ES 4 Lab 5: Building an ALU and displaying the result

Prelab due 24 hours before your lab session, week of October 6

Lab report due at your lab time, week of Oct 20

1 Introduction

You've built some useful computational circuits in the past labs, but implementing larger or more complex computations with 74-series logic on a breadboard quickly becomes impractical.[1]

Instead, we will use the Lattice iCE40 UltraPlus FPGA, which contains about five thousand logic elements. As we discussed in class, each logic element contains a 16-value look-up-table (aka a multiplexer with constant inputs) which allows it to implement arbitrary truth tables with up to 4 variables.

In this lab, you will build a 4-bit arithmetic logic unit (ALU), and display the output on a seven-segment display. An ALU is the hardware unit at the core of a CPU that performs one of several arithmetic operations. Your ALU for this lab will perform addition, subtraction, bitwise AND, and bitwise OR, selected based on two input signals:

Input	Operation	C operator	Example
00	Bitwise AND	`a & b`	0110 & 0010 = 0010
01	Bitwise OR	`a | b`	0110 | 0010 = 0110
10	Addition	`a + b`	0110 + 0010 = 1000
11	Subtraction	`a - b`	0110 - 0010 = 0100

After successfully completing this lab, you should be able to:

- Model a non-trivial combinational circuit in VHDL, and implement it on your FPGA.
- Use the iCE40's internal pullup resistors in place of external circuitry.

2 Prelab

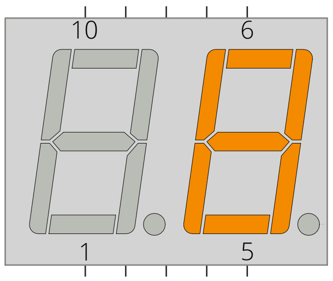
P1: Look at the datasheet for the LTD-4608JR 7-segment display contained in your lab kit (posted on the course website), and determine which segments (A-G) should be on for each hexadecimal digit 0-F. It will help to write this out like a truth table:

digit	A	B	C	D	E	F	G
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
…	…						

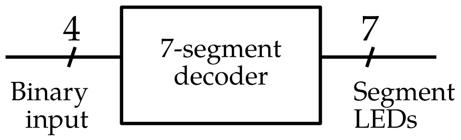
The textbook describes how to create a 7-segment display decoder for the digits 0-9, but you should extend this so you can display numbers up to 15 in hex (You'll want to use lowercase b and d, to avoid confusion with 8 and 0.)

[1]Impractical given the time constraints of the lab — computers really were built this way fifty years ago!

P2: Label the pins on the diagram below according to which segment they control (A-G). Pin 1 is the bottom left, and they are numbered sequentially counter-clockwise (just like the logic chips we've used before).

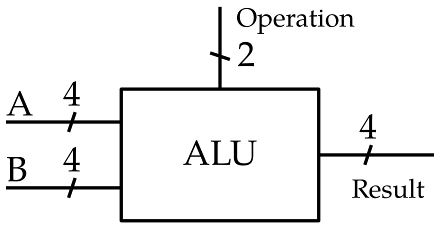


P3: Write VHDL code for the seven-segment display decoder. It must take a 4-bit binary number as input, and produce a bundle of 7 output bits which drive the display. Counterintuitively, the output should be a `0` for the LED you want to turn on (to understand why, see L6 below).



You can complete this on `vhdlweb.com`; no need to copy it into your prelab submission.

P4: Write VHDL code for the ALU. It must take two 4-bit operands and a 2-bit signal specifying the operation to perform. The output is 4 bits (which means that any result over 15 will overflow).



You may have already have completed this on `vhdlweb.com` in class. Again, there is no need to copy it into your prelab submission.

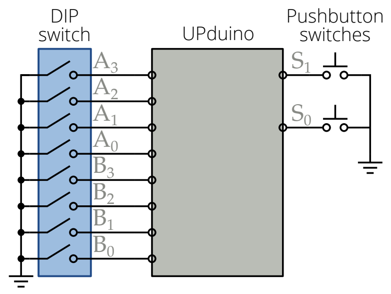
P5: Read through the rest of the lab handout. Describe how you are going to test your work in L4 and L6, and what you will record to demonstrate that your circuit behaves correctly in all cases.

You're welcome to get a head start on the lab before you come. You should already have everything you need in your lab kit!

3 In lab

This is by far the most complex thing you've built in lab so far, so you'll need to be methodical as you build and diligent to test each piece as you complete it. Building the whole thing without testing and hoping that it'll work is a good way to spend hours in frustration. Just speaking from experience.

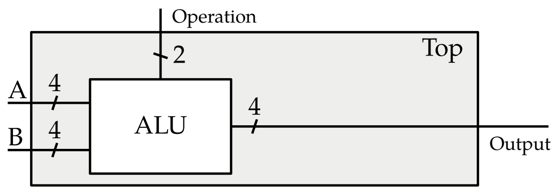
L1: Wire up the DIP switches and pushbutton switches to the UPduino according to the schematic below. You can use any of the pins you'd like. Note that you don't need pull-up resistors, because the iCE40 FPGA has pull-up resistors built in that you can enable in Radiant.



L2: Create a new project in Lattice Radiant (instructions are in the Radiant guide on the course website) and create a file for your already-written ALU module.

L3: Create a new file (called `top.vhd` or something similar) for your top-level VHDL module, which will instantiate the ALU and seven-segment decoder as submodules. See the VHDL reference sheet and the links on the course website for details of how to use one module inside of another.

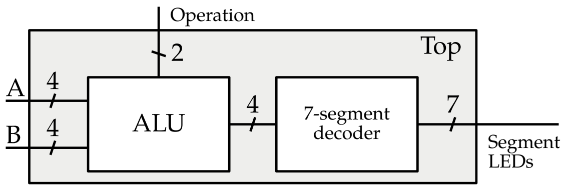
For now, just start with the ALU; you will add the seven-segment decoder later (remember that part about testing things a little at a time?) Your design should look like this:



It may seem a little silly that your top module doesn't do anything yet except act as a wrapper for the ALU. But by doing this, you can focus on testing the ALU, while making it easy to keep your design modular as you expand it.

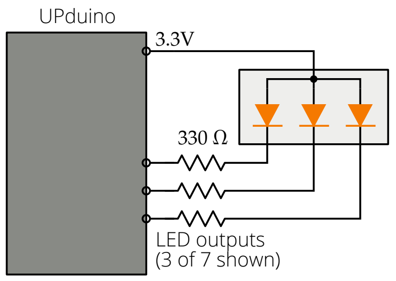
L4: Test your ALU by toggling the DIP switches and pressing each combination of buttons. You can wire up some LEDs or use the Digital Discovery to check the outputs. *Make sure to document your process and any intermediate results!*

L5: Add your 7-segment driver to your top-level VHDL module, and connect its input to the output of your ALU. Update your pin mappings as necessary.



L6: Wire up the 7-segment display according to your prelab work. The UPduino outputs should be connected through a 330 Ω resistor to the cathode (negative/output side) of the LEDs. *Note that this means that the FPGA pin must be 0 (low) to turn the LED on.*

WARNING: You must use resistors between the LED and the FPGA. If you connect the LED directly to the FPGA pin, you may burn up the LEDs or permanently damage your FPGA.



The hard part is making sure that your LED pins are connected to the correct ports. One way to simplify this process is to plug in one LED at a time and toggle through each of the numbers to confirm that the segment turns on and off for the right numbers.

L7: If you've tested carefully and worked out the bugs, you should be able to perform any of the ALU operations and see the answer on the 7-segment display. Take a moment to celebrate, and record whatever test results you'll need for your lab report.

4 What to turn in

Your lab report should contain:

- Standard "front matter" (see the lab reports handout).
- A photograph of your completed circuit.
- A description of your design. At a minimum, this includes a block diagram and an explanation of the inputs you used for the display. You may reference the VHDL code in this section, but the VHDL code should not stand on its own. Include any other key information needed to understand or replicate your design.
- Your complete VHDL code, attached to the end of your report.

  Your code must be set in a monospace font (e.g., Courier, Consolas, etc), because... have you ever tried reading code set in Times New Roman? Don't do such cruel things to your TAs.
- Your lab journal, including intermediate tests you performed (including ones that failed), steps you took to debug your design, and problems you encountered.
- Your plan(s) for testing your design, and any test results you recorded.
- Answers to the following questions:

- What was the most valuable thing you learned, and why?
- What skills or concepts are you still struggling with? What will you do to learn or practice these?
- How long did it take you to complete each part of this lab (prelab, the "lab" itself, and report)?