CALIFORNIA STATE POLYTECHNIC

UNIVERSITY, POMONA

COLLEGE OF ENGINEERING

LAB 7

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

ECE 3300L Summer 2025

Digital Circuit Design using Verilog

Professor Mohamed Aly

By: Clay Kim and Changwe Musonda

Objective

The objective of this lab was to design, simulate, and implement a 16-bit barrel shifter and rotator on a Nexys A7-100T FPGA. The system is required to perform logical shifts and rotations in both left and right directions, with the shift amount determined by a combination of on-board controls. The 16-bit input data is provided by slide switches, and the 16-bit output result is displayed as four hexadecimal digits on the board's 7-segment display. This project integrates concepts of combinational logic design, input debouncing and latching, clock management, and multi-digit display scanning.

Hardware Pin Mapping

- CLK: 100MHz On-board master clock
- SW15–SW0: SW[15:0] 16-bit input word for the barrel shifter.
- BTNC: BTN0 Asynchronous reset for the system (active low). Also advances SHAMT[3:2].
- BTNU, BTND, BTNL, BTNR: BTN[4:1] Debounced control toggles for DIR, ROT, and SHAMT[1:0].
- AN3–AN0: AN[3:0] Digit enables for the four right-most 7-segment display digits.
- SEG6–SEG0: SEG[6:0] Cathode drivers for the seven segments (A-G).
- LED7–LED0: LED[7:0] Optional debug display for DIR, ROT, and SHAMT values.

Verilog Code

top_lab7

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


    // Clock divider (no reset needed)
    wire clk_2hz, clk_1khz;
    clock_divider_fixed clkdiv(
```

```
        .CLK(CLK),
        .CLK_2HZ(clk_2hz),
        .CLK_1KHZ(clk_1khz)
    );

    // Debounced toggles for all 4 control buttons (no reset)
    wire dir_toggle, rot_toggle, shamt0_toggle, shamt1_toggle;
    debounce_toggle deb_dir(.CLK_1KHZ(clk_1khz), .BTN_RAW(BTNU),
.BTN_TOGGLE(dir_toggle));
    debounce_toggle deb_rot(.CLK_1KHZ(clk_1khz), .BTN_RAW(BTND),
.BTN_TOGGLE(rot_toggle));
    debounce_toggle deb_shamt0(.CLK_1KHZ(clk_1khz), .BTN_RAW(BTNL),
.BTN_TOGGLE(shamt0_toggle));
    debounce_toggle deb_shamt1(.CLK_1KHZ(clk_1khz), .BTN_RAW(BTNR),
.BTN_TOGGLE(shamt1_toggle));

    // SHAMT[3:2] via BTNC press (no reset)
    wire [1:0] shamt_high;
    shamt_counter cnt(
        .CLK(clk_1khz),
        .BTNC(BTNC),
        .SHAMT_HIGH(shamt_high)
    );

    // Build SHAMT[3:0]
    wire [3:0] shamt = {shamt_high, shamt1_toggle, shamt0_toggle};

    // Barrel shifter
    wire [15:0] barrel_out;
    barrel_shifter16 shifter(
```

```
        .DATA_IN(SW),
        .SHAMT(shamt),
        .DIR(dir_toggle),
        .ROTATE(rot_toggle),
        .DATA_OUT(barrel_out)
    );

    // HEX to 7-seg (only rightmost 4 digits show output, others blank)
    wire [6:0] seg0, seg1, seg2, seg3;
    hex_to_7seg h0(.HEX(barrel_out[3:0]), .SEG(seg0));
    hex_to_7seg h1(.HEX(barrel_out[7:4]), .SEG(seg1));
    hex_to_7seg h2(.HEX(barrel_out[11:8]), .SEG(seg2));
    hex_to_7seg h3(.HEX(barrel_out[15:12]), .SEG(seg3));
    wire [6:0] blank = 7'b1111111;

    // 8-digit display scanner
    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)
    );

    // Debug LEDs (show DIR, ROT, SHAMT)
    assign LED[7:0] = {2'b00, shamt, rot_toggle, dir_toggle};

endmodule
```

barrel_shifter16

```
module barrel_shifter16(
    input [15:0] DATA_IN,
    input [3:0] SHAMT,
    input DIR,          // 0 = left, 1 = right
    input ROTATE,       // 0 = logical, 1 = rotate
    output [15:0] DATA_OUT
);
    wire [15:0] stage1, stage2, stage3, stage4;

    assign stage1 = SHAMT[0] ?
        (DIR ?
            (ROTATE ? {DATA_IN[0], DATA_IN[15:1]} : {1'b0, DATA_IN[15:1]}) :
            (ROTATE ? {DATA_IN[14:0], DATA_IN[15]} : {DATA_IN[14:0], 1'b0})
        )
        : DATA_IN;

    assign stage2 = SHAMT[1] ?
        (DIR ?
            (ROTATE ? {stage1[1:0], stage1[15:2]} : {2'b00, stage1[15:2]}) :
            (ROTATE ? {stage1[13:0], stage1[15:14]} : {stage1[13:0], 2'b00})
        )
        : stage1;

    assign stage3 = SHAMT[2] ?
        (DIR ?
            (ROTATE ? {stage2[3:0], stage2[15:4]} : {4'b0000, stage2[15:4]}) :
            (ROTATE ? {stage2[11:0], stage2[15:12]} : {stage2[11:0], 4'b0000})
        )
```

```
        : stage2;

    assign stage4 = SHAMT[3] ?
        (DIR ?
            (ROTATE ? {stage3[7:0], stage3[15:8]} : {8'b00000000, stage3[15:8]}) :
            (ROTATE ? {stage3[7:0], stage3[15:8]} : {stage3[7:0], 8'b00000000})
        )
        : stage3;

    assign DATA_OUT = stage4;
endmodule
```

clock_divider_fixed

```
module clock_divider_fixed(
    input CLK,
    output reg CLK_2HZ,
    output reg CLK_1KHZ
);
    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_2HZ <= ~CLK_2HZ;
        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

```
module debounce_toggle(
    input CLK_1KHZ,
    input BTN_RAW,
    output reg BTN_TOGGLE
);
    reg [2:0] shift_reg = 0;
    reg last_state = 0;

    always @(posedge CLK_1KHZ) begin
        shift_reg <= {shift_reg[1:0], BTN_RAW};
```

```
        if (shift_reg == 3'b111 && !last_state) begin
            BTN_TOGGLE <= ~BTN_TOGGLE;
            last_state <= 1;
        end else if (shift_reg == 3'b000) begin
            last_state <= 0;
        end
    end
endmodule
```

hex_to_7seg

```
module hex_to_7seg(
    input [3:0] HEX,
    output reg [6:0] SEG
);
    always @(*) begin
        case(HEX)
            4'h0: SEG = 7'b1000000;
            4'h1: SEG = 7'b1111001;
            4'h2: SEG = 7'b0100100;
            4'h3: SEG = 7'b0110000;
            4'h4: SEG = 7'b0011001;
            4'h5: SEG = 7'b0010010;
            4'h6: SEG = 7'b0000010;
            4'h7: SEG = 7'b1111000;
            4'h8: SEG = 7'b0000000;
            4'h9: SEG = 7'b0010000;
            4'hA: SEG = 7'b0001000;
```

```
            4'hB: SEG = 7'b0000011;
            4'hC: SEG = 7'b1000110;
            4'hD: SEG = 7'b0100001;
            4'hE: SEG = 7'b0000110;
            4'hF: SEG = 7'b0001110;
        endcase
    end
endmodule
```

seg7_scan8

```
module seg7_scan8(
    input CLK_1KHZ,
    input [6:0] SEG0, SEG1, SEG2, SEG3, SEG4, SEG5, SEG6, SEG7,
    output reg [7:0] AN,
    output reg [6:0] SEG
);
    reg [2:0] scan_cnt = 0;
    always @(posedge CLK_1KHZ) begin
        scan_cnt <= scan_cnt + 1;
    end
    always @(*) begin
        case(scan_cnt)
            3'd0: begin AN = 8'b11111110; SEG = SEG0; end // rightmost digit
            3'd1: begin AN = 8'b11111101; SEG = SEG1; end
            3'd2: begin AN = 8'b11111011; SEG = SEG2; end
            3'd3: begin AN = 8'b11110111; SEG = SEG3; end
            3'd4: begin AN = 8'b11101111; SEG = SEG4; end // blank
```

```
            3'd5: begin AN = 8'b11011111; SEG = SEG5; end // blank
            3'd6: begin AN = 8'b10111111; SEG = SEG6; end // blank
            3'd7: begin AN = 8'b01111111; SEG = SEG7; end // blank
        endcase
    end
endmodule
```

shamt_counter

```
module shamt_counter(
    input CLK,
    input BTNC,
    output reg [1:0] SHAMT_HIGH
);
    reg btnc_last = 0;
    always @(posedge CLK) begin
        btnc_last <= BTNC;
        if (BTNC & ~btnc_last)
            SHAMT_HIGH <= SHAMT_HIGH + 1;
    end
endmodule
```

Testbench Code

barrel_shifter16_tb

```
module barrel_shifter16_tb();

    reg [15:0] DATA_IN;
    reg [3:0] SHAMT;
    reg DIR;
    reg ROTATE;
    wire [15:0] DATA_OUT;

    integer i, j, k;

    barrel_shifter16 dut (
        .DATA_IN(DATA_IN),
        .SHAMT(SHAMT),
        .DIR(DIR),
        .ROTATE(ROTATE),
        .DATA_OUT(DATA_OUT)
    );

    initial begin

        DATA_IN <= 16'h1234;

        for (i = 0; i <= 1; i = i + 1) begin
            DIR <= i;
            for (j = 0; j <= 1; j = j + 1) begin
                ROTATE <= j;
                for (k = 0; k < 16; k = k + 1) begin
```

```
            SHAMT <= k;
            #10;
        end
    end
end

    #10;
    DATA_IN <= 16'hABCD;

    for (i = 0; i <= 1; i = i + 1) begin
        DIR <= i;
        for (j = 0; j <= 1; j = j + 1) begin
            ROTATE <= j;
            for (k = 0; k < 16; k = k + 1) begin
                SHAMT <= k;
                #10;
            end
        end
    end

    #20;
    $finish;
  end
endmodule
```

debounce_toggle_tb

```
module debounce_toggle_tb();

  reg  CLK_1KHZ = 0;
```

```
reg  BTN_RAW = 0;
wire btn_toggle;

always #5_000 CLK_1KHZ = ~CLK_1KHZ;

debounce_toggle dut (
    .CLK_1KHZ  (CLK_1KHZ),
    .BTN_RAW   (BTN_RAW),
    .BTN_TOGGLE(btn_toggle)
);

initial begin
    #20_000;

    BTN_RAW = 1; #500;
    BTN_RAW = 0; #400;
    BTN_RAW = 1; #800;
    BTN_RAW = 0; #600;
    BTN_RAW = 1;
    #100_000;

    BTN_RAW = 0; #500;
    BTN_RAW = 1; #400;
    BTN_RAW = 0; #800;
    BTN_RAW = 1; #600;
    BTN_RAW = 0;
    #100_000;

    BTN_RAW = 1; #200_000;
```

```
        BTN_RAW = 0; #200_000;

        BTN_RAW = 1; #200;
        BTN_RAW = 0; #100_000;

        BTN_RAW = 1; #200_000;
        BTN_RAW = 0; #200_000;

        #50_000;
        $finish;
    end
endmodule
```

seg7_scan8_tb

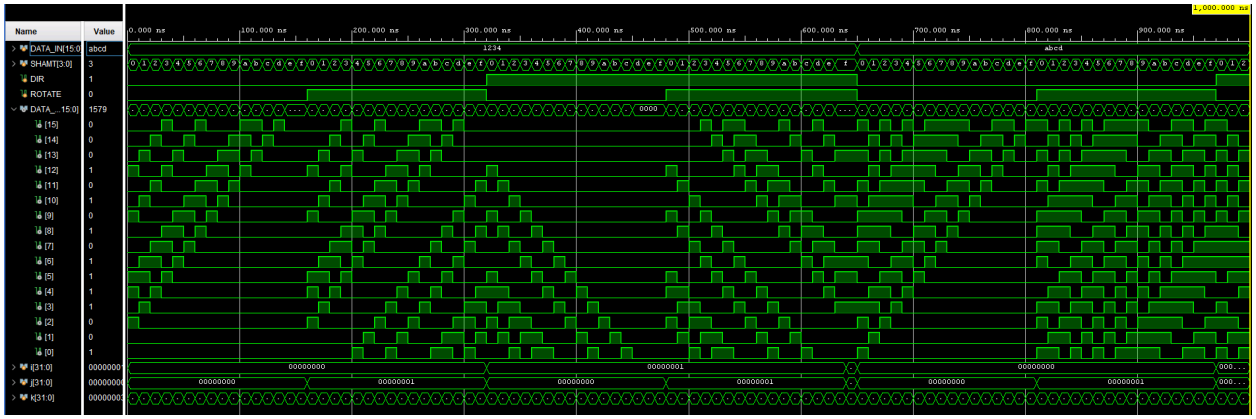
```
module seg7_scan8_tb();
    reg  clk_1kHz = 0;
    reg [6:0] SEG0, SEG1, SEG2, SEG3, SEG4, SEG5, SEG6, SEG7;
    wire [7:0] AN;
    wire [6:0] SEG;

    seg7_scan8 dut (
        .CLK_1KHZ(clk_1kHz),
        .SEG0(SEG0), .SEG1(SEG1), .SEG2(SEG2), .SEG3(SEG3),
        .SEG4(SEG4), .SEG5(SEG5), .SEG6(SEG6), .SEG7(SEG7),
        .AN(AN),
        .SEG(SEG)
    );
```

```
    always #500_000 clk_1kHz = ~clk_1kHz;



      initial begin
            SEG0 = 7'b1000000; // 0
            SEG1 = 7'b1111001; // 1
            SEG2 = 7'b0100100; // 2
            SEG3 = 7'b0110000; // 3
            SEG4 = 7'b0011001; // 4
            SEG5 = 7'b0010010; // 5
            SEG6 = 7'b0000010; // 6
            SEG7 = 7'b1111000; // 7

            #20000; // 20 ms
            $finish;
        end

endmodule
```

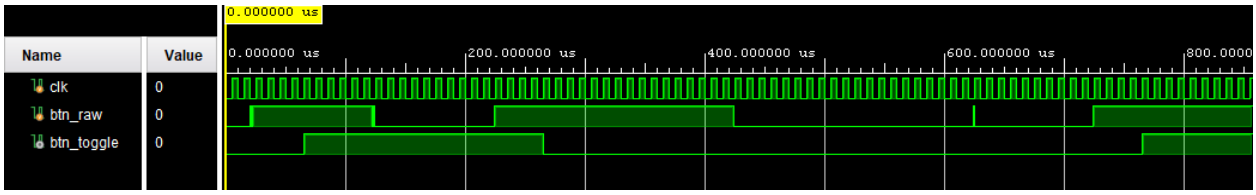
Synthesis and Implementation

Screenshots

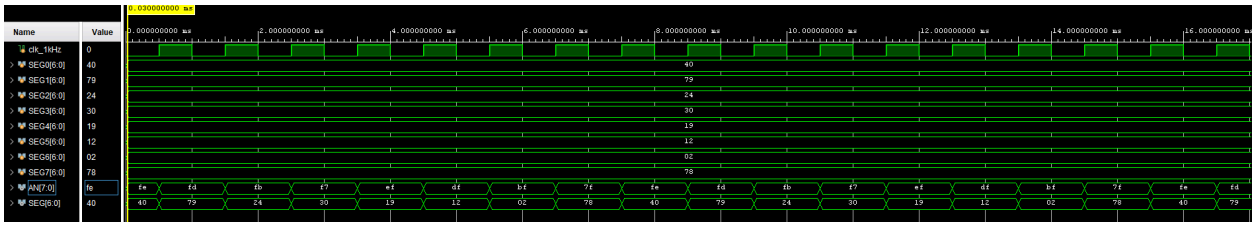
barrel_shifter16_tb



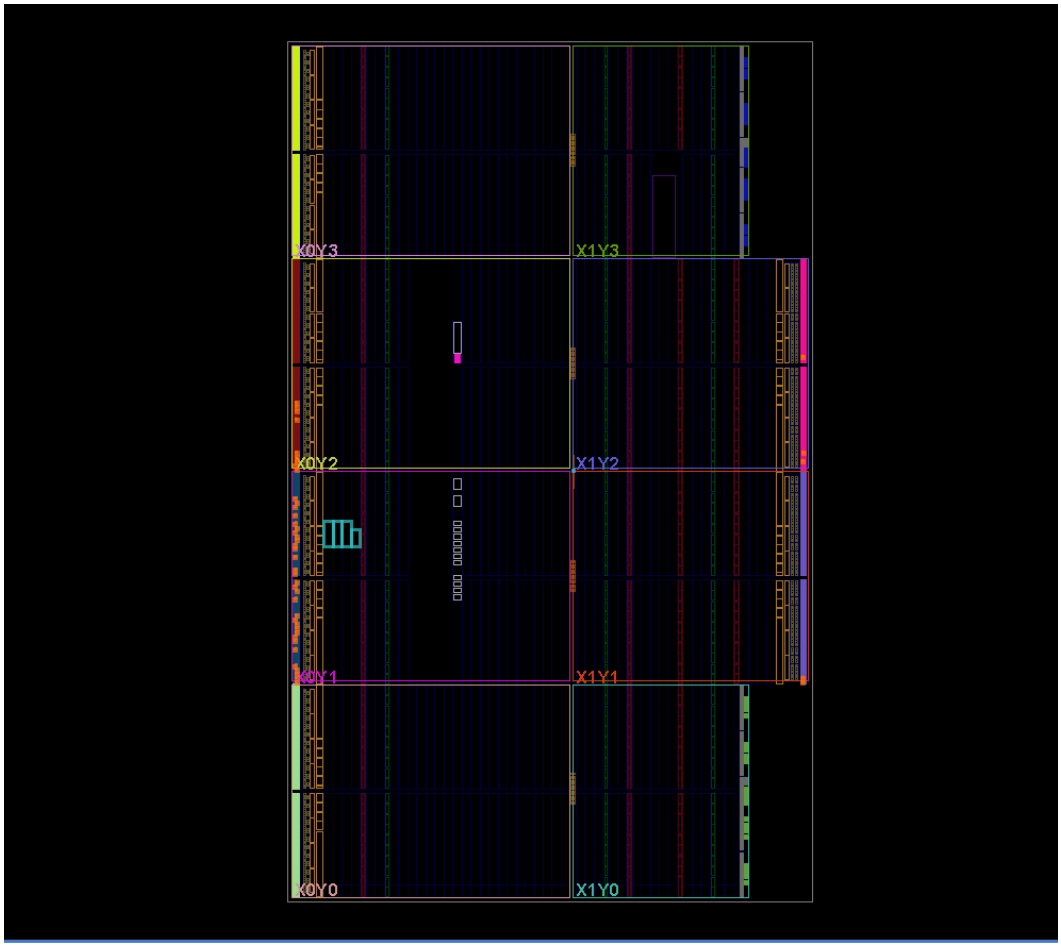
debounce_toggle_tb



seg7_scan8_tb



Vivado Utilization

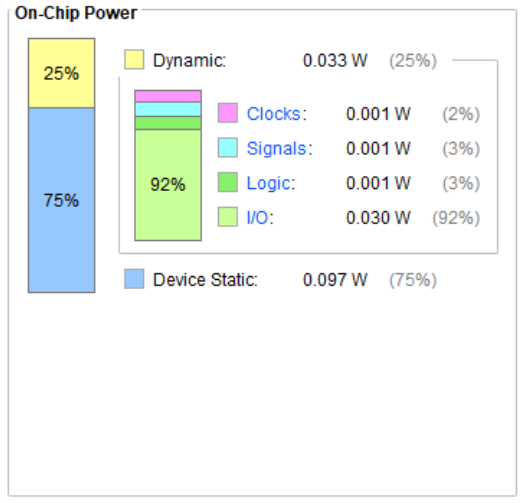


Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.13 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.6°C**
Thermal Margin:	59.4°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



DRC | Methodology | Power | Timing | Utilization

Hierarchy

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
top_lab7	103	44	39	103	45	1
clkdiv (clock_divider_fixed)	5	18	7	5	0	0
cnt (shamt_counter)	1	3	1	1	0	0
deb_dir (debounce_toggle)	1	5	2	1	0	0
deb_rot (debounce_toggle_0)	5	5	4	5	0	0
deb_shamt0 (debounce_toggle_1)	84	5	28	84	0	0
deb_shamt1 (debounce_toggle_2)	1	5	2	1	0	0
scanner (seg7_scan8)	6	3	4	6	0	0

Partner Contribution

Clay Kim: Testbench and Simulation

Changwe Musonda: Demonstration video and implementation

Reflections

This lab successfully culminated in the implementation of a 16-bit barrel shifter and rotator on a Nexys A7 FPGA. The core of the design was a purely combinational module that performed logical shifts and rotations on the 16-bit input from SW[15:0]. By structuring the logic into four logarithmic stages of 2-to-1 multiplexers, the system could permute the input word in a single clock cycle. This design effectively demonstrated how complex permutation operations can be built from simple, repeated hardware blocks, with DIR and ROTATE signals dynamically controlling the data path.

A robust user interface was created using debounced push-buttons with edge-to-toggle logic to generate stable control signals. A key feature was the use of BTNC to drive a 2-bit counter, allowing all 16 shift positions (SHAMT[3:0]) to be tested without consuming extra switches. The final 16-bit result was displayed in hexadecimal on four 7-segment digits, managed by a scanner and a dual-rate clock divider that provided both a slow ~2 Hz clock for visual demonstration and a fast ~1 kHz clock for a flicker-free display. Overall, the project was a comprehensive exercise in integrating core logic with practical I/O handling and system timing.