Lab 2: Left Shift 32

9/1/2022

1 Possible Point

Add Comment

8/28/2022

Details

Task: In this lab, you will work towards getting a working 32 bit LeftShifter circuit.

Collaboration Policy: As with all labs, you are welcome to get help from fellow students or any 3410 staff member. However, the purpose of this lab is to still further familiarize yourself with Logisim (which you will need to use for Projects 1 & 2), so we want each of you working on your own circuit. *If you have questions, please ask them in your Lab Section!*

Note: You are graded based on attendance and effort rather than on correctness.

For this lab, there are 3 designated **Checkoffs**.

Overview

In the first two projects of this course, you will design a subset of the RISC-V architecture. The goal of these projects is to move you from designing small special-purpose circuits to building complex, general-purpose CPUs. By the end of the second project, you will have designed a 32-bit RISC-V CPU. For these assignments, we will ignore more advanced features, including the RISC-V coprocessor instructions, how to implement pipelining, and traps/exceptions **but**, you will touch on some of these topics in lecture and will be expected to understand the topics that are covered.

In your first project you will design a RISC-V ALU (arithmetic and logic unit), which performs all of the core computations dictated by the assembly language. In this lab, you will create a subcircuit that will ultimately be a part of your RISC-V ALU: the LeftShift32.

Logisim comes with libraries containing basic gates, memory chips, multiplexers and decoders, and other simple components. In the aforementioned future projects, you will use many of these components to build your final CPU.

However, for this assignment you may only use the following Logisim elements:

- Anything in the **Wiring** folder **except** for the resistor, constant, power, ground and transistor elements.
- Anything in the **Base** folder (wires, text, etc.)
- Anything in the **Gates** folder **except** for the even parity, odd parity, and controlled buffer elements.
- Anything in the **Plexers** folder.

****Important** - Use the **Logisim design guidelines (https://canvas.cornell.edu/courses/42905/pages/logisim-design-guidelines)** for designing your circuits to avoid losing points on projects in the future!**

Circuit 1: LeftShift32

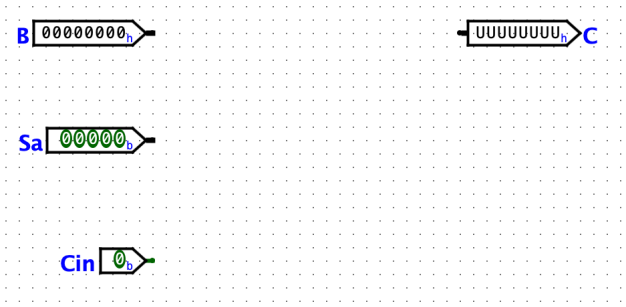
LeftShift32:	C = (B << Sa) | carrybits
Inputs:	B[32], Sa[5], Cin
Outputs:	C[32]

Task 1: Getting Things Set Up Correctly

(https://canvas.cornell.edu/courses/42905/modules/items/1511783)

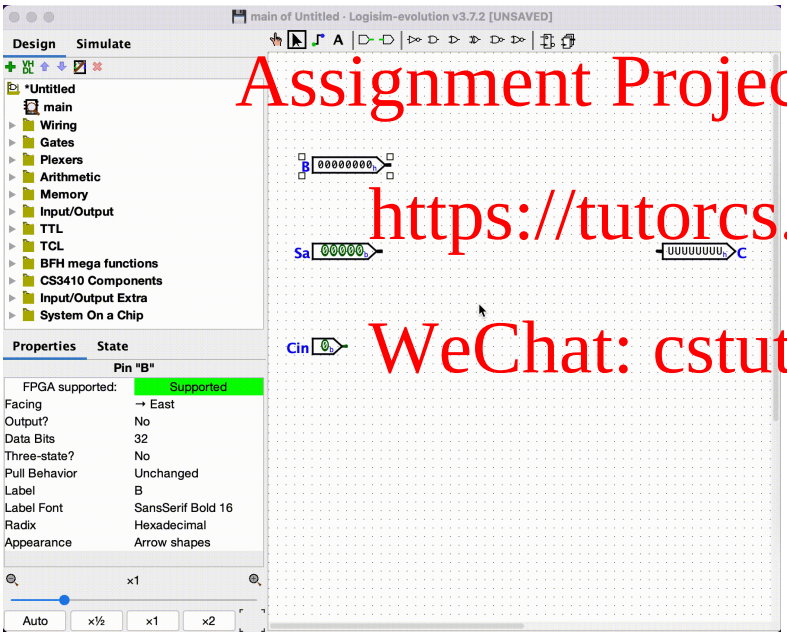
(https://canvas.cornell.edu/courses/42905/modules/item

correct, it should look something like this (if it doesn't, please let a TA know immediately):



Notice something weird? The input pin **B** and output pin **C** are displaying 00000000h even though the specs state that they are 32 bit pins, is something wrong here? Fortunately (or maybe rather unfortunately for the majority of you as this is the first time you've come into contact with circuit designing software and the last thing you want is more **fun** quirks), the Logisim software isn't broken.

For those curious as to what is happening, the subscript **h** is denoting that the value of the pin is being displayed in hexadecimal (each digit corresponds to 4 bits). However, for the purposes of this lab, hexadecimal will probably not be very helpful to you so, let's change it to binary for easier testing purposes down the line. Go to the properties section of the pin after clicking on it, locate the **Radix** option, and set it back to binary.



Checkoff #1: Show your newly customized template circuit to your TA

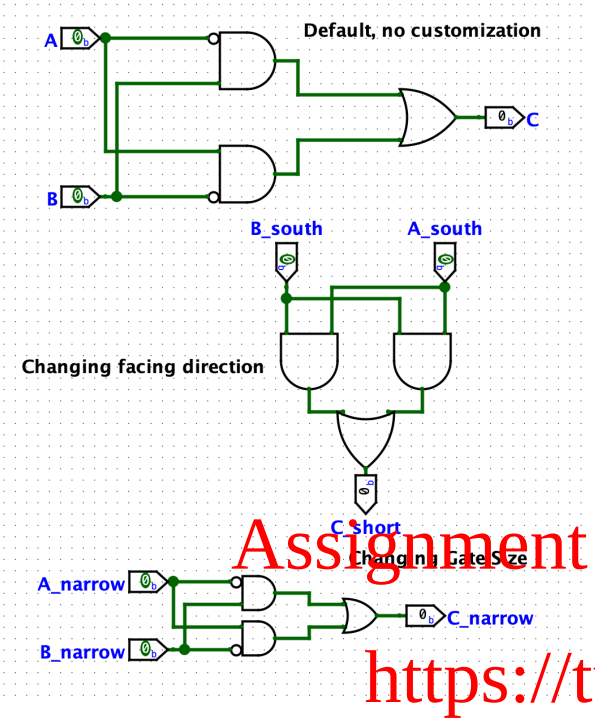
Task 2: 32 Bit LeftShift-ing

To break down the above specification for our Leftshift into prose, the output C is computed by shifting B to the left Sa bits, and filling the vacated bits on the right with "carry-in-bits", which are just **Sa** copies of **Cin**. The shift amount **Sa** can be anything from 0 to 31, encoded as an **unsigned** integer.

To get you started, one way to implement such a shifter is to perform the shift as a series of stages: the first stage shifts either 0 or 16 bits, the second stage either 0 or 8 bits, the third stage either 0 or 4 bits, and so on. By enabling different combinations of stages the circuit can shift any desired amount. Think back to our first lab where we built the **Add4** circuit and the steps we took to get to that point, as you might find that a lot of the necessary sub-circuitry needed for your ALU

Hint: Shifting a value on a 32-bit bus, by a constant amount, either left or right, is simply a matter of adding, removing, and renaming wires, and so requires no gates at all.

As you do this task, you might start finding your circuit design a little cumbersome or hard to manage. For this, you might find it helpful to rotate or change the appearance of your gates and circuitry through clicking on any individual part and accessing the **Properties** section. I recommend looking at **Gate Size** and **Facing** to help clean or better customize your circuit.



On specifications

It is important that your implementation of the circuit described above adheres to the specification in this document. Adhering to specification is important in most all design processes, and it will also have a more immediate effect on your grade for projects (you will find LeftShift32 to be extremely useful in P1 in particular). Automated grading will expect that the circuit above (and their inputs and outputs) are named **exactly** as specified (case-sensitive!) and behave exactly as specified.

Recall that when the specification denotes a pin as A[32], the pin should be named "A" and be 32 bits wide. The pin should not be named "A[32]".

To Drive a Point Home

We highly recommend you take some time to save your work now before moving onto the Test Vector section of this lab. To reiterate from the previous lab, the autosave feature will **NOT** go into effect without your initial manual save. You never know when this habit might save you from having to redo hours of work.

Checkoff #2: Show your LeftShift circuitry to your TA

Test Vectors

Extensively testing your circuit is important to ensure correctness. Logisim, luckily, allows the use of test vectors for automated testing of circuits.

While it is not feasible to test every possible input vector, it is feasible in Logisim to test up to several thousand input

complete functionality of your circuits. Some of your tuples might be written by hand to test corner cases (e.g. adding combinations of zero, +1, -1, max_int, min_int, etc.). Some might be generated systematically (e.g. testing every possible shift amount), and others might be generated randomly (to catch cases you didn't think of). Testing is an important part of these projects and will be graded.

For this lab, you will be required to write a test vector for your **Left Shifter**. Since the Left Shifter will be an important component in the ALU you are designing, you should create a test vector to ensure that it is working correctly. You can find an example of a test vector for your LeftShift32 on the assignment page. You can download it at this link: **LeftShift_test_vector_example (https://canvas.cornell.edu/courses/42905/files/6057404/download?wrap=1)** . This test vector is not holistic and does not cover edge cases you should consider. Therefore, use this as a reference but not as a thorough replacement.

Example Test Vector:

```
#First line labels input and output pins, labeled with [bit width] if >1
#Each line after is an individual test with each column representing a pin
#Numbers can be in decimal, binary, hexadecimal, or even octal if you want

B[32]                                 Sa[5]   Cin   C[32]
34                                    2       0     136                                   #Decimal
325948595                             15      0     -900104192                            #Negative Decimal
00000000000000000000000000000001      00100   1     00000000000000000000000000011111      #Binary
0x00000005                            0x01    0x0   0x0000000A                            #Hexadecimal
0o17777777777                         0o1     0o0   0o37777777776                         #Octal
00011111001010100010110111110001      0o10    0x0   707653888                             #Everything together!
```


Once you have created your test vector text file, you can run the tests automatically within Logisim, by accessing Simulate --> Test Vector --> Load Vector and selecting your test vector file.

Note: We recommend you look back to this section when working on your test vectors for both P1 and P2 if you are having trouble figuring out what is considered "good" testing.

Checkoff #3: Show your test vector to your TA

Now you have a (hopefully) fully functioning 32-bit LeftShift circuit! I think a nice round of applause is in order.

[prev] **Lab 1: Intro to Logisim** **(https://canvas.cornell.edu/courses/42905/assignments/381696)**

[next] **Lab 3: LEDs & Minimization** **(https://canvas.cornell.edu/courses/42905/assignments/381698)**

(https://canvas.cornell.edu/courses/42905/modules/items/1511783)

(https://canvas.cornell.edu/courses/42905/modules/item