# EX#2-Sequential Synthesis and FPGA Programming

Ali Ghahari 810100201 Mohammad Sadeghi 810100175

***Introduction**: In this experience, we will design an Multi-channel Synchronous Serial communication Demultiplexer(MSSD) which see the Huffman coding style , simulation and synthesis through it.*

## A) RTL Design

*In this module we get serin input after that start signal change from 1 to 0. The first two bit of serin indicate the port number (p) which output transmit on it. The 4 next bit stands for bit number of data (n) and n last bit is data value.at the end done signal become 1 when start return to 1.*

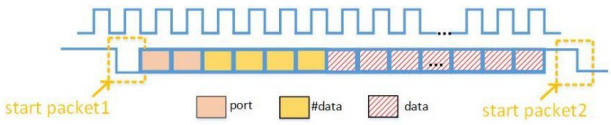



Fig1

*The circuit component show in fig2 and we will explain:*

1) *We need one_pulser to create a clk pulse during button push.*
2) *Two data_counter and port_counter to calculate the port detecting clock cycles and data count numbers.*
3) *Two shift register for saving the port number and data bit number.*
4) *Demux for selecting the choosen port to transsmit the output.*
5) *A Controller unit to manage the module states.*
6) *Seven segment display for showing the entered bit of data.*

*By giving each bit of data the number of seven segment decrease until reach zero.also for each bit of serin we must push button so one_puser can create a single clock.*

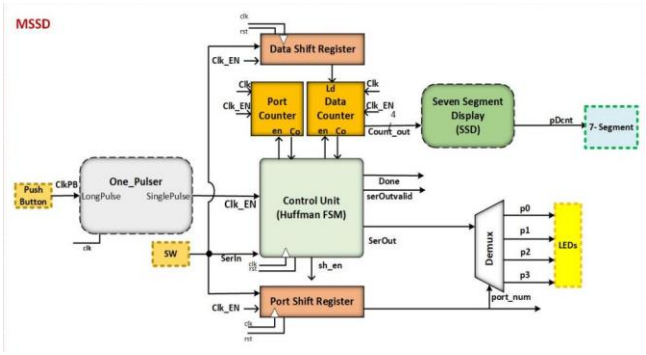


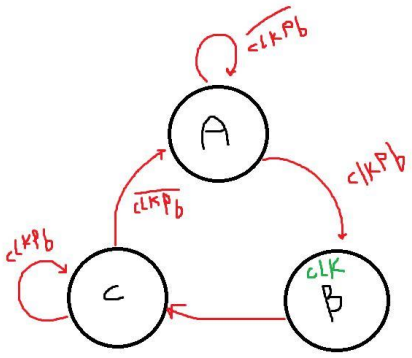

Fig2

## One_pulser

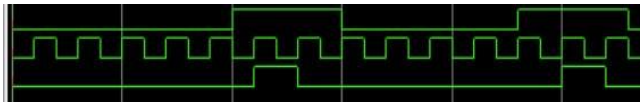
*We will see verilog code of module and simulateion waveform:*

```
module OnePulser(clkpb , clk , rst , clken);
input clkpb , clk , rst;
output clken;

parameter A = 2'b00 , B = 2'b01 , C = 2'b10;

reg [1:0]ps;
reg [1:0]ns;

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

assign clken = ps == B;
      AliGhAliGh, 6 days ago • init
always @(ps , clkpb) begin
    ns = A;
    case (ps)
        A: ns = clkpb ? B : A;
        B: ns = C;
        C: ns = clkpb ? C : A;
        default: ns = 2'bxx;
    endcase
end

endmodule
```

*This one pulser has state diagram with 3 states.when we push button clkpb become 1 and we go to B state and from there we go to C unconditionally.we stay at last state until clkpb toggle to 0 and then go back to first state.*



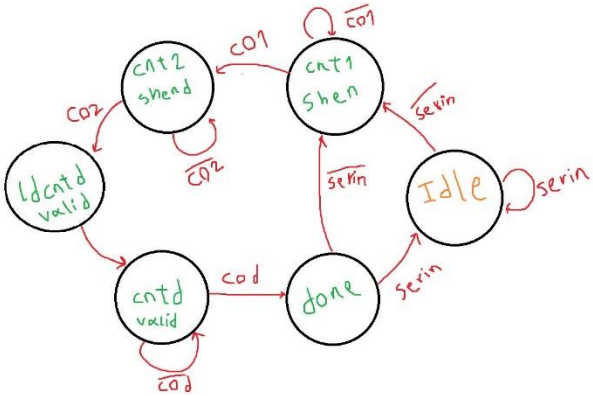

*So we have simulate result in wave:*



*We can see during clkpb become 1 we have clk pulse in output.*

## *FSM controller*

*First let see the state diagram for MMSD controller:*




*Let list the input signals:*

1) *Serin:main input which read bit by bit*
2) *co1&co2: carry out of two counter which count two for port number and four for datanumber*
3) *cod:its for loadcounter which become 0 when we send all bit of data.*
4) *Clk,rst,clken: synchronous controll signals*

*Now output of state machine:*

1) *Cnt1&cnt2 : these signals allow counters of datapath to statr countig.*
2) *Cntd : do as well for load counter.*
3) *Ldcounter:this signal will initialize the seven_segment number for show as data bit number and with each bit we have read this number decrease by loadcounter.*
4) *Shen & shend : allow shift_registers to recive and save datas.*
5) *Valid: identify that we are reading valid bit data for data value.*
6) *Done : it is signal for ensure that proccess get end.*

```
module Controller(clk ,clken ,rst ,serin ,co1 ,co2 ,cod ,cnt1 ,cnt2 ,cntd ,ldcntd ,shen ,shend ,valid ,done);
input serin ,co1 ,co2 ,cod ,clk ,rst ,clken;
output cnt1 ,cnt2 ,cntd ,ldcntd ,shen ,shend ,valid ,done;

parameter IDLE = 3'b000 , PORT = 3'b001 , NUM = 3'b010 , DATA = 3'b011 , DONE = 3'b100 , LDC = 3'b101;

reg[2:0] ns ,ps;

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

assign done = ps == DONE;
assign cnt1 = ps == PORT;
assign cnt2 = ps == NUM;
assign cntd = ps == DATA;
assign ldcntd = ps == LDC;
assign shen = ps == PORT;
assign shend = ps == NUM;
assign valid = (ps == DATA)||(ps == LDC);


always @(serin ,co1 ,co2 ,cod, ps) begin
    ns = 3'bxx;
    case (ps)
        IDLE: ns = serin ? IDLE : PORT;
        PORT: ns = co1 ? NUM : PORT;
        NUM: ns = co2 ? LDC : NUM;
        DATA: ns = cod ? DONE : DATA;
        DONE: ns = serin ? IDLE : PORT;
        LDC: ns = DATA;
        default: ns = 3'bxx;
    endcase
end

endmodule
```

## Counters

*We have counter code with 2bit state  which pass it the load value as parameter. What that means?*

*For example when load set to 0, the couner count four and go back to 0;*

*And for two count we set load to 2;*

*This is verilog of counter:*

```
module Counter #(parameter load) (clk ,rst ,clken ,cnt ,co);
    input clk ,rst ,clken ,cnt;
    output co;

    reg[1:0] ps;


    always @(posedge clk, posedge rst) begin
        if(rst)
            ps = load;
        else if(clken && cnt)
            ps = ps + 1;
    end

    assign co = ps == 2'b11;
endmodule
```

*As you see, when we reach to state 3 the cout become one.*

*Exactly, we are counting one unit less.*

*But however we must read one bit to go to next state so it is no matter.*

*But the load counter dose not have parameter:*

```
module LdCounter (clk ,rst ,clken ,data ,cnt ,ld ,co ,out);
    input clk ,rst ,clken ,cnt ,ld;
    input[3:0] data;
    output co;
    output reg[3:0] out;

    always @(posedge clk, posedge rst) begin
        if(rst)
            out = 0;
        else if(ld && clken)
            out = data - 1;
        else if(clken && cnt)
            out = out - 1;
    end

    assign co = out == 1;
endmodule
```

*In this counter we have the data number in first and with each bit that we have recive during clkpb we decrease the number of counter output which say to seven_segment its value to show.*

## The Demux

*This component will select that which port should data go on it.*

*The selector have input and 4 port as output that select one of them from value of port register to send input on that port.*

```
module Demux(serin, port ,p0 ,p1 ,p2 ,p3);
input serin;
input[1:0] port;
output p0 ,p1 ,p2 ,p3;

assign p0 = port == 0 ? serin : 1'b0;
assign p1 = port == 1 ? serin : 1'b0;
assign p2 = port == 2 ? serin : 1'b0;
assign p3 = port == 3 ? serin : 1'b0;
endmodule
```

## Shift_Registers

*There is no complex thing with this part.we recivie bits and set them in register from left(means if we send 1,0,0,0 in order register final value will be 0001)*

```
1  module ShReg #(parameter n)(serin ,en ,rst ,clk ,clken ,out);
2  input serin ,en ,rst ,clk ,clken;
3  output reg[n-1:0] out;
4
5  always @(posedge clk ,posedge rst) begin
6      if(rst)
7          out = 0;
8      else if(clken && en)
9          out = {out ,serin};
10 end
11 endmodule       AliGhAliGh, 6 days ago • init
```

*the port number register must be 2 bit and data num register must be 4 so we use parameter method in verilog code.*

## *Seven segment*

*In last component we have SSD which can show us 0 to 15 in hex based on input it gets:*

```
module SSD(inp ,out);
input[3:0] inp;
output reg[6:0] out;

parameter _0 = 7'h40;
parameter _1 = 7'h79;
parameter _2 = 7'h24;
parameter _3 = 7'h30;
parameter _4 = 7'h19;
parameter _5 = 7'h12;
parameter _6 = 7'h02;
parameter _7 = 7'h78;
parameter _8 = 7'h00;
parameter _9 = 7'h10;
parameter _a = 7'h08;
parameter _b = 7'h03;
parameter _c = 7'h46;
parameter _d = 7'h21;
parameter _e = 7'h06;
parameter _f = 7'h0e;

always @(inp) begin
    out = 0;
    case (inp)
        0: out = _0;
        1: out = _1;
        2: out = _2;
        3: out = _3;
        4: out = _4;
        5: out = _5;
        6: out = _6;
        7: out = _7;
        8: out = _8;
        9: out = _9;
        10: out = _a;
        11: out = _b;
        12: out = _c;
        13: out = _d;
        14: out = _e;
        15: out = _f;
        default: out = 7'bx;
    endcase
end
endmodule
```

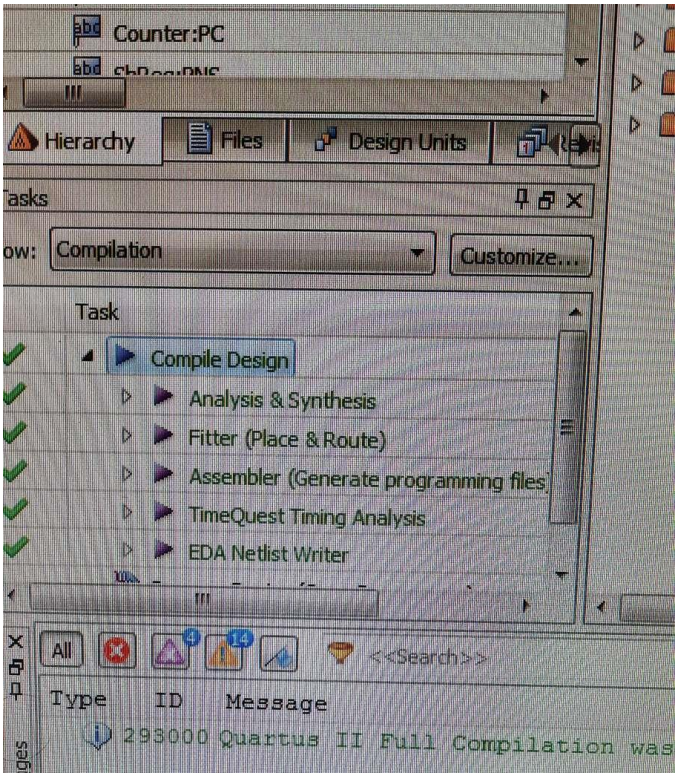
*According to fig-1 in page 1 this SSD has 4bit input that represent data bit number and set 7 bit output which is value of seven_segment circuit.*

## *MMSD Implementation*

*First of all we wiring all components and also controller unit in main module:*

```
module MSSD(serin, clkpb, clk, rst, p0 ,p1 ,p2 ,p3 ,done ,valid ,sevseg);
input serin, clkpb, clk, rst;
output p0 ,p1 ,p2 ,p3 ,done ,valid;
output[6:0] sevseg;

wire clken ,co1 ,co2 , cod ,cnt1 ,cnt2 ,cntd ,ldcntd ,shen ,shend ,valid ,done;
wire[1:0] port;
wire[3:0] segdata ,dnsout;

OnePulser OP(.clkpb(clkpb),.clk(clk),.rst(rst),.clken(clken));
Controller C(.clk(clk) ,.clken(clken) ,.rst(rst) ,.serin(serin) ,.co1(co1) ,.co2(co2) ,.cod(cod) ,.cnt1(cnt1) ,.cnt2(cnt2) ,.cntd(cntd)
,.ldcntd(ldcntd) ,.shen(shen) ,.shend(shend) ,.valid(valid) ,.done(done));
Counter #(2)PC(.clk(clk) ,.rst(rst) ,.clken(clken) ,.cnt(cnt1) ,.co(co1));
Counter #(0)DNC(.clk(clk) ,.rst(rst) ,.clken(clken) ,.cnt(cnt2) ,.co(co2));
Demux DM(.serin(serin), .port(port) ,.p0(p0) ,.p1(p1) ,.p2(p2) ,.p3(p3));
LdCounter LC(.clk(clk) ,.rst(rst) ,.clken(clken) ,.data(dnsout) ,.cnt(cntd) ,.ld(ldcntd) ,.co(cod) ,.out(segdata));
ShReg #(2)PNS (.serin(serin) ,.en(shen) ,.rst(rst) ,.clk(clk) ,.clken(clken) ,.out(port));
ShReg #(4)DNS (.serin(serin) ,.en(shend) ,.rst(rst) ,.clk(clk) ,.clken(clken) ,.out(dnsout));
SSD ssd(.inp(segdata) ,.out(sevseg));
endmodule
```

*Now we convey these codes on new wizard project which we create in Quartus 12.1 and before synthsis we should compiled all files:*

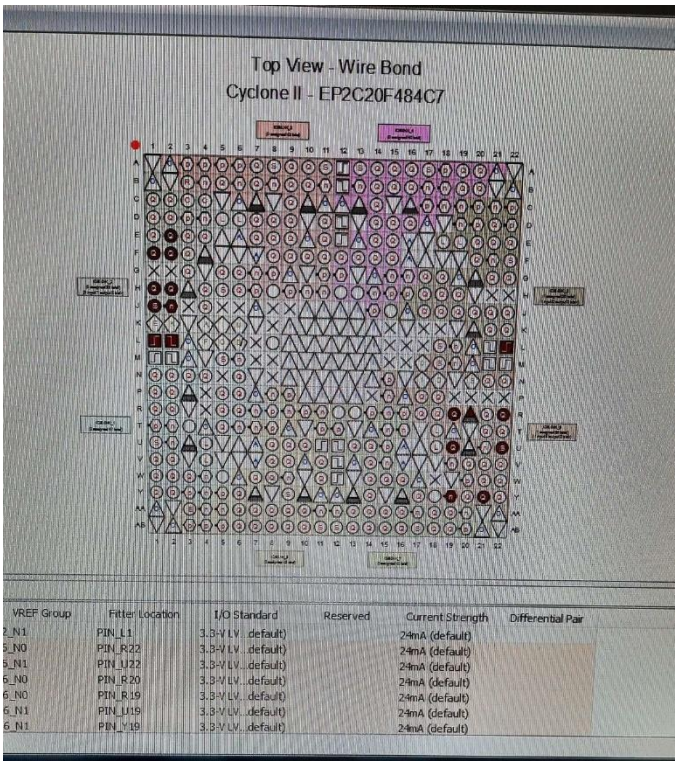



*As we can see analysis and synthsis , routing and prepost synthesis sections done well withouth error(just some warnings)*

## *Pin assignment*

*At next part as we use cyclone II for this project we must set our input map with circuit pin which its informations are avilable in manuall.*

*We get input(serin,rst,clkpb) from board buttons and see outputs(port,done,valid) on LEDs and data numbers on seven_segment:*




*The pins assign in bellow way:*

1) *Four red LED pins for port number*
2) *One switch pin for rst*
3) *One switch pin for serin*
4) *One button pin for clkpb*
5) *Two green LED pin for done and valid signal.*
6) *Default pins for seven_segment*

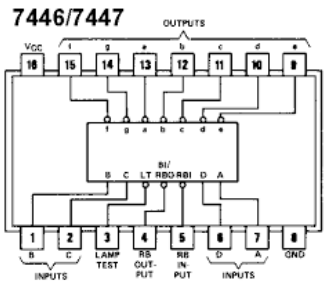



Fig3-SSD pins

## *Observation*

*At last by running synthesisable version of code on board we saw the result and went through this procedure:*

1) *Set rst to 1 for reseting all things*
2) *Set serin as 0 to start.*
3) *Set two serial 1 to set port 3.*
4) *Set two 0 and two 1 again to set data number 3 (means 3 bit)*
5) *By giving first bit of data valid LED turn on but seven_segment isnt active yet because it is in LDC state in controller.*
6) *By sending next bit seven_segment begin count down from 2 (because one bit of data already recieved)*
7) *At last when seven_segment reach to 0 we send 1 in serin again and waiting for rst because in this implementation the counter don not have internal restart signal*

*Here is one shot from our final circuit when SSD begin to count down (data num was 6):*