ES4 Lab 4

Flashing Displays

Qijin Chau

04/10/2020

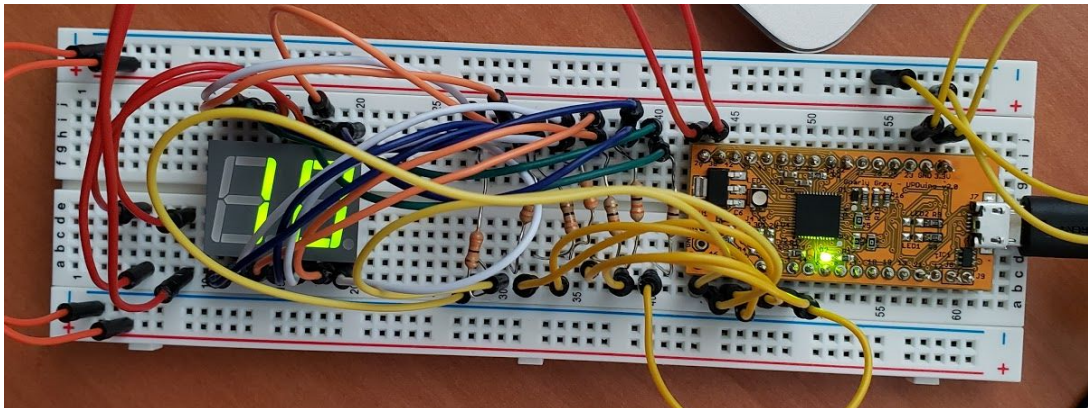
TA: Yiwen Jiang

Thursdays 8:00 AM

Combine a sequential circuit using combinational logic to display a 2-digit in base 10 on a seven-segment display.

Design

Image of completed circuit



Circuit is missing the two transistors
Hard coded the purpose of the transistors on Radiant instead

Debugging Log

There wasn't too much struggle with completing this lab. This was mostly thanks to essentially having an one-on-one session for my final lab and being able to screen share my Radiant code to the TA. The most difficult parts of the lab like *L4* and putting together all the components and signals in the top level module were made a lot easier because of this direct help whenever needed. The TA advised me of declaring and using signals that probably would have caused large amount of debugging time. Other than that, the only other trouble I had during the lab was getting the LED-blinking code working on the UPduino. The debugging process of that is described below.

Test - Running the LED-blinking code on the UPduino
Problem: I could not get the LEDs to blink. I tried using segments from the seven-segment display and also just two separate LED lights. For both methods, only one LED would be lit the entire time. It was always only one LED lit up and never neither of them or both of them.
Debugging: I tried altering parts of the code. At first, I tried changing the bit in the counter used for the led outputs but that made no different. In the end, what worked was putting `leds(0) <= counter(25)` and `leds(1) <= not counter(25)` inside the process rather than outside and changing the `26d"1"` and `26d"0"` notation when initialing the counter and for if-conditions to just typing out twenty-six 0's and twenty-six 1's. These changes in the code allowed the LEDs to blink or alternate.

Test - Wiring the seven-segment display
Problem: There wasn't really a problem except that with help form the TA, I decided to not use transistors to power each digit on the display because of the lack of time and ability to directly help with wiring on the breadboard.
Debugging: I decided to hard code the transistors in the top level module at the end. The if-else statement below was added and used in replacement for the transisors

```
if(leds(0) = '0') then
      sevdigdisplay <= ones_sig;
else
      sevdigdisplay <= tens_sig;
```

Test - Compiling errors for part *L4* and the top level module
Problem: None of the errors were major. It was mainly syntax and also I was missing a signal.
Debugging: To debug, I looked at the line location of the error and corrected the syntax by usually adding a colon or changing `end ...` to `end component`. Also, I had to hard code a signal to port map to the value input of the dddd component and give the signal a value which was changed each time when testing.

Testing

The first testing I did was for part *L1* of the lab. To find the bit on the counter so that the LEDs did not seem to blink anymore I had to test bits starting at the most significant bit and selecting a lower bit each time. I decided to do this by decrementing by 2 bits each test until the lights seemed to be barely blinking. By the 21nd bit, I noticed very slight blinking. So I decided to lower the bit one bit at a time until the LEDs stopped appearing to blink. Doing this test, the 18th bit or counter(18) was when the LEDs stopped appearing like they were blinking to me. Proper testing of this part of the lab is important because this bit where the LEDs stop appreaking to blink is the bit that will be used in the end. Not appearing to blink anymore is part of the trick/purpose of this lab of showing how to control and keep 14 LEDs lit up with only 9 outputs.

When working on the lab and my circuit for the final time, I forgot about the intermediate step of *L3* and went directly to coding the top module for the final part *L4*. If I was going to test *L3* I would have used the recommended testing of using a group of input pins controlled with a DIP switch. I am very comfortable with DIP switches and I feel like testing with DIP switches is very efficient because you can easily see what the inputs are and how the outputs correspond to those inputs.

The second testing I did was for part *L4* of the lab. But instead of writing code for toggling the number being displayed on the 2-digits of the seven-segment display, I hard coded a signal to represent the number generation for the input value of the dddd and gave it a value. To test my circuit and the code, I changed the value of the signal each time and reprogrammed the UPduino. The `"000000"` in the line below was changed each test

```
signal value_sig : unsigned(5 downto 0) := "000000";
```

I tested digits like 0, 3, 10, 17, 25. The results of each test matched the expected result. The display for numbers 10 and 17 are shown in the appendix section.

Reflection

1) I think the most valuable thing I learned was gaining more practice and familiarity working with Radiant, VHDL, and FPGAs. Even though these things are not completely new, they have been a skill that I have been struggling with and I feel like this lab really improved my confidence and proficiency with them. In Lab 3, one of the biggest struggles was setting up a project in Radiant and creating files with the proper setting then uploading to the UPduino. There was a lot of technical difficulties in Lab 3 which were not as prevalent in Lab 4, disregarding the complications with downloading Radiant on my personal computer. This allowed me to focus and designate more time towards creating the VHDL for each entity and combining them in the end, which helped with the learning process.
2) The skills I am still struggling with are linking components from different VHDL files to one top level module. I understand what I am doing conceptually and why I have to do it however they actual implementation and coding is still confusing. Specifically, the confusion are with how the components are port mapped after declaration, if additional signals are needed and how they would be used, and how the new components and signals will change the top entity and the process.
3) Excluding getting Radiant to work on my computer, this lab took me about for about 4 hours to complete. I used one and a half lab sessions and also an hour or so on my own time. However downloading Radiant and getting it to work was a process that took up a lot of time.

Appendix

Test case for displaying the digit 10

Test case for displaying the digit 17

VHDL Code

```
-- VHDL code for the top-level module,
-- including HSOSC and counter logic
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity top is
  port(
        leds : out std_logic_vector(1 downto 0);
        sevdigdisplay : out std_logic_vector(6 downto 0)
  );
end top;

architecture Behavioral of top is

signal my_CLK : std_logic;
signal counter : unsigned (25 downto 0) := "00000000000000000000000000";
signal value_sig : unsigned(5 downto 0) := "000000";
signal tens_sig : std_logic_vector(6 downto 0);
signal ones_sig : std_logic_vector(6 downto 0);

component HSOSC is
generic (
    CLKHF_DIV : String := "0b00"); -- Divide 48MHz clock by 2^N (0-3)
port (
    CLKHFPU : in std_logic := 'X'; -- Set to 1 to power up
    CLKHFEN : in std_logic := 'X'; -- Set to 1 to enable output
    CLKHF : out std_logic := 'X'); -- Clock output
end component;

component dddd is
  port(
    value : in unsigned(5 downto 0);
    tensdigit : out std_logic_vector(6 downto 0);
    onesdigit : out std_logic_vector(6 downto 0)
  );
end component;

begin
    clock : HSOSC port map('1', '1', my_CLK);
    dd    : dddd port map(value_sig, tens_sig, ones_sig);

    process(my_CLK)
    begin
```

```
        if(rising_edge(my_CLK)) then
            counter <= counter + 1;
        if(counter = "11111111111111111111111111") then
            counter <= "00000000000000000000000000";
        end if;
        end if;

        leds(1) <= counter(18);
        leds(0) <= not counter(18);

        if(leds(0) = '0') then
            sevdigdisplay <= ones_sig;
        else
            sevdigdisplay <= tens_sig;
        end if;
    end process;
end;


-- VHDL code for the dddd
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity dddd is
  port(
    value : in unsigned(5 downto 0);
    tensdigit : out std_logic_vector(6 downto 0);
    onesdigit : out std_logic_vector(6 downto 0)
  );
end dddd;

architecture sim of dddd is

signal lowBCD : unsigned(3 downto 0) := 4d"0";
signal highBCD : unsigned(3 downto 0) := 4d"0";
signal tensplace : unsigned(12 downto 0) := 13d"0";

component sevenseg is
  port(
        S : in unsigned(3 downto 0);
        seg : out std_logic_vector(6 downto 0)
  );
end component;

begin
      lowBCD <= value mod 4d"10";
```

```
      tensplace <= value * to_unsigned(52, 7);
      highBCD <= tensplace(12 downto 9);

      seven1 : sevenseg port map(lowBCD, onesdigit);

      seven2 : sevenseg port map(highBCD, tensdigit);
end;


-- VHDL code for the seven-segment display
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity sevenseg is
  port(
        S : in unsigned(3 downto 0);
        seg : out std_logic_vector(6 downto 0)
  );
end sevenseg;

architecture synth of sevenseg is
signal segments : std_logic_vector(6 downto 0);
begin
  test_display : process(S)
  begin
  case S is
  when "0000" =>
    segments <= "1111110";
  when "0001" =>
    segments <= "0110000";
  when "0010" =>
    segments <= "1101101";
  when "0011" =>
    segments <= "1111001";
  when "0100" =>
    segments <= "0110011";
  when "0101" =>
    segments <= "1011011";
  when "0110" =>
    segments <= "1011111";
  when "0111" =>
    segments <= "1110000";
  when "1000" =>
    segments <= "1111111";
  when "1001" =>
    segments <= "1110011";
```

```
  when "1010" =>
    segments <= "1110111";
  when "1011" =>
    segments <= "0011111";
  when "1100" =>
    segments <= "1001110";
  when "1101" =>
    segments <= "0111101";
  when "1110" =>
    segments <= "1001111";
  when others =>
    segments <= "1000111";
  end case;
end process test_display;
seg <= not segments;
end;
```