CS152B Lab 0 - Introduction Report

Introduction

This lab aims to familiarize us with the Xilinx programming environment. We went through the process of building and running our design of a simple 4-bit counter on the board.

Implementation

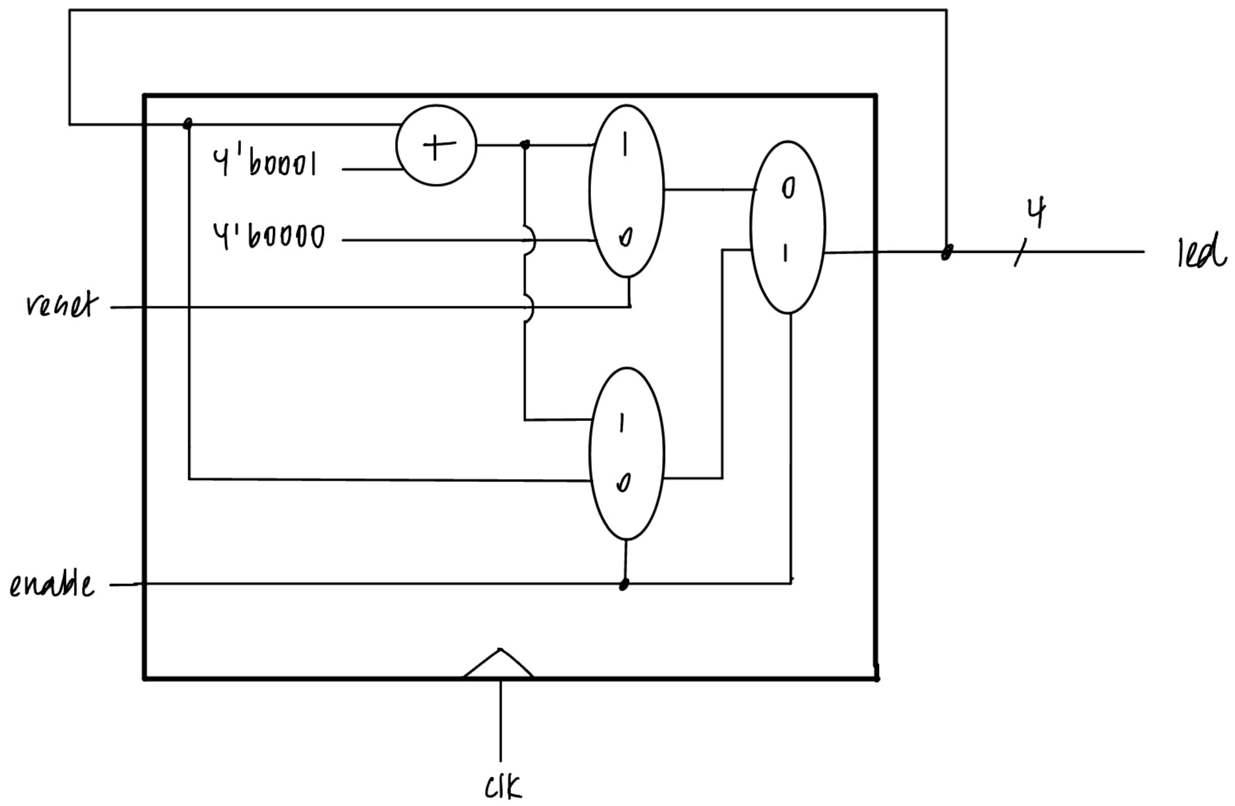


Figure 1. Schematic of module my_counter.

module my_counter

This module is responsible for implementing a 4-bit counter. On every rising edge of the clock, the module checks the state of reset and enable. If reset is high, the led output is reset to 0. Else if enable is high, the led value increments by 1. If reset and enable is low, the counter maintains its current value.

Inputs:

- clock: connected to flip flop that triggers updates to the led register on the positive edge of the clock.
- reset: resets counter to 0 when pressed.
- enable: starts counting when pressed.

Outputs:

- led: A 4-bit output register that holds the current count value.

Adder:
- Adds current value of led and 4'b0001.

Mux for reset:
- A 2-to-1 multiplexer to choose between 4'b0000 and the output of the adder.

Mux for enable:
- 2-to-1 multiplexer to choose between the current value of led and the output of the first mux.

Mux for reset/enable:
- 2-to-1 multiplexer to choose the output from either reset/enable.

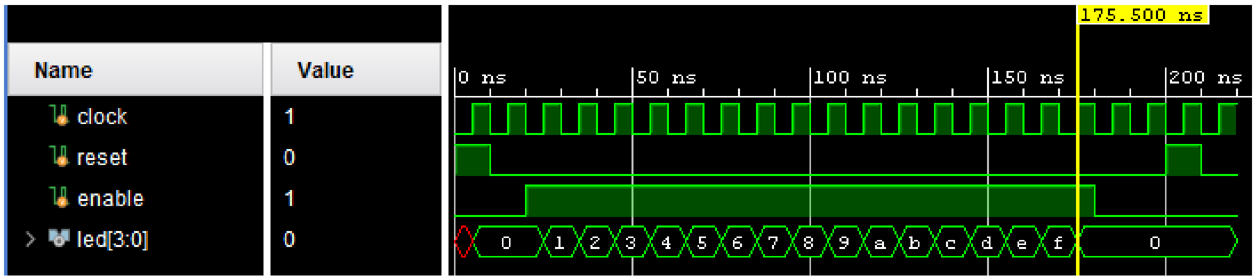
```
module my_counter(
   input clock,
   input reset,
   input enable,
   output reg [3:0] led
   );

   always @(posedge clock)
   begin
      if (reset) begin
         led <= 4'b0000;
      end else if (enable) begin
         led <= led + 1;
      end
   end
endmodule
```

module lab0_tb
This module is responsible for implementing the testbench.

```
module lab0_tb;

   reg clock;
   reg reset;
   reg enable;
   wire [3:0] led;
```

```
    my_counter lab0_counter (
        .clock(clock),
        .reset(reset),
        .enable(enable),
        .led(led)
    );

    // Note: testbench referenced from ChatGPT

    initial begin
        clock = 0;
        forever #5 clock = ~clock;
    end

    initial begin
        reset = 1;
        enable = 0;
        #10;

        reset = 0;
        #10;

        enable = 1;
        #80;

        enable = 0;
        #20;

        reset = 1;
        #10;

        reset = 0;
        #10;

        $stop;
    end
endmodule
```

Testing



The waveform above depicts the functionality and input responses of our 4-bit counter. The counter starts with the enable flag set to 0 and the reset flag is set to 1; the output hasn't been initialized by the counter module yet. At 4ns, the reset flag is set to 0, initializing the counter output to 0. Since the enable flag is still 0, the counter has not started increasing yet. At 24ns, the enable flag is set to 1, and the counter starts increasing on every positive clock edge, where each clock pulse is 10ns apart. Once the counter reaches f (15), the counter overflows and loops back to 0. The enable flag is again set to 0, stopping incrementation.

Challenges

One challenge we faced was uploading our code onto the board. We modified the .xdc file containing the board's constraints to properly map inputs/outputs to physical pins on the board. In particular, we used the board's clock signal, the center and up buttons for enable/reset respectively, and 4 LED pins to display counter output.

Responsibilities

905699244 - Implemented the my_counter module
105817312 - Implemented the lab0_tb (testbench) with test cases for the my_counter module. Drew schematic.