

Julio Flores (ID# : 016326856)
Victor Perez (ID# : 016196050)

Group I

Session E02

Lab 7

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

August 6, 2025

ECE 3300L

Summer 2025

Objective:

The objective of this lab is to design, implement, and test a 16-bit barrel shifter and rotator using VHDL (or Verilog) on the Nexys A7 FPGA development board. The circuit allows for left or right shifting and rotating operations, controlled by user inputs using the 4 buttons and central reset button. Additionally, the output of the barrel shifter is displayed on a 4-digit 7-segment display to visualize the data in real-time. This lab reinforces digital design principles including data manipulation, control signal usage, and output interfacing through hardware display components.

Design(Code):

Clock_Divider_Fixed.V

- Uses a clock to set the speed to a readable speed for 7seg displays and shifter

```

module clock_divider_fixed(
    input wire clk,
    output reg clk_1kHz = 0,
    output reg clk_demo = 0
);

    reg [15:0] count_1kHz = 0;
    reg [25:0] count_demo = 0;

    always @(posedge clk) begin
        // ~1 kHz from 100 MHz
        if (count_1kHz == 49999) begin
            count_1kHz <= 0;
            clk_1kHz <= ~clk_1kHz;
        end else begin
            count_1kHz <= count_1kHz + 1;
        end

        // ~2 Hz from 100 MHz
        if (count_demo == 24_999_999) begin
            count_demo <= 0;
            clk_demo <= ~clk_demo;
        end else begin
            count_demo <= count_demo + 1;
        end
    end
endmodule

```

Debounce_toggle.v

- Prevents glitches from affecting the buttons especially during toggling left/right shifts and SHAMT functions.

```

| module debounce_toggle(
|     input clk_1kHz,           // sampling clock (e.g., 1 kHz)
|     input rst,               // synchronous reset
|     input btn_raw,           // raw button input
|     output reg state,        // toggled output
|     output reg btn_toggle    // 1-cycle pulse on clean rising edge
| );
|
|     reg [2:0] shift_reg = 3'b000;
|     reg btn_clean = 0;
|     reg btn_prev = 0;
|
|     always @(posedge clk_1kHz) begin
|         // Debounce logic: 3-sample shift register
|         shift_reg <= {shift_reg[1:0], btn_raw};
|         if (shift_reg == 3'b111)
|             btn_clean <= 1;
|         else if (shift_reg == 3'b000)
|             btn_clean <= 0;
|
|         // Edge detection and toggle logic
|         if (rst) begin
|             state <= 0;
|             btn_prev <= 0;
|             btn_toggle <= 0;
|         end else begin
|             btn_toggle <= 0; // default to 0
|             if (btn_clean && !btn_prev) begin
|                 state <= ~state;
|                 btn_toggle <= 1; // generate 1-cycle pulse
|             end
|             btn_prev <= btn_clean;
|         end
|     end
| end
| endmodule

```

barrel_shifter16.V

- Rotates and shifts by doing the logical math and shifts

```

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

    genvar i;
    generate
        for (i = 0; i < 4; i = i + 1) begin : shift_stages
            localparam SHIFT_AMOUNT = 1 << i;
            wire [15:0] in = stage[i];
            wire [15:0] out;

            assign out = (shamt[i]) ?
                ( (dir == 0) ? // Left Shift/Rotate
                    ( (rotate) ? {in[15 - SHIFT_AMOUNT:0], in[15:16 - SHIFT_AMOUNT]} :
                        {in[15 - SHIFT_AMOUNT:0], {SHIFT_AMOUNT{1'b0}}} ) :
                    ( (rotate) ? {in[SHIFT_AMOUNT - 1:0], in[15:SHIFT_AMOUNT]} :
                        {{SHIFT_AMOUNT{1'b0}}, in[15:SHIFT_AMOUNT]} )
                ) :
                in;

            assign stage[i+1] = out;
        end
    endgenerate

    assign data_out = stage[4];
endmodule

```

Shamt_counter.V

- Uses a counter to determine by how much the shifting will be done.
- Lower 2 bits are controlled by left and right button

```

) module shamt_counter(
    input clk,
    input reset,
    input inc,
    output reg [1:0] shamt_high
);
    always @(posedge clk) begin
        if (reset)
            shamt_high <= 2'b00;
        else if (inc)
            shamt_high <= shamt_high + 1;
        end
    endmodule

```

hex_to_7segment.V

- Initializes the hex code to display on the 7seg display using a case statement.

```

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;
        default: SEG = 7'b1111111;
    endcase
    end
endmodule

```

seg7_scan8.V

- Scans which 7seg display is needed to be used and sends over hex to show the values on the 7seg display

```

module seg7_scan8(
    input clk_1kHz,
    input [15:0] hex,
    output reg [3:0] AN,
    output reg [6:0] SEG
);
    reg [1:0] digit_counter = 0;

    // Wire to connect the combinational output of hex_to_7seg
    wire [3:0] nibble_to_display;
    wire [6:0] segment_pattern;

    // Instance of the hex_to_7seg module
    hex_to_7seg h2s_inst (
        .hex(nibble_to_display),
        .SEG(segment_pattern)
    );

    always @(posedge clk_1kHz) begin
        // Cycle through the four digits (0 to 3)
        digit_counter <= (digit_counter == 2'b11) ? 2'b00 : digit_counter + 1;

        // Activate the correct anode for the current digit
        AN <= ~(4'b0001 << digit_counter);

        // Update the segment output with the pattern from the hex_to_7seg module
        SEG <= segment_pattern;
    end

    // Combinational logic to select the correct nibble to display
    assign nibble_to_display = (digit_counter == 2'b00) ? hex[3:0] :
        (digit_counter == 2'b01) ? hex[7:4] :
        (digit_counter == 2'b10) ? hex[11:8] :
        hex[15:12];

endmodule

```

top_lab7.V

- Calls all modules to the top module for the project to run.


```

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

    wire clk_1kHz, clk_demo;

    // Instantiate clock_divider_fixed
    clock_divider_fixed div_fixed_inst (
        .clk(clk),
        .clk_1kHz(clk_1kHz),
        .clk_demo(clk_demo)
    );

    // Synchronous reset from button center
    wire rst;
    assign rst = BTNC;

    // Debounce and toggle logic for direction and rotation
    wire state_dir, state_rot;
    wire toggle_dir, toggle_rot;

    debounce_toggle db_dir(
        .clk_1kHz(clk_1kHz),
        .rst(rst),
        .btn_raw(BTNL),
        .state(state_dir),
        .btn_toggle(toggle_dir)
    );

    debounce_toggle db_rot(
        .clk_1kHz(clk_1kHz),
        .rst(rst),
        .btn_raw(BTNR),
        .state(state_rot),
        .btn_toggle(toggle_rot)
    );

```

```

wire state_inc, state_dec;
wire toggle_inc, toggle_dec;

// Use debounce_toggle for BTNU
debounce_toggle db_inc(
    .clk_1kHz(clk_1kHz),
    .rst(BTNC),
    .btn_raw(BTNU),
    .state(state_inc),
    .btn_toggle(toggle_inc)
);

// Use debounce_toggle for BTND
debounce_toggle db_dec(
    .clk_1kHz(clk_1kHz),
    .rst(BTNC),
    .btn_raw(BTND),
    .state(state_dec),
    .btn_toggle(toggle_dec)
);

// Shamt counter for BTNU
wire [3:0] shamt;
wire [1:0] shamt_counter_out;

shamt_counter sc(
    .clk(clk_1kHz),
    .reset(BTNC),
    .inc(toggle_inc), // Use the one-cycle pulse from the debouncer to increment
    .shamt_high(shamt_counter_out)
);

// Assign the full 4-bit shamt
assign shamt = {shamt_counter_out, state_dec}; // Example, adjust as needed

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

// 7-segment display scanner
wire [3:0] AN_4digit;
seg7_scan8 scan(
    .clk_1kHz(clk_1kHz),
    .hex(result),
    .SEG(SEG),
    .AN(AN_4digit)
);

// Connect AN outputs
assign AN[3:0] = AN_4digit;
assign AN[7:4] = 4'b1111; // Disable unused digits

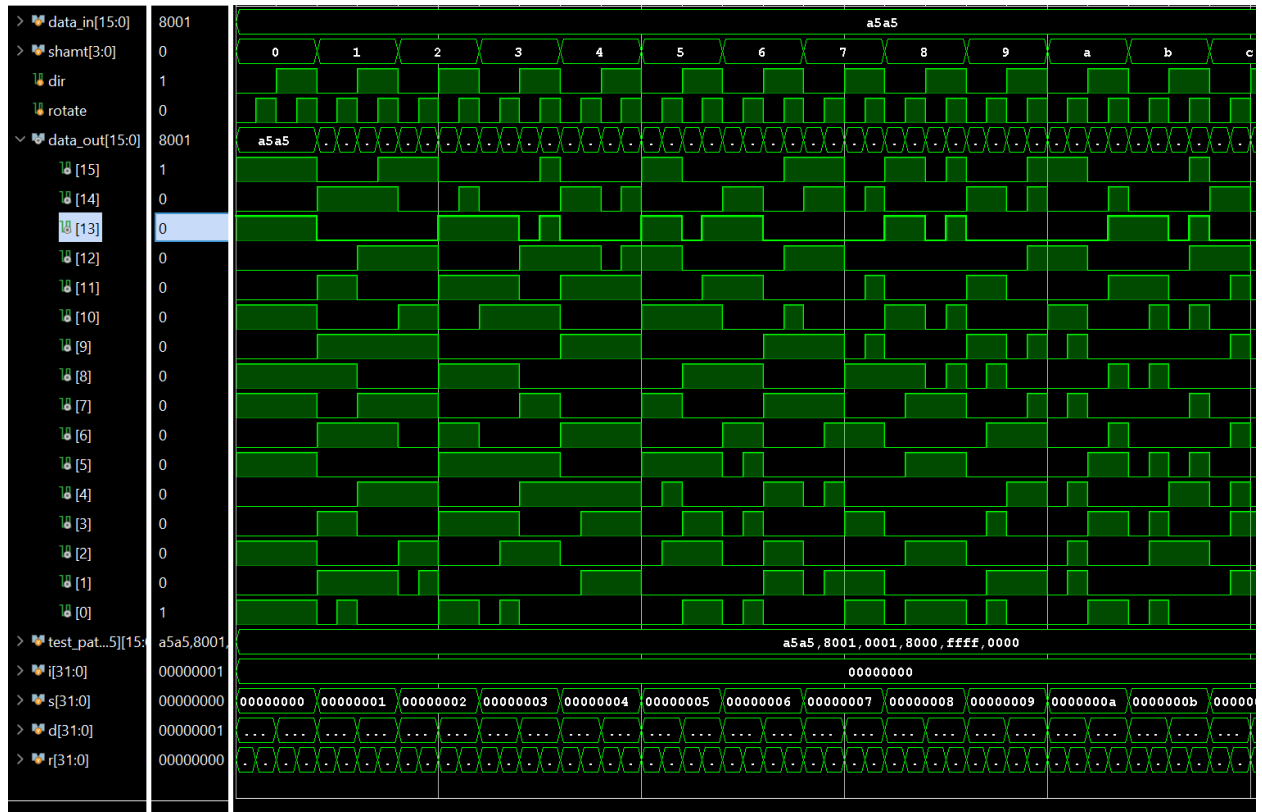
// LED assignments for debugging and status
assign LED = {2'b00, state_rot, state_dir, shamt};
) endmodule

```


















Testbench and Waveform:

barrel_shifter16_tb.V

- Shifting the bit values



seg7_scan8_tb.V

✓  SEG[6:0]	40					40
 [6]	1					
 [5]	0					
 [4]	0					
 [3]	0					
 [2]	0					
 [1]	0					
 [0]	0					
✓  AN[7:0]	fe					fe
 [7]	1					
 [6]	1					
 [5]	1					
 [4]	1					
 [3]	1					
 [2]	1					
 [1]	1					
 [0]	0					

Implementation:

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	107	0	0	63400	0.17
LUT as Logic	107	0	0	63400	0.17
LUT as Memory	0	0	0	19000	0.00
Slice Registers	71	0	0	126800	0.06
Register as Flip Flop	71	0	0	126800	0.06
Register as Latch	0	0	0	126800	0.00
F7 Muxes	0	0	0	31700	0.00
F8 Muxes	0	0	0	15850	0.00

Contributions:

Julio Flores: 50% - Verilog Code (shamt, hex_to_7seg, andseg7_scan8 code and testbench for these modules) and report and demo

Victor Perez : 50% - Verilog Code (clock divider, debounce, and barrel shifter code and testbench for these modules) and report and demo

Reflection:

In this lab, we designed and implemented a 16-bit barrel shifter and rotator along with a 4-digit 7-segment display interface on the Nexys A7 FPGA board. The project involved integrating multiple components such as control logic for selecting shift/rotate directions and distances, a barrel shifter capable of handling left/right operations, and a display driver for real-time output visualization. This lab reinforced the understanding of combinational logic in data manipulation through shifting and rotating, while also applying sequential logic for managing display timing and multiplexing. It highlighted the importance of modular design and provided practical experience in interfacing digital components in a cohesive system.