns = SerIn ? B : C;

      C:
        ns = SerIn ? D : A;

      D:
        ns = SerIn ? E : C;

      E:
      begin
        ns = Co ? A : E;
        SerOutValid = 1;
        inc_cnt = 1;
      end

      default:
        ns = A;
    endcase
  end

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

Fig. 6 Sequence detector description



```verilog
module Counter (
    input Clk_EN, clk, inc_cnt, rst_cnt,
    output Co,
    output reg [3:0] Count_out
  );

  always @(posedge clk)
  begin
    if (Clk_EN)
      if(rst_cnt)
        Count_out = 4'd0;
      else if (inc_cnt)
        Count_out = Count_out + 1;
  end

  assign Co = (Count_out == 4'd9) ? 1'b1 : 1'b0;
endmodule
```

Fig. 7 Counter description

3) Testbench

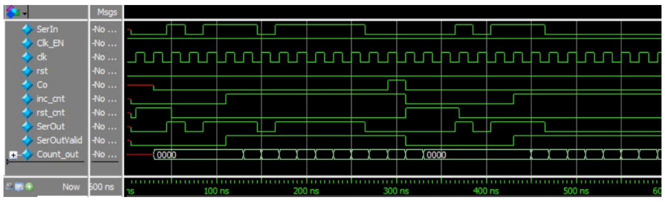


Fig. 8 OTHFSM simulation that shows after detecting 1011, starts counting and after counting it can detect sequences again.

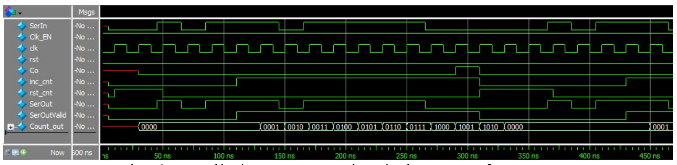
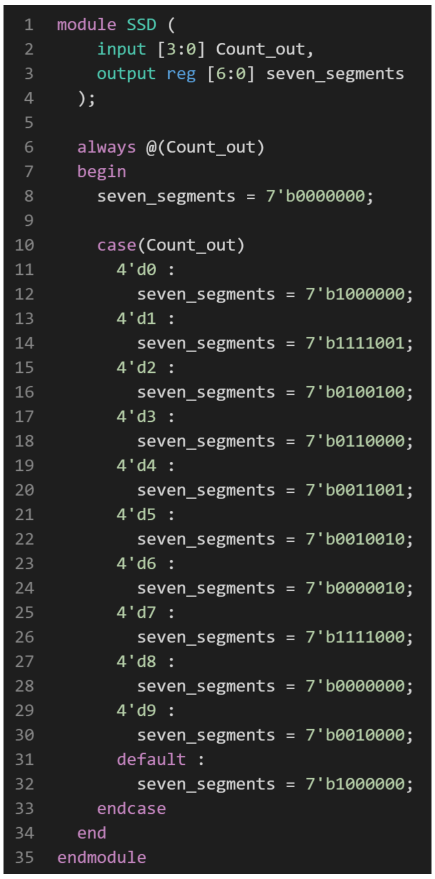


Fig. 9 Detailed OTHFSM simulation waveform

D. Seven Segment Display

It’s like a decoder that get CountOut and decode it to 7 bit that work with SSD.

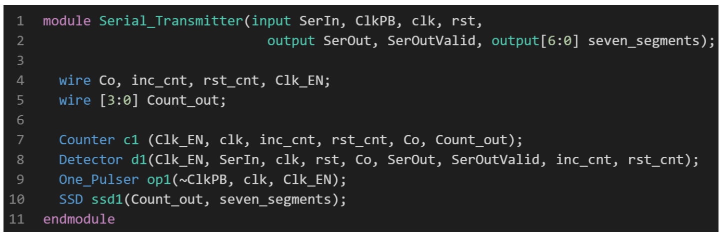


```verilog
module SSD (
    input [3:0] Count_out,
    output reg [6:0] seven_segments
  );

  always @(Count_out)
  begin
    seven_segments = 7'b0000000;

    case(Count_out)
      4'd0 :
        seven_segments = 7'b1000000;
      4'd1 :
        seven_segments = 7'b1111001;
      4'd2 :
        seven_segments = 7'b0100100;
      4'd3 :
        seven_segments = 7'b0110000;
      4'd4 :
        seven_segments = 7'b0011001;
      4'd5 :
        seven_segments = 7'b0010010;
      4'd6 :
        seven_segments = 7'b0000010;
      4'd7 :
        seven_segments = 7'b1111000;
      4'd8 :
        seven_segments = 7'b0000000;
      4'd9 :
        seven_segments = 7'b0010000;
      default :
        seven_segments = 7'b1000000;
    endcase
  end
endmodule
```

Fig. 10 Verilog description of SSD.

E. Top Module

We put all modules together to build top module to implement it on FPGA.



```
module Serial_Transmitter(input SerIn, ClkPB, clk, rst,
                             output SerOut, SerOutValid, output[6:0] seven_segments);

  wire Co, inc_cnt, rst_cnt, Clk_EN;
  wire [3:0] Count_out;

  Counter c1 (Clk_EN, clk, inc_cnt, rst_cnt, Co, Count_out);
  Detector d1(Clk_EN, SerIn, clk, rst, Co, SerOut, SerOutValid, inc_cnt, rst_cnt);
  One_Pulser op1(~ClkPB, clk, Clk_EN);
  SSD ssd1(Count_out, seven_segments);
endmodule
```

Fig. 11 Serial Transmitter description

II. FPGA IMPLEMENTATION

We import codes to Quartus and synthesize them. Then define Altera Cyclone II as device. So we can use pin planner to assign our module input and outputs to FPGA pins.

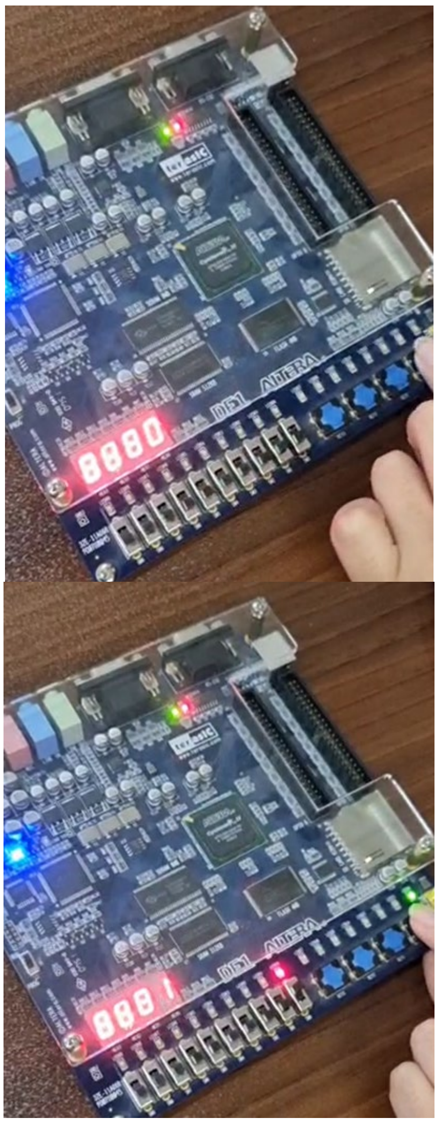


Fig. 12 FPGA working properly. When it detects sequence, green led turns on and then starts counting.