

Pretty Gate Machine

**Updated** Report

11/16/2018

**Updated Description:** Arithmetic Logic Unit with full data path.

~~As part of our Digital Logic and Computer Design course, we are going to realize a 4-bit Arithmetic and Logic Unit. An arithmetic logic unit (ALU) is a device that performs the basic arithmetic and logic operations of a processor.~~

~~Thus, our ALU can perform a minimum of 10 different operations on two 4-bit inputs, producing a single 4-bit output as well as an indicator of the output status (overflow, sign, zero, output hold).~~

**The final ALU uses a 3-bit input termed the Opcode in Verilog to output one of nine functions. All operations are performed simultaneously, but the Opcode runs through MUX selectors to produce the desired output.**

The main functions of an ALU are:

1. **Basic logical operations.** An arithmetic unit is used to perform the basic logical operations on numeric data (generally on 8 bits):

- logical AND
- logical OR
- logical NOT, and
- logical XOR (exclusive OR)

2. **Arithmetic operations.** An arithmetic and logical unit also makes it possible to carry out the operations on numerical data:

- Addition,
- Subtraction,
- ~~Division, and~~ **Division is not included in the final product.**
- ~~Multiplication~~ **Multiplication is not included in the final product.**

3. **The binary comparison:** An arithmetic and logical unit also makes it possible to compare two numbers by indicating in a state register whether the result is larger, smaller or equal.

4. **The Shifter.** A shifter is formed of  $(n + 1)$  inputs  $D1, \dots, Dn, C$  (control bit) and  $n$  outputs  $S1, \dots, Sn$  and operates a shift of 1 bit on the inputs  $D1, \dots, Dn$ :

- ~~If  $C = 1$ , this is a shift to the right,~~
- ~~And if  $C = 0$ , a shift to the left.~~
- **All operations are performed but are selected to be output near the mux.**

The various tasks of this project are composed but not limited to:

- Coordination and project management
- Development of truth tables
- Simplification of the equations
- Circuits design
- Choice of implementation mode (software discovery)
- Translation of the code to the implementation platform
- Implementation of the different parts of ALU
- Tests and validation
- Production of the project report
- **State Machine Diagrams**
- **Parts List**
- **Input Lists**
- **Output Lists**
- **Interface List**

- **Module List**
- **Mode List**
- **Output**

**Updated Member Tasks:**

Paul Jacobo:	Alex Palm:	Patrick Meyomesse:
Choice of implementation (Software Discovery)	Coordination and Project Management	Simplification of Equations
Translation of the code to the chosen platform	<del>Development of Truth Tables</del>	<del>Circuit Design</del>
<b>Implementation of ALU Parts: Binary Comparison</b>	Production of the Project Report	Tests and Validation
Implementation of ALU Parts: Basic Logical Operations	<del>Implementation of ALU Parts: Arithmetic Operations</del>	<del>Implementation of ALU Parts: Binary Comparison</del>
Homework: Verilog Portion Primarily	Homework: Split the Problems	Homework: Split the Problems
<b>Circuit Design</b>	<b>State Machine Diagrams</b>	<b>State Machine Diagrams</b>
<b>Implementation of ALU Parts: Arithmetic Operations</b>	<b>Input List</b>	<b>Mode List</b>
<b>Output</b>	<b>Output List</b>	<b>Interface List</b>
	<b>Parts List</b>	<b>Module List</b>

### **Updated Software Discovery:**

For the Term Project Pretty Gate Machine is building an ALU with Verilog. The platform the Verilog code is compiling and simulating on is macOS Sierra version 10.12.6 running on a 2015 MacBook Pro. The Terminal accesses a specific file location to compile and run Verilog code. For ease, a split screen set up of The Terminal and text editor, Atom, is used to quickly code and run again. A unique extension is used while writing and editing code on Atom that allows the text editor to notify of any Verilog syntax errors.

Logisim will be used to create ideas and get a better sense of how modules are being built. Currently, Logisim is only being used for CS 4141 lab reports but is planned to be used for to plan the ALU.

Verilog was installed onto the platform via a GitHub Repository. In order to successfully complete the installation, the GitHub Desktop client was used to clone onto the platform.

Learning Verilog has been through trial and error. Resources such as <http://iverilog.icarus.com/>, stackoverflow.com, <http://iverilog.wikia.com>, are incredibly useful with plentiful information that leads the programmer in the way of solving the problem.

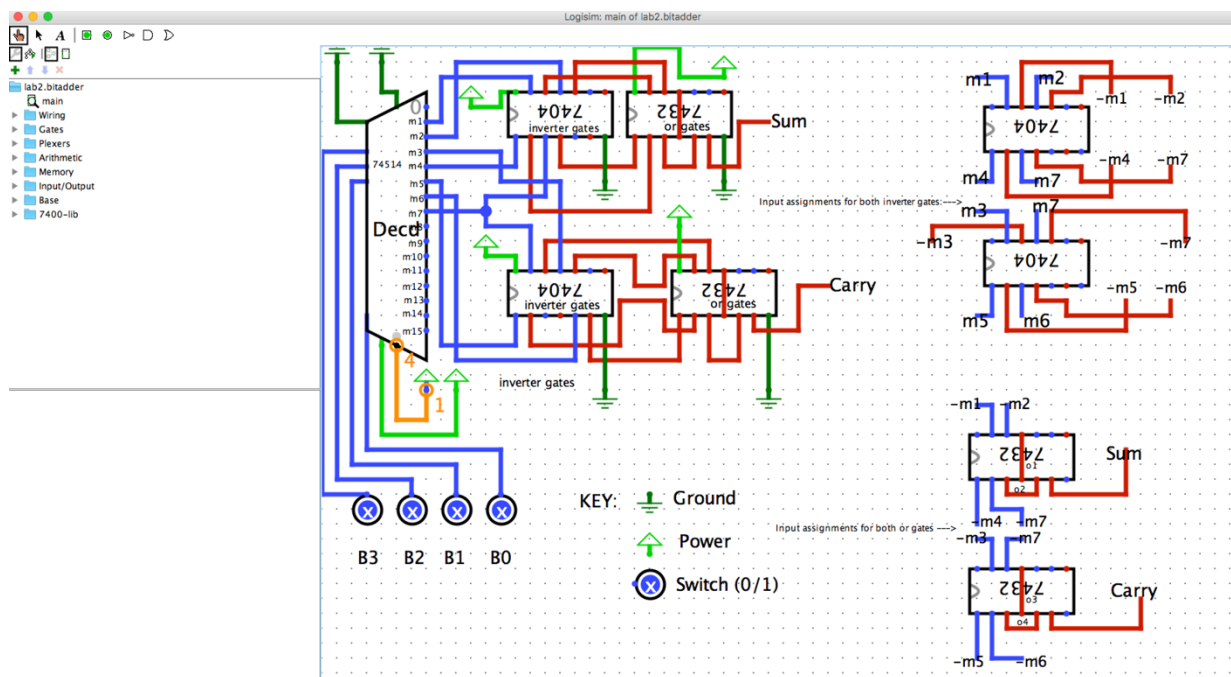
~~Implementing the Term Project's ALU is currently planned to contain multiple Verilog files with each being a complex module. The main source code will run include a module of all Verilog files' individual modules, creating the ALU as well as a test bench to run multiple delays and input.~~

**All modules are included in one single file. No additional software was used however, Verilog issues were primarily solved on websites such as Stackoverflow.com and iverilog.wikia.com. Additionally, before**

implementing any systems into code, models were simulated on Logisim to accurately display results.

SCREEN CAPTURES OF THE PLATFORM SETUP AND LOGISIM DIAGRAM HAVE BEEN INCLUDED

\*Logisim Diagram for 2<sup>nd</sup> CS 4141 Lab Report\*



\*Platform setup\*

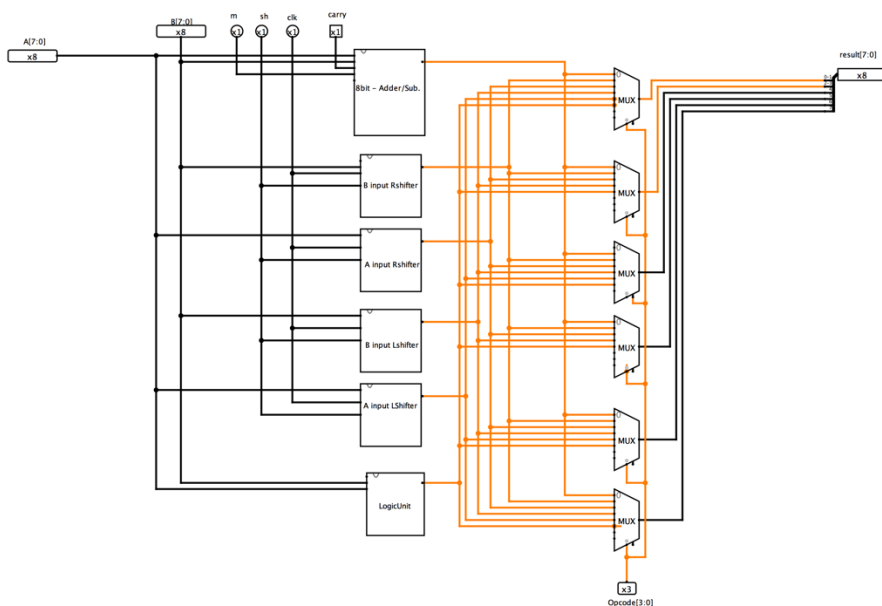
program2.txtversion

PrettyGateMachine.Program2.V

```
1 //
2 //Homework 2 - Hardware Programming Language Problem*
3 //DUE - 9/28/18
4 //Cohort name: Pretty Gate Machine
5 //Members/netID: Paul Jacobo pdj170036
6 //Patrick Meyomasse pgm170030
7 //Alex Palm awp140130
8 //Software: Icarus Verilog
9 //Source: Cloned from Icarus/Verilog repository.
10 //https://github.com/steveicarus/iverilog.git
11 //
12
13 //half adder - provided in the instructions
14 module Add_half (input a, b, output c_out, sum);
15     xor G1(sum, a, b);
16     and G2(c_out, a, b);
17 endmodule
18
19 //full adder - provided in the instructions
20 module Add_full (input a, b, c_in, output c_out, sum);
21     wire w1, w2, w3;
22     Add_half M1(a,b, w1, w2);
23     Add_half M0(w2, c_in, w3, sum);
24     or (c_out, w1, w3);
25 endmodule
26
27 //MODULE: 4-bit Unsigned Adder-Subtractor as diagramed in the instructions.
28 module four_b_asu(input [3:0] a, input [3:0] b, input M, output C, output [3:0] sum);
29
30     /*
31     Two sets of wires:
32     w# set - carries output from xor gates to input into full adder
33     c# set - carries output from full adders to input into the next full adder
34     */
35     wire w0,w1,w2,w3;
36     wire c1,c2,c3;
37
38     /*
39     Each 1/4 of the circuit is represented below by
40     having an XOR gate and FULL Adder.
41     Each XOR gate does not have input dependent of the output of the Full Adders.
42
43     xor GATES(eouput, input1, input2);
44     *output : - carried through wire (w#) to the input of a full Adder
45     *input1/2 : - carries back from add1/a parameters
46
47 PrettyGateMachine.Program2.V 5:30 LF UTF-8 Verilog 0 files
```

## UPDATED – Final Circuit Diagram

### ALU



Updated Participation Census:

