3300 Lab 7
16-bit Barrel Shifter / Rotator & 4-Digit 7-Segment Display

Group C
Rohan Walia
Parsa Ghasemi
8/6/25

```
`timescale 1ns / 1ps

module top_lab7(
    input wire CLK,
    input wire [15:0] SW,
    input wire [4:0] BTN,
    output wire [7:0] LED,
    output wire [6:0] SEG,
    output wire [7:0] AN
);

    wire clk1k, clk_2Hz;
    clock_divider_fixed clkdiv_inst (
        .clk(CLK),
        .clk_1kHz(clk1k),
        .clk_2Hz(clk_2Hz)
    );

    wire dir_toggle, rot_toggle, shamt0_toggle, shamt1_toggle;
    debounce_toggle db_dir (.clk1k(clk1k), .btn_raw(BTN[0]), .btn_toggle(dir_toggle));
    debounce_toggle db_rot (.clk1k(clk1k), .btn_raw(BTN[1]), .btn_toggle(rot_toggle));
    debounce_toggle db_sh0 (.clk1k(clk1k), .btn_raw(BTN[2]), .btn_toggle(shamt0_toggle));
    debounce_toggle db_sh1 (.clk1k(clk1k), .btn_raw(BTN[3]), .btn_toggle(shamt1_toggle));

    wire tick;
    debounce_tick db_btnc (.clk1k(clk1k), .btn_raw(BTN[4]), .tick(tick));

    wire [1:0] shamt_high_bits;
    shamt_counter shamt_ctr_inst (
        .clk1k(clk1k),
        .tick(tick),
        .shamt_high(shamt_high_bits)
    );

    wire [3:0] shamt = {shamt_high_bits, shamt1_toggle, shamt0_toggle};

    wire [15:0] shifted_output;
    barrel_shifter16 shifter_inst (
        .data_in(SW),
        .shamt(shamt),
        .dir(dir_toggle),
        .rotate(rot_toggle),
        .data_out(shifted_output)
    );

    assign LED = {
        shamt_high_bits,
        rot_toggle,
        dir_toggle,
        shamt1_toggle,
        shamt0_toggle,
        2'b00
    };

    wire [6:0] seg3, seg2, seg1, seg0;
    hex_to_7seg hex3 (.hex(shifted_output[15:12]), .seg(seg3));
    hex_to_7seg hex2 (.hex(shifted_output[11:8]),  .seg(seg2));
    hex_to_7seg hex1 (.hex(shifted_output[7:4]),   .seg(seg1));
    hex_to_7seg hex0 (.hex(shifted_output[3:0]),   .seg(seg0));

    wire [55:0] seg_data = {
        7'h7F, 7'h7F, 7'h7F, 7'h7F,
        seg3, seg2, seg1, seg0
    };

    seg7_scan8 scanner_inst (
        .clk(clk1k),
        .seg_data(seg_data),
        .an(AN),
        .seg(SEG)
    );

endmodule
```

```
`timescale 1ns / 1ps

module barrel_shifter16(
   input wire [15:0] data_in,
   input wire [3:0] shamt,
   input wire dir,
   input wire rotate,
   output wire [15:0] data_out
);

   wire [15:0] stage [0:4];
   assign stage[0] = data_in;

   genvar i;
   generate
      for (i = 0; i < 4; i = i + 1) begin : STAGES
         wire [15:0] current_data = stage[i];

         wire [15:0] logical_left  = current_data << (1 << i);
         wire [15:0] logical_right = current_data >> (1 << i);

         wire [15:0] rotate_left  = {current_data[15 - (1 << i):0], current_data[15:16 - (1 << i)]};
         wire [15:0] rotate_right = {current_data[(1 << i) - 1:0], current_data[15:(1 << i)]};

         wire [15:0] left_result  = rotate ? rotate_left : logical_left;
         wire [15:0] right_result = rotate ? rotate_right : logical_right;

         wire [15:0] shifted_data = (dir == 1'b0) ? left_result : right_result;

         assign stage[i + 1] = shamt[i] ? shifted_data : current_data;
      end
   endgenerate

   assign data_out = stage[4];

endmodule
```

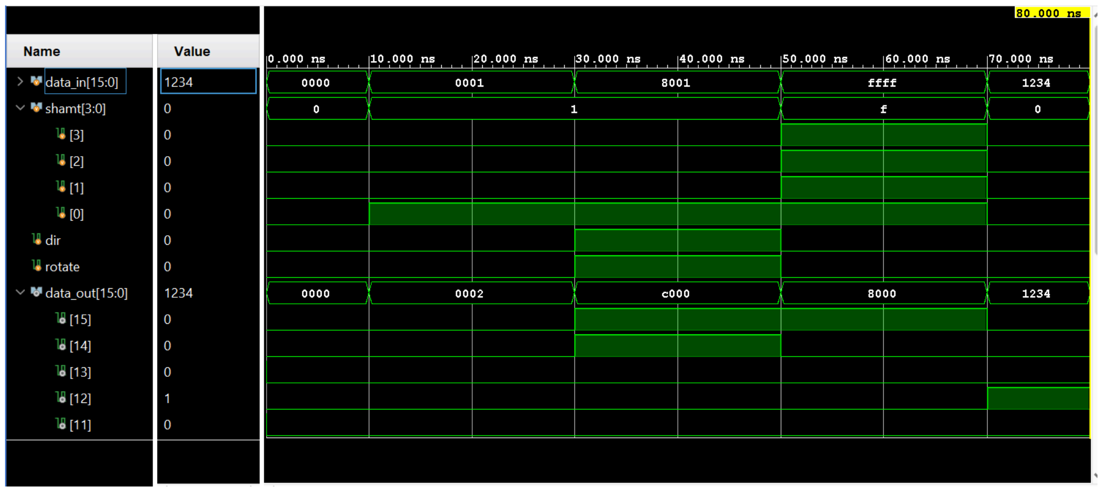
```
`timescale 1ns / 1ps

module clock_divider_fixed(
    input wire clk,
    output reg clk_1kHz = 1'b0,
    output reg clk_2Hz = 1'b0
);

    localparam COUNT_1KHZ_MAX = 16'd50_000;
    reg [15:0] counter_1kHz = 16'd0;

    localparam COUNT_2HZ_MAX = 25'd25_000_000;
    reg [24:0] counter_2Hz = 25'd0;

    always @(posedge clk) begin
        if (counter_1kHz == COUNT_1KHZ_MAX - 1) begin
            counter_1kHz <= 16'd0;
            clk_1kHz <= ~clk_1kHz;
        end else
            counter_1kHz <= counter_1kHz + 1;
    end

    always @(posedge clk) begin
        if (counter_2Hz == COUNT_2HZ_MAX - 1) begin
            counter_2Hz <= 25'd0;
            clk_2Hz <= ~clk_2Hz;
        end else
            counter_2Hz <= counter_2Hz + 1;
    end

endmodule
```

```
module shamt_counter(
    input  wire clk1k,
    input  wire tick,
    output reg [1:0] shamt_high = 0
);
    always @(posedge clk1k) begin
        if (tick) shamt_high <= shamt_high + 1;
    end
endmodule



`timescale 1ns / 1ps

module hex_to_7seg(
    input wire [3:0] hex,
    output reg [6:0] seg
);
    always @(*) begin
        case (hex)
            4'h0: seg = 7'b0000001;
            4'h1: seg = 7'b1001111;
            4'h2: seg = 7'b0010010;
            4'h3: seg = 7'b0000110;
            4'h4: seg = 7'b1001100;
            4'h5: seg = 7'b0100100;
            4'h6: seg = 7'b0100000;
            4'h7: seg = 7'b0001111;
            4'h8: seg = 7'b0000000;
            4'h9: seg = 7'b0001100;
            4'hA: seg = 7'b0001000;
            4'hB: seg = 7'b1100000;
            4'hC: seg = 7'b0110001;
            4'hD: seg = 7'b1000010;
            4'hE: seg = 7'b0110000;
            4'hF: seg = 7'b0111000;
            default: seg = 7'b1111111;
        endcase
    end
endmodule
```

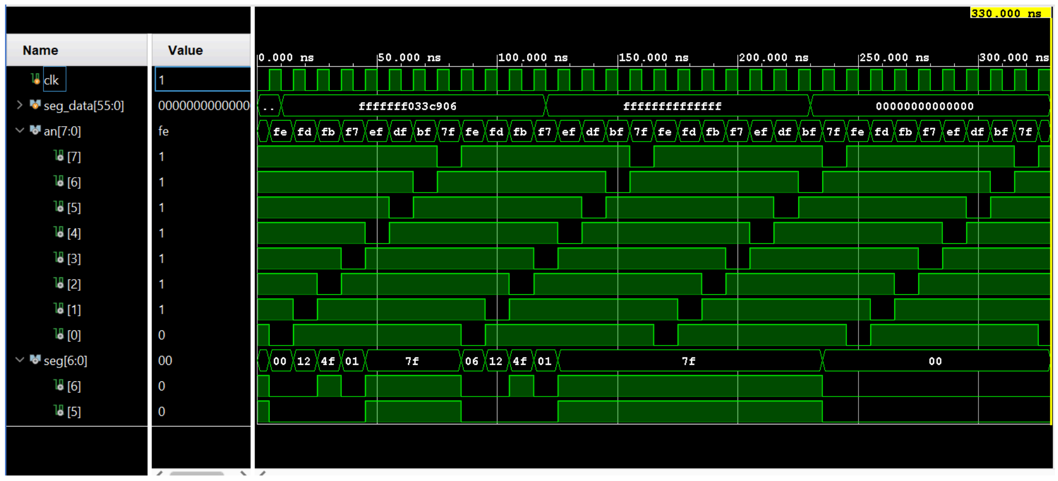
```
module debounce_toggle(
    input wire clk1k,
    input wire btn_raw,
    output reg btn_toggle = 1'b0
);
    wire tick;
    debounce_tick dt_inst (
        .clk1k(clk1k),
        .btn_raw(btn_raw),
        .tick(tick)
    );

    always @(posedge clk1k) begin
        if (tick) begin
            btn_toggle <= ~btn_toggle;
        end
    end
endmodule

module debounce_tick(
    input wire clk1k,
    input wire btn_raw,
    output reg tick = 1'b0
);
    reg [2:0] shift_reg = 3'b000;
    reg debounced_state = 1'b0;
    reg prev_debounced_state = 1'b0;

    always @(posedge clk1k) begin
        shift_reg <= {shift_reg[1:0], btn_raw};

        if (shift_reg == 3'b111) debounced_state <= 1'b1;
        else if (shift_reg == 3'b000) debounced_state <= 1'b0;

        prev_debounced_state <= debounced_state;

        tick <= (debounced_state == 1'b1 && prev_debounced_state == 1'b0);
    end
endmodule
```

```
module seg7_scan8(
    input wire clk,
    input wire [55:0] seg_data,
    output reg [7:0] an = 8'b11111111,
    output reg [6:0] seg = 7'b1111111
);
    reg [2:0] scan_index = 3'd0;

    always @(posedge clk) begin
        scan_index <= scan_index + 1;
        an <= 8'b11111111;
        an[scan_index] <= 1'b0;

        case (scan_index)
            3'd0: seg <= seg_data[6:0];
            3'd1: seg <= seg_data[13:7];
            3'd2: seg <= seg_data[20:14];
            3'd3: seg <= seg_data[27:21];
            3'd4: seg <= seg_data[34:28];
            3'd5: seg <= seg_data[41:35];
            3'd6: seg <= seg_data[48:42];
            3'd7: seg <= seg_data[55:49];
            default: seg <= 7'b1111111;
        endcase
    end
endmodule
```

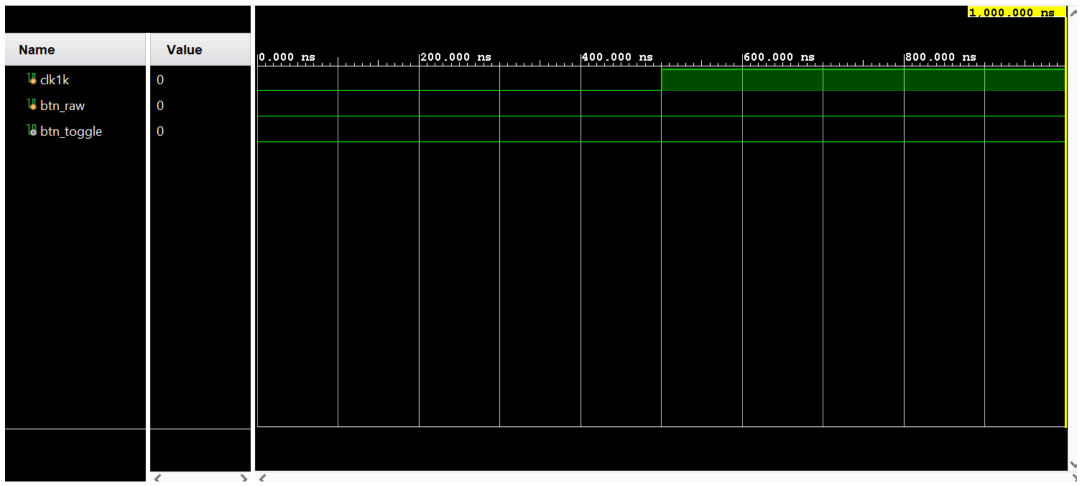
Barrel_shifter16



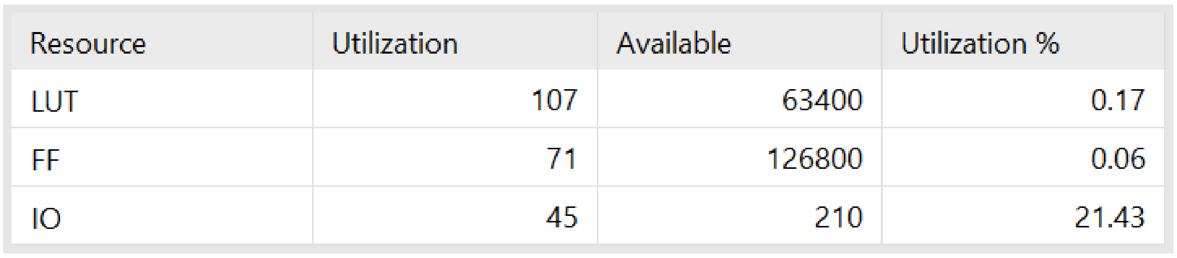
Seg7_scan8



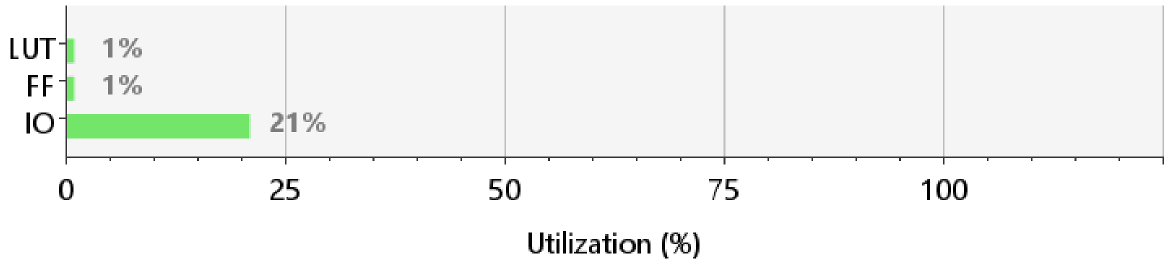
Debounce_toggle



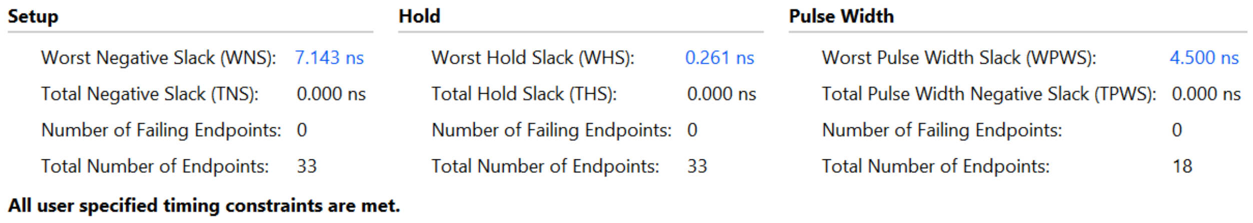
LUTs and FF screenshot:



Resource	Utilization	Available	Utilization %
LUT	107	63400	0.17
FF	71	126800	0.06
IO	45	210	21.43



Timing screenshot:



Setup

Worst Negative Slack (WNS):	7.143 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	33

Hold

Worst Hold Slack (WHS):	0.261 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	33

Pulse Width

Worst Pulse Width Slack (WPWS):	4.500 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	18

All user specified timing constraints are met.

Contribution:
Rohan Walia (50%): Implementation, demo, verilog, report
Parsa Ghasemi (50%): testbench code, testing, report