ES 4 Lab 5: Building an ALU and Displaying the Result

Katherine Deane

21 October 2024

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

The objective of this lab is to design and build a circuit using an ALU and 7-segment display that can perform the addition, subtraction, AND, and OR operations—which operation is performed is determined by two buttons, one for each bit S_1 and S_0—on two 4-bit numbers given by a DIP switch. The inputs are the 2 buttons and DIP switches, 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

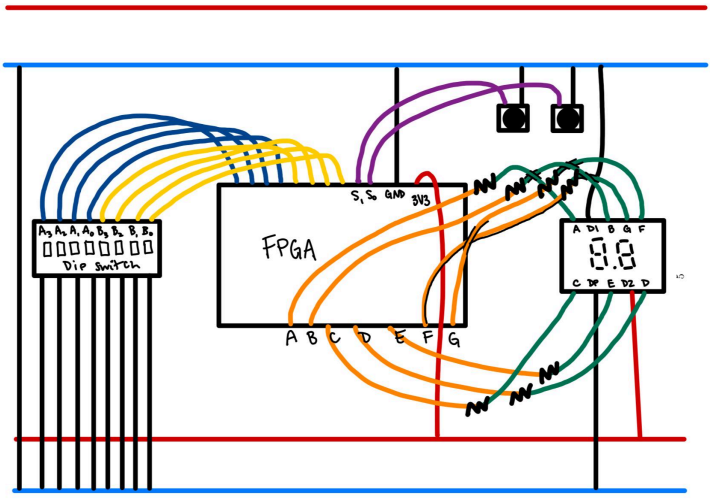
F	0	1	1	1	0	0	0

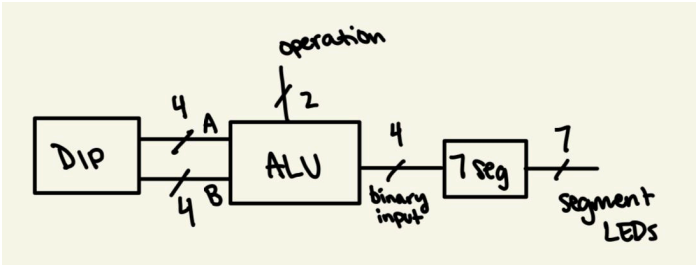
Design description/circuit layout:

A description of your design. At a minimum, this includes a block diagram and an explanation of the inputs you used for the display.

I drew out a breadboard design of my circuit below and included a block diagram. The inputs of the ALU are the two 4-bit numbers (A and B; blue and yellow) determined by the DIP switch and the 2-bit number (S; purple) which is determined by the two buttons and chooses the operation/comparison to be performed on A and B.

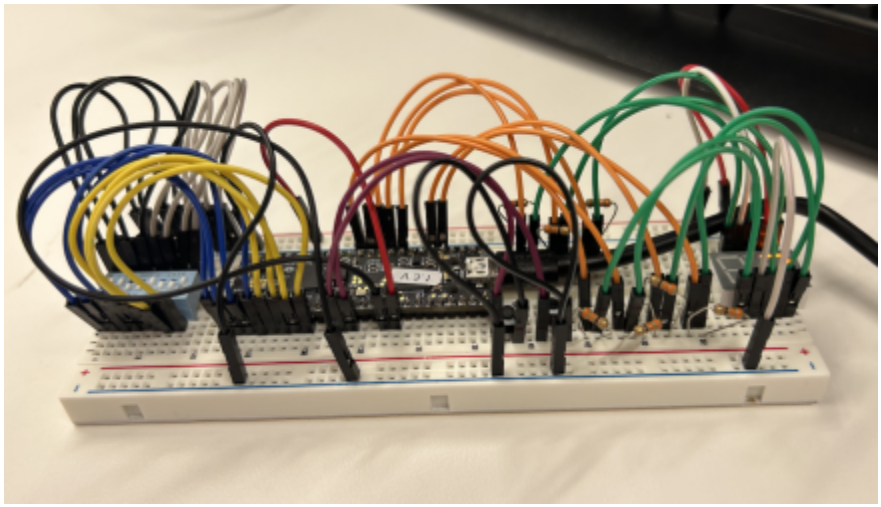
The 4-bit output of whatever operation/comparison is performed by the ALU is connected to the seven segment display as the input value to be displayed. This is seen in the port map in my VHDL file named **top.vhd** (below), which maps the result of the ALU to the 4-bit input of the display (`sevenseg1: sevenseg port map(binary => aluresult, segments => finalResult);`). The VHDL file **sevenseg.vhd** then takes that 4-bit input and determines which bits to assign to the 7-bit output (A-F) so that the correct segments light up.

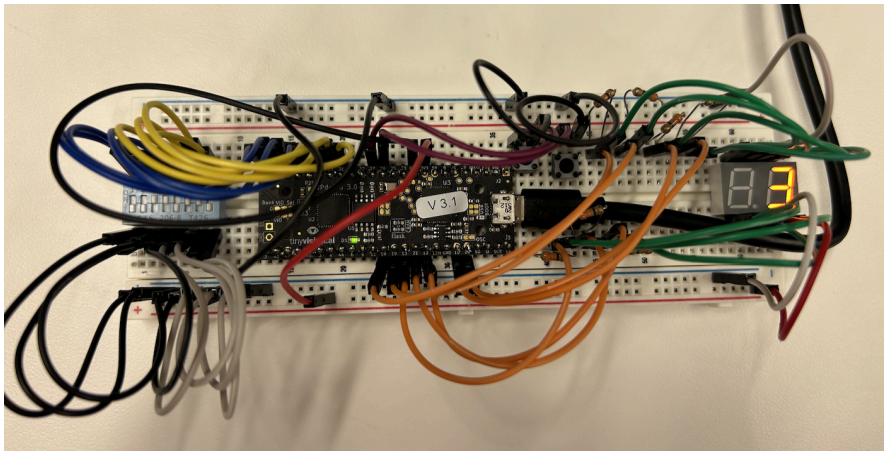




Completed Circuit:

The blue wires are the first binary number (A), the yellow wire are the second binary number (B), the black and gray wires are ground, the red wires connect to 3.3V power, the purple wires connect to the 2 buttons, and the orange/green wires are each input bit of the seven segment display.





Debugging Log

- To test my ALU before setting up the seven segment display, I attached an LED to each bit of the output and tested various inputs that cover many different cases of inputs A and B. I did this because I couldn't test all 16^2*4 possible input combinations, so the idea was if I covered a broad enough array of test cases, hopefully it would uncover any potential errors with the circuit thus far.

A	B	Y (S = 11, -)	Y (S = 10, +)	Y (S = 01, ||)	Y (S = 00, &&)
0000	0000	0000	0000	0000	0000
0000	0001	1111	0001	0001	0000
0100	0011	0001	0111	0111	0000
1100	1111	1101	1011	1111	1100

1111	1100	0011	1011	1111	1100
1001	1100	1101	0101	1101	1000
1111	1111	0000	1110	1111	1111
0110	1101	1001	0011	1111	0100
1101	0010	1011	1111	1111	0000
1010	1100	1110	0110	1110	1000
0101	0011	0010	1000	0111	0001
0001	1000	1001	1001	1001	0000
0111	0100	0011	1011	0111	0100

- All outputs matched the expected values.

- Next I connected the seven segment display to my ALU setup. It was giving me totally funky output: only the A and C segments were lighting up, and they were lighting up at totally arbitrary times. If I changed A, B, S_1, and S_0, only A and/or C would be on, and they would change without any obvious reason or pattern. I asked Matt for help, and he showed me that I had my MSBs and LSBs flipped in my pin setup in radiant, so I flipped the pin numbers.
- It still wasn't working! I was still only getting A and/or C to light on for any given input. Griffin helped me look at it and we couldn't figure out what was wrong, so he flashed the FPGA and it started working. This taught me that I have to flash the FPGA after inputting the pin values so that my code in radiant actually connects to the FPGA. Who would have thought!
- Now to test the whole circuit, I put in all of the same A, B, S_1, and S_0 inputs that I used in my initial testing of the ALU; all outputs matched the expected outputs. The numbers/letters in the output columns Y are what were shown on the 7-segment display.

A	**B**	**Y (S = 11, -)**	**Y (S = 10, +)**	**Y (S = 01, ||)**	**Y (S = 00, &&)**
0000	0000	0	0	0	0
0000	0001	F	1	1	0
0100	0011	1	7	7	0
1100	1111	d	b	F	C

1111	1100	3	b	F	C
1001	1100	d	5	d	8
1111	1111	0	E	F	F
0110	1101	9	3	F	4
1101	0010	b	F	F	0
1010	1100	E	6	E	8
0101	0011	2	8	7	1
0001	1000	9	9	9	0
0111	0100	3	b	7	4
0001	1001	8	A	9	1

- All of the outputs matched the expected output values on the first try. Since this was a simple circuit, and all of my outputs are matching the expected ones, I'm very confident that my circuit is fully functional. Matt checked over my circuit and confirmed that it worked properly.

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. It seemed like a really daunting task, but after doing it, I can see that the hardest part is just understanding how the VHDL syntax works, and the rest is pretty simple. Knowing how to use a module connect multiple components (such as an ALU and a seven segment display) in one electrical system is incredibly valuable because it enables you to produce a circuit with multiple functions (like comparing/performing calculations on numbers and displaying the results) simply by connecting the output of one component to the input of another.
2) I'm still struggling a lot with the syntax of VHDL. It's really not intuitive like other coding languages, and I had a ton of trouble with instantiating the top module and

understanding how to connect the submodules to it. 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 6 hours to complete the lab. It took me about an hour to do the prelab, both full lab sessions for the actual lab (1 to get the ALU working and the other to get the seven segment display working), and about an hour to write up this report. I didn't encounter any major time-consuming issues with the circuit, other than forgetting to flash the FPGA.

VHDL Code:

top.vhd:

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

entity top is
     port(
          signalA : in unsigned(3 downto 0);
          signalB : in unsigned(3 downto 0);
          signalC : in std_logic_vector(1 downto 0);
          finalResult : out std_logic_vector(6 downto 0)
     );
end top;

architecture synth of top is
component alu is
     port(
          a : in unsigned(3 downto 0);
          b : in unsigned(3 downto 0);
          s : in std_logic_vector(1 downto 0);
          y : out unsigned(3 downto 0)
     );
end component;

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

```
signal aluresult : unsigned(3 downto 0);

begin
     alu1 : alu port map(a => signalA, b => signalB, y => aluresult,
     s => signalC);
     sevenseg1: sevenseg port map(binary => aluresult, segments =>
     finalResult);
end;
```

---

alu.vhd:

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


entity alu is
     port(
          a : in unsigned(3 downto 0);
          b : in unsigned(3 downto 0);
          s : in std_logic_vector(1 downto 0);
          y : out unsigned(3 downto 0)
     );
end alu;

architecture synth of alu is
begin
     y <= (a and b) when (s = "00") else
           (a or b) when (s = "01") else
           (a + b) when (s = "10") else
           (a - b) when (s = "11");
end;
```

---

sevenseg.vhd:

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

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

architecture synth of sevenseg is
begin
  process(binary)
   begin
      case binary 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 "1010" => segments <= "0001000"; --A
        when "1011" => segments <= "1100000"; --b
        when "1100" => segments <= "0110001"; --C
        when "1101" => segments <= "1000010"; --d
        when "1110" => segments <= "0110000"; --E
        when "1111" => segments <= "0111000"; --F
        when others => segments <= "1111111"; -- All segments off
      end case;
   end process;
end;
```