Lab 3

Milen Dimitrov

(Lecture 5/6)

FPGA Verilog

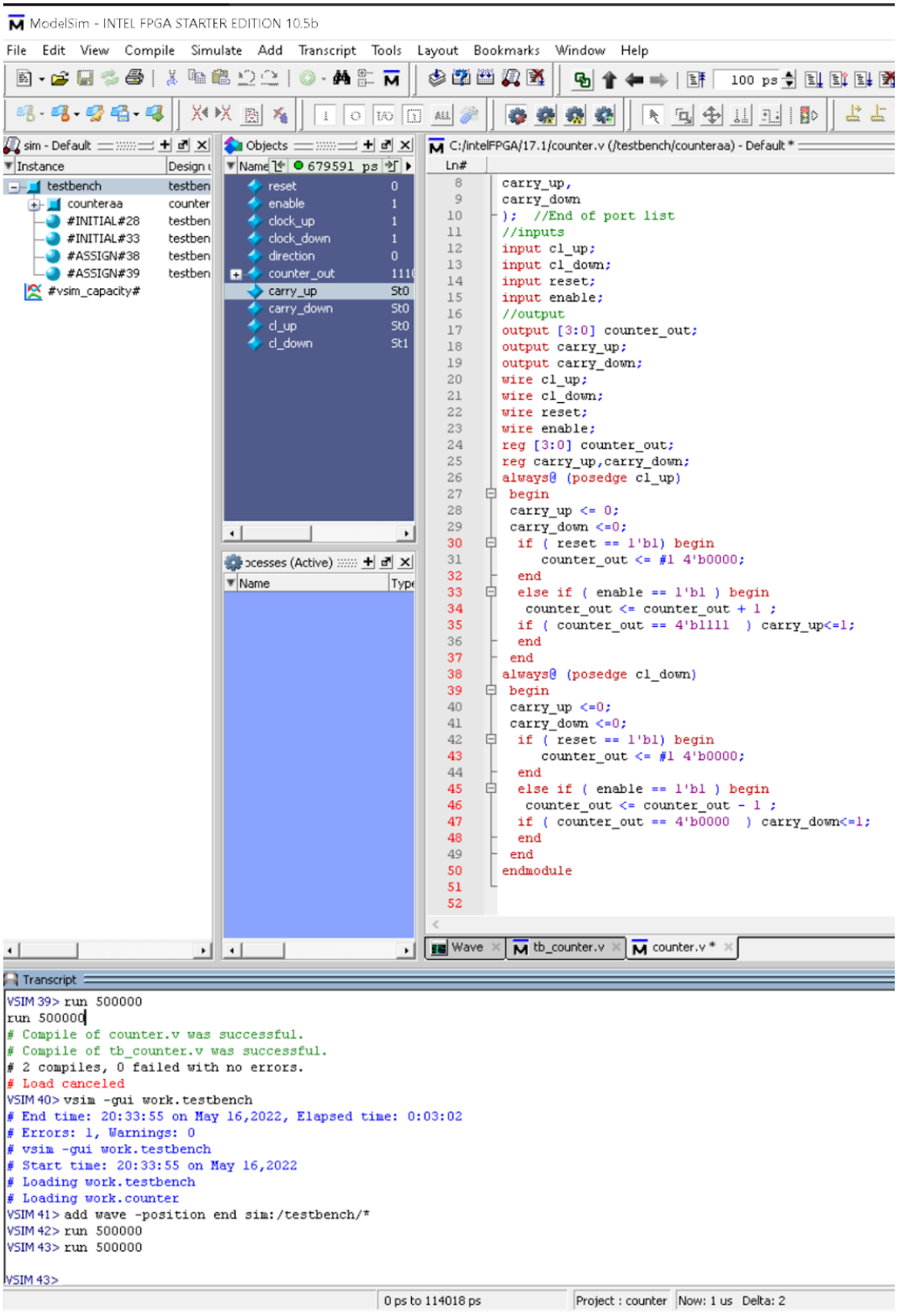
1. Use ModelSim

2. Design a 4 bit up-down counter

3. Simulate it in ModelSim.

4. Submit a single PDF file that contains the Verilog code and a screenshot of the output.

1. ModelSim simple version reversive counter



counter.v

```
`timescale 1ns/1ps
module counter(
cl_up ,
cl_down,
reset ,
enable ,
counter_out,
carry_up,
carry_down
);  //End of port list
//inputs
input cl_up;
input cl_down;
input reset;
input enable;
//output
output [3:0] counter_out;
output carry_up;
output carry_down;
wire cl_up;
wire cl_down;
wire reset;
wire enable;
reg [3:0] counter_out;
reg carry_up,carry_down;
always@ (posedge cl_up)  //counting up
 begin
 carry_up <= 0;
 carry_down <=0;
  if ( reset == 1'b1) begin            //reset to 0
     counter_out <= #1 4'b0000;
  end
  else if ( enable == 1'b1 ) begin
   counter_out <= counter_out + 1 ;
  if ( counter_out == 4'b1111  ) carry_up<=1;  //generating carry up when overflow
  end
 end
always@ (posedge cl_down)  //counting down
 begin
 carry_up <=0;
 carry_down <=0;
  if ( reset == 1'b1) begin
     counter_out <= #1 4'b0000;
  end
  else if ( enable == 1'b1 ) begin
   counter_out <= counter_out - 1 ;
  if ( counter_out == 4'b0000  ) carry_down<=1; //generating carry down
  end
 end
endmodule
```

tb_counter.v

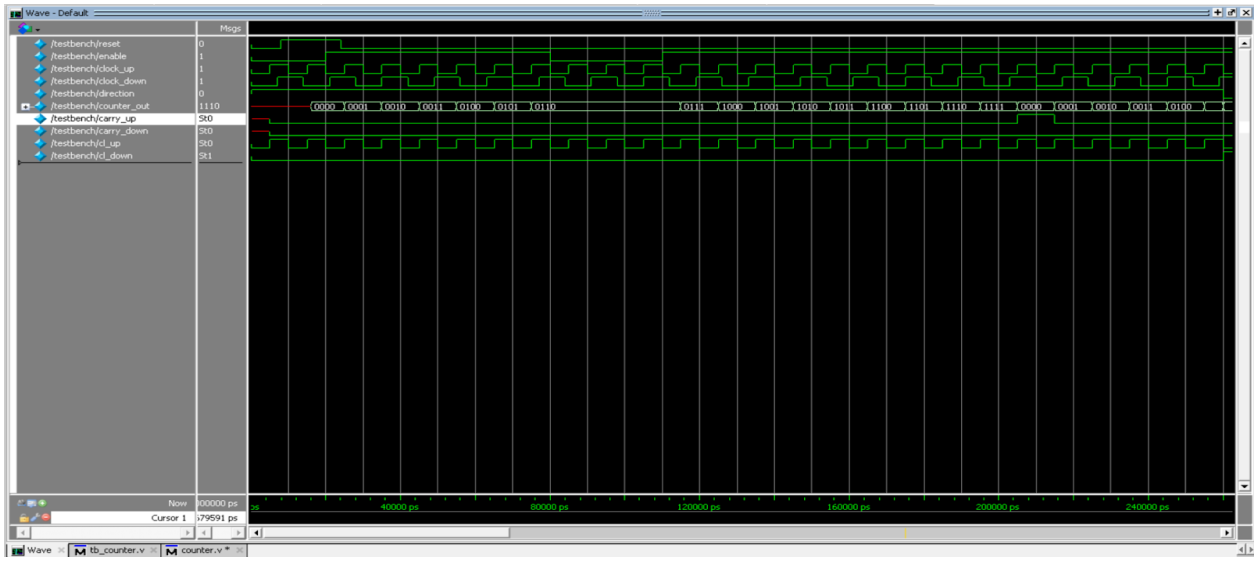
```
`timescale 1ns/1ps
module testbench ();
reg reset,enable,clock_up,clock_down ,direction;
wire [3:0] counter_out;
wire carry_up, carry_down,cl_up,cl_down;
```

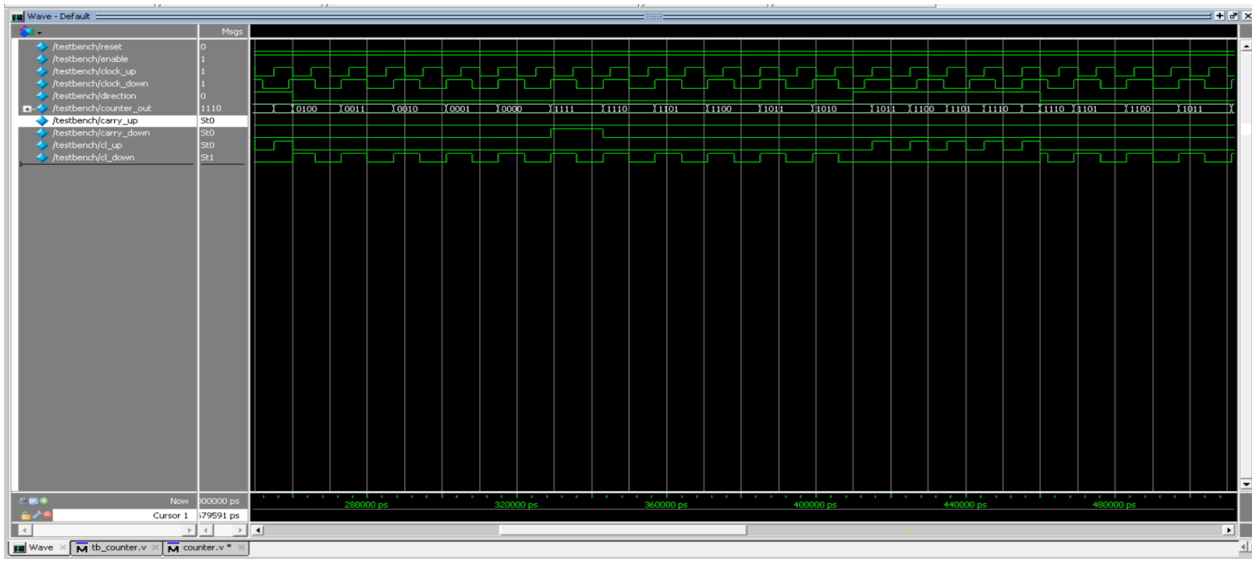
```
initial begin
        reset = 0;
        #8 reset = 1;
        #16 reset = 0 ;
end
initial begin
        enable = 0;
        direction = 1;
        #20 enable = 1;
        #60 enable = 0 ;
        #30 enable = 1;
        #150 direction = 0;
        #150 direction = 1;
        #50 direction = 0;
        #224 direction = 1;
end
initial begin
  clock_up = 0;
   forever  #5 clock_up = ~clock_up;
end
initial begin
  clock_down = 0;
   forever  #7 clock_down = ~clock_down;
end
assign cl_up=clock_up&direction;
assign cl_down=clock_down&(~direction);
counter counteraa(
        .cl_up (cl_up),
        .cl_down (cl_down),
        .reset (reset),
        .enable (enable),
        .counter_out (counter_out),
        .carry_up (carry_up),
        .carry_down (carry_down)
);
endmodule
```

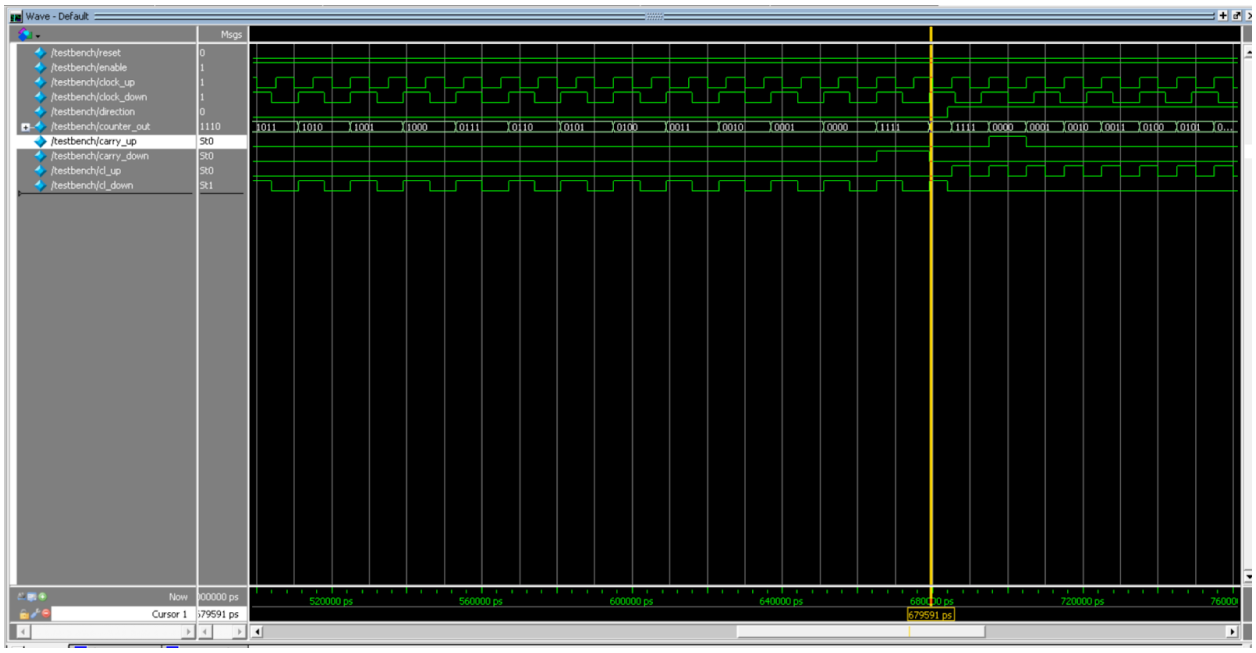
compiling and simulating:

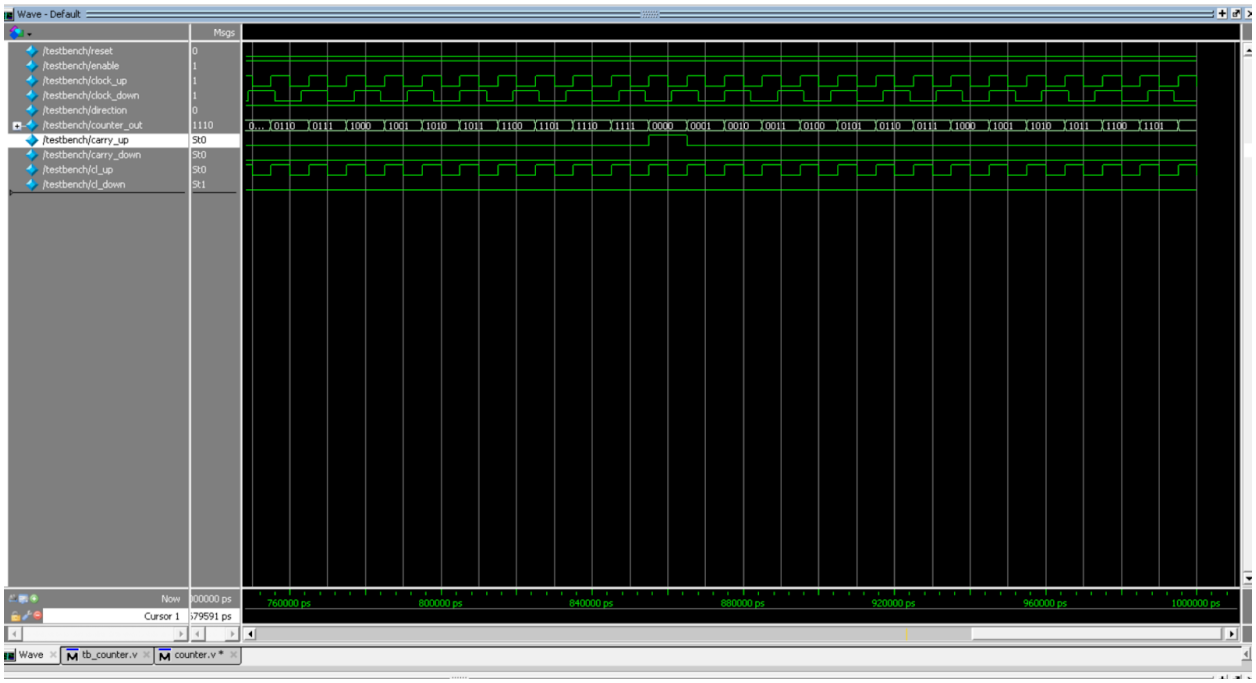
```
# Compile of counter.v was successful.
# Compile of tb_counter.v was successful.
# 2 compiles, 0 failed with no errors.
# Load canceled
vsim -gui work.testbench
# End time: 20:33:55 on May 16,2022, Elapsed time: 0:03:02
# Errors: 1, Warnings: 0
# vsim -gui work.testbench
# Start time: 20:33:55 on May 16,2022
# Loading work.testbench
# Loading work.counter
add wave -position end sim:/testbench/*
run 500000
run 500000
```

Here are the waveforms, below will zoom on some interesting moments



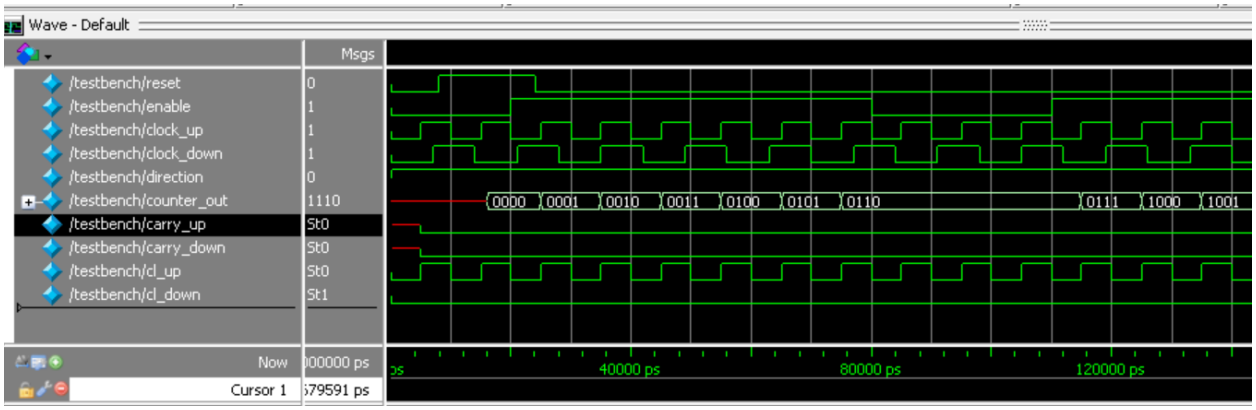




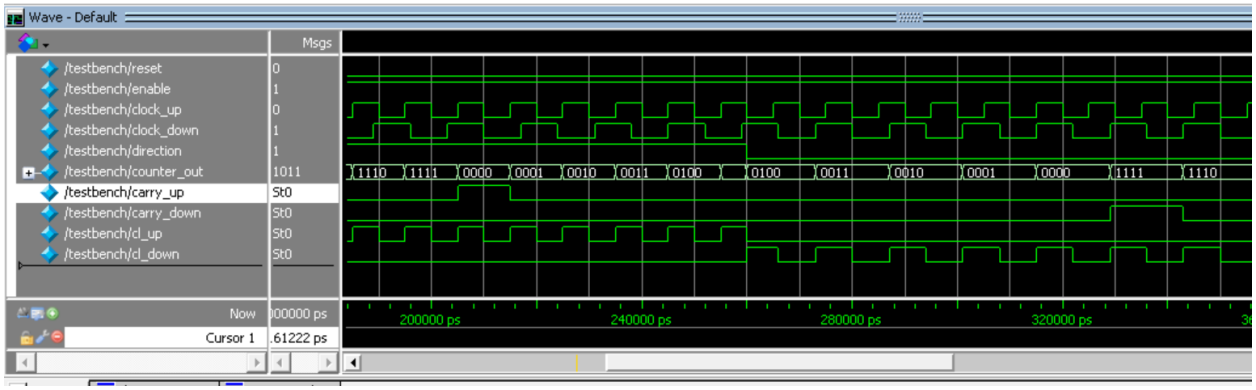


zoom-outs:

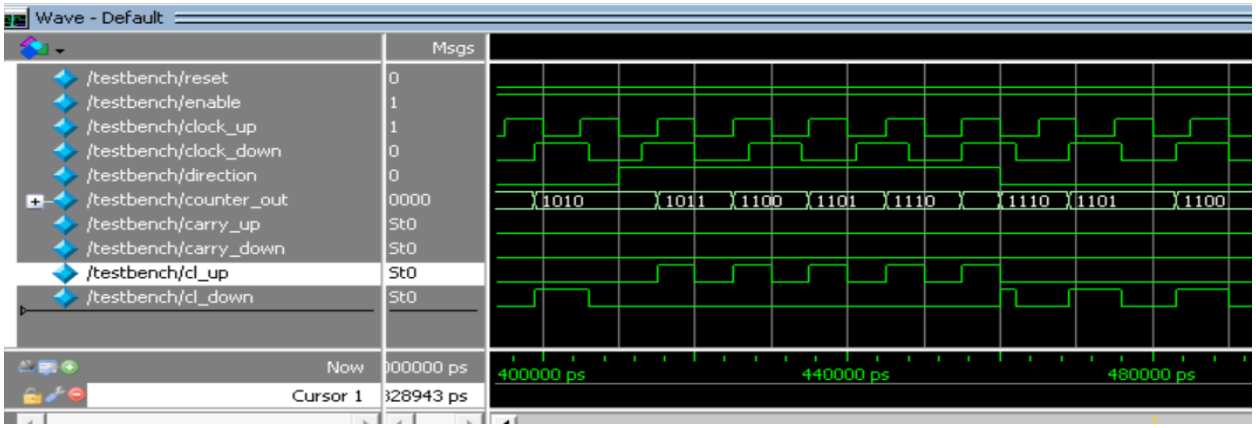
After the reset became 0 , the counter was set to 0000. Enable =1 let it count, when Enable=0 at 80ns, counter not count. Direction=1 - counting up , clock-up is enabled , clock_down is masked.



at 205ns counter overfill from 1111 to 0000 and carry_up becomes 1.|
at 260ns direction becomes 0, and clock is enabled to clock_down, clock_up is masked (&0)
at 329ns carry_down is produced because counter changed from 1111 to 0000



change the direction up and down around the critical points 1111/0000 to be sure no false carry-up or carry-down are generated



2. Real case 4 BCD counters with multiplexed dynamic 7-segment display

counter.v , the top module

```
`timescale 1ns/1ps

module counter(
output  [6:0] seg ,
output  [3:0] an,
output reg [13:8] LED,
output reg [7:0] JC,
output reg [7:0] JB,
output  [3:0] counter_out0,
output  [3:0] counter_out1,
output  carry_up,
output  carry_down,
input cl_up ,
input cl_down,
input reset ,
input enable,
input sw1,
input sw0,
input sw2,
input sw3,
input direction,
input  [7:0] JA,
input [11:2] sw,
input clk);

wire cl;
reg  carry_up_internal,carry_down_internal ;
wire  [3:0] counter_out2;
wire  [3:0] counter_out3;
reg  [26:0] i;

//generating different frequencies on LEDs and JB/JC by dividing
//100MHz by 2 each bit. the lowest frequency on LED13 and JB7 is
// 100MHz/2^27 0.745Hz, 1.34 sec period
initial i=0;
always@(posedge clk) begin
LED<=i[26:21];
JB<=i[24:17];
JC<=i[16:9];
i<=i+1;
end

//last 2 LEDs indicate CARRY-UP and CARRY_DOWN , selecting which counter output
// will be routed to the LED by sw0-sw3
assign carry_up= carry_up0&sw0 |carry_up1&sw1 |carry_up2&sw2 |carry_up3&sw3 ;
assign carry_down=carry_down0&sw0|carry_down1&sw1|carry_down2&sw2|carry_down3&sw3  ;
assign cl_up_int=cl_up  | (~JA[1])    |direction&((~JA[0])|JB[7]&sw[2]|

//creating composite input with selector - fron buttons Up and Down, or external
```

```
//  JA[1] for count up ,JA[2] count down, or JA[0] - reversible ,or internal
//connections to JB/JC controlled by sw[2]-sw[11]
        JB[6]&sw[3]|JB[5]&sw[4]|JB[4]&sw[5]|JB[3]&sw[6]|JB[2]&sw[7]|
        JB[1]&sw[8]|JB[0]&sw[9]|JC[7]&sw[10]|JC[6]&sw[11]);
assign cl_down_int=cl_down|(~JA[2])|(~direction)&((~JA[0])|JB[7]&sw[2]|
        JB[6]&sw[3]|JB[5]&sw[4]|JB[4]&sw[5]|JB[3]&sw[6]|JB[2]&sw[7]|
        JB[1]&sw[8]|JB[0]&sw[9]|JC[7]&sw[10]|JC[6]&sw[11]) ;

//4 instances of  BCD counters, cascades by carry-up and carry-down
bcd bcd0 (counter_out0,carry_up0,carry_down0,cl_up_int,cl_down_int,reset,enable);
bcd bcd1 (counter_out1,carry_up1,carry_down1,carry_up0,carry_down0,reset,enable);
bcd bcd2 (counter_out2,carry_up2,carry_down2,carry_up1,carry_down1,reset,enable);
bcd bcd3 (counter_out3,carry_up3,carry_down3,carry_up2,carry_down2,reset,enable);
//decoder and multiplexer for dynamic 7-seg LED
decoder dec (  seg ,  an, counter_out0,counter_out1,counter_out2,counter_out3 , clk ) ;
endmodule
```

bcd.v - the 4bit bcd counter

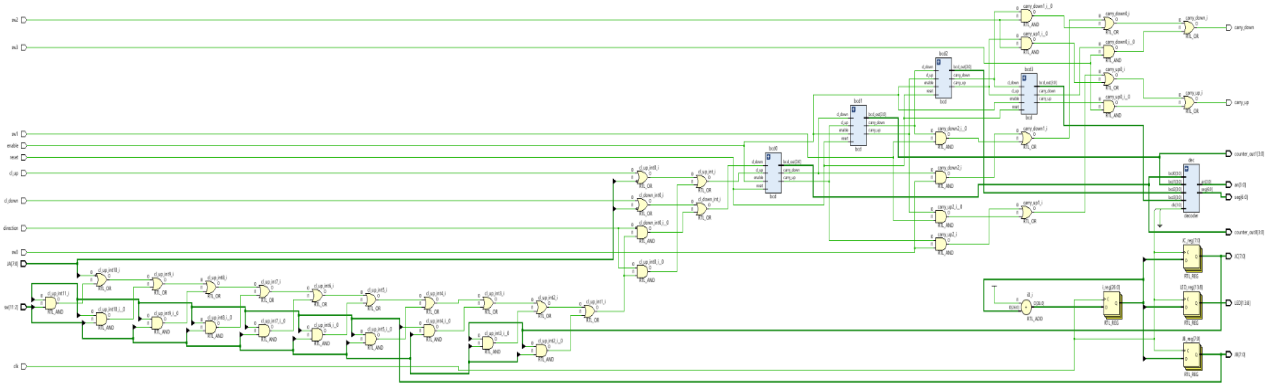
```
`timescale 1ns / 1ps
 module bcd(
 output reg [3:0] bcd_out,
 output reg carry_up,
 output reg carry_down,
 input cl_up ,
 input cl_down,
 input reset ,
 input enable
);

wire c;
assign cl=cl_up | cl_down ;

always@ (posedge cl or posedge  reset )
begin
  if ( reset == 1'b1)
      begin
      bcd_out <=  4'b0000;
      carry_up<=0;
      carry_down<=0;
      end
  else if ( enable == 1'b1 )
      begin
         if ( ~cl_down  )   //count up
           begin
           bcd_out <= bcd_out + 1 ;
           if ( bcd_out == 9  )  // generate carry up
               begin
               carry_up<=1 ;
               carry_down <=0;
               bcd_out<=4'b0000;
               end
           else
               begin
               carry_up <=0;
               carry_down <=0;
```

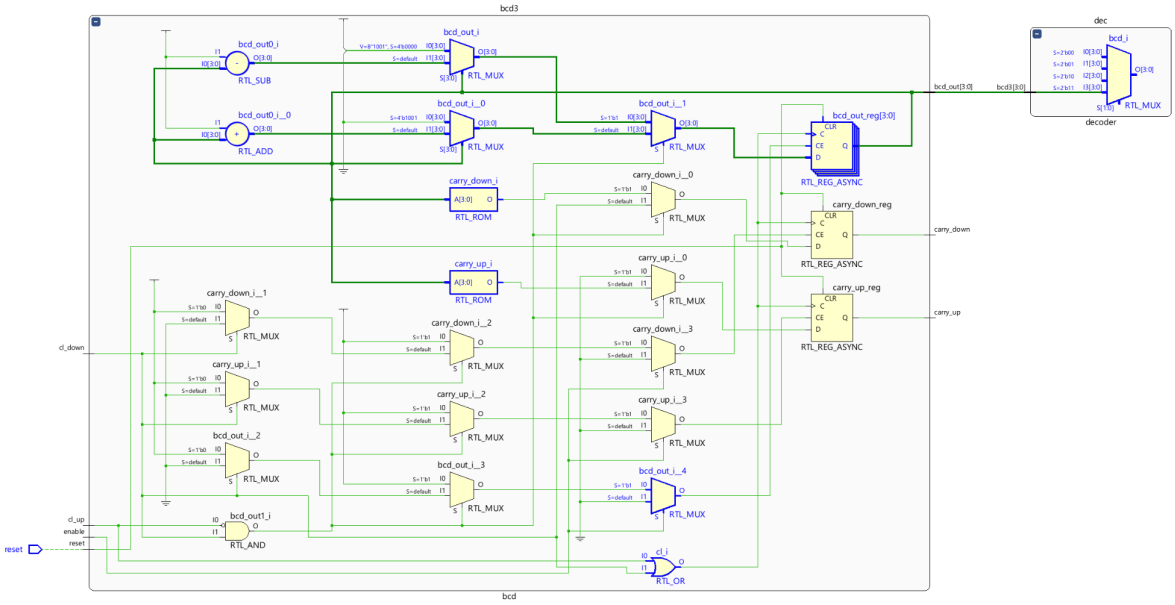
```
                end
            end
          if ( ~cl_up & cl_down)    // dount down
            begin
            bcd_out <= bcd_out - 1 ;
            if ( bcd_out == 4'b0000  )  //generate carry down
                 begin
                 carry_down<=1;
                 carry_up <=0;
                 bcd_out <= 4'b1001;
                 end
            else
                begin
                carry_down <=0;
                carry_up <=0;
                end
            end
  end
end

endmodule
```

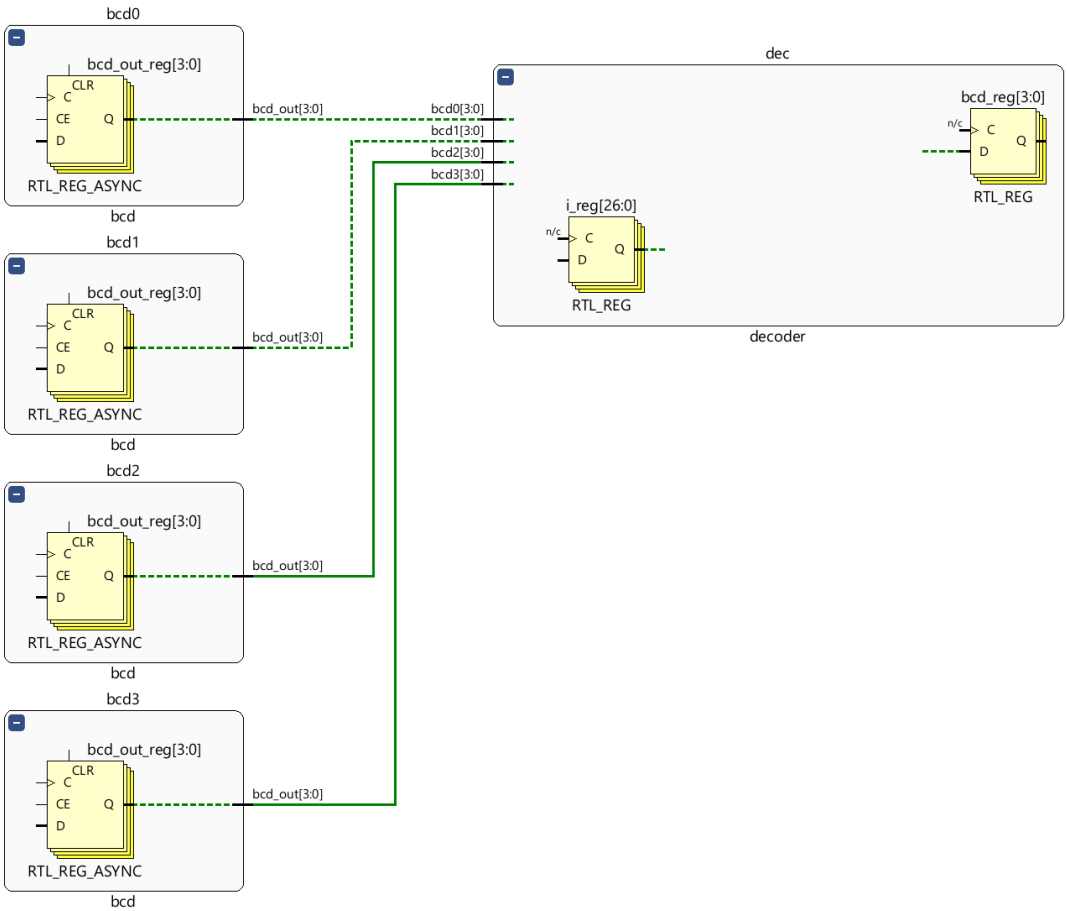
decoder.v - 7-seg decoder and 4 way 4 bits multiplexer

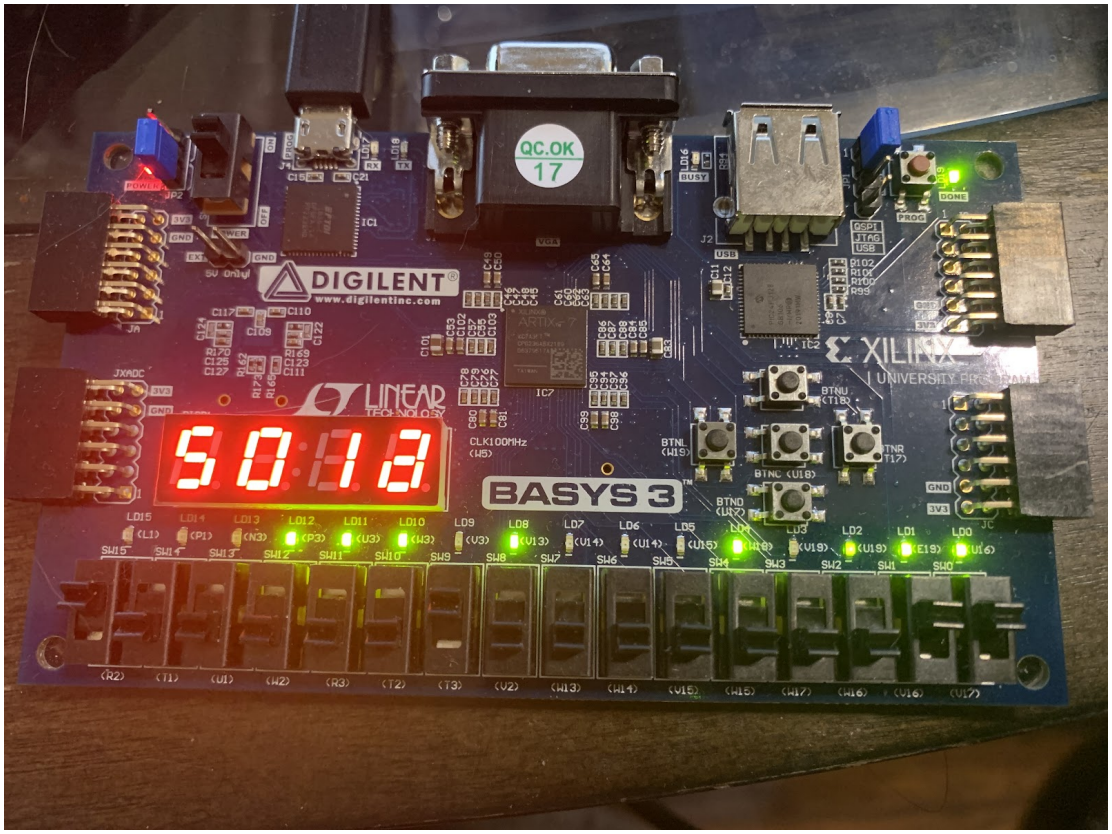
```
`timescale 1ns / 1ps
module  decoder (
output reg [6:0] seg ,
output reg [3:0] an,
input [3:0] bcd0,
input [3:0] bcd1,
input [3:0] bcd2,
input [3:0] bcd3 ,
input [1:0] clk );
  reg [3:0] bcd;
  reg  [26:0] i;

initial i=0;
always @(posedge clk) //multiplexer
    begin
    i<=i+1;
    case (  i[19:18] )
       0: begin  an<=14;  bcd<=bcd0;     end
       1: begin  an<=13;  bcd<=bcd1;     end
       2: begin  an<=11;  bcd<=bcd2;     end
       3: begin  an<=7;  bcd<=bcd3;     end
    endcase
    end

  always@(bcd)   //7seg decoder
  begin
  case (bcd) //case statement
            0 : seg = 7'b1000000;
            1 : seg = 7'b1111001;
            2 : seg = 7'b0100100;
            3 : seg = 7'b0110000;
            4 : seg = 7'b0011001;
```

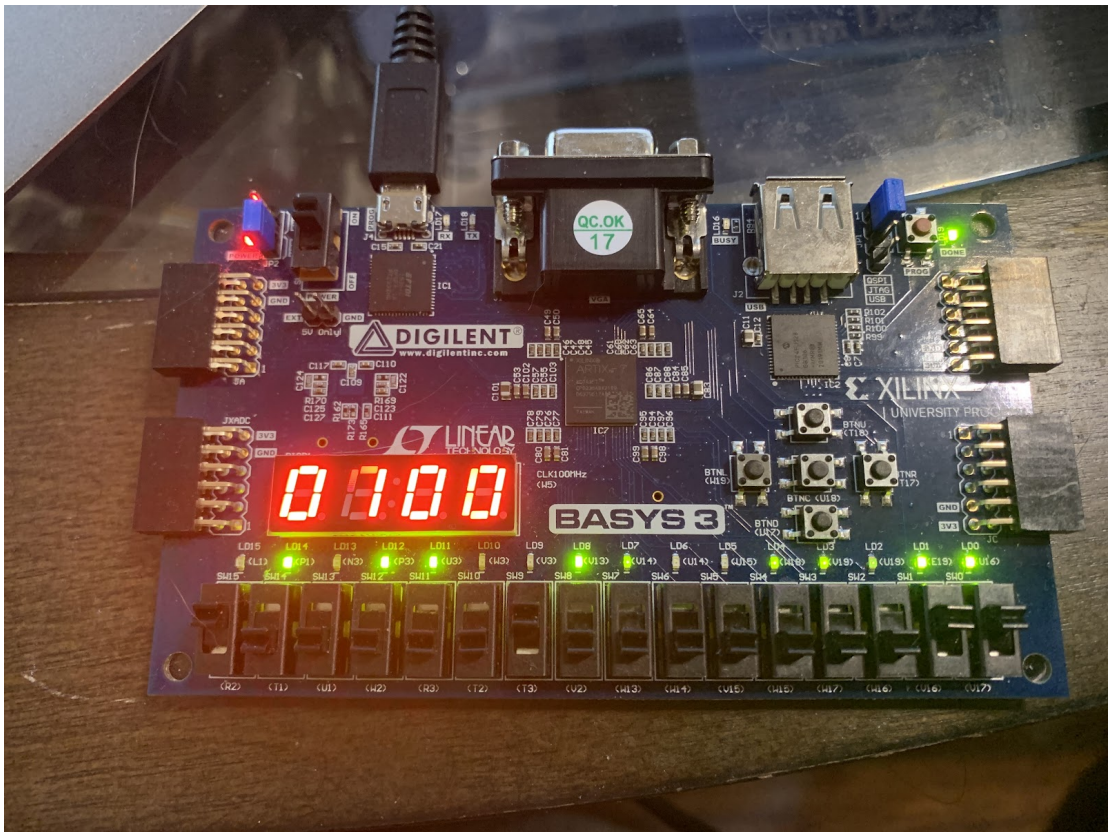
```
            5 : seg = 7'b0010010;
            6 : seg = 7'b0000010;
            7 : seg = 7'b1111000;
            8 : seg = 7'b0000000;
            9 : seg = 7'b0010000;
            10 : seg = 7'b0001000;
            11 : seg = 7'b0000011;
            12 : seg = 7'b1000110;
            13 : seg = 7'b0100001;
            14 : seg = 7'b0000110;
            15 : seg = 7'b0001110;
            default : seg = 7'b1111111;
  endcase
  end
endmodule
```

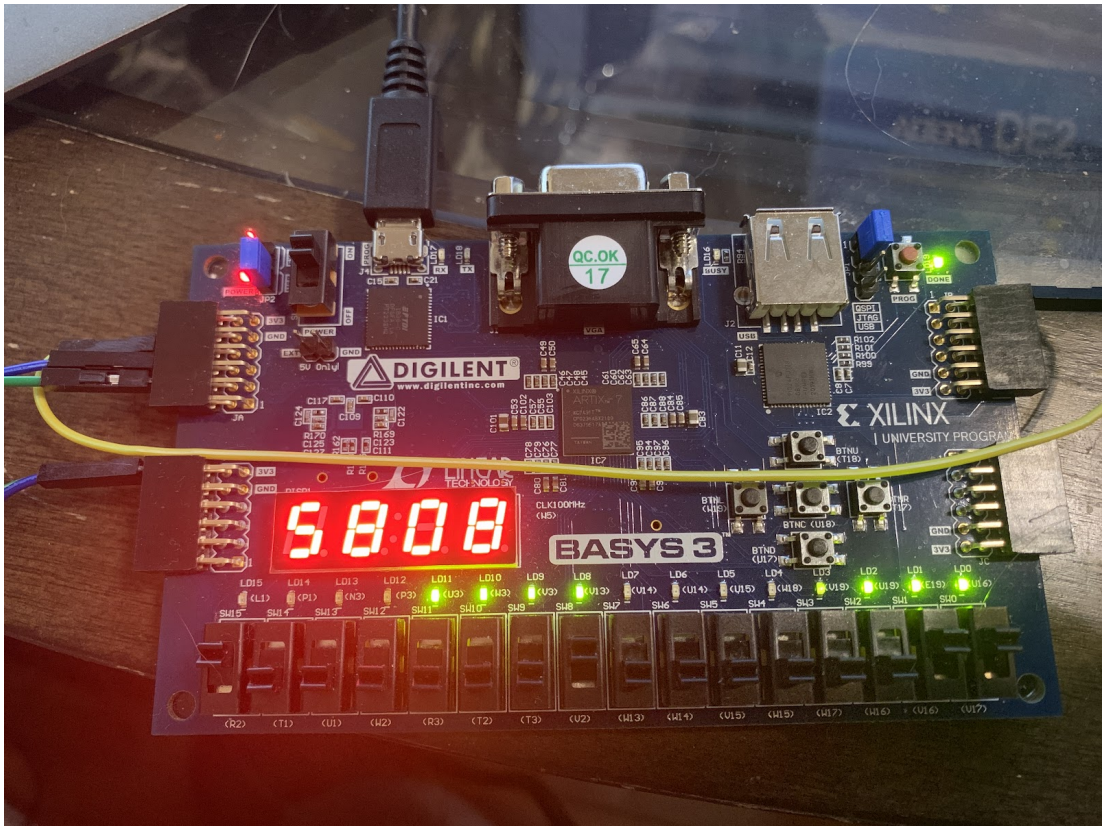



bcd3
bcd_out0_i
RTL_SUB
bcd_out0_i__0
RTL_ADD
bcd_out_i
RTL_MUX
bcd_out_i__0
RTL_MUX
bcd_out_i__1
RTL_MUX
bcd_out_reg[3:0]
RTL_REG_ASYNC
carry_down_i
RTL_ROM
carry_down_i__0
RTL_MUX
carry_down_reg
RTL_REG_ASYNC
carry_down
carry_up_i
RTL_ROM
carry_up_i__0
RTL_MUX
carry_up_reg
RTL_REG_ASYNC
carry_up
carry_down_i__1
RTL_MUX
carry_down_i__2
RTL_MUX
carry_down_i__3
RTL_MUX
carry_up_i__1
RTL_MUX
carry_up_i__2
RTL_MUX
carry_up_i__3
RTL_MUX
bcd_out_i__2
RTL_MUX
bcd_out_i__3
RTL_MUX
bcd_out_i__4
RTL_MUX
bcd_out1_i
RTL_AND
cl_i
RTL_OR
cl_down
cl_up
enable
reset
bcd_out[3:0]
bcd[3:0]
dec
bcd_i
RTL_MUX
decoder
bcd


bcd0
bcd_out_reg[3:0]
RTL_REG_ASYNC
bcd_out[3:0]
bcd
bcd1
bcd_out_reg[3:0]
RTL_REG_ASYNC
bcd_out[3:0]
bcd
bcd2
bcd_out_reg[3:0]
RTL_REG_ASYNC
bcd_out[3:0]
bcd
bcd3
bcd_out_reg[3:0]
RTL_REG_ASYNC
bcd_out[3:0]
bcd
dec
bcd0[3:0]
bcd1[3:0]
bcd2[3:0]
bcd3[3:0]
bcd_reg[3:0]
RTL_REG
i_reg[26:0]
RTL_REG
decoder


QC.OK
17
DIGILENT
www.digilentinc.com
LINEAR TECHNOLOGY
XILINX
BASYS 3
5012
CLK100MHz
(W5)


QC.OK
17
DIGILENT
www.digilentinc.com
LINEAR TECHNOLOGY
XILINX
BASYS 3
0100
CLK100MHz
(W5)


QC.OK
17
DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGRAM
BASYS 3
5808