ECE3300L Lab 5

Group B

By Faris Khan (ID #: 012621102)

AND

Nicholas Williams (ID#:016556982)

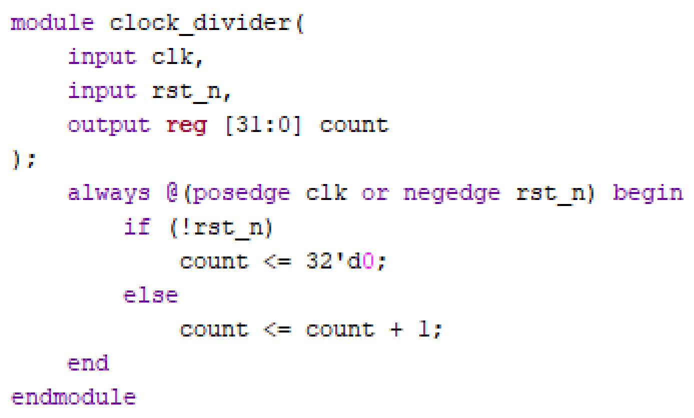
20 July 2025

Introduction

This lab focuses on the design and implementation of a two-digit BCD up/down counter on the Nexys A7 FPGA board. The system allows a user to control the counting speed via a 5-bit switch input and to select the counting direction (up or down) and reset the counter using two onboard pushbuttons. A 32-bit free-running counter and a 32-to-1 multiplexer are used to generate a variable clock signal, which drives a cascaded BCD counter capable of counting from 00 to 99 in either direction. The output is displayed on a dual-digit 7-segment display using time-multiplexing, where only one digit is illuminated at a time at a rate fast enough to appear steady to the human eye. Additionally, the current BCD value is shown on the onboard LEDs for debugging and binary visualization. This lab reinforces modular design, hardware interfacing, and clock division techniques while providing hands-on experience with Verilog simulation, synthesis, and real-time testing on FPGA hardware.

Code

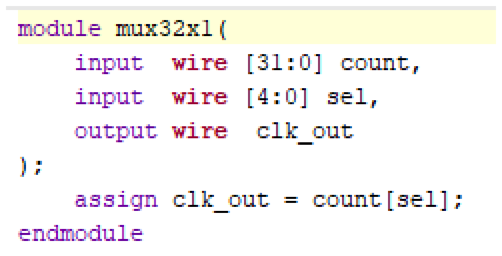
clock_divider.v



```
module clock_divider(
    input clk,
    input rst_n,
    output reg [31:0] count
);
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            count <= 32'd0;
        else
            count <= count + 1;
    end
endmodule
```

The clock_divider module is a 32-bit counter that just keeps counting up every time the clock ticks. If the reset signal is pressed (active low), it resets the count back to zero. We use this counter to slow down the 100 MHz clock by picking one of its bits later on, so we can control how fast the BCD counter updates.

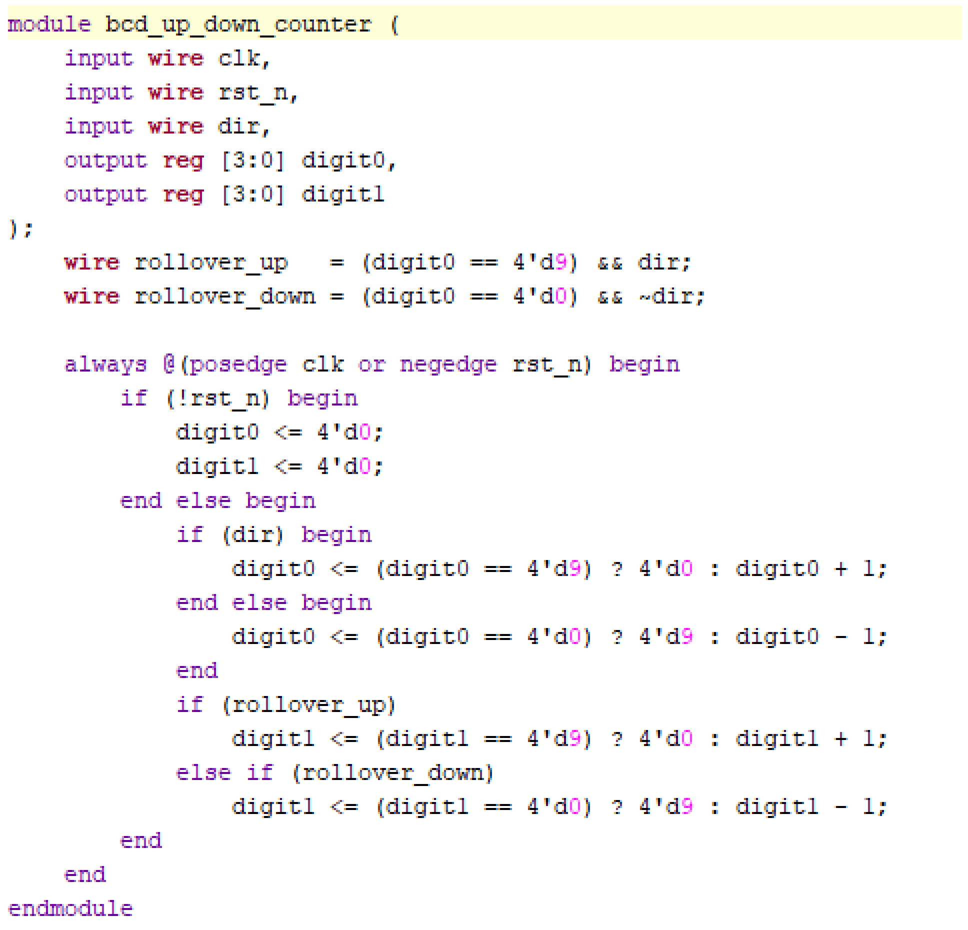
mux32x1.v



```
module mux32x1(
    input  wire [31:0] count,
    input  wire [4:0] sel,
    output wire  clk_out
);
    assign clk_out = count[sel];
endmodule
```

The mux32x1 module takes in the 32-bit output from the clock divider and a 5-bit select input from the switches. It picks one bit out of the 32 based on the value of the select input and sends that out as clk_out. This lets us change how fast the counter runs by choosing a slower or faster clock pulse from the divider.

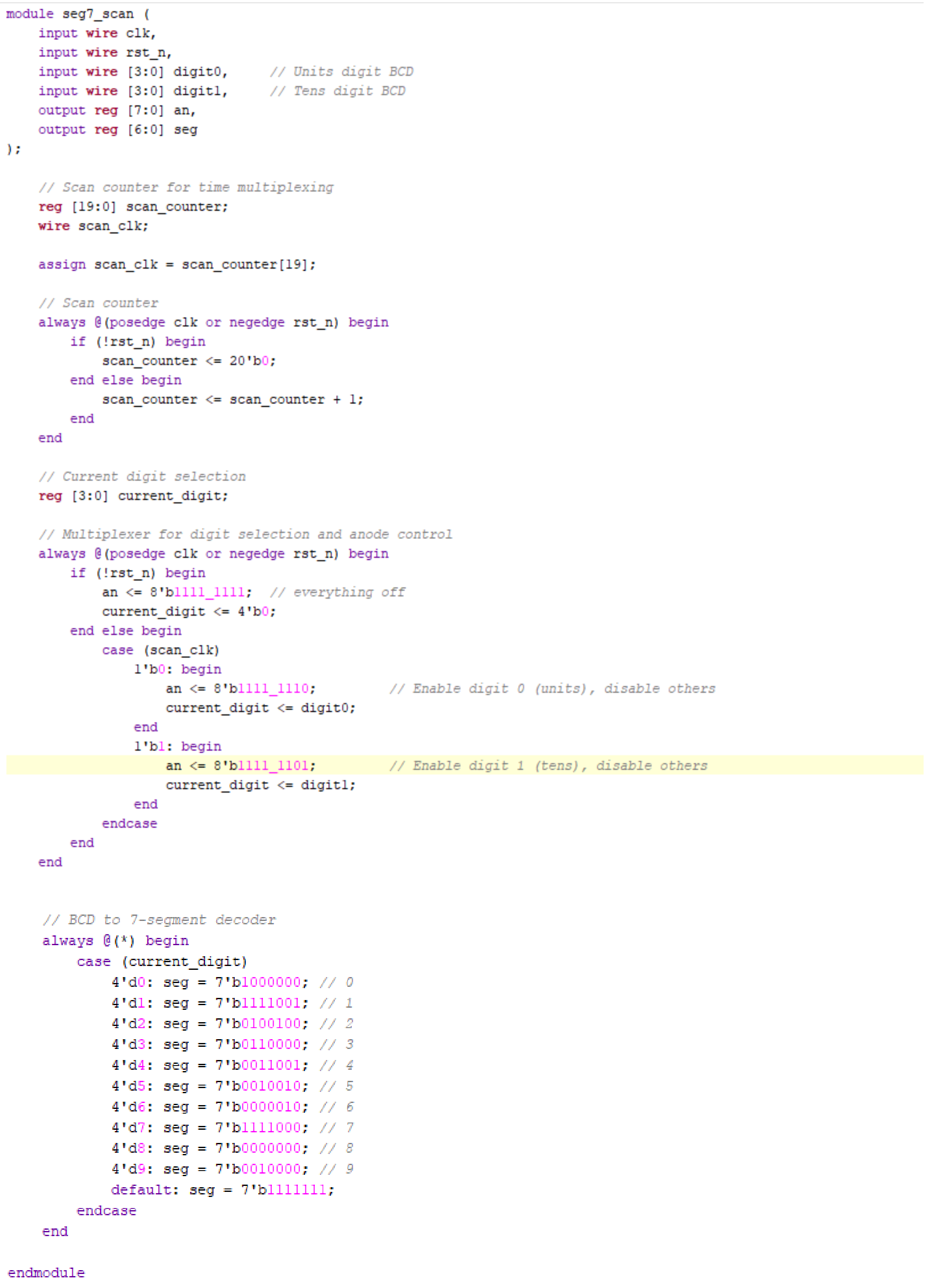
bcd_up_down_counter.v



```
module bcd_up_down_counter (
    input wire clk,
    input wire rst_n,
    input wire dir,
    output reg [3:0] digit0,
    output reg [3:0] digit1
);
    wire rollover_up   = (digit0 == 4'd9) && dir;
    wire rollover_down = (digit0 == 4'd0) && ~dir;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            digit0 <= 4'd0;
            digit1 <= 4'd0;
        end else begin
            if (dir) begin
                digit0 <= (digit0 == 4'd9) ? 4'd0 : digit0 + 1;
            end else begin
                digit0 <= (digit0 == 4'd0) ? 4'd9 : digit0 - 1;
            end
            if (rollover_up)
                digit1 <= (digit1 == 4'd9) ? 4'd0 : digit1 + 1;
            else if (rollover_down)
                digit1 <= (digit1 == 4'd0) ? 4'd9 : digit1 - 1;
        end
    end
endmodule
```

The bcd_up_down_counter module keeps track of a two-digit BCD count. It uses digit0 for the ones place and digit1 for the tens place. Depending on the dir input, it either counts up or down. If reset is pressed, both digits go back to 0. When counting up, digit0 goes from 0 to 9 and rolls over to 0 again, while digit1 increases by 1. When counting down, digit0 goes from 0 to 9 in reverse, and if it hits 0, digit1 decreases. It handles rollover logic so the counter correctly goes from 09 → 10 or 00 → 99 depending on direction.

seg7_scan.v

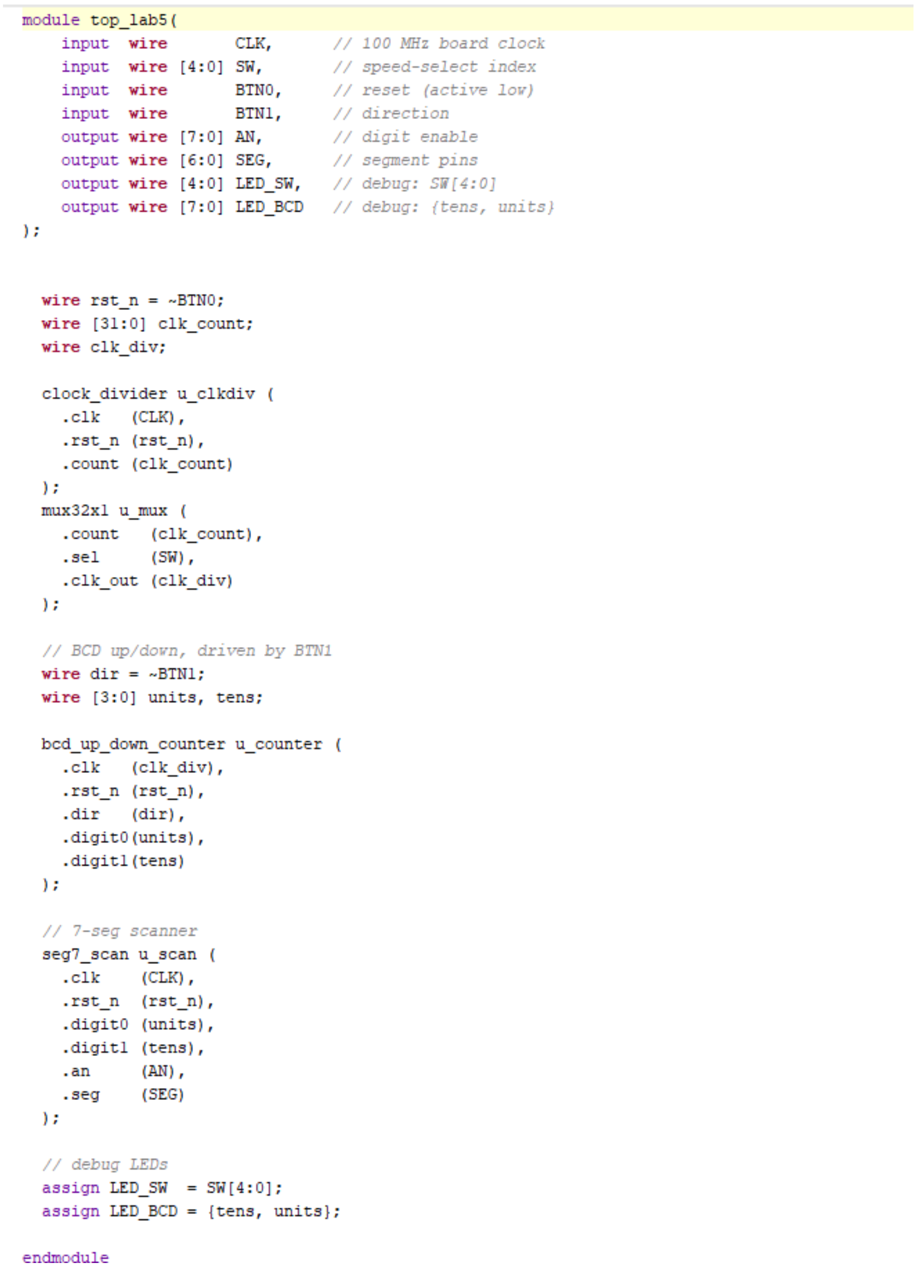


```verilog
module seg7_scan (
    input wire clk,
    input wire rst_n,
    input wire [3:0] digit0,     // Units digit BCD
    input wire [3:0] digit1,     // Tens digit BCD
    output reg [7:0] an,
    output reg [6:0] seg
);

    // Scan counter for time multiplexing
    reg [19:0] scan_counter;
    wire scan_clk;

    assign scan_clk = scan_counter[19];

    // Scan counter
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            scan_counter <= 20'b0;
        end else begin
            scan_counter <= scan_counter + 1;
        end
    end

    // Current digit selection
    reg [3:0] current_digit;

    // Multiplexer for digit selection and anode control
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            an <= 8'b1111_1111;  // everything off
            current_digit <= 4'b0;
        end else begin
            case (scan_clk)
                1'b0: begin
                    an <= 8'b1111_1110;         // Enable digit 0 (units), disable others
                    current_digit <= digit0;
                end
                1'b1: begin
                    an <= 8'b1111_1101;         // Enable digit 1 (tens), disable others
                    current_digit <= digit1;
                end
            endcase
        end
    end


    // BCD to 7-segment decoder
    always @(*) begin
        case (current_digit)
            4'd0: seg = 7'b1000000; // 0
            4'd1: seg = 7'b1111001; // 1
            4'd2: seg = 7'b0100100; // 2
            4'd3: seg = 7'b0110000; // 3
            4'd4: seg = 7'b0011001; // 4
            4'd5: seg = 7'b0010010; // 5
            4'd6: seg = 7'b0000010; // 6
            4'd7: seg = 7'b1111000; // 7
            4'd8: seg = 7'b0000000; // 8
            4'd9: seg = 7'b0010000; // 9
            default: seg = 7'b1111111;
        endcase
    end

endmodule
```

The seg7_scan module controls the two-digit 7-segment display using time-multiplexing. It takes in the BCD values for the ones and tens digits and switches between them fast enough that they both appear to be lit at the same time. It uses a counter to create a slower clock (scan_clk) and decides which digit to show based on that. When scan_clk is 0, it turns on digit0 (the ones place), and when it’s 1, it shows digit1 (the tens place). The module also has a decoder that turns the 4-bit BCD number into the correct segment pattern to light up the right number on the display.

\

top_lab5.v

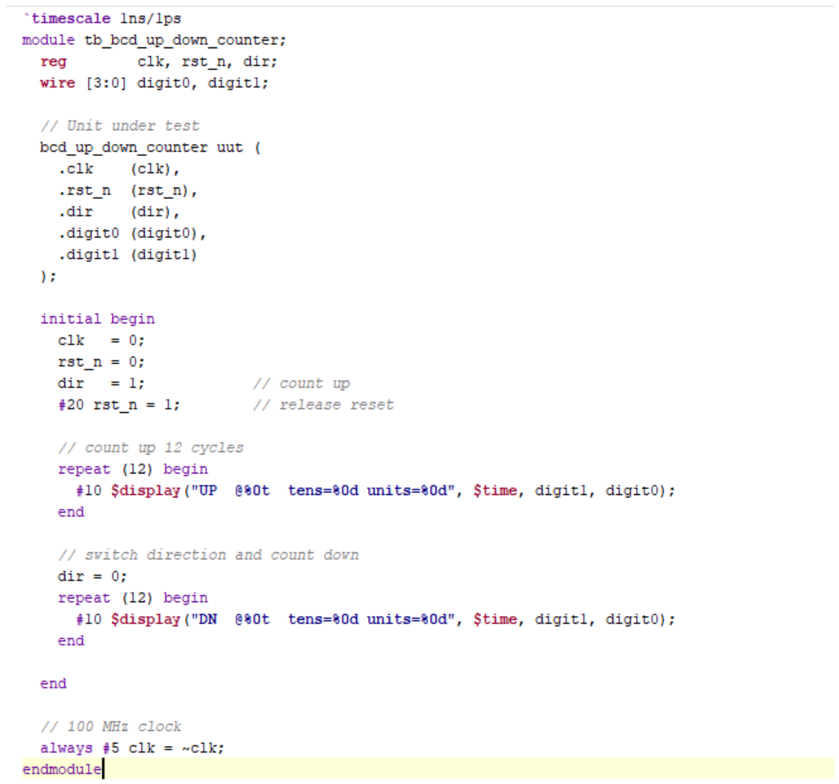


```verilog
module top_lab5(
    input  wire       CLK,      // 100 MHz board clock
    input  wire [4:0] SW,       // speed-select index
    input  wire       BTN0,     // reset (active low)
    input  wire       BTN1,     // direction
    output wire [7:0] AN,       // digit enable
    output wire [6:0] SEG,      // segment pins
    output wire [4:0] LED_SW,   // debug: SW[4:0]
    output wire [7:0] LED_BCD   // debug: {tens, units}
);


  wire rst_n = ~BTN0;
  wire [31:0] clk_count;
  wire clk_div;

  clock_divider u_clkdiv (
    .clk   (CLK),
    .rst_n (rst_n),
    .count (clk_count)
  );
  mux32x1 u_mux (
    .count   (clk_count),
    .sel     (SW),
    .clk_out (clk_div)
  );

  // BCD up/down, driven by BTN1
  wire dir = ~BTN1;
  wire [3:0] units, tens;

  bcd_up_down_counter u_counter (
    .clk   (clk_div),
    .rst_n (rst_n),
    .dir   (dir),
    .digit0(units),
    .digit1(tens)
  );

  // 7-seg scanner
  seg7_scan u_scan (
    .clk    (CLK),
    .rst_n  (rst_n),
    .digit0 (units),
    .digit1 (tens),
    .an     (AN),
    .seg    (SEG)
  );

  // debug LEDs
  assign LED_SW  = SW[4:0];
  assign LED_BCD = {tens, units};

endmodule
```

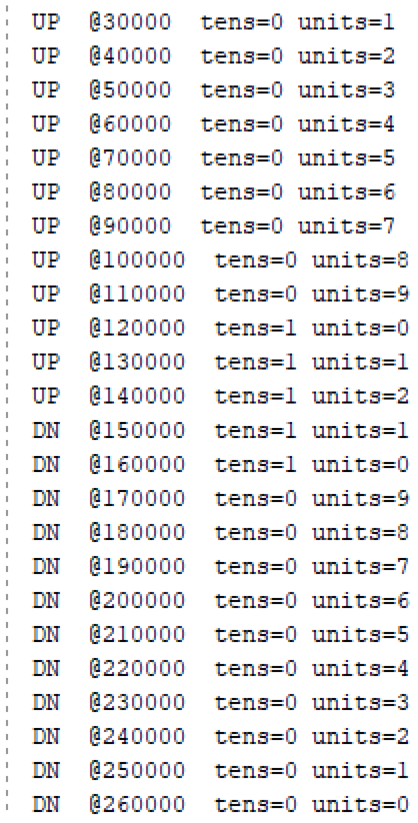
The top_lab5 module connects everything together for the project. It takes in the main 100 MHz clock, switch inputs, and pushbuttons, then wires them up to the submodules. First, the clock_divider creates a big 32-bit counter. That counter gets passed into a mux32x1, which uses the switches to pick how fast the BCD counter should run. The selected bit (slower clock) drives the bcd_up_down_counter, which counts up or down depending on BTN1. The two digits of the counter are passed to the seg7_scan module to display them on the 7-segment display using time-multiplexing. For debugging, the switch values are shown on LED_SW, and the current BCD digits are sent to LED_BCD.

Testbenches

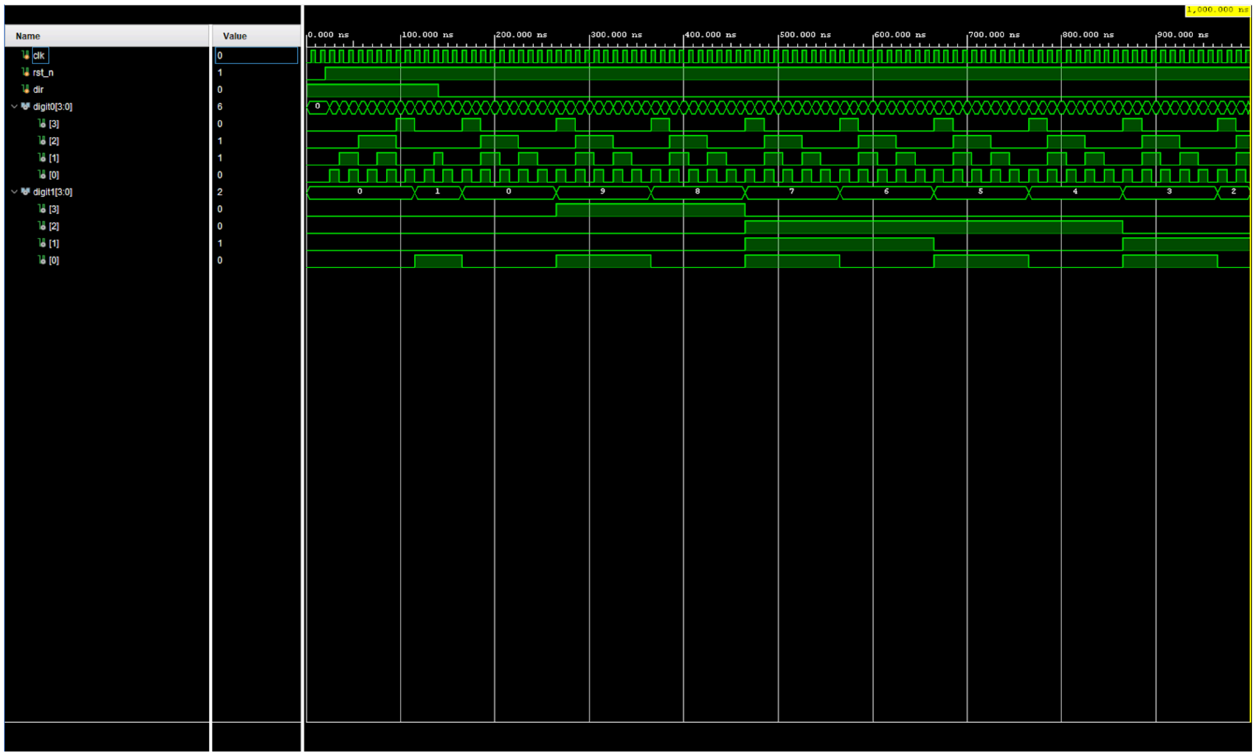
tb_bcd_up_down_counter.v



```
`timescale 1ns/1ps
module tb_bcd_up_down_counter;
  reg        clk, rst_n, dir;
  wire [3:0] digit0, digit1;

  // Unit under test
  bcd_up_down_counter uut (
    .clk    (clk),
    .rst_n  (rst_n),
    .dir    (dir),
    .digit0 (digit0),
    .digit1 (digit1)
  );

  initial begin
    clk   = 0;
    rst_n = 0;
    dir   = 1;            // count up
    #20 rst_n = 1;        // release reset

    // count up 12 cycles
    repeat (12) begin
      #10 $display("UP  @%0t  tens=%0d units=%0d", $time, digit1, digit0);
    end

    // switch direction and count down
    dir = 0;
    repeat (12) begin
      #10 $display("DN  @%0t  tens=%0d units=%0d", $time, digit1, digit0);
    end

  end

  // 100 MHz clock
  always #5 clk = ~clk;
endmodule
```



```
UP  @30000  tens=0 units=1
UP  @40000  tens=0 units=2
UP  @50000  tens=0 units=3
UP  @60000  tens=0 units=4
UP  @70000  tens=0 units=5
UP  @80000  tens=0 units=6
UP  @90000  tens=0 units=7
UP  @100000  tens=0 units=8
UP  @110000  tens=0 units=9
UP  @120000  tens=1 units=0
UP  @130000  tens=1 units=1
UP  @140000  tens=1 units=2
DN  @150000  tens=1 units=1
DN  @160000  tens=1 units=0
DN  @170000  tens=0 units=9
DN  @180000  tens=0 units=8
DN  @190000  tens=0 units=7
DN  @200000  tens=0 units=6
DN  @210000  tens=0 units=5
DN  @220000  tens=0 units=4
DN  @230000  tens=0 units=3
DN  @240000  tens=0 units=2
DN  @250000  tens=0 units=1
DN  @260000  tens=0 units=0
```

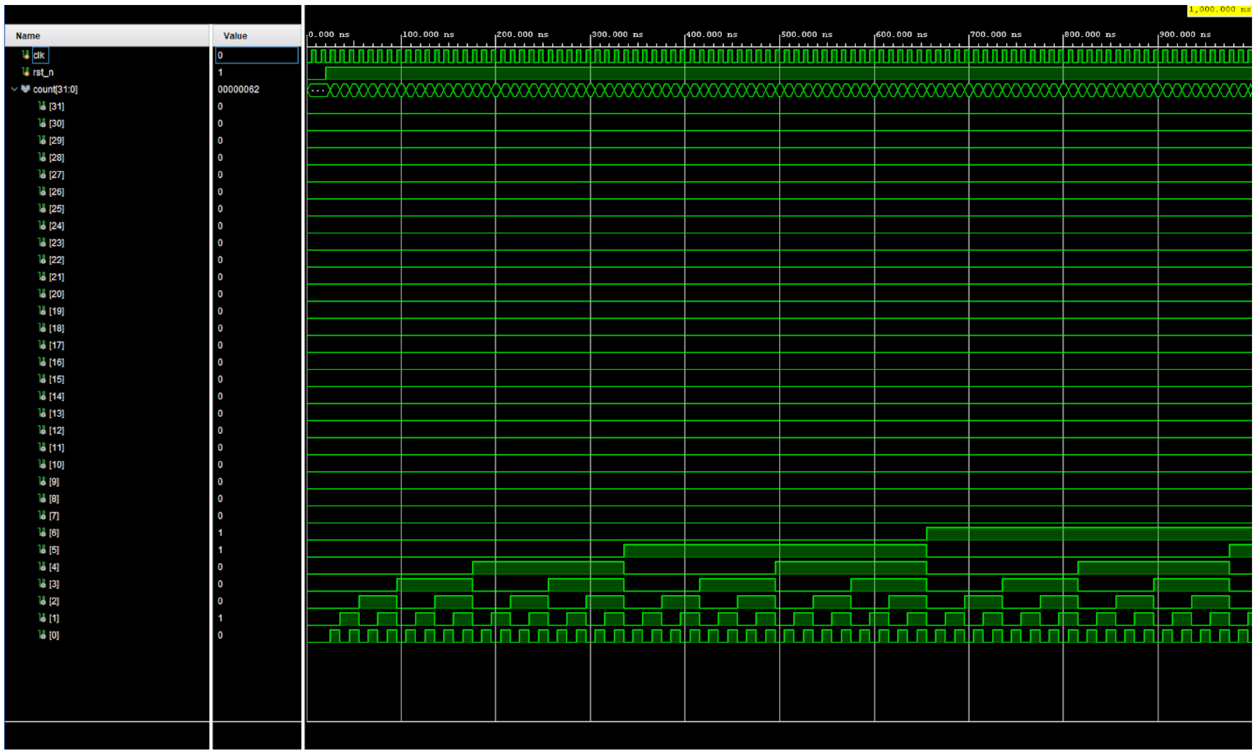


Name
Value
clk 0
rst_n 1
dir 0
digit0[3:0] 6
[3] 0
[2] 1
[1] 1
[0] 0
digit1[3:0] 2
[3] 0
[2] 0
[1] 1
[0] 0
1,000.000 ns
0.000 ns
100.000 ns
200.000 ns
300.000 ns
400.000 ns
500.000 ns
600.000 ns
700.000 ns
800.000 ns
900.000 ns
0
0
1
0
9
8
7
6
5
4
3
2

Tb_clock_divider.v

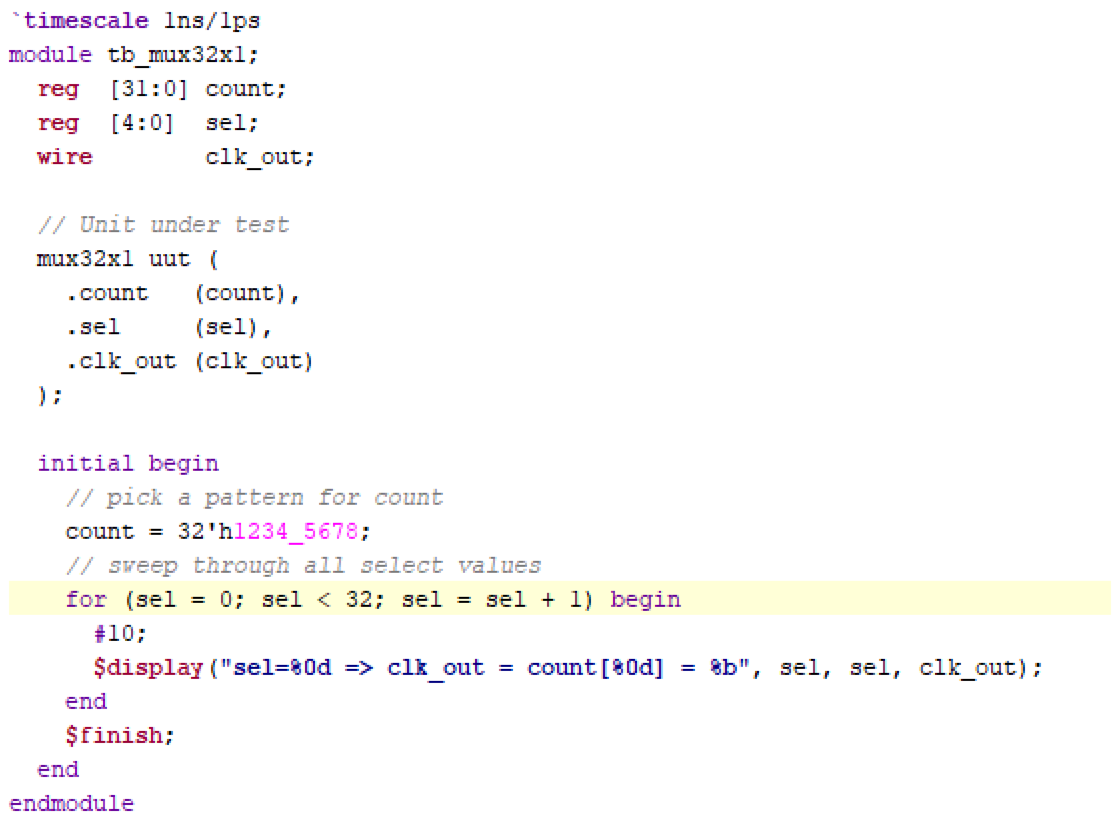
```
`timescale 1ns/1ps
module tb_clock_divider;
  reg         clk;
  reg         rst_n;
  wire [31:0] count;

  // Unit under test
  clock_divider uut (
    .clk   (clk),
    .rst_n (rst_n),
    .count (count)
  );

  initial begin
    clk   = 0;
    rst_n = 0;
    #20 rst_n = 1;         // release reset

    // print count every 200 ns
    repeat (10) begin
      #200;
      $display("[%0t] count = %0d", $time, count);
    end
    $finish;
  end

  // 100 MHz clock
  always #5 clk = ~clk;
endmodule
```

```
[220000] count = 20
[420000] count = 40
[620000] count = 60
[820000] count = 80
```



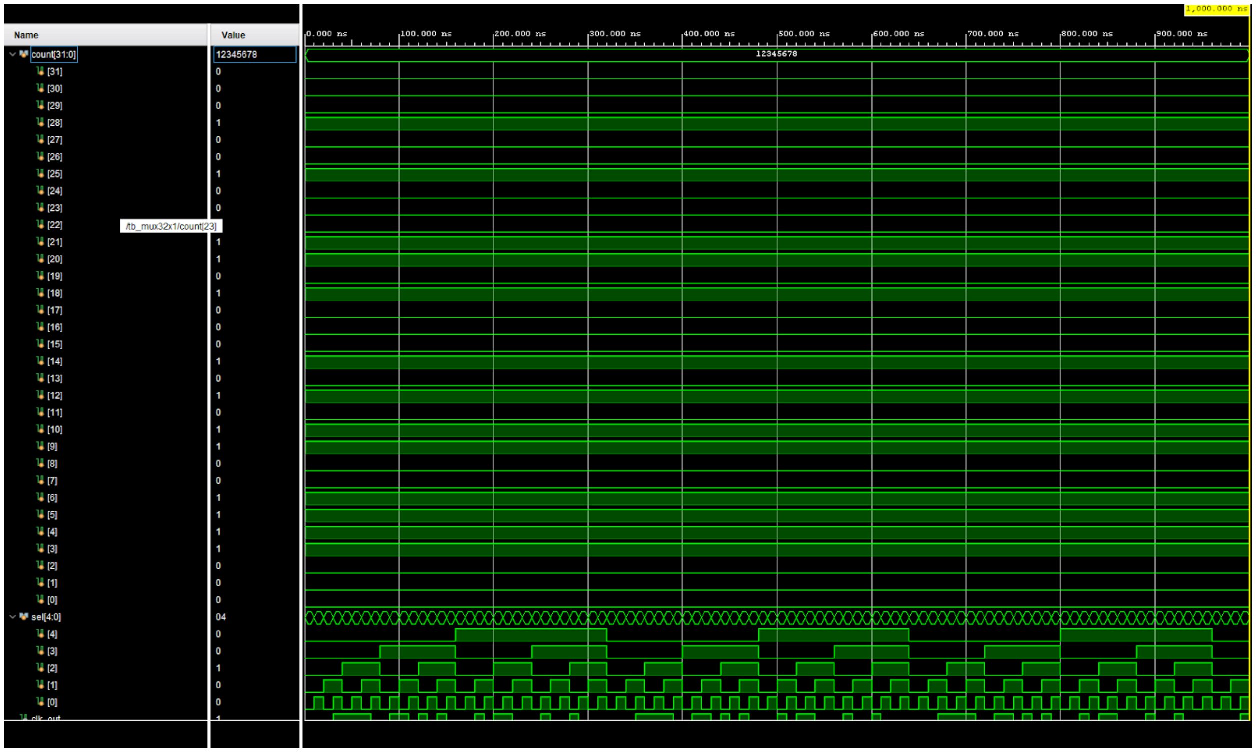
Name	Value
clk	0
rst_n	1
count[31:0]	00000062
[31]	0
[30]	0
[29]	0
[28]	0
[27]	0
[26]	0
[25]	0
[24]	0
[23]	0
[22]	0
[21]	0
[20]	0
[19]	0
[18]	0
[17]	0
[16]	0
[15]	0
[14]	0
[13]	0
[12]	0
[11]	0
[10]	0
[9]	0
[8]	0
[7]	0
[6]	1
[5]	1
[4]	0
[3]	0
[2]	0
[1]	1
[0]	0

Tb_mux32x1.v



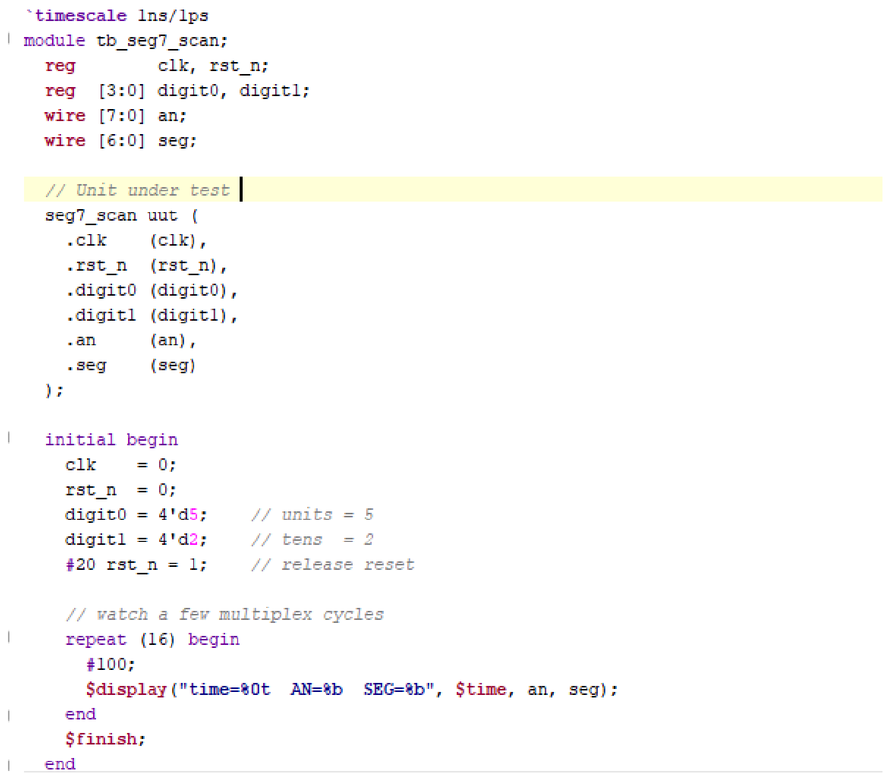
```
`timescale 1ns/1ps
module tb_mux32x1;
  reg  [31:0] count;
  reg  [4:0]  sel;
  wire        clk_out;

  // Unit under test
  mux32x1 uut (
    .count   (count),
    .sel     (sel),
    .clk_out (clk_out)
  );

  initial begin
    // pick a pattern for count
    count = 32'h1234_5678;
    // sweep through all select values
    for (sel = 0; sel < 32; sel = sel + 1) begin
      #10;
      $display("sel=%0d => clk_out = count[%0d] = %b", sel, sel, clk_out);
    end
    $finish;
  end
endmodule
```

```
sel=1 => clk_out = count[1] = 0
sel=2 => clk_out = count[2] = 0
sel=3 => clk_out = count[3] = 1
sel=4 => clk_out = count[4] = 1
sel=5 => clk_out = count[5] = 1
sel=6 => clk_out = count[6] = 1
sel=7 => clk_out = count[7] = 0
sel=8 => clk_out = count[8] = 0
sel=9 => clk_out = count[9] = 1
sel=10 => clk_out = count[10] = 1
sel=11 => clk_out = count[11] = 0
sel=12 => clk_out = count[12] = 1
sel=13 => clk_out = count[13] = 0
sel=14 => clk_out = count[14] = 1
sel=15 => clk_out = count[15] = 0
sel=16 => clk_out = count[16] = 0
sel=17 => clk_out = count[17] = 0
sel=18 => clk_out = count[18] = 1
sel=19 => clk_out = count[19] = 0
sel=20 => clk_out = count[20] = 1
sel=21 => clk_out = count[21] = 1
sel=22 => clk_out = count[22] = 0
sel=23 => clk_out = count[23] = 0
sel=24 => clk_out = count[24] = 0
sel=25 => clk_out = count[25] = 1
sel=26 => clk_out = count[26] = 0
sel=27 => clk_out = count[27] = 0
sel=28 => clk_out = count[28] = 1
sel=29 => clk_out = count[29] = 0
sel=30 => clk_out = count[30] = 0
sel=31 => clk_out = count[31] = 0
sel=0 => clk_out = count[0] = 0
sel=1 => clk_out = count[1] = 0
sel=2 => clk_out = count[2] = 0
sel=3 => clk_out = count[3] = 1
sel=4 => clk_out = count[4] = 1
sel=5 => clk_out = count[5] = 1
sel=6 => clk_out = count[6] = 1
sel=7 => clk_out = count[7] = 0
sel=8 => clk_out = count[8] = 0
sel=9 => clk_out = count[9] = 1
sel=10 => clk_out = count[10] = 1
sel=11 => clk_out = count[11] = 0
sel=12 => clk_out = count[12] = 1
sel=13 => clk_out = count[13] = 0
sel=14 => clk_out = count[14] = 1
sel=15 => clk_out = count[15] = 0
sel=16 => clk_out = count[16] = 0
sel=17 => clk_out = count[17] = 0
sel=18 => clk_out = count[18] = 1
sel=19 => clk_out = count[19] = 0
sel=20 => clk_out = count[20] = 1
sel=21 => clk_out = count[21] = 1
sel=22 => clk_out = count[22] = 0
sel=23 => clk_out = count[23] = 0
sel=24 => clk_out = count[24] = 0
sel=25 => clk_out = count[25] = 1
sel=26 => clk_out = count[26] = 0
sel=27 => clk_out = count[27] = 0
sel=28 => clk_out = count[28] = 1
sel=29 => clk_out = count[29] = 0
sel=30 => clk_out = count[30] = 0
```

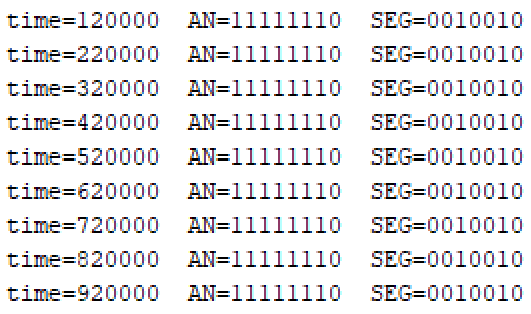


```
Name            Value
count[31:0]     12345678
  [31]          0
  [30]          0
  [29]          0
  [28]          1
  [27]          0
  [26]          0
  [25]          1
  [24]          0
  [23]          0
  [22]
  [21]          1
  [20]          1
  [19]          0
  [18]          1
  [17]          0
  [16]          0
  [15]          0
  [14]          1
  [13]          0
  [12]          1
  [11]          0
  [10]          1
  [9]           1
  [8]           0
  [7]           0
  [6]           1
  [5]           1
  [4]           1
  [3]           1
  [2]           0
  [1]           0
  [0]           0
sel[4:0]        04
  [4]           0
  [3]           0
  [2]           1
  [1]           0
  [0]           0
clk_out         1
```

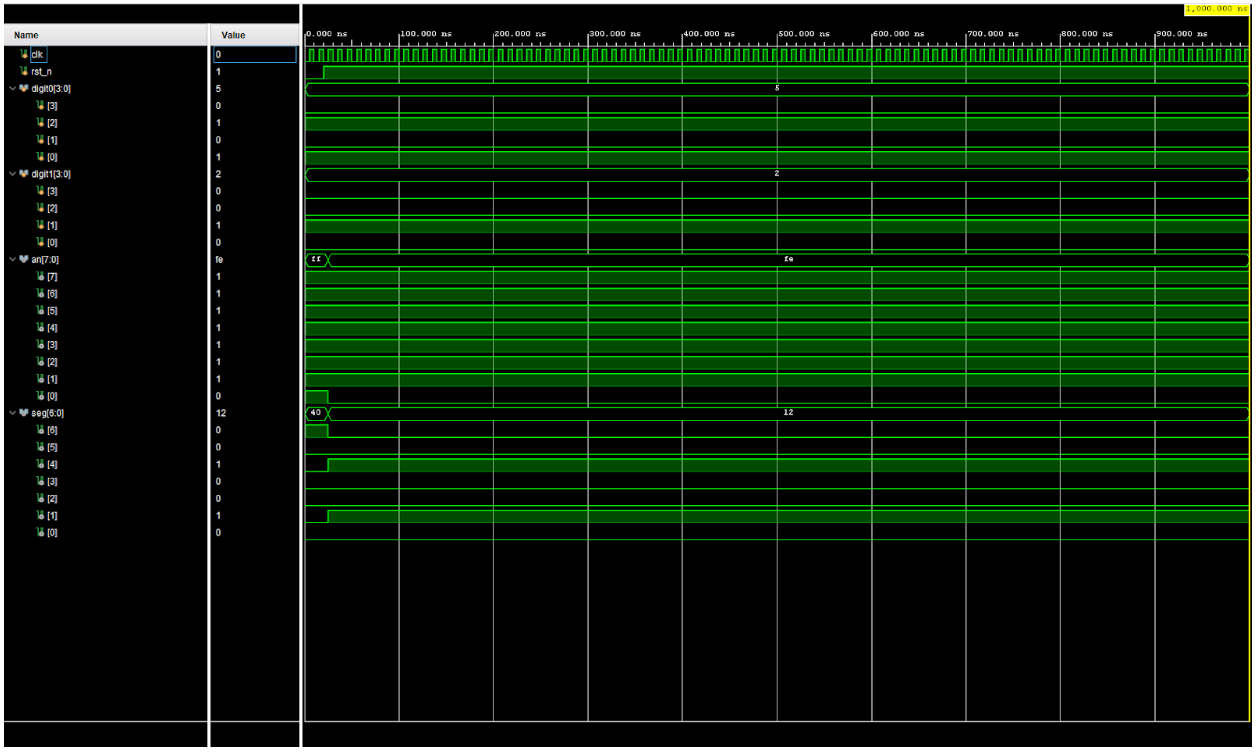
tb_seg7_scan.v



```
`timescale 1ns/1ps
module tb_seg7_scan;
  reg        clk, rst_n;
  reg  [3:0] digit0, digit1;
  wire [7:0] an;
  wire [6:0] seg;

  // Unit under test
  seg7_scan uut (
    .clk    (clk),
    .rst_n  (rst_n),
    .digit0 (digit0),
    .digit1 (digit1),
    .an     (an),
    .seg    (seg)
  );

  initial begin
    clk    = 0;
    rst_n  = 0;
    digit0 = 4'd5;    // units = 5
    digit1 = 4'd2;    // tens  = 2
    #20 rst_n = 1;    // release reset

    // watch a few multiplex cycles
    repeat (16) begin
      #100;
      $display("time=%0t  AN=%b  SEG=%b", $time, an, seg);
    end
    $finish;
  end
```

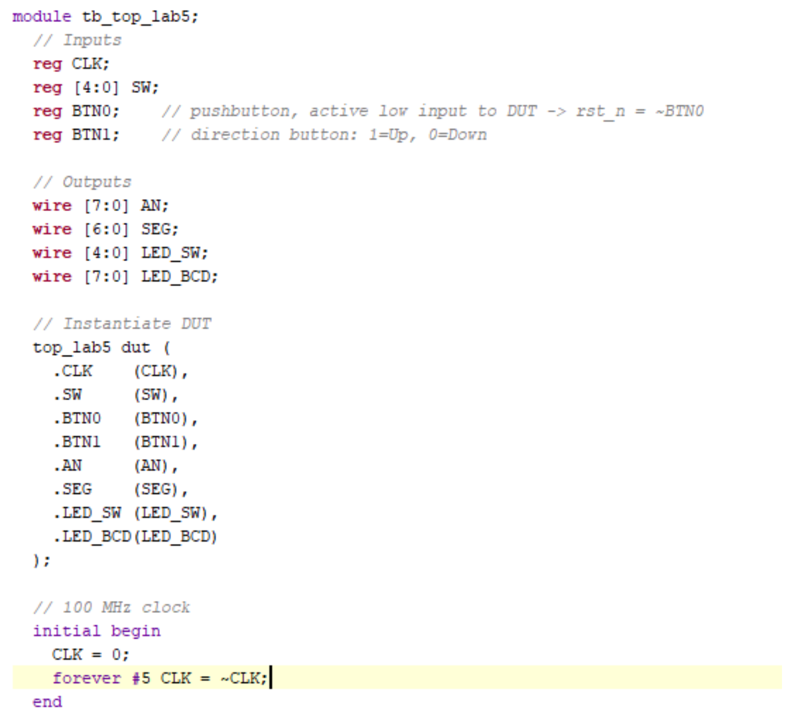


```
time=120000  AN=11111110  SEG=0010010
time=220000  AN=11111110  SEG=0010010
time=320000  AN=11111110  SEG=0010010
time=420000  AN=11111110  SEG=0010010
time=520000  AN=11111110  SEG=0010010
time=620000  AN=11111110  SEG=0010010
time=720000  AN=11111110  SEG=0010010
time=820000  AN=11111110  SEG=0010010
time=920000  AN=11111110  SEG=0010010
```

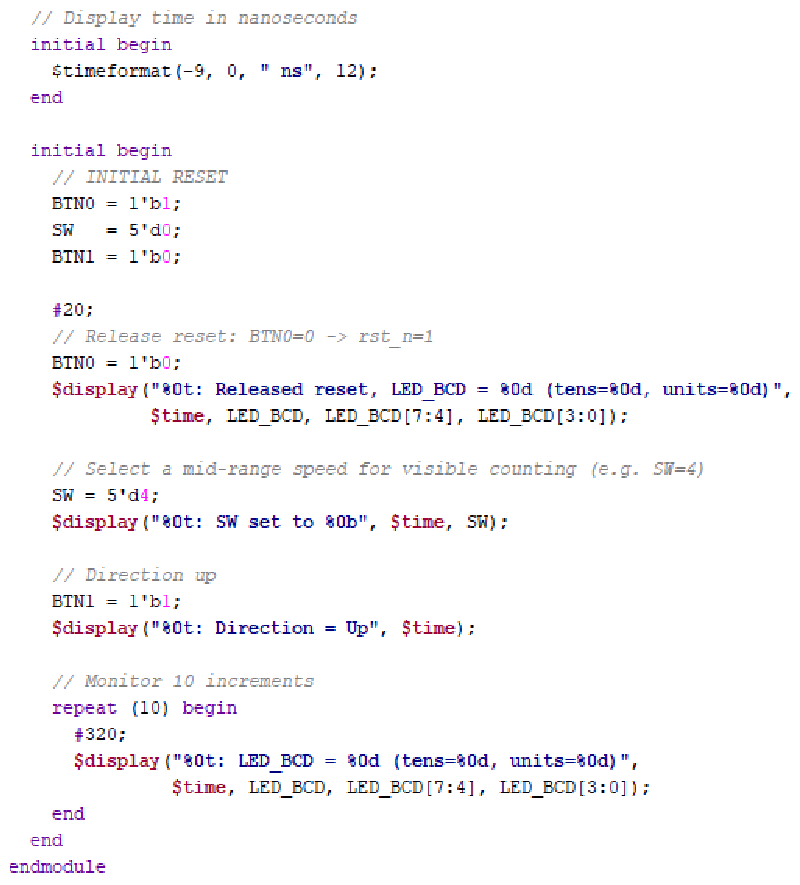


Name	Value
clk	0
rst_n	1
digit0[3:0]	5
[3]	0
[2]	1
[1]	0
[0]	1
digit1[3:0]	2
[3]	0
[2]	0
[1]	1
[0]	0
an[7:0]	fe
[7]	1
[6]	1
[5]	1
[4]	1
[3]	1
[2]	1
[1]	1
[0]	0
seg[6:0]	12
[6]	0
[5]	0
[4]	1
[3]	0
[2]	0
[1]	1
[0]	0

Tb_top_lab5.v

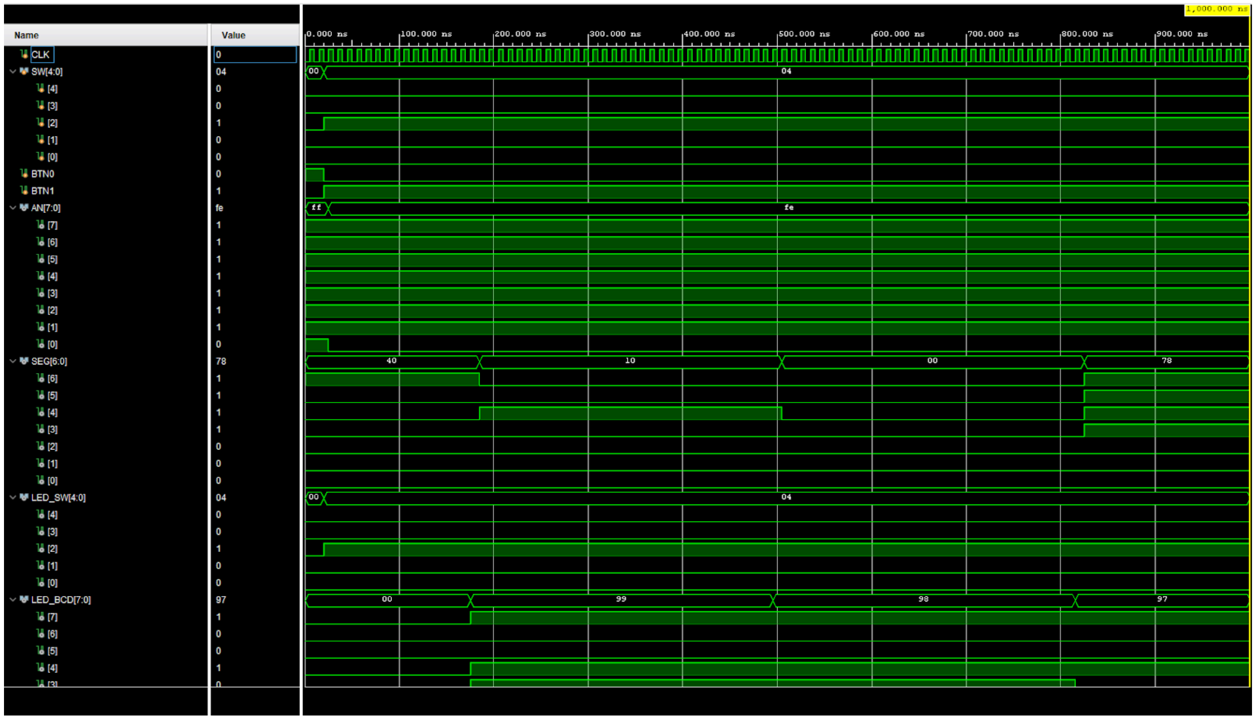


```
module tb_top_lab5;
  // Inputs
  reg CLK;
  reg [4:0] SW;
  reg BTN0;    // pushbutton, active low input to DUT -> rst_n = ~BTN0
  reg BTN1;    // direction button: 1=Up, 0=Down

  // Outputs
  wire [7:0] AN;
  wire [6:0] SEG;
  wire [4:0] LED_SW;
  wire [7:0] LED_BCD;

  // Instantiate DUT
  top_lab5 dut (
    .CLK    (CLK),
    .SW     (SW),
    .BTN0   (BTN0),
    .BTN1   (BTN1),
    .AN     (AN),
    .SEG    (SEG),
    .LED_SW (LED_SW),
    .LED_BCD(LED_BCD)
  );

  // 100 MHz clock
  initial begin
    CLK = 0;
    forever #5 CLK = ~CLK;
  end
```



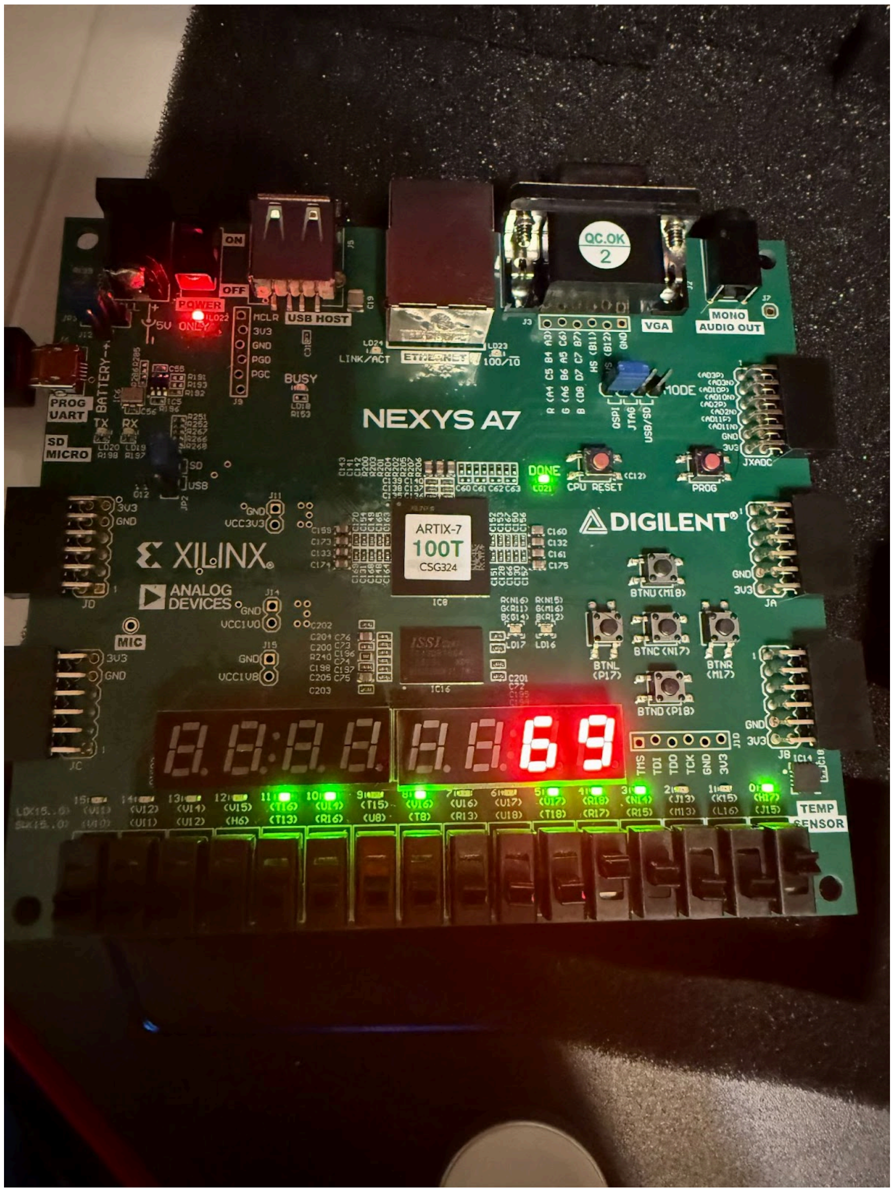
```
 // Display time in nanoseconds
 initial begin
   $timeformat(-9, 0, " ns", 12);
 end

 initial begin
   // INITIAL RESET
   BTN0 = 1'b1;
   SW   = 5'd0;
   BTN1 = 1'b0;

   #20;
   // Release reset: BTN0=0 -> rst_n=1
   BTN0 = 1'b0;
   $display("%0t: Released reset, LED_BCD = %0d (tens=%0d, units=%0d)",
            $time, LED_BCD, LED_BCD[7:4], LED_BCD[3:0]);

   // Select a mid-range speed for visible counting (e.g. SW=4)
   SW = 5'd4;
   $display("%0t: SW set to %0b", $time, SW);

   // Direction up
   BTN1 = 1'b1;
   $display("%0t: Direction = Up", $time);

   // Monitor 10 increments
   repeat (10) begin
     #320;
     $display("%0t: LED_BCD = %0d (tens=%0d, units=%0d)",
              $time, LED_BCD, LED_BCD[7:4], LED_BCD[3:0]);
   end
 end
endmodule
```

```
# run 1000ns
20 ns: Released reset, LED_BCD = 0 (tens=0, units=0)
20 ns: SW set to 100
20 ns: Direction = Up
340 ns: LED_BCD = 153 (tens=9, units=9)
660 ns: LED_BCD = 152 (tens=9, units=8)
980 ns: LED_BCD = 151 (tens=9, units=7)
```

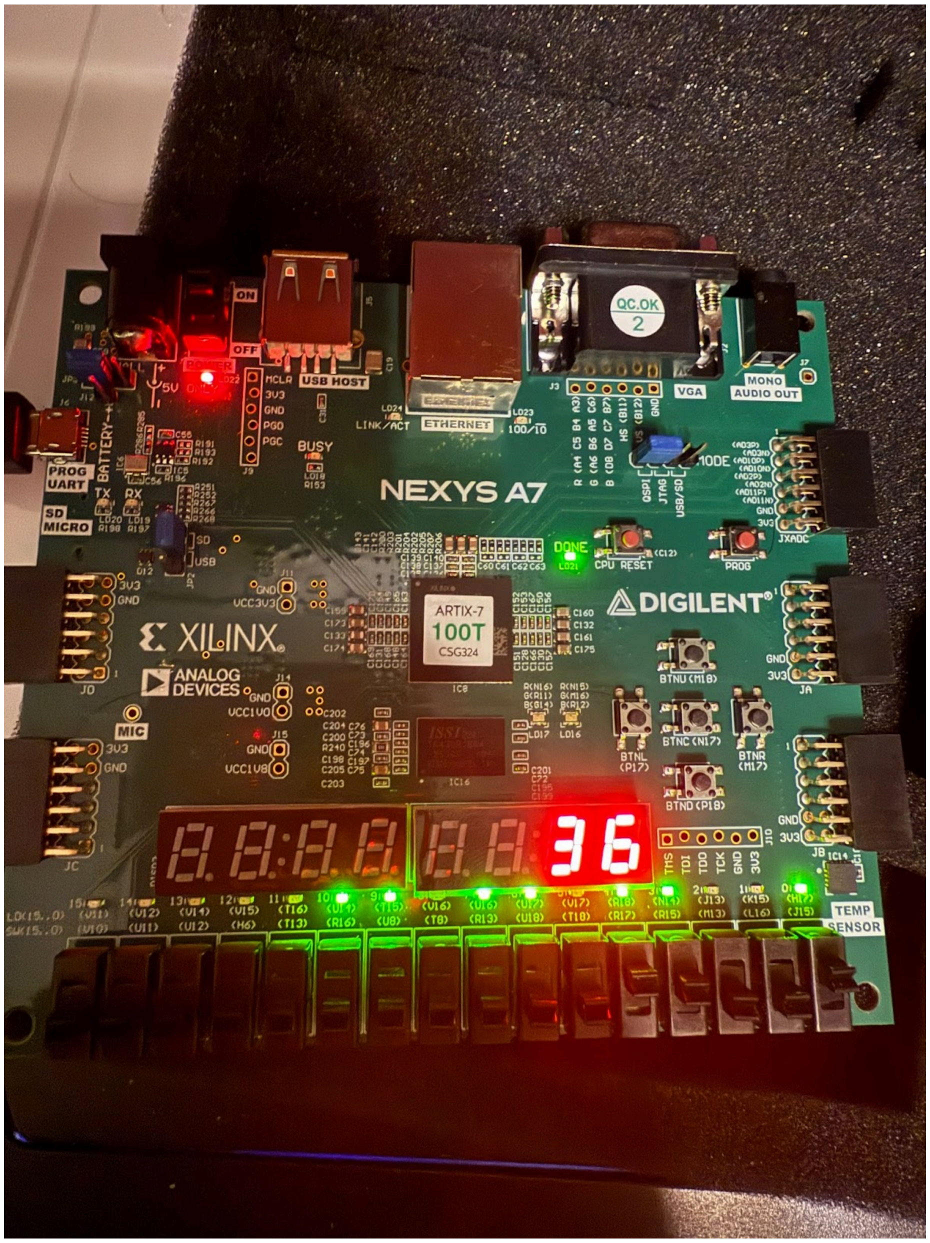


Name	Value
CLK	0
SW[4:0]	04
[4]	0
[3]	0
[2]	1
[1]	0
[0]	0
BTN0	0
BTN1	1
AN[7:0]	fe
[7]	1
[6]	1
[5]	1
[4]	1
[3]	1
[2]	1
[1]	1
[0]	0
SEG[6:0]	78
[6]	1
[5]	1
[4]	1
[3]	1
[2]	0
[1]	0
[0]	0
LED_SW[4:0]	04
[4]	0
[3]	0
[2]	1
[1]	0
[0]	0
LED_BCD[7:0]	97
[7]	1
[6]	0
[5]	0
[4]	1
[3]	0

Hardware Results



We can see that the corresponding LEDs of the first 4 switches are lit up. LEDs 5-8 are for the ones digit and 9-12 are for the tens digit and we can see that we currently have 1001, or 9, for LEDs 5-8 and 0110, or 6, for LEDs 9-12 which displays 69.

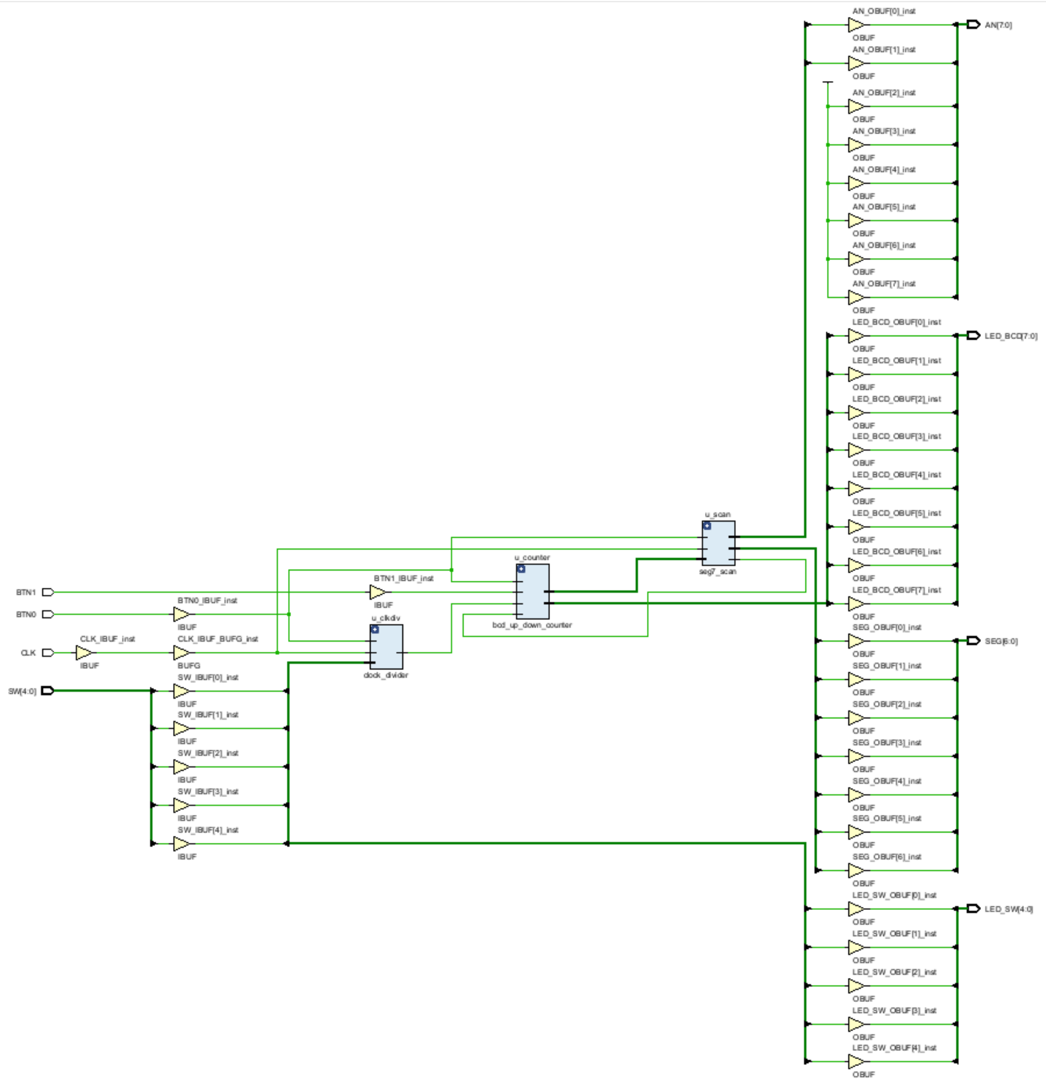


We can see that we currently have 0110, or 6, for LEDs 5-8 and 0011, or 3, for LEDs 9-12 which displays 36.

Cascading, direction and reset functionalities shown in the demo.

Screenshots

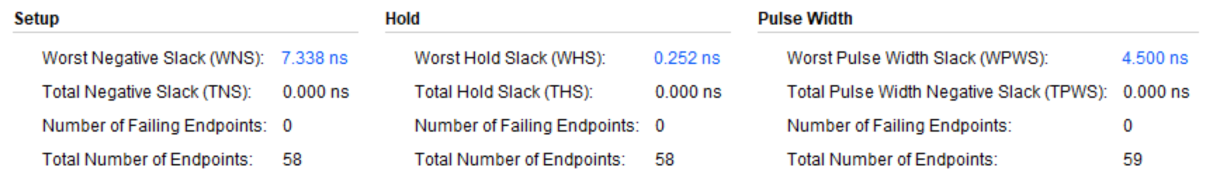
Schematics



LUT & FF

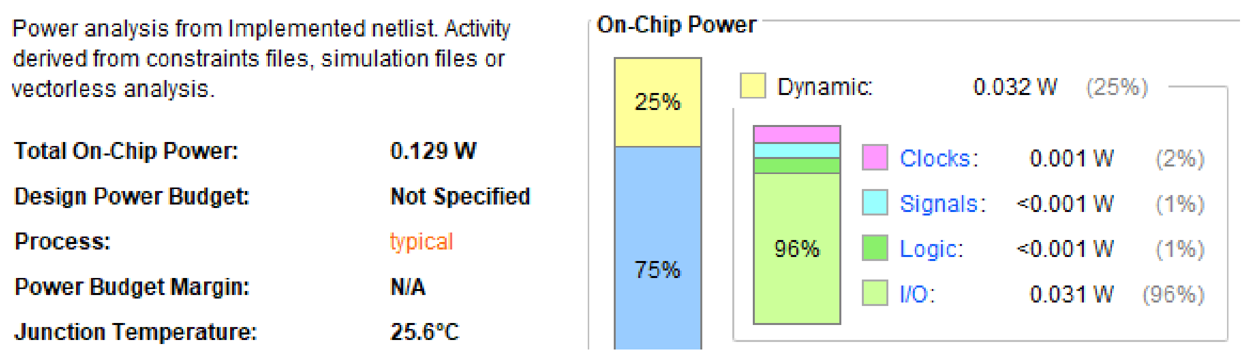
LUT	FF
24	66
24	66

Timing Summary



Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.338 ns	Worst Hold Slack (WHS):	0.252 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	58	Total Number of Endpoints:	58	Total Number of Endpoints:	59

Power Summary



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.129 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.6°C**

On-Chip Power

Dynamic:		0.032 W	(25%)
	Clocks:	0.001 W	(2%)
	Signals:	<0.001 W	(1%)
	Logic:	<0.001 W	(1%)
	I/O:	0.031 W	(96%)

Conclusion

This lab delved into designing and testing a digital system using Verilog and the Nexys A7 FPGA board. We built a working two-digit BCD up/down counter that responds to button presses, changes speed based on switch input, and displays the result on a 7-segment display using multiplexing. By breaking the design into smaller modules like the clock divider, mux, BCD counter, and 7-segment scanner, we were able to test each part individually and then connect everything together in the top module. The LEDs also helped us verify the internal values in binary. Overall, the lab helped reinforce how to handle clock division, binary counting, and real-time output on an FPGA, and it was a solid example of modular digital design.

Contributions

Faris (50%) - Source code, helped with lab report

Nicholas (50%) - Testbench, handled simulations, helped with report, demo