ECE 3300 – Lab 7 Report
16-bit Barrel Shifter / Rotator & 4-Digit 7-Segment Display
Team Name: Group V
Date: August 6, 2025

Group Members

Nathan Marlow, Khristian Chan

1. Objective

The goal of this lab was to design and implement a 16-bit combinational barrel shifter system on the Nexys A7 FPGA using Verilog. The system allows a 16-bit input word to be logically shifted or rotated either left or right by an arbitrary shift amount from 0 to 15. The direction, rotate/logical selection, and the shift amount were controlled using five push-buttons, and the output was displayed in real-time on a 4-digit 7-segment display. Additional design elements included a fixed clock divider, a debounce toggle system, and a 2-bit SHAMT counter, all of which were integrated to allow clean and observable shift operations.

2. Design Modules

2.1 Barrel Shifter

```
`timescale 1ns / 1ps
module barrel_shifter16(
    input [15:0] data_in,
    input [3:0] shamt,
    input dir,
    input rotate,
    output reg [15:0] data_out
);
    wire [15:0] stage0, stage1, stage2, stage3;

    assign stage0 = (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 stage1 = (shamt[1]) ? (dir ?
        (rotate ? {stage0[1:0], stage0[15:2]} : {2'b00, stage0[15:2]}) :
        (rotate ? {stage0[13:0], stage0[15:14]} : {stage0[13:0], 2'b00})) : stage0;

    assign stage2 = (shamt[2]) ? (dir ?
        (rotate ? {stage1[3:0], stage1[15:4]} : {4'b0000, stage1[15:4]}) :
        (rotate ? {stage1[11:0], stage1[15:12]} : {stage1[11:0], 4'b0000})) : stage1;

    assign stage3 = (shamt[3]) ? (dir ?
        (rotate ? {stage2[7:0], stage2[15:8]} : {8'b00000000, stage2[15:8]}) :
        (rotate ? {stage2[7:0], stage2[15:8]} : {stage2[7:0], 8'b00000000})) : stage2;

    always @(*) begin
        data_out = stage3;
    end
endmodule
```

A fully combinational 16-bit barrel shifter was implemented using 4 stages (1, 2, 4, 8-bit shifts). The design supports both logical and rotational shifting in either direction (left or right), as selected by control bits. The barrel_shifter16.v module uses conditional logic and concatenation to implement the shifts efficiently.

2.2 Clock Divider

```
`timescale 1ns / 1ps

// clock_divider_fixed.v
// for the two seperate speeds rather than a sel
module clock_divider_fixed(
    input clk,
    input rst_n,
    output reg clk_scan, // ~1kHz
    output reg clk_demo  // ~2Hz
);
    reg [25:0] scan_cnt;
    reg [25:0] demo_cnt;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            scan_cnt <= 0;
            demo_cnt <= 0;
            clk_scan <= 0;
            clk_demo <= 0;
        end else begin
            // ~1kHz from 100MHz clock (divide by 100000)
            if (scan_cnt == 100_000) begin
                scan_cnt <= 0;
                clk_scan <= ~clk_scan;
            end else begin
                scan_cnt <= scan_cnt + 1;
            end

            // ~2Hz from 100MHz clock (divide by 25_000_000)
            if (demo_cnt == 25_000_000) begin
                demo_cnt <= 0;
                clk_demo <= ~clk_demo;
            end else begin
                demo_cnt <= demo_cnt + 1;
            end
        end
    end
endmodule
```

The clock_divider_fixed.v module divides the 100 MHz master clock into two frequencies:

- ~2 Hz: Used for shift timing and SHAMT counter

- ~1 kHz: Used for debouncing and display scanning

This dual-clock design enables precise control over shift steps while maintaining fast, flicker-free 7-segment display updates.

2.3 Debounce Toggle

```
`timescale 1ns / 1ps

module debounce_toggle(
    input clk_1kHz,
    input btn_raw,
    output reg btn_toggle
);
    reg [2:0] shift_reg;
    reg state;

    always @(posedge clk_1kHz) begin
        shift_reg <= {shift_reg[1:0], btn_raw};

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

Mechanical button bounces were cleaned using the debounce_toggle.v module, which samples the button state using the 1 kHz clock and registers a toggle only when a clean rising edge is detected. This ensures reliable toggling behavior without glitches.

2.4 SHAMT Counter

```
module shamt_counter(
    input clk,
    input rst_n,
    input inc,
    output reg [1:0] shamt_high
);
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            shamt_high <= 0;
        else if (inc)
            shamt_high <= shamt_high + 1;
    end
endmodule
```

To reduce the number of input buttons required, a 2-bit counter was implemented using the shamt_counter.v module. This counter increments on each press of the center button (BTNC), allowing full 4-bit shift amounts (0–15) to be selected using only 3 buttons.

2.5 7-Segment Display

```
`timescale 1ns / 1ps

module seg7_scan8(
    input clk,
    input rst_n,
    input clk_1kHz,
    input [3:0] dig0,
    input [3:0] dig1,
    input [3:0] dig2,
    input [3:0] dig3,
    output reg [7:0] an,
    output [6:0] seg
);
    reg [1:0] sel;
    reg [3:0] digit;

    always @(posedge clk_1kHz or negedge rst_n) begin
        if (!rst_n)
            sel <= 0;
        else
            sel <= sel + 1;
    end

    always @(*) begin
        case(sel)
            2'b00: begin an = 8'b11111110; digit = dig0; end
            2'b01: begin an = 8'b11111101; digit = dig1; end
            2'b10: begin an = 8'b11111011; digit = dig2; end
            2'b11: begin an = 8'b11110111; digit = dig3; end
            default: begin an = 8'b11111111; digit = 4'd0; end
        endcase
    end

    hex_to_7seg u_hex(.hex(digit), .seg(seg));

endmodule
```

```
`timescale 1ns / 1ps

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
```

The 16-bit output of the barrel shifter is broken into four 4-bit nibbles, which are converted to hexadecimal using the hex_to_7seg.v module. The seg7_scan8.v module handles multiplexed scanning of the display and drives digits AN0 to AN3.

2.6 Top-Level Integration

```
`timescale 1ns / 1ps

module top_lab7(
    input clk,
    input rst_n,
    input [15:0] sw,
    input btnU,
    input btnD,
    input btnL,
    input btnR,
    input btnC,
    output [7:0] led,
    output [6:0] seg,
    output [7:0] an
);
    wire clk_1kHz, clk_demo;
    wire dir, rot, shamt1, shamt0;
    wire [1:0] shamt_high;
    wire [3:0] shamt;
    wire [15:0] result_word;

    // Clock divider
    clock_divider_fixed u_clk_div(
        .clk(clk), .rst_n(rst_n),
        .clk_scan(clk_1kHz), .clk_demo(clk_demo)
    );

    // Debounce toggle buttons
    debounce_toggle u_dir(.clk_1kHz(clk_1kHz), .btn_raw(btnU), .btn_toggle(dir));
    debounce_toggle u_rot(.clk_1kHz(clk_1kHz), .btn_raw(btnD), .btn_toggle(rot));
    debounce_toggle u_s1(.clk_1kHz(clk_1kHz), .btn_raw(btnL), .btn_toggle(shamt1));
    debounce_toggle u_s0(.clk_1kHz(clk_1kHz), .btn_raw(btnR), .btn_toggle(shamt0));

    assign shamt[1:0] = {shamt1, shamt0};
    // SHAMT[1:0] for lower two bits and are manual toggles
    // SHAMT[3:2] counter for the upper two bits for shifting based off center button press
    shamt_counter u_shamt_count(
        .clk(clk_demo), .rst_n(rst_n), .inc(btnC), .shamt_high(shamt_high)
    );

    assign shamt[3:2] = shamt_high;
```

```
    assign shamt[3:2] = shamt_high;

    // Barrel shifter
    barrel_shifter16 u_shift(
        .data_in(sw), .shamt(shamt), .dir(dir), .rotate(rot), .data_out(result_word)
    );

    // Hex to 7-seg and scanner (reuse your hex_to_7seg and seg7_scan8)
    wire [3:0] nib0 = result_word[3:0];
    wire [3:0] nib1 = result_word[7:4];
    wire [3:0] nib2 = result_word[11:8];
    wire [3:0] nib3 = result_word[15:12];

    seg7_scan8 u_scan(
        .clk(clk), .rst_n(rst_n), .clk_1kHz(clk_1kHz),
        .dig0(nib0), .dig1(nib1), .dig2(nib2), .dig3(nib3),
        .an(an), .seg(seg)
    );

    // LED debug output: {DIR, ROT, SHAMT[3:0]}
    assign led = {dir, rot, shamt};

endmodule
```

The top_lab7.v module integrates all the above components. It connects:

- SW15–SW0 → input data word
- BTNU/BTND/BTNL/BTNR → DIR, ROT, SHAMT[1:0]
- BTNC → increments SHAMT[3:2]
- LED[7:0] → debug output for DIR, ROT, SHAMT[3:0]
- AN0–AN3 → display output

3. Implementation Results

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	78	0	0	126800	0.06
Register as Flip Flop	78	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

Total On-Chip Power (W)	0.133
Design Power Budget (W)	Unspecified*
Power Budget Margin (W)	NA
Dynamic (W)	0.036
Device Static (W)	0.097
Effective TJA (C/W)	4.6
Max Ambient (C)	84.4
Junction Temperature (C)	25.6
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

* Specify Design Power Budget using, set_operating_conditions -design_power_budget <value in Wat

1.1 On-Chip Components

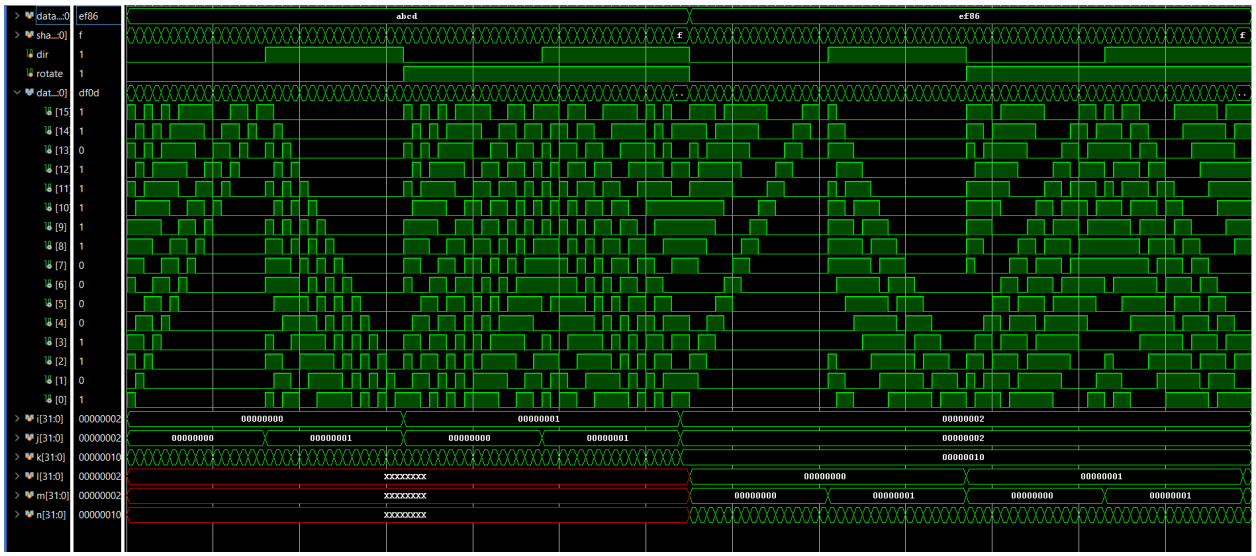
On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	<0.001	3	---	---
Slice Logic	0.001	252	---	---
LUT as Logic	<0.001	107	63400	0.17
Register	<0.001	78	126800	0.06
CARRY4	<0.001	14	15850	0.09
Others	0.000	14	---	---
Signals	0.001	242	---	---
I/O	0.033	46	210	21.90
Static Power	0.097			
Total	0.133			

Above are the results for both utilization and power. We see that even as project complexity gets higher, then overall utilization and power usage is very low compared to what a more involved FPGA/RTL project can be. We are using a very small amount of what is possible to use with the Nexys A7.

4. Simulation Results:

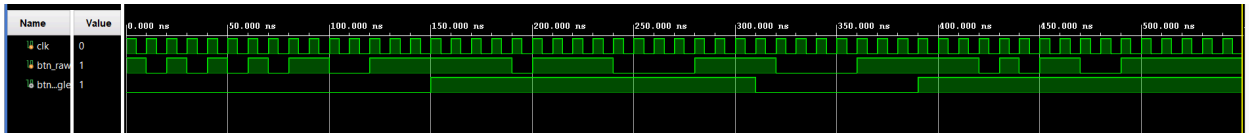
4.1 - Barrel Shifter

The barrel shifter simulation was successful, we went through two different input values and we see how we go from logical to normal rotate bitshift, along with forward and backwards, and different bitshifts. Using 3 for loops, we are able to iterate through every permutation of the possible bitshifts.



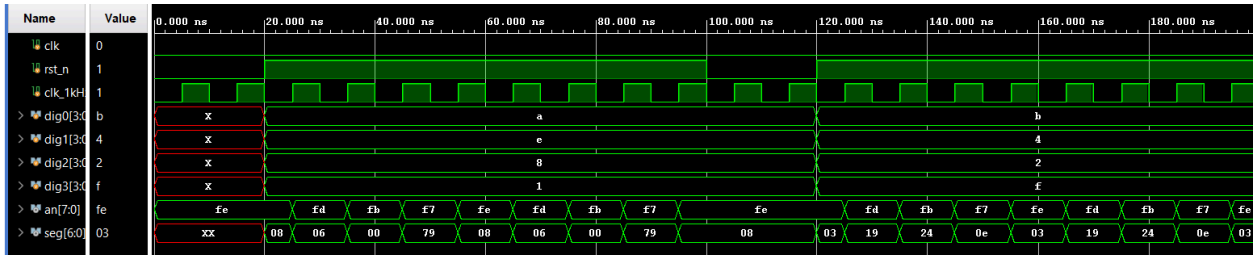
4.2- Debounce Toggle

We see our debounce simulation is successful, when we use inputs of less than 30ns, we get an output that latches low until we find a high input that is longer than 30ns. Same thing can be said for a low input that is long than 30ns. It will latch until it finds a long enough input of either 3 high/low in a row.



4.3- 7 Segment Scanner

We see our scanner iterate through each of our seven segment displays multiple times before we swap numbers. We see that hitting reset will freeze the anode position until it is let go.



6. Demonstration Video

https://youtu.be/szGX48fbeYA

Contributions:

Khristian Chan- 50%: Testbench, Simulation, Debugging, Report, Demo
Nathan Marlow- 50%: Implementation, Code, XDC, Report