Team Project: ALU With Full Datapath Final Submission

Prepared for:

Dr. Eric Becker

Prepared by:

Gerardo Rodriguez

Lawrence Eric C. Evangelista

Mohamed Mohamed

CS 4341.001

20 Nov 2017

**Part 1: Updated Writeup and Member Tasks**

Description:

ALUs are present inside of processors and are used for resolving arithmetic operations, such as adding and subtracting or shifting bits left or right. Since processors are used in all computers and computers are used for almost everything nowadays, ALUs are useful for all sorts of tasks that require automatic computation. This includes all manners of counters, calculators, measurers, and regulators that may be encountered in any STEM-related field.

However, our ALU will be simple, only being able to perform a limited number of operations on 8-bit integers. For this reason, our ALU will not find much purpose in the real world besides a simple calculator. Provided we have the right circuitry and digital components, we could possibly create a fully working desk calculator. Although not much useful than a desk calculator from the store, our creation could be used to demonstrate (and perhaps show-off a little) our understanding of digital logic.

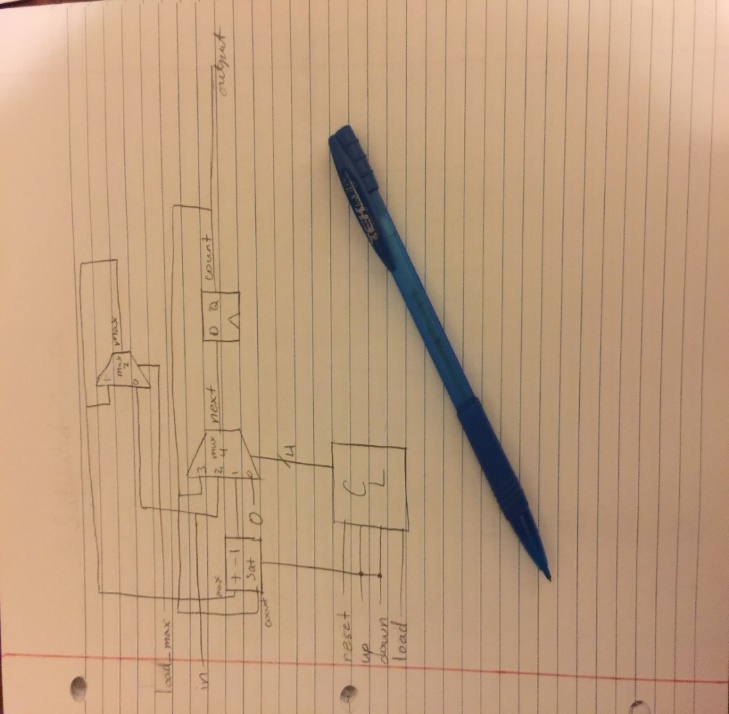
Updated Member Tasks

|  |  |  |  |
| --- | --- | --- | --- |
| Tasks | Gerardo Rodriguez | Lawrence Evangelista | Mohamed Mohamed |
| Add two numbers |  |  | YES |
| Subtract two numbers |  |  | YES |
| Multiply two numbers |  |  | YES |
| Bitwise AND two numbers |  | YES |  |
| Bitwise OR two numbers |  | YES |  |
| Bitwise NOT two numbers |  | YES |  |
| Bitwise XOR two numbers | YES |  |  |
| **Handle out of bounds errors** |  | **YES** | **YES** |
| **Create control unit** | **YES** | **YES** |  |
| Value persistency/reset command | YES |  |  |
| Updated Member Writeup and Tasks | YES | **YES** | **YES** |
| Updated Software Research |  | YES |  |
| Descriptions and Diagrams | **YES** | **YES** | YES |

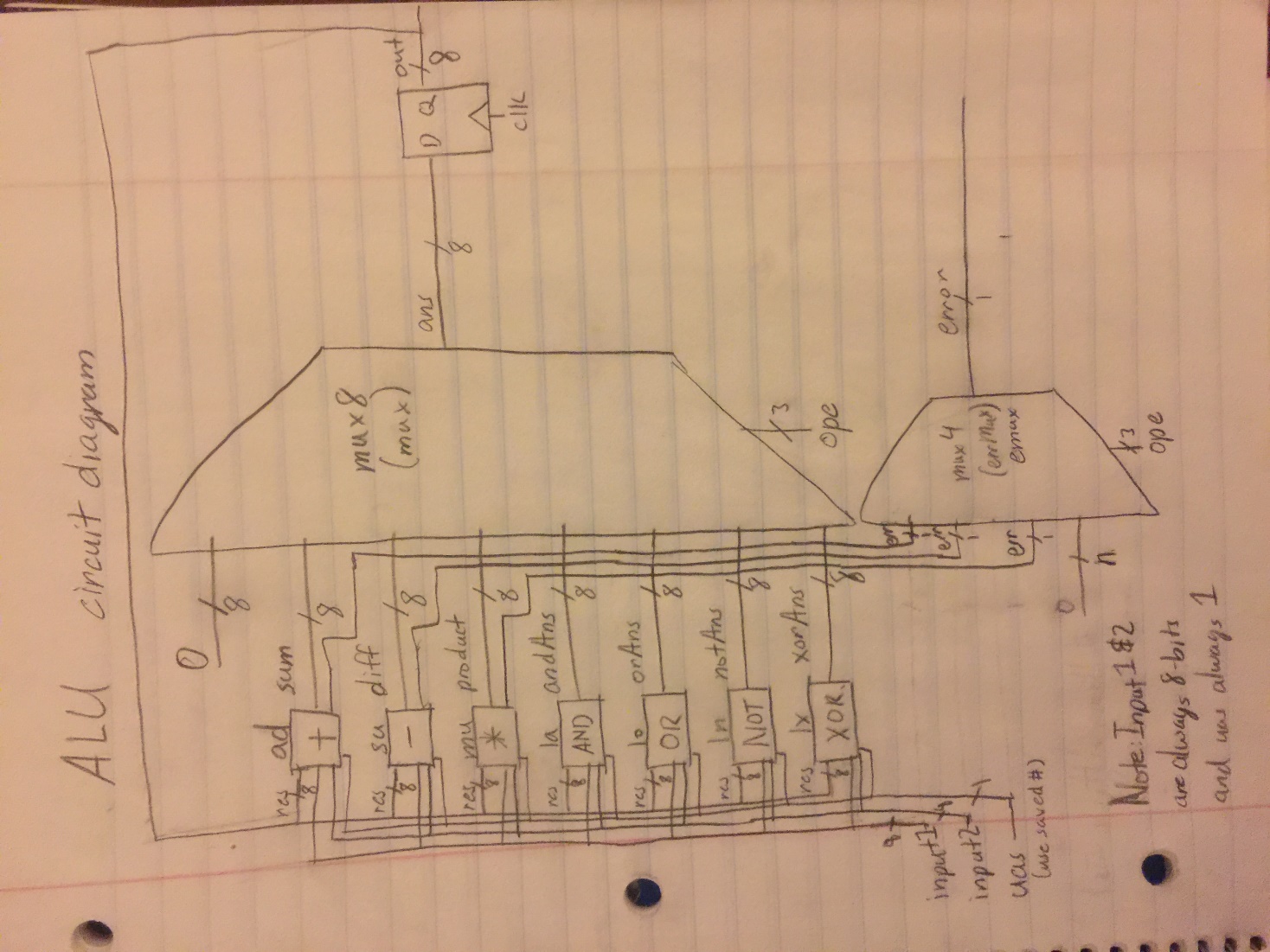
**Part 2: Updated Software Research**

Software Discovery:

~~At the moment our plans regarding circuit development software and drawing software are two-fold. Each of our circuits are going to be created twice.~~ The first time by hand using pencil and paper ~~and the second time on the computer using the free Logisim download available from Logisim’s SourceForge.net page that the cbchurch logisim website directs you to~~. We decided to have each of our circuits initially drawn on pencil and paper because of the ease of access, availability, and modifiability that come with it. It’s more likely that we’ll have a piece of paper to work on at any given moment in time rather than always being at a computer or having a laptop (which could potentially run out of charge) on us. It’s also less effort to just begin writing or drawing a diagram down on a piece of paper than learning how to use a piece of software first. ~~After being finalized on pen and paper we will transition our circuits to a digital diagram via the free Logisim software. We chose to use the Logisim software because it is free and from what we’ve done with it so far it doesn’t seem so difficult to use.~~ Below are screen captures of the pencil and paper ~~and Logisim~~ that we intend to use. **We decided to not use Logisim because we found it much easier to just draw the circuit on paper and Logisim clunky to operate.**

 **Part 3: Systems Design: Diagrams, tables, write-up**

Main Circuit Diagram: (Pencil and Paper)



Parts List:

Type = generic module name in Verilog code. Named = module’s instantiation name. The instantiation names and all inputs/outputs are labeled in the above diagram. All arithmetic and logic modules interact on two numbers (either input 1 and input 2 or input 1 and the previously saved result) except the NOT module (operates on either input 1 or the previously saved result).

* Mux8 (type Mux, named mux)
  + Features: Chooses which arithmetic/logic module output to save as the solution and output based on the ‘ope’ selector.
  + Inputs: sum, diff, product, andAns, orAns, notAns, xorAns, ope, Zero (0)
  + Outputs: ans
* Mux4 (type errMux, named emux)
  + Features: Chooses whether to output the presence of an ‘out of range’ error based on the error signals from the addition, subtraction, and multiplication modules and the ‘ope’ selector.
  + Inputs: aer, ser, mer, ope
  + Outputs: err
* Addition module (type Adder, named ad)
  + Features: Performs addition on two numbers and outputs the sum. Produces an error signal if the sum is greater than 8 bits.
  + Inputs: input 1, input 2, uas, res,
  + Outputs: sum, aer
* Subtraction module (type Subtractor, named su)
  + Features: Performs subtraction on two numbers and outputs the difference (input 1 minus either input 2 or saved result). Produces an error signal if the difference is negative.
  + Inputs: input 1, input 2, uas, res
  + Outputs: sum, ser
* Multiplication module (type Multiplier, named mu)
  + Features: Performs multiplication on two numbers and outputs the product. Produces an error signal if the product is greater than 8 bits.
  + Inputs: input 1, input 2, uas, res
  + Outputs: product, mer
* Bitwise AND module (type logicAnd, named la)
  + Features: Performs bitwise AND on two numbers and outputs the 8-bit result.
  + Inputs: input 1, input 2, uas, res
  + Outputs: andAns
* Bitwise OR module (type logicOr, named lo)
  + Features: Performs bitwise OR on two numbers and outputs the 8-bit result.
  + Inputs: input 1, input 2, uas, res
  + Outputs: orAns
* Bitwise NOT module (type logicNot, named ln)
  + Features: Performs bitwise NOT on one number and outputs the 8-bit result.
  + Inputs: input 1, input 2, uas, res
  + Outputs: notAns
* Bitwise XOR module (type logicXor, named lx)
  + Features: Performs bitwise XOR on two numbers and outputs the 8-bit result.
  + Inputs: input 1, input 2, uas, res
  + Outputs: xorAns
* DFF module (type DFF, named result)
  + Features: Saves the input and outputs it. (Input: ans, Output: out)

Input and Output Definitions:

The above parts list shows which elements are taken as an input or produced as an output by each of the modules in the circuit.

* input 1: A programmable 8-bit number used as an operand by the arithmetic/logic modules.
* input 2: A programmable 8-bit number used as an operand by the arithmetic/logic modules.
* ope: Programmable 3-bit number indicating which operation to perform (select).
* clk: Clock signal that determines when DFF saves and outputs.
* uas: A programmable 1-bit indicator of whether the arithmetic/logic module should use the previously saved result as one of its operands.
* out: The operation answer saved in the DFF.
* res: The previously saved output.
* ans: The arithmetic/logic result selected by the Mux8
* err: 1-bit indicator of whether an out of bounds error is present (produced by the Mux4).
* aer: 1-bit indicator of whether the addition module experienced an out of bounds error.
* ser: 1-bit indicator of whether the subtraction module experienced an out of bounds error.
* mer: 1-bit indicator of whether the multiplication module experienced an out of bounds error.
* sum: 8-bit sum of the two numbers passed to the addition module.
* diff: 8-bit difference of the two numbers passed to the subtraction module.
* product: 8-bit difference of the two numbers passed to the multiplication module.
* andAns: 8-bit bitwise AND result from the AND module.
* orAns: 8-bit bitwise OR result from the OR module.
* notAns: 8-bit bitwise NOT result from the NOT module.
* xorAns: 8-bit bitwise XOR result from the XOR module.

**Part 4: Verilog Simulator with Output**

Because the Verilog program is ~430 lines long we will not be including it within this document. Instead it shall be attached to the submission in a .v file named alue3.v. Below is the output of running the Verilog program. For improved readability the Verilog output will also be provided in an attached text file.

**NB: The outputs of the Result column are delayed by one clock cycle. To view the proper relationship between all columns, please note that you have to shift the Result column up by one row. All other columns are accurate.**

C:\iverilog>iverilog alu2.v

C:\iverilog>vvp a.out

Clock| Number1| Number2|Operation|UseSaved| Result| Error

-----+--------+--------+---------+--------+--------+------

0|00000000|00000000|000 (Rst)| 0|xxxxxxxx| None

0|00000000|00000000|000 (Rst)| 0|00000000| None

0|00000001|00000001|001 (Add)| 0|00000000| None

0|00000001|00000010|001 (Add)| 0|00000010| None

0|00000001|00000011|001 (Add)| 0|00000011| None

0|00000100|00000010|001 (Add)| 0|00000100| None

0|00000100|00000011|001 (Add)| 0|00000110| None

0|00000010|00000010|001 (Add)| 1|00000111| None

0|00000111|00000010|001 (Add)| 1|00001001| None

0|00000001|00000010|001 (Add)| 1|00010000| None

0|00000010|00000010|001 (Add)| 0|00010001| None

0|00000011|00000011|001 (Add)| 0|00000100| None

0|11111111|00000001|001 (Add)| 0|00000110| Out of Range Error

0|11111111|00000011|001 (Add)| 0|00000000| Out of Range Error

0|00000001|11111111|001 (Add)| 0|00000010| Out of Range Error

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000000|00000000|010 (Sub)| 0|00000000| None

0|00000001|00000001|010 (Sub)| 0|00000000| None

0|00000001|00000010|010 (Sub)| 0|00000000| Out of Range Error

0|00000001|00000011|010 (Sub)| 0|11111111| Out of Range Error

0|00000100|00000010|010 (Sub)| 0|11111110| None

0|00000100|00000011|010 (Sub)| 0|00000010| None

0|00000010|00000010|010 (Sub)| 1|00000001| None

0|00000111|00000010|010 (Sub)| 1|00000001| None

0|00000001|00000010|010 (Sub)| 1|00000110| Out of Range Error

0|00000010|00000010|010 (Sub)| 0|11111011| None

0|00000011|00000011|010 (Sub)| 0|00000000| None

0|11111111|00000001|010 (Sub)| 0|00000000| None

0|11111111|00000011|010 (Sub)| 0|11111110| None

0|00000001|11111111|010 (Sub)| 0|11111100| Out of Range Error

0|00000001|00000010|000 (Rst)| 0|00000010| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000000|00000000|011 (Mul)| 0|00000000| None

0|00000001|00000001|011 (Mul)| 0|00000000| None

0|00000001|00000010|011 (Mul)| 0|00000001| None

0|00000001|00000011|011 (Mul)| 0|00000010| None

0|00000100|00000010|011 (Mul)| 0|00000011| None

0|00000100|00000011|011 (Mul)| 0|00001000| None

0|00000010|00000010|011 (Mul)| 1|00001100| None

0|00000111|00000010|011 (Mul)| 1|00011000| None

0|00000001|00000010|011 (Mul)| 1|10101000| None

0|00000010|00000010|011 (Mul)| 0|10101000| None

0|00000011|00000011|011 (Mul)| 0|00000100| None

0|11111111|00000001|011 (Mul)| 0|00001001| Out of Range Error

0|11111111|00001111|011 (Mul)| 0|11111111| None

0|00000001|11111111|011 (Mul)| 0|11110001| Out of Range Error

0|00000010|11111111|011 (Mul)| 0|11111111| Out of Range Error

0|11111111|11111111|011 (Mul)| 0|11111110| None

0|00000001|00000010|000 (Rst)| 0|00000001| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|11111111|00000001|100 (AND)| 0|00000000| None

0|11111111|00000011|100 (AND)| 0|00000001| None

0|11111111|00000111|100 (AND)| 0|00000011| None

0|11111111|00001111|100 (AND)| 0|00000111| None

0|11111111|00011111|100 (AND)| 0|00001111| None

0|11111111|00011111|100 (AND)| 1|00011111| None

0|00011111|00000001|100 (AND)| 1|00011111| None

0|00001111|00000001|100 (AND)| 1|00011111| None

0|00000111|00000001|100 (AND)| 1|00001111| None

0|00000011|00000001|100 (AND)| 1|00000111| None

0|00000001|00000001|100 (AND)| 1|00000011| None

0|11111111|00000001|100 (AND)| 0|00000001| None

0|11111111|00000011|100 (AND)| 0|00000001| None

0|00000001|11111111|100 (AND)| 0|00000011| None

0|00000001|00000010|000 (Rst)| 0|00000001| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|11111111|00000000|101 ( OR)| 0|00000000| None

0|01111111|00000000|101 ( OR)| 0|11111111| None

0|00111111|00000000|101 ( OR)| 0|01111111| None

0|00011111|00000000|101 ( OR)| 0|00111111| None

0|00001111|00000000|101 ( OR)| 0|00011111| None

0|00000111|00000000|101 ( OR)| 0|00001111| None

0|00000011|00000000|101 ( OR)| 0|00000111| None

0|00000001|00000000|101 ( OR)| 0|00000011| None

0|00000110|00000001|101 ( OR)| 1|00000001| None

0|00011000|00000001|101 ( OR)| 1|00000111| None

0|01100000|00000000|101 ( OR)| 1|00011111| None

0|10000000|00000001|101 ( OR)| 1|01111111| None

0|11111111|00000011|101 ( OR)| 0|11111111| None

0|00000001|11111111|101 ( OR)| 0|11111111| None

0|00000001|00000010|000 (Rst)| 0|11111111| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|11111111|00000001|110 (NOT)| 0|00000000| None

0|01111111|00000011|110 (NOT)| 0|00000000| None

0|00111111|00000111|110 (NOT)| 0|10000000| None

0|00011111|00001111|110 (NOT)| 0|11000000| None

0|00001111|00011111|110 (NOT)| 0|11100000| None

0|00000111|00011111|110 (NOT)| 0|11110000| None

0|00000011|00000001|110 (NOT)| 0|11111000| None

0|00000001|00000001|110 (NOT)| 0|11111100| None

0|00000000|00000001|110 (NOT)| 0|11111110| None

0|10101010|00000001|110 (NOT)| 0|11111111| None

0|01010101|00000001|110 (NOT)| 0|01010101| None

0|00110011|00000001|110 (NOT)| 0|10101010| None

0|11001100|00000011|110 (NOT)| 0|11001100| None

0|00000000|00000001|110 (NOT)| 1|00110011| None

0|00000000|00000001|110 (NOT)| 1|11001100| None

0|00000001|00000010|000 (Rst)| 0|00110011| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|11111111|00000000|111 (XOR)| 0|00000000| None

0|11111111|11111111|111 (XOR)| 0|11111111| None

0|01010101|10101010|111 (XOR)| 0|00000000| None

0|00110011|11001100|111 (XOR)| 0|11111111| None

0|00001111|00001111|111 (XOR)| 0|11111111| None

0|00001111|11111111|111 (XOR)| 0|00000000| None

0|00000001|00000001|111 (XOR)| 0|11110000| None

0|00000011|00000001|111 (XOR)| 0|00000000| None

0|00000010|00000001|111 (XOR)| 0|00000010| None

0|00000100|00000001|111 (XOR)| 1|00000011| None

0|00001001|00000001|111 (XOR)| 1|00000111| None

0|00110000|00000001|111 (XOR)| 1|00001110| None

0|00000001|00000010|000 (Rst)| 0|00111110| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

0|00000001|00000010|000 (Rst)| 0|00000000| None

\*\* VVP Stop(0) \*\*

\*\* Flushing output streams.

\*\* Current simulation time is 1280 ticks.

> finish

\*\* Continue \*\*

C:\iverilog>