

Emmett Stralka

Lab2 Report MircoP's // Sep 7th 2025

Introduction

This lab involved implementing a design on an FPGA to verify the proper operation of multiplexing control for a dual seven-segment display system. The goal was to display two independent hexadecimal digits on a shared set of segment lines by rapidly switching the active digit using time multiplexing. Inputs were taken from switches to provide values for each digit, and the sum of the two digits was output on LEDs. A single seven-segment decoder module was utilized to drive both digits in a multiplexed fashion, with additional hardware support from transistor switching circuits to handle the current requirements for the common anode displays.

Design and Testing Methodology

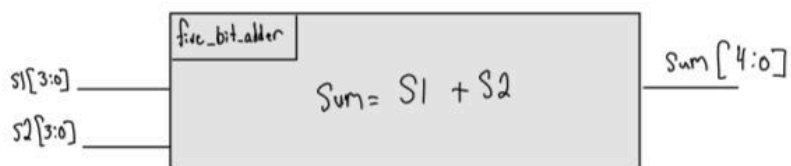
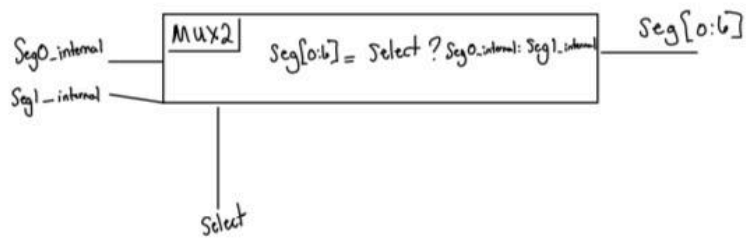
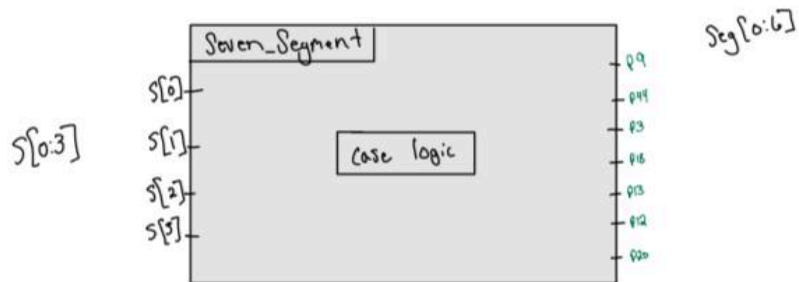
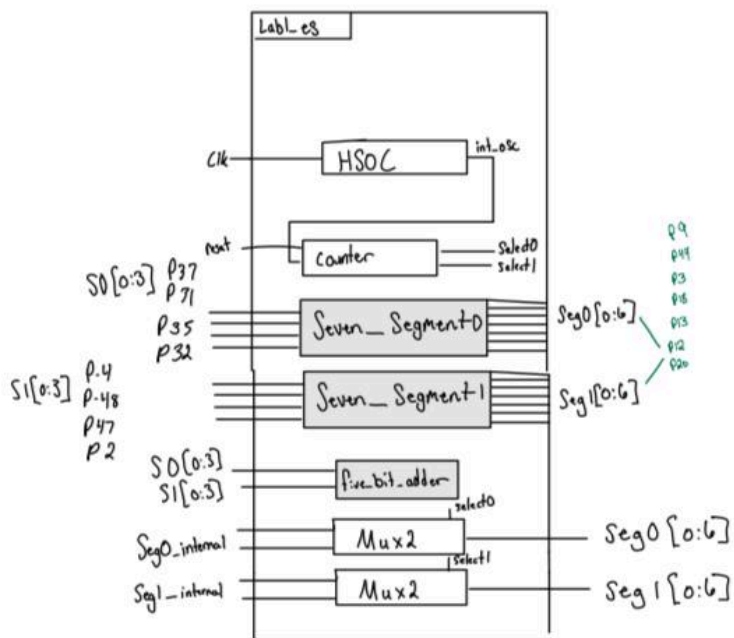
The system was developed using System Verilog and structured modularly, with separate modules for the seven-segment decoder, clock divider, a summing, and top digit multiplexing controller. Simulation was performed using QuestaSim with a comprehensive testbench to verify correct segment patterns and multiplex timing for all hexadecimal values. The hardware was tested on an FPGA development board, where the transistor circuits for driving the display anodes were validated and input switch functionality confirmed. Observations were made to ensure no flickering occurred due to timing or current issues, and that the LED sum outputs matched the expected results.

Technical Documentation:

The source code for the project can be found in the associated [Github repository](#).

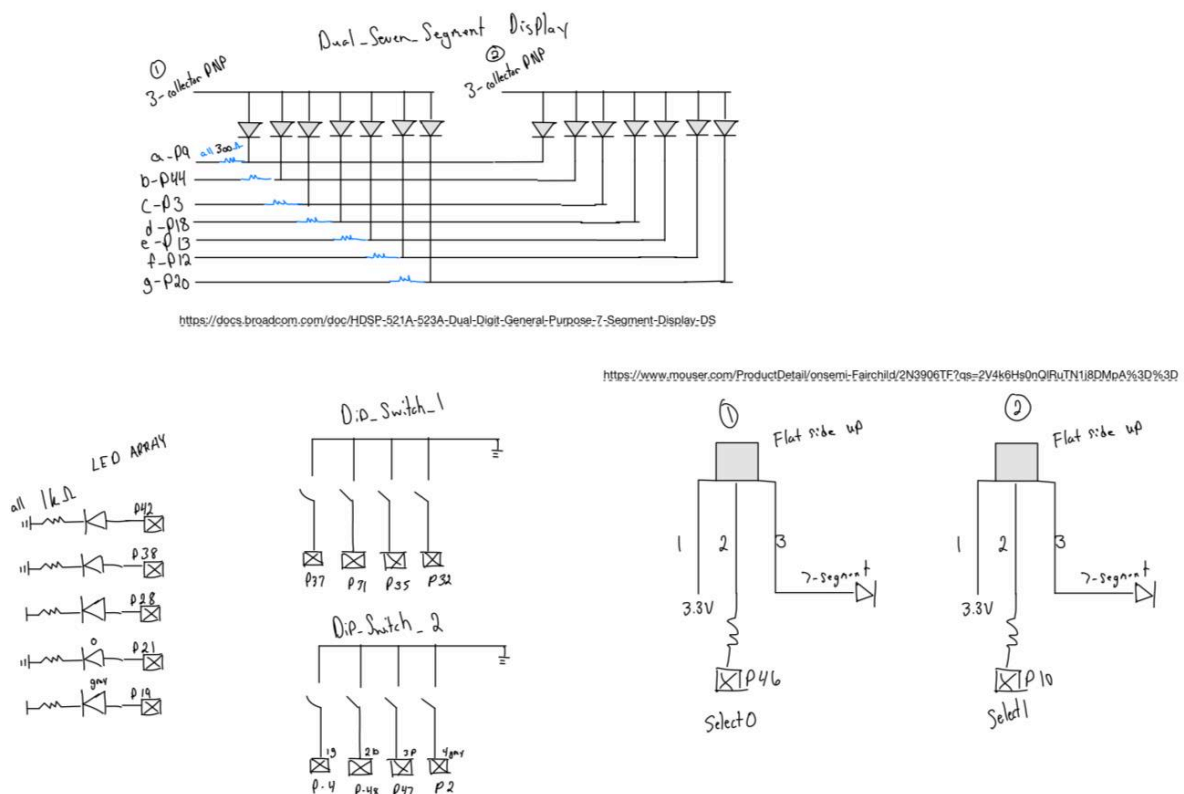
Lab2 Specific Developed Code can be found at: [Github repository](#)

Block Diagram



The block diagram in [Figure 1](#) demonstrates the overall architecture of the design. The top-level module `Lab2_es` includes four submodules: the high-speed oscillator block (`HSOSC`), the clock divider module (`clk_divider`), the seven_segment module (`seven_segment`), the Muxilplexer with 2 inputs module (`MUX2`), and the five-bit adder module (`five_bit_adder`)

Lab2. FS_BreadBoard



The physical circuit schematic (see Figure 2) shows the wiring of FPGA I/O pins to the display and external components. Transistor circuits using 2N3906 PNP transistors drive the higher-current common anode lines for each digit, with base resistors sized for

safe operation (10kΩ). Segment lines pass through 300Ω resistors to limit current and protect the FPGA pins. Switch inputs are connected with pull-up or pull-down resistors to ensure stable input levels. The LED array and seven-segment segments are wired for multiplexed operation with careful grounding considerations to avoid floating nodes.

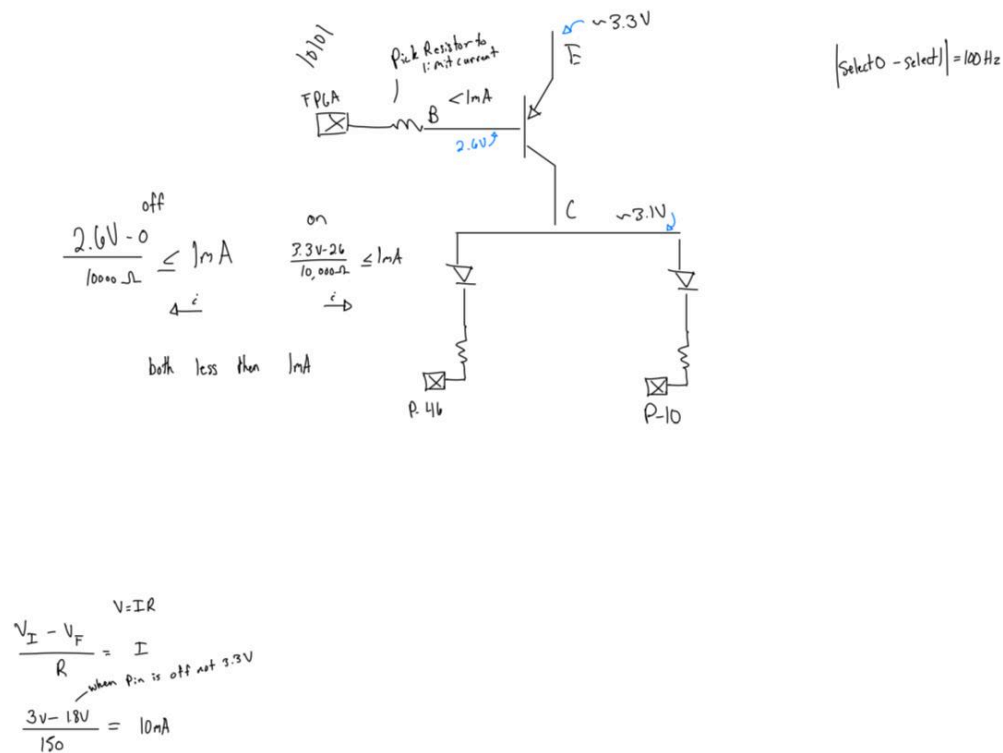


Figure 3: Calculations for physical circuit LEDS segments and Cycle Divider

Results and Discussion

Testbench Simulation

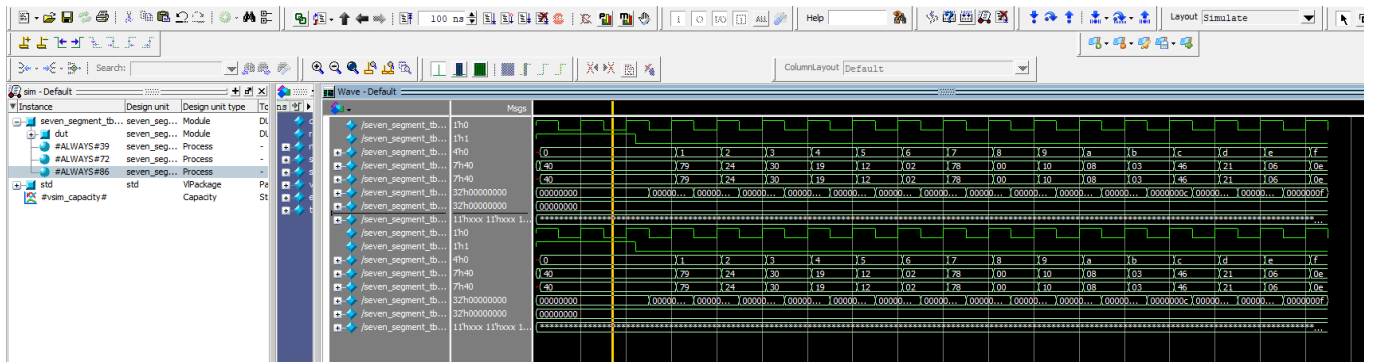


Figure 4: A screenshot of a QuestaSim simulation demonstrating the seven-segment display output.

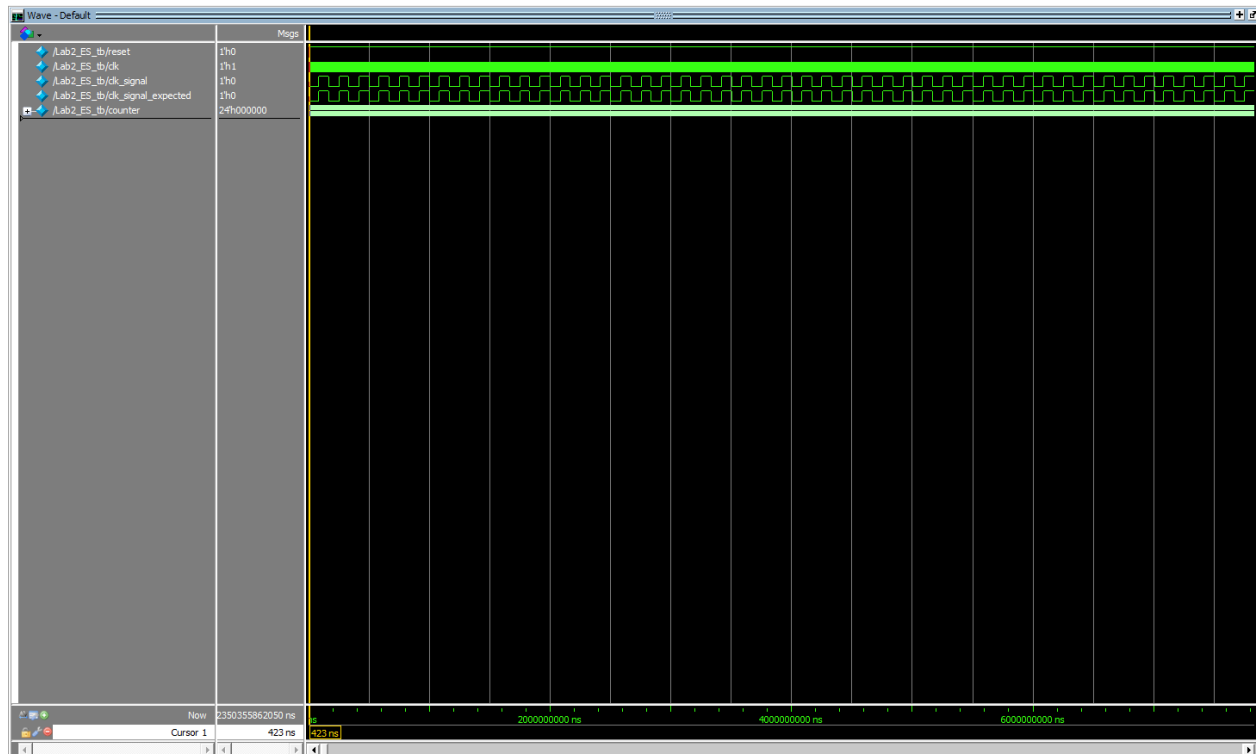


Figure 5: A screenshot of a QuestaSim simulation demonstrating the Lab2_es display output for clk and clk expected modules for multiplexing

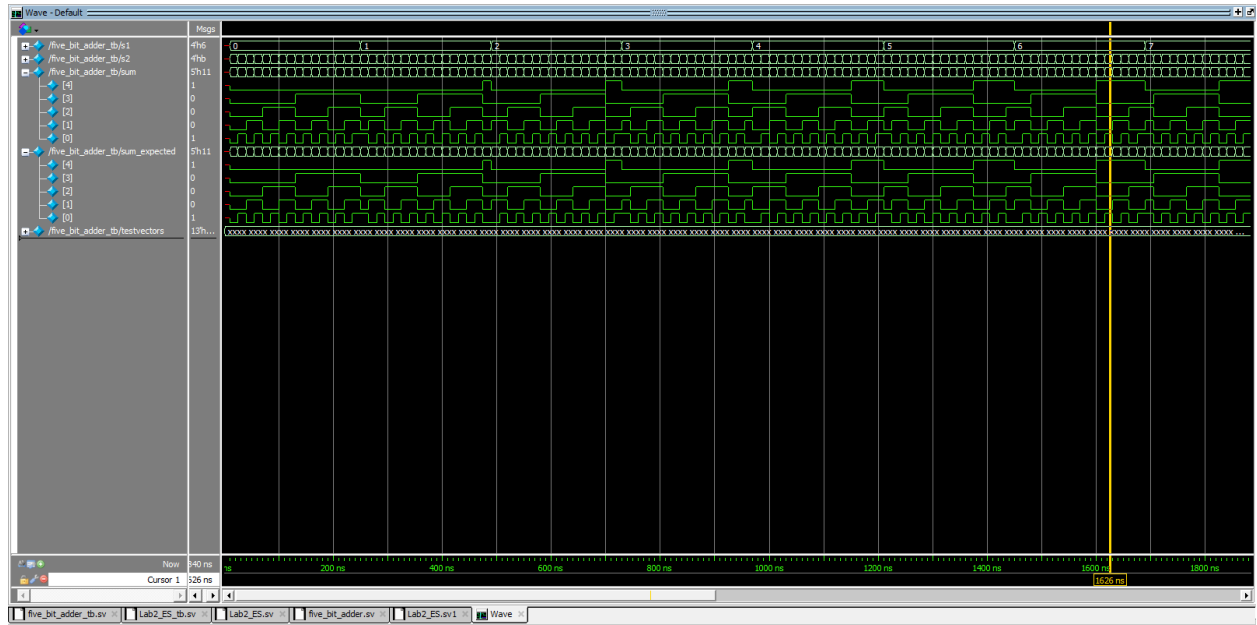


Figure 6: A screenshot of a QuestaSim simulation demonstrating the `five_bit_adder` module with test vectors

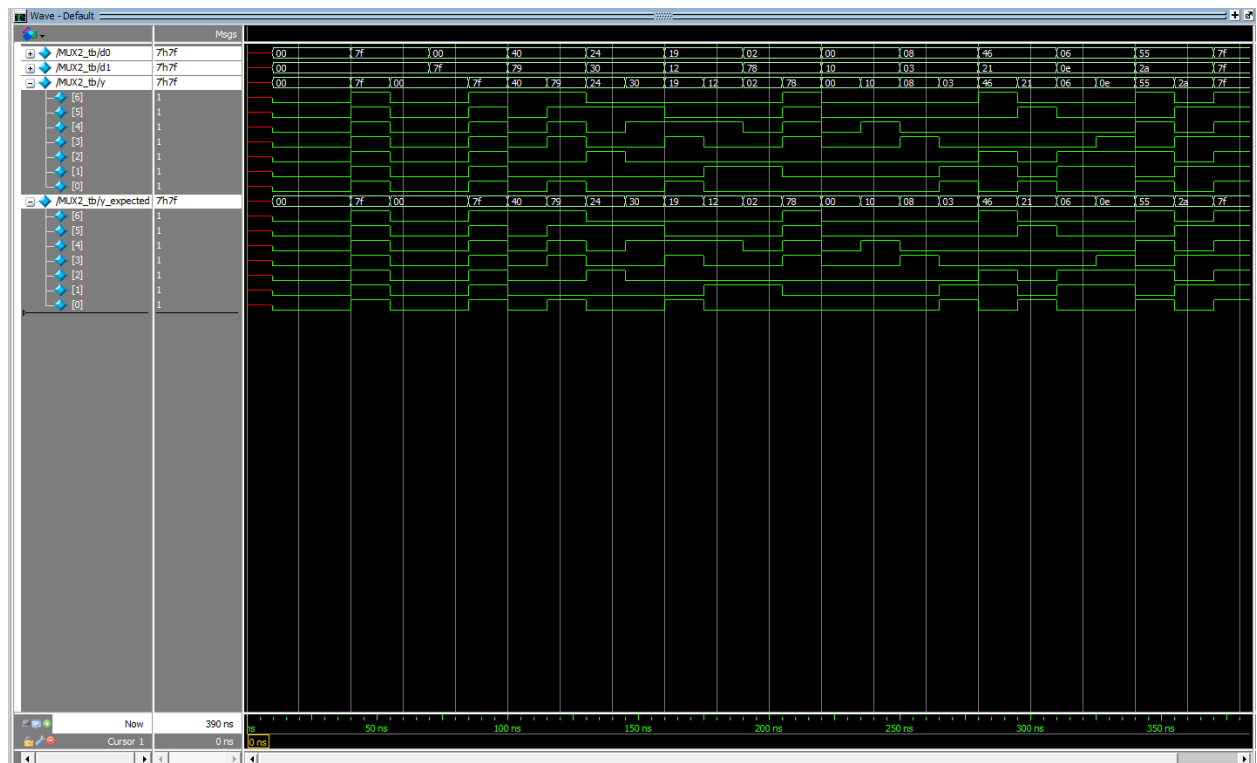


Figure 7: A screenshot of a QuestaSim simulation demonstrating the `MUX2` with test vectors

Simulation testbenches demonstrated that the seven-segment decoder produced correct segment outputs for hexadecimal digits 0 to F. Multiplexing correctly activated at 100Hz frequency each digit in rapid succession, producing a stable and flicker-free display in hardware tests. LEDs representing the sum of the two input digits illuminated correctly in all test cases.

Conclusion

The design met all lab requirements, including correct display of hexadecimal digits, proper LED logic, simulation verification, and safe current draw. Approximately **12 hours** were spent on coding, testing verification, and documentation.

AI Prototype Summary

Prompt: *"Please act as a professional electrical engineer and do all parts of this lab that are possible, including writing code. Please finish completely and do not skip steps or make mistakes."*

AI created a pretty decent module design and also implemented a test bench that looks pretty good. It would not work to simulate in questa due to HSOC however it is pretty efficient code generation. It used assign statements instead of implementation modules for the MUX2 and bit bit adder which was a design choice and is not future proof. In addition, AI could not place pin selection; however, it did generate a pin file for me. Furthermore, the test bench was not completed properly, and no test vectors were generated for me to use because it was included in the top module.

AI take my job score: 78%

E155 Lab 2 – Multiplexed Seven-Segment Display Starter Pack

This pack includes ready-to-use SystemVerilog for the FPGA controlling two multiplexed seven-segment displays, a simulation testbench, a constraints template for pin assignments, and repository scaffold to organize files consistently. All logic matches the lab's specification: common-anode 7-segment displays, active-low segment outputs, input switches with internal pull-ups, transistor-enabled digit anode switching, and sum output on LEDs.

Replace all `xx` with your initials or preferred identifier for consistency. Names are descriptive and aligned with lab specification.

Repo Layout

text

/

└─ README.md

└─ fpga

| └─ src

| | └─ lab2_mux_top.sv # Top-level module
controlling multiplexing

| | └─ seven_segment_decoder.sv # 4-bit hex to 7-seg
decoder (common-anode)

| | └─ clk_divider.sv # Parameterized clock
divider for multiplex freq

| | └─ mux_controller.sv # Digit multiplex control
module

| └─ sim

| | └─ lab2_mux_tb.sv # Questa/ModelSim
testbench file

| └─ radiant_project

| | └─ constraints_template.lpf # Pin location &
pull-up/down config

| └─ .gitignore

└─ .gitignore # Optional root ignore
file

README.md

E155 Lab 2 – Multiplexed Dual Seven-Segment Display

This repository contains:

- FPGA (Lattice iCE40 UP5K / UPduino v3.1)
 - `lab2_mux_top.sv`: Top level module managing multiplexing between two digits
 - `seven_segment_decoder.sv`: Hex (0–F) to 7-segment decoder for common-anode display with active-low segment outputs
 - `clk_divider.sv`: Parameterized clock divider to generate multiplex frequency (~1kHz)
 - `mux_controller.sv`: Provides digit enable signals switching between the two digits at multiplex frequency
 - `constraints_template.lpf`: Pin mapping with pull-ups enabled on inputs
 - `lab2_mux_tb.sv`: Testbench exercising multiplexing logic and segment decoding

Behavior summary

- Digit multiplexing: Alternates active common anode for two digits at ~1kHz to avoid flicker.
- Input: Two 4-bit hexadecimal numbers from DIP switches or input pins.
- Output: Multiplexed segment signals driving both digits; LEDs show the sum of inputs.
- Segment outputs: Active-low for common-anode displays.
- Hardware: Transistor-based current driver for common anode pins; 1kΩ resistors on each segment line.

Build & Program (Radiant)

1. Create a new Lattice Radiant project targeting iCE40UP5K or equivalent.
2. Add files from `fpga/src/`.
3. Assign pins according to `constraints_template.lpf`, ensuring `PULLMODE=UP` on input switch pins.
4. Synthesize, generate bitstream, and program FPGA via USB.

Simulation (Questa/ModelSim)

- Run `lab2_mux_tb.sv` testbench, which cycles through digit values and validates outputs.
 - The clock divider is adjusted in simulation for quick observable multiplexing.
-

Code Examples

lab2_mux_top.sv

verilog

```
module lab2_mux_top (
    input wire clk,                // 48 MHz clock input
    input wire [3:0] digit0_in,    // First hex digit input
    input wire [3:0] digit1_in,    // Second hex digit input
    output wire [6:0] seg,          // Common 7-seg segment signals
    (active-low)
    output reg [1:0] an,            // Digit enable signals (active
    low)
    output wire [4:0] led_sum       // Sum output on LEDs
);

    wire mux_sel; // 1-bit signal toggling at multiplex frequency

    // Instantiate clock divider for ~1kHz multiplex frequency
    clk_divider #(.HALF_CYCLES(24000)) clkdiv_inst (
        .clk(clk), .rst_n(1'b1), .tick(), .q(mux_sel)
    );
```

```

// Digit enable logic

always @(*) begin

    case (mux_sel)

        1'b0: an = 2'b10; // Activate Digit 0 (active low)

        1'b1: an = 2'b01; // Activate Digit 1

    endcase

end

// Instantiate seven-segment decoder

wire [6:0] seg0, seg1;

seven_segment_decoder decoder0 (.hex(digit0_in),
    .seg(seg0));

seven_segment_decoder decoder1 (.hex(digit1_in),
    .seg(seg1));

// Multiplex output segments to display either digit0 or
digit1

assign seg = mux_sel ? seg1 : seg0;

// Calculate sum of inputs for LED output (up to 0x1E)

assign led_sum = digit0_in + digit1_in;

endmodule

```

seven_segment_decoder.sv

verilog

```
module seven_segment_decoder (  
    input  wire [3:0] hex,  
    output reg  [6:0] seg      // Active-low segments [A..G] =  
seg[0..6]  
);  
    always @(*) begin  
        case (hex)  
            4'h0: seg = 7'b0000001;  
            4'h1: seg = 7'b1001111;  
            4'h2: seg = 7'b0010010;  
            4'h3: seg = 7'b0000110;  
            4'h4: seg = 7'b1001100;  
            4'h5: seg = 7'b0100100;  
            4'h6: seg = 7'b0100000;  
            4'h7: seg = 7'b0001111;  
            4'h8: seg = 7'b0000000;  
            4'h9: seg = 7'b0000100;  
            4'hA: seg = 7'b0001000;  
        endcase  
    end
```

```

        4'hB: seg = 7'b1100000;

        4'hC: seg = 7'b0110001;

        4'hD: seg = 7'b1000010;

        4'hE: seg = 7'b0110000;

        4'hF: seg = 7'b0111000;

        default: seg = 7'b1111111; // Blank display
    endcase

end

endmodule

```

clk_divider.sv

verilog

```

module clk_divider #(parameter int unsigned HALF_CYCLES =
24'd24000)(

    input wire clk,

    input wire rst_n,

    output reg tick,

    output reg q

);

    localparam WIDTH = $clog2(HALF_CYCLES);

    reg [WIDTH-1:0] count;

```

```
always @(posedge clk or negedge rst_n) begin

    if (!rst_n) begin

        count <= 0;

        q <= 0;

        tick <= 0;

    end else if (count == HALF_CYCLES - 1) begin

        count <= 0;

        q <= ~q;

        tick <= 1;

    end else begin

        count <= count + 1;

        tick <= 0;

    end

end

endmodule
```

lab2_mux_tb.sv

verilog

```
`timescale 1ns / 1ps
```

```
module lab2_mux_tb;
```

```
reg clk = 0;

always #10 clk = ~clk;  // 50 MHz clock for simulation


reg [3:0] digit0_in = 0;
reg [3:0] digit1_in = 0;


wire [6:0] seg;
wire [1:0] an;
wire [4:0] led_sum;


lab2_mux_top dut (
    .clk(clk),
    .digit0_in(digit0_in),
    .digit1_in(digit1_in),
    .seg(seg),
    .an(an),
    .led_sum(led_sum)
);


initial begin

    // Sweep digits from 0 to F
```



```
    for (integer i = 0; i < 16; i = i + 1) begin
        digit0_in = i[3:0];
        digit1_in = (15 - i)[3:0];
        #50000; // wait some cycles
    end
    $finish;
end
endmodule
```

constraints_template.lpf (Pin Assignments Example)

tcl

Switch inputs - enable internal pull-ups due to active low hardware

LOCATE COMP "digit0_in[0]" SITE "P12";

LOCATE COMP "digit0_in[1]" SITE "P13";

LOCATE COMP "digit0_in[2]" SITE "P14";

LOCATE COMP "digit0_in[3]" SITE "P15";

LOCATE COMP "digit1_in[0]" SITE "P16";

LOCATE COMP "digit1_in[1]" SITE "P17";

LOCATE COMP "digit1_in[2]" SITE "P18";

```
LOCATE COMP "digit1_in[3]" SITE "P19";
```

```
IOBUF PORT "digit0_in[0]" PULLMODE=UP;
```

```
IOBUF PORT "digit0_in[1]" PULLMODE=UP;
```

```
IOBUF PORT "digit0_in[2]" PULLMODE=UP;
```

```
IOBUF PORT "digit0_in[3]" PULLMODE=UP;
```

```
IOBUF PORT "digit1_in[0]" PULLMODE=UP;
```

```
IOBUF PORT "digit1_in[1]" PULLMODE=UP;
```

```
IOBUF PORT "digit1_in[2]" PULLMODE=UP;
```

```
IOBUF PORT "digit1_in[3]" PULLMODE=UP;
```

```
# Seven segment segment pins (A..G)
```

```
LOCATE COMP "seg[0]" SITE "P20";
```

```
LOCATE COMP "seg[1]" SITE "P21";
```

```
LOCATE COMP "seg[2]" SITE "P22";
```

```
LOCATE COMP "seg[3]" SITE "P23";
```

```
LOCATE COMP "seg[4]" SITE "P24";
```

```
LOCATE COMP "seg[5]" SITE "P25";
```

```
LOCATE COMP "seg[6]" SITE "P26";
```

Digit common anodes driven via transistor - active low enable

LOCATE COMP "an[0]" SITE "P27";

LOCATE COMP "an[1]" SITE "P28";

LEDs for sum output

LOCATE COMP "led_sum[0]" SITE "P29";

LOCATE COMP "led_sum[1]" SITE "P30";

LOCATE COMP "led_sum[2]" SITE "P31";

LOCATE COMP "led_sum[3]" SITE "P32";

LOCATE COMP "led_sum[4]" SITE "P33";

Notes

- Use 1kΩ resistors in series with each segment cathode to limit current.
- Common anode lines require transistor switching (e.g., 2N3906 PNP) due to high current.
- Internal pull-ups are enabled on switch inputs to work with active-low hardware buttons.
- Adjust clock divider parameter per clock speed for ~1 kHz multiplex frequency.
- Simulation testbench is simplified for faster testing cycles.

None

E155 Lab 2 – Multiplexed Seven-Segment Display Starter Pack

This pack includes ready-to-use SystemVerilog for the FPGA controlling two multiplexed seven-segment displays, a simulation testbench, a constraints template for pin assignments, and repository scaffold to organize files consistently. All logic matches the lab's specification: common-anode 7-segment displays, active-low segment outputs, input switches with internal pull-ups, transistor-enabled digit anode switching, and sum output on LEDs.

Replace all `xx` with your initials or preferred identifier for consistency. Names are descriptive and aligned with lab specification.

-

Repo Layout

text

```
/
├── README.md
├── fpga
│   ├── src
│   │   ├── lab2_mux_top.sv          # Top-level module
│   │   │   controlling multiplexing
│   │   ├── seven_segment_decoder.sv # 4-bit hex to 7-seg
│   │   │   decoder (common-anode)
│   │   ├── clk_divider.sv          # Parameterized clock
│   │   │   divider for multiplex freq
│   │   └── mux_controller.sv       # Digit multiplex
│   │       control module
│   └── sim
│       └── lab2_mux_tb.sv          # Questa/ModelSim
│           testbench file
```

```
|   |— radiant_project
|   |   |— constraints_template.lpf # Pin location &
pull-up/down config
|   |   |— .gitignore
|— .gitignore # Optional root
ignore file
```

-

README.md

E155 Lab 2 – Multiplexed Dual Seven-Segment Display

This repository contains:

FPGA (Lattice iCE40 UP5K / UPduino v3.1)

`lab2_mux_top.sv`: Top level module managing multiplexing between two digits

`seven_segment_decoder.sv`: Hex (0–F) to 7-segment decoder for common-anode display with active-low segment outputs

`clk_divider.sv`: Parameterized clock divider to generate multiplex frequency (~1kHz)

`mux_controller.sv`: Provides digit enable signals switching between the two digits at multiplex frequency

`constraints_template.lpf`: Pin mapping with pull-ups enabled on inputs

`lab2_mux_tb.sv`: Testbench exercising multiplexing logic and segment decoding

Behavior summary

Digit multiplexing: Alternates active common anode for two digits at ~1kHz to avoid flicker.

Input: Two 4-bit hexadecimal numbers from DIP switches or input pins.

Output: Multiplexed segment signals driving both digits; LEDs show the sum of inputs.

Segment outputs: Active-low for common-anode displays.

Hardware: Transistor-based current driver for common anode pins; 1kΩ resistors on each segment line.

Build & Program (Radiant)

Create a new Lattice Radiant project targeting iCE40UP5K or equivalent.

Add files from `fpga/src/`.

Assign pins according to `constraints_template.lpf`, ensuring `PULLMODE=UP` on input switch pins.

Synthesize, generate bitstream, and program FPGA via USB.

Simulation (Questa/ModelSim)

Run `lab2_mux_tb.sv` testbench, which cycles through digit values and validates outputs.

The clock divider is adjusted in simulation for quick observable multiplexing.

-

Code Examples

lab2_mux_top.sv

verilog

```
module lab2_mux_top (
    input wire clk,                // 48 MHz clock input
    input wire [3:0] digit0_in,    // First hex digit input
    input wire [3:0] digit1_in,    // Second hex digit input
```

```

        output wire [6:0] seg,          // Common 7-seg segment
signals (active-low)
        output reg [1:0] an,          // Digit enable signals
(active low)
        output wire [4:0] led_sum      // Sum output on LEDs
    );
    wire mux_sel; // 1-bit signal toggling at multiplex
frequency

    // Instantiate clock divider for ~1kHz multiplex
frequency
    clk_divider #(.HALF_CYCLES(24000)) clkdiv_inst (
        .clk(clk), .rst_n(1'b1), .tick(), .q(mux_sel)
    );

    // Digit enable logic
    always @(*) begin
        case (mux_sel)
            1'b0: an = 2'b10; // Activate Digit 0 (active
low)
            1'b1: an = 2'b01; // Activate Digit 1
        endcase
    end

    // Instantiate seven-segment decoder
    wire [6:0] seg0, seg1;
    seven_segment_decoder decoder0 (.hex(digit0_in),
        .seg(seg0));
    seven_segment_decoder decoder1 (.hex(digit1_in),
        .seg(seg1));

    // Multiplex output segments to display either digit0 or
digit1

```

```

    assign seg = mux_sel ? seg1 : seg0;

    // Calculate sum of inputs for LED output (up to 0x1E)
    assign led_sum = digit0_in + digit1_in;
endmodule

-

seven_segment_decoder.sv
verilog

module seven_segment_decoder (
    input  wire [3:0] hex,
    output reg  [6:0] seg      // Active-low segments [A..G]
= seg[0..6]
);
    always @(*) begin
        case (hex)
            4'h0: seg = 7'b0000001;
            4'h1: seg = 7'b1001111;
            4'h2: seg = 7'b0010010;
            4'h3: seg = 7'b0000110;
            4'h4: seg = 7'b1001100;
            4'h5: seg = 7'b0100100;
            4'h6: seg = 7'b0100000;
            4'h7: seg = 7'b0001111;
            4'h8: seg = 7'b0000000;
            4'h9: seg = 7'b0000100;
            4'hA: seg = 7'b0001000;
            4'hB: seg = 7'b1100000;
            4'hC: seg = 7'b0110001;
            4'hD: seg = 7'b1000010;
            4'hE: seg = 7'b0110000;

```



```

        4'hF: seg = 7'b0111000;
        default: seg = 7'b1111111; // Blank display
    endcase
end
endmodule

```

-

clk_divider.sv

verilog

```

module clk_divider #(parameter int unsigned HALF_CYCLES =
24'd24000)(
    input wire clk,
    input wire rst_n,
    output reg tick,
    output reg q
);
    localparam WIDTH = $clog2(HALF_CYCLES);
    reg [WIDTH-1:0] count;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            count <= 0;
            q <= 0;
            tick <= 0;
        end else if (count == HALF_CYCLES - 1) begin
            count <= 0;
            q <= ~q;
            tick <= 1;
        end else begin
            count <= count + 1;
            tick <= 0;
        end
    end
endmodule

```

```
        end
    end
endmodule
```

-

lab2_mux_tb.sv

verilog

```
`timescale 1ns / 1ps
module lab2_mux_tb;
    reg clk = 0;
    always #10 clk = ~clk; // 50 MHz clock for simulation

    reg [3:0] digit0_in = 0;
    reg [3:0] digit1_in = 0;

    wire [6:0] seg;
    wire [1:0] an;
    wire [4:0] led_sum;

    lab2_mux_top dut (
        .clk(clk),
        .digit0_in(digit0_in),
        .digit1_in(digit1_in),
        .seg(seg),
        .an(an),
        .led_sum(led_sum)
    );

    initial begin
        // Sweep digits from 0 to F
        for (integer i = 0; i < 16; i = i + 1) begin
```

```

        digit0_in = i[3:0];
        digit1_in = (15 - i)[3:0];
        #50000; // wait some cycles
    end
    $finish;
end
endmodule

```

-

constraints_template.lpf (Pin Assignments Example)

```
tcl
```

```
# Switch inputs - enable internal pull-ups due to active low hardware
```

```
LOCATE COMP "digit0_in[0]" SITE "P12";
LOCATE COMP "digit0_in[1]" SITE "P13";
LOCATE COMP "digit0_in[2]" SITE "P14";
LOCATE COMP "digit0_in[3]" SITE "P15";
```

```
LOCATE COMP "digit1_in[0]" SITE "P16";
LOCATE COMP "digit1_in[1]" SITE "P17";
LOCATE COMP "digit1_in[2]" SITE "P18";
LOCATE COMP "digit1_in[3]" SITE "P19";
```

```
IOBUF PORT "digit0_in[0]" PULLMODE=UP;
IOBUF PORT "digit0_in[1]" PULLMODE=UP;
IOBUF PORT "digit0_in[2]" PULLMODE=UP;
IOBUF PORT "digit0_in[3]" PULLMODE=UP;
```

```
IOBUF PORT "digit1_in[0]" PULLMODE=UP;
IOBUF PORT "digit1_in[1]" PULLMODE=UP;
```

```
IOBUF PORT "digit1_in[2]" PULLMODE=UP;  
IOBUF PORT "digit1_in[3]" PULLMODE=UP;
```

```
# Seven segment segment pins (A..G)
```

```
LOCATE COMP "seg[0]" SITE "P20";  
LOCATE COMP "seg[1]" SITE "P21";  
LOCATE COMP "seg[2]" SITE "P22";  
LOCATE COMP "seg[3]" SITE "P23";  
LOCATE COMP "seg[4]" SITE "P24";  
LOCATE COMP "seg[5]" SITE "P25";  
LOCATE COMP "seg[6]" SITE "P26";
```

```
# Digit common anodes driven via transistor - active low enable
```

```
LOCATE COMP "an[0]" SITE "P27";  
LOCATE COMP "an[1]" SITE "P28";
```

```
# LEDs for sum output
```

```
LOCATE COMP "led_sum[0]" SITE "P29";  
LOCATE COMP "led_sum[1]" SITE "P30";  
LOCATE COMP "led_sum[2]" SITE "P31";  
LOCATE COMP "led_sum[3]" SITE "P32";  
LOCATE COMP "led_sum[4]" SITE "P33";
```

-

Notes

Use 1k Ω resistors in series with each segment cathode to limit current.
Common anode lines require transistor switching (e.g., 2N3906 PNP) due to high current.
Internal pull-ups are enabled on switch inputs to work with active-low hardware buttons.

Adjust clock divider parameter per clock speed for ~1 kHz multiplex frequency.

Simulation testbench is simplified for faster testing cycles.