ECE 3300L.01 - Lab 7

Barrel Shifter & 7-Segment Display

Professor Mohamed Aly
Group Q: Kevin Tang(015429622) &
Jared Mocling(015215057)

August 6th, 2025

Design:

Barrel_shifter16.v

```
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] s_stage1, s_stage2, s_stage3, s_stage4;

    // shift/rotate by 1
    assign s_stage1 = (shamt[0]) ?
        (rotate ? (dir ? {data_in[0], data_in[15:1]} : {data_in[14:0], data_in[15]}) :
                  (dir ? {1'b0, data_in[15:1]}       : {data_in[14:0], 1'b0})) :
        data_in;

    // shift/rotate by 2
    assign s_stage2 = (shamt[1]) ?
        (rotate ? (dir ? {s_stage1[1:0], s_stage1[15:2]} : {s_stage1[13:0], s_stage1[15:14]})
:
                  (dir ? {2'b00, s_stage1[15:2]}         : {s_stage1[13:0], 2'b00})) :
        s_stage1;

    // shift/rotate by 4
    assign s_stage3 = (shamt[2]) ?
        (rotate ? (dir ? {s_stage2[3:0], s_stage2[15:4]} : {s_stage2[11:0], s_stage2[15:12]})
:
                  (dir ? {4'b0000, s_stage2[15:4]}       : {s_stage2[11:0], 4'b0000})) :
        s_stage2;

    // shift/rotate by 8
    assign s_stage4 = (shamt[3]) ?
        (rotate ? (dir ? {s_stage3[7:0], s_stage3[15:8]} : {s_stage3[7:0], s_stage3[15:8]}) :
                  (dir ? {8'b00000000, s_stage3[15:8]}   : {s_stage3[7:0], 8'b00000000})) :
        s_stage3;

    assign data_out = s_stage4;                                   // Final
shifted/rotated result
endmodule
```

Clock_divider_fixed.v

```
module clock_divider_fixed(
                    input clk,
                    output reg clk_1khz,
                    output reg clk_2khz
                    );
                    parameter DIV_2HZ = 26'd25_000_000;    // for 2Hz (toggle)
                    parameter DIV_1KHZ = 17'd50_000;       // for 1kHz

                    reg [25:0] cnt_2hz = 0;
                    reg [16:0] cnt_1khz = 0;

                    always @(posedge clk) begin
                       if (cnt_2hz == DIV_2HZ-1) begin
                          cnt_2hz <= 0;
                          clk_2khz <= ~clk_2khz;
                       end else begin
                          cnt_2hz <= cnt_2hz + 1;
                       end
                    end

                    always @(posedge clk) begin
                       if (cnt_1khz == DIV_1KHZ-1) begin
                          cnt_1khz <= 0;
                          clk_1khz <= ~clk_1khz;
                       end else begin
                          cnt_1khz <= cnt_1khz + 1;
                       end
                    end
endmodule
```

Debounce_toggle.v

```
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 value = 1 if all 3 bits are high, 0 if all 3
bits are low
         debounced <= 1;
      else if (shift_reg == 3'b000)
         debounced <= 0;
      if (~prev_debounced && debounced)
         btn_toggle <= ~btn_toggle;

      prev_debounced <= debounced;         // Save previous debounced state
   end
Endmodule
```

Hex_to_7seg.v

```
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; // Blank;
```

```
        endcase
    end
endmodule
```

Shamt_count.v

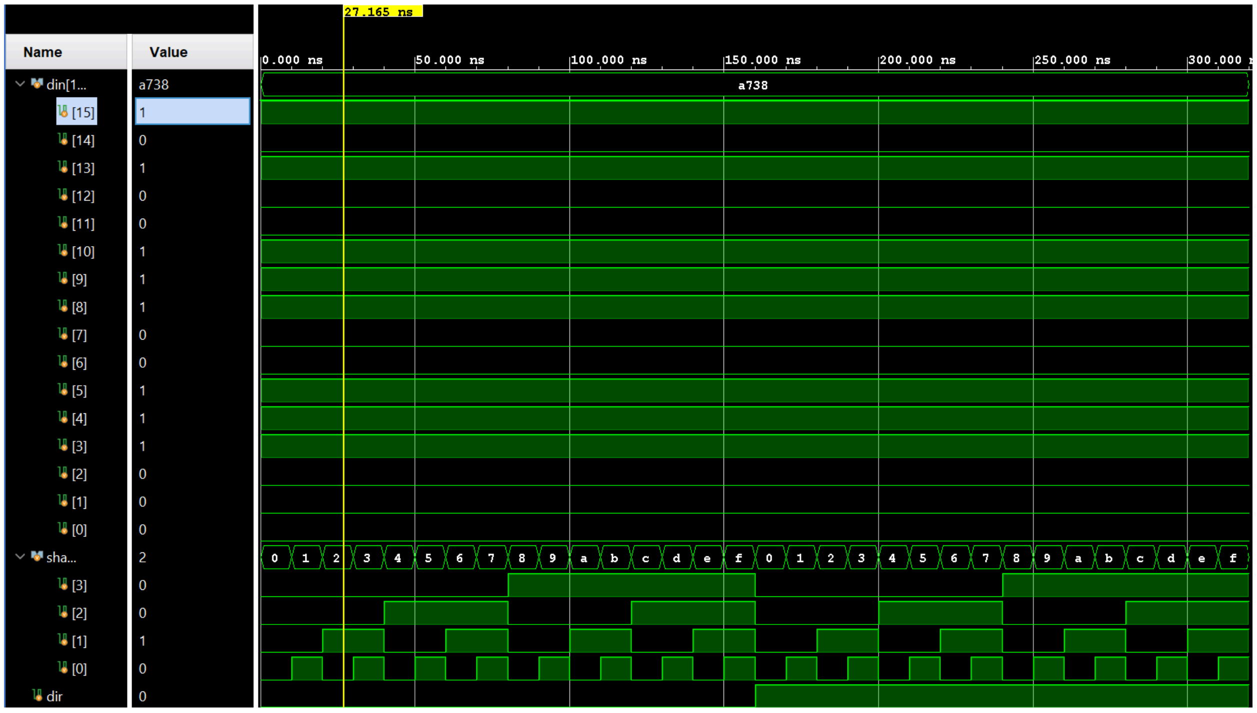
```
`timescale 1ns / 1ps

module shamt_counter(
                    input wire clk,
                    input BTNC,
                    output reg [1:0] shamt_hi
                   );

                   reg btnc_last = 0;
                   always @(posedge clk) begin
                       btnc_last <= BTNC;
                       if (BTNC & ~btnc_last)
                           shamt_hi <= shamt_hi + 1;
                   end
endmodule
```

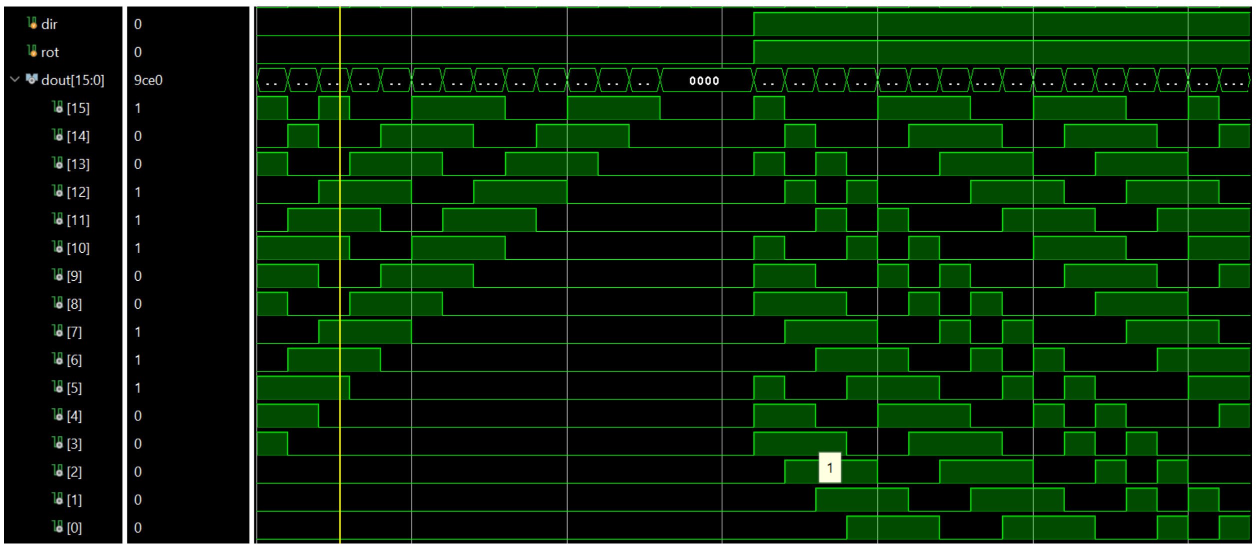
Top_lab7,v

```
`timescale 1ns / 1ps

module top_lab7(
    input wire clk,
    input wire rst,
    input wire [15:0] SW,
    input wire BTNC, BTNU, BTND, BTNL, BTNR,
    output wire [7:0] AN,
    output wire [6:0] SEG,
    output wire [7:0] LED
);
    // Clock dividers
    wire clk_2khz, clk_1khz;
    clock_divider_fixed clkdiv(.clk(clk), .clk_2khz(clk_2khz), .clk_1khz(clk_1khz));

    // Debounce buttons to toggles
    wire dir, rot, sh0, sh1;
    debounce_toggle d0(.clk_1khz(clk_1khz), .btn_raw(BTNU), .btn_toggle(dir));
    debounce_toggle d1(.clk_1khz(clk_1khz), .btn_raw(BTND), .btn_toggle(rot));
    debounce_toggle d2(.clk_1khz(clk_1khz), .btn_raw(BTNL), .btn_toggle(sh1));
    debounce_toggle d3(.clk_1khz(clk_1khz), .btn_raw(BTNR), .btn_toggle(sh0));

    // Shift amount counter high bits
    wire [1:0] shamt_hi;
    shamt_counter sc(.clk(clk), .BTNC(BTNC), .shamt_hi(shamt_hi));

    wire [3:0] shamt = {shamt_hi, sh1, sh0};

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

    // 7-seg display
    wire [6:0] seg0, seg1, seg2, seg3;
    hex_to_7seg h0(.HEX(shifted[3:0]), .SEG(seg0));
    hex_to_7seg h1(.HEX(shifted[7:4]), .SEG(seg1));
    hex_to_7seg h2(.HEX(shifted[11:8]), .SEG(seg2));
    hex_to_7seg h3(.HEX(shifted[15:12]), .SEG(seg3));

      wire [6:0] blank = 7'b1111111;
      seg7_scan8 scanner(
        .clk_1khz(clk_1khz),
        .seg0(seg0),
        .seg1(seg1),
        .seg2(seg2),
        .seg3(seg3),
        .seg4(blank),
        .seg5(blank),
        .seg6(blank),
        .seg7(blank),
        .AN(AN),
        .SEG(SEG)
                                );

    assign LED[3:0] = shamt;
    assign LED[4] = dir;
    assign LED[5] = rot;
    assign LED[7:6] = 2'b00;

endmodule
```

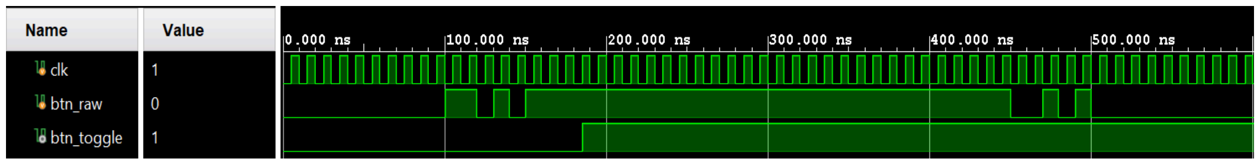
Testbench Waveform:

barrel_shifter16_tb

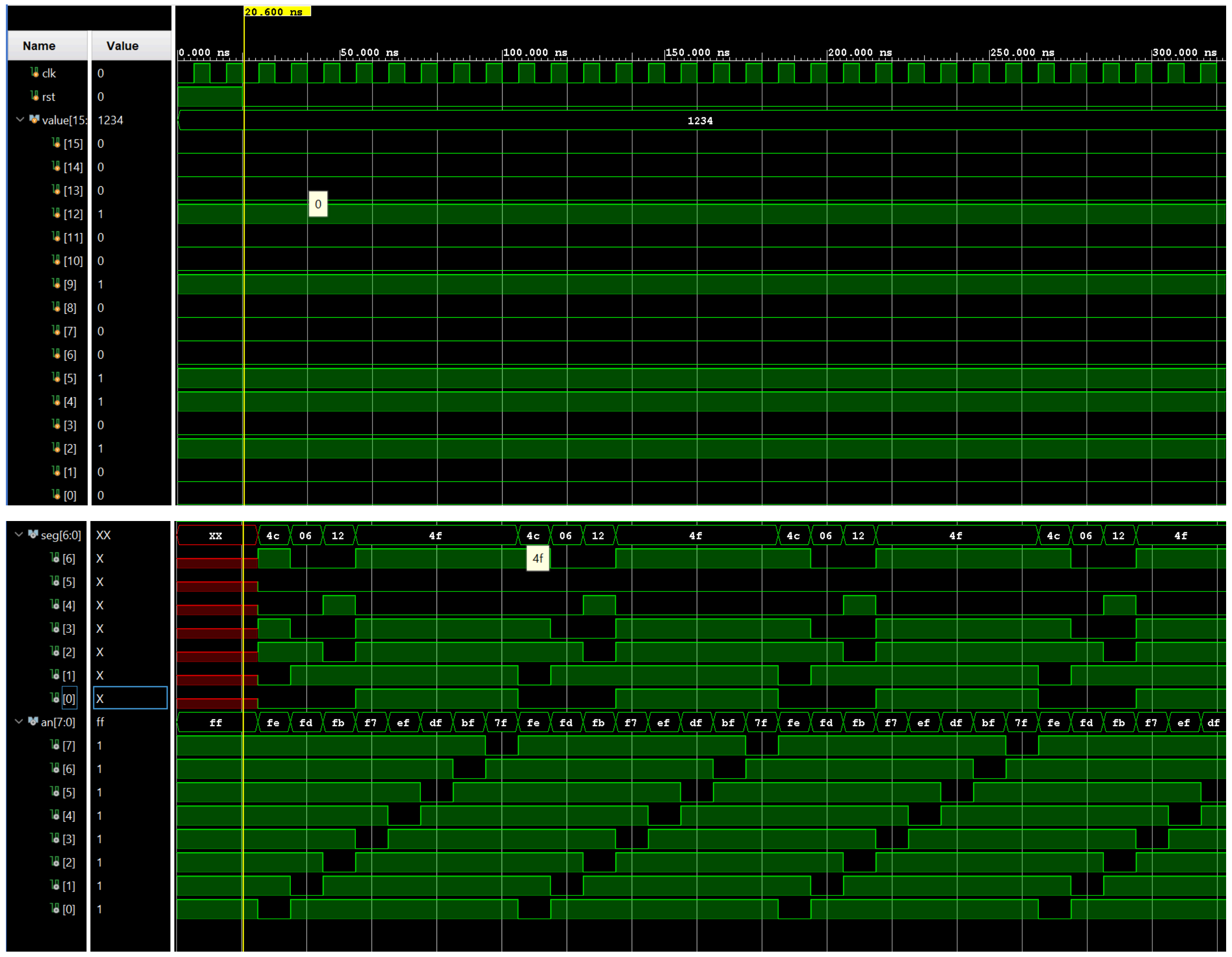




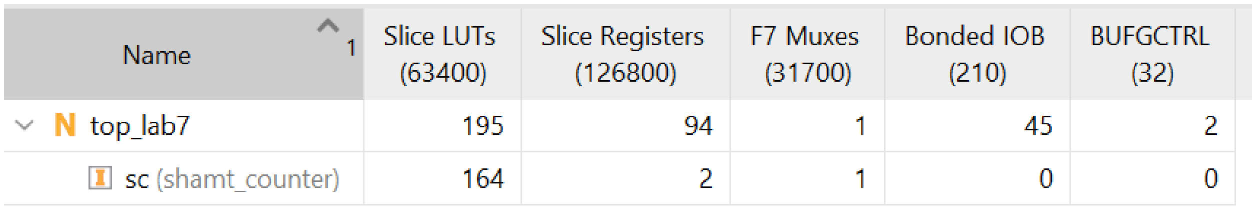
debounce_toggle_tb



seg7_scan8_tb

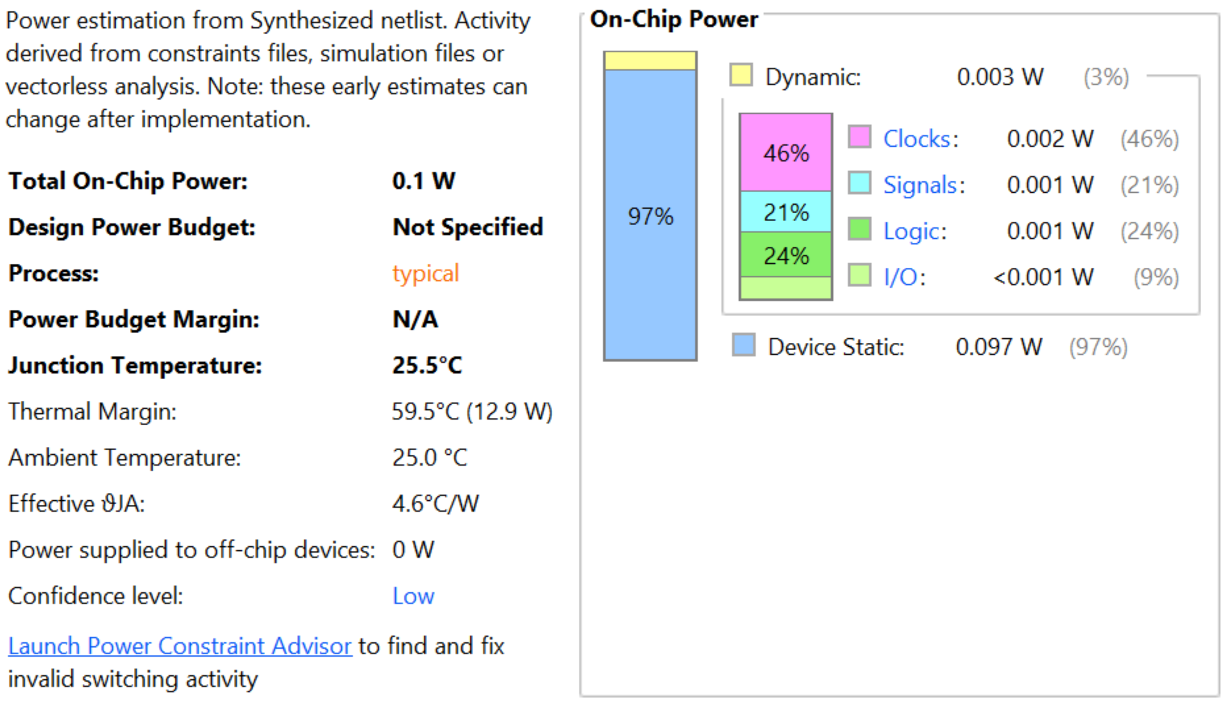


Utilization/Implementation:



Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Bonded IOB (210)	BUFGCTRL (32)
top_lab7	195	94	1	45	2
sc (shamt_counter)	164	2	1	0	0

Power



Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.1 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.5°C**
Thermal Margin:	59.5°C (12.9 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:		0.003 W	(3%)
	Clocks:	0.002 W	(46%)
	Signals:	0.001 W	(21%)
	Logic:	0.001 W	(24%)
	I/O:	<0.001 W	(9%)
Device Static:		0.097 W	(97%)

Contributions:

Jared Mocling (50%) - testbench code, compiled code, Synthesis reports, report
Kevin Tang (50%)- board demo, compiled code, report