Digital Didgeridoos
Team Project

“THE CLASSIC”

CHASE CLOUTIER
JON IBARRA
KHA NGUYEN
ROLAND SMITH
NEUNZO THOMAS

~~Draft~~ Description:

We will be working on "The Classic", which is to create a full ALU with an accompanying full datapath. The ALU we will be creating will need to be able to add, multiply, subtract two numbers, while also remembering the result to do an operation on a new input. This is just like a calculator function, as any simple calculator will be able to remember previous results to do another operation to a new input. However, for our ALU we will be using 8 bit integers for our inputs, as well as a ~~2 bit~~ **3 bit** opcode input. The opcode input will let us decide which operation to execute. Additionally, the ALU must be able to do AND, OR, NOT, and XOR on the numbers as well. Finally, the ALU must also be able to store the last result until cleared. ~~A cleared function will be included as well.~~ **Clearing is performed by using a 1 bit input.**

The whole point of this is to create something akin to a calculator, and an ALU is used in many different things, such as being the fundamental building block of a CPU of a computer.

The ALU will be divided into several parts.

-Adding and Subtracting
-Multiplying
-AND, NOT, OR, XOR
-Combinational Logic for Opcode
-Finite State Machine components

Most of the above sections are simple with the exception of the finite state machine components and the combinational logic for the opcode. The opcode will be ~~resolved with a decoder feeding~~ **fed** into a mux, and the finite state machine will have ~~two~~ **nine** states. The states are going to be ~~initial and result~~ **ready, stored, add, sub, mul, AND, OR, NOT and XOR**. ~~It states in initial, and if there is two inputs and an opcode, will store the result and move to the next state~~ **When the system is in the ready state, it does not currently have a stored value. When it receives two inputs and an opcode, it will proceed to the correct operation, which is determined by the opcode. The operation will compute the result, output it, and store it for future use**. The ~~result~~ **stored** state **has two options. It can** ~~will~~ take one input and one opcode and use the saved result as another input, and on the execution of a clear command, will reset back to the initial state. **Any mathematical errors that may occur are ignored by the system.**

There will be four inputs and one output.
Inputs: 8 bit Input1, 8 bit Input2, ~~2~~ **3** bit Opcode, 1 bit Clear.
Output: 8 bit Result.

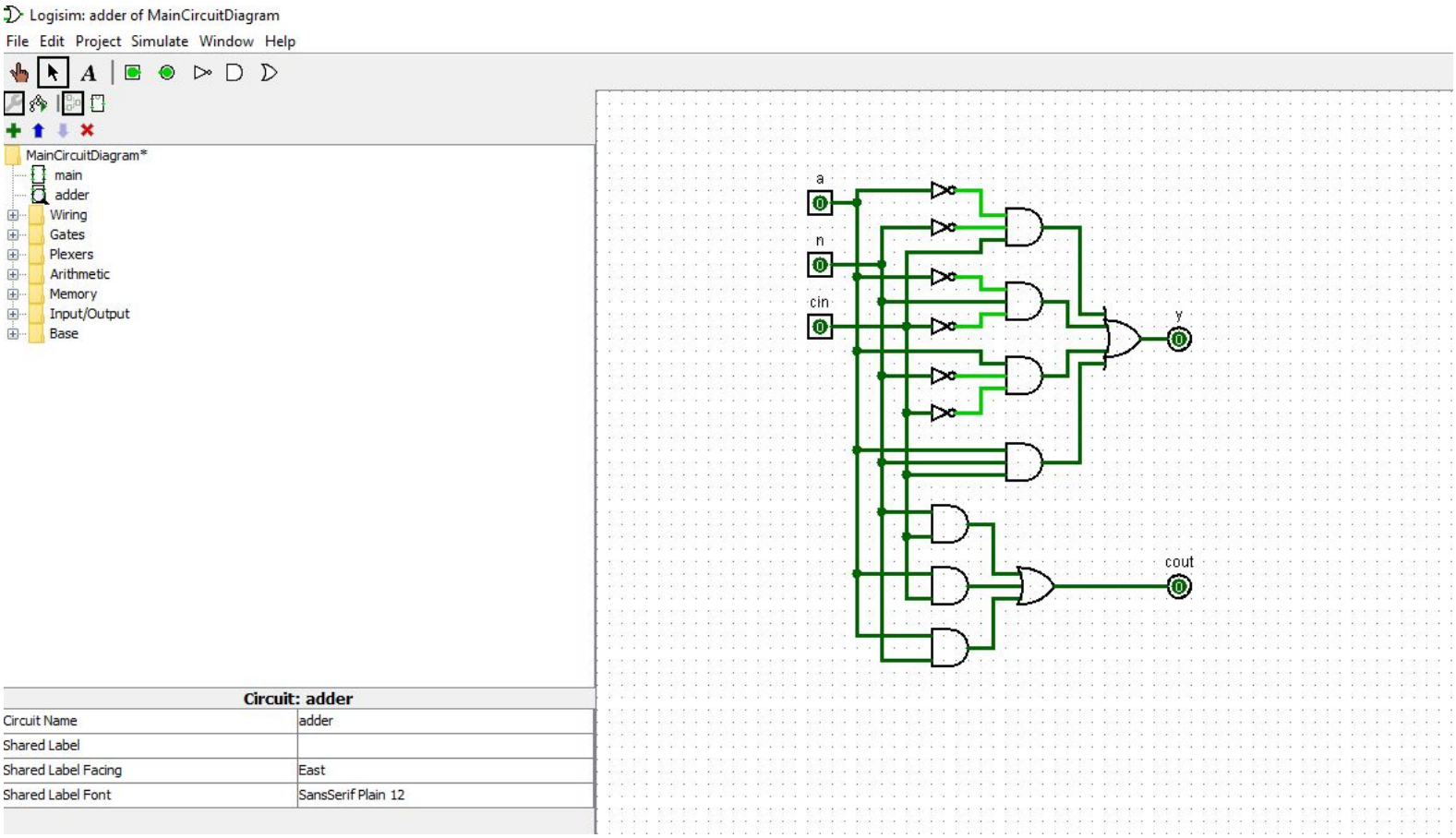
Parts List:
-1 8-to-1 Mux
-3 2-to-1 Mux
-1 Add Module
-1 Subtract Module
-1 Multiply Module
-1 AND Module
-1 OR Module
-1 XOR Module
-1 NOT Module
-3 1-to-1 Flip-flops
-1 Inverter Gate
-1 AND Gate

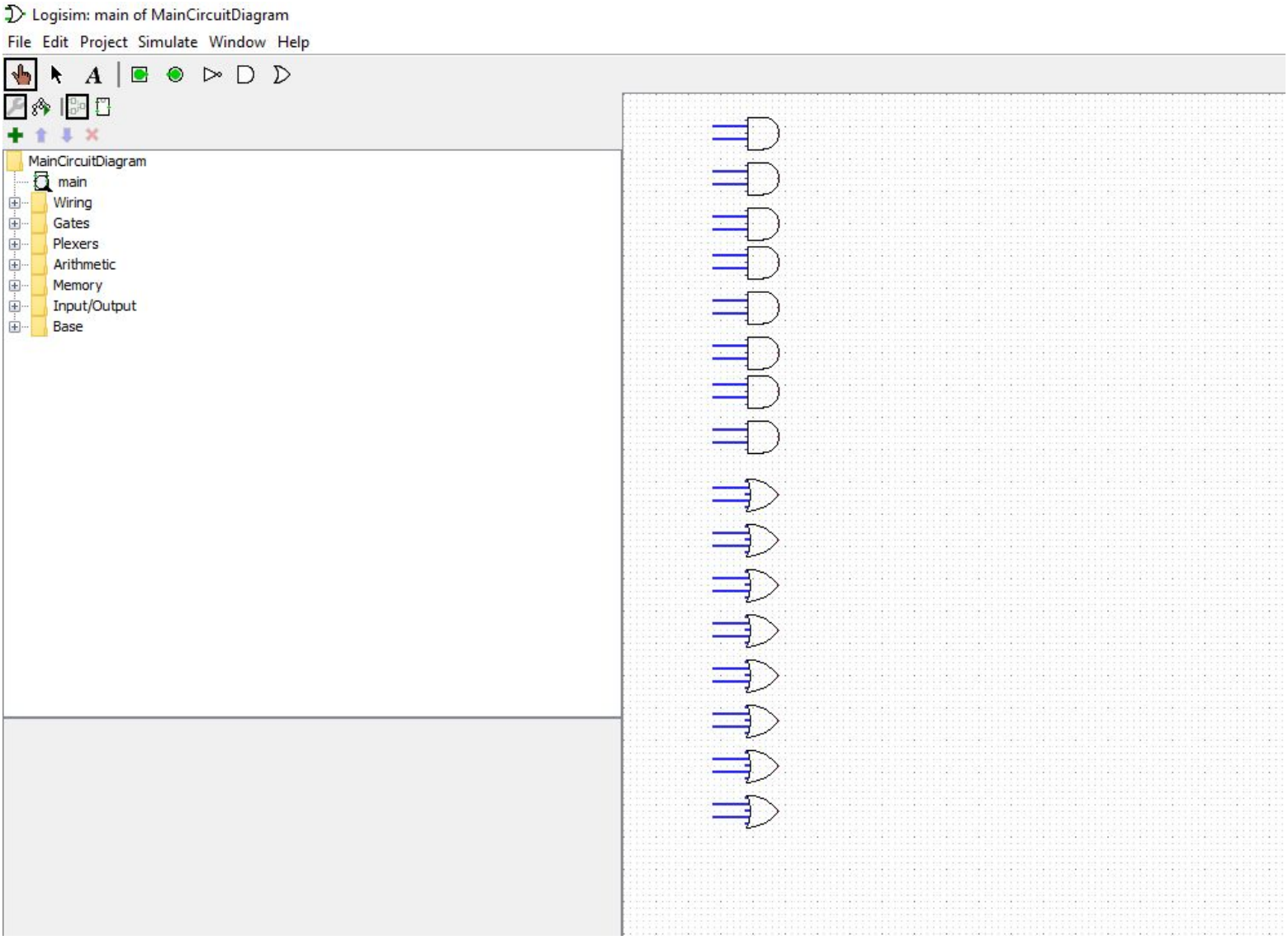
Member Tasks:

Chase Cloutier	AND/OR/NOT/XOR functionality**, Circuit Diagram**	List and define inputs
Roland Smith	Add/Subtract functionality**, Parts List, Input List, States List, Output List, Module List, Register List**	List and define outputs
Neunzo Thomas	Multiplication functionality	List and define modes and states
Jon Ibarra	Opcode selection unit **Input List, States List, Output List, Module List, Register List**	Description of the registers or other components that control the current state
Kha Nguyen	Finite State Machine logic	Draw FSM diagram
All	Building of Verilog code and testbench for output	List and define individual components; combine individual components into final circuit diagram

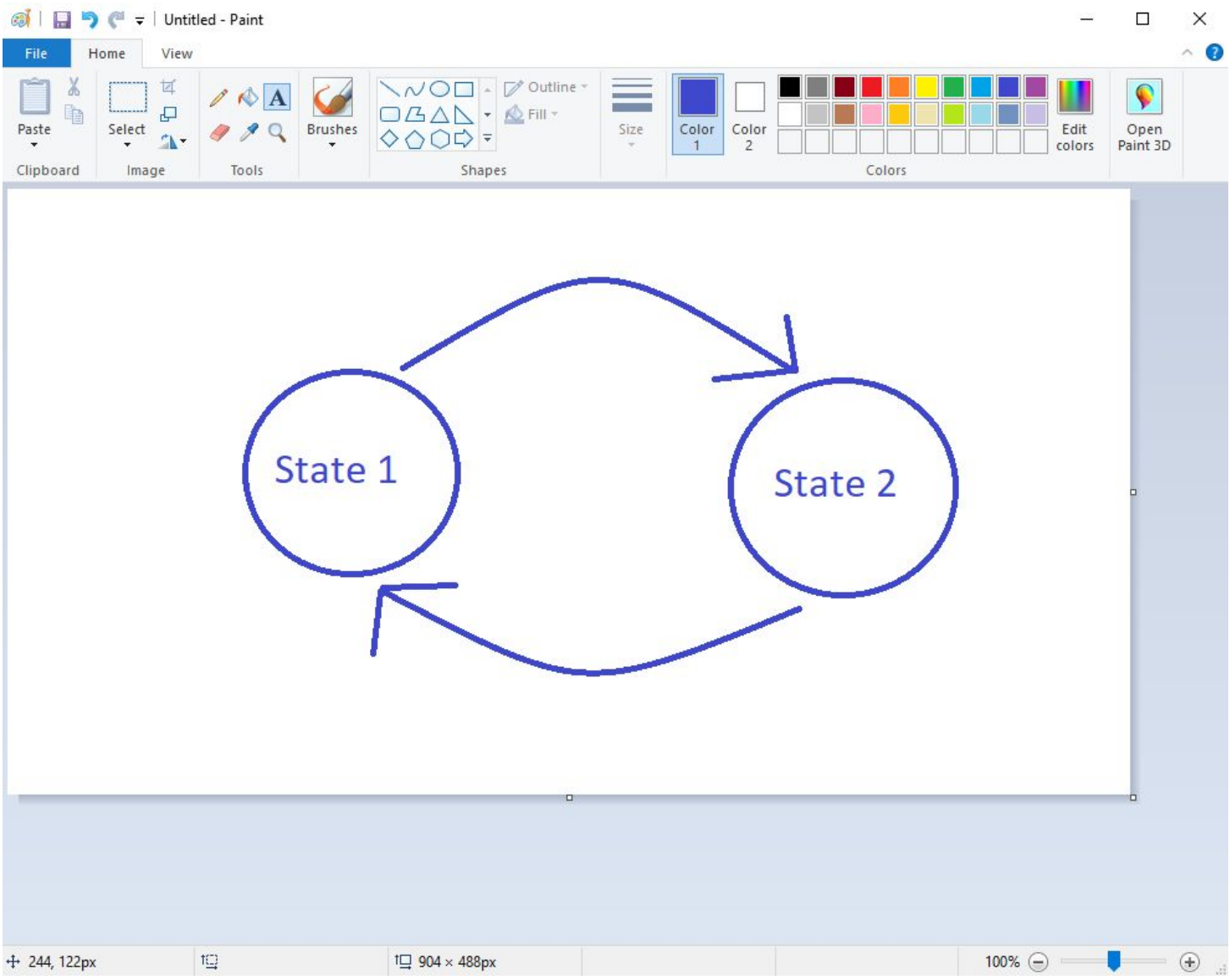
After perusing the market for a free circuit development and drawing software, we decided to settle upon using Logisim. Logisim provides a very simple yet effective interface for creating circuits, along with the ability to minimize a circuit. The top 3 most used logic gates are available to the user on the top bar, which speeds up the process of having to build circuits, while the rest are compartmentalized in logical order in folders to the left. It also comes with a circuit building tool where you can input the truth table or expression and it will build the gates for you using sum of products or product of sums.

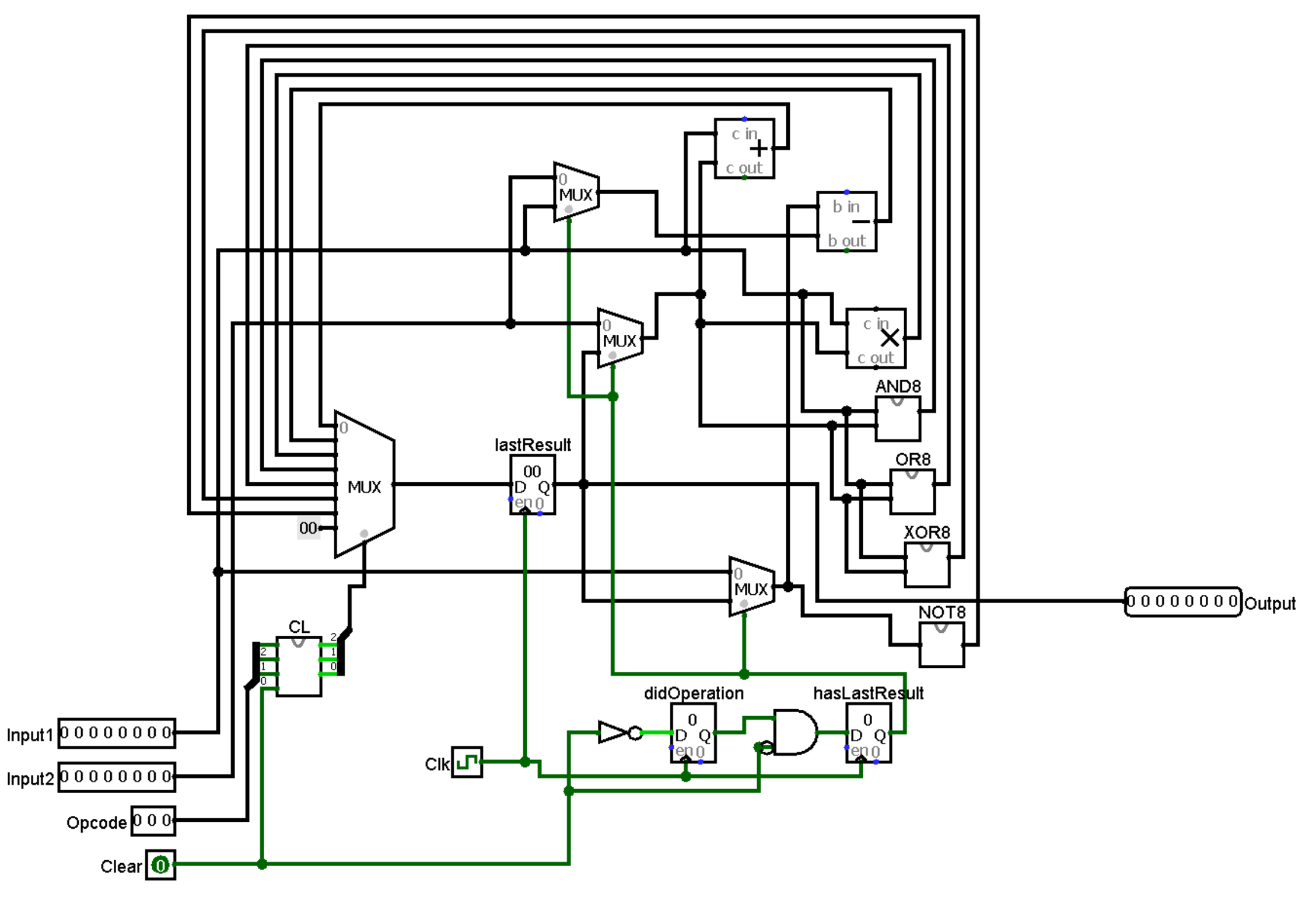
The best part about Logisim was the fact that it was free and available to anyone with an internet connection. The software was downloaded from http://www.cburch.com/logisim/download.html on 10/19/17.





Microsoft's Paint was also used to draw the finite state machine. Paint was chosen since it is already freely available to Windows users, has an extremely shallow learning curve, and is capable of doing everything that was needed. Paint was not downloaded since, as of November 20th, 2017, it is included with all versions of Windows.


Untitled - Paint
File
Home
View
Paste
Clipboard
Select
Image
Tools
Brushes
Shapes
Outline
Fill
Size
Color 1
Color 2
Edit colors
Colors
Open Paint 3D
State 1
State 2
244, 122px
904 × 488px
100%


c in
c out
b in
b out
c in
c out
MUX
MUX
MUX
MUX
AND8
OR8
XOR8
NOT8
lastResult
00
D Q
en 0
00
CL
didOperation
hasLastResult
Clk
Input1
0 0 0 0 0 0 0 0
Input2
0 0 0 0 0 0 0 0
Opcode
0 0 0
Clear
0
0 0 0 0 0 0 0 0
Output

System Descriptions

Modules:
-Module mult8(in1, in2, product, overflow): Multiply two 8 bit integers
Inputs:

- In1 - The first operand for multiplication maximum of 8 bits
- In2 - The second operand for multiplication maximum of 8 bits

Outputs:

- Product - The result from multiplying in1 by in2 maximum of 8 bits
- Overflow- Overflow bit that goes high when product exceeds 8 bit register, low otherwise

-Module add8(in1, in2, result, overflow): Add two 8 bit integers
Inputs:

- In1 - The first operand for addition maximum of 8 bits
- In2 - The second operand for addition maximum of 8 bits

Outputs:

- Result - The result from adding in1 by in2 maximum of 8 bits

-Module sub8(in1, in2, result, overflow): Subtract two 8 bit integers
Inputs:

- In1 - The first operand for subtraction maximum of 8 bits
- In2 - The second operand for subtraction maximum of 8 bits

Outputs:

- Result - The result from subtracting in1 by in2 maximum of 8 bits
- Overflow- Overflow bit that goes high when signs don't match, low otherwise

-Module AND8(in1, in2, result): Bit-wise AND between two 8 bit integers
Inputs:

- In1 - First operand for the AND function
- In2 - Second operand for the AND function

Outputs:

- Result - Result when in1 and in2 are "anded" together

-Module OR8(in1, in2, result): Bit-wise OR between two 8 bit integers
Inputs:

- In1 - First operand for the OR function
- In2 - Second operand for the OR function

Outputs:

- Result - Result when in1 and in2 are "ored" together

-Module XOR8(in1, in2, result): Bit-wise XOR between two 8 bit integers
Inputs**:**

- In1 - First operand for the XOR function
- In2 - Second operand for the XOR function

Outputs:

- Result - Result when in1 and in2 are “XORed” together

-Module NOT8(in1, in2, result): Bit-wise NOT between two 8 bit integers
Inputs:

- In1 - First operand for the NOT function
- In2 - Second operand for the NOT function

Outputs:

- Result - Result when in1 and in2 are “NOTTed” together

Registers:

-Register hasLastResult
Is used to determine if in2 should be used for operations, or what is stored in lastResult
-Register lastResult
Stores the result of the previous operation which can be used in future operations
-Register didOperation
Used to tell the system an operation was performed

States:

-State: On Ready
ALU does not have a stored number and selects operations with fresh inputs
-State: On Add
ALU is currently performing the Add operation
-State: On Sub
ALU is currently performing the Sub operation
-State: On Mul
ALU is currently performing the Mul operation
-State: On AND
ALU is currently performing the AND operation
-State: On OR
ALU is currently performing the OR operation
-State: On NOT
ALU is currently performing the NOT operation
-State: On XOR
ALU is currently performing the XOR operation
-State: Store
ALU has a stored number that can be used in a following operation

Inputs:

Input1: 8 bit integer, is always used during operations
Input2: 8 bit integer, is used if there is not an integer stored in lastResult
Opcode: 3 bit integer, used to tell the ALU which operation should be performed
Clear: 1 bit integer, used to reset the ALU

Outputs:

Output: 8 bit integer, is the output of the ALU's function's output