ECE 3300L

Lab Report #7

Group A

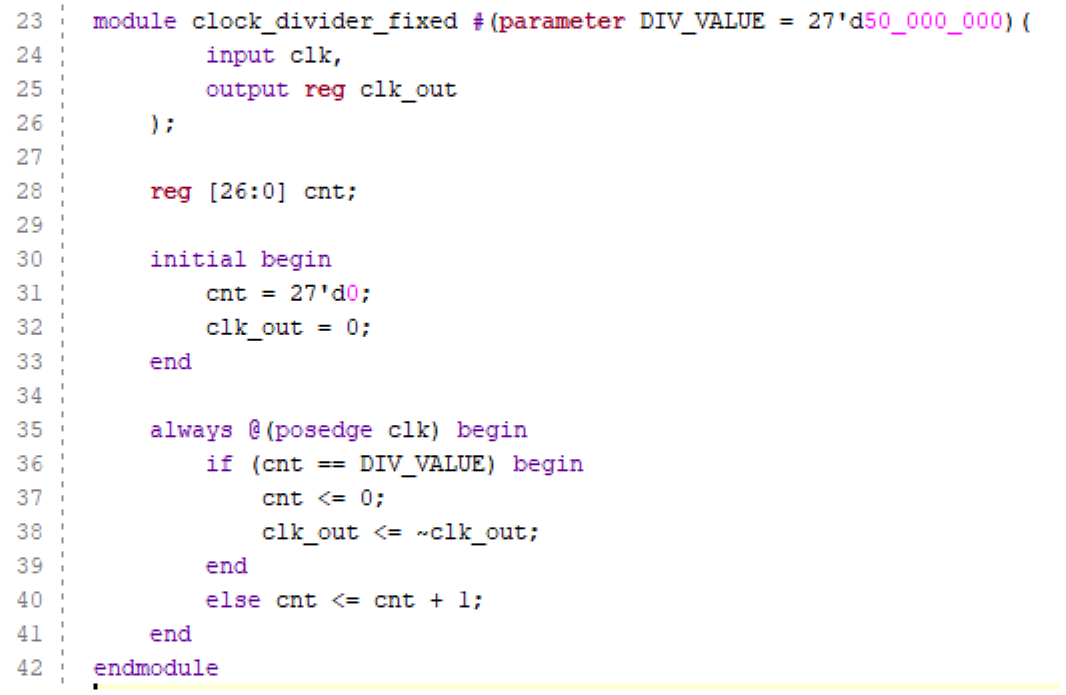
Edwin Estrada (#015897050)

Michelle Lau (#016027427)

August 6, 2025

Design

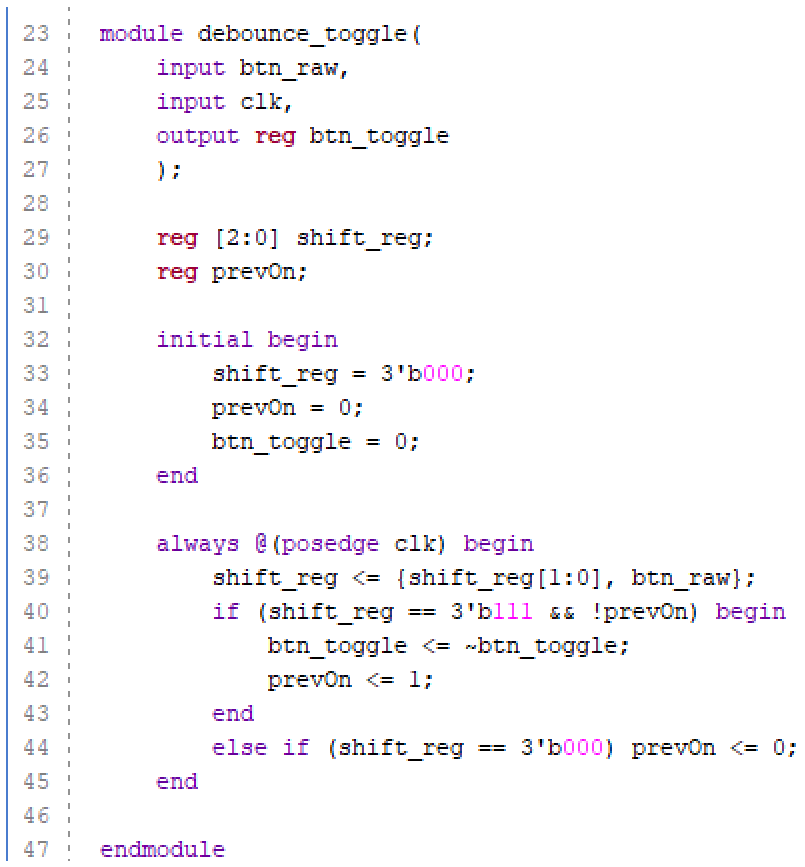
Fixed Clock Divider



```
module clock_divider_fixed #(parameter DIV_VALUE = 27'd50_000_000)(
        input clk,
        output reg clk_out
    );

    reg [26:0] cnt;

    initial begin
        cnt = 27'd0;
        clk_out = 0;
    end

    always @(posedge clk) begin
        if (cnt == DIV_VALUE) begin
            cnt <= 0;
            clk_out <= ~clk_out;
        end
        else cnt <= cnt + 1;
    end
endmodule
```

This module divides the base 100Mhz clock speed provided by the board. Included is a parameter to reuse the module to set different clock speeds. The default parameter value is set so the resulting clock is 1Hz, but the module is used to create 2Mhz and 1kHz clocks for this lab.

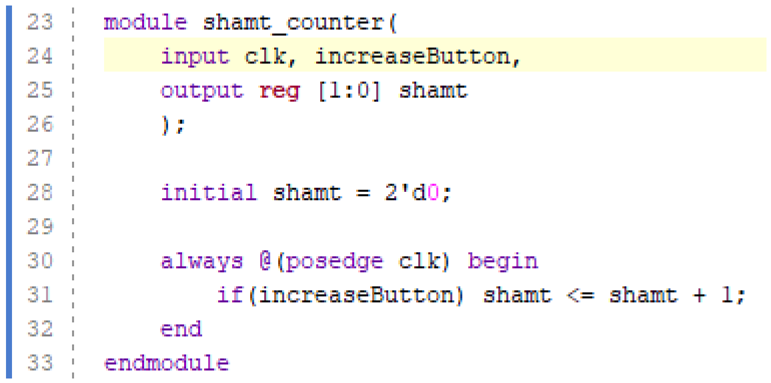
Debounce Toggle Button



```
module debounce_toggle(
    input btn_raw,
    input clk,
    output reg btn_toggle
    );

    reg [2:0] shift_reg;
    reg prevOn;

    initial begin
        shift_reg = 3'b000;
        prevOn = 0;
        btn_toggle = 0;
    end

    always @(posedge clk) begin
        shift_reg <= {shift_reg[1:0], btn_raw};
        if (shift_reg == 3'b111 && !prevOn) begin
            btn_toggle <= ~btn_toggle;
            prevOn <= 1;
        end
        else if (shift_reg == 3'b000) prevOn <= 0;
    end

endmodule
```

This module has two functions: Address button debouncing and making state toggles when its respective buttons are pressed, rather than the state depending on the state of the button.

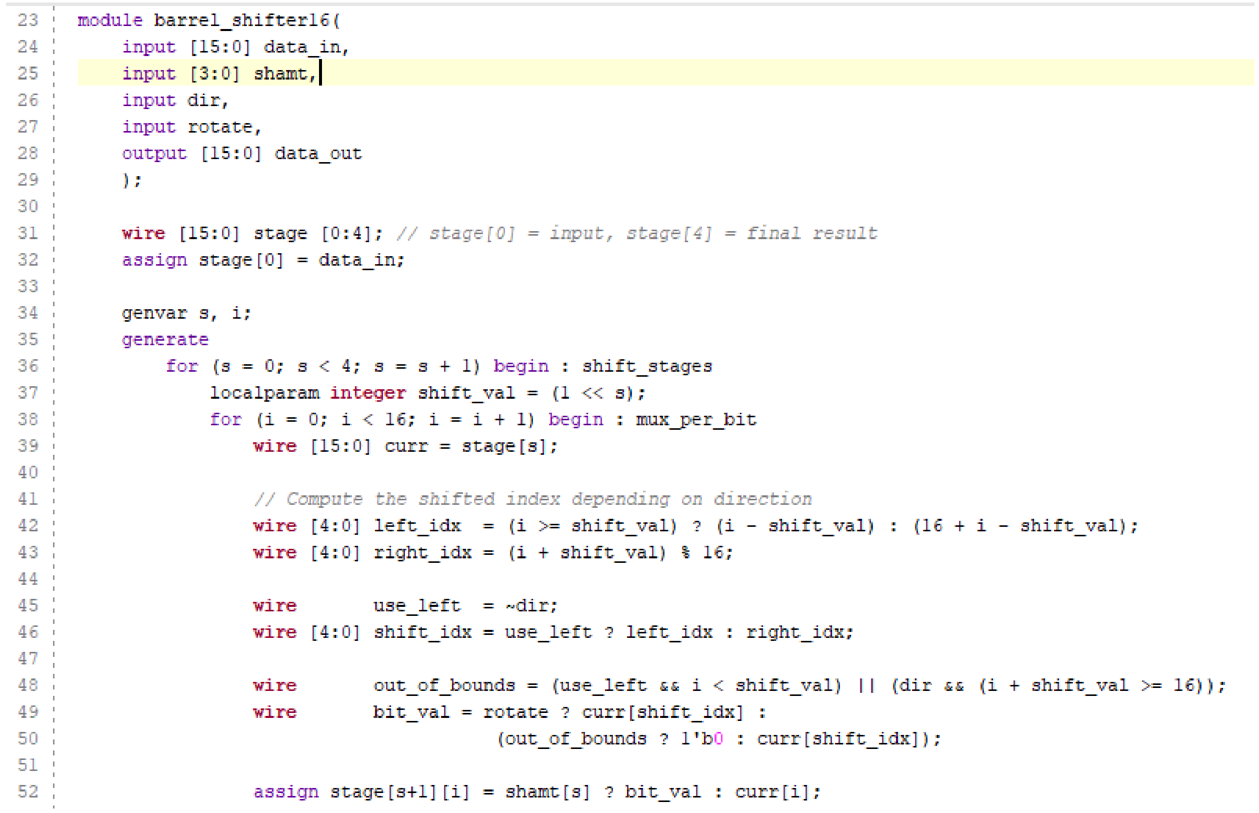
SHAMT Counter



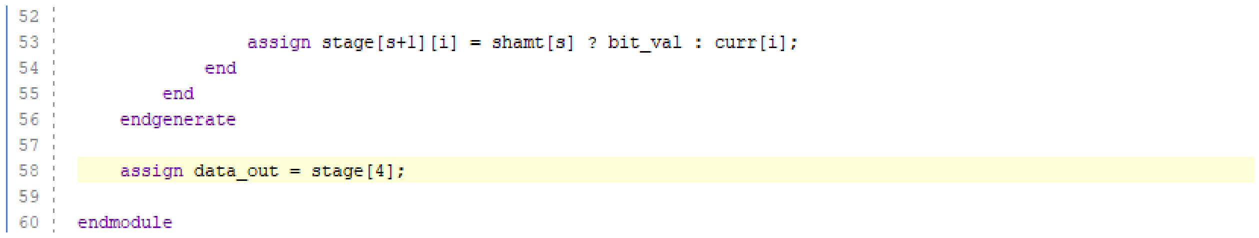
```
module shamt_counter(
    input clk, increaseButton,
    output reg [1:0] shamt
    );

    initial shamt = 2'd0;

    always @(posedge clk) begin
        if(increaseButton) shamt <= shamt + 1;
    end
endmodule
```

Since more input methods are needed, we have a state saved in memory that increments when the center button is held and the 2Hz clock is positive edge triggered. The state switches between the 4 possible states of the upper two bits of shamt, resetting to 0 after being on the third state.

16-bit Barrel Shifter



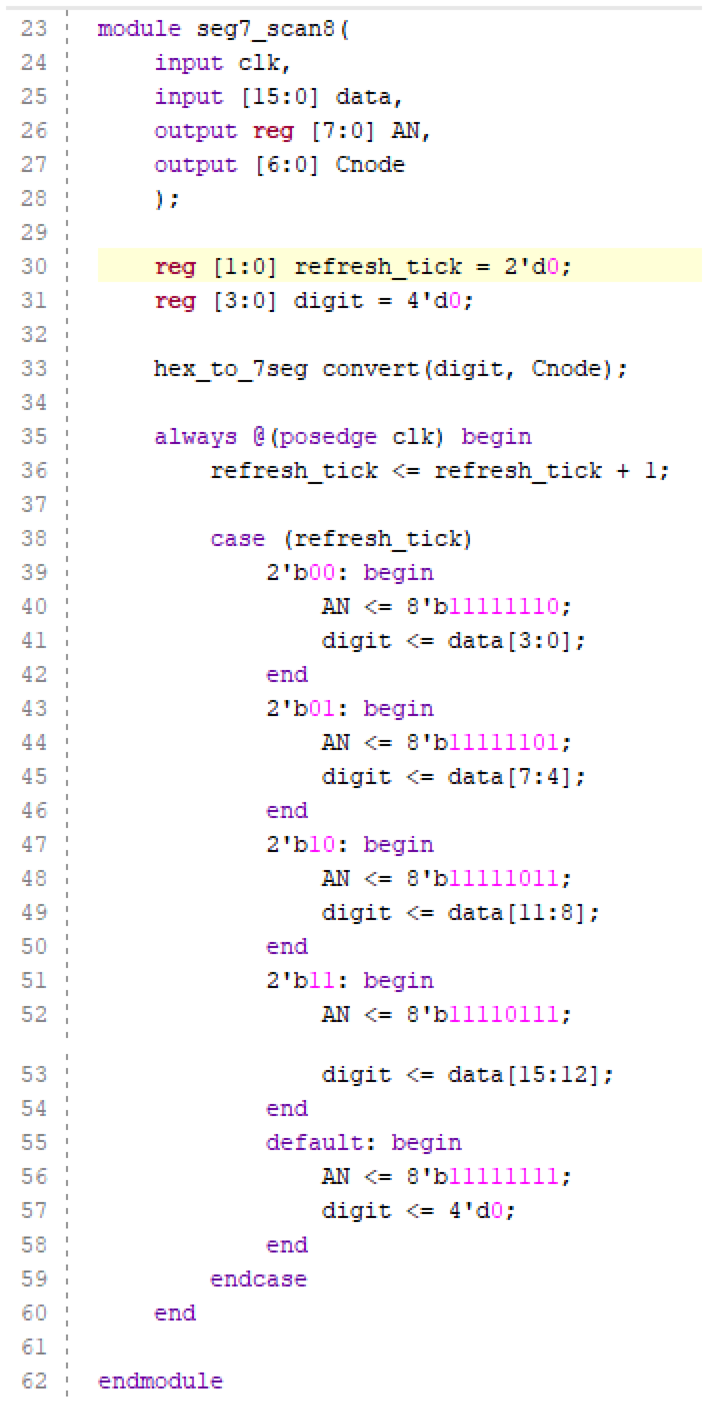
```
module barrel_shifter16(
    input [15:0] data_in,
    input [3:0] shamt,
    input dir,
    input rotate,
    output [15:0] data_out
    );

    wire [15:0] stage [0:4]; // stage[0] = input, stage[4] = final result
    assign stage[0] = data_in;

    genvar s, i;
    generate
        for (s = 0; s < 4; s = s + 1) begin : shift_stages
            localparam integer shift_val = (1 << s);
            for (i = 0; i < 16; i = i + 1) begin : mux_per_bit
                wire [15:0] curr = stage[s];

                // Compute the shifted index depending on direction
                wire [4:0] left_idx  = (i >= shift_val) ? (i - shift_val) : (16 + i - shift_val);
                wire [4:0] right_idx = (i + shift_val) % 16;

                wire       use_left  = ~dir;
                wire [4:0] shift_idx = use_left ? left_idx : right_idx;

                wire       out_of_bounds = (use_left && i < shift_val) || (dir && (i + shift_val >= 16));
                wire       bit_val = rotate ? curr[shift_idx] :
                                      (out_of_bounds ? 1'b0 : curr[shift_idx]);

                assign stage[s+1][i] = shamt[s] ? bit_val : curr[i];
```



```

                assign stage[s+1][i] = shamt[s] ? bit_val : curr[i];
            end
        end
    endgenerate

    assign data_out = stage[4];

endmodule
```

The barrel shifter has 4 stages. Stage 1 shifts the 16-bit word by 1 bit, 2 by 2 bits, 3 by 4 bits, and 4 by 8 bits. If bit 0 isn't on for example, then stage 1 results in the same word. Right and left shifts are calculated while accounting for logical and rotating shifts, handling cases where data is out of bounds. Finally, we assign the resulting 16-bit word to its resulting stage, where the final output is stored in stage[4].

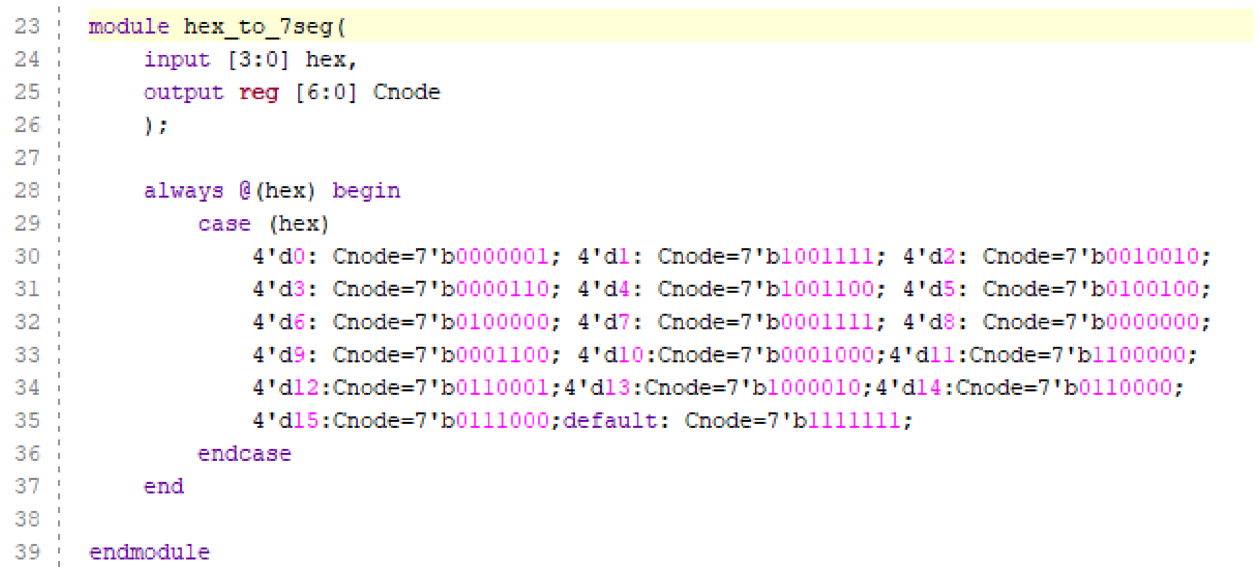
Seg7 Scan



```verilog
module seg7_scan8(
    input clk,
    input [15:0] data,
    output reg [7:0] AN,
    output [6:0] Cnode
    );

    reg [1:0] refresh_tick = 2'd0;
    reg [3:0] digit = 4'd0;

    hex_to_7seg convert(digit, Cnode);

    always @(posedge clk) begin
        refresh_tick <= refresh_tick + 1;

        case (refresh_tick)
            2'b00: begin
                AN <= 8'b11111110;
                digit <= data[3:0];
            end
            2'b01: begin
                AN <= 8'b11111101;
                digit <= data[7:4];
            end
            2'b10: begin
                AN <= 8'b11111011;
                digit <= data[11:8];
            end
            2'b11: begin
                AN <= 8'b11110111;

                digit <= data[15:12];
            end
            default: begin
                AN <= 8'b11111111;
                digit <= 4'd0;
            end
        endcase
    end

endmodule
```

This is a typical seg 7 scan, basically the same with what we have been doing for the last couple of labs. The difference now is the base clock is removed with a 1kHz clock and the hex to 7seg decoder is moved to its own module for readability.

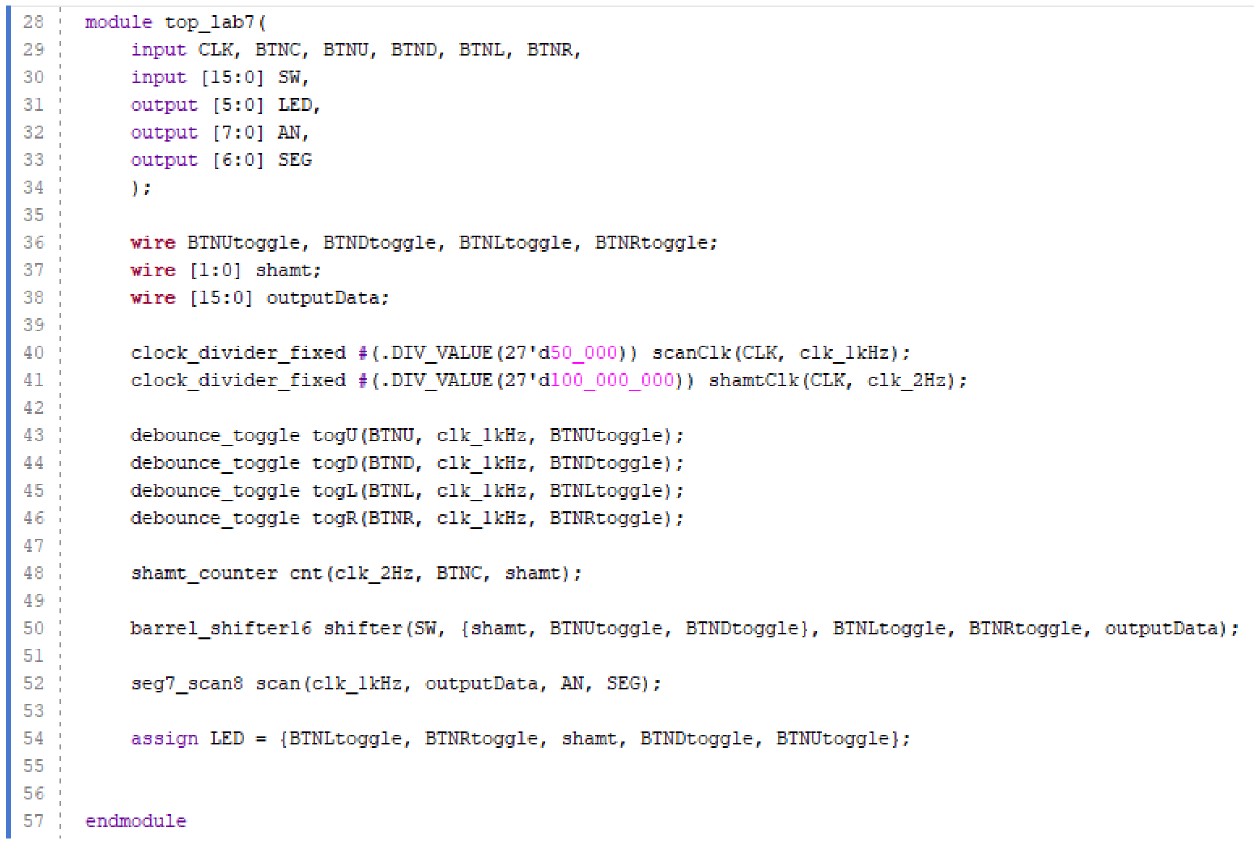
Hex to 7 Seg Decoder



```verilog
module hex_to_7seg(
    input [3:0] hex,
    output reg [6:0] Cnode
    );

    always @(hex) begin
        case (hex)
            4'd0: Cnode=7'b0000001; 4'd1: Cnode=7'b1001111; 4'd2: Cnode=7'b0010010;
            4'd3: Cnode=7'b0000110; 4'd4: Cnode=7'b1001100; 4'd5: Cnode=7'b0100100;
            4'd6: Cnode=7'b0100000; 4'd7: Cnode=7'b0001111; 4'd8: Cnode=7'b0000000;
            4'd9: Cnode=7'b0001100; 4'd10:Cnode=7'b0001000;4'd11:Cnode=7'b1100000;
            4'd12:Cnode=7'b0110001;4'd13:Cnode=7'b1000010;4'd14:Cnode=7'b0110000;
            4'd15:Cnode=7'b0111000;default: Cnode=7'b1111111;
        endcase
    end

endmodule
```

As mentioned before, the Hex to 7seg decoder is placed in its own module to improve readability in the code. It's the same as other implementations in previous labs. This is used in the seven seg driver module to print the digit on the used places on the seven seg display.

High Level Implementation

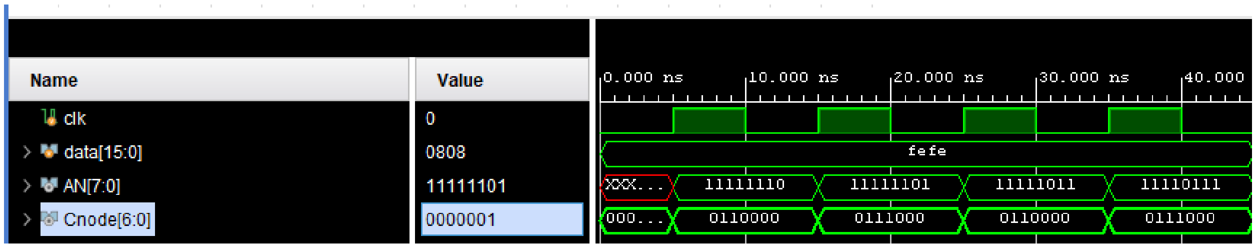


```verilog
module top_lab7(
    input CLK, BTNC, BTNU, BTND, BTNL, BTNR,
    input [15:0] SW,
    output [5:0] LED,
    output [7:0] AN,
    output [6:0] SEG
    );

    wire BTNUtoggle, BTNDtoggle, BTNLtoggle, BTNRtoggle;
    wire [1:0] shamt;
    wire [15:0] outputData;

    clock_divider_fixed #(.DIV_VALUE(27'd50_000)) scanClk(CLK, clk_1kHz);
    clock_divider_fixed #(.DIV_VALUE(27'd100_000_000)) shamtClk(CLK, clk_2Hz);

    debounce_toggle togU(BTNU, clk_1kHz, BTNUtoggle);
    debounce_toggle togD(BTND, clk_1kHz, BTNDtoggle);
    debounce_toggle togL(BTNL, clk_1kHz, BTNLtoggle);
    debounce_toggle togR(BTNR, clk_1kHz, BTNRtoggle);

    shamt_counter cnt(clk_2Hz, BTNC, shamt);

    barrel_shifter16 shifter(SW, {shamt, BTNUtoggle, BTNDtoggle}, BTNLtoggle, BTNRtoggle, outputData);

    seg7_scan8 scan(clk_1kHz, outputData, AN, SEG);

    assign LED = {BTNLtoggle, BTNRtoggle, shamt, BTNDtoggle, BTNUtoggle};


endmodule
```

This is the high level implementation, where the 1kHz and 2Hz clocks are made and the buttons' debouncing issues are handled while giving them toggle states. Additionally, the output from the 16-bit barrel shifter goes into the seg 7 scan module to be displayed on the board. Finally, debug LED's are implemented to make sure we know what buttons are toggled, as well as the current state of shamt.

Simulation Waveform

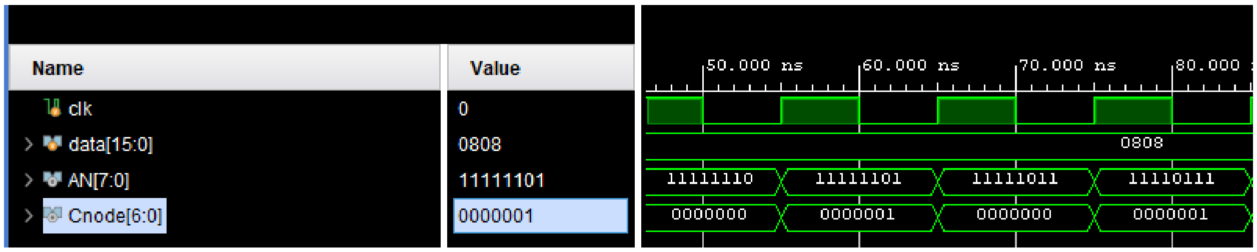
Seg7_Scan8



Name	Value
clk	0
data[15:0]	0808
AN[7:0]	11111101
Cnode[6:0]	0000001

When data = fefe:

1. When AN = 0, Cnode represents E
2. When AN = 1, Cnode represents F
3. When AN = 2, Cnode represents E
4. When AN = 3, Cnode represents F

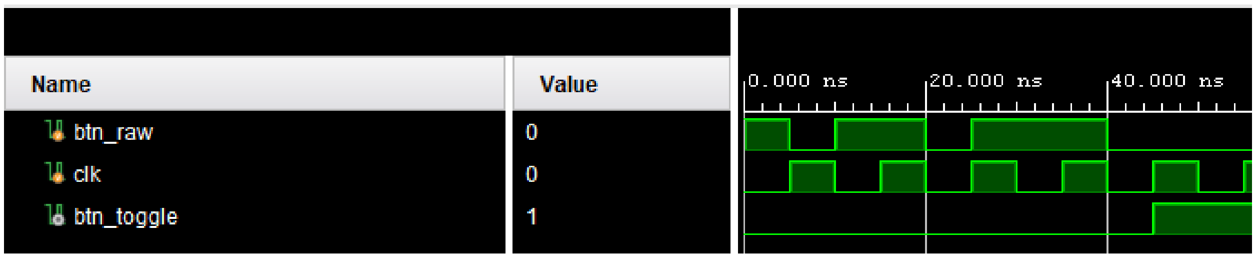


Name	Value
clk	0
data[15:0]	0808
AN[7:0]	11111101
Cnode[6:0]	0000001

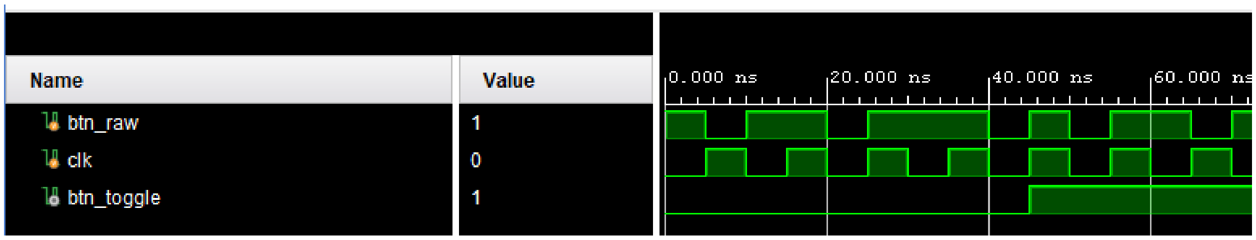
When data = 0808:

1. When AN = 0, Cnode represents 8
2. When AN = 1, Cnode represents 0
3. When AN = 2, Cnode represents 8
4. When AN = 3, Cnode represents 0

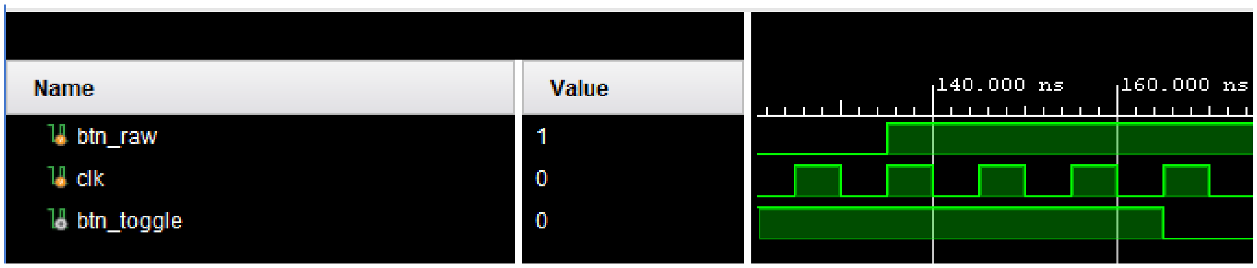
Debounce Toggle Button



When the button is held for one or two positive clock edge triggers, the btn_toggle variable stays low. It's only when btn_raw is high for 3 clock positive edge triggers that the btn_toggle variable goes high.

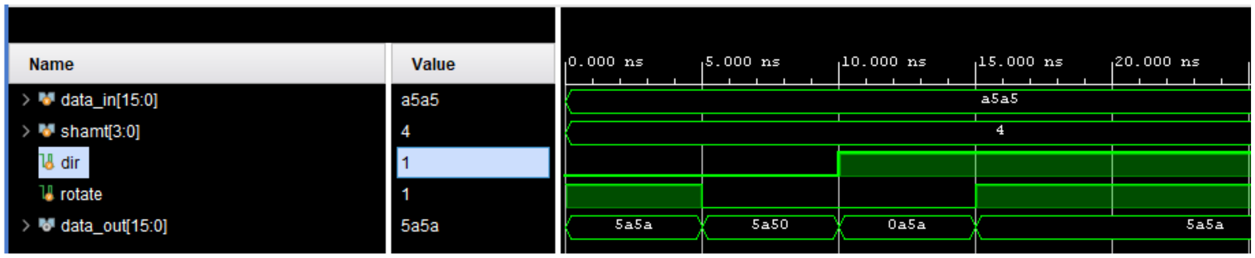


When btn_toggle is high, if we don't let btn_raw stay low for at least 3 clock positive edge triggers, btn_toggle will remain low



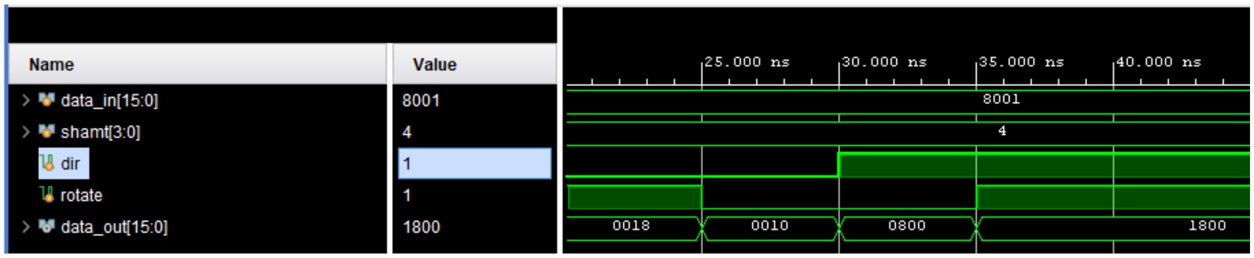
When btn_raw is left low long enough, it can be put high for long enough again to finally put btn_toggle on low

16-bit Barrel Shift



Name	Value
data_in[15:0]	a5a5
shamt[3:0]	4
dir	1
rotate	1
data_out[15:0]	5a5a

Using the value "a5a5", we use 4 for shamt to easily see the hex digits being shifted. When direction is 0 (left) and rotate is 1, every digit shifts left and wraps. When the shift is logical instead, the leftmost "a" is discarded and the rightmost digit is replaced with a 0. Now moving to the right, when rotate is 0, the rightmost "5" is discarded, everything shifts right, and the leftmost digit is replaced with 0. Finally, when rotate = 1, everything shifts right and wraps when overflow occurs.



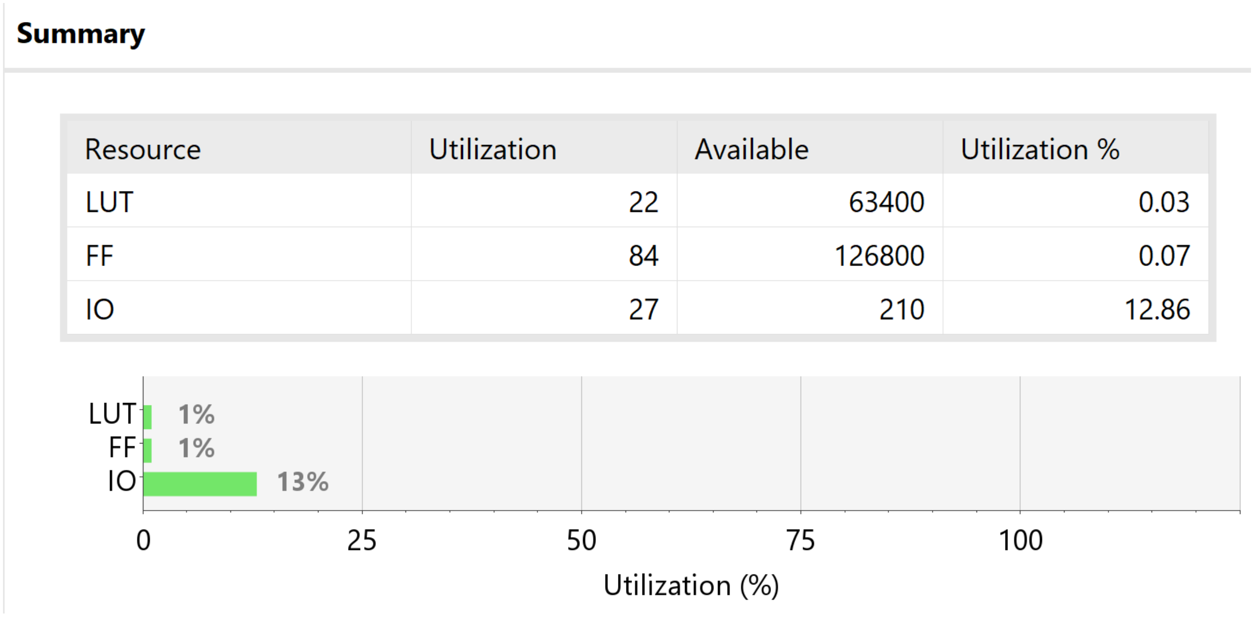
Name	Value
data_in[15:0]	8001
shamt[3:0]	4
dir	1
rotate	1
data_out[15:0]	1800

Using the value "8001", we use 4 for shamt to easily see the hex digits being shifted. When direction is 0 (left) and rotate is 1, every digit shifts left and wraps.

When the shift is logical instead, the leftmost “8” is discarded and the rightmost digit is replaced with a 0. Now moving to the right, when rotate is 0, the rightmost “1” is discarded, everything shifts right, and the leftmost digit is replaced with 0. Finally, when rotate = 1, everything shifts right and wraps when overflow occurs.

Implementation

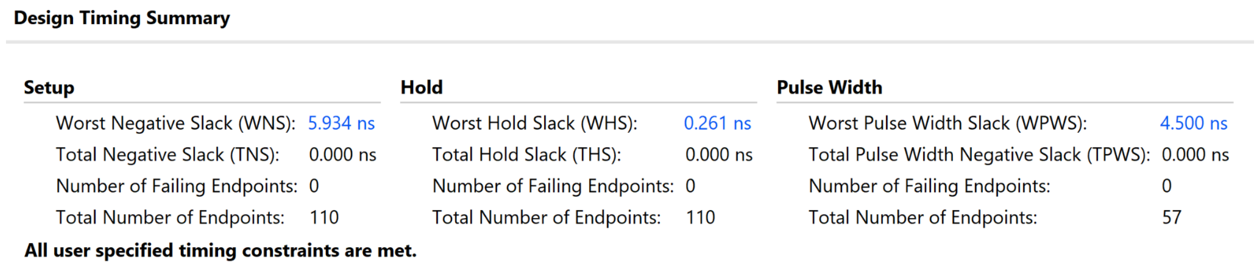
Utilization Table:



Summary

Resource	Utilization	Available	Utilization %
LUT	22	63400	0.03
FF	84	126800	0.07
IO	27	210	12.86

Timing Summary:



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.934 ns	Worst Hold Slack (WHS):	0.261 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:	110	Total Number of Endpoints:	110	Total Number of Endpoints:	57

All user specified timing constraints are met.

Contribution

Michelle Lau (50%) - Debugging, implementation and board demo
Edwin Estrada (50%) - Programming, debugging, and test benching