Emmett Stralka

Lab1 Report MircoP's // Aug 31st 2025

Introduction

In this lab, a design was implemented on the FPGA to verify the proper assembly of the V4 microP's board. In addition, this lab demonstrated the functionality of the onboard oscillator, switches, and LEDs that were surface-mounted and used to control a seven-segment display. The HSOSC from Lattice ICE40 UltraPlus primitive library was configured to gernate a 38MHz clock signal and was divided down to get a 2.4Hz blinking signal. Additionally, combinational logic (XOR and AND) was implemented to test LED and switch inputs.

Design and Testing Methodology

The development board was soldered, power rails were verified, and test code confirmed that the MCU and FPGA were functional. The System Verilog design included an oscillator, a clock divider, LED control logic, and a seven-segment decoder. The system was first verified with Questa simulation with a Test Vectors (.tv) file and then tested on hardware to ensure LEDs and display outputs matched specifications.

Technical Documentation:

The source code for the project can be found in the associated Github repository.

Lab1 Specific Developed Code can be found at: Github repository

Block Diagram

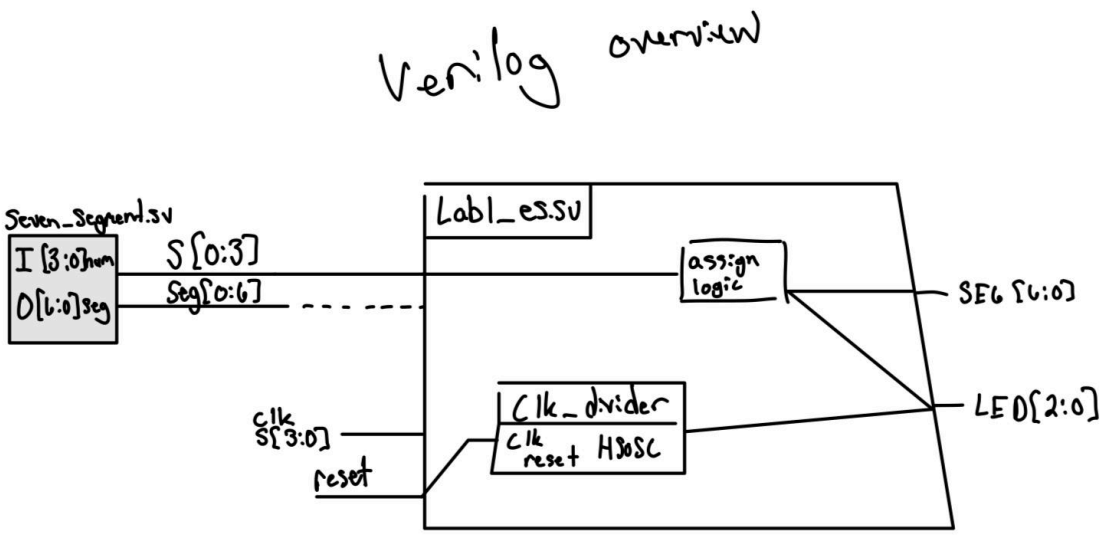

Figure 1: Block diagram of the System Verilog design.

The block diagram in Figure 1 demonstrates the overall architecture of the design. The top-level module `Lab1_es` includes two submodules: the high-speed oscillator block (`HSOSC`), the clock divider module (`clk_divider`), the seven_segment module (`seven_segment`).

Schematic

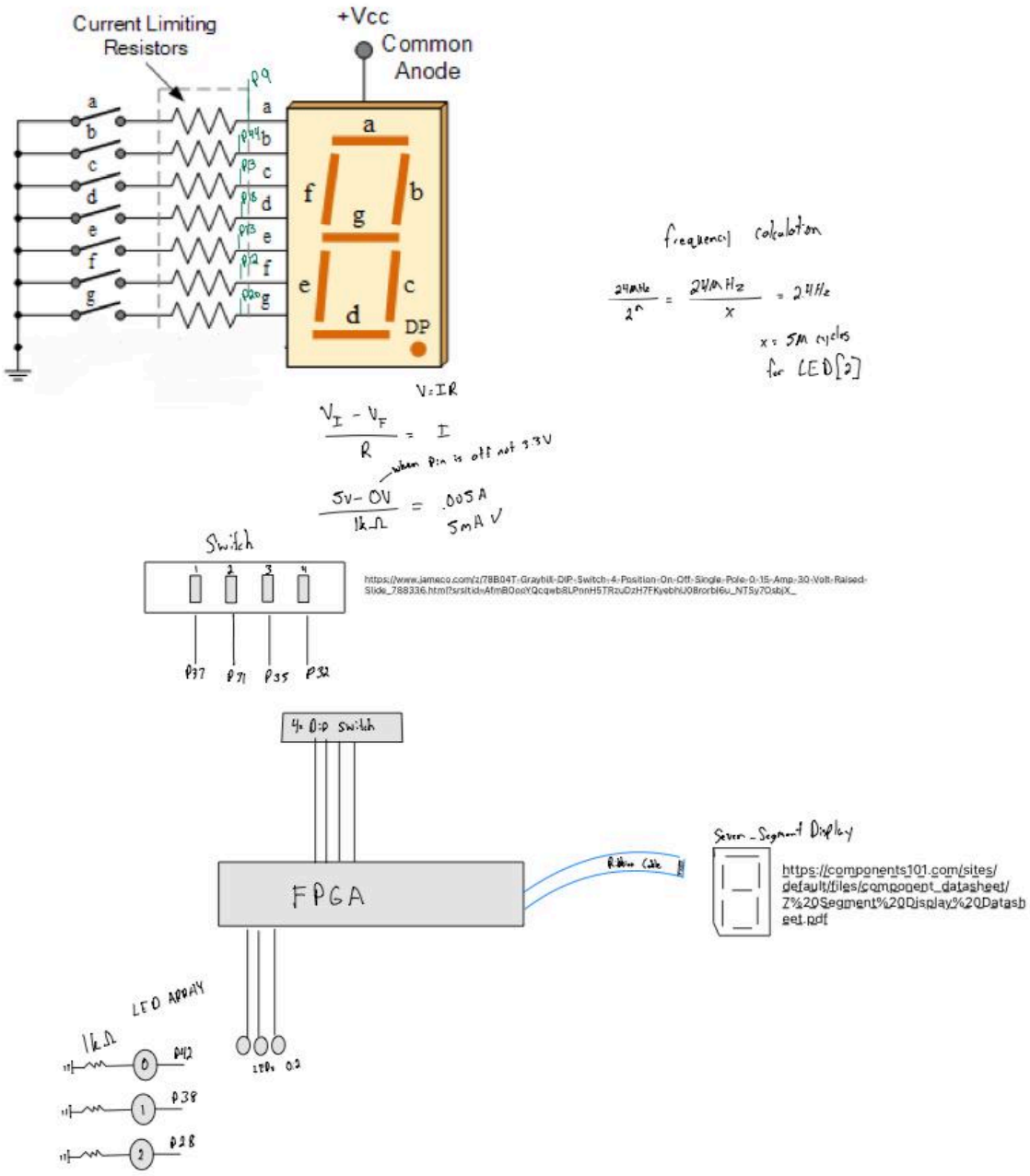

Figure 2: Schematic of the physical circuit.

Figure 2 shows the physical layout of the design. An internal 100 k pullup resistor was used to ensure the active low reset pin was not floating. The output LED was connected using a 1 k current-limiting resistor to ensure the output current (~2.6

mA) did not exceed the maximum output current of the FPGA I/O pins. The LED array of leds 0:2 were tied to pins 42, 28, 28. The seven-segment display was tied to 7 pin,s which was currently limited with a 1k Ohm resistor to ensure each segment got 5mA of current.

Results and Discussion

Testbench Simulation

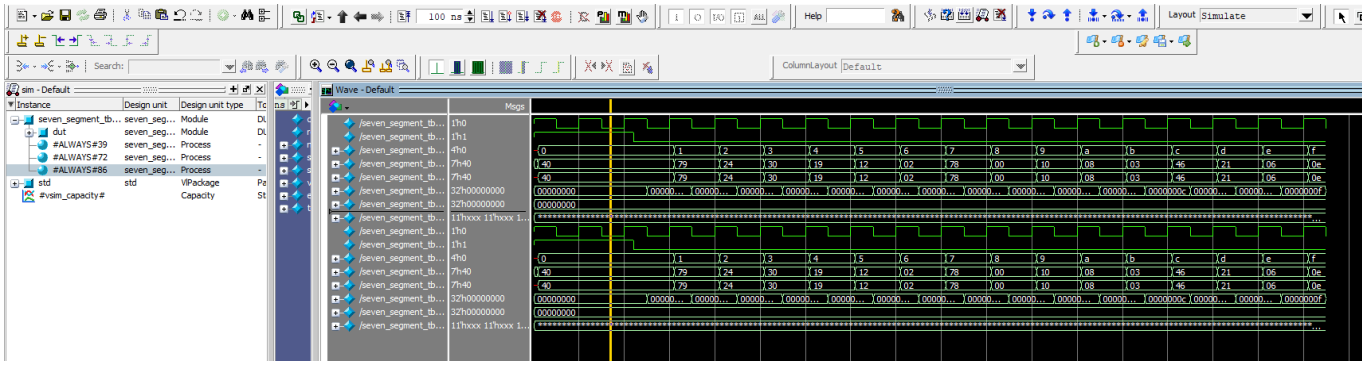

Figure 3: A screenshot of a QuestaSim simulation demonstrating the seven-segment display output.

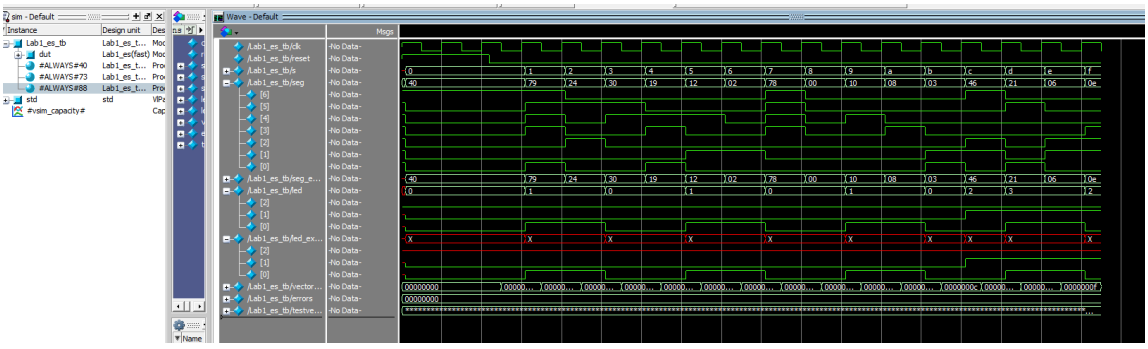

Figure 4: A screenshot of a QuestaSim simulation demonstrating the Lab1_es display output for the seven segment test vectors and the LED test vectors.

The design met all intended design objectives. Figure 3 and Figure 4 show a screenshot of the QuestaSim simulation. If a more precise output frequency was desired, a more sophisticated counter could be developed. The current design only allows for the clock to be divided by powers of two. The system passes all possible test vectors for the LED 0 and 1 and all 0x0-0xf displays on the seven-segment display.

Conclusion

The design met all lab requirements, including correct display of hexadecimal digits, proper LED logic, simulation verification, and safe current draw. Approximately 1**5 hours** were spent on assembly, coding, debugging, and documentation. Debugging a floating ground due to a soldering error on a capacitor and an LED not soldered correctly on the v4 Micro P's board was tedious to find.

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 separate module for the CLK divider, and also did not output the correct calculation for the clock divider. I do not think it followed the orival top.sv code so that could be the case for this 48 MHz reading. 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.

AI take my job score: 40%

E155 Lab 1 – Complete Starter Pack

This pack includes ready-to-use SystemVerilog for the FPGA, an STM32L432 MCU echo program for SEGGER/ GCC, simulation testbenches, constraints templates, and repo scaffolding. All logic matches the lab's spec (common-anode 7-segment, active-low inputs with internal pull-ups, LED truth tables, and a precise 2.4 Hz blinker from the 48 MHz HSOSC).

> Replace all `xx` with your initials if you like. I kept consistent, descriptive names.

Repo Layout

```
None
/
├── README.md
├── fpga
│   ├── src
│   │   ├── lab1_es_top.sv
│   │   ├── lab1_es_core.sv
│   │   ├── clk_divider.sv
│   │   └── seg7_hex_ca.sv
│   ├── sim
│   │   └── lab1_es_tb.sv
│   ├── radiant_project
│   │   └── constraints_template.lpf
│   └── .gitignore
├── mcu
```

```
|   ├── src
|   |   └── main.c
|   ├── segger_project  (create this in SES; just drop `main.c` in)
|   └── .gitignore
└── .gitignore (optional root ignore)
```

README.md

None

```
# E155 Lab 1 — FPGA and MCU Setup and Testing

This repository contains:
- **FPGA (Lattice iCE40 UP5K / UPduino v3.1)**
  - `lab1_es_top.sv`: Synthesis top using the internal 48 MHz HSOSC
  - `lab1_es_core.sv`: Core logic (takes `clk` and `s[3:0]`), easy to simulate
  - `clk_divider.sv`: Parameterized divider; default produces 2.4 Hz blink from 48 MHz
  - `seg7_hex_ca.sv`: Hex (0–F) 7-seg decoder for *common-anode* display, active-low outputs [A..G] = `seg[0]..seg[6]`
  - `constraints_template.lpf`: Pin/IO template; set PULLMODE=UP for `s[3:0]`
  - `lab1_es_tb.sv`: Questa/ModelSim testbench for quick verification

- **MCU (STM32L432KC / Nucleo-L432KC, SEGGER Embedded Studio GCC)**
  - `main.c`: Configures PA9 as input with pull-up and PA10 as push-pull output; echoes PA9 to PA10. Matches the lab wiring via J5/SW7 (PA9↔P25, PA10↔P23).

## Behavior summary
```

```
- **LED mapping** (after inverting active-low switches to
active-high internally):
  - `led[0] = s1 ^ s0` (ON for 01 and 10; OFF for 00 and 11)
  - `led[1] = s3 & s2` (only ON when both high)
  - `led[2]` blinks at **2.4 Hz** (exact: 48 MHz / (2×2.4) =
10,000,000 half-cycles)
- **7-segment**: Displays `s[3:0]` as a distinct hex digit `0…F`;
active-low outputs for a **common-anode** display, `seg[0]=A …
seg[6]=G`.

## Build & Program (Radiant)
1. Create a new Radiant project for **iCE40UP5K** (UPduino v3.1)
and add files in `fpga/src`.
2. Use the Device Constraint Editor to assign pins (see
`radiant_project/constraints_template.lpf`).
3. Set **PULLMODE=UP** on `s[3:0]` to use internal pull-ups
(switches/buttons are active-low on the board).
4. Synthesize, generate bitstream, and program over USB (onboard
programmer).

## Simulation (Questa/ModelSim)
- Use `lab1_es_tb.sv` (drives `lab1_es_core`) to verify LED truth
tables and 7-seg decode.
- The divider parameter is reduced in the TB for quick blinking.

## MCU Build (SEGGER Embedded Studio)
1. Create a new SES project for STM32L432KC (GCC). Add
`mcu/src/main.c`.
2. Ensure clock defaults to MSI (any default is fine for simple
GPIO).
3. Connect **PA9** (input with pull-up) to FPGA **P25** and
**PA10** (output) to FPGA **P23** using J5/SW7 as instructed.
4. Build/flash; PA10 echoes PA9, so you'll see D2 (PA10/P23) and
D7 (P38 via FPGA) blink along with D3 (PA9/P25).

## Notes
```

```
- The board’s slide/push switches are **active-low** with no
external pull-ups; we **enable pull-ups** in the FPGA and invert
`s` inside the logic.
- Use ~330–1k resistors in series with each 7-seg cathode line.
- Tie the chosen 7-seg anode (`VDD1` or `VDD2`) to **3.3 V**.
```

---

fpga/src/lab1_es_top.sv

None

```
// lab1_es_top.sv
// Author: Emmett Stralka (estralka@hmc.edu) – prepared pack
// Date: 2025-08-30
// Top-level for synthesis on iCE40UP5K. Uses the internal 48 MHz
HSOSC and
// instantiates the core logic that meets the Lab 1 spec.

module lab1_es_top (
    input  logic [3:0] s,       // SW6 DIP switches (active-low
on board)
    output logic [2:0] led,     // Connect to on-board LEDs
(e.g., D6/D7 + one header LED)
    output logic [6:0] seg      // Seven-seg segments [A..G] =
[0..6], active-low
);
    // Internal 48 MHz oscillator (iCE40 Technology Library)
    logic clk48;
    HSOSC #(.CLKHF_DIV("0b00")) // 48 MHz
    u_hsosc (
        .CLKHFPU(1'b1),      // Power up
        .CLKHFEN(1'b1),      // Enable
        .CLKHF(clk48)
    );

    // Core logic with explicit clock
```

```
    lab1_es_core u_core (
        .clk (clk48),
        .s   (s),
        .led (led),
        .seg (seg)
    );
endmodule
```

---

fpga/src/lab1_es_core.sv

None

```
// lab1_es_core.sv
// Author: Emmett Stralka (estralka@hmc.edu)
// Date: 2025-08-30
// Core logic: consumes a free-running clock and the 4 slide
switches.
// Implements LED truth tables and a 2.4 Hz blinker from 48 MHz,
and drives
// a common-anode 7-seg with active-low segment outputs.

module lab1_es_core (
    input  logic        clk,       // 48 MHz in deployment; any
clk for sim
    input  logic [3:0]  s,         // SW6 DIP switches
(active-low on board)
    output logic [2:0]  led,       // led[2]: 2.4 Hz blink;
led[1]: s3&s2; led[0]: s1^s0
    output logic [6:0]  seg        // Common-anode 7-seg,
active-low [A..G]=[0..6]
);
    // Convert active-low hardware inputs to active-high logic
    logic [3:0] sw;  // sw[i] == 1 means switch asserted
    assign sw = ~s;
```

```
    // LED[0] truth table: ON for 01 and 10 -> XOR of (S1,S0)
    assign led[0] = sw[1] ^ sw[0];

    // LED[1] truth table: ON only when S3=1 and S2=1
    assign led[1] = sw[3] & sw[2];

    // LED[2] blinks at 2.4 Hz from 48 MHz: half-period counts =
10,000,000
    logic blink2p4;
    clk_divider #(
        .HALF_CYCLES(24'd10_000_000)  // exact for 48 MHz -> 2.4
Hz
    ) u_div (
        .clk   (clk),
        .rst_n (1'b1),
        .tick  (),
        .q     (blink2p4)
    );
    assign led[2] = blink2p4;

    // 7-seg hex decoder (active-low outputs for common-anode)
    seg7_hex_ca u_seven (
        .hex  (sw),   // display the asserted (active-high)
switch value
        .seg  (seg)
    );
endmodule
```

---

fpga/src/clk_divider.sv

None

```
// clk_divider.sv
// Author: Emmett Stralka (estralka@hmc.edu)
// Date: 2025-08-30
```

```
// Parameterized clock divider that toggles q every HALF_CYCLES
clocks.
// Emits a one-cycle tick at the toggle moment as well.

module clk_divider #(
    parameter int unsigned HALF_CYCLES = 24'd10_000_000  //
default: 2.4 Hz from 48 MHz
) (
    input  logic clk,
    input  logic rst_n,
    output logic tick,
    output logic q
);
    localparam int WIDTH = $clog2(HALF_CYCLES);
    logic [WIDTH-1:0] cnt;

    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            cnt  <= '0;
            q    <= 1'b0;
            tick <= 1'b0;
        end else begin
            if (cnt == HALF_CYCLES-1) begin
                cnt  <= '0;
                q    <= ~q;
                tick <= 1'b1;
            end else begin
                cnt  <= cnt + 1'b1;
                tick <= 1'b0;
            end
        end
    end
endmodule
```

---

fpga/src/seg7_hex_ca.sv

None

```
// seg7_hex_ca.sv
// Author: Emmett Stralka (estralka@hmc.edu)
// Date: 2025-08-30
// Hex (0..F) to 7-seg decoder for a **common-anode** display.
// seg[0]=A ... seg[6]=G, **active-low** outputs (0 lights a
segment).

module seg7_hex_ca (
    input  logic [3:0] hex,
    output logic [6:0] seg
);
    // Start with active-high segment patterns, then invert for
active-low outputs.
    //        gfedcba (human order) -> we use [A..G] = [0..6]
    // Distinguish b vs 8 (b = 0b1111000 segments c,d,e,f,g?
Wait—here we define hex digits.)
    logic [6:0] seg_hi;  // 1 means segment ON (before inversion)

    always_comb begin
        unique case (hex)
            4'h0: seg_hi = 7'b1111110; // 0 -> A B C D E F on, G
off
            4'h1: seg_hi = 7'b0110000; // 1 -> B C
            4'h2: seg_hi = 7'b1101101; // 2 -> A B D E G
            4'h3: seg_hi = 7'b1111001; // 3 -> A B C D G
            4'h4: seg_hi = 7'b0110011; // 4 -> B C F G
            4'h5: seg_hi = 7'b1011011; // 5 -> A C D F G
            4'h6: seg_hi = 7'b1011111; // 6 -> A C D E F G
            4'h7: seg_hi = 7'b1110000; // 7 -> A B C
            4'h8: seg_hi = 7'b1111111; // 8 -> A B C D E F G
            4'h9: seg_hi = 7'b1111011; // 9 -> A B C D F G
            4'hA: seg_hi = 7'b1110111; // A -> A B C E F G
            4'hB: seg_hi = 7'b0011111; // b -> C D E F G
(lowercase b, distinct from 8)
            4'hC: seg_hi = 7'b1001110; // C -> A D E F
```

```
            4'hD: seg_hi = 7'b0111101; // d -> B C D E G
(lowercase d)
            4'hE: seg_hi = 7'b1001111; // E -> A D E F G
            4'hF: seg_hi = 7'b1000111; // F -> A E F G
            default: seg_hi = 7'b0000000;
        endcase
    end

    // Map seg_hi[A..G] into seg[0..6] and invert for active-low
    // seg_hi bits order here is [A..G] = [6:0] above; we
produced in A..G order already
    assign seg = ~seg_hi;
endmodule
```

---

fpga/sim/lab1_es_tb.sv

None

```
// lab1_es_tb.sv — Questa/ModelSim testbench for lab1_es_core
// Drives a fast clock and steps through all s[3:0] values.

`timescale 1ns/1ps
module lab1_es_tb;
    logic clk;
    logic [3:0] s;       // active-low switches as in hardware
    logic [2:0] led;
    logic [6:0] seg;

    // 20 MHz test clock (50 ns period) — arbitrary; fast enough
for sim
    initial clk = 1'b0;
    always #25 clk = ~clk;

    // DUT: use a much smaller divider to observe blinking in sim
    lab1_es_core dut (
```

```
        .clk (clk),
        .s   (s),
        .led (led),
        .seg (seg)
    );
    // Override parameter via defparam (or edit clk_divider
default for sim)
    defparam dut.u_div.HALF_CYCLES = 10;  // toggle every 10
cycles in simulation

    // Stimulus: sweep s from 0 to 15 as if switches were set
(active-low)
    integer i;
    initial begin
        s = 4'b1111;  // all OFF (active-low inputs idle high)
        #200;
        for (i = 0; i < 16; i = i + 1) begin
            // represent hex value i on active-low switches:
ON=0, OFF=1
            s = ~i[3:0];
            #1000; // 1 us per code; inspect seg/led in waves
        end
        #5000;
        $finish;
    end
endmodule
```

---

fpga/radiant_project/constraints_template.lpf

None

```
# constraints_template.lpf – Lattice Radiant LPF template for
UP5K
# Fill in exact pin sites per your board's schematic/silkscreen.
# Set pull-ups on switch inputs (active-low on the board).
```

```
# Example known on-board LED connections (verify on your PCB
revision):
# D6 (blue)  <- P42, D7 (blue) <- P38. Map led[?] as you prefer.

# Assign LED pins
LOCATE COMP "led[2]" SITE "P38";   # example: blink LED on D7
(P38)
LOCATE COMP "led[1]" SITE "P42";   # example: LED on D6 (P42)
# LOCATE COMP "led[0]" SITE "<choose a free I/O>";

# Assign switch pins s[3:0] to SW6 signals (check board silk /
schematic)
# Example placeholders (REPLACE with real pins):
# LOCATE COMP "s[0]" SITE "Pxx";
# LOCATE COMP "s[1]" SITE "Pxx";
# LOCATE COMP "s[2]" SITE "Pxx";
# LOCATE COMP "s[3]" SITE "Pxx";

# Enable internal pull-ups on inputs (critical for active-low
switches)
IOBUF PORT "s[0]" PULLMODE=UP;
IOBUF PORT "s[1]" PULLMODE=UP;
IOBUF PORT "s[2]" PULLMODE=UP;
IOBUF PORT "s[3]" PULLMODE=UP;

# 7-seg segment outputs (common-anode): choose 7 free pins; add
series resistors!
# LOCATE COMP "seg[0]" SITE "Pxx";  # A
# LOCATE COMP "seg[1]" SITE "Pxx";  # B
# LOCATE COMP "seg[2]" SITE "Pxx";  # C
# LOCATE COMP "seg[3]" SITE "Pxx";  # D
# LOCATE COMP "seg[4]" SITE "Pxx";  # E
# LOCATE COMP "seg[5]" SITE "Pxx";  # F
# LOCATE COMP "seg[6]" SITE "Pxx";  # G
```

fpga/.gitignore

None

```
# Lattice Radiant files
*.html
impl*/
*.xml
.build_status
.run_manager.ini
.recovery
.spread_sheet.ini
.spreadsheet_view.ini
*.dir/
*.log
*.tcl
*.ccl
*.srp
*.dmp
._Real_._Math_.vhd
```

---

mcu/src/main.c

C/C++

```
// main.c — STM32L432KC (Nucleo-L432KC), SEGGER Embedded Studio (GCC)
// Echo PA9 (input with pull-up) to PA10 (push-pull output).
// Board wiring per lab: J5/SW7 routes PA9 <-> FPGA P25; PA10 <-> FPGA P23.
// This minimal bare-metal setup uses direct register access (no HAL needed).

#include <stdint.h>

// Minimal device defs (addresses from RM0394 / STM32L4 ref; SES projects include CMSIS headers)
```

```
#define PERIPH_BASE       0x40000000UL
#define AHB2PERIPH_BASE   (PERIPH_BASE + 0x08000000UL)
#define GPIOA_BASE        (AHB2PERIPH_BASE + 0x0000UL)
#define RCC_BASE          (0x40021000UL)

#define RCC_AHB2ENR       (*(volatile uint32_t *)(RCC_BASE +
0x4C))
#define GPIOA_MODER       (*(volatile uint32_t *)(GPIOA_BASE +
0x00))
#define GPIOA_PUPDR       (*(volatile uint32_t *)(GPIOA_BASE +
0x0C))
#define GPIOA_IDR         (*(volatile uint32_t *)(GPIOA_BASE +
0x10))
#define GPIOA_OTYPER      (*(volatile uint32_t *)(GPIOA_BASE +
0x04))
#define GPIOA_OSPEEDR     (*(volatile uint32_t *)(GPIOA_BASE +
0x08))
#define GPIOA_ODR         (*(volatile uint32_t *)(GPIOA_BASE +
0x14))

static inline void delay(volatile uint32_t d) { while (d--)
__asm__("nop"); }

int main(void) {
    // Enable GPIOA clock
    RCC_AHB2ENR |= (1u << 0);

    // PA9: input, pull-up
    // MODER: 00=input for pin 9
    GPIOA_MODER &= ~(0x3u << (9 * 2));
    // PUPDR: 01=pull-up for pin 9
    GPIOA_PUPDR &= ~(0x3u << (9 * 2));
    GPIOA_PUPDR |=  (0x1u << (9 * 2));

    // PA10: general purpose output push-pull, medium speed, no
pull
```

```
    // MODER: 01=output for pin 10
    GPIOA_MODER &= ~(0x3u << (10 * 2));
    GPIOA_MODER |=  (0x1u << (10 * 2));
    // OTYPER: 0=push-pull for pin 10
    GPIOA_OTYPER &= ~(1u << 10);
    // OSPEEDR: 01=medium speed for pin 10 (optional)
    GPIOA_OSPEEDR &= ~(0x3u << (10 * 2));
    GPIOA_OSPEEDR |=  (0x1u << (10 * 2));
    // PUPDR: 00=no pull for pin 10
    GPIOA_PUPDR &= ~(0x3u << (10 * 2));

    while (1) {
        uint32_t in = (GPIOA_IDR >> 9) & 1u;  // read PA9
        if (in) {
            // PA9 reads high (switch open / LED off on FPGA
input) -> drive PA10 high
            GPIOA_ODR |= (1u << 10);
        } else {
            // PA9 low (FPGA is toggling low) -> drive PA10 low
            GPIOA_ODR &= ~(1u << 10);
        }
        // small delay to reduce bus churn (optional)
        delay(100);
    }
}
```

---

mcu/.gitignore

None

```
# Segger Embedded Studio
**/Output/
**/Debug/
*.emSession
*.jlink
```

Top-Level .gitignore (optional)

```
None
.DS_Store
*.zip
```

Wiring & Bring-Up Checklist

- **Power rails**: Verify +5 V and +3.3 V are correct before installing boards.
- **Jumpers**: Short **J3 (FPGA_+5V)** and **J4 (MCU_+5V)** when powering from either USB.
- **J5 / SW7 mapping**: Route **PA9** ↔ **P25** and **PA10** ↔ **P23** per silkscreen. Confirm with a DMM.
- **Pull-ups**: In Radiant Device Constraint Editor, set `PULLMODE=UP` for `s[3:0]`.
- **7-segment**: Tie `VDD1` or `VDD2` to **3.3 V**; each segment line goes FPGA->resistor(330–1k)->segment cathode.
- **Programming order**: Program FPGA first (see D3/D6/D7 blink), then flash MCU and confirm echo behavior.

Simulation Notes

- If you simulate `HSOSC`, add the iCE40 tech library in Questa (`-L` option). Alternatively, simulate `lab1_es_core` as done here.
- The TB reduces the divider to show visible blinking in a short run.

Troubleshooting

- **Floating inputs**: If LEDs behave erratically, re-check `PULLMODE=UP` on all `s[3:0]` pins.
- **Shared pins**: If an FPGA pin is odd, confirm the MCU isn't driving it; flash an empty MCU program.
- **Solder bridges**: Measure resistance between adjacent header pins.
- **7-seg too dim/bright**: Adjust series resistors (typ. 330–680 Ω at 3.3 V).

Attribution

Prepared for HMC E155 Lab 1. Replace contact info/comments with your details as required by your submission guidelines.

None