EE405(B), Fall 2022

Electronics Design Lab. <Advanced Digital System Design>

- Student Names: Hyemin Lee
- Student IDs: 20190533
- LAB Project Number: Lab0, Lab1

1. Goal / Specification

A. Lab 0

i. The goal of this lab project is to setup the FPGA development environment. The development environment will be constructed using FPGA design tools (Xilinx Vivado & Xilinx Vitis) and FPGA evaluation board (Xilinx Arty A7-35T).

ii. Field Programmable Gate Array (FGPA) is VLSI circuit that consists of programmable logic blocks, IO blocks, and interconnections. A board named Xilinx Arty A7-35T will be used for all the projects in this course.

iii. Vivado is hardware design tool for synthesis and implementation on Xilinx's FPGAs. It also supports simulation and analysis of HDL designs.

iv. Vitis is software design tool for creating embedded software applications on Xilinx's microprocessors. It also supports C/C++ based design, debug, and performance analysis.

B. Lab 1

i. The goal of this lab project is to learn whole process of FPGA programming with a simple example of LED display.

ii. The first problem (Problem 1A) was to implement an LED blinking system using buttons (BTN0 and BTN1) and LED (LD4). The BTN0 was reset button and the BTN1 was enabling button for LD4.

iii. The second problem (Problem 1B) was to implement an LED counter system using buttons (BTN0, 1, 2) and RGB LEDs (LD0, 1, 2, 3). The BTN0 was reset button. The BTN1 was used to increase the number of the counter, and BTN2 was used to change the RGB LEDs color among red, green, and blue.

2. Architecture / Design

A. Description

i. Problem 1A: Implement an LED blinking system

1. Input ports: i_clk (clock), i_rst (BTN0), i_en (BTN1)
2. Output ports: o_led (LD4)
3. A single flip-flop named *led* was used and directly assigned to o_led.

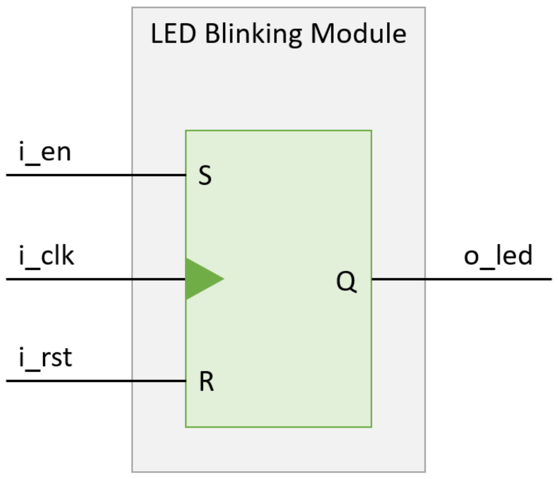
4. If i_rst is 1, *led* becomes 0, and if i_en is 1, *led* becomes 1. Nothing happens when i_st or i_en is 0.

ii. Problem 1B: Implement an LED counter system

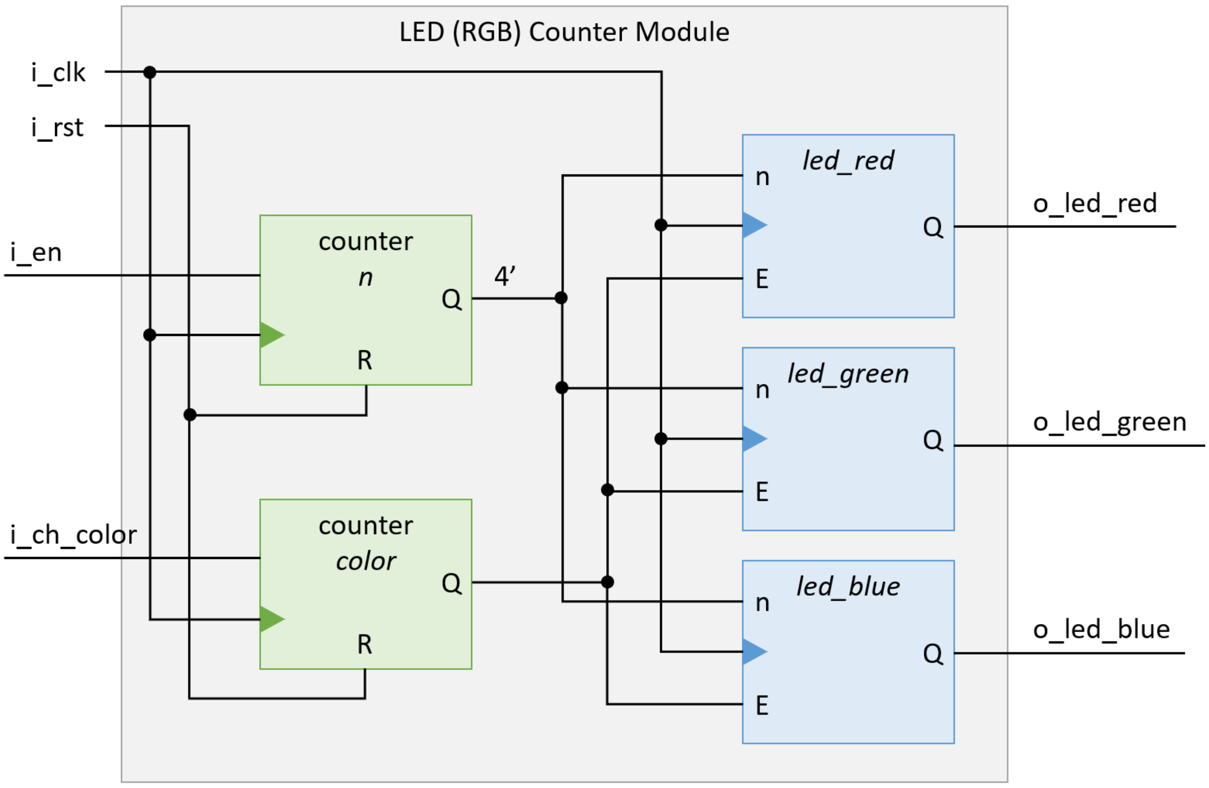
1. Input ports: i_clk (clock), i_rst (BTN0), i_en (BTN1), i_ch_color (BTN2)
2. Output ports: o_led_red[3:0], o_led_green[3:0], o_led_blue[3:0]
3. MSB 0 of each output port was connected to red, green, and blue of LD0. MSB 1 of each output port was connected to red, green, and blue of LD1. In the same manner, MSB 2 and MSB 3 were connected to LD2 and LD3.
4. Five flip-flops named *n*, *color*, *led_red*, *led_green*, and *led_blue* were used. *led_red*, *led_green*, and *led_blue* were directly assigned to o_led_red, o_led_green, and o_led_blue, respectively.
5. All the flip-flops were updated to the next value synchronously at positive edge of every clock. The next value was calculated through combinational logic in between clocks.
6. Nothing happened if i_en or i_ch_color is 0.
7. If i_en is 1, the value of *n* increased by 1. But reset to 0 if the value exceeds 15, the maximum number counter can calculate.
8. If i_ch_color is 1, the value of *color* increased by 1. But reset to 0 if the value exceeds 2. This was because there are only three colors: red(0), green(1), and blue(2).
9. According to the value of *n* and *color*, *led_red*, *led_green*, and *led_blue* was updated. If *color* was 0, *led_red* updated to *n* while *led_green* and *led_blue* became 0. If *color* was 1, *led_green* updated to *n* while *led_red* and *led_blue* became 0. Lastly, if *color* was 2, *led_blue* updated to *n* while *led_red* and *led_green* became 0.

B. Hardware Block Diagram (Use Visio, at least Power point, etc., Don't draw by hand)

i. Problem 1A



ii. Problem 1B

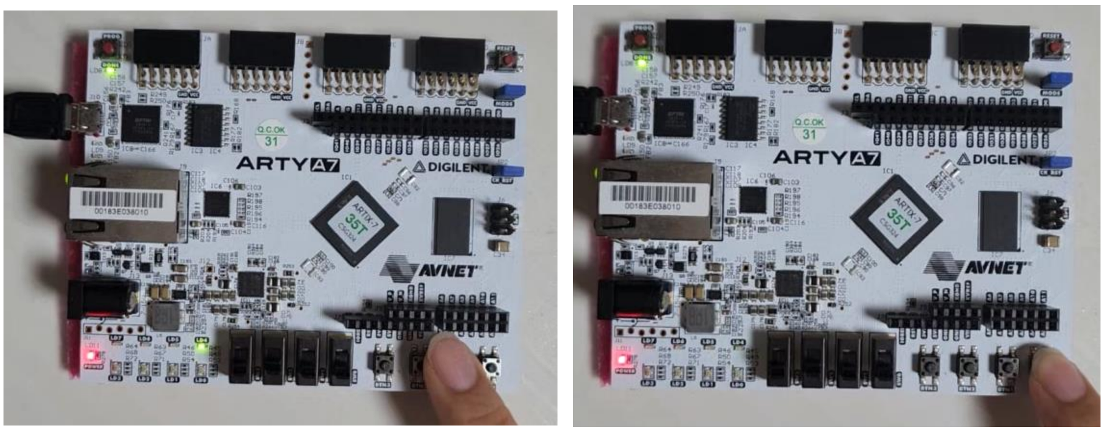


3. Experiment Results

A. Demo Picture

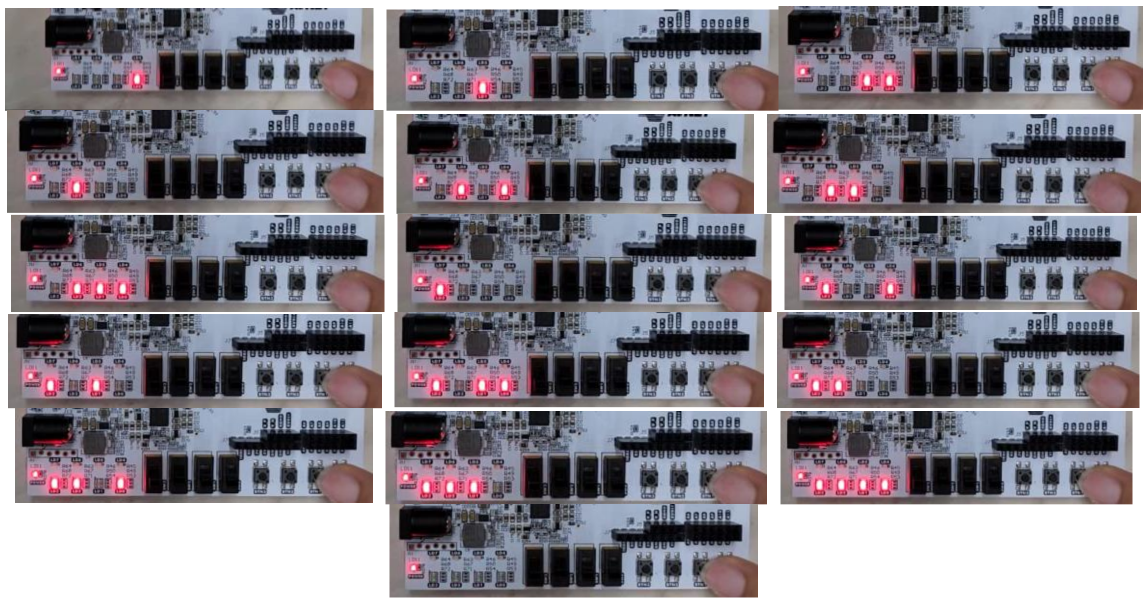
i. Problem 1A

1. (Left) Press BT1: Turn on LD4
2. (Right) Press BT0: Turn off LD4

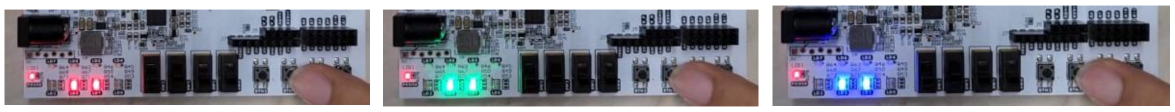


ii. Problem 1B

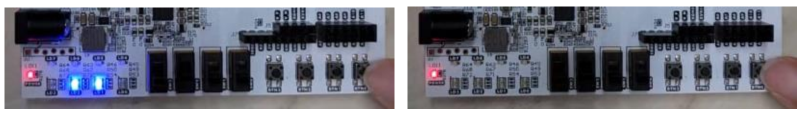
1. Press BT1: Increase the counter number from 0000_2 to 1111_2.



2. Press BT2: Change color in order of red, green, and blue.



3. Press BT0: Reset the counter.



4. Discussion

A. Explain briefly about the Vivado IP and Microblaze.

  i. IP stands for Intellectual Property. Xilinx provides numerous libraries of pre-validated IP cores to help users get to market faster. In Vivado, there is IP integrator that allows users to easily interconnect complex IP blocks.

  ii. MicroBlaze is Xilinx's 32-bit RISC soft processor core, optimized for embedded applications on Xilinx devices. It is implemented entirely in the general-purpose memory and logic fabric of Xilinx FPGAs which meets the requirements of many diverse applications including Industrial, Medical, Automotive, Consumer, and Communications markets. It includes three preset configurations – Microcontroller, Real-Time Processor, and Application Processor – to help users quickly deploy their application.

B. In Lab0, we synthesized the Microblaze to the Arty A7 board. Explain why we did it and how we can utilize the Microblaze.

  i. We do synthesis to convert the schematic design to FPGA-specific LUTs and other primitive components. It creates Xilinx specific netlist files that is required during implementation process.

ii. Using Microblaze, we can accelerate the development of const-sensitive, high-volume applications that traditionally required one or more microcontrollers. Especially, since Arty A7 board is one of the Xilinx's Cost-Optimized Portfolio FPGAs, it offers advances in tool suite and FPGA platform to help simplify development effort and minimize system budgets. Microblaze can be utilized in various fields such as industrial, communications infrastructure, medical device, automotive, and consumer markets.

C. Explain the difference between Verilog and SystemVerilog (in terms of usage, syntax, etc.).

i. The main difference between Verilog and SystemVerilog is that Verilog is a Hardware Description Language (HDL), while SystemVerilog is a HDL and Hardware Verification Language (HVL) based on Verilog. HDL is used to model electronic systems, and HVL is used to verify the electronic circuits written in a HDL. Overall, SystemVerilog is a superset of Verilog.

ii. Verilog only supports structed paradigm, while SystemVerilog supports structured and object-oriented paradigms. Also, Verilog only supports wire and reg data types, while SystemVerilog supports various data types such as emun, struct, union, class and string in addition to wire and reg.

iii. The file extension of Verilog is .v or .vh, while that of SystemVerilog is .sv or .svh. Verilog has a single always block to implement combinational and sequential logic, while SystemVerliog has always_comb, always_ff, and always_latch procedural blocks to implement.

D. Explain briefly about the FPGA implementation process.

i. Implementation is the process of determining which physical resources on the FPGA to program with which logic, and how to connect (route) them. This produces the bitstream that is loaded onto the device for FPGA programming.

E. In the LED counter system, there might be a problem that the counter increases several times even if the button is pressed once. Why does this problem occur, and how did you solve this problem?

i. Since uncountable number of clock passes while pressing the button, if you only write the code like 'if (i_en == 1) n = n + 1', the counting number will increase countlessly. Therefore, I proposed a new variable named *pressing_i_en* and *pressing_i_ch_color*. These variables act as flag and indicates whether the button is holding. If the button is released, the flag resets its value and wait for another pressing.

ii. However, although I introduced *pressing_i_en* and *pressing_i_ch_color*, the counter often increased several times as mentioned in the question. This was due to vibration of switch. In order to solve this problem, switch debouncing technique should be implemented. The technique saves the old value of the switch and maintains the value for certain period of times so that the flickering of input does not directly reflected as result. I'll implement this technique from the next project.

5. Appendix: Include your code here

A. Problem 1A

```
module LED(
   // input ports
   input wire      i_clk   ,
   input wire      i_rst   ,
   input wire      i_en    ,
   output wire     o_led
   );

   // RTL Code
   reg led;
   initial led = 0;

   // assign register to wire
   assign o_led = led;

   // sequential logic : operate with clock
   always @ (posedge i_clk) begin
      // Reset
      if (i_rst) begin
         led <= 0;
      end
      // Run
      else begin
         if (i_en) begin
            led <= 1;
         end
      end
   end

endmodule
```

B. Problem 1B

```
module LED(
   // input ports
   input wire     i_clk   ,
   input wire     i_rst   ,
   input wire     i_en    ,
   input wire     i_ch_color,
   output wire    [3:0] o_led_red,
   output wire    [3:0] o_led_green,
   output wire    [3:0] o_led_blue
   );

   // RTL Code
   reg [3:0] n, nxt_n;
   reg [3:0] led_red, led_green, led_blue;
   reg [3:0] nxt_led_red, nxt_led_green, nxt_led_blue;
   reg [1:0] color, nxt_color;   // 0:red, 1:green, 2:blue
   reg pressing_i_en, pressing_i_ch_color;

   // assign register to wire
   assign o_led_red = led_red;
   assign o_led_green = led_green;
   assign o_led_blue = led_blue;



   // sequential logic : operate with clock
   always_ff @ (posedge i_clk) begin
      // Update LED
      n <= nxt_n;
      color <= nxt_color;
      led_red <= nxt_led_red;
      led_green <= nxt_led_green;
      led_blue <= nxt_led_blue;
   end

   // combinational logic
   always_comb begin
      if (i_rst) begin
```

```
            nxt_n = 0;
            nxt_led_red = 0;
            nxt_led_green = 0;
            nxt_led_blue = 0;
            nxt_color = 0;
        end else begin
            // i_en: increase number
            if (i_en) begin                 // button is pressed
                if (~pressing_i_en) begin   // if this is the first clock while button
is pressed, increase the number
                    if (n == 4'b1111) nxt_n = 0;
                    else nxt_n = n + 1;
                    pressing_i_en = 1;
                end
            end else begin                   // if the button is released, reset
'pressing_i_en'
                if (pressing_i_en) pressing_i_en = 0;
            end

            // i_ch_color: change color
            if (i_ch_color) begin
                if (~pressing_i_ch_color) begin
                    if (color == 2'b10) nxt_color = 0;
                    else nxt_color = color + 1;
                    pressing_i_ch_color = 1;
                end
            end else begin
                if (pressing_i_ch_color) pressing_i_ch_color = 0;
            end

            // Update the next LED color
            case (color)
                2'b00: begin        // red
                    nxt_led_red = n;
                    nxt_led_green = 0;
                    nxt_led_blue = 0;
                end
                2'b01: begin        // green
                    nxt_led_red = 0;
                    nxt_led_green = n;
                    nxt_led_blue = 0;
                end
                2'b10: begin          // blue
                    nxt_led_red = 0;
                    nxt_led_green = 0;
                    nxt_led_blue = n;
                end
            endcase
        end
    end

endmodule
```