Dillon Dragomir

Philip Wilsey

EECE 3026 - Computer Architecture and Organization

21 January 2018

**Computer Architecture**

**Logisim**

Logisim has many things going for it. One being it can be ran on the JVM and is essentially cross platform. I first tried to use the build for Mac, but it wanted me to use an older version of Java, so instead I decided to use the straight jar file. Otherwise the installation was fairly easy.

One of the pros of Logisim is that it allows you to use sub circuits. This allows developers to create a component then use it repeatedly and abstract away some complexity. Sub circuits also make the additional task of doing bitwise operations for say a 16-bit architecture much easier to accomplish. Another pro of Logisim, is it allows you to combine wires together into a bus wire and carry multiple bits and then use a splitter component to separate out all the separate bits.

The only real con I can see with Logisim may be that the components it has to work with are limited compared to like a Multisim and that wires can only cross perpendicular. Wires can be combined as bus and carry multiple bits and this does abstract complexity enough to decrease wire complexity.

**CEDAR Logic**

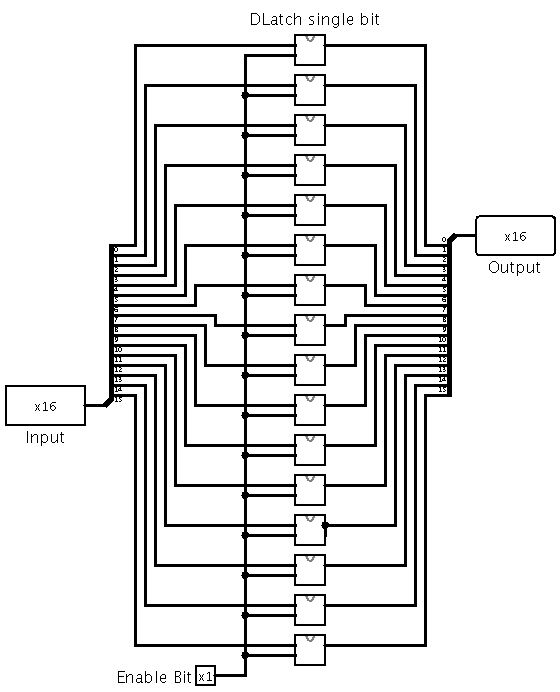
CEDAR logic falls short in what seems every area I have tried using it. First, there may be some way to build it for Mac, if I were to compile it myself, but no easy solution. So if you are on a Mac you will have to run a virtual box on your machine. An alternative solution, which was also easier for me, is to spin up an EC2 instance of windows 10 and remote into the instance using Microsoft Remote Desktop. I got this setup running with CEDAR Logic in about 15 minutes, but for someone not familiar with pem files or AWS this could take an hour or so.

As for the positives, the simulation seems fairly easy. I also found it very convenient that CEDAR Logic automatically connects wires when you drag between two components. Finally, and this is something you should drastically not overlook, this software is free and in many instances, you get what you pay for.

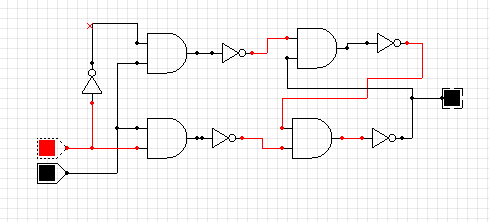
The cons list is much longer. There is no support for sub-circuits and the multi-bit inputs kept making the program crash, so I had to create an input for every individual bit. This makes abstraction much harder to accomplish under our current constraints, but if you are going to use the components already included in the software it could still be an invaluable tool. Additionally, running in parallels the delete component functionality does not work. Finally, the amount of times this program crashed on me added annoyance to my workflow and was very inconvenient.

**16-bit latch**

I simulated a D-Latch with load a load enable bit. This circuit can store 16-bits of data and the data is passed from the input into each latch circuit when the enable bit is binary “1”. When the enable bit is binary “0”, changes to the input do not affect the bit being stored in the latch circuit or the output. This circuit was tested and validated to properly store and update the input to the output when the enable bit is on. Also, the export image functionality was not working in my parallel environment for CEDAR Logic so the background has the grid from a screenshot.

****

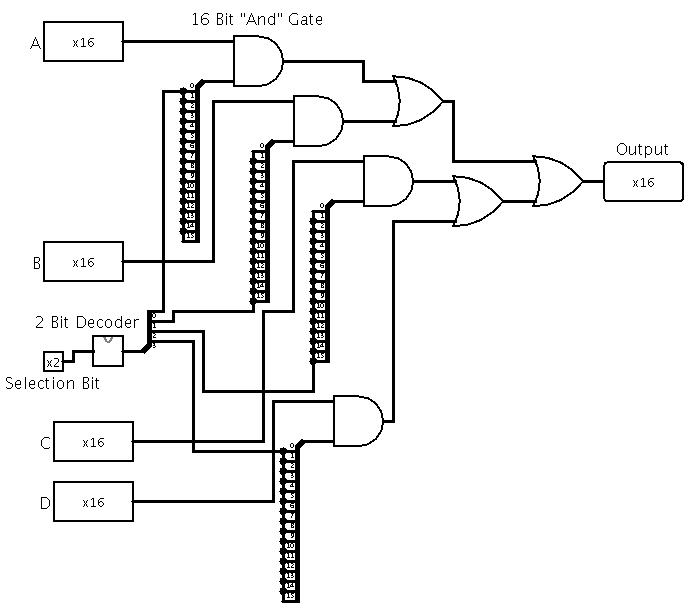
**Figure 1: D-latch 16-bit in Logisim**

****

**Figure 2: D-latch single bit in CEDAR Logic**

**4 x 1 Multiplexer**

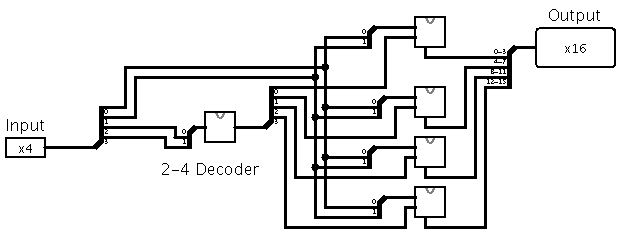
This circuit uses 2 bits to choose the input, which is then passed on to the output. This circuit was tested and validated to properly pass through one 16-bit input based off of a 2-1 decoder input and pass that to the output. Refer to figure 5 to see an example of a 2-bit decoder circuit. The two bit decoder circuit is used to turn on the “and” gate and pass through the given input. The “or” gates then pass through the input which is on to the output. In this circuit, I used components that allow for 16-bit operation and then just split out the enable bit so that it enables all of the 16-bits from the input. Finally, I realize, in a circuit where gates cannot do operations on a 16 bit number, it would be required use of a 16-bit architecture or logical “and”/”or” components from the ALU to do the 16 bit operations.



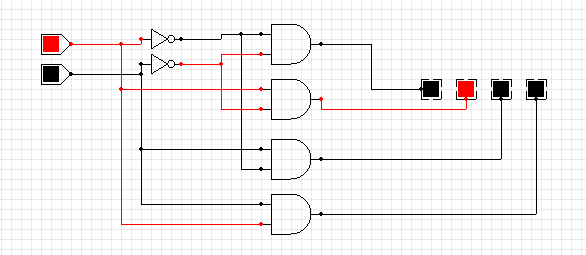
**Figure 3: 16-bit 4x1 multiplexer in Logisim**

**4-16 bit decoder**

For this circuit, I used 5 2-4 bit decoders that each had an enable bit. Since Logisim allows for spliiters, I used splitters with all of my components and fed in a single input with multiple bits and then split out the indiviual bits when necessary. The output of the first decoder is fed into the enable bit of the second layer of decoders and logic “0” forces output of that decoder to be all zeros. The first two inputs are then used on the input of the second layer of the decoder to chose which bit out of the last four bits is the one to be enabled. This design reuses the 2-4 decoder that was used in previous designs. The circuit displayed in figure 5 is the individual 2-4 bit decoder that was created in Logisim and CEDAR Logic. Each of the small boxes in figure 4 is a decoder from figure 5.



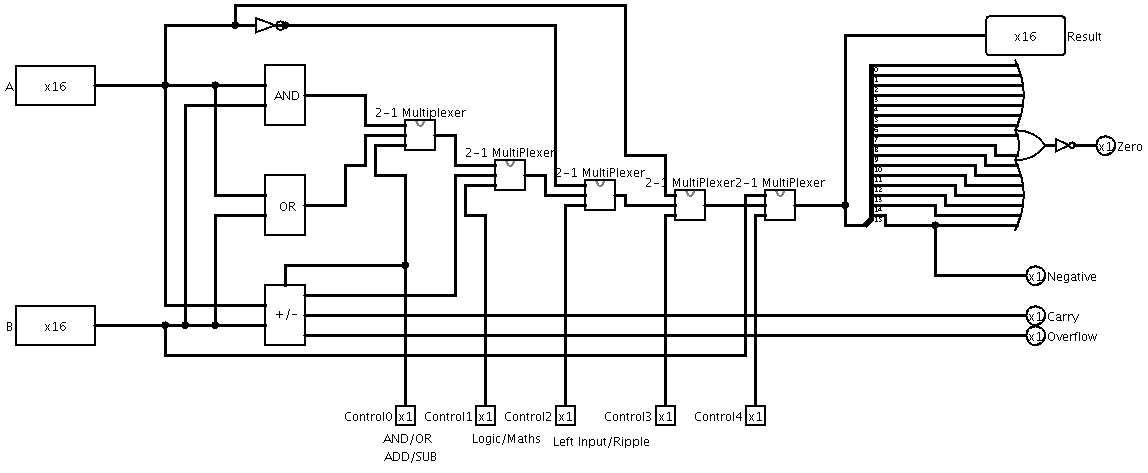
**Figure 4: 4-16 bit decoder in Logisim**

****

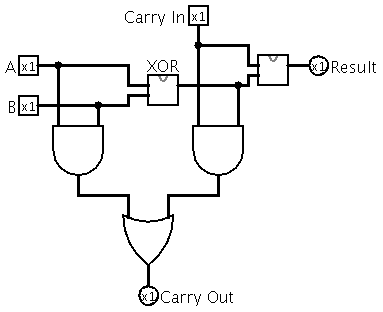
**Figure 5: 2-4 bit Decoder in CEDAR Logic**

**ALU**

ALU stands for arithmetic logic unit and it does logical and math operations. This ALU is fairly simple and has a few main functionalities. It has logical “and”, “or” and “not” functionality. Additionally, it has an adder and subtracter that uses 2’s compliment for subtraction and implements overflow and carry out bits. At a lower level the adder works off of a premise that is called a ripple carry adder that does single bitwise addition and then the carry bit is outputted to the next single bit adder and ripples through the circuit until all bits are added. The XOR that was used in the ripple adder was created using the specified gates. And also to do 2’s compliement in the adder when subtraction is enabled. Finally, all of the inputs from these logic and math modules are fed into multiplexers, which take the inputs and feed them to the output based on inputed bits. Since this ALU utilizes 2’s compliment the leftmost bit must be checked for logical “1” and determines whether the number is negative or positive. The overflow bit tells us whether the numbers have overflowed and thus cannot fit into the 16 bit output.

****

**Figure 6: ALU in Logisim**

****

**Figure 7: Single bit adder**