ES 4 Lab 6: Flashing Displays

Katherine Deane

22/29 October 2024

6-8pm Tuesday Section, TAs: Matt, Quan, Griffin

The objective of this lab is to design and build a circuit using a 7-segment display that can display the two digits of a 6-bit number given by a DIP switch. The input is the DIP switch, and the output is the 7-segment display.

Design – Truth Table

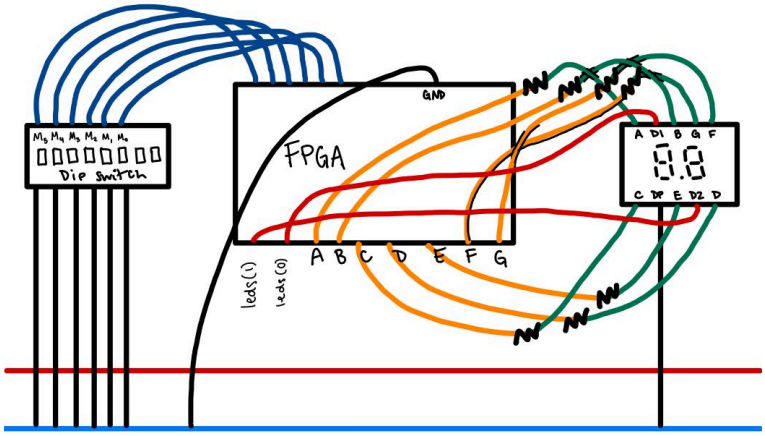
0 indicates when the segment is on, and 1 indicates when the segment is off.

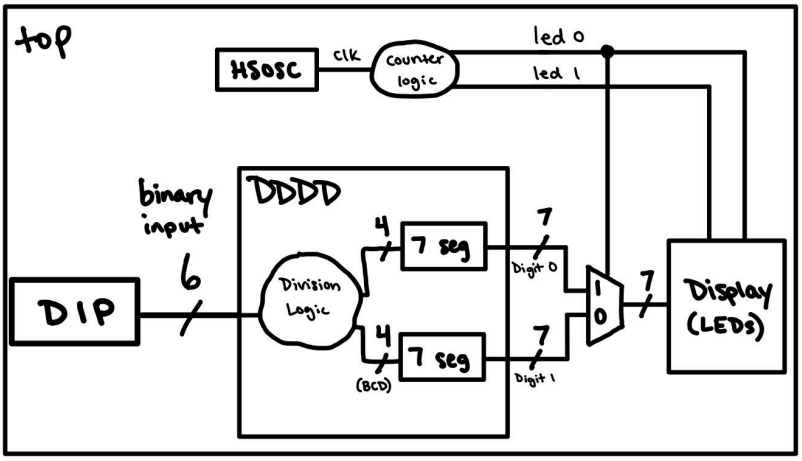
digit	**A**	**B**	**C**	**D**	**E**	**F**	**G**
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

Design description/circuit layout:

The input of the FPGA is the 6-bit number determined by the DIP switch. The FPGA is programmed to take that 6-bit binary number and determine the 4-bit binary numbers that represent the ones and tens digit (VHDL code below).

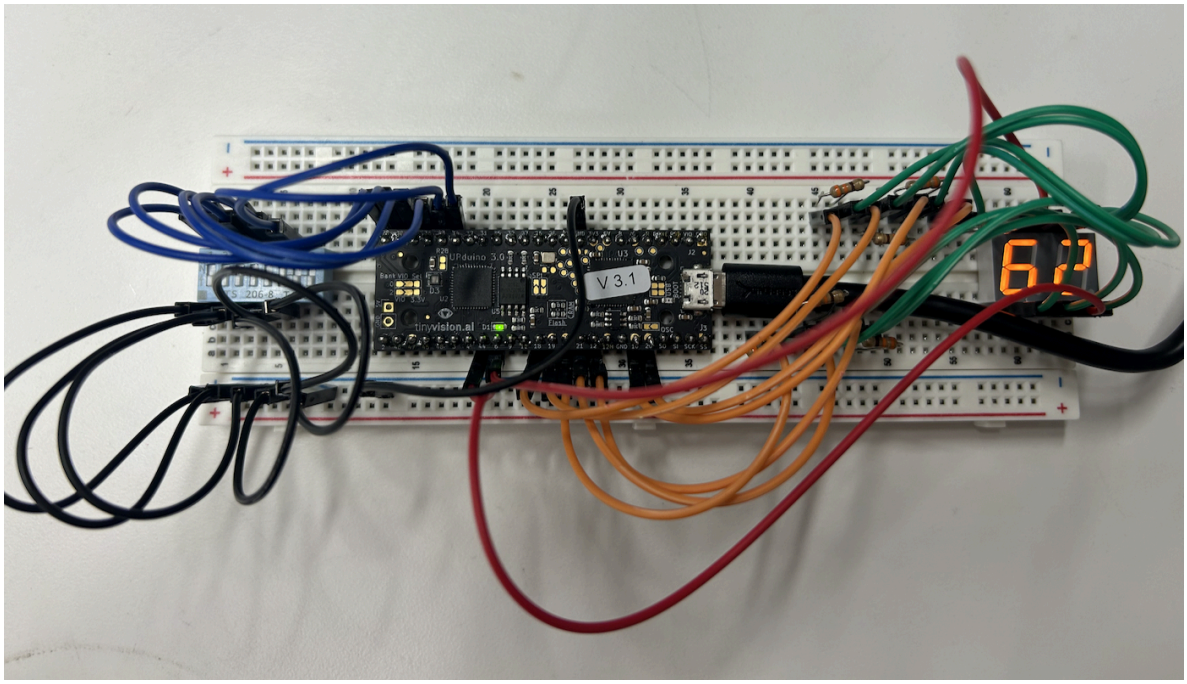
Those 4-bit outputs are connected to seven segments as the input values to be displayed, and the display gets the 7-bit output of each seven seg depending on whether leds(0) is high or low (effectively acting as a multiplexer). This is seen in the port map in my VHDL file named **top.vhd** (below), which maps the 7-bit inputs from each seven segment (depending on leds(0)) to the display (`display <= finalOnes when (leds(0)) else finalTens;`). The digits appear to be lit up simultaneously due to the speed of the counter.

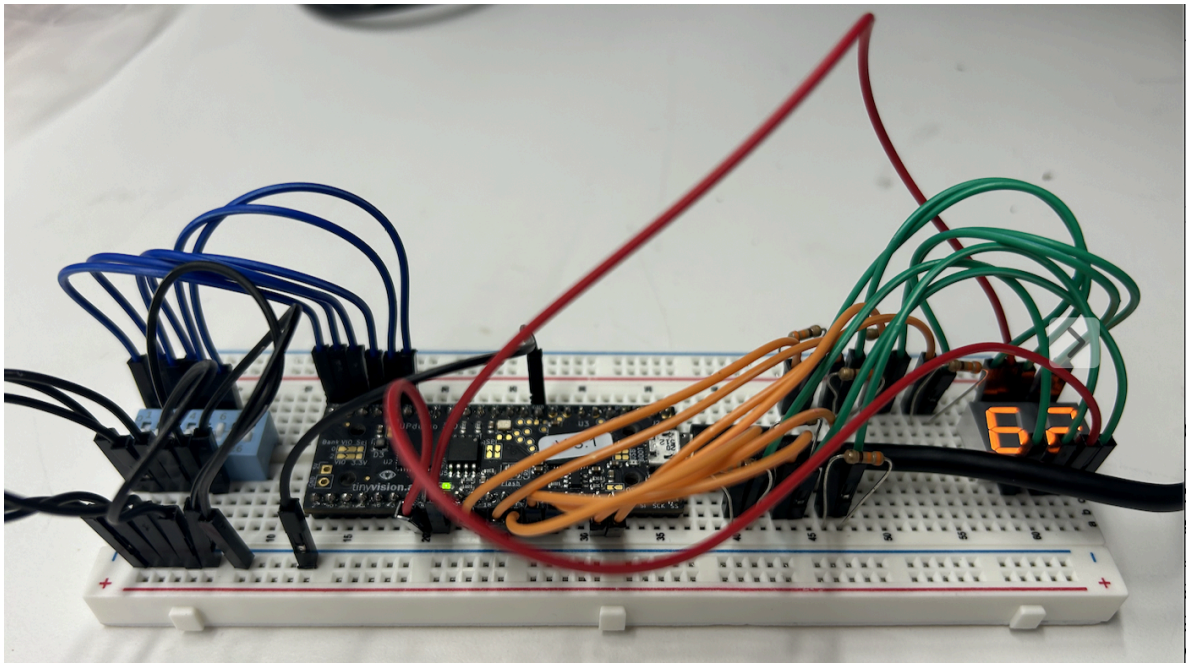




Completed Circuit:

The blue wires are the binary number to be displayed, the black wires are ground, the red wires connect to the 2 leds output (which rapidly switch between high and low to determine which digit is high), and the orange/green wires are each input bit of the seven segment display.





Debugging Log

- Since I utilized most of my circuit from the previous lab—I only removed the buttons, added an additional power wire for the second digit, and used 6 bits from the dip switch rather than 8—I didn't have any issues to debug in regards to my hardware.

- I did, however, have an issue with my VHDL code. At first, when I tested the input 48 (110000), the display showed 84. This was easy to spot the source of since it was just backwards, so I knew it was due to an issue with the second to last line of my top.vhd file:

```
display <= finalOnes when (leds(0)) else finalTens;
```

  which initially had finalOnes and finalTens in the opposite order, so I had a simple fix of just flipping the order, which worked. I think it also would have worked if I kept them in the same order but switched the destinations of the leds(0) and leds(1) wires to be opposite.
- I also had a fair amount of trouble coding the VHDL in general because I had confusion about syntax and also wasn't sure how the leds(0)/leds(1) output conceptually related to the display. Griffin helped me a lot with understanding which signals were necessary to connect the ***dddd*** component to the top module, and then get the proper digit (ones vs. tens) to display in the correct digit place by assigning the binary value of each digit to the 6-bit display output according to whether leds(0) was high or low. ***[Please ignore the following 3 bullets if you don't care about how I set that up]***
  - For example, when leds(0) is high, display will light up the ones digit spot (it will know to do this because of the red wire connecting the leds(0) output to the digit1 pin of the seven segment display) with the correct value (which is determined by the calculations for the ones digit in the ***dddd*** file and received through the orange/green wires).
  - This same process occurs for the tens digit when leds(1) is high (a.k.a. when leds(0) is low). The red wire connecting the leds(1) output to the digit2 pin of the seven segment display will tell the display to light up the tens place with the correct value (which is determined by the calculations for the tens digit in the ***dddd*** file and received from the orange/green wires).
  - I feel like I should mention that I do understand why the leds output makes it so that both digits appear lit at the same time, but feel free to ignore this if it doesn't matter: the counter is incrementing at a rapid rate, and taking a smaller bit from the 26-bit counter variable to dictate whether leds(0) is high or low means that it will be switching constantly due to the speed at which the counter is incrementing. If we assign one bit of the leds vector to be high when that 12th bit is high and the second bit of the leds vector to be high when the 12th bit is low, then they will both be flashing on and off so quickly that they will both appear to be on continuously.
- My plan for testing the design was just to put a bunch of random 6-bit numbers into the dip switch and confirm that the display was showing the correct 2-digit number. I did this for many different numbers, and it worked for all of them, so I think it's safe to assume that the circuit works.

6-bit input	Display
000001	1
111111	63
110000	48
110101	53
010100	20
000110	6
100011	35
001010	10
111110	62
001111	15
011011	27
000000	0

- All of the outputs matched the expected output values on the first try. Since this was a simple circuit (95% of it was reused from the previous lab), and all of my outputs are matching the expected ones, I'm very confident that my circuit is fully functional.
- I was checked off by Quan! I believe. It was 2 weeks ago.

Testing:

I tested the final circuit using the inputs recorded in the table above. All of the outputs listed in the table are both the expected and the actual values, so I am relatively confident that the circuit works for all possible inputs.

Reflection:

1) The most valuable thing I learned was how to instantiate a module with submodules in VHDL and integrate clock signals into that. The hardest part is still just understanding how the VHDL syntax works. Knowing how to use a module to connect multiple components (such as a DDDD, HSOSC clock, and seven segment display) in one electrical system is incredibly valuable because it taught us how we can produce a circuit that has multiple LEDs appearing to be lit simultaneously (though they are really just

rapidly blinking) simply by controlling the transfer of data with a clock and connecting various components.

2) I'm still struggling a lot with the syntax of VHDL. I still had a ton of trouble with instantiating the top module and understanding how to connect the submodules to it. I especially had trouble in this lab with the intermediate signals finalOnes and finalTens that were used to connect dddd.vhd to top.vhd. I also didn't realize that I needed a 6-bit display output in my top.vhd module, and I was clueless with what to assign to it (specifically that one line that I referenced in the debugging section); after talking to Griffin and Quan about it, it makes sense to me. To learn and become more comfortable with these skills, I'm just going to keep messing around with the in-class and homework examples on VHDL, as well as try to really understand how and why the VHDL code that I write in lab works the way it does.
3) It took me about 3 hours to complete the lab. It took me about an hour to do the prelab (I needed to go to OH for help), 1 hour in the lab to get the circuit working the way I wanted, and about an hour to write up this report. I didn't encounter any major time-consuming issues with the circuit.

VHDL Code:

top.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity top is
  port(
        dipBits : in unsigned(5 downto 0);
        display : out std_logic_vector(6 downto 0);
        leds : out std_logic_vector(1 downto 0)
  );
end top;

architecture synth of top is
signal clk : std_logic;
signal count : unsigned(25 downto 0) := (others => '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;

signal finalTens : std_logic_vector(6 downto 0);
signal finalOnes : std_logic_vector(6 downto 0);

begin
    osc : HSOSC generic map ( CLKHF_DIV => "0b00")
            port map (CLKHFPU => '1',
            CLKHFEN => '1',
            CLKHF => clk); -- clk is run by CLKHF now

process (clk) begin
    if rising_edge(clk)
        count <= count + '1';
    end if;
end process;

     leds(0) <= count(12);
     leds(1) <= not count(12);

     dddd1: dddd port map(value => dipBits, tensdigit => finalTens,
onesdigit => finalOnes);
     display <= finalOnes when (leds(0)) else finalTens;
end;
```

---

dddd.vhd

```
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

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

signal lowBCD : unsigned(3 downto 0);
signal highBCD : unsigned(3 downto 0);
signal tensplace : unsigned(12 downto 0);

begin

-- Do the math to split up the digits. Input `value` is 6 bit
unsigned
lowBCD <= value mod 4d"10"; -- Low digit result is 4 bit unsigned
-- Multiply by 52. Intermediate term is 13 bit unsigned
tensplace <= value * 7d"52";
-- Divide by 512 (2^9). High digit result is 4 bit unsigned
highBCD <= tensplace(12 downto 9);

sevenseg1: sevenseg port map(S => lowBCD(3 downto 0), segments =>
onesdigit);
sevenseg2: sevenseg port map(S => highBCD(3 downto 0), segments =>
tensdigit);

end;
```

---

sevenseg.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
use IEEE.numeric_std.all;

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

architecture synth of sevenseg is
begin
  process(S)
   begin
      case S is
        when "0000" => segments <= "0000001"; --0
        when "0001" => segments <= "1001111"; --1
        when "0010" => segments <= "0010010"; --2
        when "0011" => segments <= "0000110"; --3
        when "0100" => segments <= "1001100"; --4
        when "0101" => segments <= "0100100"; --5
        when "0110" => segments <= "0100000"; --6
        when "0111" => segments <= "0001111"; --7
        when "1000" => segments <= "0000000"; --8
        when "1001" => segments <= "0001100"; --9
        when others => segments <= "1111111"; -- All segments off
invalid input
      end case;
   end process;

end;
```