16-bit Barrel Shifter / Rotator& 4-Digit 7-Segment Display
3300L
Dia Agrawal and Robert Lainez Torres

Objective:
This lab focuses on designing a 16-bit combinational barrel shifter on the Nexys A7-100T FPGA that supports logical and rotational shifts in both left and right directions, with a shift amount from 0 to 15. Inputs come from slide switches, while push-buttons control the shift direction, mode, and shift amount using debounced toggles and a 2-bit counter. The shifted output is displayed in real time on four digits of the 7-segment display. Key learning outcomes include understanding multiplexer-based shifter design, implementing button debouncing, managing multiple clock domains, and practicing modular FPGA development. The lab also emphasizes simulation and waveform analysis using testbenches, while introducing hardware trade-offs between flexibility and resource use.

Design Description

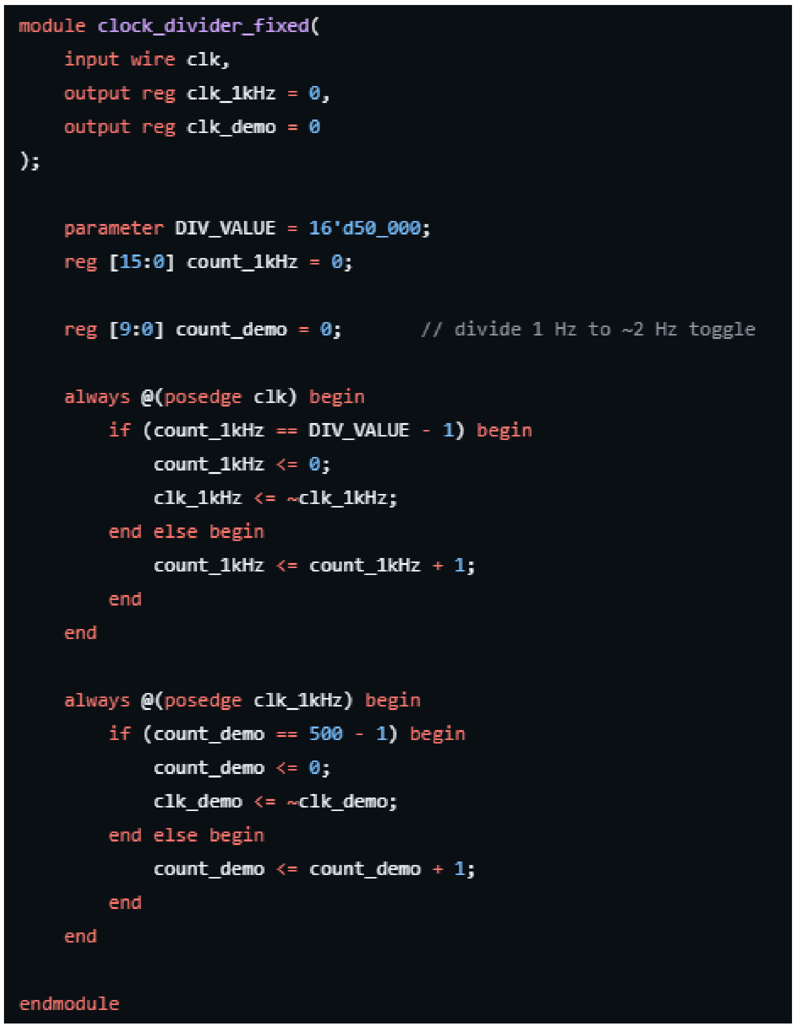
Barrel Shifter16:

```
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] stage1, stage2, stage3, stage4;
   // Stage shift amount = 1
   assign stage1 = shamt[0] ?
      (dir ?
         // right
         (rotate ? {data_in[0],      data_in[15:1]}  // ror 1
                 : {1'b0,            data_in[15:1]}) // >> 1 logical
        :
         // left
         (rotate ? {data_in[14:0],   data_in[15]}    // rol 1
                 : {data_in[14:0],   1'b0})          // << 1 logical
      )
      : data_in;
   // Stage shift amount = 2
   assign stage2 = shamt[1] ?
      (dir ?
         // right
         (rotate ? {stage1[1:0],     stage1[15:2]}   // ror 2
                 : {2'b00,           stage1[15:2]})  // >> 2 logical
        :
         // left
      (rotate ? {stage1[13:0],    stage1[15:14]}  // rol 2
                 : {stage1[13:0],    2'b00})         // << 2 logical
      )
      : stage1;

   // Stage shift amount = 4
```

```
    assign stage3 = shamt[2] ?
      (dir ?
        // right
        (rotate ? {stage2[3:0],     stage2[15:4]}    // ror 4
                : {4'b0000,         stage2[15:4]})  // >> 4 logical
       :
        // left
        (rotate ? {stage2[11:0],    stage2[15:12]}  // rol 4
                : {stage2[11:0],    4'b0000})       // << 4 logical
      )
      : stage2;

    // Stage shift amount = 8
    assign stage4 = shamt[3] ?
      (dir ?
        // right
        (rotate ? {stage3[7:0],     stage3[15:8]}    // ror 8
                : {8'b0000_0000,    stage3[15:8]})  // >> 8 logical
       :
        // left
        (rotate ? {stage3[7:0],     stage3[15:8]}    // rol 8 (same wiring as ror 8 on 16-bit)
                : {stage3[7:0],     8'b0000_0000})  // << 8 logical
      )
      : stage3;

    assign data_out = stage4;

Endmodule
```

This Verilog module implements a 16-bit combinational barrel shifter capable of performing both **logical and rotational shifts** in **left or right** directions based on control inputs. The input data in is shifted by an amount specified by shamt[3:0], using a staged approach where each bit in shamt controls a specific shift amount: 1, 2, 4, and 8 bits respectively. Each stage conditionally shifts the data depending on the direction (dir) and mode (rotate). If rotate is enabled, bits wrap around during the shift (e.g., a rotate-right brings LSBs to the MSB side); otherwise, logical shifts pad with zeros. The output from each stage feeds into the next, creating an efficient MUX-based design that completes the shift in one clock cycle. The final result is assigned to data_out..

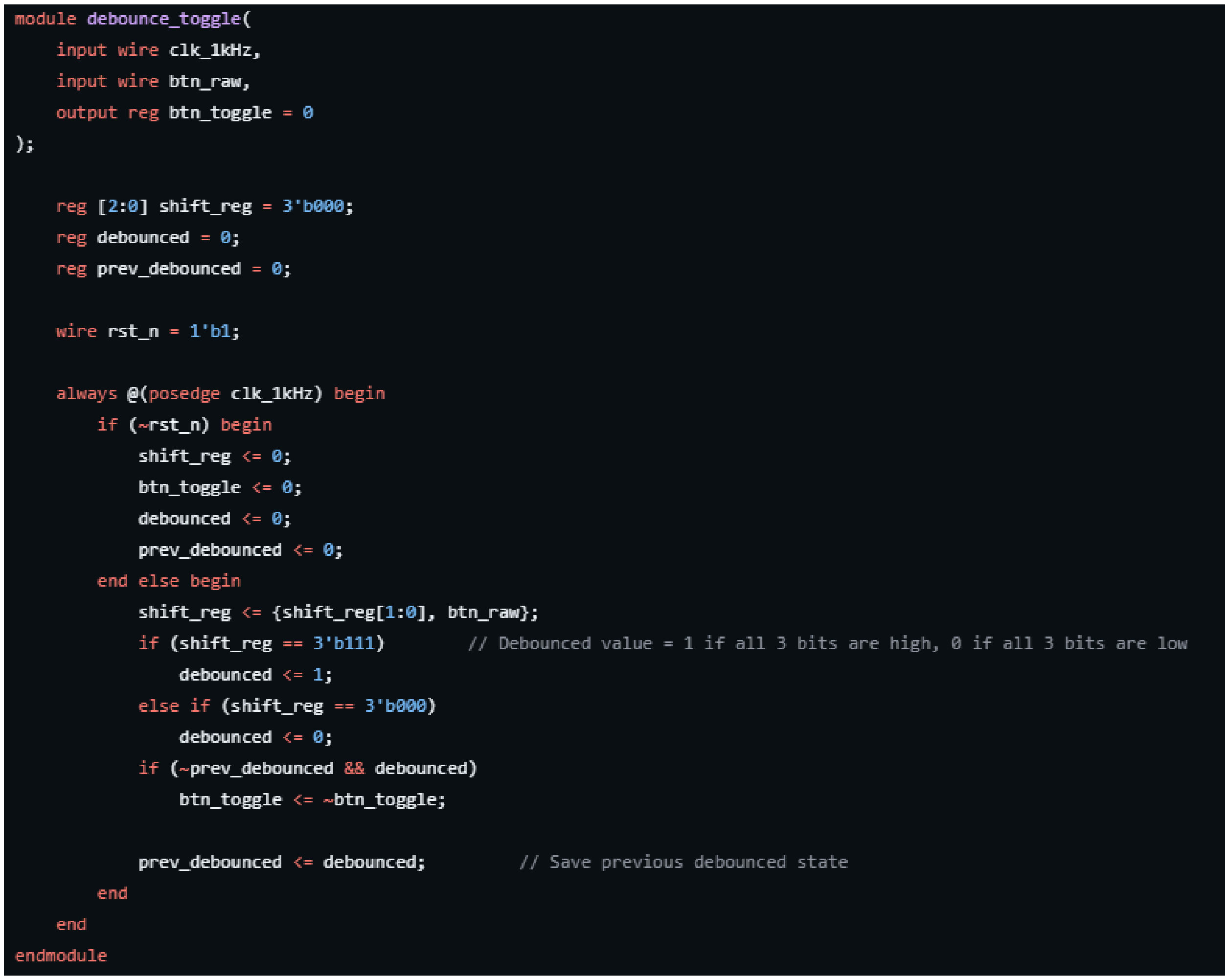
Clock Divider:



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

    parameter DIV_VALUE = 16'd50_000;
    reg [15:0] count_1kHz = 0;

    reg [9:0] count_demo = 0;       // divide 1 Hz to ~2 Hz toggle

    always @(posedge clk) begin
        if (count_1kHz == DIV_VALUE - 1) begin
            count_1kHz <= 0;
            clk_1kHz <= ~clk_1kHz;
        end else begin
            count_1kHz <= count_1kHz + 1;
        end
    end

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

endmodule
```

This Verilog module clock_diver_fixed generates two slower clocks from a fast input clock (clk) using simple counter-based division. The first output,clk_hrtz, toggles at a rate determined by the parameter div_value (set to 50,000), which divides a 100 MHz clock down to 1 kHz by counting clock cycles and flipping the output every 50,000 cycles. The second output, clk_demo, toggles at an even slower rate—about 2 Hz—by counting 500 rising edges of clk_1kHz and then toggling. This setup provides a 1 kHz clock suitable for tasks like 7-segment display multiplexing, and a visibly slow clock (clock demo) for demo purposes, such as cycling through shift amounts or observing output changes manually.

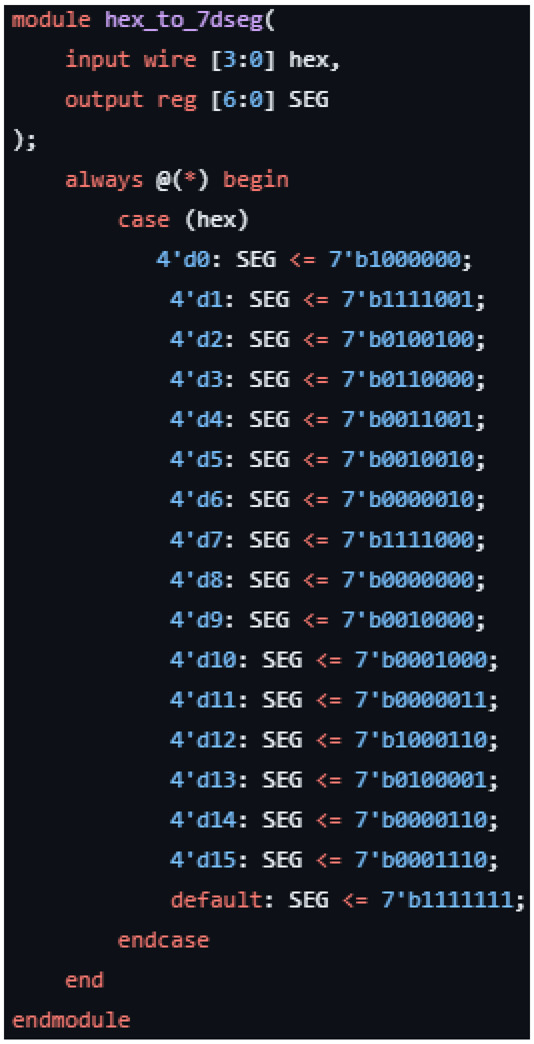
Debounce Toggle:



```
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;

    wire rst_n = 1'b1;

    always @(posedge clk_1kHz) begin
        if (~rst_n) begin
            shift_reg <= 0;
            btn_toggle <= 0;
            debounced <= 0;
            prev_debounced <= 0;
        end else 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
    end
endmodule
```

This debounce_toggle module filters out noise from a raw push-button signal (btn_raw) using a 3-bit shift register and produces a clean, toggled output btn_toggle on each legitimate button press. Clocked by a 1 kHz signal clk_1khz, it samples the button and stores recent states in Shift_reg Only when all three samples are consistently high or low does it update the debounced signal, preventing false triggers due to mechanical bouncing. A rising edge on the debounced signal (i.e., a clean button press) causes the output btn_toggle to invert, allowing a single button to switch states on each press—ideal for mode toggling or multi-state selection.

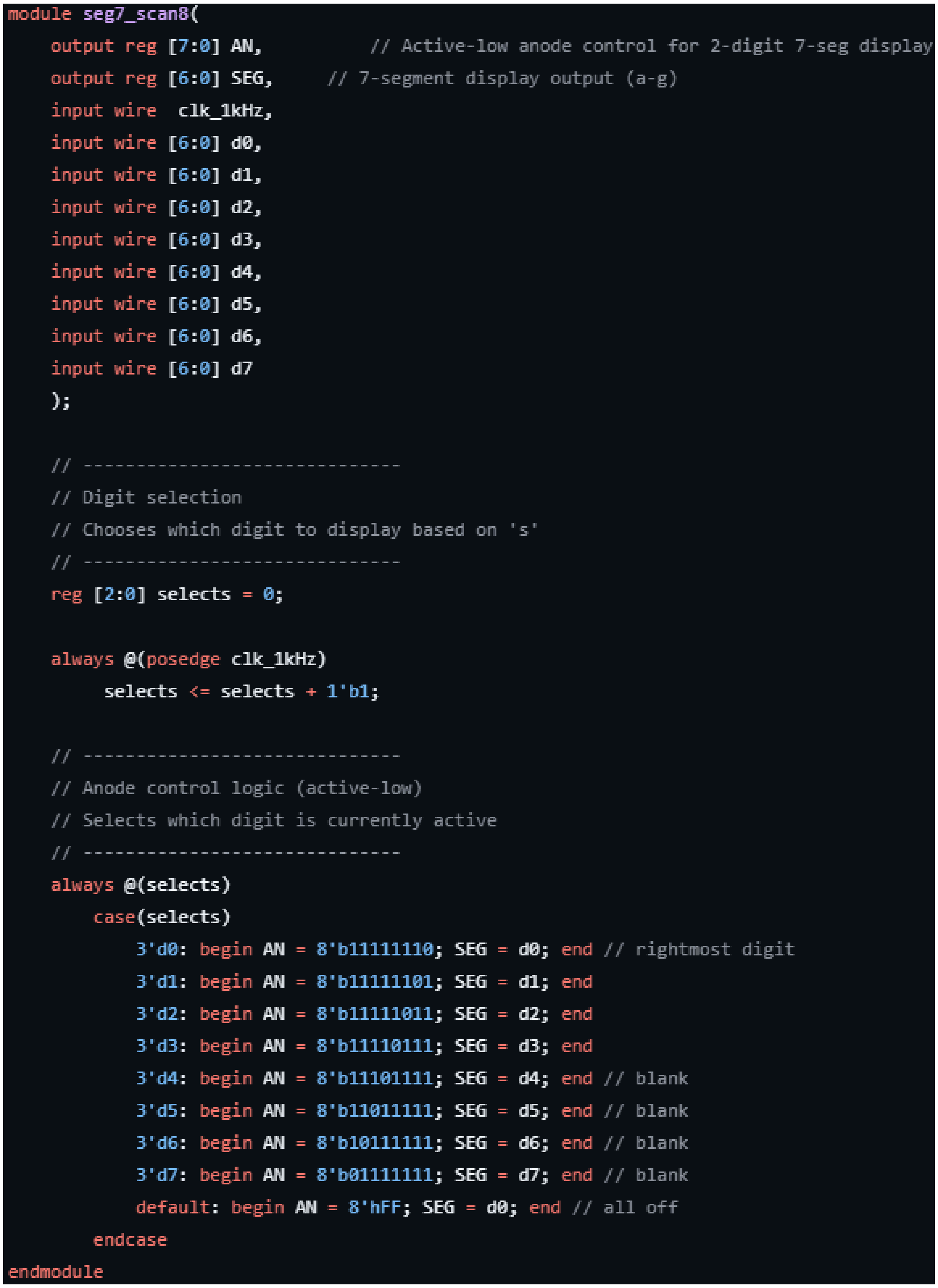
Hexadecimal to 7 display segment:



```
module hex_to_7dseg(
    input wire [3:0] hex,
    output reg [6:0] SEG
);
    always @(*) begin
        case (hex)
           4'd0: SEG <= 7'b1000000;
            4'd1: SEG <= 7'b1111001;
            4'd2: SEG <= 7'b0100100;
            4'd3: SEG <= 7'b0110000;
            4'd4: SEG <= 7'b0011001;
            4'd5: SEG <= 7'b0010010;
            4'd6: SEG <= 7'b0000010;
            4'd7: SEG <= 7'b1111000;
            4'd8: SEG <= 7'b0000000;
            4'd9: SEG <= 7'b0010000;
            4'd10: SEG <= 7'b0001000;
            4'd11: SEG <= 7'b0000011;
            4'd12: SEG <= 7'b1000110;
            4'd13: SEG <= 7'b0100001;
            4'd14: SEG <= 7'b0000110;
            4'd15: SEG <= 7'b0001110;
            default: SEG <= 7'b1111111;
        endcase
    end
endmodule
```

Converts a 4 bit hexadecimal input in a 7 bit output to drive the 7 segment display

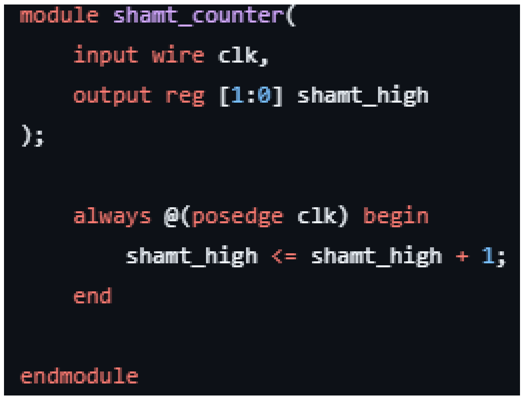
Segment 7 to scan 8:



```
module seg7_scan8(
    output reg [7:0] AN,          // Active-low anode control for 2-digit 7-seg display
    output reg [6:0] SEG,     // 7-segment display output (a-g)
    input wire  clk_1kHz,
    input wire [6:0] d0,
    input wire [6:0] d1,
    input wire [6:0] d2,
    input wire [6:0] d3,
    input wire [6:0] d4,
    input wire [6:0] d5,
    input wire [6:0] d6,
    input wire [6:0] d7
    );

    // ------------------------------
    // Digit selection
    // Chooses which digit to display based on 's'
    // ------------------------------
    reg [2:0] selects = 0;

    always @(posedge clk_1kHz)
         selects <= selects + 1'b1;

    // ------------------------------
    // Anode control logic (active-low)
    // Selects which digit is currently active
    // ------------------------------
    always @(selects)
        case(selects)
            3'd0: begin AN = 8'b11111110; SEG = d0; end // rightmost digit
            3'd1: begin AN = 8'b11111101; SEG = d1; end
            3'd2: begin AN = 8'b11111011; SEG = d2; end
            3'd3: begin AN = 8'b11110111; SEG = d3; end
            3'd4: begin AN = 8'b11101111; SEG = d4; end // blank
            3'd5: begin AN = 8'b11011111; SEG = d5; end // blank
            3'd6: begin AN = 8'b10111111; SEG = d6; end // blank
            3'd7: begin AN = 8'b01111111; SEG = d7; end // blank
            default: begin AN = 8'hFF; SEG = d0; end // all off
        endcase
endmodule
```

The seg7_scan8 module drives an 8-digit 7-segment display by cycling through each digit rapidly to create the appearance that all digits are lit simultaneously (a method called multiplexing). It takes a 1 kHz clock to control the refresh rate and inputs for each digit d0 through d7, each being a 7-bit segment pattern.

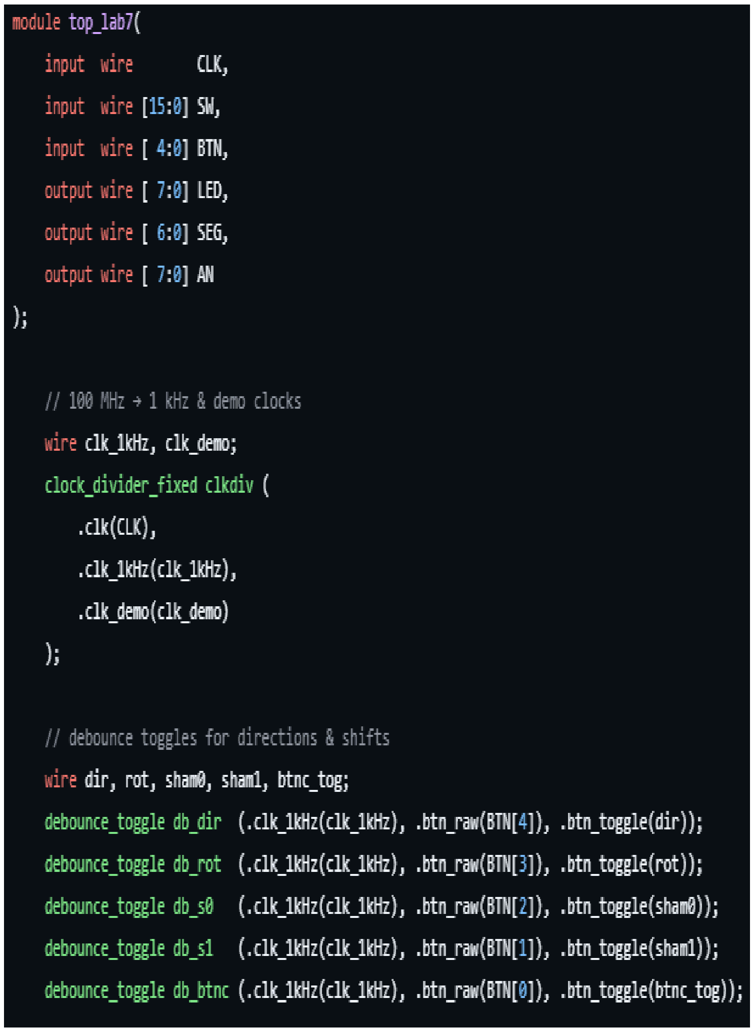
Sham Counter:



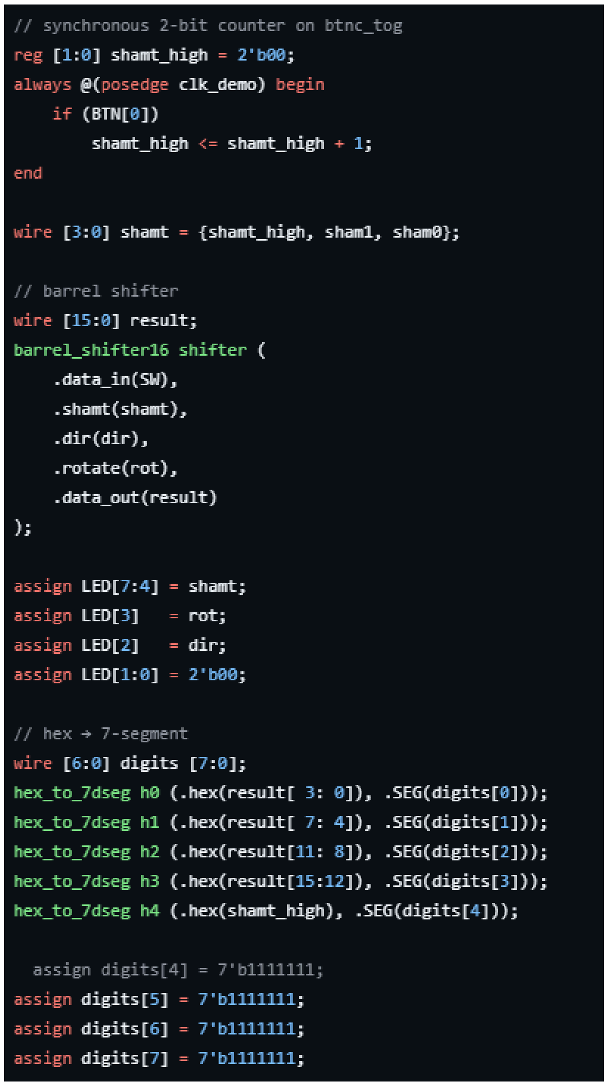
```
module shamt_counter(
    input wire clk,
    output reg [1:0] shamt_high
);

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

endmodule
```

The sham counter module is a simple 2-bit counter that increments on every rising edge of the clock signal. The output shamt_counter holds the current count value, cycling repeatedly from 0 to 3 2'b00 through 2'b11 and then wrapping back to 0 due to its 2-bit width. This kind of counter can be used to cycle through 2-bit values, such as controlling higher-order shift amounts or selecting among four options in digital logic circuits.

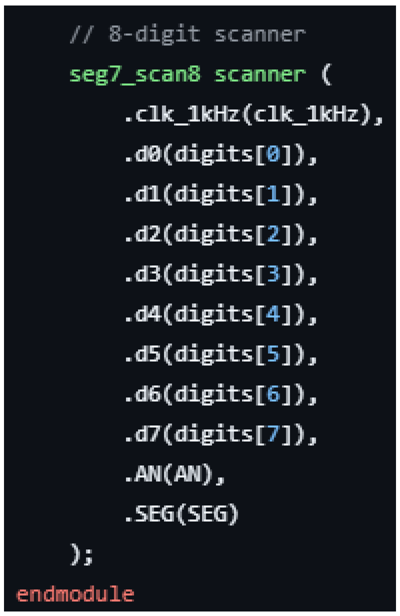
Top Module:



```
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
);

    // 100 MHz → 1 kHz & demo clocks
    wire clk_1kHz, clk_demo;
    clock_divider_fixed clkdiv (
        .clk(CLK),
        .clk_1kHz(clk_1kHz),
        .clk_demo(clk_demo)
    );

    // debounce toggles for directions & shifts
    wire dir, rot, sham0, sham1, btnc_tog;
    debounce_toggle db_dir  (.clk_1kHz(clk_1kHz), .btn_raw(BTN[4]), .btn_toggle(dir));
    debounce_toggle db_rot  (.clk_1kHz(clk_1kHz), .btn_raw(BTN[3]), .btn_toggle(rot));
    debounce_toggle db_s0   (.clk_1kHz(clk_1kHz), .btn_raw(BTN[2]), .btn_toggle(sham0));
    debounce_toggle db_s1   (.clk_1kHz(clk_1kHz), .btn_raw(BTN[1]), .btn_toggle(sham1));
    debounce_toggle db_btnc (.clk_1kHz(clk_1kHz), .btn_raw(BTN[0]), .btn_toggle(btnc_tog));
```



```
// synchronous 2-bit counter on btnc_tog
reg [1:0] shamt_high = 2'b00;
always @(posedge clk_demo) begin
    if (BTN[0])
        shamt_high <= shamt_high + 1;
end

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

// barrel shifter
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;

// hex → 7-segment
wire [6:0] digits [7:0];
hex_to_7dseg h0 (.hex(result[ 3: 0]), .SEG(digits[0]));
hex_to_7dseg h1 (.hex(result[ 7: 4]), .SEG(digits[1]));
hex_to_7dseg h2 (.hex(result[11: 8]), .SEG(digits[2]));
hex_to_7dseg h3 (.hex(result[15:12]), .SEG(digits[3]));
hex_to_7dseg h4 (.hex(shamt_high), .SEG(digits[4]));

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

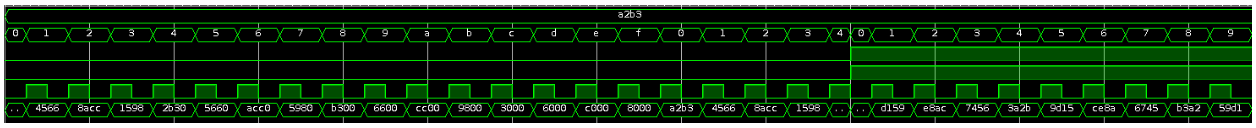


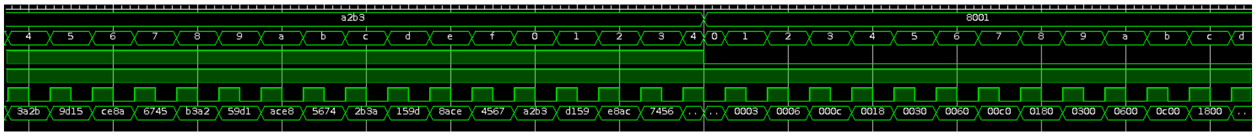
```
    // 8-digit scanner
    seg7_scan8 scanner (
        .clk_1kHz(clk_1kHz),
        .d0(digits[0]),
        .d1(digits[1]),
        .d2(digits[2]),
        .d3(digits[3]),
        .d4(digits[4]),
        .d5(digits[5]),
        .d6(digits[6]),
        .d7(digits[7]),
        .AN(AN),
        .SEG(SEG)
    );
endmodule
```

This module connects inputs like switches sw and pushbuttons BTN to a 16-bit barrel shifter, displaying results through LEDs and 7-segment displays. A clock_divider_fixed generates slower clocks 1kHZ and demo from the 100 MHz system clock for button debouncing and timed actions. Each button is passed through a debounce_toggle module to clean up mechanical noise, allowing users to control the shift direction dir , shift/rotate mode rot, and lower shift amount bits sham0, sham1. Pressing the center button btn[0] increments a 2-bit counter on the slower demo clock, forming a 4-bit shift amount when combined with sham1 and sham0. The 16-bit input from SW is passed into a barrel_shifer module that performs logical or rotate left/right shifts depending on the control signals. The result is displayed on LEDs LED[7:4] for shift amount, LED [3] and LED[2] for control bits) and broken into 4-bit chunks for display on the first 4 digits of a 7-segment display via hex_to_7seg decoders. One digit shows the current shamt_high value, and the remaining digits are blank. Finally, the seg7_scan8   module scans all 8 digits rapidly, creating a stable multi-digit display of the result.

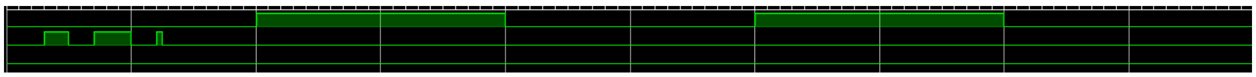
Test Bench

Barrel shifter tb

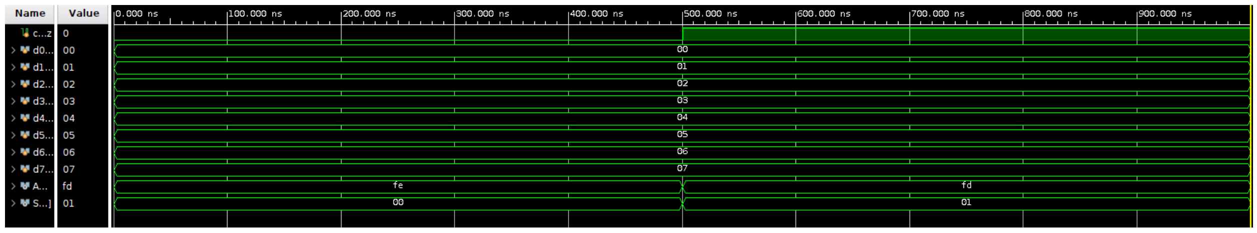




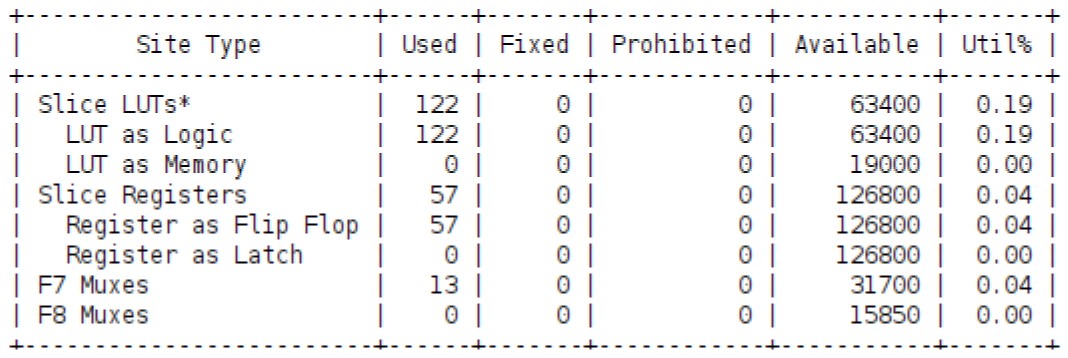
Debounce toggle tb



Seg7_scan8 tb



Utilization report



```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |  122 |     0 |          0 |     63400 |  0.19 |
|   LUT as Logic          |  122 |     0 |          0 |     63400 |  0.19 |
|   LUT as Memory         |    0 |     0 |          0 |     19000 |  0.00 |
| Slice Registers         |   57 |     0 |          0 |    126800 |  0.04 |
|   Register as Flip Flop |   57 |     0 |          0 |    126800 |  0.04 |
|   Register as Latch     |    0 |     0 |          0 |    126800 |  0.00 |
| F7 Muxes                |   13 |     0 |          0 |     31700 |  0.04 |
| F8 Muxes                |    0 |     0 |          0 |     15850 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

Team Contributions

As usual, we both contributed throughout the design, testing, and documentation process, often working together to integrate and verify our modules.

Robert's Contributions:

Created the barrel shifter, clock divider, debouncing, hex to 7 segment and shamt counter modules

Dia's Contributions:

Crated all the test benches barreal shifter, debouncing seg7 to scan8 and top module, along with recording the demo video.

We both participated in debugging, testbench design, and reviewing each other’s code to ensure correctness. Final documentation and polishing were done collaboratively.