Lab 2: Higher Order Logic

Due May 10th, 2024, 11:59 PM

Minimum Submission Requirements

- Ensure that your lab2 folder in your Gradescope submission contains the following files (note the capitalization convention):
  - lab2_part1.dig
  - lab2_part2.dig
  - lab2_part3.dig

Objective

The objective of this lab is to build higher order logic structure and introduce an overly simple data path.

Breakdown

This assignment consists of three parts:

Part-1: Single ALU Operation ->complete *lab2_part1.dig*

Part-2: Display Decimal from 2's Complement ->complete *lab2_part2.dig*

Part-3: Data Path ->complete *lab2_part3.dig*

External Resources

Registers, Flip-Flops, and Modular Design This short video provides a very brief introduction to storage elements in digital logic. We will be utilizing their principles in lab2. If by now, storage elements have not yet been covered in lecture, this quick video will give you a primer in this subject. After watching this video, please go through this note on registers to ensure you have the working knowledge of storage elements needed for lab 2.

Lab Demos

You may approach the resident TA/tutor in your lab section to show you a demonstration of how these lab parts should execute should you be successful in correctly implementing it.

Tests

Each interface file provided contains a Test Component. These are not extensive tests. Rather, these tests serve as examples you may wish to follow when implementing your own tests. Your tests will not be graded but we highly encourage you to test your circuits exhaustively to ensure proper functionality.

Specifications: Part-1

Part-1: Description

In this part of the lab you will build a single operation ALU. This ALU will implement a **Bitwise left Rotation**. ***For this lab assignment you are not allowed to use Digital's Arithmetic components.***

IF YOU ARE FOUND USING THEM, YOU WILL RECEIVE A ZERO FOR LAB2!

The ALU you will be implementing consists of two 4-bit inputs (named inA and inB) and one 4-bit output (named out). Your ALU must rotate the bits in inA by the amount given by inB (0-15).

Part-1: User Interface

You are provided an interface file lab2_part1.dig; start Part-1 from this file. **You are not permitted to edit the content inside the dotted lines rectangle.**

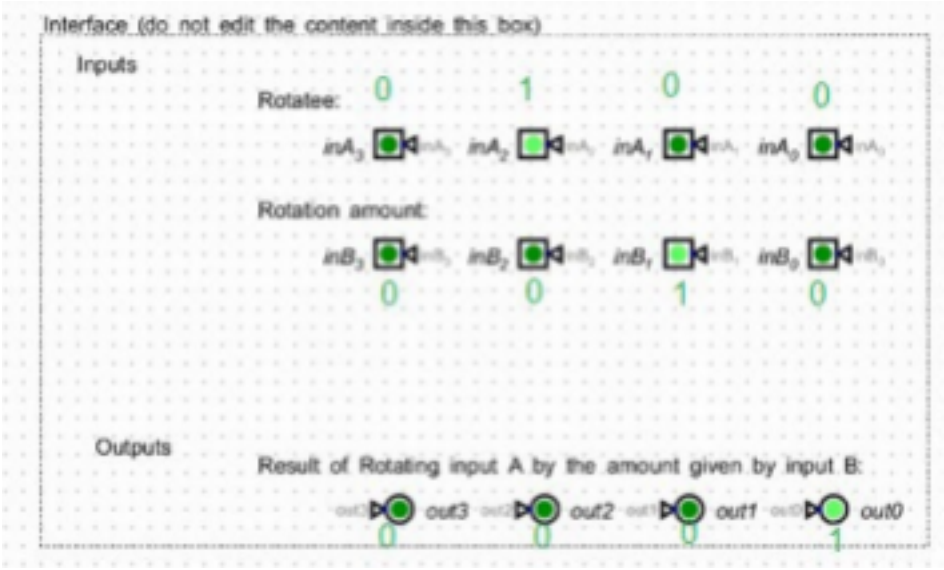

Figure: lab2_part1.dig Interface

Part-1: Example

In the figure above, the input values that we have selected to test are inA = {inA_3, inA_2, inA_1, inA_0} = {0, 1, 0, 0} and inB = {inB_3, inB_2, inB_1, inB_0} = {0, 0, 1, 0}. Therefore, we must rotate the bus 0100 **bitwise left** by 0010_2, or 2 in base 10, to get {0, 0, 0, 1}. *Please note that a **rotation left** is not the same thing as a **shift** left (which many newcomers are apt to confuse)!*

When you open the lab2_part1.dig file, you will of course not see these exact values in green because that is what we used to verify your circuit correctly working by clicking on the Simulate button. On your end, you should go through all possible test cases to verify your design works. **Make sure to name all the wires and input/outputs correctly so that they match the test interface names within lab2_part1.dig.** You will need to do the same for part2 and part 3 as well.

Hint: *Using Multiplexers will help you out a lot here.*

To help you make sure you are testing your rotation results left, you can try using this C++ program for verification. Modify the values of the array arr elements and the rotation value rotateBy. If you don't have a C++ compiler on your local machine, try running the code in onlinegdb.

The "Tunnel" component in Digital for Lab2

In Lab1, you might have noticed how much effort it took to draw so many inputs and wire them all over the place without unintentionally connecting them to the incorrect outputs and nodes! Turns out, Digital does provide some means to help reduce this burden.

This manifests as the Tunnel component. Go to Components->Wires->Tunnel. The symbol looks like a

triangle with one vertex being a bleft dot. The idea is that instead of extending your wires physically all over your circuit drawing space by drawing them as lines, you can use tunnels as placeholders in different locations on your diagram ***for the same wire***. These tunnels must have the same name so as to symbolize the same electrical connection.

You are provided the file playWithTunnels.dig to give you an idea of how tunnels work in Digital. Think of them as being like wormholes that can connect 2 physically separate points on your circuit diagram through the same wire connection.

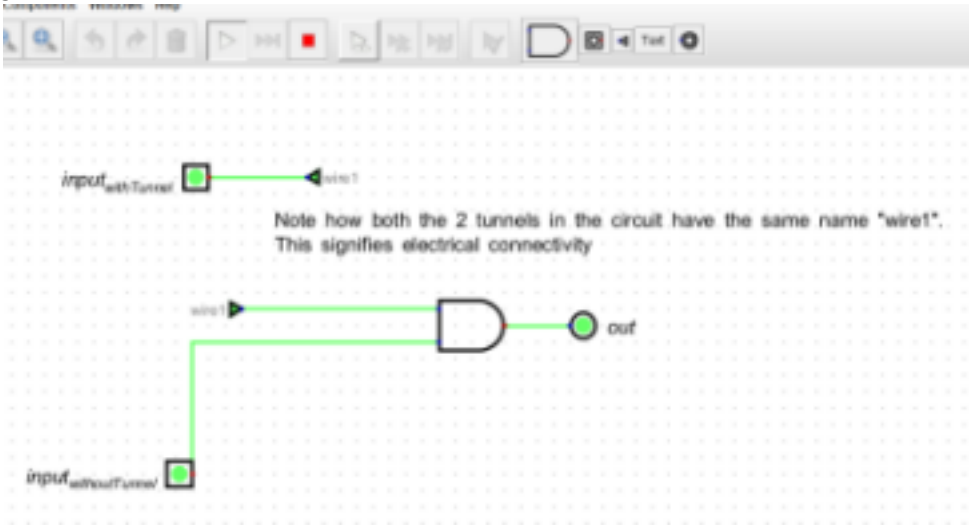

Figure: playWithTunnels.dig *screenshot*

Note that all the Interface areas in the .dig files for part1, part2 and part3 have tunnel components in them. That means when you draw your circuit outside the interface box in these files, you will need to draw the appropriate number of tunnel wires which hook up (via wormholes!) to the provided inputs and outputs within the interface box. **Make sure the tunnels have the correct wire names.**

Specifications: Part-2

Part-2: Description

In this part of the lab, we will illuminate two 7-segment displays. You will need to understand 2's Complement to determine when the input **4-bit binary number** corresponds to a negative or positive number. To understand how an LED display works in Digital, please refer to the playWithLED_Display.dig file provided. You should play with different input combinations to see how it influences the LED Display value. In the screenshot below, note how I was able to generate the display of "3" on the Hex display by lighting up only certain input wires to the unit.

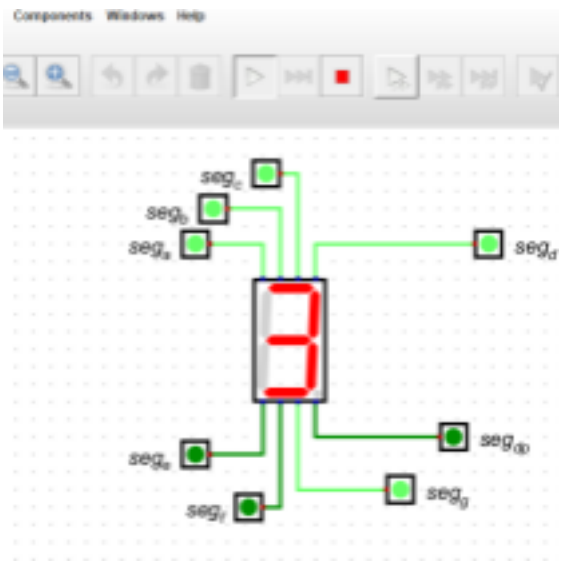

Figure: playWithLED_Display.dig *screenshot*

Here is a picture of how the different segments light up to produce the different displays:

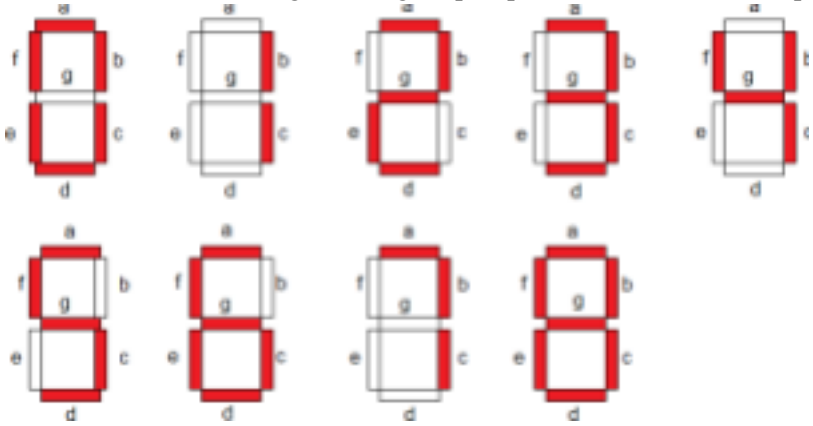

Figure: Lighting up different segments to produce display of 0-8

Note in the picture above that we showed displays only from 0-8 since in 4-bit 2s complement representation, 8 is the largest modulus value you can represent (the range of integers would be -8 to +7).

Your circuit in Part-2 must accept a 4-bit 2's complement input {in3, in2, in1, in0} where in3 is the most significant bit and in0 is the least significant bit. The outputs of your circuit must illuminate the two 7-segment displays such that they display the decimal value of the input. Since this a 4-bit 2s complement represented number, that means the range of numbers (2^4=16) that can be shown correctly on the hex display (with sign) is [-2^{4-1}, $+2^{4-1}$-1] = [-8, +7]

Part-2: User Interface

You are provided an interface file lab2_part2.dig, start Part-2 from this file. You are not permitted to edit the content inside the dotted lines rectangle.

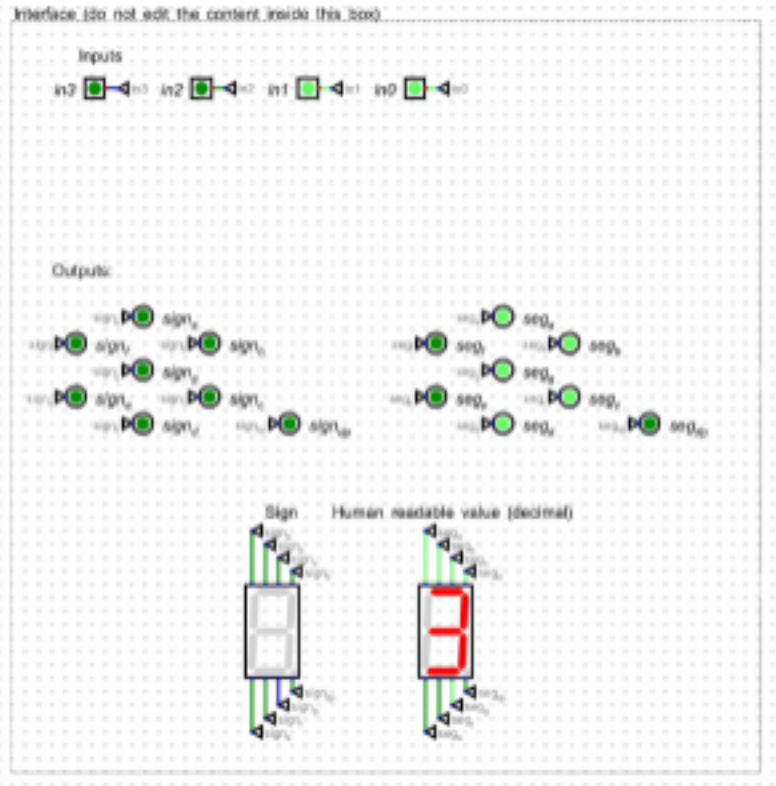

Figure: lab2_part2.dig Interface

Part-2: Example

In the figure above, the input is {in3, in2, in1, in0} = {0, 0, 1, 1} Which means the input signal in, as a bus, reads 0011. Which is the 2's complement binary representation of the value +3. **Note:** only the middle segment, sign_g, (it
appears as $sign_g$ on the Digital screenshot) on the left display will be illuminated, and that too ONLY for negative values. Since +3 is a positive number, it is not illuminated in the above example. All the remaining sign signals remain turned off for both positive and negative numbers.

Specifications: Part-3

Part-3: Description

In this part of the lab, you will create a data path for the ALU you build in Part-1. This data path will consist of 4 registers.

In this lab, we are using the register components. Refer to playwithRegister.dig to get a good starting idea of how this component works.

You will need to address 1 register via the interface select signals to determine which 4-bit register to write the input value to. Then using the D-Flip-flops in Digital.

You will use only one Clock Input to keep the circuit synchronized. That is, with Clock=0, set up your register write values. Once the values are set up, set Clock to 1. For this lab, a manual setting of clock signal from 0 to 1 is needed. Do not create a periodic clock signal.

Part-3: User Interface

You are provided an interface file lab2_part3.dig, start Part-3 from this file. You are not permitted to edit the content inside the dotted lines rectangles.

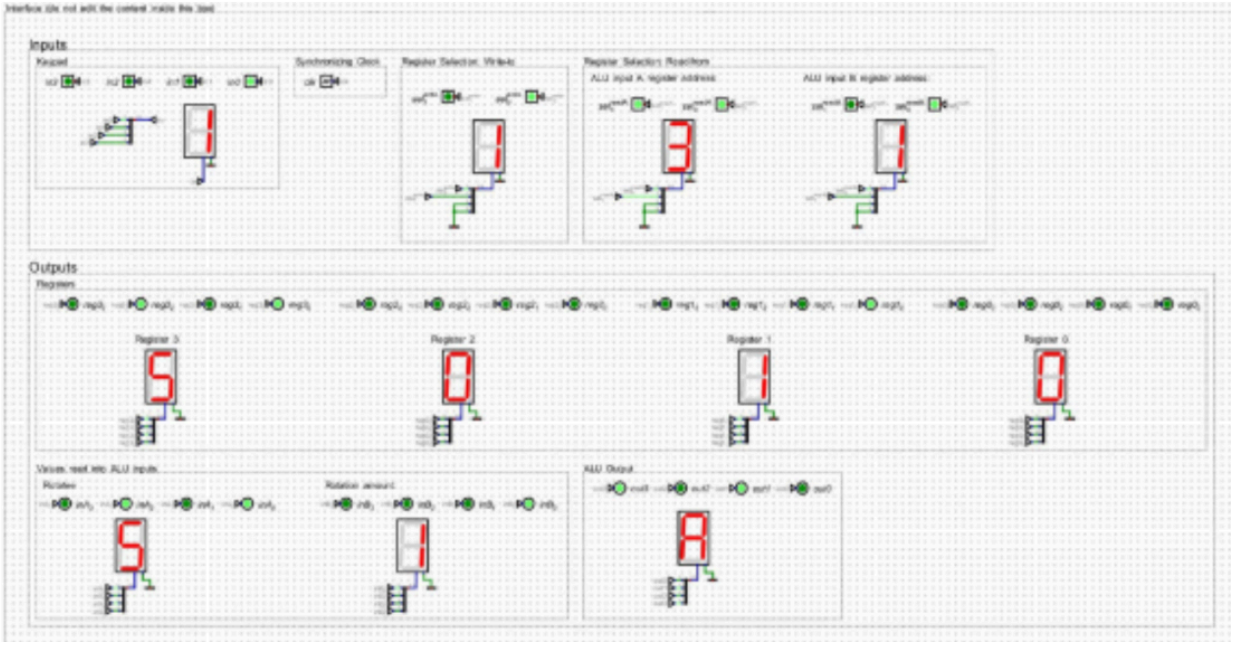

Figure: lab2_part3.dig Interface

Part-3: Example

In this design, the user can choose out of the 4 registers where to store the 4-bit value which needs to be rotated (inA from part-1) and also which register to store the rotation amount (inB from part-1), using the sel_{write} register selection inputs. The user ensures they direct these two registers to the correct Register Selection: Read From values for inA
and inB inputs values to the ALU.

In the figure above, we have written the value 5 to Register 3 and the value 1 to Register 1. Then, we read from Register 3 the value 5 to ALU input A and read from Register 1 the value 1 to ALU input B. Rotating 5 left by 1 b it results in {1, 0, 1, 0} displayed in the ALU Output as hexadecimal A.

Grading Rubric (total 100 points)

30 pt Part-1 completed
30 pt Part-2 completed
40 pt Part-3 completed