California Polytechnic State University, Pomona
Department of Electrical and Computer Engineering

Digital Circuit Design Using Verilog Laboratory
ECE 3300L

Lab 5

BCD Up/Down Counter on 7-Segment Display

By

Jetts Crittenden (ID# 015468128) and
Evan Tram(#ID 016404570)

7/21/2025

Design:

This design is based on the basic principles of Lab 4. We are using multiplexing to display two digits on 2 7-segment displays by dividing the clock signal into a refresh signal. However, this lab also implements the basic structures from previous labs, such as muxes, to create a counter. This counter works by allowing inputs to change the speed of the counter by combining a mux and a clock divider. The values of the switches change the value that is given through the mux, which either speeds up or slows down the clock cycle. This clock cycle is then controlled by a toggled button with debouncing that changes the direction of the count. This count is output as two BCD digits, one for the tens place and the other for the ones place. These digits are then routed into the 7seg_scan module, which outputs the values to the 7-segment. This gives the user a visual image of an incrementing counter that loops from 99 to 0 and vice versa. Overall, this design requires many of the basic systems used before, and it is perfect for demonstrating timing and more complex systems.

Code:

top_lab5.v

```
module top_lab5(
    input wire CLK,
    input wire [4:0] SW,
    input wire RST, // reset
    input wire BTN1, // dir
    output wire [4:0] LED_SW,
    output wire [7:0] LED_BCD,
    output wire [6:0] SEG,
    output wire [7:0] AN
);
    wire rst_n = ~RST;
    wire dir;

    wire [31:0] cnt;
    wire clk_out;

    wire [3:0] units, tens;

    assign LED_SW = SW;
```

```
    assign LED_BCD = {tens, units};

    toggle_switch dir_toggle (
        .clk(CLK),
        .btn_raw(BTN1),
        .state(dir)
    );

    clock_divider u_clkdiv (
        .clk(CLK),
        .rst_n(rst_n),
        .cnt(cnt)
    );

    mux32x1 u_mux (
        .cnt(cnt),
        .sel(SW),
        .clk_out(clk_out)
    );

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

    seg7_scan u_seg (
        .clk(CLK),
        .rst_n(rst_n),
        .digit0(units),
        .digit1(tens),
        .SEG(SEG),
        .AN(AN)
    );
```

clock_divider.v

```
`timescale 1ns / 1ps

module clock_divider(
    input wire clk,
    input wire rst_n,
    output reg [31:0] cnt
);

    always @(posedge clk or negedge rst_n)
        begin
            if (!rst_n)
                cnt <= 0;
            else
                cnt <= cnt + 1;
        end

endmodule
```

mux32x1.v

```
`timescale 1ns / 1ps

module mux32x1(
    input [31:0] cnt,
    input [4:0] sel,
    output clk_out
    );

    assign clk_out = cnt[31 - sel];

endmodule
```

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
);

    always @(posedge clk or negedge rst_n)
           begin
                if (!rst_n)
                     begin
                          digit0 <= 0;
                          digit1 <= 0;
                     end
                else if (dir)
                     begin
                          if (digit0 == 9)
                               begin
                                    digit0 <= 0;
                                    if (digit1 == 9)
                                         digit1 <= 0;
                                    else
                                         digit1 <= digit1 + 1;
                               end
                          else
                               digit0 <= digit0 + 1;
                     end
                else
                     begin
                          if (digit0 == 0)
                               begin
                                    digit0 <= 9;
                                    if (digit1 == 0)
                                         digit1 <= 9;
                                    else
                                         digit1 <= digit1 - 1;
                               end
```

```
                            else
                                digit0 <= digit0 - 1;
                        end
            end

endmodule
```

seg7_scan.v

```
`timescale 1ns / 1ps

module seg7_scan(
    input wire clk,
    input wire rst_n,
    input wire [3:0] digit0,
    input wire [3:0] digit1,
    output reg [6:0] SEG,
    output reg [7:0] AN
);

    reg [19:0] refresh_counter = 0;
    wire sel;
    reg [3:0] digit_val;

    assign sel = refresh_counter[19];

    always @(posedge clk or negedge rst_n)
        begin
            if (!rst_n)
                refresh_counter <= 0;
            else
                refresh_counter <= refresh_counter + 1;
        end

    always @(*)
        begin
            case(sel)
                1'b0:
                    begin
```

```
                    AN = 8'b11111110;
                    digit_val = digit0;
                end
            1'b1:
                begin
                    AN = 8'b11111101;
                    digit_val = digit1;
                end
            default: AN = 8'b11111111;
        endcase
    end

always@(*)
    begin
        case(digit_val)
            4'd0: SEG = 7'b1000000;
            4'd1: SEG = 7'b1111001;
            4'd2: SEG = 7'b0100100;
            4'd3: SEG = 7'b0110000;
            4'd4: SEG = 7'b0011001;
            4'd5: SEG = 7'b0010010;
            4'd6: SEG = 7'b0000010;
            4'd7: SEG = 7'b1111000;
            4'd8: SEG = 7'b0000000;
            4'd9: SEG = 7'b0010000;
            default: SEG = 7'b1111111;
        endcase
    end

endmodule
```

toggle_switch.v

```
module toggle_switch (
    input wire clk,
    input wire btn_raw,     // Raw button input
    output reg state        // Toggled output state
);

    wire btn_clean;
    reg btn_prev;

    // Debounce module
    debounce db (
        .clk(clk),
        .btn_in(btn_raw),
        .btn_clean(btn_clean)
    );

    always @(posedge clk) begin

            if (btn_clean && !btn_prev)
                state <= ~state; // Toggle on rising edge
            btn_prev <= btn_clean;
        end

endmodule
```

debounce.v

```
module debounce (
    input clk,
    input btn_in,
    output reg btn_clean
    );
        reg [2:0] shift_reg;
        always @(posedge clk) begin
        shift_reg <= {shift_reg[1:0], btn_in};
        if (shift_reg == 3'b111) btn_clean <= 1;
        else if (shift_reg == 3'b000) btn_clean <= 0;
    end
endmodule
```

XDC Snippet:

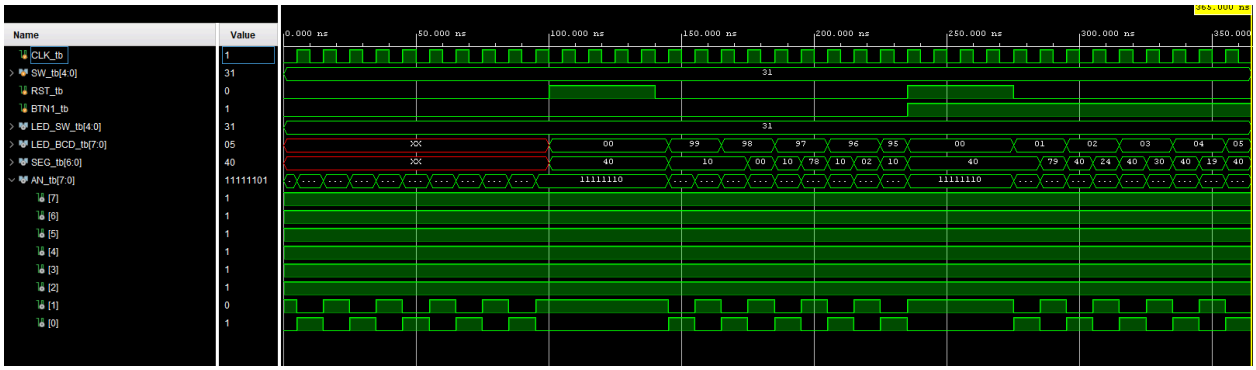
```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 }
[get_ports { CLK }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports {CLK}];
##Switches

set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 }
[get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 }
[get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 }
[get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 }
[get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 }
[get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
## LEDs

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 }
[get_ports { LED_SW[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 }
[get_ports { LED_SW[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 }
[get_ports { LED_SW[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 }
[get_ports { LED_SW[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 }
[get_ports { LED_SW[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 }
[get_ports { LED_BCD[0] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 }
[get_ports { LED_BCD[1] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 }
[get_ports { LED_BCD[2] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 }
[get_ports { LED_BCD[3] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
```

```
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 }
[get_ports { LED_BCD[4] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 }
[get_ports { LED_BCD[5] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 }
[get_ports { LED_BCD[6] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 }
[get_ports { LED_BCD[7] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
##7 segment display

set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 }
[get_ports { SEG[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 }
[get_ports { SEG[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 }
[get_ports { SEG[2] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 }
[get_ports { SEG[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 }
[get_ports { SEG[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 }
[get_ports { SEG[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 }
[get_ports { SEG[6] }]; #IO_L4P_T0_D04_14 Sch=cg

set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 }
[get_ports { dp }]; #IO_L19N_T3_A21_VREF_15 Sch=dp

set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 }
[get_ports { AN[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 }
[get_ports { AN[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 }
[get_ports { AN[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 }
[get_ports { AN[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 }
[get_ports { AN[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 }
```

```
[get_ports { AN[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 }
[get_ports { AN[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 }
[get_ports { AN[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]


##Buttons

#set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 }
[get_ports { CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 }
[get_ports { BTN1 }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 }
[get_ports { RST }]; #IO_L4N_T0_D05_14 Sch=btnu
```

Simulation:

Top_lab5_tb.v



```
`timescale 1ns / 1ps

module top_lab5_tb(

    );

    reg CLK_tb;
    reg [4:0] SW_tb;
    reg RST_tb; // reset
    reg BTN1_tb; // dir
    wire [4:0] LED_SW_tb;
    wire [7:0] LED_BCD_tb;
    wire [6:0] SEG_tb;
    wire [7:0] AN_tb;

    top_lab5 tb(
        .CLK(CLK_tb),
        .SW(SW_tb),
        .RST(RST_tb),
        .BTN1(BTN1_tb),
        .LED_SW(LED_SW_tb),
        .LED_BCD(LED_BCD_tb),
        .SEG(SEG_tb),
        .AN(AN_tb)
    );

    always #5 CLK_tb = ~CLK_tb;

    initial
        begin
            CLK_tb = 0;
            SW_tb = 5'd31; // fastest
            RST_tb = 0;
            BTN1_tb = 0;
            #100

            RST_tb = 1; #40; RST_tb = 0;
            repeat (10) @(posedge CLK_tb);

            BTN1_tb = 1;
            RST_tb = 1; #40; RST_tb = 0;
            repeat (10) @(posedge CLK_tb);

            $finish;
        end

endmodule
```

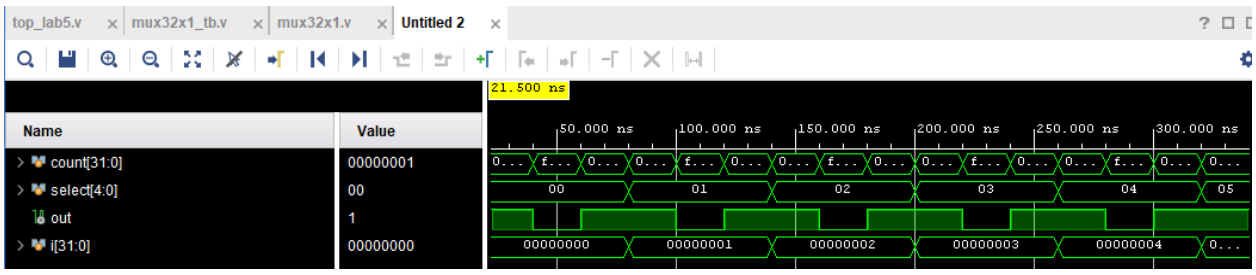
The clock division rates were equivalent with the lowest value giving the slowest count speed and 31 giving the fastest speed of 50 Mhz because of the divided clock.

sel	Selected Bit	Frequency
0	`cnt[31]`	~0.000023 Hz
15	`cnt[16]`	~1.5 kHz
31	`cnt[0]`	50 MHz

Mux32x1_tb.v

```
module mux32x1_tb(
    );

    reg [31:0] count;
    reg [4:0] select;
    wire out;
    mux32x1 TBX (
           .cnt(count),
          .sel(select),
          .clk_out(out)
           );
           integer i;
           initial
               begin
                   count = 32'd1;
                   select = 0;
               #20
           for (i = 0; i < 32; i = i + 1) begin

                   select = i;
                   #20
                   count = ~count;
                   #20
                   count = ~count;
                   #20
                   count = count << 1;
                   end
           $finish;
    end
endmodule
```

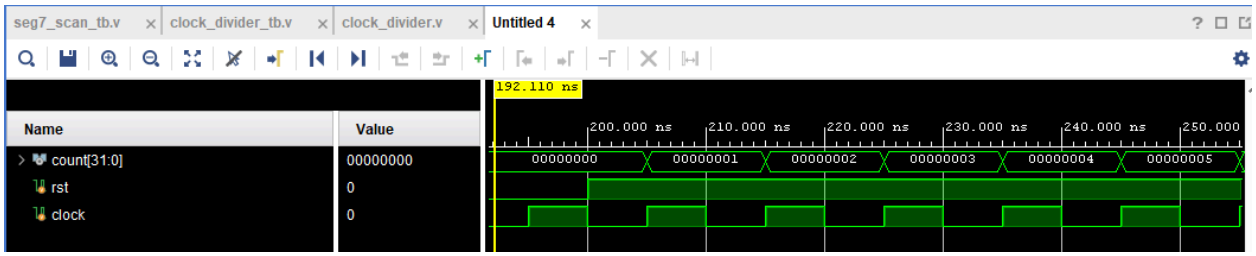
Waveform:



clock_divider_tb.v

```
module clock_divider_tb;
    wire [31:0] count;
    reg rst;
    reg clock;

    clock_divider inc(
        .cnt(count),
        .rst_n(rst),
        .clk(clock)
    );

    always #5 clock = ~clock;

    initial begin
        clock = 0;
        rst = 1;        // inactive reset
        #100 rst = 0;   // active reset
        #100 rst = 1;   // release reset
        #5000;
        $finish;
    end
endmodule
```

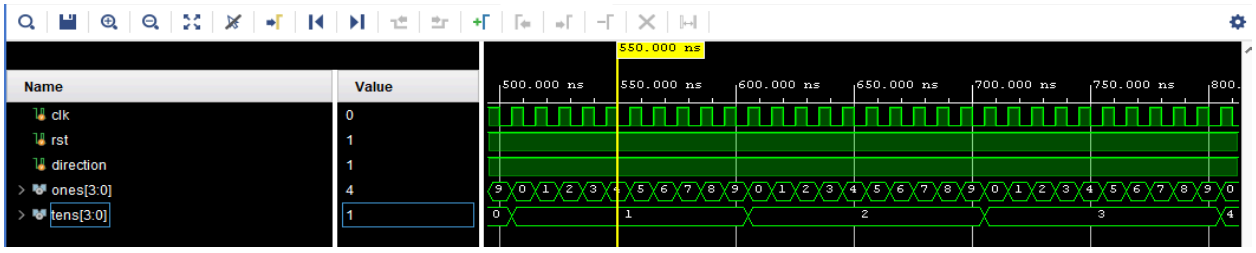
Waveform:



Bcd_up_down_counter_tb.v

```
module bcd_up_down_counter_tb(
    );
    reg clk;
    reg rst;
    reg direction;
    wire [3:0] ones;
    wire [3:0] tens;
    bcd_up_down_counter cntr(
            .clk_div(clk),
            .rst_n(rst),
            .dir(direction),
            .digit0(ones),
            .digit1(tens)
            );
    always #5 clk = ~clk;
    initial
        begin
            clk = 0;
            rst = 0;
            direction = 0;
            #100;
            direction = 1;
            #100
            rst = 1;
            direction = 0;
            #100;
            direction = 1;
            #100
            $finish;
        end
endmodule
```

Waveform:

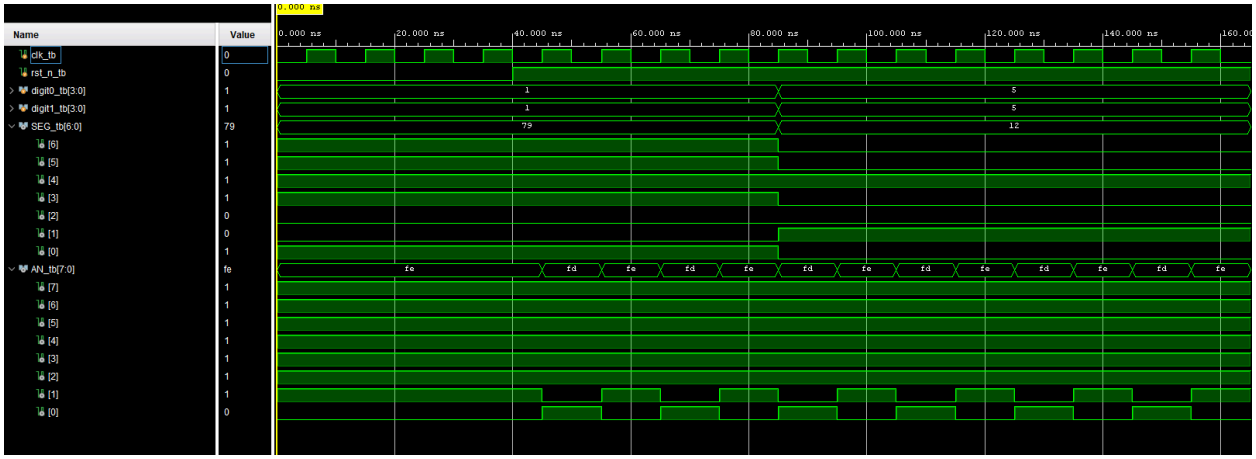


Seg7_scan_tb.v

```
module seg7_scan_tb(

    );

    reg clk_tb, rst_n_tb;
    reg [3:0] digit0_tb, digit1_tb;
    wire [6:0] SEG_tb;
    wire [7:0] AN_tb;

    seg7_scan tb(
        .clk(clk_tb),
        .rst_n(rst_n_tb),
        .digit0(digit0_tb),
        .digit1(digit1_tb),
        .SEG(SEG_tb),
        .AN(AN_tb)
    );

    always #5 clk_tb = ~clk_tb;

    task check_output;
        $display("Time=%0t, AN=%b, SEG=%b", $time, AN_tb, SEG_tb);
    endtask
```

```
    initial
        begin
            clk_tb = 0;
            rst_n_tb = 0;
            digit0_tb = 4'd1;
            digit1_tb = 4'd1;
            #40

            rst_n_tb = 1;
            repeat (5)
                begin
                    @(posedge clk_tb);
                    check_output;
                end

            digit0_tb = 4'd5;
            digit1_tb = 4'd5;
            #40
            repeat (5)
                begin
                    @(posedge clk_tb);
                    check_output;
                end


            $finish;

        end

endmodule
```

Waveform:



Implementation: Resource utilization table(s):

Slice LUTs (Logic):

Used: 30 of 63,400 → 0.05% utilization

All LUTs are used as logic, none as memory

Slice Registers (Flip-Flops):

Used: 66 of 126,800 → 0.05% utilization

Muxes:

F7 Muxes: 4 used (0.01%), F8: 0 used

Register Behavior:

Clock Enable: All 66 registers use clock enable logic

Asynchronous Reset: 60 registers include async reset

Synchronous Reset: 6 registers include sync reset

Memory:

Block RAM Tiles & Subtypes (RAMB36, RAMB18, FIFO)

Used: 0 of 135 tiles → **0.00% utilization**

Bonded IOBs:

Used: 36 of 210 → **17.14% utilization**

Group video link:

https://youtu.be/Dry0O3-prio?si=hrmUwiDbmuPtNf8r

Contributions:

Equal contributions were done across the board. Coding and development were done separately and then intermittent collaboration was done for debugging, brainstorming, and simulation. The testbenches were done with each others code and were developed together as well. Lastly, the modules were integrated together with both partners working on the top_lab5.v module to integrate the circuit. The paper was handled by both partners with Jetts Crittenden filming the demonstration video and Evan Tram posting simulation and waveforms. Overall, equal effort was demonstrated.