Experiment 2 - Sequential Synthesis and FPGA Programming

Ali Banihashemi - MohammadSadegh Aghili

810100245 - 810100274

Abstract: In this experiment we got familiar with Concepts of state machines and Sequence detectors, FPGA devices and implementation and designing a MSSD component.

Keywords: *state machine – Sequence detector – FPGA – MSSD– Huffman coding*

I. EXPERIMENTS

2-1. Onepulser

This component is used to freeze the clock until we make changes to the serIn and then after releasing it, circuit will use our new input.

Fig.1: Verilog description of the one-pulser module

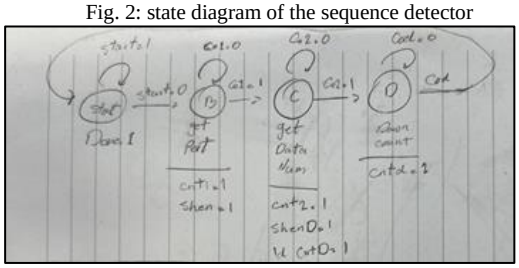
```
module onePulser(clkPB,clk,rst,clkEn);
  input clkPB,clk,rst;
  output reg clkEn;
  parameter[1:0] A= 2'b00,B = 2'b01,C = 2'b10;
  reg[1:0]ns,ps;
    always@(ps,clkPB)
    begin
      ns = 2'b00;

      case(ps)
          A : ns = clkPB?B:A;
          B : ns = C;
          C : ns = clkPB?C:A;
    endcase
    end
    always@(ps)
    begin
      case(ps)
        A:clkEn = 1'b0;
        B:clkEn = 1'b1;
        C:clkEn = 1'b0;
      endcase
    end
    always@(posedge clk , posedge rst)
    begin
      if(rst)
        ps <= 2'b0;
      else
        ps <= ns;
    end
endmodule
```

2-2. Finite State Machine and the counters

This state machine is implemented using Huffman coding style, in each state we decide to whether change present state or not based on the inputs of the state machine. Some of inputs are generated in the data-path such as counters' carry-out and some are inputs of data-path as well(serIn).

Fig. 2: state diagram of the sequence detector



Then writing the Verilog description of the FSM module is required. This description includes two always blocks. One of them changes next state and control signals based on present state. The other one changes present state to next state based on clock and reset signals.

Fig.3: Verilog description of the FSM module

```
module controller(input serIn,co1,co2,coD,clk,
rst,clkEn, output reg cnt1,cnt2,cntD,output ldcntD,
output reg shen,shenD,output seroutvalid, done);
  parameter[1:0] A= 2'b00,B = 2'b01,
        C = 2'b10, D = 2'b11;
  reg[1:0]ns,ps;
    always@(ps,co1,co2,coD,serIn)
    begin
      ns = 2'b00;
      {cnt1,cnt2,cntD,shen,shenD} = 5'b0;
      case(ps)
          A :begin ns = serIn ? A : B;  end
          B :begin ns = co1 ? C : B;
         {cnt1,shen} = 2'b11; end
          C :begin ns = co2 ? D : C;
         {cnt2,shenD} = 2'b11; end
          D :begin ns = coD ? A : D; {cntD} = 1'b1;
                end
    endcase
    end
    always@(posedge clkEn , posedge rst)
    begin
      if(rst)
        ps <= 2'b0;
      else
        ps <= ns;
    end
    assign ldcntD = co2;
    assign done = coD;
    assign seroutvalid = (ps==D && ~coD);
endmodule
```

The two counters has the same logic but one of them count to 2 (decimal) and the other one counts two 4 , because there are 2 bits that represent the port number and 4 bits that determine number of inputs. So first one can be 1-bit and the second one can be 2-bits. In each one carry-out signal shows if it reached the end of counting or not:

Fig. 4: Verilog description of the counter module

```
module cnt1bit(cnt,clk,rst,clkEn,co);
  input cnt,clk,rst,clkEn;
  output co;
  reg count;
  always@(posedge clkEn, posedge rst)
  begin
      count = rst ? 1'b0:~cnt?count:
      (count==1'b1)?1'b0:count+1;
  end
  assign co = (count == 1'b1 && cnt)?
          1'b1 : 1'b0;
endmodule
```

```
module cnt2bit(cnt,clk,rst,clkEn,co);
  input cnt,clk,rst,clkEn;
  output co;
  reg [2:0] count;
  always@(posedge clkEn, posedge rst)
  begin
      count = rst ? 3'b000:~cnt?count:
      (count==3'b100)?3'b000:count+1;
  end
  assign co = (count == 3'b100 && cnt)?
          1'b1 : 1'b0;
endmodule
```

2-3. Shift Registers and Demultiplexers

Shift registers are used to store port and data number so one of them is storing a 2-bit value and the other one is storing a 4-bit value , but their logic are exactly the same and they are just different in number of bits.

Fig. 5: Verilog description of the 4-bit shift register module

```
module shr4bit(serIn , shen , clk,rst,clkEn,portnum);
  input serIn,shen,clk,rst,clkEn;
  output[3:0] portnum;
  reg [3:0] shift;
  always@(posedge clkEn , posedge rst)
  begin
    if(rst)
      shift <= 4'b0;
    else if(shen)
      shift <= {shift[2:0] , serIn};
  end
  assign portnum=shift;
endmodule
```

In this DMUX , there is a 2-bit select signal which chooses one of the 4 available ports, then we assign serIn input to the selected port and the Z-value to the other 3 ports.

Fig. 6: Verilog description of the demultiplexer module

```
module Dmux(input [1:0] sel,
  serIn, output [3:0] p);
  assign p = (sel==2'b00) ? {serIn,3'bz} :
 (sel==2'b01) ? {1'bz,serIn,2'bz} :
 (sel==2'b10) ? {2'bz,serIn,1'bz} :
 {3'bz,serIn};
endmodule
```

2-4. Seven Segment Display

In the Verilog description of this module, there is an always block we for every change of 4-bit input which is a 4-bit binary number , changes the 7-bit output that this 7-bit number represent seven segment display number.

Fig. 7: Verilog description of the seven-segment display module

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

3. MSSD Implementation

To make a top-level design , there are two major parts; controller and data-path. Controller has been written so after writing data-path, these two should be wired together in a top-level design which is called MSSD.

Data-path includes all the components such as counters and shift registers and it wire them together:

Fig. 8: Verilog description of the data-path

```
module datapath(input clkPB,clk,rst,
cnt1,cnt2,cntD,ldcntD,shen,
shenD,seroutvalid,done,serIn,
output co1,co2,coD,output [6:0] ssdout
,output [3:0] p,output clkEn);
  onePulser onep(clkPB,clk,rst,clkEn);
  wire[1:0] portnum;
  cnt2bit cnt22(cnt2,clk,rst,clkEn,co2);
  shr2bit portsh(serIn,shen,clk,rst,clkEn,portnum);
  wire[3:0] portnum2;
  shr4bit datash(serIn,shenD,clk,rst,clkEn,portnum2);
  cnt1bit cnt11(cnt1,clk,rst,clkEn,co1);
  wire[3:0] count2;
  Dcnt dcnt(cntD,portnum2,clk,rst,
  clkEn,ldcntD,coD,count2);
  Dmux dmux(portnum,serIn,p);
  SSD ssd(count2,ssdout);
endmodule
```

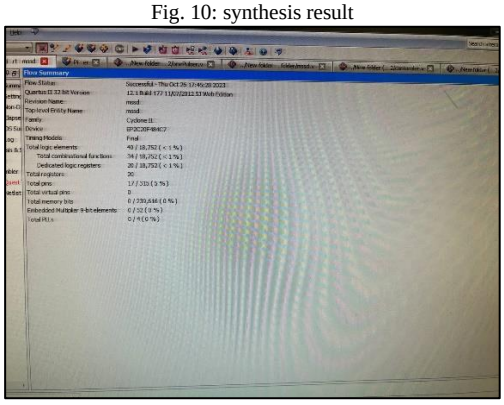
Then to finish the top level design , only wiring the data-path which contains all the components with the controller is required.

Fig. 9: top-level Verilog description

```
module mssd(input clk,rst,serIn,clkPB,
output done,seroutvalid,
output [6:0] ssdout,output [3:0] p);
  wire cnt1,cnt2,cntD,ldcntD,shen,shenD,co1,co2,coD;
  datapath dt(clkPB,clk,rst,cnt1,cnt2,cntD
   ,ldcntD,shen,shenD,seroutvalid,done,serIn,
   co1,co2,coD,ssdout,p,clkEn);
   controller ct( serIn,co1,co2,coD,clk,rst,clkEn,
   cnt1,cnt2,cntD,ldcntD,
   shen,shenD,seroutvalid,done);
endmodule
```

Next step is performing the synthesis of the top level design in the Quartus.

Fig. 10: synthesis result



Then pin assignments should be done. In this step signals that are in Verilog description of MSSD, will be assigned to actual components on a physical board which in this experiment is a Cyclone II device:

Fig. 11: Pin assignments

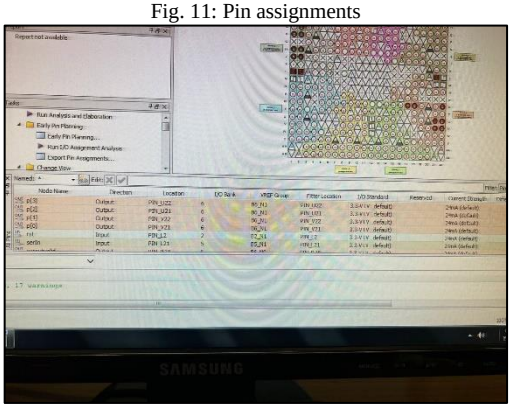
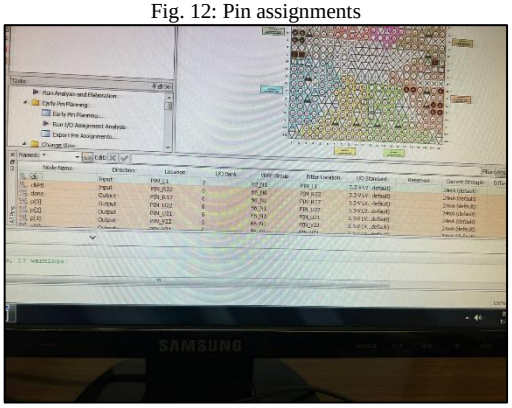
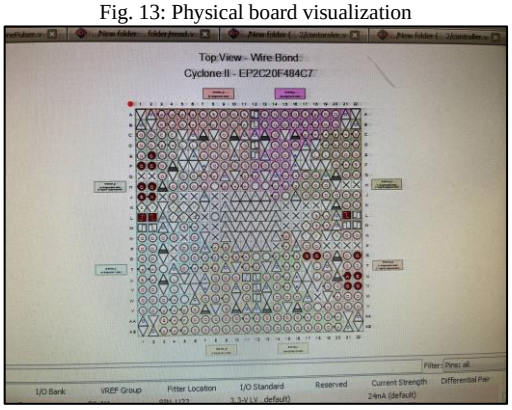


Fig. 12: Pin assignments



And at last visualization of the physical board will be as below:

Fig. 13: Physical board visualization



II. Conclusions

In this experiment, we learnt Concepts of state machines and Sequence detectors, Designing simulations, Synthesis and FPGA programming and implementation.

III. References

[1] Katayoon Basharkhah and Zain Navabi, *Digital Logic Laboratory*, University of Tehran, Fall 1402.