ECE3300L.E01

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

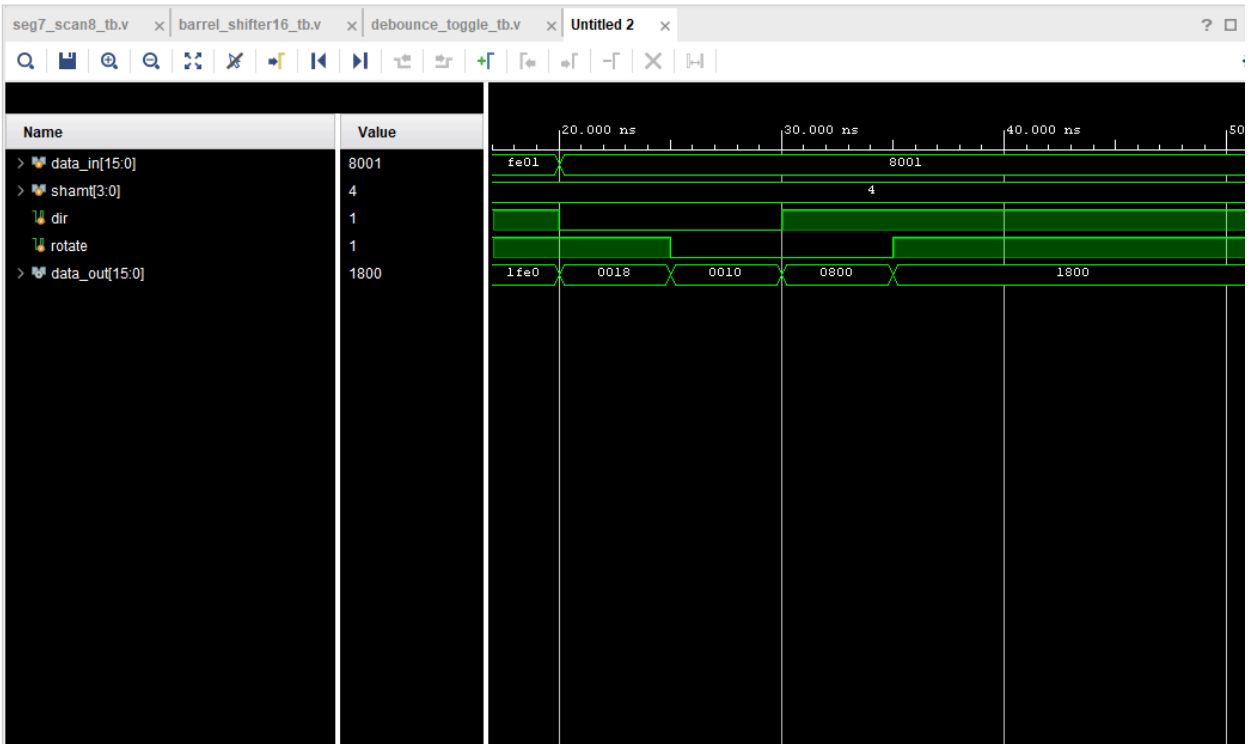
Priyanka Ravinder and Raj Gokidi

08/4/2025

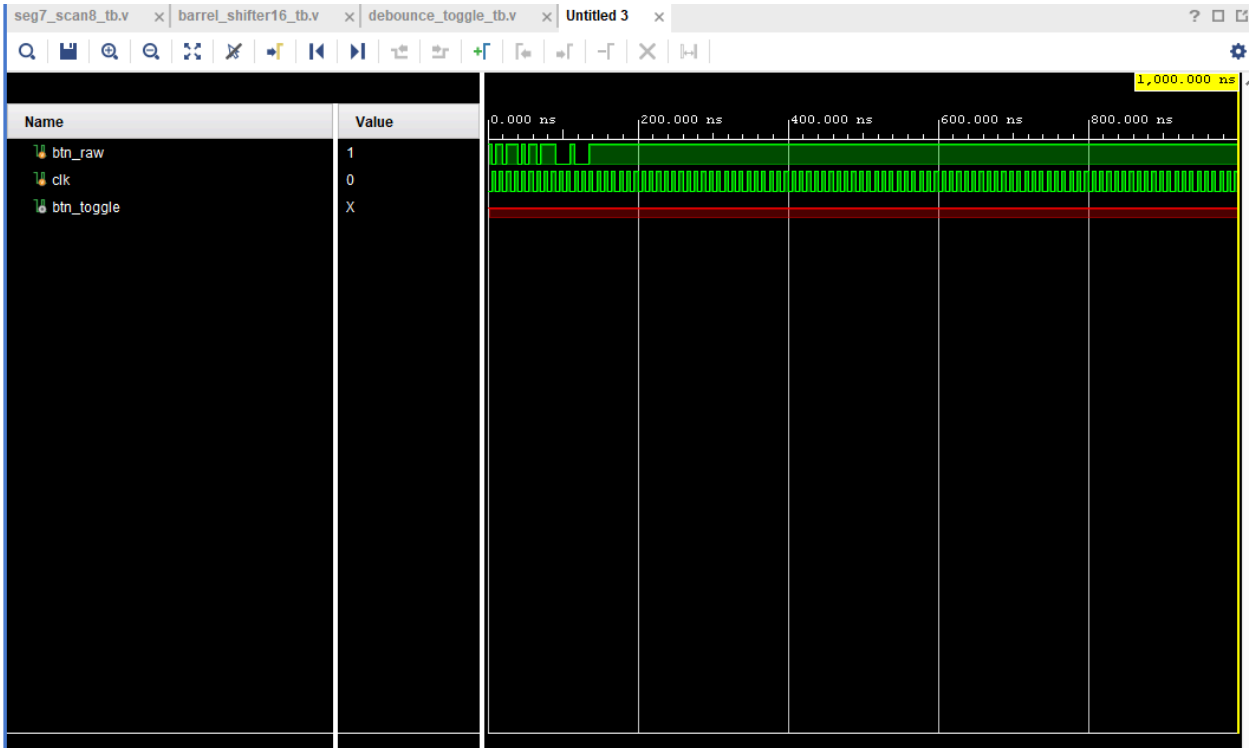
Introduction:

The objective of this lab was to create a 16-bit barrel shifter and rotator, using Verilog code and the Nexys A7 FPGA board. This barrel shifter should be able to rotate a 16-bit integer left or right by 0-15 in a single clock cycle. The switches and buttons on the board are used to control the shift direction, rotating mode and the amount the integer is being shifted by. Once the desired result is obtained, it is then displayed on four of the 7-segment displays in hexadecimal. This lab ultimately helped us practice 2-to-1 multiplexers, debouncing push buttons, displaying results on the 7-segment display and designing modules using Verilog.

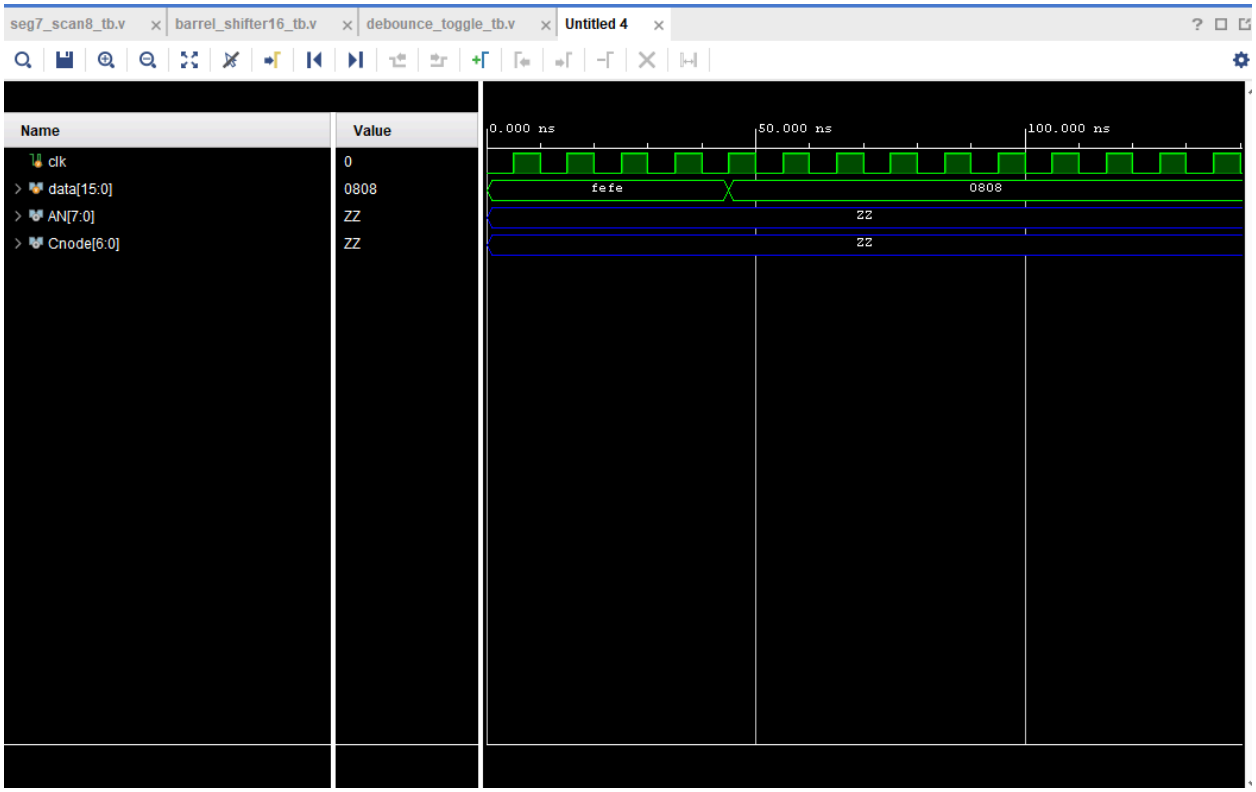
Waveforms:



Barrel_shifter_16_tb



Debounce_toggle_tb



seg7_scan8_tb

Code:

barrel_shifter16.v

```
`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[4:0];
assign stage[0] = data_in;

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

      if (dir == 1'b0) begin
         assign shifted = rotate ?
            {in[15 - (1<<i):0], in[15:16 - (1<<i)]} :
            {in[15 - (1<<i):0], {(1<<i){1'b0}}};
      end else begin
         assign shifted = rotate ?
            {in[(1<<i)-1:0], in[15:(1<<i)]} :
            {{(1<<i){1'b0}}, in[15:(1<<i)]};
      end

      assign stage[i+1] = shamt[i] ? shifted : in;
   end
endgenerate
```

```
    assign data_out = stage[4];
endmodule
```

clock_divider_fixed.v

```
`timescale 1ns / 1ps
module clock_divider_fixed(
    input wire clk,
    output reg clk_1kHz = 0,
    output reg clk_demo = 0
);

    parameter DIV_VALUE = 26'd50_000_000;
    reg [25:0] count_1Hz = 0;
    reg clk_1Hz = 0;

    reg [9:0] count_1kHz = 0;
    reg [7:0] count_demo = 0;

    always @(posedge clk) begin
        if (count_1Hz == DIV_VALUE - 1) begin
            count_1Hz <= 0;
            clk_1Hz <= ~clk_1Hz;
        end else begin
```

```
            count_1Hz <= count_1Hz + 1;
        end
    end

    always @(posedge clk_1Hz) begin
        if (count_1kHz == 500 - 1) begin
            count_1kHz <= 0;
            clk_1kHz <= ~clk_1kHz;
        end else begin
            count_1kHz <= count_1kHz + 1;
        end
        if (count_demo == 2 - 1) begin
            count_demo <= 0;
            clk_demo <= ~clk_demo;
        end else begin
            count_demo <= count_demo + 1;
        end
    end

endmodule
```

debounce_toggle.v

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

```
module debounce_toggle(
    input wire clk_1kHz,
    input wire btn_raw,
    output reg btn_toggle = 0
);

    reg [2:0] shift_reg = 3'b000;
    reg debounced = 0;
    reg prev_debounced = 0;

    always @(posedge clk_1kHz) begin
        shift_reg <= {shift_reg[1:0], btn_raw};
        if (shift_reg == 3'b111)
            debounced <= 1;
        else if (shift_reg == 3'b000)
            debounced <= 0;
        if (~prev_debounced && debounced)
            btn_toggle <= ~btn_toggle;

        prev_debounced <= debounced;
    end
endmodule
```

hex_to_7seg.v

```
`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'b1000000; // 0
            4'h1: seg = 7'b1111001; // 1
            4'h2: seg = 7'b0100100; // 2
            4'h3: seg = 7'b0110000; // 3
            4'h4: seg = 7'b0011001; // 4
            4'h5: seg = 7'b0010010; // 5
            4'h6: seg = 7'b0000010; // 6
            4'h7: seg = 7'b1111000; // 7
            4'h8: seg = 7'b0000000; // 8
            4'h9: seg = 7'b0011000; // 9
            4'hA: seg = 7'b0001000; // A
            4'hB: seg = 7'b0000011; // b
            4'hC: seg = 7'b1000110; // C
            4'hD: seg = 7'b0100001; // d
```

```
            4'hE: seg = 7'b0000110; // E
            4'hF: seg = 7'b0001110; // F
            default: seg = 7'b1111111;
        endcase
    end
endmodule
```

seg7_scan8.v

```
`timescale 1ns / 1ps
module seg7_scan8(
    input wire clk,
    input wire [6:0] hex_seg [7:0],
    output reg [7:0] an = 8'b11111111,
    output reg [6:0] seg = 7'b1111111
);
    reg [2:0] scan_index = 0;

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

        an <= 8'b11111111;
        an[scan_index] <= 1'b0;
        seg <= hex_seg[scan_index];
```

```
    end
endmodule
```

shamt_counter.v

```
`timescale 1ns / 1ps
module shamt_counter(
    input wire clk,
    output reg [1:0] shamt_high
);

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

endmodule
```

top_lab7.v

```
`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 clk_1kHz, clk_demo;
    clock_divider_fixed clkdiv (
        .clk(CLK),
        .clk_1kHz(clk_1kHz),
        .clk_demo(clk_demo)
    );
    wire dir, rot, sham0, sham1;
    debounce_toggle db_dir (.clk_1kHz(clk_1kHz), .btn_raw(BTN[0]), .btn_toggle(dir));   // btnu
    debounce_toggle db_rot (.clk_1kHz(clk_1kHz), .btn_raw(BTN[1]), .btn_toggle(rot));   // btnd
    debounce_toggle db_s0  (.clk_1kHz(clk_1kHz), .btn_raw(BTN[2]), .btn_toggle(sham0)); //
btnl
    debounce_toggle db_s1  (.clk_1kHz(clk_1kHz), .btn_raw(BTN[3]), .btn_toggle(sham1)); //
btnr

    wire [1:0] shamt_high;
    shamt_counter shamtctr (
        .clk(BTN[4]),
        .shamt_high(shamt_high)
    );
```

```
wire [3:0] shamt = {shamt_high, sham1, sham0};

wire [15:0] result;
barrel_shifter16 shifter (
    .data_in(SW),
    .shamt(shamt),
    .dir(dir),
    .rotate(rot),
    .data_out(result)
);

assign LED[7:4] = shamt;
assign LED[3] = rot;
assign LED[2] = dir;
assign LED[1:0] = 2'b00;

wire [6:0] digits[7:0];

hex_to_7seg h0 (.hex(result[3:0]),   .seg(digits[0]));
hex_to_7seg h1 (.hex(result[7:4]),   .seg(digits[1]));
hex_to_7seg h2 (.hex(result[11:8]),  .seg(digits[2]));
hex_to_7seg h3 (.hex(result[15:12]), .seg(digits[3]));
```

```
    assign digits[4] = 7'b1111111;
    assign digits[5] = 7'b1111111;
    assign digits[6] = 7'b1111111;
    assign digits[7] = 7'b1111111;

    seg7_scan8 scanner (
        .clk(clk_1kHz),
        .hex_seg(digits),
        .an(AN),
        .seg(SEG)
    );

endmodule
```

Simulations:

barrel_shifter16_tb.v

```
`timescale 1ns / 1ps
module barrel_shifter16_tb;

    reg [15:0] data_in;
    reg [3:0] shamt;
    reg dir;
    reg rotate;
    wire [15:0] data_out;
```

```
barrel_shifter16 uut (
    .data_in(data_in),
    .shamt(shamt),
    .dir(dir),
    .rotate(rotate),
    .data_out(data_out)
);

initial begin
    data_in = 16'hA5A5;

    for (dir = 0; dir <= 1; dir = dir + 1) begin
        for (rotate = 0; rotate <= 1; rotate = rotate + 1) begin
            for (shamt = 0; shamt < 16; shamt = shamt + 1) begin
                #10;
            end
        end
    end

    data_in = 16'h8001;

    for (dir = 0; dir <= 1; dir = dir + 1) begin
```

```
            for (rotate = 0; rotate <= 1; rotate = rotate + 1) begin
                for (shamt = 0; shamt < 16; shamt = shamt + 1) begin
                    #10;
                end
            end
        end

        #20;
        $finish;
    end
endmodule
```

debounce_toggle_tb.v

```
`timescale 1ns / 1ps
module debounce_toggle_tb;

    reg clk = 0;
    reg btn_raw = 0;
    wire btn_toggle;

    debounce_toggle uut (
        .clk_1kHz(clk),
        .btn_raw(btn_raw),
```

```
    .btn_toggle(btn_toggle)
);

always #500 clk = ~clk;

initial begin
    #2000;

    btn_raw = 1; #100;
    btn_raw = 0; #100;
    btn_raw = 1; #200;
    btn_raw = 0; #100;
    btn_raw = 1; #1000;

    #5000;

    btn_raw = 0; #100;
    btn_raw = 1; #100;
    btn_raw = 0; #200;
    btn_raw = 1; #100;
    btn_raw = 0; #1000;
```

```
        btn_raw = 1; #3000;
        btn_raw = 0; #3000;

        #1000;
        $finish;
    end
endmodule
```

seg7_scan8_tb.v

```
`timescale 1ns / 1ps
module seg7_scan8_tb;

    reg clk = 0;
    wire [7:0] an;
    wire [6:0] seg;

    reg [6:0] hex_seg [7:0];

    seg7_scan8 uut (
        .clk(clk),
        .hex_seg(hex_seg),
        .an(an),
        .seg(seg)
```

```
    );

    always #500 clk = ~clk;

    initial begin
        hex_seg[0] = 7'b1000000;
        hex_seg[1] = 7'b1111001;
        hex_seg[2] = 7'b0100100;
        hex_seg[3] = 7'b0110000;
        hex_seg[4] = 7'b0011001;
        hex_seg[5] = 7'b0010010;
        hex_seg[6] = 7'b0000010;
        hex_seg[7] = 7'b1111000;

        #10000;
        $finish;
    end
endmodule
```

Video Link: https://youtu.be/p0ApLwK8DFI

Contributions:

Priyanka Ravinder: Code and Report

Raj Gokidi: Code and Report

Conclusion:

Overall, this lab was essential in aiding our understanding of barrel shifters and rotators using Verilog. Because we used the push-buttons in order to control how the input was moved, we were able to display the desired results on all four of the 7-segment displays, and see the changes quickly. This lab also helped us understand how barrel shifting is useful in the industry, and the waveforms were able to help us visualize this importance as well.