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

Digital Circuit Design Using Verilog Laboratory
ECE 3300L

Lab 7

16-bit Barrel Shifter / Rotator & 4-Digit 7-Segment Display

By

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

8/7/2025

Design:

This design uses the basic principles from previous labs and reuses them in tandem with new methods. This project uses the generation of different trees of shifts with basic logic design to provide a system that allows the user to shift, change shift type (Logical / Rotate), and the direction of the shift. It controls these inputs through debounced and controlled inputs from the push buttons on the Nexys board. The original word value is also input through the switches on the board. It then displays the shifted 4-digit hex value on the 7-segment board just like in previous labs.

Code:

top_lab7.v

```
`timescale 1ns / 1ps

module top_lab7(
        input wire clk,                 // 100 MHz system clock
/
        input wire [15:0] SW,           // Slide switches for input
word
        input wire [4:0] BTN,           // BTN[4] = BTNC, BTN[3:0] =
BTNU, BTND, BTNL, BTNR
        output wire [7:0] LED,          // Optional debug LEDs
        output wire [6:0] SEG,          // 7-segment segments
        output wire DP,
        output wire [7:0] AN            // Digit enables
    );

        wire rst_n;
        wire clk_rst;
        assign DP = 1;
        wire clk_1kHz, clk_2Hz;
            clock_divider_fixed clkdiv (
                .clk(clk),
                .rst_n(clk_rst),
                .clk_out(clk_1kHz)
            );

        // Debounced Button Toggles
```

```
        wire dir, rot;
        wire [1:0] shamtl;
        wire [1:0] shamtm;
        wire btnc_toggle;

                debounce_toggle db_dir (.clk(clk_1kHz),
.btn_raw(BTN[0]), .btn_toggle(dir));      // BTNL
                debounce_toggle db_rot (.clk(clk_1kHz),
.btn_raw(BTN[1]), .btn_toggle(rot));      // BTNR
                debounce_toggle db_shamt0 (.clk(clk_1kHz),
.btn_raw(BTN[2]), .btn_toggle(shamtl[0])); // BTND
                debounce_toggle db_shamt1 (.clk(clk_1kHz),
.btn_raw(BTN[3]), .btn_toggle(shamtl[1])); // BTNU


    //SHAMT Counter (MSBs)
        wire btnc_raw = BTN[4];
        assign rst_n = ~btnc_raw;
        assign clk_rst = 1; // reset disable for now
                shamt_counter shamt_ctrl (
                    .clk(clk_1kHz),            // 1 kHz clock
                    .btn_raw(btnc_raw),        // Raw BTNC input
                    .shamt(shamtm)     // One-cycle pulse for SHAMT
increment
        );



        wire [3:0] shamt = {shamtm, shamtl};  // Combine MSBs and
LSBs

    // Barrel Shifter
        wire [15:0] barrel_out;
            barrel_shifter16 shifter (
                .data_in(SW[15:0]),
                .shamt(shamt),
                .dir(dir),
                .rotate(rot),
                .data_out(barrel_out)
```

```
            );

    // Segment Scanner
        wire [3:0] ones, tens, hundreds, thousands;
        assign ones = barrel_out[3:0];
        assign tens = barrel_out[7:4];
        assign hundreds = barrel_out[11:8];
        assign thousands = barrel_out[15:12];
            seg7_scan8 scanner (
                .clk(clk),
                .rst_n(rst_n),
                .onesPlace(ones),
                .tensPlace(tens),
                .hundredsPlace(hundreds),
                .thousandsPlace(thousands),
                .SEG(SEG),
                .AN(AN)
            );



    // 💡 Optional Debug LEDs
    assign LED[7] = dir;
    assign LED[6] = rot;
    assign LED[5:2] = shamt;
    assign LED[1:0] = 2'b00;  // Unused or reserved
endmodule
```

barrel_shifter16.v
`timescale 1ns / 1ps

```
module barrel_shifter16(
    input wire [15:0] data_in,
    input wire [3:0] shamt,
    input wire dir,           // 0 = left, 1 = right
    input wire rotate,        // 0 = logical, 1 = rotate
    output wire [15:0] data_out
```

```
    );




    wire [15:0] stage [0:4];  // 5 stages: input + 4 shift stages
        assign stage[0] = data_in;

        genvar i;
            generate
                for (i = 0; i < 4; i = i + 1) begin : shift_stage
                    wire [15:0] shifted;
                    wire [15:0] fill;

                    // Shift amount for this stage: 2^i
                    localparam integer SHIFT = 1 << i;

                    // Fill bits depend on rotate vs logical
                    assign fill = (rotate) ?
                        (dir ? stage[i] << (16 - SHIFT) : stage[i] >>
(16 - SHIFT)) :
                        16'b0;

            assign shifted = (dir) ?
                (stage[i] >> SHIFT) | fill :
                (stage[i] << SHIFT) | fill;

            assign stage[i+1] = (shamt[i]) ? shifted : stage[i];
        end
    endgenerate

    assign data_out = stage[4];

endmodule
```

clock_divider_fixed.v

```
`timescale 1ns / 1ps
```

```
module clock_divider_fixed(
    input wire clk,           // 100 MHz input clock
    input wire rst_n,         // Active-low reset
    output reg clk_out        // Slower output clock
);

    parameter DIV_VALUE = 26'd50_000;  // Half-period count
    reg [25:0] cnt;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            cnt <= 0;
            clk_out <= 0;
        end else begin
            if (cnt == DIV_VALUE - 1) begin
                cnt <= 0;
                clk_out <= ~clk_out;  // Toggle output clock
            end else begin
                cnt <= cnt + 1;
            end
        end
    end

endmodule
```

debounce_toggle.v

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

    wire btn_clean;
    reg btn_prev;

    // Debounce module (assumed to be defined elsewhere)
    debounce db (
```

```
        .clk(clk),
        .btn_in(btn_raw),
        .btn_clean(btn_clean)
    );

    always @(posedge clk ) begin

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

endmodule
```

shamt_counter.v

```
`timescale 1ns / 1ps

module shamt_counter (
    input wire clk,             // 1 kHz clock
    input wire btn_raw,         // Raw BTNC input
    output reg [1:0] shamt      // One-cycle pulse for SHAMT increment
);
    reg shamt_pulse;
    wire btn_clean;
    reg btn_prev;

    // Debounce the raw button
    debounce db (
        .clk(clk),
        .btn_in(btn_raw),
        .btn_clean(btn_clean)
    );

    // One-cycle pulse on rising edge of debounced button
    always @(posedge clk) begin
        btn_prev <= btn_clean;
        shamt_pulse <= btn_clean & ~btn_prev;
```

```
        if (shamt_pulse) begin
            shamt <= shamt + 1;
        end
    end

endmodule
```

hex_to_7seg.v

```
`timescale 1ns / 1ps

module hex_to_7seg(
    input wire [3:0] digitVal,
    output reg [6:0] SEG
    );
    always@(*)
        begin
            case(digitVal)
                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
                4'd10: SEG = 7'b0001000; // A
                4'd11: SEG = 7'b0000011; // b
                4'd12: SEG = 7'b1000110; // C
                4'd13: SEG = 7'b0100001; // d
                4'd14: SEG = 7'b0000110; // E
                4'd15: SEG = 7'b0001110; // F
    default: SEG = 7'b1111111; // Blank
endcase
        end
```

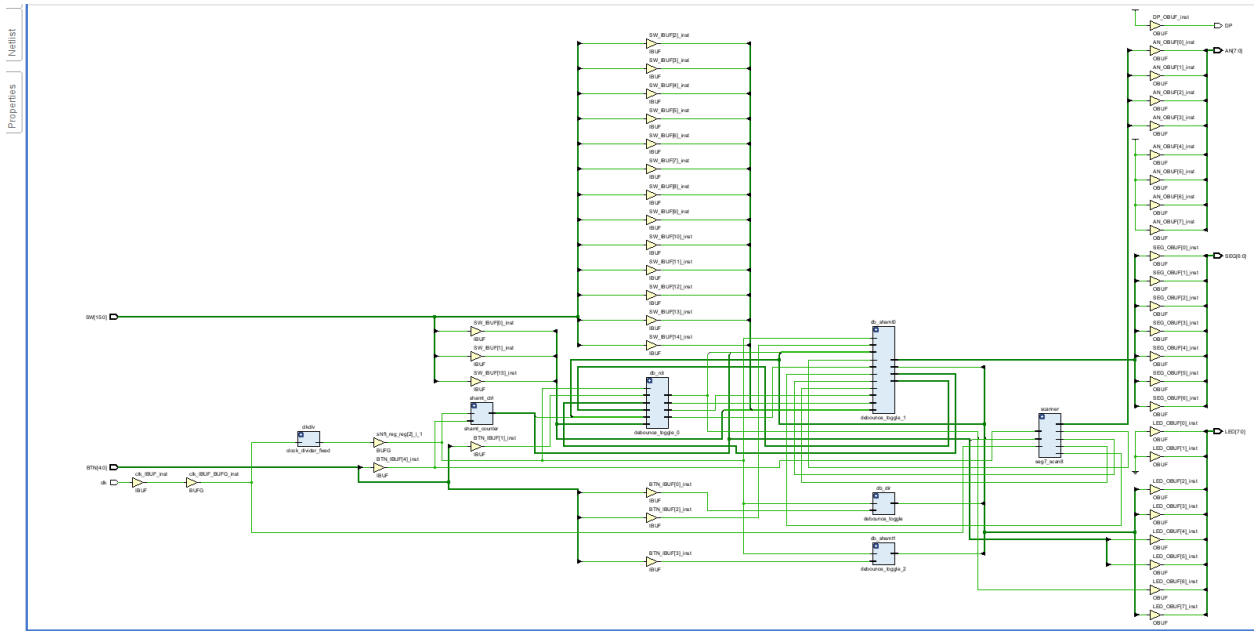
```
endmodule
```

seg7_scan8.v

```
`timescale 1ns / 1ps

module seg7_scan8(
    input wire clk,
    input wire rst_n,
    input wire [3:0] onesPlace,
    input wire [3:0] tensPlace,
    input wire [3:0] hundredsPlace,
    input wire [3:0] thousandsPlace,
    output wire [6:0] SEG,
    output reg [7:0] AN
);

    reg [19:0] refresh_counter = 0;
    wire [1:0] sel;
    reg [3:0] digit_val;
    wire [6:0] seg_val;
    assign SEG = seg_val;
    assign sel = refresh_counter[19:18];

    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)
                2'd0:
                    begin
                        AN = 8'b11111110;
```

```
                    digit_val = onesPlace;
                end
            2'd1:
                begin
                    AN = 8'b11111101;
                    digit_val = tensPlace;
                end
            2'd2:
                begin
                    AN = 8'b11111011;
                    digit_val = hundredsPlace;
                end
            2'd3:
                begin
                    AN = 8'b11110111;
                    digit_val = thousandsPlace;
                end
            default: AN = 8'b11111111;
        endcase
    end

  hex_to_7seg decoder (
      .digitVal(digit_val),
      .SEG(seg_val)
   );

endmodule
```

Hardware Schematic:



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]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 }
[get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 }
[get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
```

```
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 }
[get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 }
[get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 }
[get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 }
[get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 }
[get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 }
[get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 }
[get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 }
[get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 }
[get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 }
[get_ports { LED[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 }
[get_ports { LED[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 }
[get_ports { LED[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 }
[get_ports { LED[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 }
[get_ports { LED[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 }
[get_ports { LED[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 }
[get_ports { LED[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 }
[get_ports { LED[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 }
[get_ports { SEG[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
```

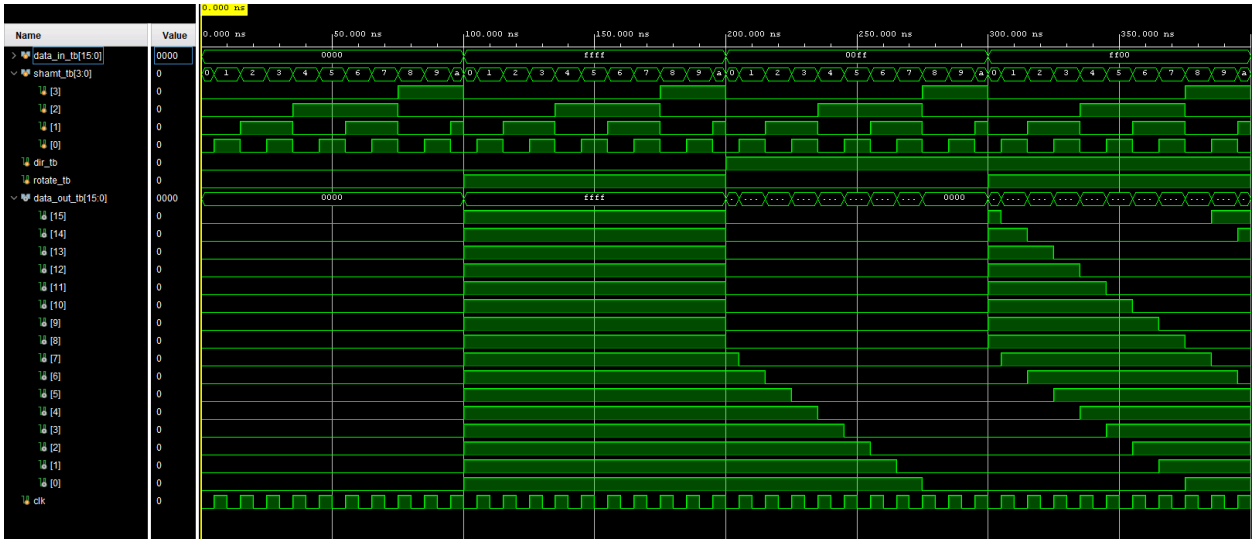
```
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 }
[get_ports { SEG[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 }
[get_ports { SEG[2] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 }
[get_ports { SEG[3] }]; #IO_L17P_T2_A26_15 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]

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

##Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 }
[get_ports { BTN[4] }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 }
```

```
[get_ports { BTN[3] }]; #IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 }
[get_ports { BTN[0] }]; #IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 }
[get_ports { BTN[1] }]; #IO_L10N_T1_D15_14 Sch=btnr
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 }
[get_ports { BTN[2] }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd
```

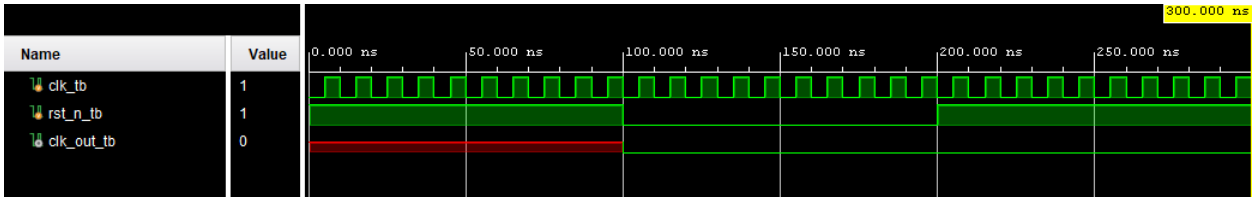
Simulation:

The testbench code was withheld from the report to avoid distracting from the content. All testbench code is found in the GitHub repository.

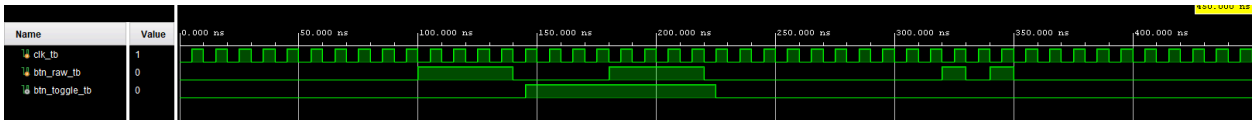
barrel_shifter16_tb.v



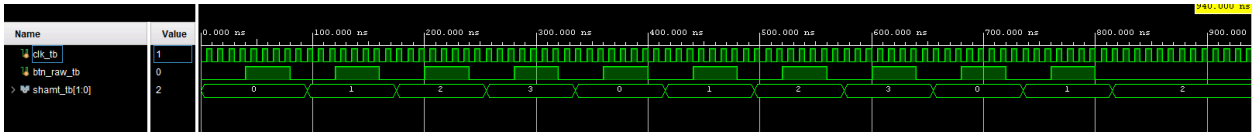
clock_divider_fixed_tb.v



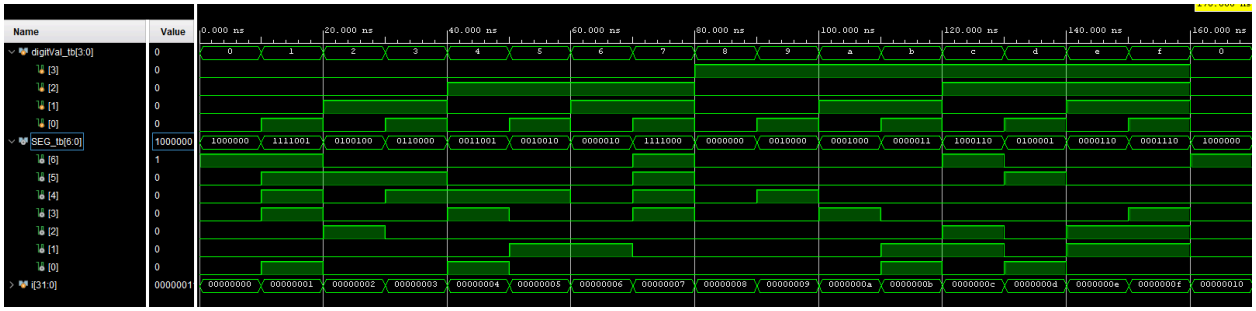
debounce_toggle_tb.v



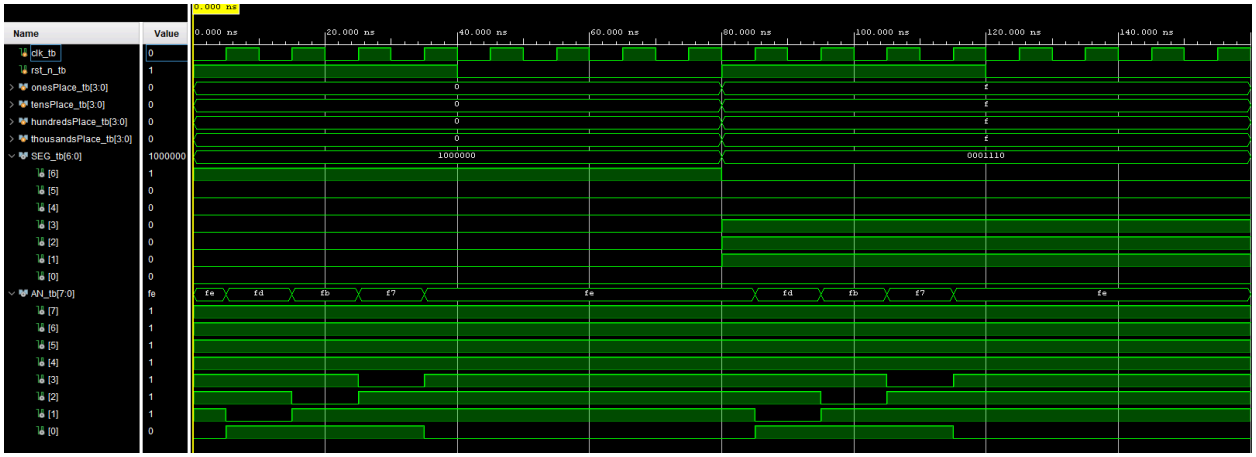
shamt_counter_tb.v



hex_to_7seg_tb.v



seg7_scan8_tb.v



Implementation: Resource utilization table(s):

Slice Logic Utilization:

LUTs Used: **80 used 0.13% Utilization**

Registers Used: 79 used 0.06% Utilization

Muxes:

F7: 0 used

F8: 0 used

Registers:

Registers used as Flip-flops

IO: 46 IO used , 21.9% Utilization

Timing:

Worst Negative Slack: 4.740 ns, 72 endpoints

Worst Hold Slack: 0.140 ns, 72 endpoints

Worst Pulse Width Slack: 4.500 ns, 48 endpoints

Timing constraints are met, providing adequate slack.

Group video link:

https://youtu.be/5cZJpbCB5Aw?si=F6NKejCM2gSPB-SU

Contributions:

The contributions of this report were equally split. Both Jetts Crittenden and Evan Tram worked on the modules and testbenches separately and collaborated during the discussion and the development of the top_lab7.v module to appropriately combine the modules. Many of the modules used in this lab were used in previous labs, such as the 7-segment scan module, which was also developed with equal effort on both parts of the team. The demo video was recorded by Jetts Crittenden, and synthesis reports were recorded by Evan Tram, with the paper being written by both team members.