2020

Design of Ternary Operations Utilizing Flow-Based Computing

James Pyrich
University of Central Florida

DESIGN OF TERNARY OPERATIONS UTILIZING FLOW-BASED COMPUTING

by

JAMES ROBERT PYRICH
M.S. Computer Science, University of Central Florida, 2018

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2020

Major Professor: Sumit Kumar Jha

ABSTRACT

The development of algorithms and circuit designs that exploit devices that have the ability to persist multiple values will lead to alternative technologies to overcome the issues caused by the end of Dennard scaling [1] and slowing of Mooreś Law [2] [3]. Flow-based designs have been used to develop binary adders and multipliers [4] [5] [6]. Data stored on non-volatile memristors are used to direct the flow of current through nanowires arranged in a crossbar. The algorithmic design of the flow-based crossbar is fast, compact, and efficient [5]. In the following paper, we seek to automate the discovery of flow-based designs of ternary circuits utilizing memristive crossbars.

ACKNOWLEDGMENTS

I recognize Sumit Kumar Jha for his work with crossbar memristors leading to the inspiration of applying multi-valued logic to new technology. The C++ test environment was developed by Jim Pyrich derived from Python code written by Dwaipayan Chakraborty. The NuSMV simulations were performed by Sunny Raj.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Ternary Based Computing

A digital system using a radix of base 3 may be a more efficient implementation of switching circuits than a radix of 2 [Alexander, 1964] [7]. The most economical radix is calculated by finding the relationship of the amount of the equipment required for a certain radix and the digit capacity of the machine. The resulting minimum radix is the natural base (e = 2.718). Since the radix needs to be an integer, this value is rounded up to 3 which implies that a ternary computer might have an advantage over existing binary based systems. When ternary computers were first proposed, there were no devices in common use that would maintain 3 stable states. The discovery of the missing circuit element [8] [9] and the subsequent fabrication of the memristor [10] provide a device to maintain multiple states required for multi-value operations.

In [11], it is stated that ternary computers (base 3) can compete with binary computers (base 2) in terms of performance, but cost may a more limiting factor. In Table 4.2, there are 18 minterms required to compute a 1-bit ternary adder. This would require 1.62 times more logic to produce a ternary computer when compared to a conventional binary computer. Ternary digits (or trits) carry $log_2 3 = 1.58$ bits of information.

A design and implementation of 2 bit ternary ALU device in [12] proposed that a ternary system might have an advantage over existing binary based systems since interconnections and chip area would be reduced. Although the ternary ALU (T-ALU) described was designed for 2 bit operations, ALUs could be cascaded resulting in n-bit operations. The functions of the T-ALU include addition, subtraction, multiplication, comparison, OR, NOR, AND, NAND, and Ex-OR.

The integration of ternary logic gates into conventional binary structures has been researched [13]

using a carbon nanotube field-effect (CNTFET) circuit. Carbon nanotubes (CNTs) can be used to implement efficient multiple valued logic (MVL) circuits by varying with multiple threshold voltages [14]. The threshold voltages can be adjusted by adjusting the diameter of the CNT [15] with a reference point of [16] as $V_\pi(\ 3.033eV)$ in a SPICE simulation. Based on certain characteristics, this design methodology was shown to be an alternative to metal oxide semiconductor field effect transistor (MOSFET) circuits to efficiently produce MLV circuits.

Memristive Systems

In 1971, Leon Chua identified a fourth basic circuit element called the memristor [8]. The fundamental circuit elements are current i, voltage v, charge i, and flux φ. The memristor is defined as the relationship between charge and flux (Memristance = Flux/Charge) [17] [18]. When controlled by charge, the measure of memristance is:

$$v(t) = M(q(t))i(t) \tag{1.1}$$

$$M(q) \equiv \frac{d\varphi(q)}{dq} \tag{1.2}$$

When controlled by flux, the measure of memductance is:

$$i(t) = W(\varphi(t))v(t) \tag{1.3}$$

$$W(\varphi) \equiv \frac{dq(\varphi)}{d\varphi} \tag{1.4}$$

In 1976, Chua and Kang broadened the definition of memristors as a class of nonlinear dynamical systems called memristive systems [19]. One of the defining properties was a zero-crossing property where the output was zero when then the input was zero. The output takes the form of a Lissajous figure which can be correlated to a pinched hysteresis loop [20] or bowtie[10].

In 2015, Chua stated that any two terminal device that produces a hysteresis loop and passes through the origin is to be considered a memristor [21]. Classes of memristors are defined as ideal, ideal generic, generic, and extended. In 2018, Biolek *et. al.* [22] documents other non-memristive systems that produce pinched hysteresis loops. The question is raised whether the newer classes of memristors are actually new circuit elements. Chua states that the definition of a memristive system should be considered a black box since he believes that the internal composition is irrelevant. In [23], Chua states "If it's pinched, it's a memristor." which Biolek considers hyperbole. This paper will focus on ideal memristors and models for general memristive devices [24].

In a presentation titled Fundamentals of Memristors [25], the following equations are presented for ideal, generalized memristors, and memristive systems [8] [19] [23]. Ideal current controlled memristors are defined in equation 1.5. The equation for a current controlled generalized memristive system is 1.6. For time invariant memristive systems, the properties in 1.7 are needed to maintain a pinched hysteresis loop.

$$V = R(q) * I \tag{1.5}$$

$$\begin{aligned} V &= R(x, I) * I \\ \hat{x} &= f(x, I) \end{aligned} \tag{1.6}$$

$$\begin{aligned} R(x, I) &\neq \infty \\ f(x, 0) &= 0 \end{aligned} \tag{1.7}$$

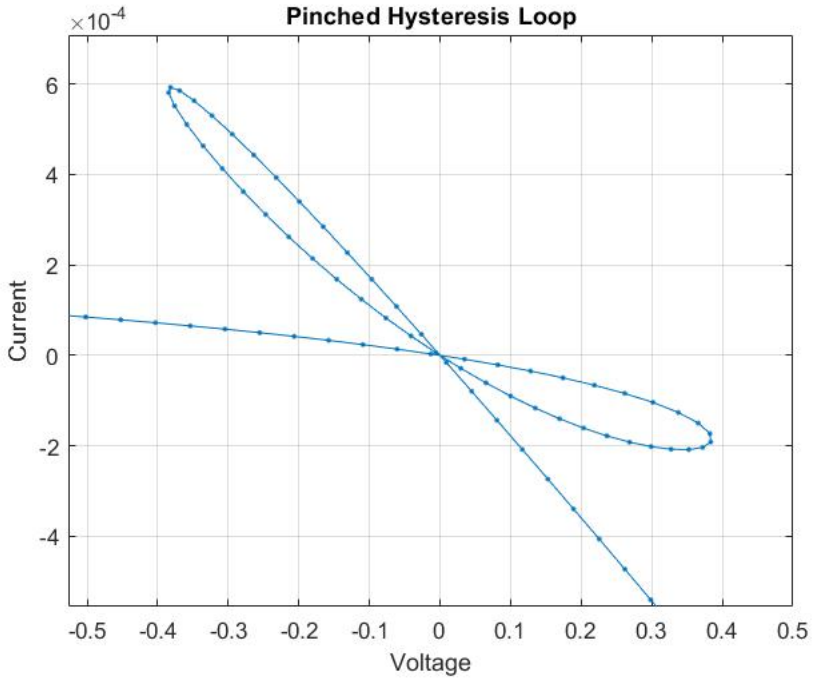

Figure 1.1: The Berkeley Model and Algorithm Prototyping Platform

In the above equations, the value x represents the inner state variable.

Crossbars and Sneak Paths

Memristive crossbars may be effective to overcome limitation of existing architectures [17]. Crossbar networks are organized with a top set of parallel nanowire electrodes perpendicular to a bottom set of the same. At each point where the wires cross is a memristor that can be turned off and on directing current flow to connected wires. A crossbar with n rows and m columns can contain up to nm memristors [4] which results in $2nm$ connections between each layer of nanowires. Sneak paths [26] are a problem when undesired paths created by ON memristors allow some current to flow to the attached set of nanowires.

Sneak paths can also be exploited as in [4] [5] [6] [27] [28] [29]. As an example, nanoscale

crossbars have been designed for a one-bit Boolean full adder with three input bits (A, B, C_{in}). The input bits determine the state of each memristor located where the crossbar wires intersect. Voltage is applied at the bottom row. The sum and carry-out bits are determined by the sneak current measured on the top two rows. A Boolean value of zero or False relates a turned-off memristor. Conversely, a Boolean value of one or True relates to a turned-on memristor. A crossbar array was fabricated at SUNY Polytechnic Institute using $1k\Omega$ resistors for the ON state and $1M\Omega$ for the OFF state. A pulse voltage of $100mV$ and $200mV$ were applied to the bottom row. The resulting voltages had a noise margin of $55mV$ which is a sufficient separation to determine the Sum and C_{out} Boolean values.

Multilevel Digital Systems

In [30], mathematical models of memristors are analyzed. Digital circuits based on number systems other than binary are discussed. One method discussed is Memristor Quantization where ranges of resistance are divided evenly by N. For ternary options, where $N = 3$, each memristor is divided evenly between R_{off} and R_{on} with $\Delta R = (R_{off} - R_{on})/3$. Considering HP's values [9] of $R_{on} = 100\Omega$ and $R_{off} = 16k\Omega$, the center values would be approximately $\{2.5k\Omega, 7.5k\Omega, 12.5k\Omega\}$. False detection becomes more of an issue with a larger N. A gap between regions is a necessity [31] for good accuracy. It may also be beneficial to divide regions unequally based on the probability of results.

Flow-based Computing

Flow-based designs have been used to develop binary adders and multipliers [4] [5] [6]. Data stored on non-volatile memristors are used to direct the flow of current through nanowires arranged in a

crossbar. The algorithmic design of the flow-based crossbar is fast, compact, and efficient [5]. The data is encoded based on the value of each input, then it is stored in the memristors affecting the flow of current across each nanowire [27] [28]. Some memristors can by permanently turned ON or OFF which can relate to literal values in the particular formula [29].

CHAPTER 2: LITERATURE REVIEW

SPICE Circuit Simulator

In [32] [33], a memristor device model is proposed to simulate the $I-V$ relationship of a memristor based on changes to a state variable with respect to time. The state variable is a value between 0 and 1 that affects current flow and conductivity of the device. The SPICE sub-circuit in [34] utilizes the hyperbolic sine function to estimate conductivity beyond a voltage threshold. The parameters a_1 and a_2 represent amplitudes based on the direction of the current. The intensity of the threshold is determined by the parameter b.

$$I(t) = \begin{cases} a_1 x(t) sinh(bV(t)) & V(t) \geq 0 \\ a_2 x(t) sinh(bV(t)) & V(t) < 0 \end{cases}$$

The conductance is determined using negative and positive thresholds defined as A_p and A_p.

$$g(V(t)) = \begin{cases} A_p(e^{V(t)} - e^{V_p}) & V(t) > V_p \\ -A_n(e^{-V(t)} - e^{V_n}) & V(t) < -V_n \\ 0 & -V_n \leq V(t) \leq V_p \end{cases}$$

The version of SPICE utilized in this study is the open source version Ngspice, a software application originally written by Berkeley University, currently maintained by the Ngspice project [35]. Ngspice is simulation software used to test circuit designs prior to the fabrication of a physical circuit. Ngspice is used with a model circuit designed specifically for testing memristive devices [36].

Simulated Annealing

Simulated Annealing (SA) can be used to solve optimization problems [37]. The SA algorithm performs a random walk through the search space looking for the lowest energy. The benefit of a random walk is the search will not get stuck in local minima while attempting to find the best solution (i.e. lowest temperature).

SA can be used to find solutions to problems that are considered NP-Hard. One heuristic that can be used to find an optimal solution is to start with a known path followed by iterative improvement based on a cost factor. Another method is start with a fully random solution [38]. Minor changes are made to the path at each step and the cost factor is recomputed.

When annealing occurs in a metal, the rate of the cooling process determines the state of the solid that is formed. In SA, starting/ending temperatures and a rate of cooling are set. To find an optimal solution, the temperature should be reduced slowly. For each iteration (or step), the cost factor (or energy) is computed and returned to the calling algorithm. A set number iterations are performed for each step so the algorithm can explore an area the for lowest energy.

The following constants were used for Simulated Annealing (SA) in the GNU Scientific Library (GSL) [39]:

Step	1000000
Iterations	1000
Step Size	1.0
Boltzmann Constant	1.0
Initial Temp	2500.0
Damping Factor	1.002
Minimum Temp	1e-25

The simulated annealing algorithm calculates the probability of taking a step using the Boltzmann

distribution:

$$p = e^{(E_{i+1}-E_i)/kT}$$

if $E_{i+1} > E_i$ and $p = 1$ when $E_{i+1} \leq E_i$.

The initial temperature T is set to a high value and is lowered by the damping factor until the minimum temperature is reached. The GSL implementation of simulated annealed uses callback functions for energy and step decisions. The energy callback function calculates the cost or best energy of the current step. If the cost is lower, a step will occur. The GSL function provides a callback function to output its progress. Upon completion, the best result is returned.

NuSMV

NuSMV is a symbolic model checker based on CMU SMV [40]. NuSMV implements a BDD-based (Binary Decision Diagram) and SAT-based (Boolean Satisfiability Problem) model checking capability. NuSMV was used to prove that a solution existed for implementation of the full-adder using memristive crossbars.

Modified Nodal Analysis

It is well known that Ohm's Law ($V = IR$) describes the linear relationship between voltage, current, and resistance. An ideal memristor is a resistor that changes based on the relationship between charge and flux over time [41]. At a snapshot in time, a resistive crossbar network can be analyzed using a modified nodal approach [42].

The following node voltage method can be used to calculate current through a resistive crossbar network [43]. To apply the node voltage method to a circuit with n nodes (with m voltage sources),

perform the following steps [44].

1. Select a reference node (usually ground).
2. Name the remaining $n-1$ nodes and label a current through each passive element and each current source.
3. Apply Kirchhoff's current law to each node not connected to a voltage source.
4. Solve the system of $n-1-m$ unknown voltages.

To solve the system of equations, the matrix formula $Ax = z$ is used where x is a $(n+m)x1$ matrix that contains the unknown n node voltages m represents independent current sources. The solution is found by using an algorithmic MNA formula that takes the inverse of the A matrix multiplied by the z matrix, $x = A^{-1}z$ [43].

Memristive Device Models

DC analysis of memristive device models can be improved by formatting the simulation algorithms in a differential equation form [24]. Memristors and other devices that exhibit an $i-v$ hyteresis require equations that result in a smooth curve on a graph. Memristive algorithms with piece-wise window functions tend not to be continuous. Popular models were modified to address the dynamics of the internal state variable and implement functions that are smooth and safe.

The Berkeley Model and Algorithm Prototyping Platform (MAPP) [45] was used to produce MATLAB graphs and to check results. MAPP utilizes ModSpec [46] to describe and prototype new devices in MATLAB. The MAPP environment is initiated in MATLAB by running startMAPP. The test function $memristorModSpec$ utilizes two switches, $f1 = 1$ (shifting between resistor states) and $f2 = 5$ (modified Yakopcic model).

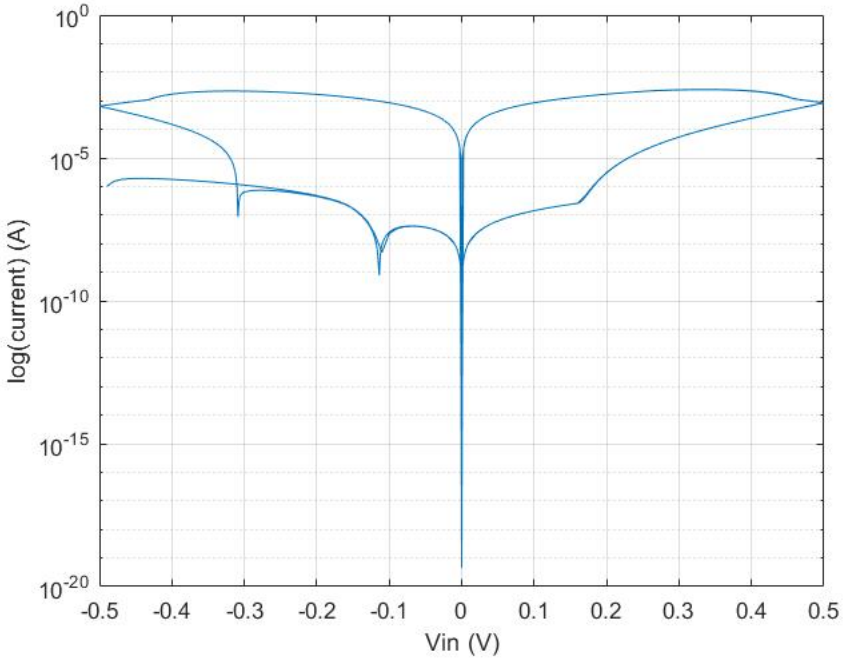

Figure 2.1: University of Berkeley: Yakopcic Model

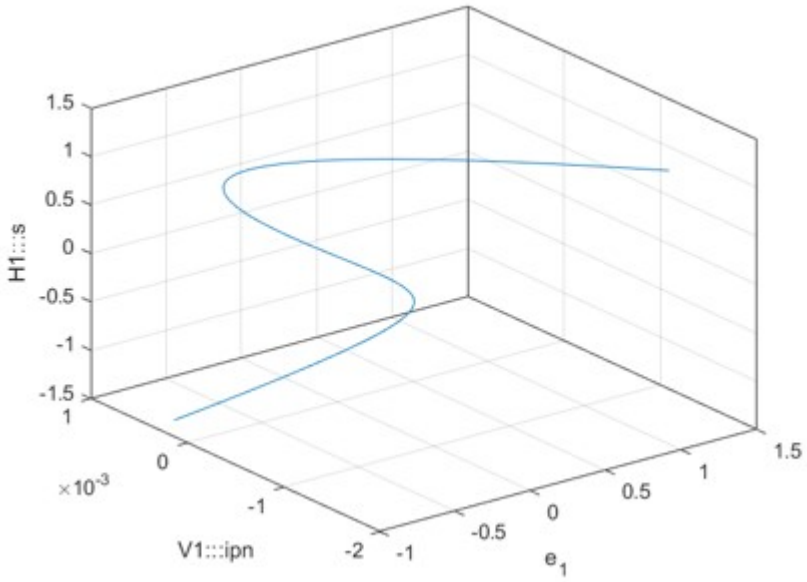

Figure 2.2: University of Berkeley: 3D Homotopy Analysis

CHAPTER 3: METHODOLOGY

The research goal is to find an efficient method of simulating ternary operations utilizing memristive crossbars. Initial tests were performed using basinhopping (SciPy) referencing a dictionary of potential memristor state values. A SPICE call was made for each truth table entry. Based on the combined results of each SPICE call, the memristor list was randomly perturbed and the cycle continued until the process timed out or was canceled. The initial program utilized 18 SPICE call since $C_{in} = 2$ was excluded.

The Python program was converted to C++ with the integration of Simulated Annealing (SA) in the GNU Scientific Library (GSL) [39]. The next step was to develop a flow-based method of estimating valid circuits to eliminate the need for external systems calls to SPICE or any other external library. This helped performance greatly although a solution was still elusive. To confirm we were on the right track, NuSMV was used to prove that a potential solution existed. It was then that we realized that we weren't testing the entire truth table for ternary addition. We included all 27 entries and removed the dictionary in lieu of a full memristor array that also included static connections. A half-ternary adder was developed that limited the number of truth table entries to nine which was very fast. A full-ternary adder followed with numerous optimizations for speed and accuracy.

A significant effort was made toward narrowing the gap of resistances for each state (i.e. on, off, mid). It was decided to test resistive networks using SPICE and modified nodal analysis in an attempt to find linear solutions. This research was mildly successful, but the gaps were too tight given the new resistances and the resulting arrays did not port when tested with memristors. There were relatively few outliers that, if resolved, will allow for tighter ranges and larger gaps needed for fabrication.

CHAPTER 4: SYNTHESIS OF A TERNARY ADDER

Approximate Synthesis of a One-Bit Adder Using Ternary Logic

In 2014, Velasquez and Jha [17] demonstrated 1-bit addition and other Boolean formulas measuring sneak path voltages on memristive crossbars. The state of each memristor was either R_{off} or R_{on} mapping to 0 or 1 on the truth table shown in Table 4.1. The Boolean formula is for computing the Sum bit is:

$$(A \wedge \neg B \wedge \neg C_{in}) \vee (\neg A \wedge B \wedge \neg C_{in}) \vee (\neg A \wedge \neg B \wedge C_{in}) \vee (A \wedge B \wedge C_{in})$$

This work shows that an n-ary Boolean function where $n = 3$ results in $2^3 = 8$ rows in Table 4.1. A set of memristors are mapped to the values of A, B, and C_{in} producing the values in the Sum column. The simulations relate voltage drops to logical values. A drop of 1 volt indicates $true$ and drops below 0.5 volts indicates the logical value of $false$. A simulation was run with the current flowing through the voltage source was 10^{-4} amperes when the formula was $true$ and less than 10^{-5} amperes when the formula was $false$.

Table 4.1: Truth Table: Binary Addition

A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 4.2: Truth Table: Ternary Addition (partial)

A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	2
0	2	0	0	2
0	2	1	1	0
1	0	0	0	1
1	0	1	0	2
1	1	0	0	2
1	1	1	1	0
1	2	0	1	0
1	2	1	1	1
2	0	0	0	2
2	0	1	1	0
2	1	0	1	0
2	1	1	1	1
2	2	0	1	1
2	2	1	1	2

Simulations

A test application was designed to use Ngspice (v.28) to test each possible circuit in the search for a solution. The program was written in Python (Chakraborty, 2018) simulating solutions of the ternary adder in Table 4.2 by using Simulated Annealing (SA) [39]. A circuit file was generated as input to Ngspice, followed by an external program call with subsequent analysis of the output. This process was repeated for each row of the truth table. The Python program didn't result in a solution initially. To speed up the process, a testing program was written in C++ (Pyrich and Jha, 2018). Some of the processes were faster, but system calls to SPICE were very inefficient. A different approach was needed to simulate the crossbar and memristor states to search for a possible solution or approximation.

It was determined that numerous system calls to SPICE was causing a bottleneck, so the energy function was refactored to a flow-based design with SPICE calls occurring only when the free energy equaled zero, indicating a possible design of a crossbar adder. During each SA callback to the energy function, the memristors on the crossbar were randomly perturbed. A SPICE circuit was then generated based on the dictionary in Table 4.3.

Table 4.3: Simulated Annealing Dictionary

Memristor	Variable	Value
0	Constant	0
1	Constant	1
2	Constant	2
3	A	truth table
4	A'	$(A{+}1) \mod 3$
5	$\neg A$	$(A{+}2) \mod 3$
6	B	truth table
7	B'	$(B{+}1) \mod 3$
8	$\neg B$	$(B{+}2) \mod 3$
9	C_{in}	truth table
10	C'_{in}	$(C_{in}{+}1) \mod 3$
11	$\neg C_{in}$	$(C_{in}{+}2) \mod 3$

Before proceeding with a redesign of the ternary adder application, NuSMV was used to prove that a memristive crossbar existed (Raj, 2018). The ternary full-adder was used ($3^3 = 27$ entries) to produce the solution in Table 4.4.

Table 4.4: NuSMV 5x5 design of the sum bit

m	0	1	2
1	0	0	1
2	0	1	0
3	0	0	2
4	0	2	0
5	0	1	2
8	0	2	2
9	1	0	0
12	1	0	2
13	1	2	0
18	2	0	0
19	2	0	1
21	2	0	2

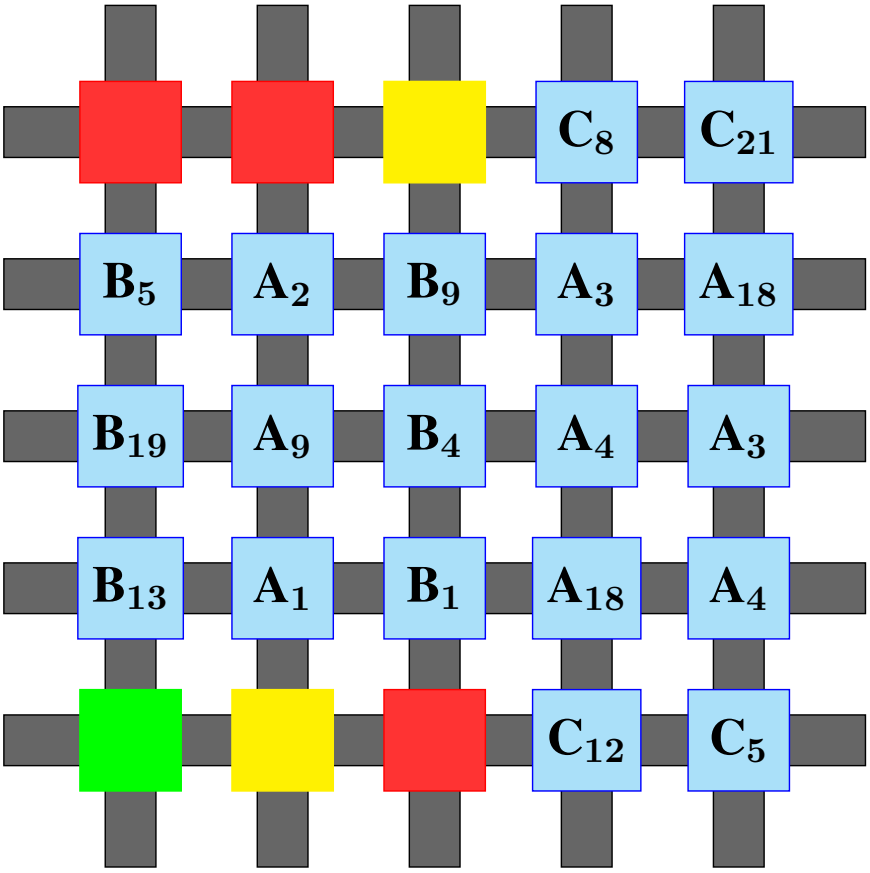

Figure 4.1: Crossbar design for computing the sum bit of ternary addition obtained from NuSMV

CHAPTER 5: DESIGN OF A TERNARY ADDER

Flow-Based Computing

A flow-based methodology was used to compute memristive crossbars based on the state of the memristor and resulting sneak paths shown in Table 5.1 [47].

Table 5.1: Crossbar Simulation

Memristor	Row	Column	Result
on	on	off	col = on
on	on	mid	col = on
on	mid	off	col = mid
on	mid	mid	col = mid+
on	off	on	row = on
on	mid	on	row = on
on	off	mid	row = mid
on	mid	mid	row = mid
mid	on	off	col = mid
mid	on	mid	col = mid+
mid	mid	off	col = mid+
mid	mid	mid	col = mid+
mid	off	on	row = mid
mid	mid	on	row = mid+
mid	off	mid	row = mid+
mid	mid	mid	row = mid+

In the previous chapter, it was shown that the SA energy function required a faster method to find potential circuits. The revised SA energy function estimates current flow based on the state of each row and column based on the state of the memristor. The state of the memristor is determined by each row in Table 5.2. For example, if the memristor value is B13, the table entry is $1, 2, 0$ which maps to $B = [on, mid, off)$ where $off = 0, mid = 2, on = 1$. Note that in the experiments listed in the chapter on Findings, values are sometimes mapped as $off = 0, mid = 1, on = 2$ for

reasons of better performance or separation of output voltage ranges.

Table 5.2: Design of a 1-bit ternary adder

m	0	1	2
1	0	0	1
2	0	1	0
3	0	0	2
4	0	2	0
7	0	1	1
8	0	2	2
13	1	2	0
16	1	2	1
18	2	0	0
21	2	0	2
22	2	2	0

Ternary Adder Algorithm

The SA energy callback function is described in the Ternary Adder search Algorithm 1. The mem value passed into the ENERGY function from SA is an array of possible memristor values. Flow-based analysis of each row in the ternary truth table is performed based on each value in the mem array. The resulting value is read from the last row for Sum and the next to the last row for C_{out}. When the energy factor $bestenergy = mismatches/num\_values$ is $zero$, a possible design has been found. A SPICE circuit is prepared for each table entry, storing the values in $spiceresults$. If $spiceresults$ match Sum and C_{out}, the ENERGY function returns $success$ and the program ends, otherwise the SA procedure continues searching for another possible design.

Algorithm 1 Ternary Adder

```
procedure SEARCH
    Initialize rows, columns, resistances
    Load ternary truth table A, B, C_in
    Randomize initial memristor states
    Initialize Simulated Annealing (SA)
    loop
        function ENERGY(mem)                    ▷ mem suggested memristor array
            Simulate Circuit Design
            Apply Current to row = 0
            Logically Determine Flow Based on Crossbar Design
            Read Resulting Voltages for C_out and Sum
            Calculate Energy Factor bestenergy
            if bestenergy > 0 then
                return bestenergy
            Prepare SPICE Circuit
            Compare SPICE Results to Acceptable Ranges spiceresults
            if spiceresults = sum and C_out then
                Report Results
                return success
            else
                return continue
```

Ternary Operations

Binary half-adders have two input bits that produce a Sum and C_{out} bit [48]. Ternary half-adders have two ternary inputs that outputs ternary values $(0, 1, 2)$ for Sum and C_{out}. Since the carry-in value C_{in} is excluded for a half-adder, the truth table 5.3 is simplified.

A ternary full-adder has three ternary inputs A, B, and C_{in} that produce output values of Sum and C_{out} based on Table 5.4.

The ternary multiplier has three ternary inputs A, B, and C_{in} that produces output values of $Product$ and C_{out} based on Table 5.5.

Table 5.3: Truth Table: Ternary Half-Adder

m	A	B	C_{in}	C_{out}	Sum
0	0	0	0	0	0
2	0	1	0	0	1
4	0	2	0	0	2
9	1	0	0	0	1
11	1	1	0	0	2
13	1	2	0	1	0
18	2	0	0	0	2
20	2	1	0	1	0
22	2	2	0	1	1

Table 5.4: Truth Table: Ternary Full-Adder

m	A	B	C_{in}	C_{out}	Sum
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	0	2	0	2
4	0	2	0	0	2
5	0	1	2	1	0
6	0	2	1	1	0
7	0	1	1	0	2
8	0	2	2	1	1
9	1	0	0	0	1
10	1	0	1	0	2
11	1	1	0	0	2
12	1	0	2	1	0
13	1	2	0	1	0
14	1	1	1	1	0
15	1	1	2	1	1
16	1	2	1	1	1
17	1	2	2	1	2
18	2	0	0	0	2
19	2	0	1	1	0
20	2	1	0	1	0
21	2	0	2	1	1
22	2	2	0	1	1
23	2	1	1	1	1
24	2	1	2	1	2
25	2	2	1	1	2
26	2	2	2	2	0

Table 5.5: Truth Table: Ternary Multiplication

m	A	B	C_{in}	C_{out}	$Product$
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	0
3	0	0	2	0	2
4	0	2	0	0	0
5	0	1	2	0	2
6	0	2	1	0	1
7	0	1	1	0	1
8	0	2	2	0	2
9	1	0	0	0	0
10	1	0	1	0	1
11	1	1	0	0	1
12	1	0	2	0	2
13	1	2	0	0	2
14	1	1	1	0	2
15	1	1	2	1	0
16	1	2	1	1	0
17	1	2	2	1	1
18	2	0	0	0	0
19	2	0	1	0	1
20	2	1	0	0	2
21	2	0	2	0	2
22	2	2	0	1	1
23	2	1	1	1	0
24	2	1	2	1	1
25	2	2	1	1	2
26	2	2	2	2	0

CHAPTER 6: FINDINGS

Ternary Adder

The initial search for a ternary design was a 4x4 memristive crossbar that outputs its Sum value on the last row the crossbar and a 6x5 array that outputs Sum on the last row and C_{out} on the next to the last row.

An input array of 11 values $(0, 1, 2, A, A', \neg A, B, B', \neg B, C_{in}, \neg C_{in})$ is prepared. The values of A and B are $(0, 1, 2)$ and C_{in} are $(0, 1)$. This results in a total of $3 * 3 * 2 = 18$ possible outcomes.

The voltage values of 0 and 1 are the same as for a binary system as for the ternary design. The value of 2 maps to a mid-range value on the crossbar and memristor. The goal is to search for a design that matches the Sum column on Table 4.2. The initial attempt resulted in a best energy of 0.333333 which represents 12 matches and 6 mismatches totaling 18 Sum values.

k	0	1	9	13	16	17
a	0	0	1	2	2	2
b	0	0	1	0	2	2
cin	0	1	0	0	0	1
sum	0	1	2	2	1	2
xbar	0	0	2	2	2	2
diff		X			X	

	Rows:	Columns:	Best Energy:
	4	3	0.333333
Memristor:	1	0	5
	5	4	4
	7	10	8
	0	3	6
k=0	1:1	0:0	5:2
	5:2	4:1	4:1
	7:1	10:1	8:2
	0:0	3:0	6:0
k=1	1:1	0:0	5:2
	5:2	4:1	4:1
	7:1	10:0	8:2
	0:0	3:0	6:0
k=16	1:1	0:0	5:1
	5:1	4:0	4:0
	7:0	10:1	8:1
	0:0	3:2	6:2
k=17	1:1	0:0	5:1
	5:1	4:0	4:0
	7:0	10:0	8:1
	0:0	3:2	6:2

The value k represents the truth table row. In the example above, the approximation was correct where $A = 1$, $B = 1$, and $C_{in} = 0$, but was incorrect where $A = 2$, $B = 2$, and $C_{in} = 0$. A number of changes were required achieve a better result including changing resistances, simulated annealing settings, and varying the mid, neg, and constant input values.

The values displayed for each k are in the form $x : y$. The value x indicates the memristor number and the value y is the best energy state returned by the GSL simulated annealing function. The input values are mapped in Table 6.1.

Table 6.1: Input values to SA function

Item	Variable	Value
0	Constant	0
1	Constant	1
2	Constant	2
3	A	truth table
4	A'	(A+1) $\mod 3$
5	$\neg$ A	(A+2) $\mod 3$
6	B	truth table
7	B'	(B+1) $\mod 3$
8	$\neg$ B	(B+2) $\mod 3$
9	C_{in}	truth table
10	$\neg$ C_{in}	$\neg C_{in}$

In some experiments, the adjustment to the constants and variables were randomized. In the table, A' and $\neg A$ are adjusted by 1 and 2, respectively. In one randomized version, the values were flipped (i.e. 2 and 1). In another case, we allowed duplicate values to see if we could achieve a better result.

The following results were obtained when the randomizer was used for each truth table row. The result of zero best energy was achieved after running simulated annealing for nearly 500 million steps. These results were considered an approximation since only a subset of the truth table was utilized.

SUCCESS:	Rows:	Columns:	Best Energy:
	4	3	0
Memristor:	6	1	3
	10	3	9
	6	7	3
	1	1	1
k=0	6:0	1:0	3:0
	10:1	3:0	9:0
	6:0	7:1	3:0
	1:0	1:0	1:0
k=1	6:0	1:1	3:0
	10:0	3:0	9:1
	6:0	7:2	3:0
	1:1	1:1	1:1
k=16	6:2	1:1	3:2
	10:1	3:2	9:0
	6:2	7:1	3:2
	1:1	1:1	1:1
k=17	6:2	1:2	3:2
	10:0	3:2	9:1
	6:2	7:1	3:2
	1:2	1:2	1:2

The cross reference Table 6.2 was discovered using NuSMV. The search took over 7 hours to complete. Once this table was loaded into the C++ program and a two level lookup was implemented, simulated annealing started to find simulated designs (i.e. best energy = 0) in 20 minutes. We then input the potential memristor into our Python-based SPICE checker to verify that the circuit simulation produced the same result. This method seemed to work well for 4x4 memristors with no carry-out bit.

Testing for the carry-out value was added resulting in possible solutions for 6x5 crossbars with some variation simulated in SPICE. The SPICE simulation was migrated to C++ to automate and speed up the process. The final step was to test for acceptable ranges of output from SPICE.

Table 6.2: Cross Reference derived from NuSMV

mem	0	1	2	mem	0	1	2
0	0	0	0	14	1	1	1
1	0	0	1	15	1	1	2
2	0	1	0	16	1	2	1
3	0	0	2	17	1	2	2
4	0	2	0	18	2	0	0
5	0	1	2	19	2	0	1
6	0	2	1	20	2	1	0
7	0	1	1	21	2	0	2
8	0	2	2	22	2	2	0
9	1	0	0	23	2	1	1
10	1	0	1	24	2	1	2
11	1	1	0	25	2	2	1
12	1	0	2	26	2	2	2
13	1	2	0				

This check was moved to the energy function when the best energy was calculated as zero (i.e. every value matched to the truth table). The last change was to continue the SA process when the SPICE checker didn't fall into a preset acceptable range for each memristor state. This produced multiple simulations resulting in the selection of memristive crossbar designs that might perform more efficiently with less potential for error.

In the tests using NuSMV, it was discovered that better results were achieved when using a full 3-bit truth table containing $3^3 = 27$ rows. When implementing SA, any truth table entry where $C_{in} = 2$ was excluded from the energy calculation, so those items would not appear in the final solution. Figure 6.1 is an example of memristor placement for a 4x4 ternary adder with no carry-out bit.

Once the design in Figure 6.1 was found, it was tested using SPICE. The SPICE column shows the simulated voltage of the last row given the inputs A, B, and C_{in}. The resistance and voltage ranges were varied in testing. The SPICE results can be clearly mapped to the Sum bit in Table 6.3.

Table 6.3: SPICE: Ternary Adder 4x4 with no carry-out bit

A	B	C_{in}	Sum	$SPICE$
0	0	0	0	0.000080
0	0	1	1	0.007803
0	1	0	1	0.007828
0	2	0	2	0.200024
0	2	1	0	0.000060
0	1	1	2	0.200339
1	0	0	1	0.007862
1	0	1	2	0.201286
1	1	0	2	0.200028
1	2	0	0	0.000060
1	1	1	0	0.000080
1	2	1	1	0.007823
2	0	0	2	0.200027
2	0	1	0	0.000080
2	1	0	0	0.000060
2	2	0	1	0.007816
2	1	1	1	0.007823
2	2	1	2	0.200336

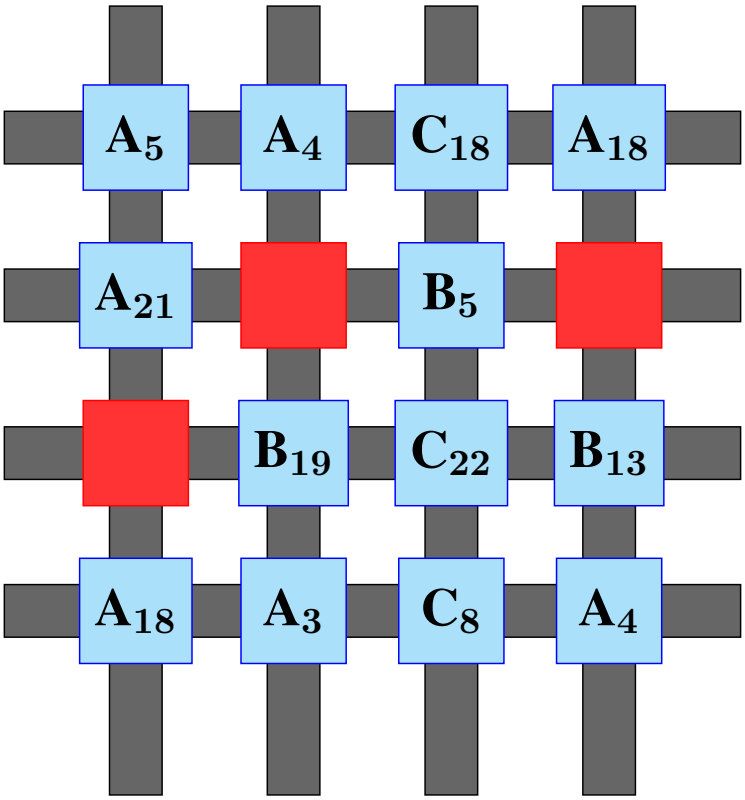

Figure 6.1: Crossbar design for 4x4 ternary adder with no carry-out bit

To implement the carry-out value, the size of the crossbar array was increased to $6x5$ and SA energy function was modified to check for the carry-out bit. The last row is checked for the Sum bit while the next to the last row is checked for C_{out}. Figure 6.2 is the design of a crossbar for a 1-bit adder. It is observed that the SPICE results are within acceptable ranges.

Ternary Full-Adder

Based on the success of the initial tests, the next step was to create a ternary full-adder using the entire ternary truth table as shown in 5.4 and defined in Algorithm 2.

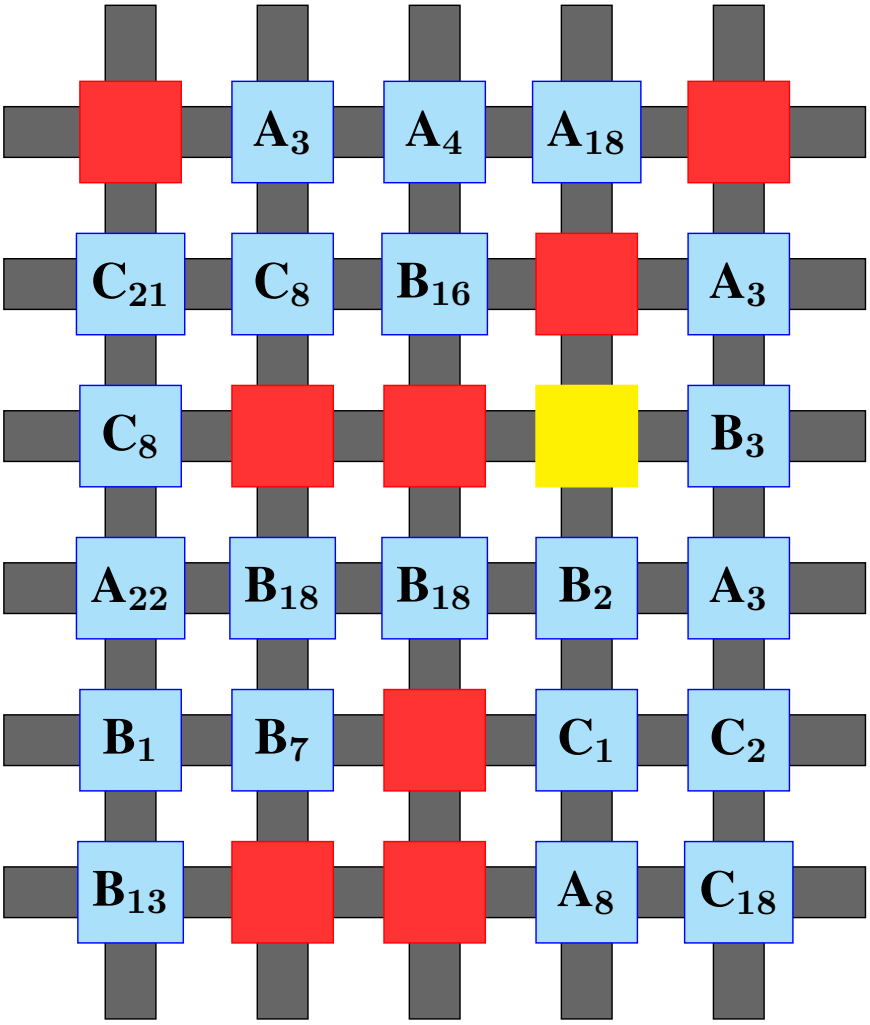

Figure 6.2: Crossbar design for 1-bit ternary addition

Algorithm 2 Ternary Adder

global values
    $a[] \leftarrow$ [0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2] ▷ ternary truth table
    $b[] \leftarrow$ [0,0,1,0,2,1,2,1,2,0,0,1,0,2,1,1,2,2,0,0,1,0,2,1,1,2,2]
    $c[] \leftarrow$ [0,1,0,2,0,2,1,1,2,0,1,0,2,0,1,2,1,2,0,1,0,2,0,1,2,1,2]
    $u[] \leftarrow$ [0,0,0,0,0,1,1,0,1,0,0,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,2]
    $s[] \leftarrow$ [0,1,1,2,2,0,0,2,1,1,2,2,0,0,0,1,1,2,2,0,0,1,1,1,2,2,0]
end global values

The MAIN procedure in Algorithm 3 calculates an initial array of memristors that is passed to the GSL Simulated Annealing (SA) function. If the SA function is returned to MAIN, a sub-optimal solution was found. The best solution is checked with a SPICE sub-circuit and the output is reported.

Algorithm 3 Ternary Adder- Main

```
1:  procedure MAIN
2:      Call random number generator with seed value
3:      for i ← 1 to rows do
4:          for j ← 1 to cols do
5:              k ← rand mod dict
6:              if k ≥ static then
7:                  mem ← (k - static) mod inputs
8:              else
9:                  k = k mod bits
10:             initstate[i * cols + j] = k
11:     Call gsl_siman_solve                          ▷ GSL Simulated Annealing
12:     Call OutputReport
13:     Call OutputCheck                                  ▷ SPICE Simulation
        return "Suboptimal solution found"
```

The SA energy callback function in Algorithm 4 is used to check whether an input array of memristors results in a valid circuit. The function iterates through each row on the truth table using flow-based analysis based on the value of each memristor. The resulting Sum and C_{out} values are determined by the resulting value on the last two rows. A best energy value between 0 and 1 is returned to the calling function. The value is determined by the formula $currentenergy \leftarrow mismatchcount$ / ($matchcount$ + $mismatchcount$. A value of $currentenergy = 0$ represents a potential circuit which is then checked using a SPICE sub-circuit based on the values of each memristor.

The memristor array that is passed into the energy function is split into two parts, an input variable (ex. A, B, C_{in} and truth table position between 0 and 26. Static values of 0, 1, and 2 are also possible. The state of the memristor is determined by the truth table value

Algorithm 4 Ternary Adder- GSL SA Energy Function

```
1: procedure ENERGY(*memrs)
2:     for k ← 1 to inputs do
3:         Calculate the value of each memristor based on truth table entry k
4:         for each i=row and j=column do
5:             m ← Memristor variable A, B, or C_in
6:             sel ← Truth table value at row k for variable m
7:             val ← Numeric part of memristor
8:             xbar[i, j] ← Value of truth table entry sel, val
9:             row[1] ← memon                    ▷ simulate voltage applied to first row
10:            loop while changes are occurring to circuit
11:                for each i=row and j=column do
12:                    Estimate current for each row and column using flow-based analysis
13:                    If memristor state is mem_hi, the highest row/column current is used
14:                    If memristor state is mem_mid, the maximum current is mem_mid
15:                    If memristor state is mem_low, no changes are made to current at crossbar
16:    currentenergy ← mismatchcount / (matchcount + mismatchcount)
17:    if currentenergy := 0 then
18:        Call OutputReport
19:        mismatchcount ← Call OutputCheck                    ▷ SPICE Simulation
20:        if mismatchcount := 0 then
21:            EXIT success
22:    Return currentenergy
```

The following are successful tests of a ternary adder using resistances of $mem\_hi = 1e + 06$, $mem\_mid = 10000$, and $mem\_low = 100$. The voltage midpoints of 0.003 and 0.25 were determined by finding the average delta between each desired result.

Table 6.4: SPICE Simulation: 7x7 Crossbar

ABC-in	Sum	SPICE-Sum	C-out	SPICE-C-out
000	0	0.001685	0	0.000230
001	1	0.201411	0	0.000321
010	1	0.200552	0	0.000635
002	2	0.024407	0	0.000295
020	2	0.010261	0	0.001198
012	0	0.001970	1	0.333405
021	0	0.001061	1	0.250621
011	2	0.010826	0	0.000882
022	1	0.080744	1	0.156926
100	1	0.200174	0	0.000603
101	2	0.019225	0	0.000418
110	2	0.010313	0	0.001077
102	0	0.001059	1	0.147804
120	0	0.001115	1	0.199882
111	0	0.001273	1	0.199771
112	1	0.102247	1	0.303276
121	1	0.150553	1	0.150228
122	2	0.005414	1	0.168877
200	2	0.019054	0	0.000645
201	0	0.001162	1	0.333047
210	0	0.001173	1	0.199828
202	1	0.028480	1	0.124897
220	1	0.103217	1	0.057031
211	1	0.044280	1	0.304228
212	2	0.003602	1	0.333539
221	2	0.009425	1	0.358887
222	0	0.000304	2	0.011560

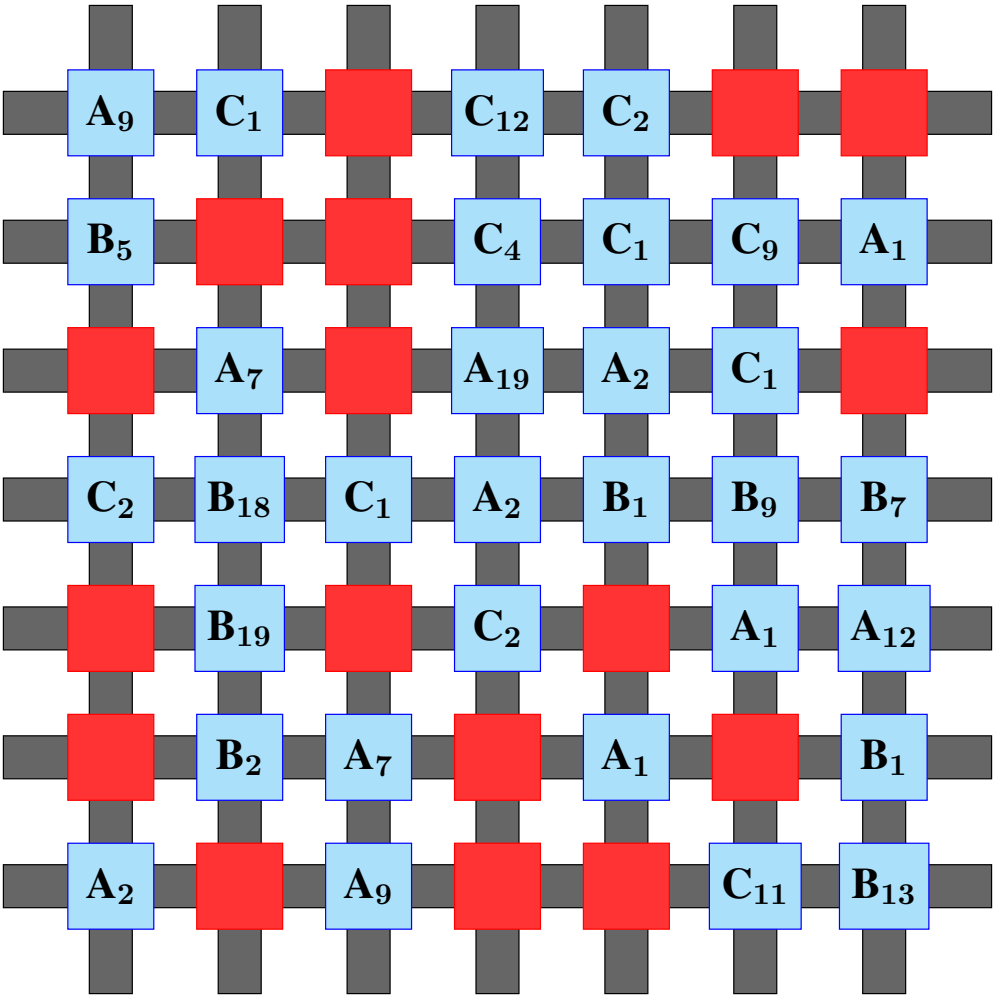

Figure 6.3: Ternary Adder: 7x7 Crossbar

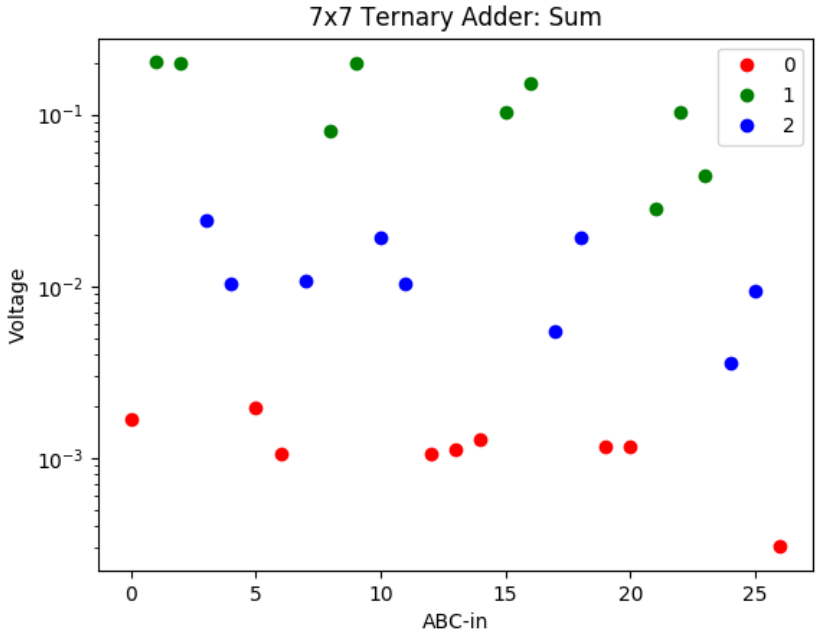

Figure 6.4: SPICE Results: 7x7 Ternary Full Adder- SUM

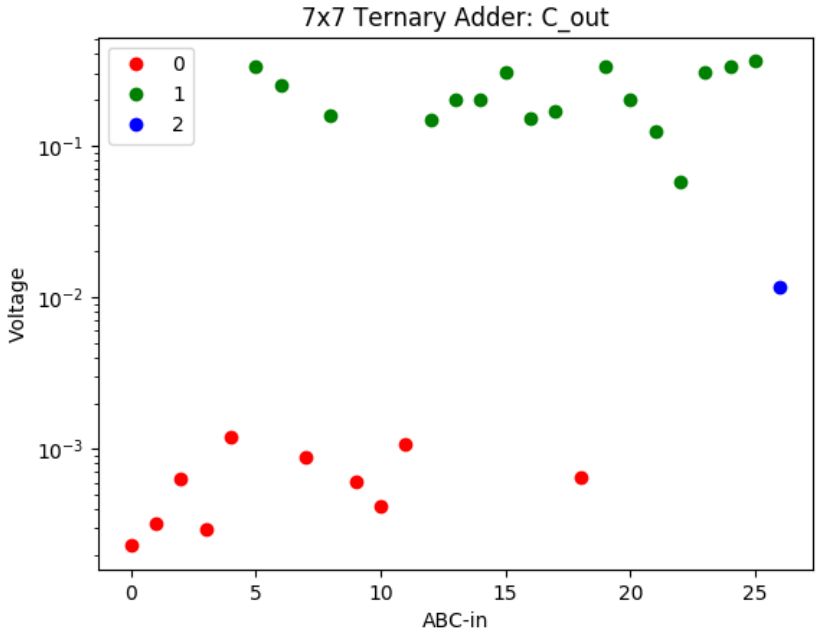

Figure 6.5: SPICE Results: 7x7 Ternary Full Adder- Carry

In the next 8x8 crossbar test, the values for 1 and 2 were flipped. In this experiment, $1 = mem\_mid$ and $2 = mem\_hi$. Since C_{out} only has one possible result equal to 2, it was thought that this might result in a larger tolerance between resulting values. The resistances used were $mem\_hi = 1e{+}06$, $mem\_mid = 10000$, and $mem\_low = 100$ and the voltage midpoints were 0.00275 and 0.080.

Table 6.5: Ternary Adder: 8x8 Crossbar (012)

ABC-in	Sum	SPICE-Sum	C-out	SPICE-C-out
000	0	0.001170	0	0.000211
001	1	0.010416	0	0.000360
010	1	0.010204	0	0.000454
002	2	0.199779	0	0.001312
020	2	0.201326	0	0.001311
012	0	0.001268	1	0.004581
021	0	0.001674	1	0.003733
011	2	0.199869	0	0.001366
022	1	0.010280	1	0.016106
100	1	0.020008	0	0.000339
101	2	0.204043	0	0.000869
110	2	0.201852	0	0.001334
102	0	0.001887	1	0.005810
120	0	0.001891	1	0.003381
111	0	0.002170	1	0.005224
112	1	0.020407	1	0.005617
121	1	0.020719	1	0.005096
122	2	0.200488	1	0.007451
200	2	0.144609	0	0.001170
201	0	0.001561	1	0.010075
210	0	0.001930	1	0.019376
202	1	0.011755	1	0.017998
220	1	0.010883	1	0.010872
211	1	0.004618	1	0.007216
212	2	0.137306	1	0.024590
221	2	0.196218	1	0.023525
222	0	0.001506	2	0.236781

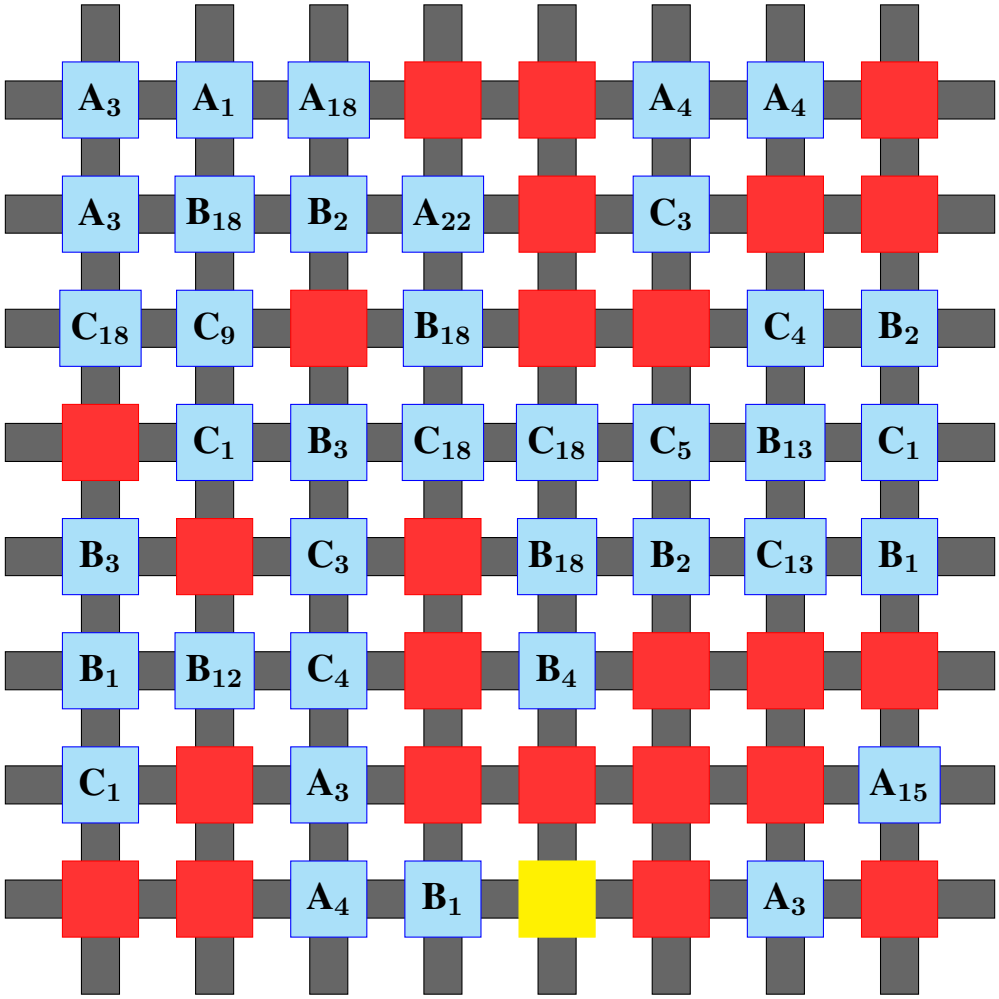

Figure 6.6: Ternary Adder: 8x8 Crossbar (012)

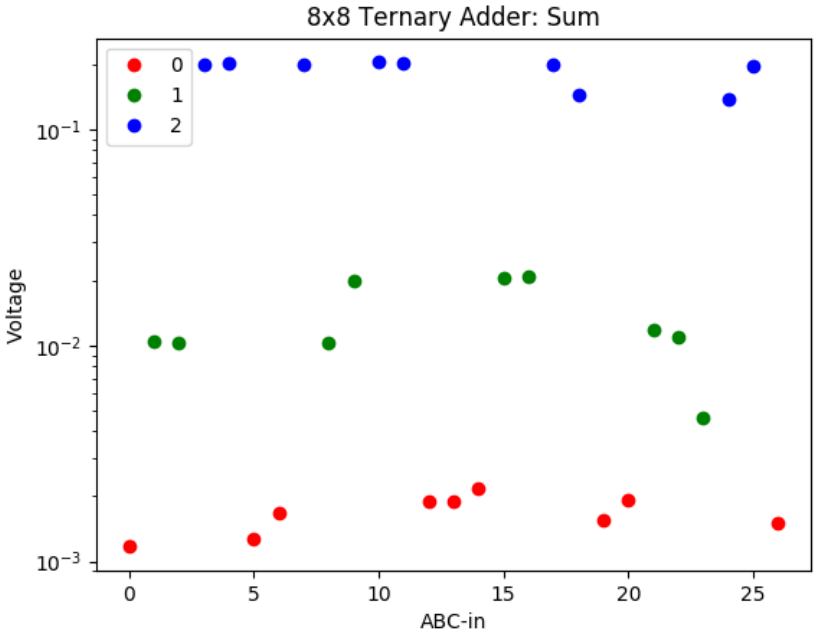

Figure 6.7: SPICE Results: 8x8 Ternary Full Adder (012)- SUM

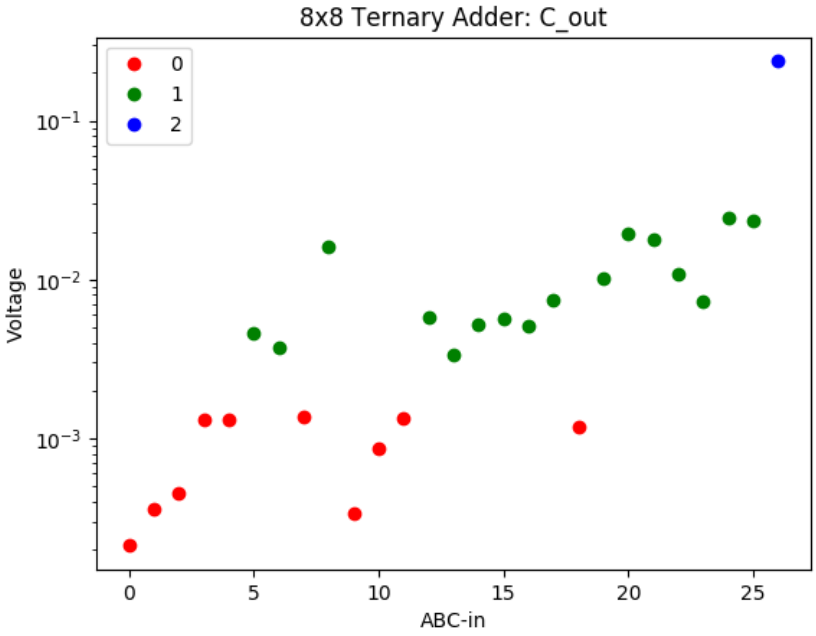

Figure 6.8: SPICE Results: 8x8 Ternary Full Adder (012)- Carry

In the following 8x8 crossbar test, the values for 1 and 2 were set back to the original $2 = mem\_mid$ and $1 = mem\_hi$. The resistances used were $mem\_hi = 1e + 06$, $mem\_mid = 10000$, and $mem\_low = 100$ and the voltage midpoints were 0.003 and 0.038.

Table 6.6: Ternary Adder: 8x8 Crossbar (021)

ABC-in	Sum	SPICE-Sum	C-out	SPICE-C-out
000	0	0.001290	0	0.000103
001	1	0.200937	0	0.000295
010	1	0.145380	0	0.000260
002	2	0.011023	0	0.001084
020	2	0.019923	0	0.000427
012	0	0.001611	1	0.208595
021	0	0.001318	1	0.333050
011	2	0.011687	0	0.000300
022	1	0.144542	1	0.156188
100	1	0.115627	0	0.000822
101	2	0.011205	0	0.000605
110	2	0.005918	0	0.001531
102	0	0.001114	1	0.290627
120	0	0.001884	1	0.200131
111	0	0.001762	1	0.199599
112	1	0.143393	1	0.242766
121	1	0.045034	1	0.303812
122	2	0.004108	1	0.245840
200	2	0.005601	0	0.000756
201	0	0.001806	1	0.144311
210	0	0.001360	1	0.143828
202	1	0.145150	1	0.150409
220	1	0.097161	1	0.161720
211	1	0.099343	1	0.160879
212	2	0.004573	1	0.259182
221	2	0.006581	1	0.329366
222	0	0.000130	2	0.029392

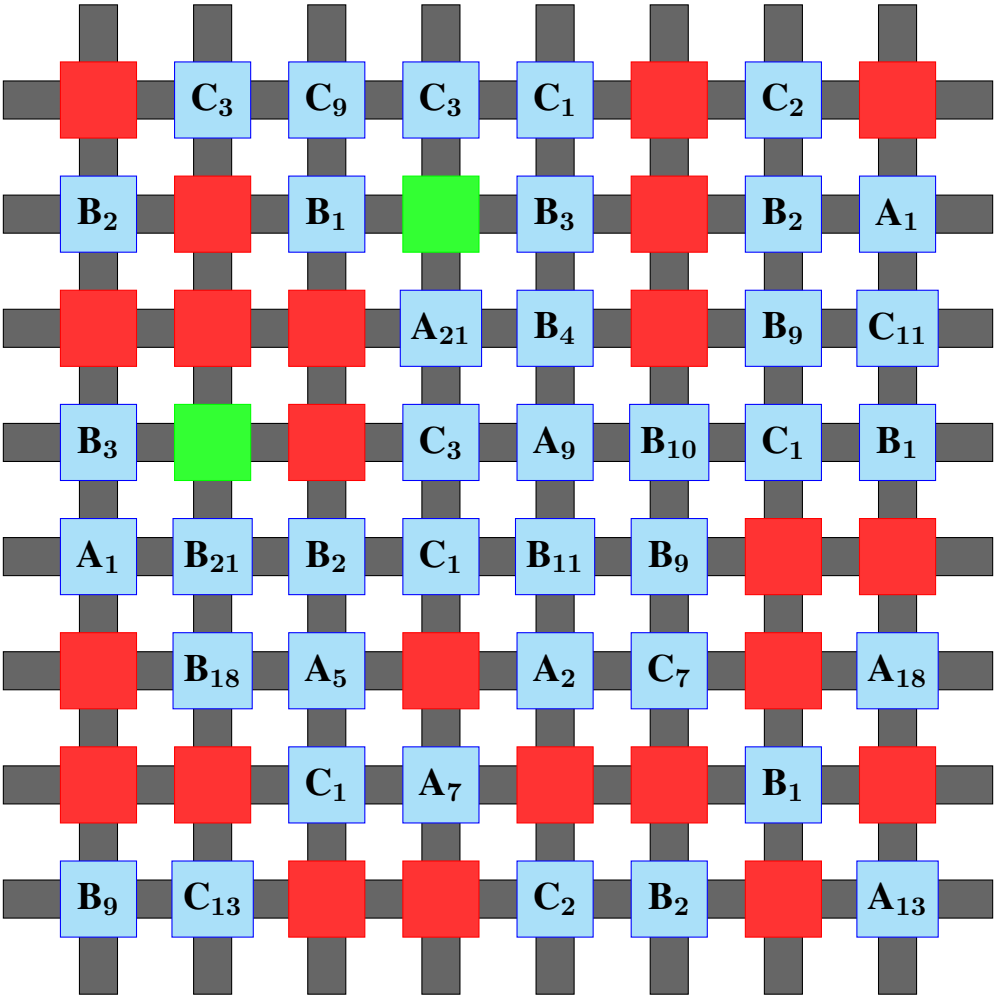

Figure 6.9: Ternary Adder: 8x8 Crossbar (021)

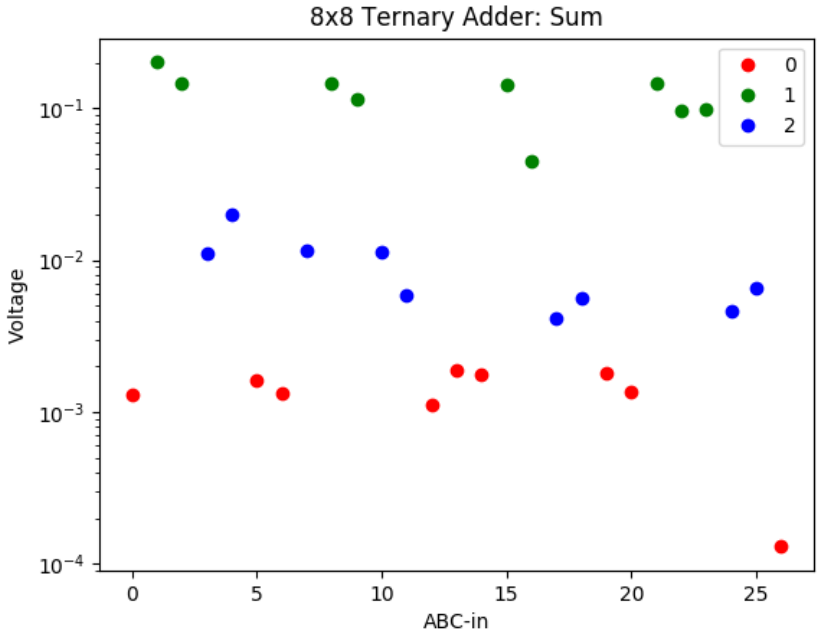

Figure 6.10: SPICE Results: 8x8 Ternary Full Adder (021)- SUM

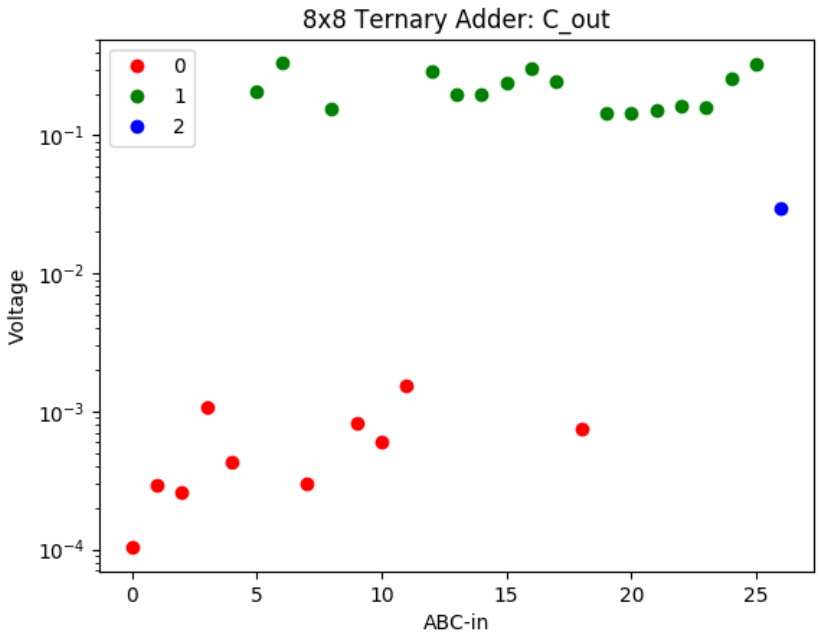

Figure 6.11: SPICE Results: 8x8 Ternary Full Adder (021)- Carry

Ternary Multiplication

The methodology to create a ternary adder can be applied to other operations, including multiplication. A ternary multiplier was designed using truth table 6.7 defined in Algorithm 5.

Algorithm 5 Ternary Multiplication

1: **global values**
2: $a[] \leftarrow$ [0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2] ▷ ternary truth table
3: $b[] \leftarrow$ [0,0,1,0,2,1,2,1,2,0,0,1,0,2,1,1,2,2,0,0,1,0,2,1,1,2,2]
4: $c[] \leftarrow$ [0,1,0,2,0,2,1,1,2,0,1,0,2,0,1,2,1,2,0,1,0,2,0,1,2,1,2]
5: $v[] \leftarrow$ [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,1,1,1,1,2]
6: $p[] \leftarrow$ [0,1,0,2,0,2,1,1,2,0,1,1,2,2,2,0,0,1,0,1,2,2,1,0,1,2,0]
7: **end global values**

Table 6.7: Truth Table: Ternary Multiplication

A	B	C_{in}	C_{out}	$Product$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	0	2	0	2
0	2	0	0	0
0	1	2	0	2
0	2	1	0	1
0	1	1	0	1
0	2	2	0	2
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	0	2	0	2
1	2	0	0	2

A	B	C_{in}	C_{out}	$Product$
1	1	1	0	2
1	1	2	1	0
1	2	1	1	0
1	2	2	1	1
2	0	0	0	0
2	0	1	0	1
2	1	0	0	2
2	0	2	0	2
2	2	0	1	1
2	1	1	1	0
2	1	2	1	1
2	2	1	1	2
2	2	2	2	0

The same tests were run for multiplication as for ternary addition. In the 7x7 crossbar test, the resistances used were $mem\_hi = 1e + 06$, $mem7\_mid = 10000$, and $mem\_low = 100$ and the voltage midpoints were 0.005 and 0.073.

Table 6.8: Ternary Multiplication: 7x7 Crossbar

ABC-in	Prod	SPICE-Prod	C-out	SPICE-C-out
000	0	0.000441	0	0.000257
001	1	0.375242	0	0.000342
010	0	0.001200	0	0.000834
002	2	0.010409	0	0.000011
020	0	0.001247	0	0.001006
012	2	0.015699	0	0.000645
021	1	0.334094	0	0.000723
011	1	0.333481	0	0.000613
022	2	0.011033	0	0.000319
100	0	0.000432	0	0.000268
101	1	0.252281	0	0.000380
110	1	0.200764	0	0.000976
102	2	0.010277	0	0.000016
120	2	0.027418	0	0.000849
111	2	0.010495	0	0.001033
112	0	0.001388	1	0.210561
121	0	0.001638	1	0.199815
122	1	0.109146	1	0.122912
200	0	0.000653	0	0.000511
201	1	0.258248	0	0.000463
210	2	0.010434	0	0.000798
202	2	0.010663	0	0.000109
220	1	0.139452	1	0.136182
211	0	0.001248	1	0.200110
212	1	0.166118	1	0.087771
221	2	0.006807	1	0.207492
222	0	0.000136	2	0.019944

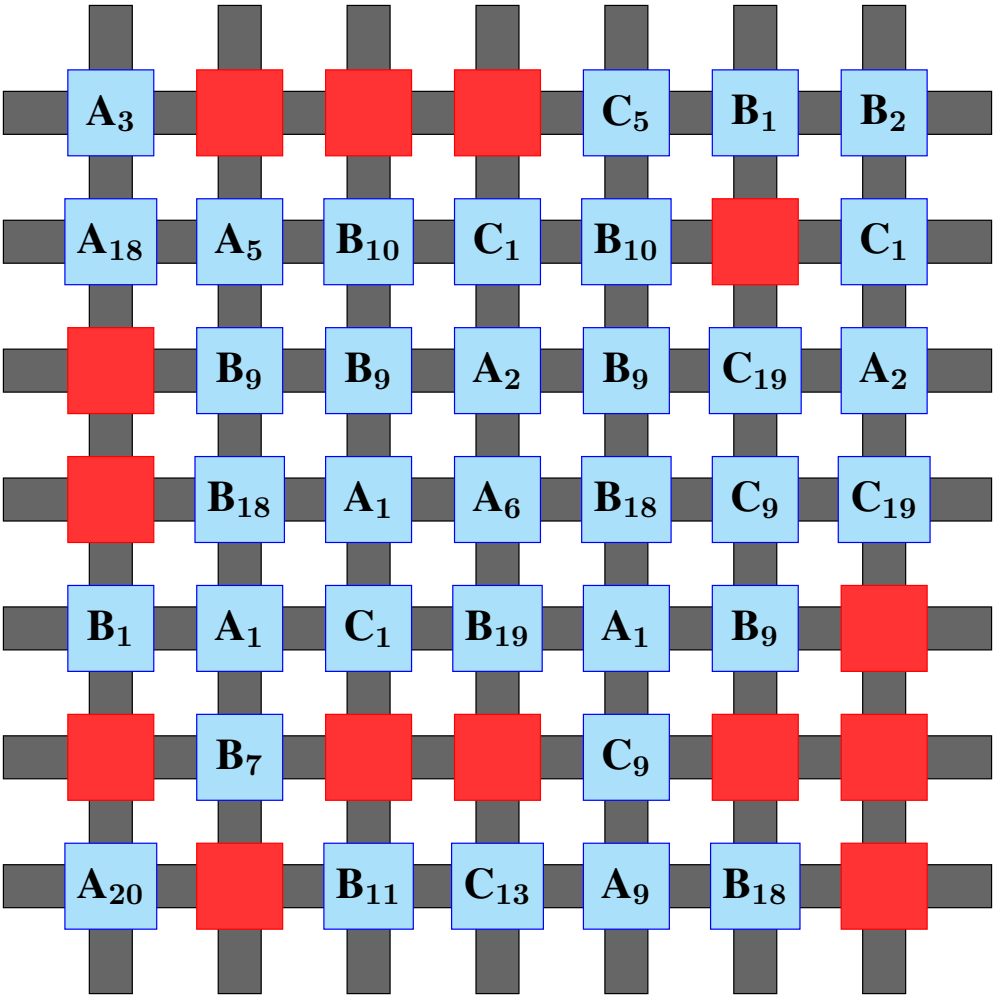

Figure 6.12: Ternary Multiplication: 7x7 Crossbar

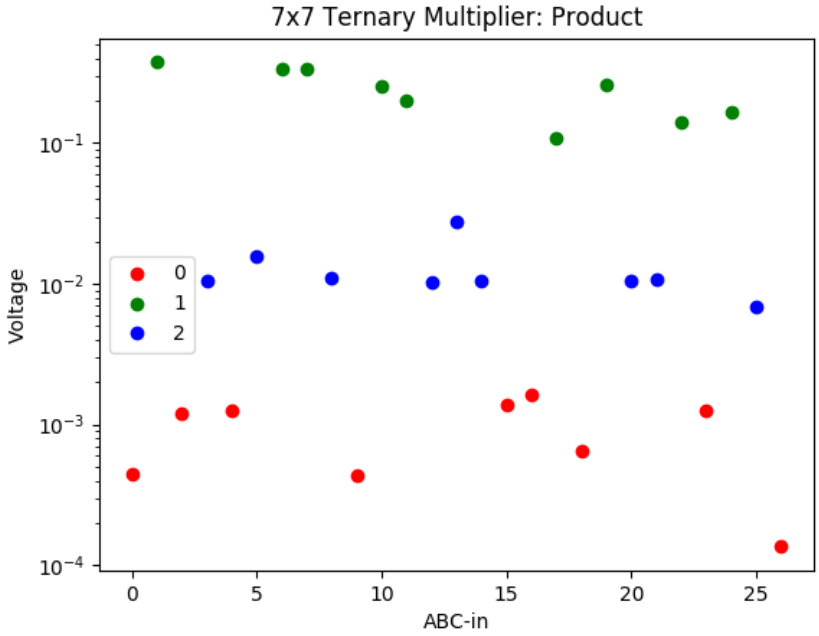

Figure 6.13: SPICE Results: 7x7 Ternary Multiplication- SUM

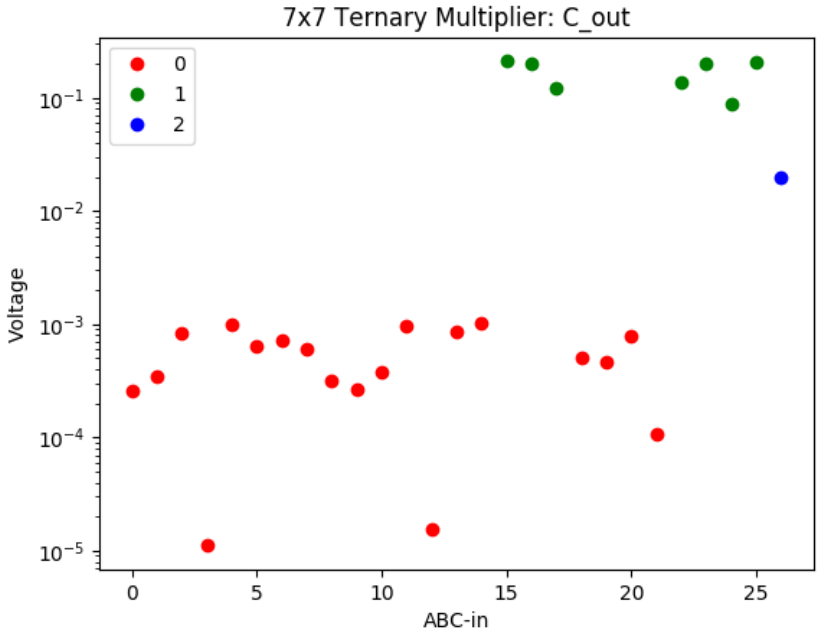

Figure 6.14: SPICE Results: 7x7 Ternary Multiplication- Carry

In the 8x8 crossbar test for multiplication, the values for 1 and 2 were again flipped. C_{out} only has one possible result equal to 2. The resistances used were $mem\_hi = 1e+06$, $mem\_mid = 10000$, and $mem\_low = 100$ and the voltage midpoints were 0.0029 and 0.080.

Table 6.9: Ternary Multiplication: 8x8 Crossbar (012)

ABC-in	Prod	SPICE-Prod	C-out	SPICE-C-out
000	0	0.000544	0	0.000252
001	1	0.019782	0	0.000038
010	0	0.000594	0	0.000204
002	2	0.259399	0	0.000467
020	0	0.000583	0	0.000215
012	2	0.178732	0	0.000906
021	1	0.015880	0	0.000234
011	1	0.015014	0	0.000220
022	2	0.157622	0	0.000881
100	0	0.001189	0	0.000801
101	1	0.020779	0	0.000221
110	1	0.019892	0	0.001898
102	2	0.259852	0	0.000611
120	2	0.149120	0	0.001652
111	2	0.200641	0	0.001424
112	0	0.001164	1	0.004803
121	0	0.001601	1	0.006089
122	1	0.004848	1	0.013958
200	0	0.001017	0	0.000378
201	1	0.016866	0	0.000174
210	2	0.153724	0	0.001317
202	2	0.209636	0	0.000544
220	1	0.019433	1	0.006247
211	0	0.001861	1	0.003981
212	1	0.007555	1	0.010164
221	2	0.143934	1	0.006056
222	0	0.001134	2	0.212312

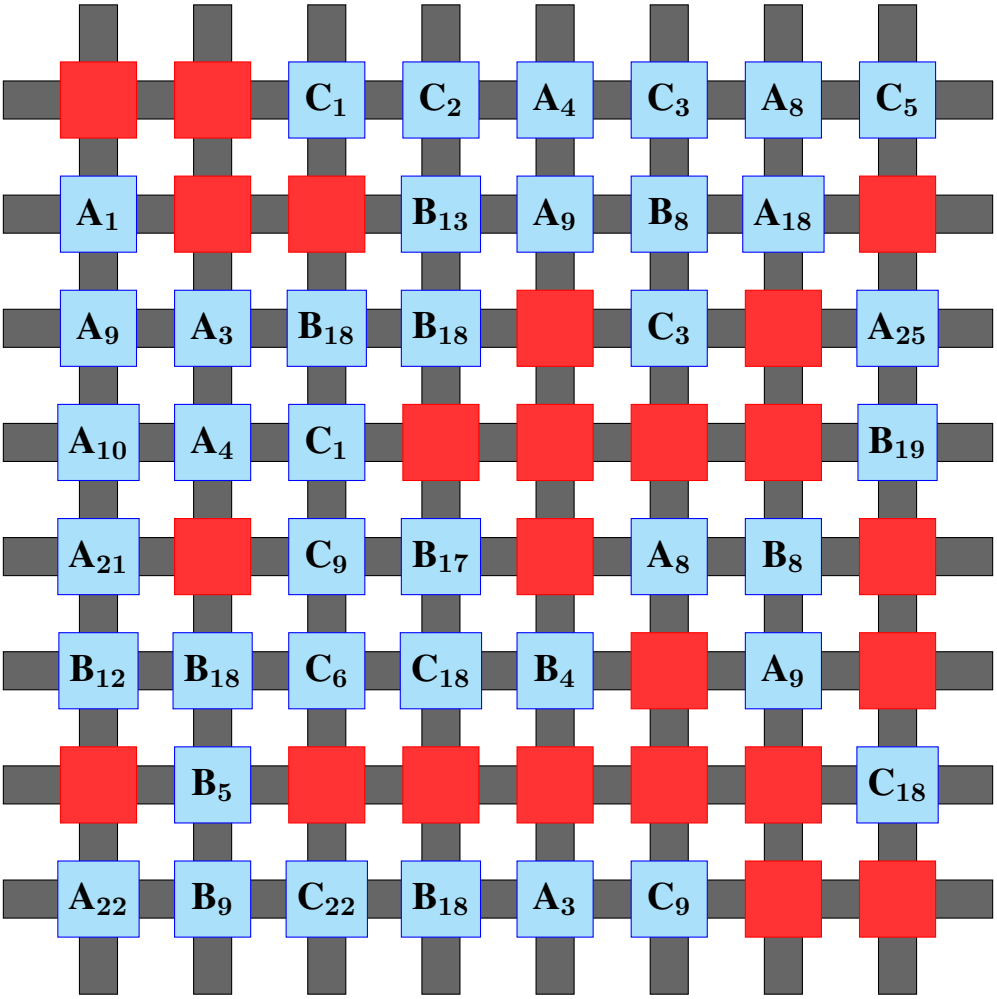

Figure 6.15: Ternary Multiplication: 8x8 Crossbar (012)

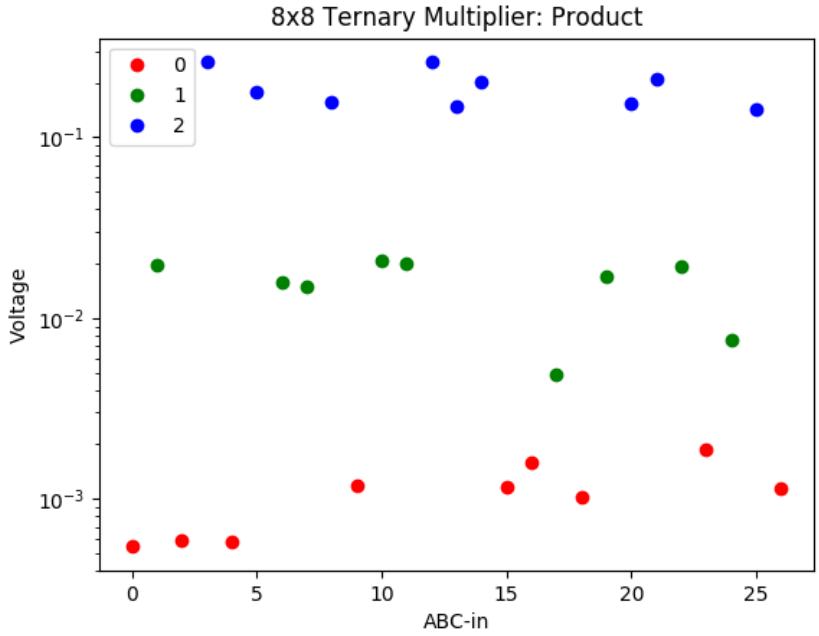

Figure 6.16: SPICE Results: 8x8 Ternary Multiplication (012)- SUM

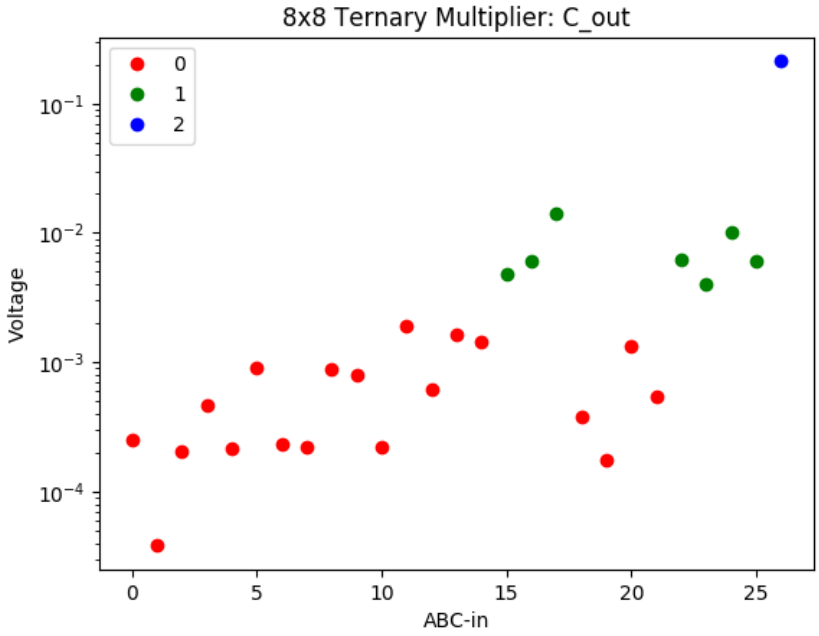

Figure 6.17: SPICE Results: 8x8 Ternary Multiplication (012)- Carry

In the following 8x8 crossbar test, the values for 1 and 2 were set back to the original $2 = mem\_mid$ and $1 = mem\_hi$. The resistances used were $mem\_hi = 1e + 06$, $mem\_mid = 10000$, and $mem\_low = 100$ and the voltage midpoints were 0.0048 and 0.0248.

Table 6.10: Ternary Multiplication: 8x8 Crossbar (021)

ABC-in	Prod	SPICE-Prod	C-out	SPICE-C-out
000	0	0.000797	0	0.000001
001	1	0.437246	0	0.000536
010	0	0.001138	0	0.000231
002	2	0.010367	0	0.000018
020	0	0.001192	0	0.000203
012	2	0.010767	0	0.000225
021	1	0.369672	0	0.000936
011	1	0.334825	0	0.000884
022	2	0.010741	0	0.000227
100	0	0.000796	0	0.000002
101	1	0.213973	0	0.000590
110	1	0.112541	0	0.000989
102	2	0.007651	0	0.000208
120	2	0.010903	0	0.001311
111	2	0.011178	0	0.001192
112	0	0.000450	1	0.115094
121	0	0.002129	1	0.200214
122	1	0.102857	1	0.078814
200	0	0.000797	0	0.000001
201	1	0.331818	0	0.000551
210	2	0.015180	0	0.001286
202	2	0.010164	0	0.000029
220	1	0.180130	1	0.026985
211	0	0.001965	1	0.200338
212	1	0.103446	1	0.078853
221	2	0.011687	1	0.249980
222	0	0.000581	2	0.019489

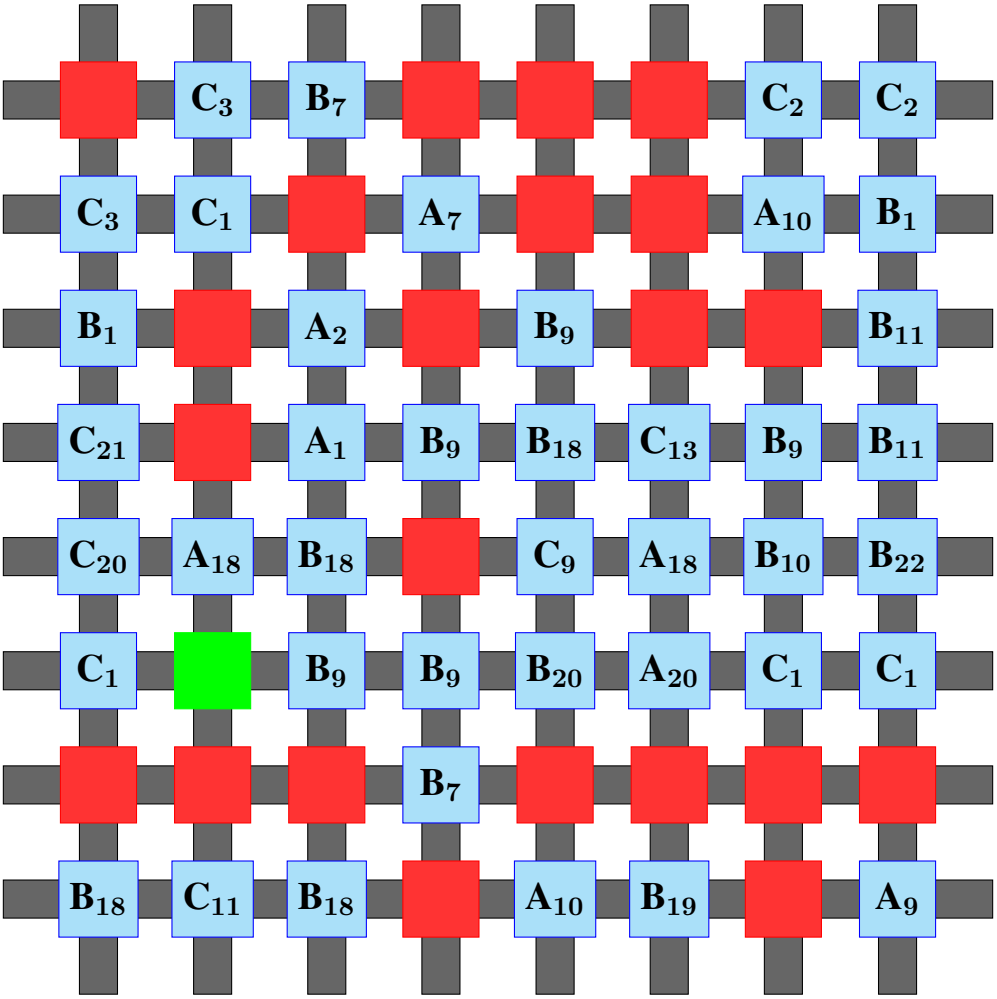

Figure 6.18: Ternary Multiplication: 8x8 Crossbar (021)

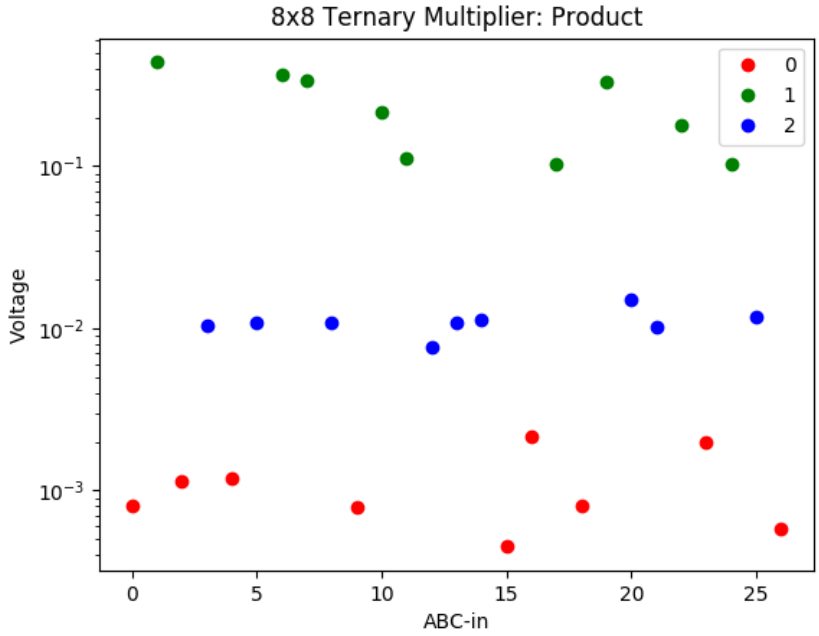

Figure 6.19: SPICE Results: 8x8 Ternary Multiplication (021)- SUM

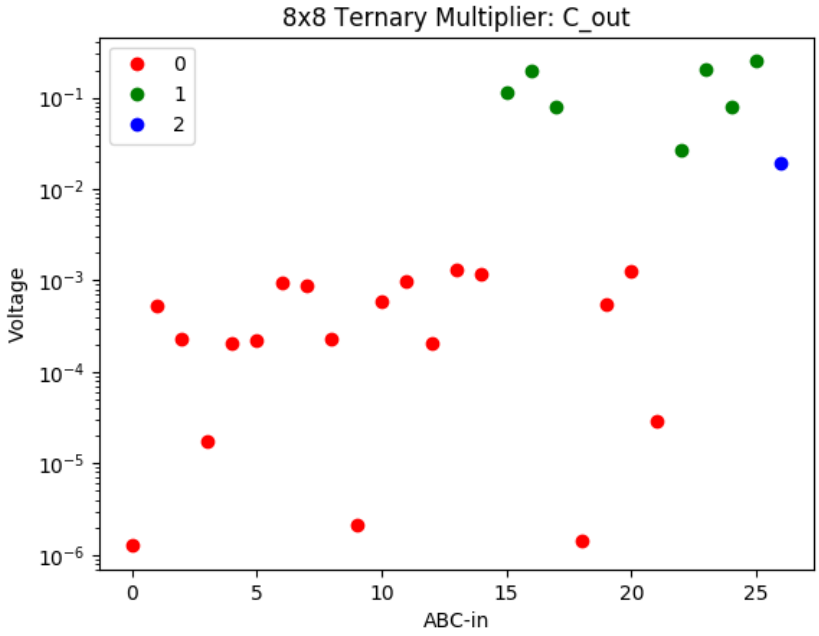

Figure 6.20: SPICE Results: 8x8 Ternary Multiplication (021)- Carry

In the previous experiments, acceptable range limits were calculated based on the average SPICE values of valid flow-based designs. For the following test, the algorithm was altered to include fixed ranges and multiple SPICE tests to reduce outliers and produce tighter ranges.

This 8x8 ternary adder used the values $1 = mem\_mid$ and $2 = mem\_hi$. The resistances used were $mem\_hi = 1e + 06$, $mem\_mid = 10000$, and $mem\_low = 100$ and the voltage midpoints were 0.0021 and 0.0805.

Table 6.11: Enhanced Ternary Addition: 8x8 Crossbar (012)

ABC-in	Prod	SPICE-Prod	C-out	SPICE-C-out
000	0	0.000182	0	0.000066
001	1	0.007695	0	0.000066
010	1	0.007841	0	0.000031
002	2	0.144499	0	0.000098
020	2	0.200315	0	0.000174
012	0	0.000170	1	0.007994
021	0	0.000155	1	0.004060
011	2	0.200034	0	0.000105
022	1	0.007898	1	0.011768
100	1	0.007810	0	0.000069
101	2	0.202304	0	0.000067
110	2	0.199991	0	0.000109
102	0	0.000168	1	0.007739
120	0	0.000161	1	0.007161
111	0	0.000189	1	0.004050
112	1	0.007804	1	0.011854
121	1	0.007907	1	0.011204
122	2	0.207165	1	0.011880
200	2	0.145351	0	0.000089
201	0	0.000171	1	0.005780
210	0	0.000118	1	0.007887
202	1	0.013365	1	0.032476
220	1	0.014953	1	0.015480
211	1	0.007874	1	0.007903
212	2	0.197981	1	0.016423
221	2	0.210937	1	0.008040
222	0	0.000144	2	0.238636

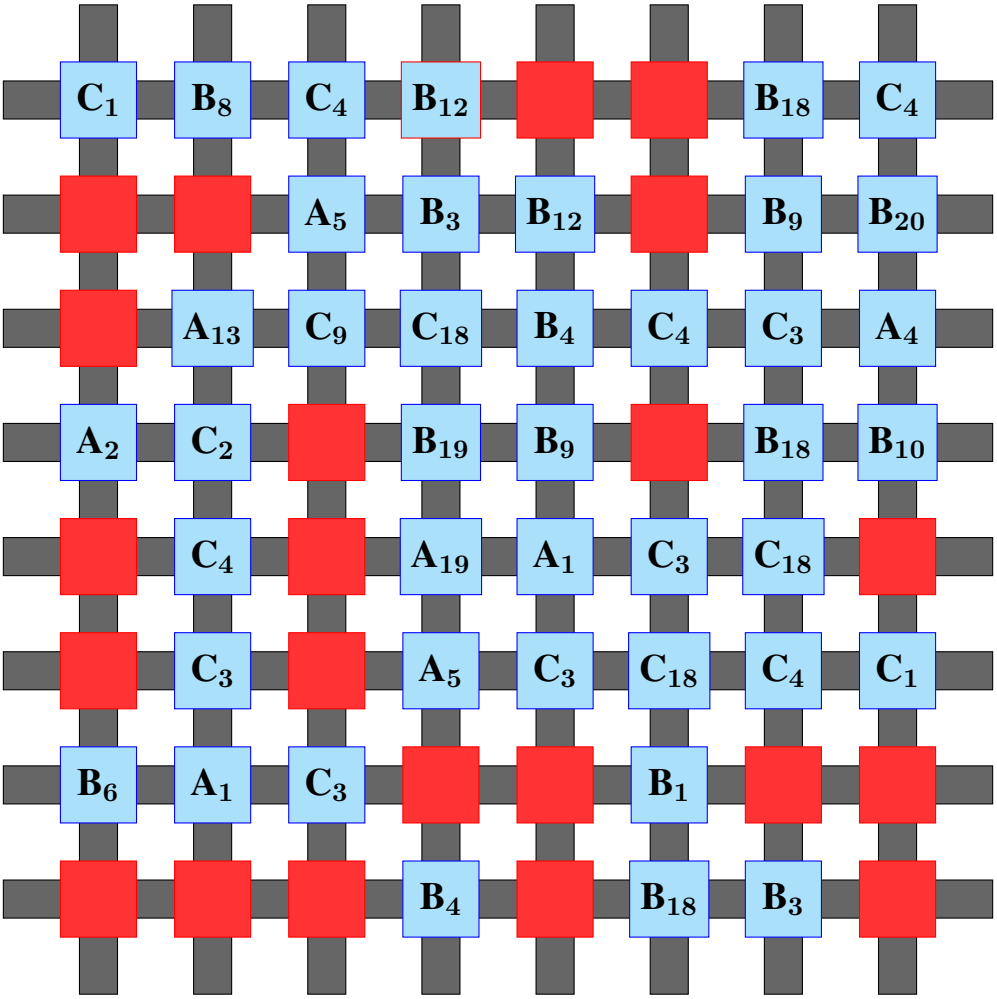

Figure 6.21: Enhanced Ternary Adder: 8x8 Crossbar (012)

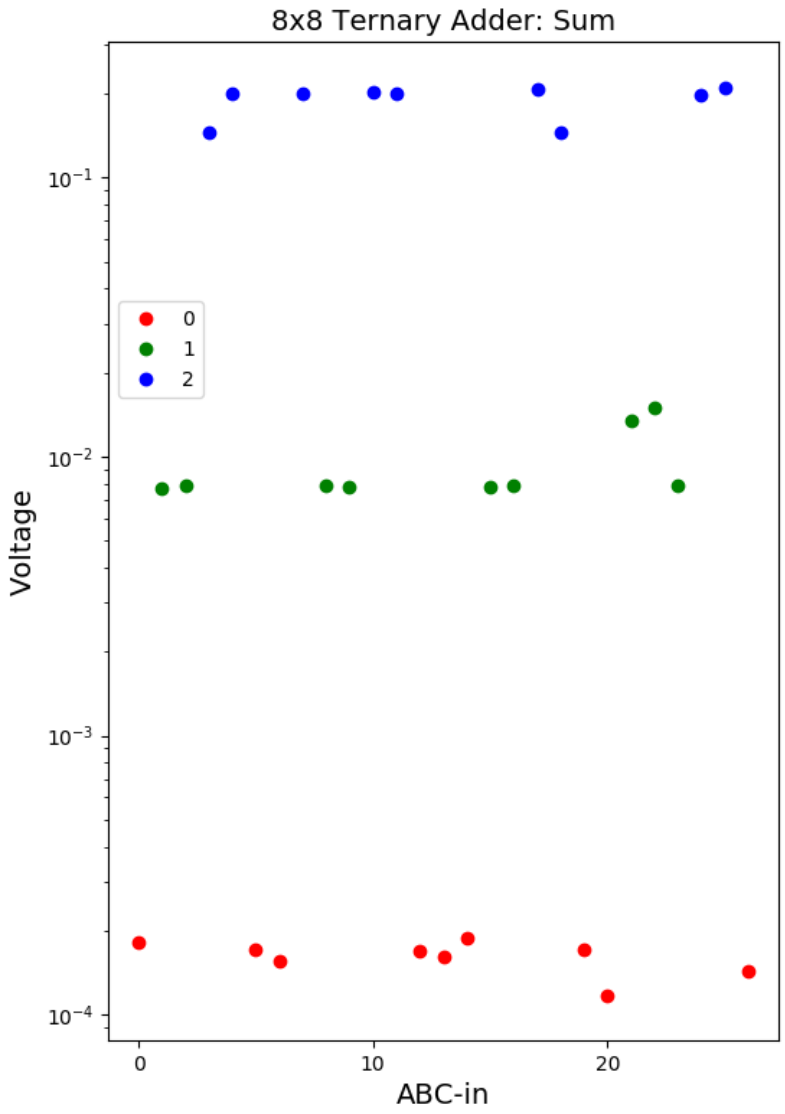

Figure 6.22: SPICE Results: 8x8 Enhanced Ternary Adder (012)- SUM

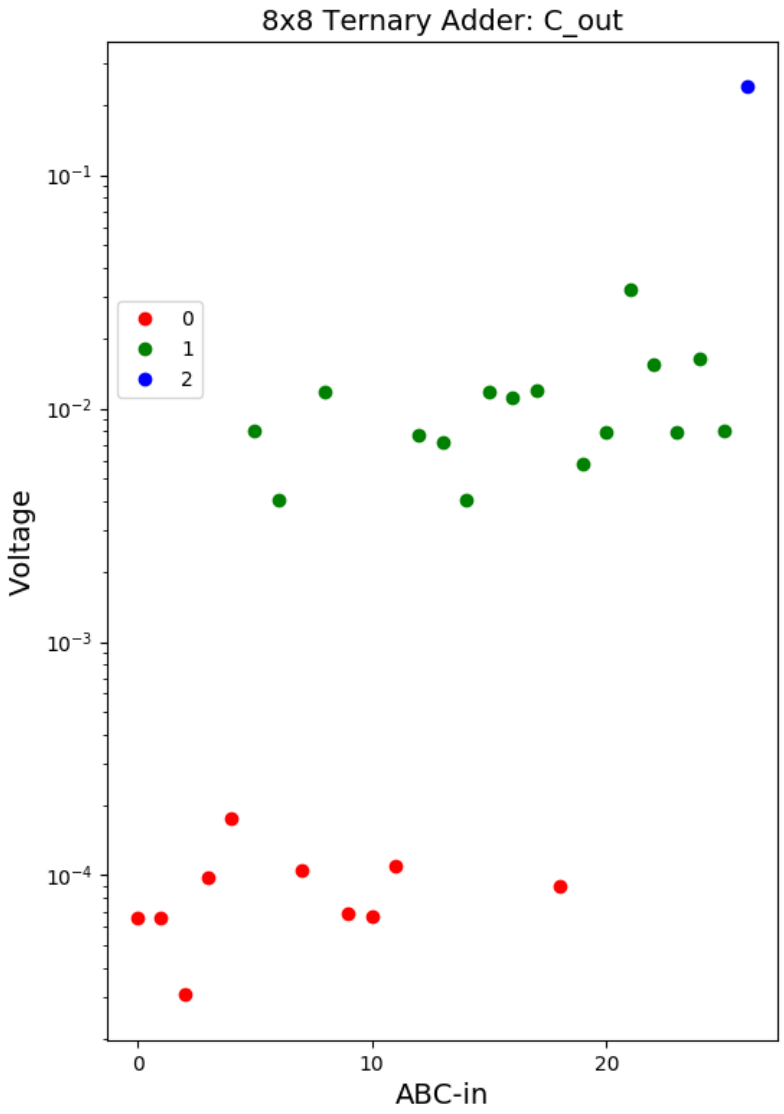

Figure 6.23: SPICE Results: 8x8 Enhanced Ternary Adder (012)- Carry

CHAPTER 7: CONCLUSION AND FUTURE WORK

Some goals for future development might include producing circuit designs with a tighter range of resistances, narrower voltage thresholds, and reducing the occurrence of strays or outliers. Further modeling of the internal state variable needs to be done prior to fabrication.

Additional memristive crossbar designs that implement a ternary n-bit adder should be a goal. Full integration of Modified Nodal Analysis to complement SPICE including input, output, wire resistances, and parasitic voltage loss needs to be addressed.

Refinement of the simulated annealing energy function will result in enhanced performance and more accurate results. The final process should continue to run to find a solution where all results are close to the midpoint of each voltage range.

LIST OF REFERENCES

[1] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[2] R. R. Schaller, "Moore's law: past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[3] S. Platt, 2018. [Online]. Available: https://www.micron.com/about/blog/2018/october/metamorphosis-of-an-industry-part-two-moores-law

[4] A. U. Hassen, D. Chakraborty, and S. K. Jha, "Free binary decision diagram-based synthesis of compact crossbars for in-memory computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 5, pp. 622–626, 2018.

[5] Z. Alamgir, K. Beckmann, N. Cady, A. Velasquez, and S. K. Jha, "Flow-based Computing on Nanoscale Crossbars: Design and Implementation of Full Adders," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 1870–1873.

[6] D. Chakraborty and S. K. Jha, "Design of compact memristive in-memory computing systems using model counting," in *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 1–4.

[7] W. Alexander, "The ternary computer," *Electronics and Power*, vol. 10, no. 2, pp. 36–39, 1964.

[8] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[9] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, p. 80, 2008.

[10] R. Stanley Williams, "How we found the missing memristor," in *Chaos, CNN, Memristors and Beyond: A Festschrift for Leon Chua With DVD-ROM, composed by Eleonora Bilotta*. World Scientific, 2013, pp. 483–489.

[11] D. W. Jones, "Fast ternary addition." [Online]. Available: http://homepage.divms.uiowa.edu/~jones/ternary/arith.shtml

[12] A. Dhande and V. Ingole, "Design and implementation of 2 bit ternary alu slice," in *Proc. Int. Conf. IEEE-Sci. Electron., Technol. Inf. Telecommun*, 2005, pp. 17–21.

[13] S. Lin, Y.-B. Kim, and F. Lombardi, "Cntfet-based design of ternary logic gates and arithmetic circuits," *IEEE transactions on nanotechnology*, vol. 10, no. 2, pp. 217–225, 2009.

[14] M. H. Moaiyeri, A. Rahi, F. Sharifi, and K. Navi, "Design and evaluation of energy-efficient carbon nanotube fet-based quaternary minimum and maximum circuits," *Journal of applied research and technology*, vol. 15, no. 3, pp. 233–241, 2017.

[15] Y. B. Kim, Y.-B. Kim, and F. Lombardi, "A novel design methodology to optimize the speed and power of the cntfet circuits," in *2009 52nd IEEE International Midwest Symposium on Circuits and Systems*. IEEE, 2009, pp. 1130–1133.

[16] J. Deng and H.-S. P. Wong, "A circuit-compatible spice model for enhancement mode carbon nanotube field effect transistors," in *2006 International Conference on Simulation of Semiconductor Processes and Devices*. IEEE, 2006, pp. 166–169.

[17] A. Velasquez and S. K. Jha, "Computation of boolean formula using sneak paths in crossbar computing," 2014.

[18] A. Velasquez. [Online]. Available: http://www.eecs.ucf.edu/~velasquez/

[19] L. O. Chua and S. M. Kang, "Memristive devices and systems," *Proceedings of the IEEE*, vol. 64, no. 2, pp. 209–223, 1976.

[20] B. Maundy, A. S. Elwakil, and C. Psychalinos, "Correlation between the theory of lissajous figures and the generation of pinched hysteresis loops in nonlinear circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2019.

[21] C. Leon, "Everything you wish to know about memristors but are afraid to ask," *Radioengineering*, vol. 24, no. 2, p. 319, 2015.

[22] D. Biolek, Z. Biolek, V. Biolkova, A. Ascoli, and R. Tetzlaff, "About vi pinched hysteresis of some non-memristive systems," *Mathematical Problems in Engineering*, vol. 2018, 2018.

[23] L. Chua, "Resistance switching memories are memristors," *Applied Physics A*, vol. 102, no. 4, pp. 765–783, 2011.

[24] T. Wang and J. Roychowdhury, "Well-posed models of memristive devices," *arXiv preprint arXiv:1605.04897*, 2016.

[25] D. Wouters and E. Linn, "Fundamentals of memristors." [Online]. Available: https://www.esa.int/gsp/ACT/doc/EVENTS/memristor_workshop/ACT-NAN-0415_2_wouters_memristors_DW_final.pdf

[26] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectronics Journal*, vol. 44, no. 2, pp. 176–183, 2013.

[27] D. Chakraborty and S. K. Jha, "Automated synthesis of compact crossbars for sneak-path based in-memory computing," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. IEEE, March 2017, pp. 770–775.

[28] D. Chakraborty, S. Raj, J. C. Gutierrez, T. Thomas, and S. K. Jha, "In-memory execution of compute kernels using flow-based memristive crossbar computing," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2017, pp. 1–6.

[29] D. Chakraborty, S. Raj, and S. K. Jha, "A compact 8-bit adder design using in-memory memristive computing: Towards solving the feynman grand prize challenge," in *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2017, pp. 67–72.

[30] A. G. Radwan and M. E. Fouda, *On the mathematical modeling of memristor, memcapacitor, and meminductor*. Springer, 2015, vol. 26.

[31] C. E. Merkel, N. Nagpal, S. Mandalapu, and D. Kudithipudi, "Reconfigurable n-level memristor memory design," in *The 2011 International Joint Conference on Neural Networks*. IEEE, 2011, pp. 3042–3048.

[32] C. Yakopcic, T. M. Taha, G. Subramanyam, R. E. Pino, and S. Rogers, "A memristor device model," *IEEE electron device letters*, vol. 32, no. 10, pp. 1436–1438, 2011.

[33] M. Laiho, E. Lehtonen, A. Russel, and P. Dudek, "Memristive synapses are becoming reality," *The Neuromorphic Engineer*, pp. 1–3, 2010.

[34] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Generalized memristive device spice model and its application in circuit design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 8, pp. 1201–1214, 2013.

[35] H. Vogt, M. Hendrix, and P. Nenzi, "Ngspice circuit simulator." [Online]. Available: http://ngspice.sourceforge.net/

[36] C. Yakopcic, "Spice model for memristive devices," 2011.

[37] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[38] E. Geltman. [Online]. Available: http://katrinaeg.com/simulated-annealing.html

[39] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, and F. Rossi, "Gnu scientific library," *Network Theory Ltd*, vol. 3, 2002.

[40] R. Cavada, A. Cimatti, C. A. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, and A. Tchaltsev, "Nusmv 2.6 user manual, tutorial," *CMU and FBK-irst*, 2010.

[41] Z. Biolek, D. Biolek, and V. Biolkova, "Differential equations of ideal memristors," *Radioengineering*, vol. 24, no. 2, pp. 369–377, 2015.

[42] C.-W. Ho, A. Ruehli, and P. Brennan, "The modified nodal approach to network analysis," *IEEE Transactions on circuits and systems*, vol. 22, no. 6, pp. 504–509, 1975.

[43] E. Cheever, E. Cheever, and S. College. [Online]. Available: https://lpsa.swarthmore.edu/Systems/Electrical/mna/MNA1.html

[44] G. Rizzoni, *Principles and applications of electrical engineering*. McGraw-Hill Science/Engineering/Math, 2005.

[45] Jaijeet, "The berkeley model and algorithm prototyping platform (mapp)." [Online]. Available: https://github.com/jaijeet/MAPP/wiki

[46] D. Amsallem and J. Roychowdhury, "Modspec: An open, flexible specification framework for multi-domain device modelling," in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2011, pp. 367–374.

[47] J. R. Pyrich, S. Raj, D. Chakraborty, and S. K. Jha, "Design of a ternary adder using flow-based computing," in *Internation Conference on Neuromorphic Systems (ICONS)*. Oak Ridge National Laboratory, 2019.

[48] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.