Building an ALU and displaying the result

Toki Nishikawa

October 11 2022
6-8pm Tuesday lab with Kim Nguyen/Nick Boudreau

The objective of the lab is to implement a 4-bit arithmetic logic unit (ALU), which can perform addition, subtraction, bitwise AND, and bitwise OR, using an FPGA and display the output on a seven-segment display.

1 Design

The first step is to create a VHDL module that describes an ALU with two 4-bit inputs, a 2-bit control signal, and a 4-bit output. Next, we must create a top-level module that instantiates our ALU module. Note that the corresponding code for all modules can be found in section 5. The schematic of the top-level module with just the ALU module is displayed in **Figure 1**. In this diagram, *A* and *B* represent the two 4-bit inputs, *Operation* represent the 2-bit control signal, and *Output* represents the 4-bit output.

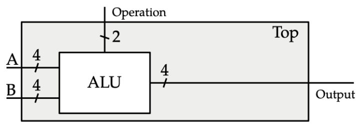

Figure 1: Schematic of top-level module with ALU module

Once the ALU module is functioning properly, we can construct and incorporate a module that describes our seven-segment display. The updated schematic of the top-level module is shown in **Figure 2**. In this diagram, *Output* now represents a 7-bit output. This is all of the logic we wish to implement using our FPGA.

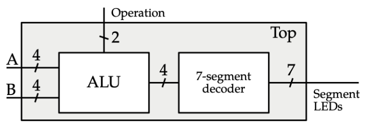

Figure 2: Schematic of top-level module with ALU and seven-seg modules

If you take a look at the code in section 5, you will see the following port declarations:

- For ALU module:
  - *a*: 4-bit unsigned
  - *b*: 4-bit unsigned
  - *s*: 2-bit standard logic vector
  - *y*: 4-bit unsigned
- For seven-segment display module:
  - *S*: 4-bit unsigned
  - *segments*: 7-bit standard logic vector

- For top module (named *lab5* in the code):
  - *a_t*: 4-bit unsigned
  - *b_t*: 4-bit unsigned
  - *s_t*: 2-bit standard logic vector
  - *y_t*: 7-bit standard logic vector

Note that the inputs to the top module, *a_t*, *b_t*, and *s_t* share the same types as the inputs to the ALU module, a, b, and s, respectively. Also, the output of the top module, *y_t* is the same type as the output of the seven-segment display module, *segments*. Similarly, the output of the ALU module, y, is the same type as the input to the seven-segment module, S.

One of the errors I encountered while writing the code was the following:

ERROR - Could not resolve blackbox module 'alu_module'.

This error occurred while I was trying to compile the code for the top module, and it took me a fair bit of time to figure out. The internet wasn't particularly helpful, and I looked through the code for my top module over and over, but I couldn't find anything wrong. It turns out that the component name for my ALU module in my top module did not match the entity name in the ALU module, which was *alu_mod* as opposed to *alu_module*. Since I was initially only looking at the code for my top module, I wasn't able to identify the issue right away. Thus, it is important to check all of the VHDL modules corresponding to your components when debugging errors in your top module!

2 Implementation

In order to easily configure different input combinations, we will use a DIP switch for the two 4-bit inputs and two pushbutton switches for each bit in the control signal. A diagram of this circuit is shown in **Figure 3**. Note that we don't need to use pull-up resistors since these are already included in the FPGA.

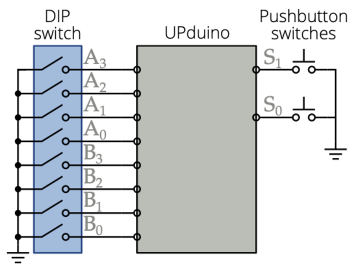

Figure 3: Schematic of DIP switch and pushbutton switches

Recall that the first step in our design was to create a top module that instantiates our ALU module. It's a good idea to test that the ALU module is functioning correctly before moving on to the seven-segment display module. Thus, we will flash the design corresponding to **Figure 1** onto our FPGA and test it by toggling the DIP switches and pressing the pushbuttons. We can also wire up some LEDs to the outputs. A picture of this circuit is shown in **Figure 4**. Notes on the debugging process for this circuit can be found in the next section.

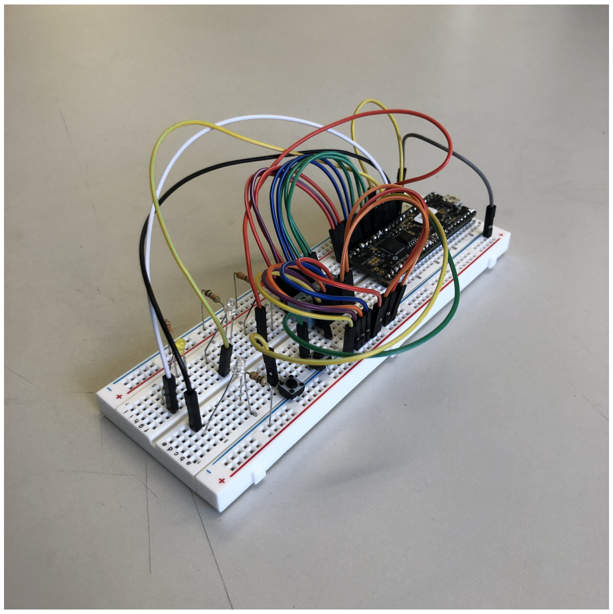

Figure 4: Circuit to test ALU

After we have achieved a functioning ALU module, we can incorporate the seven-segment display module and flash the updated design onto our FPGA. The inputs to our circuit remain unchanged, but the output is now a 7-bit standard logic vector corresponding to each segment on the seven-segment display. **Figure 5** shows the schematic for the seven-segment display while **Figure 6** shows the letters associated with each segment. The latter will be referenced during the testing section.

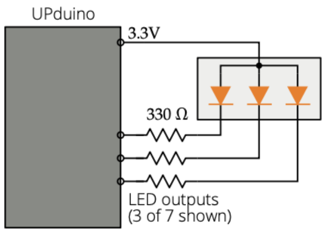

Figure 5: Schematic of seven-segment display

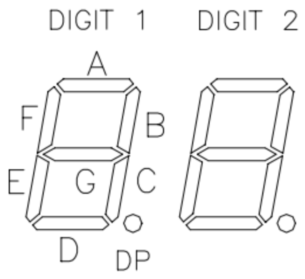

Figure 6: Letters associated with each segment

The completed circuit is shown in **Figure 7**.

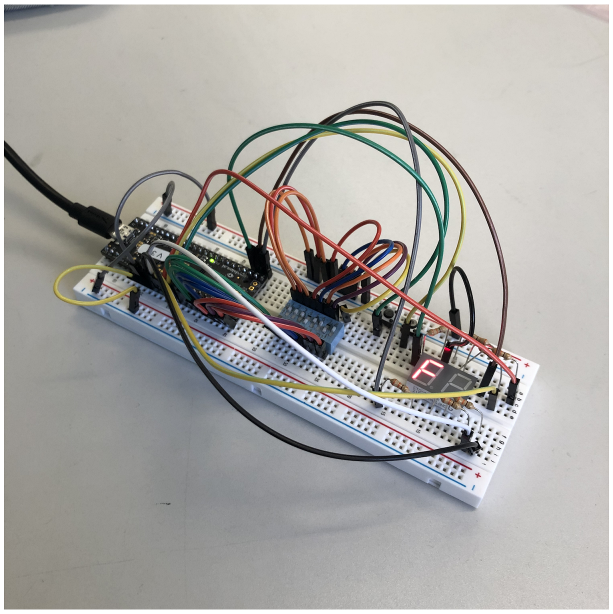

Figure 7: Completed circuit

3 Testing

3.1 Testing ALU module

Each step of the implementation process had its challenges. The circuit created to test the ALU module (shown in **Figure 4**) did not work the first time round.

At first, none of the LEDs would light up for any combination of the inputs, which is indicative of a problem that affects the entire circuit. One such issue is a faulty FPGA ground connection, but that seemed to be okay. I checked that the output pins were all set correctly, as this could have affected all of the LEDs if they happened to all be set incorrectly. I flashed the FPGA one more time to ensure that the code had uploaded properly, and I made sure that it was connected to power. After none of that worked, I got help from my TA who suggested that I try a different ground pin on the FPGA – it turned out that this was the issue. I had a faulty ground pin. As you will see throughout the rest of this lab, faulty pins are more common than I thought. After this fix, my LEDs were lighting up and it was time to test a bunch of combinations of the inputs.

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	S_1	S_0	Operation	Expected	Result
0	0	0	1	0	0	0	1	1	0	Addition	0010	0100

Table 1: First test of ALU module

Table 1 shows the inputs and outputs of the first test while **Figure 8** shows the corresponding circuit (note that the inputs, A, B, and S, represent the inputs of the top module *a_t*, *b_t*, and *s_t* in the code). As you can see, the result I recorded was not the expected 4-bit binary number. Since the result just had a 1 in $y(2)$ instead of $y(1)$, I suspected that I had swapped the two output wires associated with these middle bits. After checking the output pins on Radiant and the corresponding wiring, I discovered that I was misreading the order I had set for the LEDs. The LEDs are ordered in an L shape where the most significant bit of the 4-bit output is represented by the yellow LED at the tip of the L. Thus, the lit, red LED in **Figure 8** represents the second least significant bit, yielding the correct result 0010. However, I was initially reading the yellow LED at the tip of the L as the least significant bit. Thus, the output was actually correct. In the future, it may be worth writing the bit numbers for each input and output on sticky notes and sticking them to the appropriate circuit elements on the breadboard.

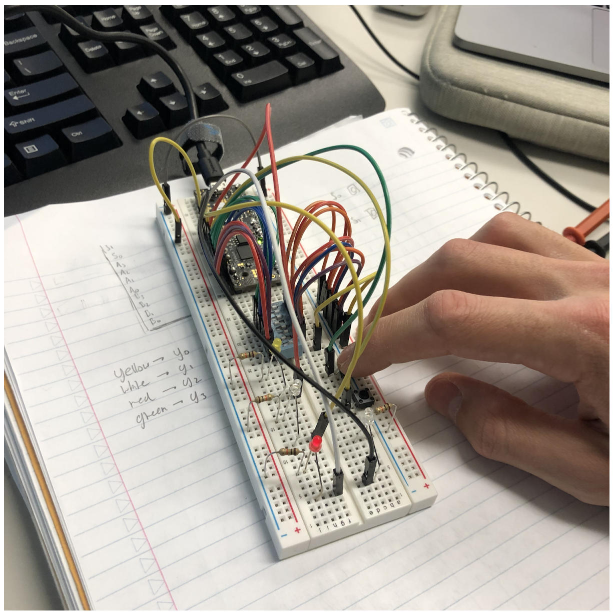

Figure 8: First test of ALU module circuit

The second round of testing is summarized in **Table 2**. I tried to add 0001 to 0010 but obtained an incorrect result. All the output connections were correct, since this is what I had double-checked previously. All of the wires seemed to be plugged into the breadboard well. I checked the code for my ALU module, and everything seemed to be okay. I decided to test the remaining operations with the same A and B to see if that would provide any hints. These results are summarized in **Table 3**. As you can see, the addition, bitwise AND, and subtraction operations were not functioning properly. Then suddenly, as I was trying different combinations, everything just started working. I really have no idea what happened. Maybe there was a connection loose that got pushed in while I was re-configuring the DIP switch. Who knows.

Table 4 shows the tests for all operations of four different configurations of A and B. As you can see, they all obtained the correct results.

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	S_1	S_0	Operation	Expected	Result
0	0	0	1	0	0	0	1	1	0	Addition	0010	0010
0	0	0	1	0	0	1	0	1	0	Addition	0011	0100

Table 2: Second test of ALU module

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	S_1	S_0	Operation	Expected	Result
0	0	0	1	0	0	1	0	1	0	Addition	0011	0100
0	0	0	1	0	0	1	0	0	1	OR	0011	0011
0	0	0	1	0	0	1	0	0	0	AND	0000	0001
0	0	0	1	0	0	1	0	0	0	Subtraction	1111	1110

Table 3: Third test of ALU module

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	S_1	S_0	Operation	Expected	Result
0	0	0	1	0	0	1	0	1	0	Addition	0011	0011
0	0	0	1	0	0	1	0	0	1	OR	0011	0011
0	0	0	1	0	0	1	0	0	0	AND	0000	0000
0	0	0	1	0	0	1	0	0	0	Subtraction	1111	1111
0	1	1	0	1	0	1	0	1	0	Addition	0000	0000
0	1	1	0	1	0	1	0	0	1	OR	1110	1110
0	1	1	0	1	0	1	0	0	0	AND	0010	0010
0	1	1	0	1	0	1	0	1	1	Subtraction	1100	1100
1	0	0	1	1	1	1	1	1	0	Addition	1000	1000
1	0	0	1	1	1	1	1	0	1	OR	1111	1111
1	0	0	1	1	1	1	1	0	0	AND	1001	1001
1	0	0	1	1	1	1	1	1	1	Subtraction	1010	1010
1	1	0	1	0	0	0	1	1	0	Addition	1110	1110
1	1	0	1	0	0	0	1	0	1	OR	1101	1101
1	1	0	1	0	0	0	1	0	0	AND	0001	0001
1	1	0	1	0	0	0	1	1	1	Subtraction	1100	1100

Table 4: Final test of ALU module

3.2 Testing seven-segment module

Next, we incorporated the seven-segment display (I'll refer to this as *sevenseg* for the rest of the report) to make the circuit shown in **Figure 7**. At first, nothing was showing up on the sevenseg for any combination of the inputs. Similar to when none of the LEDs worked in the previous circuit, this is indicative of a problem that affects all pins of the sevenseg. Thus, it is likely an issue with the anode pin. This pin was connected to a resistor, and then to $5V$ on the FPGA. I checked the voltage of the anode pin with my multimeter to ensure that it was $3.3V$, but as I had feared, its voltage was much less. I then checked the voltage of the $5V$ pin on the FPGA to make sure that it was actually outputting $5V$. As I did this, I realized I had actually connected it to the $3.3V$ pin, which I didn't know was present on the FPGA. Since the anode pin on the sevenseg was connected to $3.3V$ with a resistor, the light was so faint that I couldn't see it. I also must have used a resistor much too large since I initially measured an extremely small voltage. In the end, I just directly connected the anode to the $3.3V$ pin on the FPGA without a resistor, and I could see the outputs perfectly.

Table 5 shows the inputs and outputs of the first test. As you can see, the test failed. Instead of getting the expected 2, I obtained an L, which should never happen. I decided to look into one incorrect segment at a time. I started with the middle segment, G, which should have been lit up but wasn't. I checked the voltage of the sevenseg pin associated with G, and it was high. Note that a high voltage indicates that the corresponding segment will be off. Thus, I'm either interpreting the result incorrectly, or there is something wrong with the logic. It turns out that I was reading the result wrong. I had the two 4-bit inputs on the DIP switch ordered from most significant to least significant bit, but I was reading it the other way around.

Problem not completely solved. **Table 6** shows the new result, a lowercase c. This result should also never happen (a 12 is represented with an upper case C). Although G is now lit, there are two other segments that should be on, A and B. For some unknown reason, when I pushed the wires associated with these pins into the breadboard with enough force (on the FPGA pin side rather than the resistor side), the corresponding segments both lit up. This was very bizarre. I tried changing the wires, but that didn't help. I suspected that something was wrong with the breadboard, and so I tried connecting the wires to the soldering on the top of the FPGA instead, and it worked. A TA advised that the problem could also be due to faulty pins on the FPGA board, which turned out to be correct since we replaced the FPGA board and it worked perfectly. This still doesn't explain why A and B were lighting up when I was pushing the wires really hard into the breadboard. Maybe part of the wire would touch the soldering on the top of the FPGA board. Who knows.

Table 7 shows the tests for all operations of five different configurations of A and B. As you can see, they all obtained the correct results.

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	S_1	S_0	Operation	Expected	Result
0	0	0	1	0	0	0	1	1	0	Addition	2	L

Table 5: First test of completed circuit

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	S_1	S_0	Operation	Expected	Result
0	0	0	1	0	0	0	1	1	0	Addition	2	c

Table 6: Second test of completed circuit

4 Reflection

The most valuable takeaway from this lab is knowing how to take advantage of Radiant software to program an FPGA. Up until this point, we have constructed our circuits using only logic ICs. Now, we have the tools to create much more complex logic functions, such as the ones demonstrated in this lab.

A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	S_1	S_0	Operation	Expected	Result
0	0	0	1	0	0	1	0	1	0	Addition	3	3
0	0	0	1	0	0	1	0	0	1	OR	3	3
0	0	0	1	0	0	1	0	0	0	AND	0	0
0	0	0	1	0	0	1	0	0	0	Subtraction	F	F
0	1	1	0	1	0	1	0	1	0	Addition	0	0
0	1	1	0	1	0	1	0	0	1	OR	E	E
0	1	1	0	1	0	1	0	0	0	AND	2	2
0	1	1	0	1	0	1	0	1	1	Subtraction	C	C
1	0	0	1	1	1	1	1	1	0	Addition	8	8
1	0	0	1	1	1	1	1	0	1	OR	F	F
1	0	0	1	1	1	1	1	0	0	AND	9	9
1	0	0	1	1	1	1	1	1	1	Subtraction	A	A
1	1	0	1	0	0	0	1	1	0	Addition	E	E
1	1	0	1	0	0	0	1	0	1	OR	d	d
1	1	0	1	0	0	0	1	0	0	AND	1	1
1	1	0	1	0	0	0	1	1	1	Subtraction	C	C
0	1	1	1	1	1	0	0	1	0	Addition	3	3
0	1	1	1	1	1	0	0	0	1	OR	F	F
0	1	1	1	1	1	0	0	0	0	AND	4	4
0	1	1	1	1	1	0	0	1	1	Subtraction	b	b

Table 7: Final test of completed circuit

Although my Radiant abilities have improved massively throughout the course of this lab, I continue to struggle with syntax. I believe that the best way to practice VHDL syntax is to create my own cheat sheet that includes code I've written in previous projects. I've already started to compile a collection of VHDL modules that could be useful in the future.

The prelab took ∼30min, the lab itself took ∼6 hours, and the report took ∼6 hours.

List of Tables

5 VHDL code

5.1 Top Module

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

entity lab5 is
port(
        a_t : in unsigned(3 downto 0);
        b_t : in unsigned(3 downto 0);
        s_t : in std_logic_vector(1 downto 0);
        y_t : out std_logic_vector(6 downto 0)
);
end;

architecture synth of lab5 is
    component alu_mod 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(
            S : in unsigned(3 downto 0);
            segments : out std_logic_vector(6 downto 0)
        );
    end component;

    signal output_alu : unsigned(3 downto 0);

begin
    alu_inst : alu_mod
        port map (
            a => a_t,
            b => b_t,
            s => s_t,
            y => output_alu
        );
    sevenseg_inst : sevenseg
        port map (
            S => output_alu,
            segments => y_t
        );
end;
```

5.2 ALU Module

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

entity alu_mod 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_mod;

architecture synth of alu_mod is
begin
    y <= a and b when s(1) = '0' and s(0) = '0' else
         a or b when s(1) = '0' and s(0) = '1' else
         a + b when s(1) = '1' and s(0) = '0' else
         a - b;
end;
```

5.3 Seven-segment Module

```
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(all) begin
        case S is
            when X"0" => segments <= "0000001";
            when X"1" => segments <= "1001111";
            when X"2" => segments <= "0010010";
            when X"3" => segments <= "0000110";
            when X"4" => segments <= "1001100";
            when X"5" => segments <= "0100100";
            when X"6" => segments <= "0100000";
            when X"7" => segments <= "0001111";
            when X"8" => segments <= "0000000";
            when X"9" => segments <= "0000100";
            when X"A" => segments <= "0001000";
            when X"B" => segments <= "1100000";
            when X"C" => segments <= "0110001";
            when X"D" => segments <= "1000010";
            when X"E" => segments <= "0110000";
            when X"F" => segments <= "0111000";
            when others => segments <= "1111111";
        end case;
    end process;
end;
```