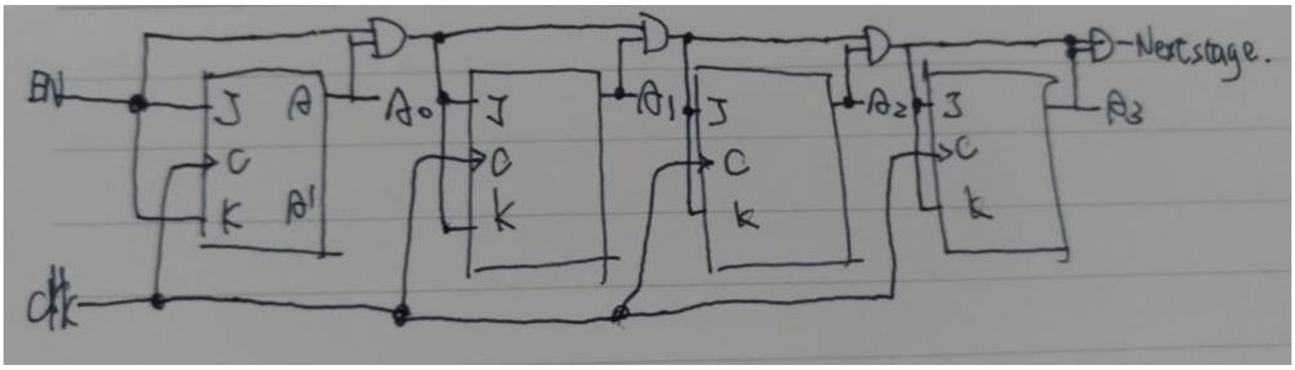
Lab03 counters 106061146 陳兆廷

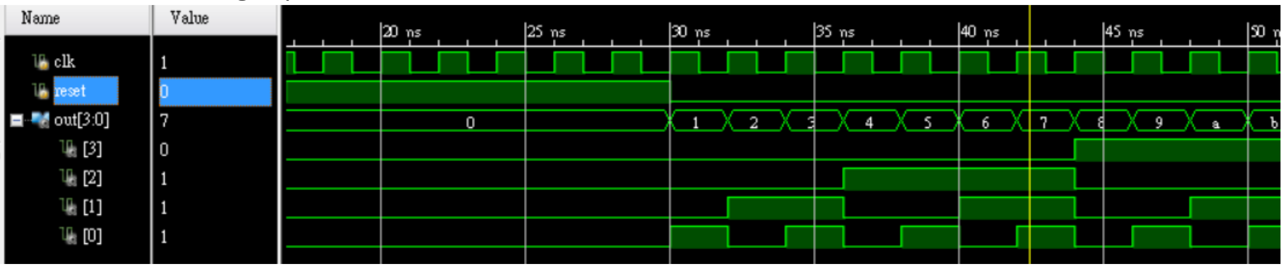
Pre-labs:

1. 4-bit synchronous binary up counter:

   1.1 Logic diagram:



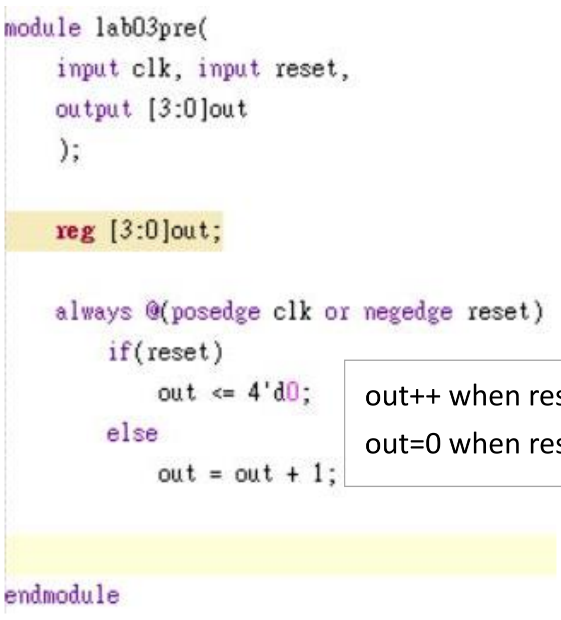
   1.2 Verilog representation:



Name	Value
clk	1
reset	0
out[3:0]	7
[3]	0
[2]	1
[1]	1
[0]	1

Verilog codes:

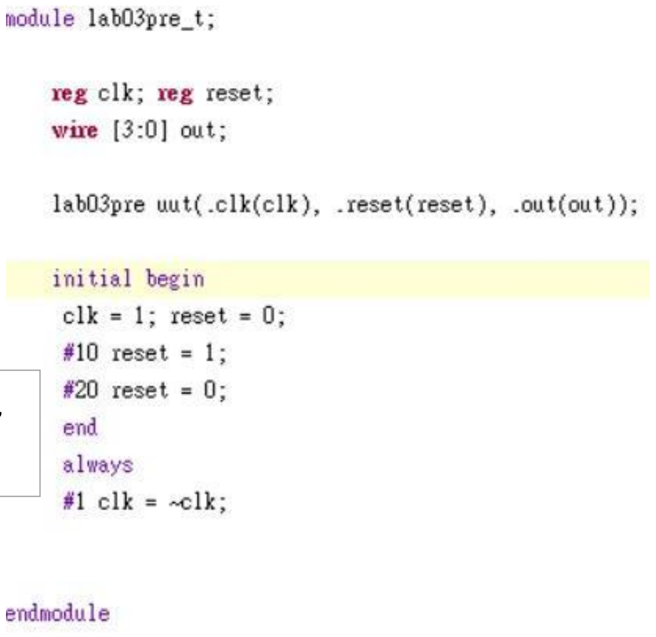
Module:



```
module lab03pre(
    input clk, input reset,
    output [3:0]out
    );

    reg [3:0]out;

    always @(posedge clk or negedge reset)
        if(reset)
            out <= 4'd0;
        else
            out = out + 1;

endmodule
```

out++ when reset!=1,
out=0 when reset = 1

Testbench:



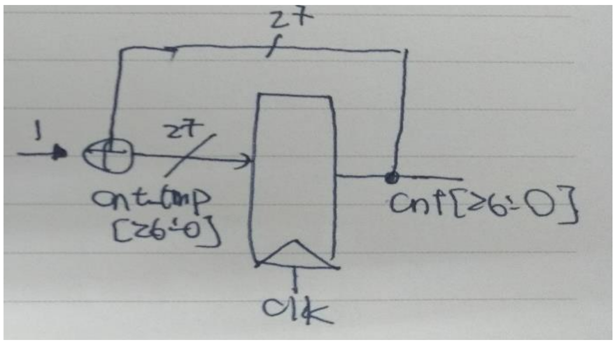
```
module lab03pre_t;

    reg clk; reg reset;
    wire [3:0] out;

    lab03pre uut(.clk(clk), .reset(reset), .out(out));

    initial begin
     clk = 1; reset = 0;
     #10 reset = 1;
     #20 reset = 0;
     end
     always
     #1 clk = ~clk;

endmodule
```

Experiments:

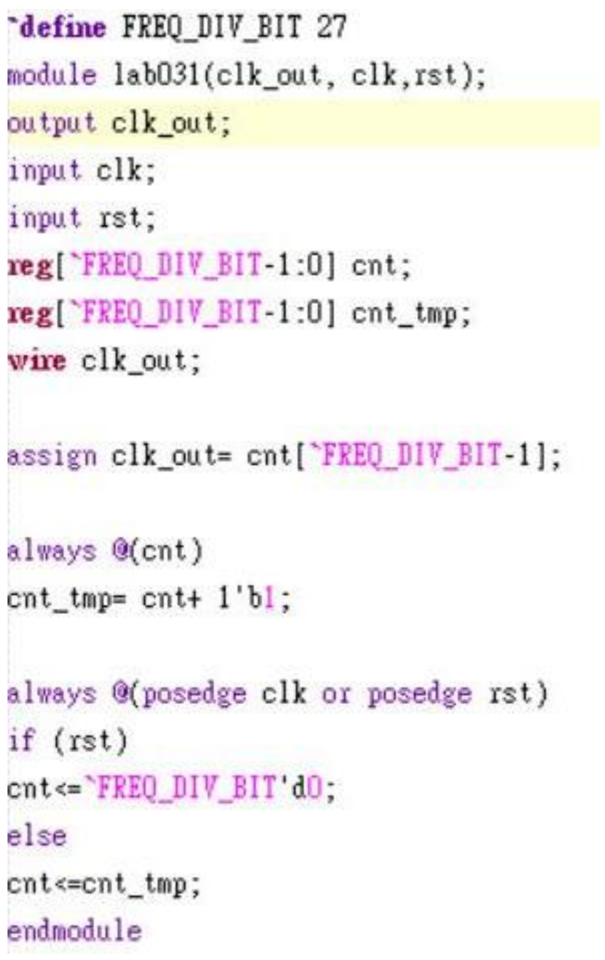
1. Frequency divider:

   1.1 specification: a circuit that takes an input signal of a frequency f, and generates an output signal of a frequency fout = f/n, where n is an integer. The n is 2^27 in this case.

1.2 Block diagram:



1.3 Implementation:



```
`define FREQ_DIV_BIT 27
module lab031(clk_out, clk,rst);
output clk_out;
input clk;
input rst;
reg[`FREQ_DIV_BIT-1:0] cnt;
reg[`FREQ_DIV_BIT-1:0] cnt_tmp;
wire clk_out;

assign clk_out= cnt[`FREQ_DIV_BIT-1];

always @(cnt)
cnt_tmp= cnt+ 1'b1;

always @(posedge clk or posedge rst)
if (rst)
cnt<=`FREQ_DIV_BIT'd0;
else
cnt<=cnt_tmp;
endmodule
```

One output signal(clk_out)

Input clock and reset(clk, rst)

a parameter [26:0]cnt to count to 2^27

temp parameter

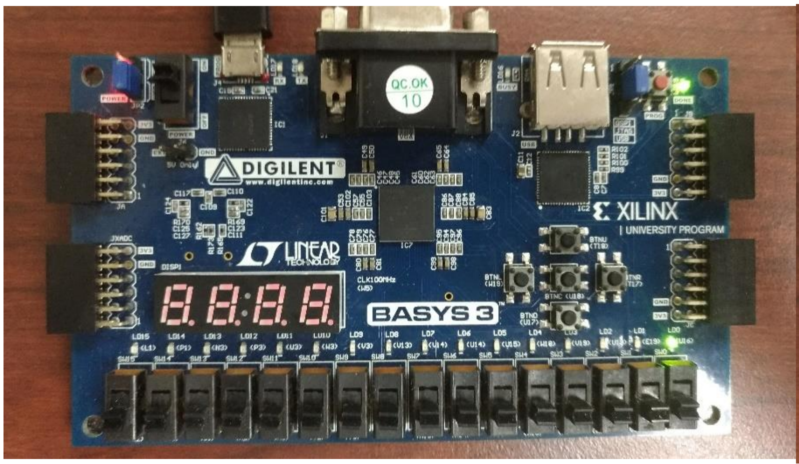
the output signal is the 26th bit of “count”

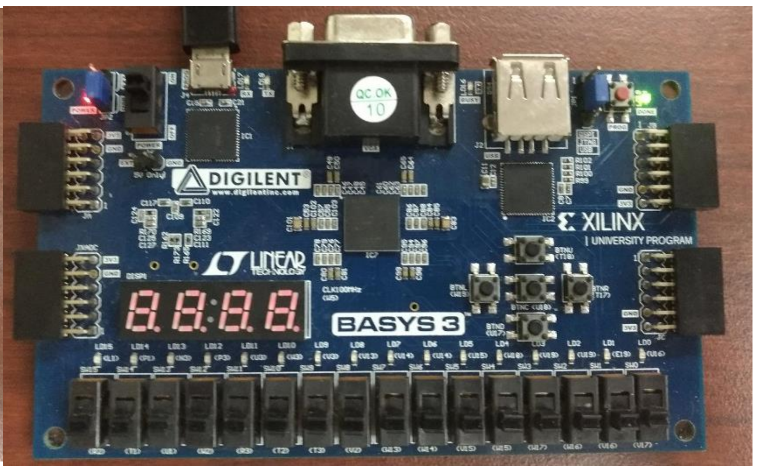
count keeps ++

if reset == 1, count returns to 0

else count ++

FPGA:

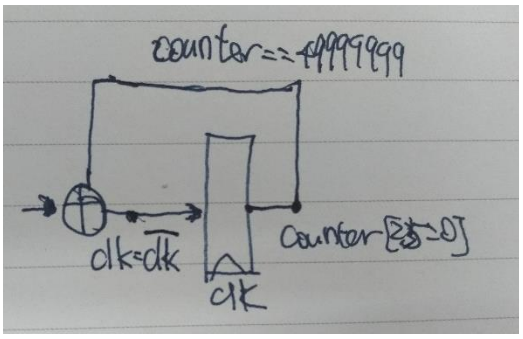




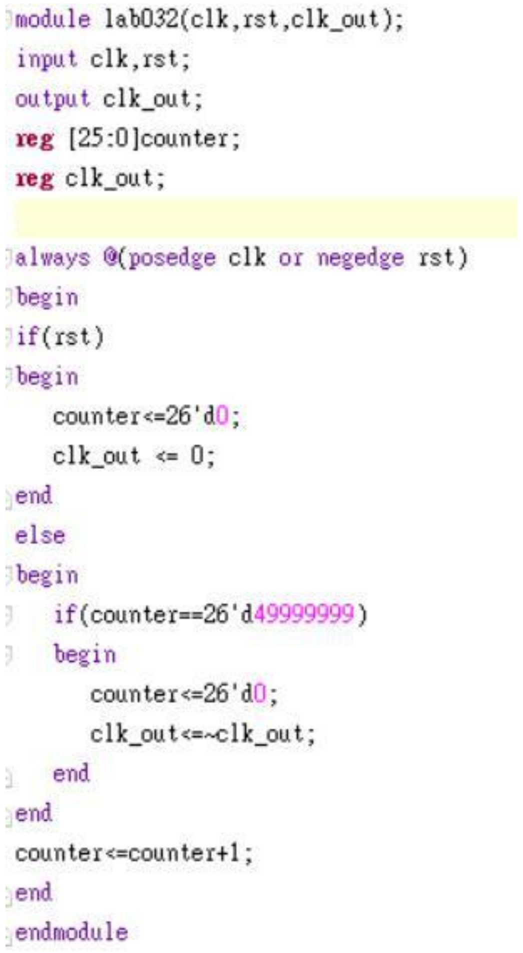
2. Count-for-50M frequency divider:

2.1 Specification: a circuit that takes an input signal of a frequency f, and generates an output signal of a frequency: fout = f/n, where n is an integer. The n is 50M in this case.

2.2 Block diagram:



2.3 Implementation:



```
module lab032(clk,rst,clk_out);
input clk,rst;
output clk_out;
reg [25:0]counter;
reg clk_out;

always @(posedge clk or negedge rst)
begin
if(rst)
begin
   counter<=26'd0;
   clk_out <= 0;
end
else
begin
   if(counter==26'd49999999)
   begin
      counter<=26'd0;
      clk_out<=~clk_out;
   end
end
counter<=counter+1;
end
endmodule
```

Counter counts to 2^26(67108864>50000000)

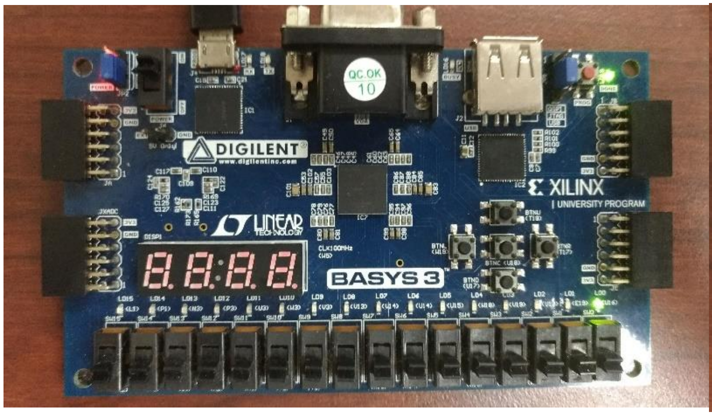
If reset == 1, then counter returns to 0.

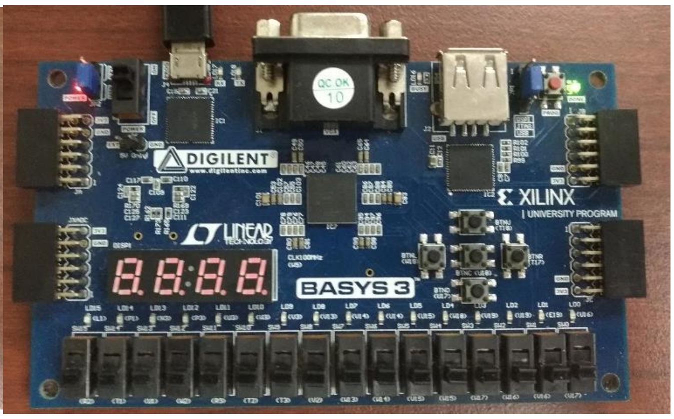
If counter counts to 49999999(from 0)

Counter returns to 0.

clk_out = ~clk_out

counter ++ (always)





3. 4-bit synchronous binary up counter:

```
module lab033(
    input clk,
    input rst,
    output [3:0]b
    );

    reg [26:0]counter;
    reg [3:0]b;

    always @(posedge clk or negedge rst)
    begin
    if(rst)
    begin
       counter<=27'd0;
       b <= 4'd0;
    end
    else
    begin
       if(counter==27'd49999999)
       begin
          counter<=27'd0;
          b <= b + 4'd1;
       end
    end
    counter<=counter+1;
    end
endmodule
```

input clock and reset

output a 4-bit b

create a 27-bit counter

if reset is 1

counter return to 0

b return to 0

if reset is 0

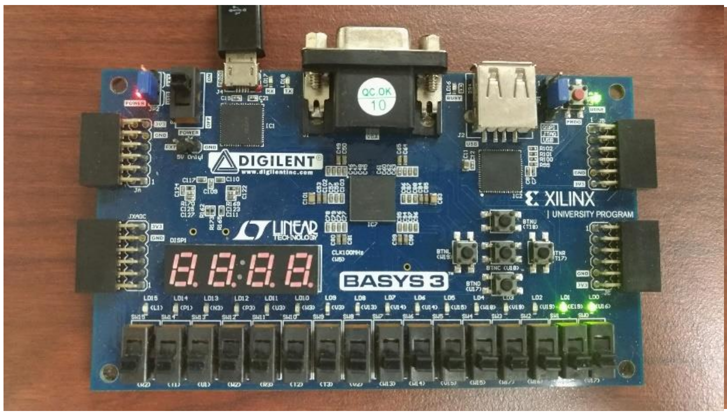
if counter added to 49999999

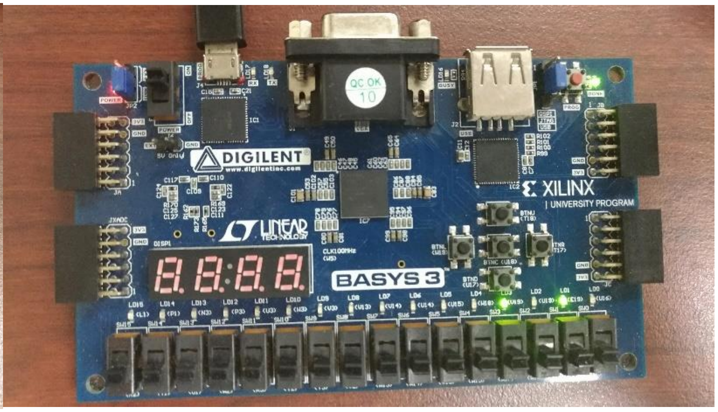
counter return to 0

b ++

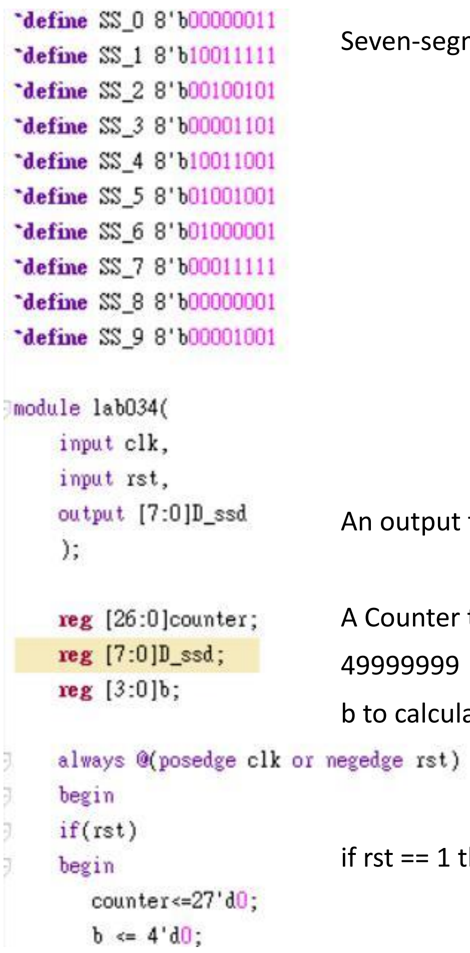
counter ++ (if reset == 0)

FPGA:





4. Single digit BCD up counter display on seven-segment display



```
`define SS_0 8'b00000011
`define SS_1 8'b10011111
`define SS_2 8'b00100101
`define SS_3 8'b00001101
`define SS_4 8'b10011001
`define SS_5 8'b01001001
`define SS_6 8'b01000001
`define SS_7 8'b00011111
`define SS_8 8'b00000001
`define SS_9 8'b00001001

module lab034(
    input clk,
    input rst,
    output [7:0]D_ssd
    );

    reg [26:0]counter;
    reg [7:0]D_ssd;
    reg [3:0]b;

    always @(posedge clk or negedge rst)
    begin
    if(rst)
    begin
       counter<=27'd0;
       b <= 4'd0;
```

Seven-segment display code

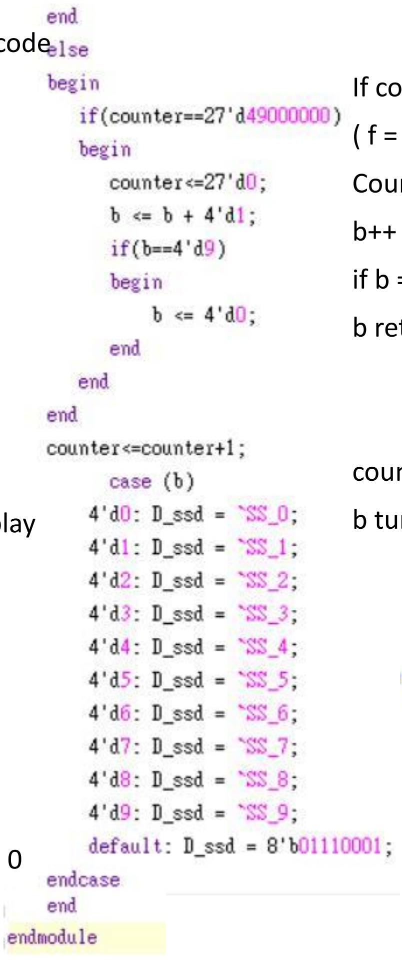
An output for 7-seg display

A Counter to count to

49999999

b to calculate 1~10

if rst == 1 then return to 0



```
end
else
begin
   if(counter==27'd49000000)
   begin
      counter<=27'd0;
      b <= b + 4'd1;
      if(b==4'd9)
      begin
          b <= 4'd0;
      end
   end
end
counter<=counter+1;
      case (b)
    4'd0: D_ssd = `SS_0;
    4'd1: D_ssd = `SS_1;
    4'd2: D_ssd = `SS_2;
    4'd3: D_ssd = `SS_3;
    4'd4: D_ssd = `SS_4;
    4'd5: D_ssd = `SS_5;
    4'd6: D_ssd = `SS_6;
    4'd7: D_ssd = `SS_7;
    4'd8: D_ssd = `SS_8;
    4'd9: D_ssd = `SS_9;
    default: D_ssd = 8'b01110001;
endcase
end
endmodule
```

If counter == 49999999

(f = 50M)

Counter return to 0

b++

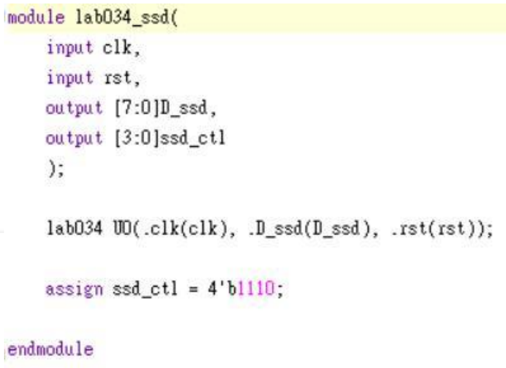
if b == 9

b return to 0

counter ++

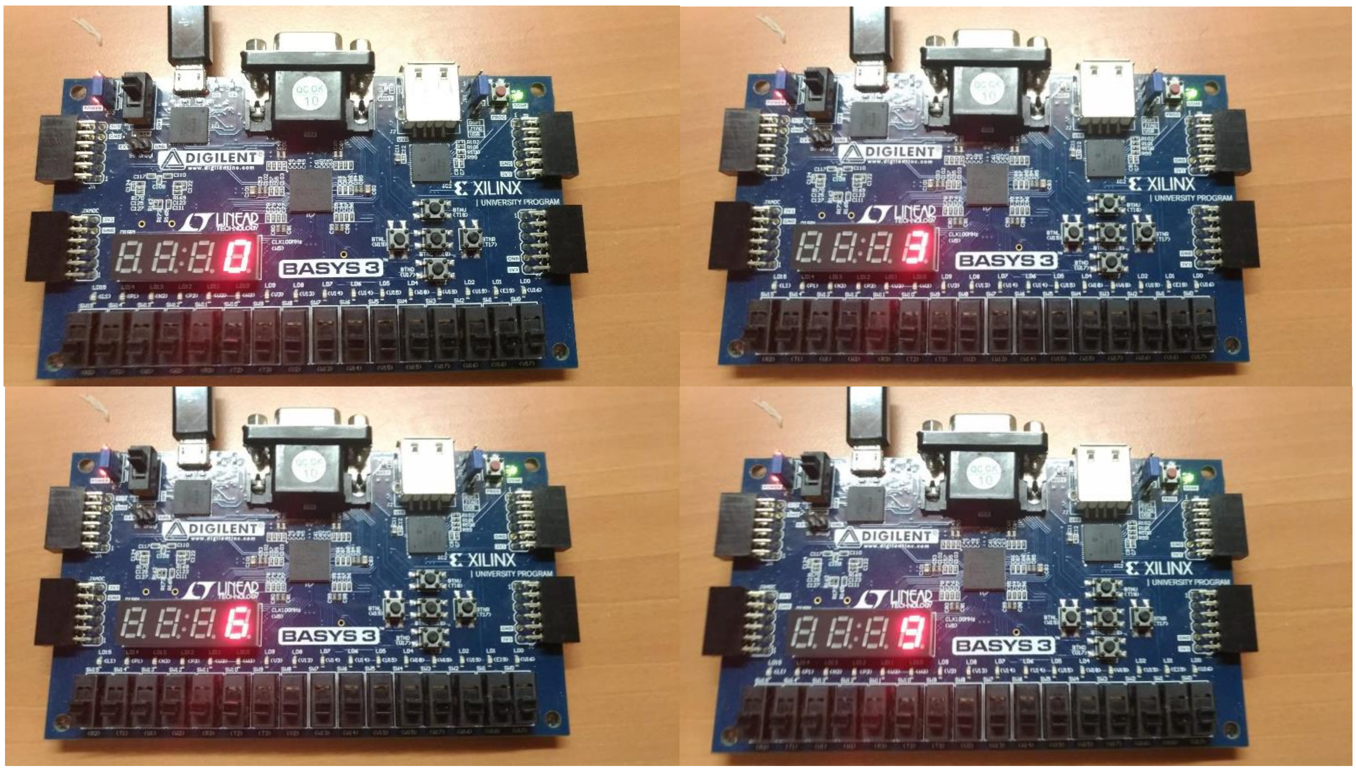
b turn to 7-seg display

Display module:

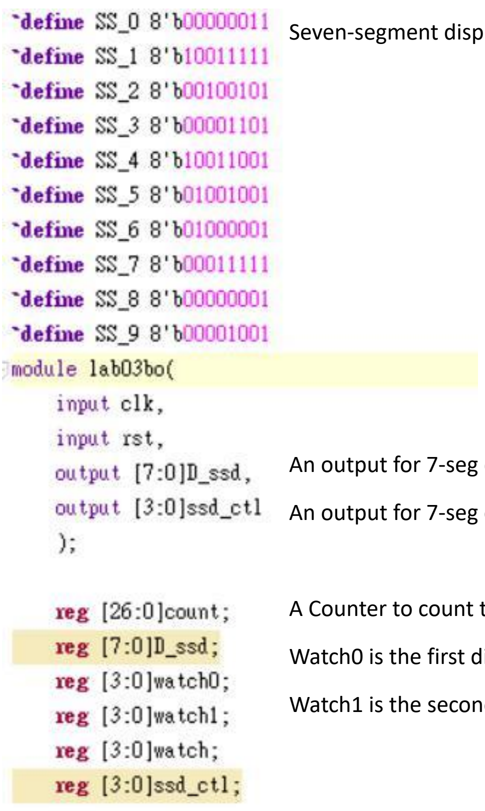


```
module lab034_ssd(
    input clk,
    input rst,
    output [7:0]D_ssd,
    output [3:0]ssd_ctl
    );

    lab034 U0(.clk(clk), .D_ssd(D_ssd), .rst(rst));

    assign ssd_ctl = 4'b1110;

endmodule
```

FPGA:



Bonus: 30-second count-down timer:



```verilog
`define SS_0 8'b00000011
`define SS_1 8'b10011111
`define SS_2 8'b00100101
`define SS_3 8'b00001101
`define SS_4 8'b10011001
`define SS_5 8'b01001001
`define SS_6 8'b01000001
`define SS_7 8'b00011111
`define SS_8 8'b00000001
`define SS_9 8'b00001001
module lab03bo(
    input clk,
    input rst,
    output [7:0]D_ssd,
    output [3:0]ssd_ctl
    );

    reg [26:0]count;
    reg [7:0]D_ssd;
    reg [3:0]watch0;
    reg [3:0]watch1;
    reg [3:0]watch;
    reg [3:0]ssd_ctl;
```

Seven-segment display code

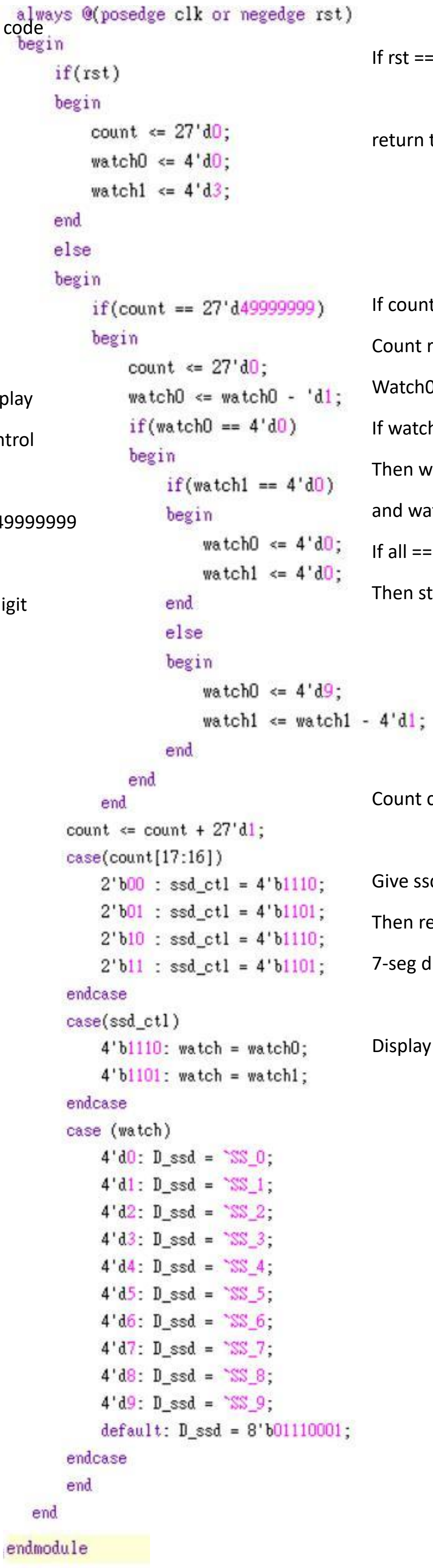
An output for 7-seg display

An output for 7-seg control

A Counter to count to 49999999

Watch0 is the first digit

Watch1 is the second digit



```verilog
always @(posedge clk or negedge rst)
begin
    if(rst)
    begin
        count <= 27'd0;
        watch0 <= 4'd0;
        watch1 <= 4'd3;
    end
    else
    begin
        if(count == 27'd49999999)
        begin
            count <= 27'd0;
            watch0 <= watch0 - 'd1;
            if(watch0 == 4'd0)
            begin
                if(watch1 == 4'd0)
                begin
                    watch0 <= 4'd0;
                    watch1 <= 4'd0;
                end
                else
                begin
                    watch0 <= 4'd9;
                    watch1 <= watch1 - 4'd1;
                end
            end
        end
    count <= count + 27'd1;
    case(count[17:16])
        2'b00 : ssd_ctl = 4'b1110;
        2'b01 : ssd_ctl = 4'b1101;
        2'b10 : ssd_ctl = 4'b1110;
        2'b11 : ssd_ctl = 4'b1101;
    endcase
    case(ssd_ctl)
        4'b1110: watch = watch0;
        4'b1101: watch = watch1;
    endcase
    case (watch)
        4'd0: D_ssd = `SS_0;
        4'd1: D_ssd = `SS_1;
        4'd2: D_ssd = `SS_2;
        4'd3: D_ssd = `SS_3;
        4'd4: D_ssd = `SS_4;
        4'd5: D_ssd = `SS_5;
        4'd6: D_ssd = `SS_6;
        4'd7: D_ssd = `SS_7;
        4'd8: D_ssd = `SS_8;
        4'd9: D_ssd = `SS_9;
        default: D_ssd = 8'b01110001;
    endcase
    end
  end
endmodule
```

If rst == 1

return to 30

If count to 49999999

Count return to 0

Watch0 – 1 (first digit)

If watch0 == 0

Then watch1 – 1 (second digit)

and watch0 return to 9

If all == 0 (00)

Then stops at 00

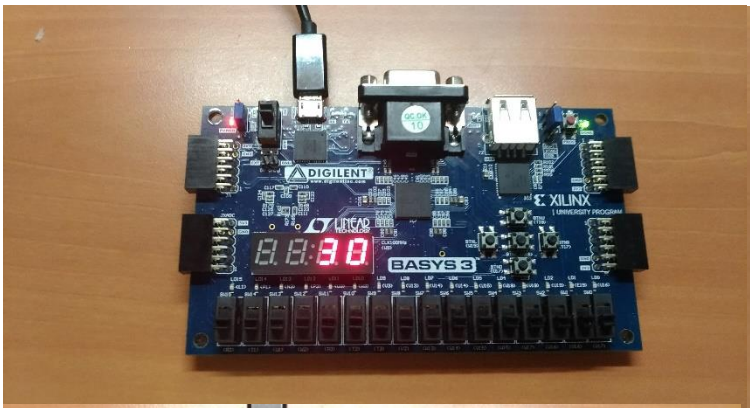
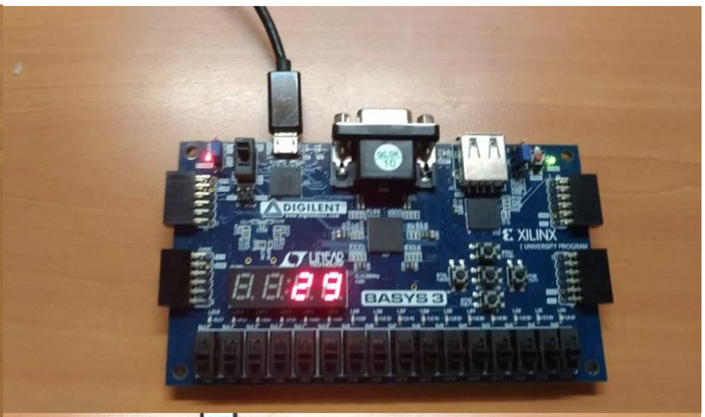
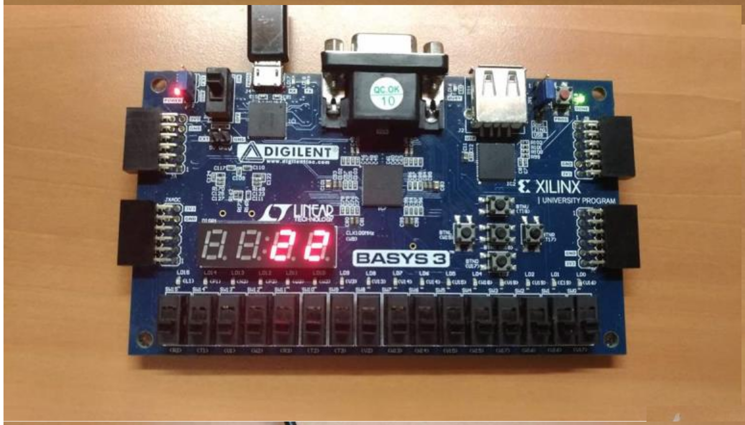
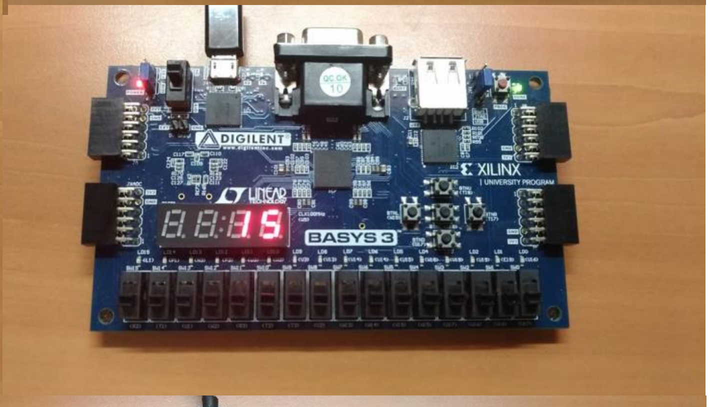
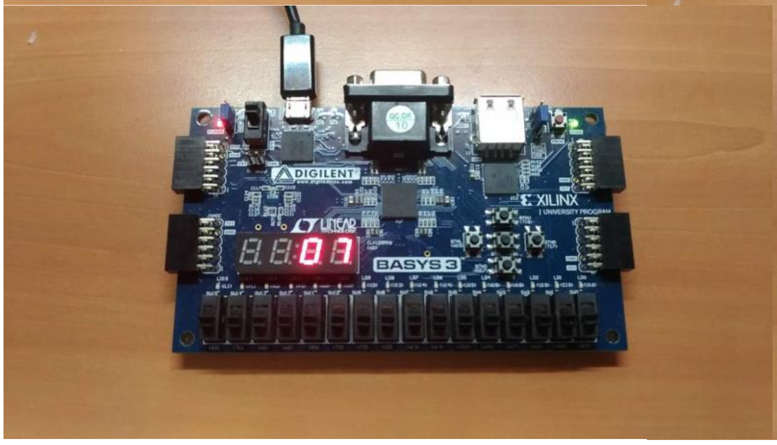
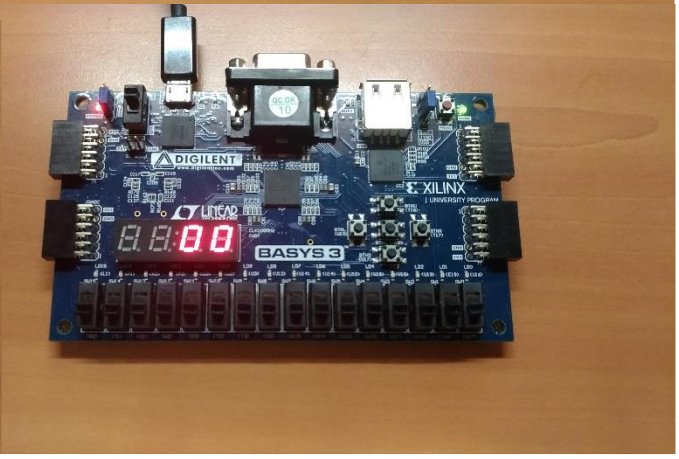
Count continue ++

Give ssd_ctl a case of [17:16]

Then repeatedly change the on position on

7-seg display

Display the number on the right position


DIGILENT
XILINX
30
BASYS 3

DIGILENT
XILINX
29
BASYS 3

DIGILENT
XILINX
22
BASYS 3

DIGILENT
XILINX
15
BASYS 3

DIGILENT
XILINX
01
BASYS 3

DIGILENT
XILINX
00
BASYS 3