Technion – Israel Institute of Technology

Logic Design with Memristors

B.S.c Semesterial Project – Winter 2011-12

Guy Satat - 039934765 Nimrod Wald– 021631676

Supervised by Shahar Kvatinsky

Table of Content

TABLE OF CONTENT **1**

LIST OF FIGURES **4**

LIST OF TABLES **6**

1. ABSTRACT **7**

2. MEMRISTORS BACKGROUND **8**

2.1. PHYSICAL MODELING OF MEMRISTORS 9

2.1.1. Linear Memristor Model [2] *9*

2.1.2. Non-Linear Memristor Model *10*

2.2. TEAM MEMRISTOR MODEL [6] 11

2.2.1. Memristor Parameters Chosen for the Project *12*

2.3. MEMRISTORS AS CIRCUIT ELEMENTS 12

3. LOGIC FAMILIES BACKGROUND **13**

3.1. IMPLY 13

3.2. MAGIC 15

AND gate: 15

OR gate: 16

NOT gate: 16

3.2.1. MAGIC NOR and NAND addition *17*

NOT gate (without additional resistor): 17

NAND gate: 17

NOR gate: 18

3.3. HYBRID CMOS MEMRISTOR LOGIC (COMPUTATIONAL UNIT) 19

AND gate: 19

OR gate: 19

3.4. PROGRAMMABLE LOGIC MEMRISTOR ARRAY (PLA) 20

4. IMPLY LOGIC ANALYSIS **21**

4.1. SINGLE GATE ANALYSIS 21

4.2. IMPLEMENTING BOOLEAN FUNCTIONS USING IMPLY GATES 26

4.3. 1 BIT FULL ADDER 28

4.3.1. Driver Implementation *32*

4.4. 8 BIT FULL ADDER 32

4.4.1. Serial Implementation *33*

4.4.2. Parallel Implementation *34*

5. MAGIC LOGIC ANALYSIS 38

5.1. NON-CROSSBAR GATES ANALYSIS 38

5.2. CROSSBAR GATES ANALYSIS 40

6. HYBRID CMOS MEMRISTOR LOGIC ANALYSIS 44

6.1. SINGLE GATE ANALYSIS 44

6.2. 1 BIT FULL ADDER 47

6.3. 8 BIT FULL ADDER 53

7. PROGRAMMABLE LOGIC ARRAY (PLA) ANALYSIS 57

7.1. 1 BIT FULL ADDER 60

7.2. 8 BIT FULL ADDER 65

8. RESULTS ANALYSIS 67

8.1. COMPARISON BETWEEN LOGIC FAMILIES 70

8.1.1. Speed 70

8.1.2. Area 70

8.1.3. Controller Complexity 70

8.1.4. Power 71

8.1.5. VERSATILITY 71

8.2. COMPARISON BETWEEN MEMRISTOR TYPES 72

8.2.1. IMPLY 72

8.2.2. Hybrid CMOS 72

8.2.3. PLA 72

9. CONCLUSIONS 74

APPENDIX A – LIST OF TEAM MODEL PARAMETERS 75

APPENDIX B – COMPLETE ANALYSIS OF ADDITION TO MAGIC 80

NOT LOGIC GATE 80

NOR LOGIC GATE 81

NAND LOGIC GATE 83

COMPARING BETWEEN NOR AND NAND GATES 85

IMPLEMENTATION OF NOR AND NOT LOGIC GATES IN CROSS BAR FORM 86

NOISE MARGIN PROBLEM OF MAGIC NOT AND NOR GATES 87

APPENDIX C **88**

C1. IMPLY 1 Bit Full Adder Sequence 88

C2. IMPLY Parallel 8 Bit Full Adder Sequence 89

APPENDIX D **90**

D1. IMPLY 1 Bit Full Adder Voltage Control Files MATLAB Code 90

D2. IMPLY Parallel 8 Bit Full Adder Voltage Control Files MATLAB Code 92

BIBLIOGRAPHY **94**

List of Figures

Figure 1 - Relations between circuit elements 8
Figure 2 - Linear meristor model 9
Figure 3 - Simmons tunnel barrier model 10
Figure 4 - Memristor symbol 12
Figure 5 - IMPLY logic gate 13
Figure 6 - MAGIC AND gate 15
Figure 7 - MAGIC OR gate 16
Figure 8 - MAGIC NOT gate 16
Figure 9 - MAGIC NOT gate (without resistor) 17
Figure 10 - MAGIC NAND gate 17
Figure 11 -MAGIC NOR gate 18
Figure 12 - Hybrid CMOS AND gate 19
Figure 13 - Hybrid CMOS OR gate 19
Figure 14 - Example of a PLA 20
Figure 15 - IMPLY gate circuit 21
Figure 16 - IMPLY gate results for A=1 w/o threshold 22
Figure 17 - IMPLY gate results for A=1 with threshold 23
Figure 18 - IMPLY gate results for A=3 23
Figure 19 - IMPLY gate results for A=5 24
Figure 20 - IMPLY gate results for A=10 24
Figure 21 - IMPLY 1 Bit Full Adder Circuit 30
Figure 22 - MPLY 1 Bit Full Adder Simulation Results 31
Figure 23 -IMPLY Voltage Driver 32
Figure 24 - IMPLY 8 Bit Full Adder Circuit 33
Figure 25 - IMPLY Serial 8 Bit Full Adder Simulation 34
Figure 26 - IMPLY Parallel 8 Bit Full Adder Circuit 35
Figure 27 - IMPLY Parallel 8 Bit Full Adder Circuit Zoomed In 36
Figure 28 - MAGIC AND Gate 38
Figure 29 - 2 MAGIC AND Gates Concatenated 38
Figure 30 - MAGIC Non-crossbar Circuit Node 39
Figure 31 – MAGIC NOR Simulation Results 40
Figure 32 – MAGIC 2 NOR Gates Simulation 41
Figure 33 - MAGIC NOR, Inputs 00(left) 01(right), R_{OFF}=1MΩ, V_{NOR}=1V 42
Figure 34 - MAGIC NOR, Inputs 00(left) 01(right), V_{NOR}=1.5V 43
Figure 35 - Hybrid CMOS OR Gate Circuit 44
Figure 36 - Hybrid CMOS OR Gate Symbol 44
Figure 37 - Hybrid CMOS OR Gate Simulation Results 45

Figure 38 - Hybrid CMOS AND Gate Circuit 45
Figure 39 - Hybrid CMOS AND Gate Symbol 46
Figure 40 - Hybrid CMOS AND Gate Simulation Results 46
Figure 41 - CMOS Inverter Circuit 47
Figure 42 – DC Response for CMOS Inverter, Buffer, and 2 Series Buffers 48
Figure 43 - Hybrid CMOS XOR Gate 49
Figure 44 – Hybrid CMOS XOR Gate Simulation Results 49
Figure 45 - Hybrid CMOS 1 Bit Full Adder Circuit 51
Figure 46 – Hybrid CMOS 1 bit Full Adder Results for Various Voltages 51
Figure 47 – Hybrid CMOS 1 bit Full Adder Results for All Possible Inputs 53
Figure 48 - Hybrid CMOS 1 Bit Full Adder Cell 53
Figure 49 - Hybrid CMOS 8 Bit Full Adder Circuit 54
Figure 50 – Hybrid CMOS 8 bit Full Adder Results (Worst Propagation Time) 55
Figure 51 - NAND4 implemented in CMOS for PLA 57
Figure 52 – CMOS NAND4 DC Response for Various Voltages 58
Figure 53 - memristor based 3 input, 2 output PLA array 60
Figure 54 - memristor based 3 input, 2 output PLA array (Zoom in on top part) 61
Figure 55 - memristor based 3 input, 2 output PLA array (Zoom in on bottom part) 61
Figure 56 – Memristor Based PLA 1 Bit Full Adder Simulation Results 64
Figure 57 – memristor based PLA array 8 bit Ful Adder 65
Figure 58 - Memristor based PLA 8 bit Full Adder simulation result 66
Figure 59 - Memristor and Logic Family Matching 74
Figure 60 -Linear memristor without threshold response 75
Figure 61 – Linear memristor with threshold response 76
Figure 62 - Nonlinear memristor with threshold (a = 3) 77
Figure 63 - Nonlinear memristor with threshold (a = 5) 78
Figure 64 - Nonlinear memristor with threshold (a = 10) 79
Figure 65 - MAGIC NOT gate 80
Figure 66 - MAGIC NOR gate 81
Figure 67 - MAGIC NAND gate 83
Figure 68 - Cross bar implementation of gate 86

List of Tables

TABLE 1 - IMPLY TRUTH TABLE 13
TABLE 2 - IMPLY VOLTAGE SEQUENCE 22
TABLE 3 - IMPLY GATE CIRCUIT PARAMETERS 22
TABLE 4 - IMPLY GATE SIMULATION RESULTS 25
TABLE 5 - 1 BIT FULL ADDER TRUTH TABLE 28
TABLE 6 - IMPLY FULL ADDER IMPLEMENTATION METHODS COMPARISSON 37
TABLE 7 - HYBRID CMOS REQUIRED BUFFERS SUMMARY 55
TABLE 8 - PLA ARRAY PROGRAMMING (LEFT PART) 63
TABLE 9 - PLA ARRAY PROGRAMMING (RIGHT PART) 63
TABLE 10 - LINEAR MEMRISTOR WITHOUT THRESHOLD PARAMETERS 75
TABLE 11 - LINEAR MEMRISTOR WITH THRESHOLD PARAMETERS 76
TABLE 12 - NONLINEAR MEMRISTOR WITH THRESHOLD (A = 3) 77
TABLE 13 - NONLINEAR MEMRISTOR WITH THRESHOLD (A = 5) 78
TABLE 14 - NONLINEAR MEMRISTOR WITH THRESHOLD (A = 10) 79
TABLE 15 - MAGIC NOR AND NAND COMPARISON 85
TABLE 16 - IMPLY 1 BIT FULL ADDER SEQUENCE 88
TABLE 17 - IMPLY 8 BIT FULL ADDER SEQUENCE 89

1. Abstract

Memristors are a novel circuit element, predicted in 1971 by Leon Chua and produced in 2008 by HP labs. Their main attribute is a varying resistance with memory, hence the name memristor. So far the main applications explored for memristors where in the field of memory arrays and neuromorphic networks.

This project explores the utilization of logic with memristors.

Due to the fact that the field is in its early stages of research, various types of memristor models exist, and there is no wide consensus as to which one best describes the physical properties. In order for the research to relevant in any case, 5 types of memristors are considered.

3 logic families are considered: memristors as logic within memory, memristors as computation units, and memristors as a switch in PLA.

The comparison is done by implementing an 8-bit full adder, with each of the families and memristor types, and examining various parameters.

The conclusions include the preferred logic family for each type of memristor, and advantages and disadvantages of each logic family.

2. Memristors background

A Memristor is a recently discovered passive circuit element. It was first predicted by Leon Chua in 1971 [1], and was produced by HP labs in 2008 [2].
This element was predicted according to symmetry, to complement the preexisting 3 passive circuit elements (resistor, capacitor and inductor). The symmetry is visualized in following figure.

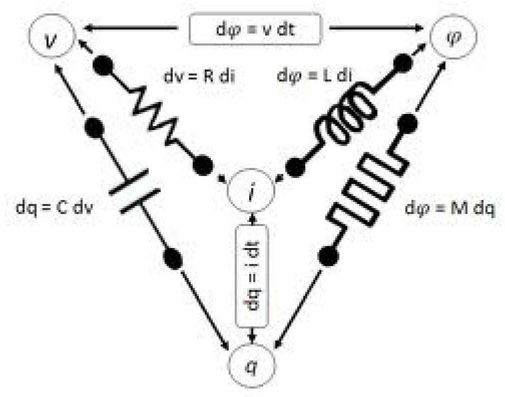


Figure 1 - Relations between circuit elements

As can be seen, the memristance M, which is the basic attribute of memristors, binds the flux with charge: $d\varphi = \mathrm{M}\ \mathrm{dq}$. Dividing by dq yields $M(q) = \frac{d\varphi}{dq}$, and by dt yields $V = M(q)\, I$.
The above equations show that memristors are in fact resistive elements, with varying resistance (memristance), this varying resistance is a function of the current (or charge) which passed through the element. This is a resistor with memory (memristor).

This new element is very attractive for replacement of memory cells. Since it is non-volatile, consumes no static power, and can be produced on minimal die area.

Memristors can be produced stacked above the CMOS layer, since they are formed between two adjacent metal layers.

Due to the mentioned memristor qualities, memristors can also be used as logic elements. Either as logic in memory (exceeding beyond the Von Neumann architecture), or as logic with CMOS (to reduce die area and parallel computing). This work will address the challenge of implementing logic with memristors.

2.1. Physical Modeling of Memristors

Research and production of memristors is in its early stages, and so there is no widely accepted physical model to describe memristance. For this reason there are a lot of mathematical models, 2 of the suggested physical models are presented:

2.1.1. Linear Memristor Model [2]

This model was developed by HP labs upon production of the first memristor. This is a very simplistic model, assuming 1 state variable. The model assumes that charged ions are free to move under the influence of large field, this movement changes the conductance of the media. This is modeled by an element which is composed of 2 series resistors (R_{ON}, R_{OFF}, where $R_{ON} < R_{OFF}$). The state variable W represents the relative part of each resistor.

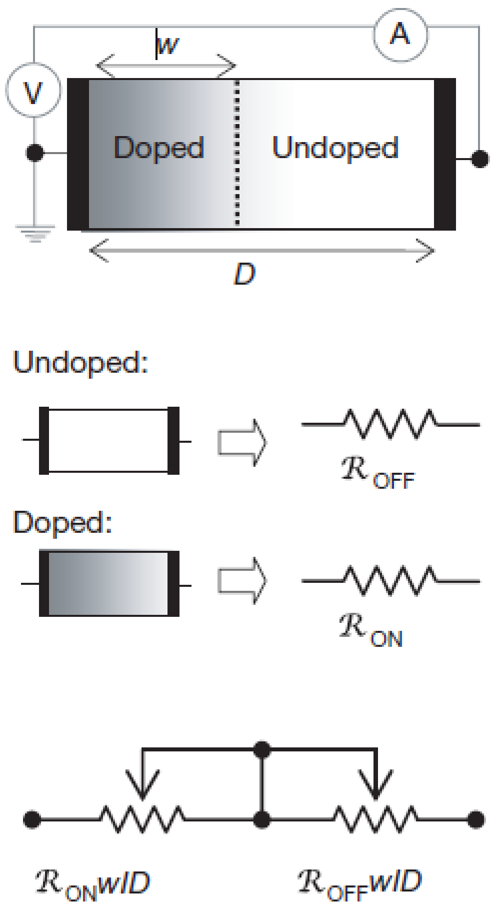


Figure 2 - Linear meristor model

The equations which descibe this physical model are:

$$V(t) = \underbrace{\left(R_{ON}\frac{w(t)}{D} + R_{OFF}\left(1 - \frac{w(t)}{D}\right)\right)}_{M(q)} i(t)$$

$w(t) = \mu_V \frac{R_{ON}}{D}\ i(t)$ (where μ_V is the ion mobility)

2.1.2. Non-Linear Memristor Model

Few nonlinear models exist. HP's Simmons tunnel barrier model [3] is presented in detail:

The linear model turned out to be insufficient in describing HP's memristor. The new model which was suggested added a tunneling effect based on Simmons tunneling [4].

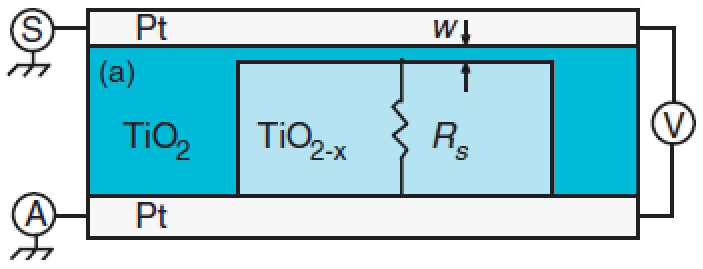


Figure 3 - Simmons tunnel barrier model

The new model suggests the same relation between $i(t), v(t), M(q)$, where the state variable W represents the width of a tunneling barrier, which is changed by current. The change in W is different between opening and closing the memristor. The equations which describe the state variable W are:

$$\frac{dw}{dt}\Big|_{off\ switching, i>0} = f_{off} sinh\left(\frac{i}{i_{off}}\right) e^{-e^{\frac{w-a_{off}}{w_C} - \frac{|i|}{b}} - \frac{w}{w_C}}$$

$$\frac{dw}{dt}\Big|_{on\ switching, i<0} = f_{on} sinh\left(\frac{i}{i_{on}}\right) e^{-e^{\frac{w-a_{on}}{w_C} - \frac{|i|}{b}} - \frac{w}{w_C}}$$

Where all the unmentioned parameters are fitting parameters of the model.

Another example of a nonlinear model is the nonlinear ion-drift model [5].

2.2. TEAM Memristor Model [6]

TEAM (ThrEshold Adaptive Memristor model) is a mathematical model designed to overcome the complexity and variety of the models. In essence it is a sum of terms expansion of a given model. The model separates between the current response and state variable $\frac{dx}{dt} = f(x,i) \approx g(i)f(x)$ (the state variable x replaces the previous w). The model equations are:

$$\frac{dx}{dt} = \begin{cases} K_{off}\left(\frac{i(t)}{i_{off}} - 1\right)^{\alpha_{off}} \cdot f(x) & 0 < i_{off} < i \\ K_{on}\left(\frac{i(t)}{i_{on}} - 1\right)^{\alpha_{on}} \cdot f(x) & \mathrm{i} < i_{on} < 0 \\ 0 & otherwise \end{cases}$$

Measurements show exponential dependence between the state variable and the memristance which can be written as $M(x) = R_{on} \cdot \exp\left(\frac{\lambda}{x_{off} - x_{on}} \cdot (x - x_{on})\right)$. The model also enables the use of linear dependence $M(x) = R_{on} + \frac{R_{off} - R_{on}}{x_{off} - x_{on}}(x - x_{on})$.

$f(x)$ is a window function which describes the response of the memristor to the current state variable, and can be fitted from measurements or deduced from physical models.

This project uses a window function which is deduced from Simmons Tunnel Barrier model: $f_{off} = e^{-e^{\frac{x - a_{off}}{w_c}}}$, $f_{on} = e^{-e^{\frac{x - a_{on}}{w_c}}}$.

When using this window function, the list of needed parameters are:

- K_{off}, K_{on} – Controls magnitude of state change.
- $\alpha_{off}, \alpha_{on}$ – Controls the linearity of the model.
- i_{off}, i_{on} – Determine the current threshold of the memristor.
- R_{on}, λ – Determine the edge resistances of the memristor.
- $x_{on}, x_{off}, a_{on}, a_{off}, w_c$ – Fitting parameters.

2.2.1. Memristor Parameters Chosen for the Project

For the purpose of comparing between different memristor types, 5 classes of memristors were chosen, and parameterized according to the TEAM model.

The parameters were chosen to achieve the following types of memristors:

- Linear memristor without current threshold.
- Linear memristor with current threshold.
- 3 types of nonlinear memristors with current threshold
    - $\alpha = 3, 5, 10$.

Additional considerations in choosing parameters were reasonable working voltage levels, full swing of the memristance in the given voltage, and reasonable switch time (All the models where chosen to be symmetric between ON and OFF switching).

The full list of parameters and response graphs is given in appendix A.

2.3. Memristors as Circuit Elements

The symbol of memristors is:

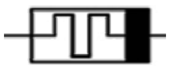


Figure 4 - Memristor symbol

When current flows into the thick black line (right side in the above picture) the memristance decreases. When current flows out of the thick black line the memristance increases.

Most models include a current threshold, below which there is no change in memristance (or a small change). This current threshold can be substituted by a voltage threshold (memristance dependent) when the memristor is a part of a circuit.

3. Logic families background

3 separate logic families can be distinguished:

1. Logic in memory, where the logic operations are done within memory cells. In this family the logic values are stored as resistance. Gates in this family include IMPLY and MAGIC.
2. Hybrid CMOS Memristor Logic. In this family memristors are used as computational units only, it requires both CMOS gates along with memristor based gates. Logic values are stored as voltages.
3. Programmable Logic Memristor Array (PLA). This family is in fact a regular PLA, where the connections between horizontal and vertical wires are done with memristors.

3.1. IMPLY

Logic implication (IMPLY) is a fundamental 2 input Boolean function, along with "False" it constitutes a complete set of operations.

P	Q	P→Q
0	0	1
0	1	1
1	0	0
1	1	1

Table 1 - IMPLY truth table

IMPLY function is native for implementation with memristors in a cross bar array, as demonstrated in [7].

Implementation of IMPLY gate with memristors is shown in the figure below:

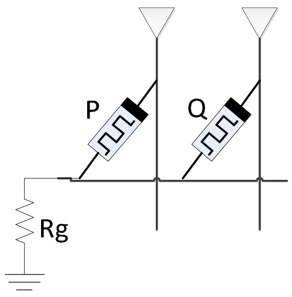


Figure 5 - IMPLY logic gate

In this form the logic values are stored as memristance of the memristors P and Q, where ‘0’ is defined as R_{OFF}, and ‘1’ is defined as R_{ON}.
It should be noted that the input memristors at the beginning of the operation are P and Q, and the output memristor at the end of the operation is Q (the input value of Q is destroyed).

The value of the resistor R_g is chosen to meet $R_{ON} < R_g < R_{OFF}$.
The operation of the gate is done by applying voltages V_{set} and V_{cond} to Q and P respectively, where $V_{set} > V_{cond}$. Since it is assumed that memristors have current threshold, a critical voltage V_c can be defined, so that voltage lower than it will not change the state of the memristor when applied. In order for the IMPLY operation to work, the following conditions should be met: $V_{set} > V_c > V_{cond}$, $V_{set} - V_{cond} < V_c$.

The IMPLY gate operation:
Since the current in above circuit can only flow from top to bottom (due to the applied voltages), the memristance of P and Q can only decrease, and there for if Q starts in ‘1’ (low resistance), it will remain unchanged.
If P=’0’ and Q=’0’, since $R_{OFF} > R_g$, the voltage in central node is approximately zero, and the applied voltages fall only on the memristors, and Q is switched to ‘1’.
If P=’1’ and Q=’0’, the central node voltage is approximately V_{cond}, and voltage applied on Q is $V_{set} - V_{cond}$, which is not sufficient to switch Q.

It should be noted that the case of P=0’ and Q=’0’ determines the time of the operation, it also determines the state change of P.
The case of P=’1’ and Q=’0’ determines the robustness of the gate, since Q drifts from ‘0’ to ‘1’, this phenomenon is called “state drift”.

A suggested design methodology for selecting the specific values is given in [8].

3.2. MAGIC

This logic family has gates build with separate memristor for each input, and a separate memristor for output. Like IMPLY, logic values are stored as resistance. This family contains AND, OR, NOT logic operations. Each logic gate is operated applying a single voltage which is divided between the memristors according to their logic values, and so switches them or not. Due to this fact the output memristor must be initialized prior to the operation.

AND gate:

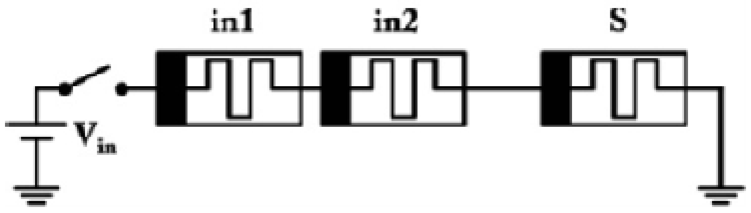


Figure 6 - MAGIC AND gate

Prior to performing the AND operation, the output memristor S, must be initialized to ‘0’ (R_{OFF}).

In case both inputs are ‘1’, a switch is required so that $i_{threshold} < \frac{V_{in}}{2R_{ON}+R_{OFF}}$. In case that 1 of the inputs is ‘0’ and the other is ‘1’, no switch is needed so that $i_{threshold} < \frac{V_{in}}{R_{ON}+2R_{OFF}}$.

Combining the two equations yields: $i_{th}(2R_{ON} + R_{OFF}) < V_{in} < i_{th}(R_{ON} + 2R_{OFF})$.

The gate can be generalized for multiple inputs (χ), in that case the condition becomes: $i_{th}(\chi R_{ON} + R_{OFF}) < V_{in} < i_{th}((\chi - 1)R_{ON} + 2R_{OFF})$.

OR gate:

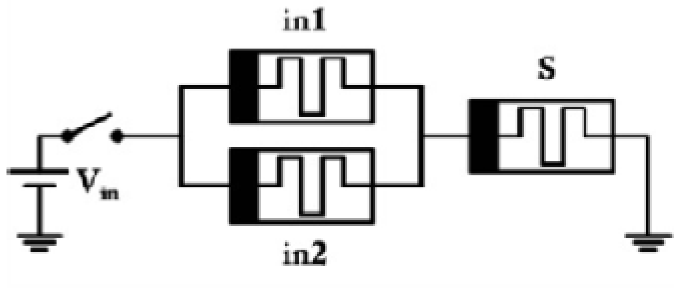


Figure 7 - MAGIC OR gate

This gate is initialized and analyzed like an AND gate. The resulting equation is:

$$i_{th}\left(R_{ON}||\left(\frac{R_{OFF}}{\chi-1}\right)+R_{OFF}\right)<V_{in}<i_{th}\left(1+\frac{1}{\chi}\right)R_{OFF}$$

NOT gate:

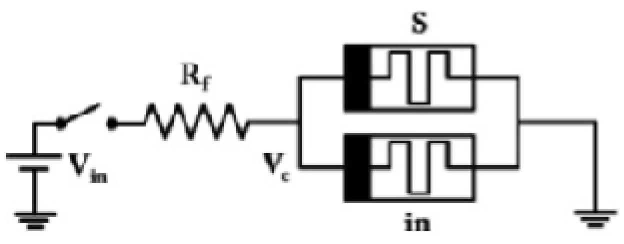


Figure 8 - MAGIC NOT gate

This gate consists of an input and output memristor, and a regular resistor used for voltage division. The output memristor is initialized to ‘0’ prior to calculation. The constrains on the input voltage are: $\left(\frac{1}{2}+\frac{R_f}{R_{off}}\right)i_{th}<V_{in}<\left(1+\frac{R_f}{R_{on}}\right)i_{th}$.

3.2.1. MAGIC NOR and NAND addition

During the work on the project we suggested additional gates built in similar architecture. These gates are analyzed in a similar way. A complete analysis of these gates is brought in Appendix B, This section contains a brief introduction.

NOT gate (without additional resistor):

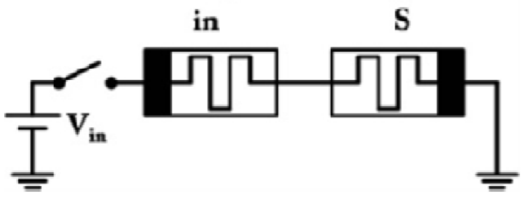


Figure 9 - MAGIC NOT gate (without resistor)

In this gate, the output memristor S, must be initialized to '1' prior to the operation. Analyzing the required voltage is done in a similar way, and results in:

$2\, i_{th} R_{on} < V_{in} < i_{th}(R_{off} + R_{on})$.

NAND gate:

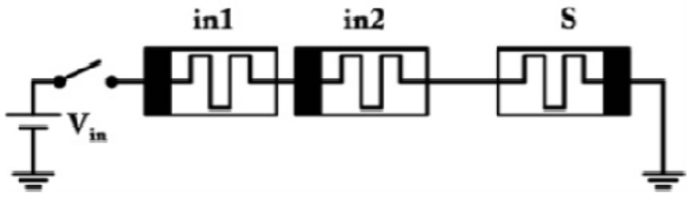


Figure 10 - MAGIC NAND gate

The output memristor of the NAND gate is initialized to '1' prior to calculation. The condition on the input voltage is: $(1+\chi) i_{th} R_{on} < V_{in} < i_{th}\left((\chi - 1) R_{off} + 2R_{on}\right)$

NOR gate:

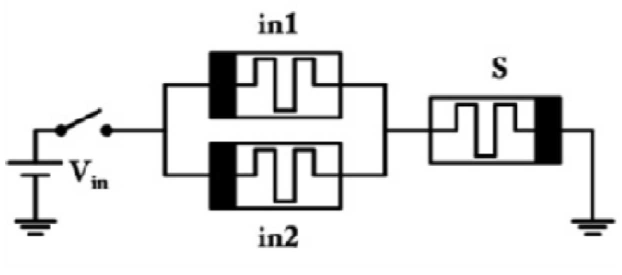


Figure 11 - MAGIC NOR gate

The output memristor of the NOR gate is initialized to ‘1’ prior to calculation. The voltage constraints of this gate are:

$$i_{off}\left(\frac{R_{off}}{\chi-1}||R_{on}+R_{on}\right)<V_{in}<i_{off}\left(\frac{R_{off}}{\chi}+R_{on}\right)$$

It should be noted that the new NOT and NOR gates can be implemented in a cross bar form which is native for memristors, the other gates require a complex wiring between memristors. Yet the NOR gate suffers from significant state drift and noise margin which is analyzed in Appendix B.

3.3. Hybrid CMOS Memristor logic (computational unit)

The Hybrid CMOS memristor logic is a family in which logic values are stored as voltages, and memristors are used as computation units only. This family consists only of AND and OR logic gates. Since AND and OR, are not a complete set of functions, additional gates (such as NOT) are needed to be implemented. In this project CMOS NOT gate is used.

AND gate:

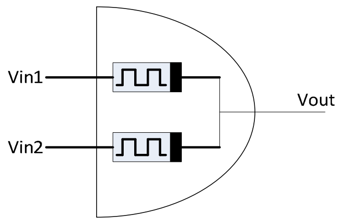


Figure 12 - Hybrid CMOS AND gate

When voltages in both inputs are identical, there is no voltage upon the memristors, and the output voltage equals to the input voltage.
When one of the inputs is high and the other is low, current flows from the high voltage node to the lower voltage node. According to the polarity of the memristors, the current increases the memristance of the memristor connected to the high voltage node, and decreases the memristance of the other one. Due to the new voltage divider the results is $V_{out} = \frac{R_{on}}{R_{on}+R_{off}} V_{high} \approx 0$.

OR gate:

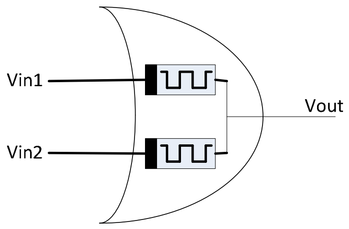


Figure 13 - Hybrid CMOS OR gate

The OR gate operates in same way as the AND gate. The difference is in the polarities of the memristors. This change causes the high memristance to be located near the low voltage, and vice versa. This changes the voltage divider, and so:

$$V_{out} = \frac{R_{off}}{R_{on}+R_{off}} V_{high} \approx V_{high}$$

3.4. Programmable Logic Memristor Array (PLA)

A programmable logic memristor array is a standard PLA, where memristors are used as connections between the wires [9].

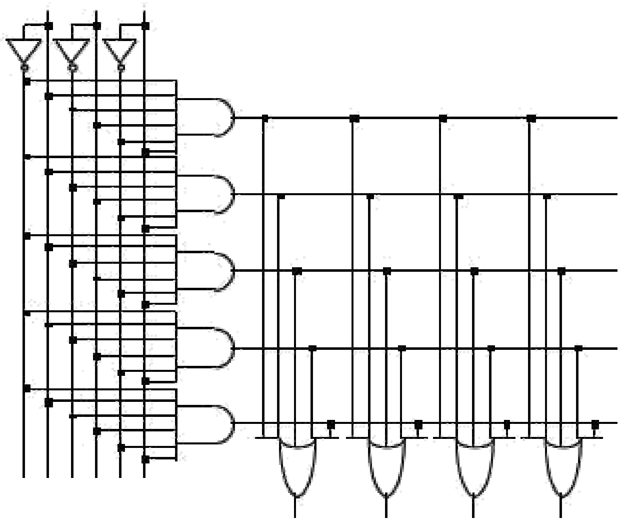


Figure 14 - Example of a PLA

The logic gates used in the PLA are standard CMOS or other gates.

Memristors act as connections between the wires, the low resistance R_{ON} is considered short circuit, and the high resistance R_{OFF} is considered disconnected.

Programming the PLA to realize any logic function is done by setting or resetting each of the memristors.

Performing logic calculations is done in sub threshold currents / voltages (in order not to change the programming of the memristors).

4. IMPLY Logic Analysis

This chapter will discuss the results of IMPLY logic gate analysis. In general, all results shown are for the use of TEAM model and the memristor parameters given in Appendix A.

Early work on IMPLY gates was done using the Linear and Nonlinear models discussed in the background chapter. These models did not yield the expected results due to the models themselves or their implementation in Verilog A.

4.1. Single gate analysis

A single IMPLY gate was created and tested for each of the memristor types. The circuit that was built is:

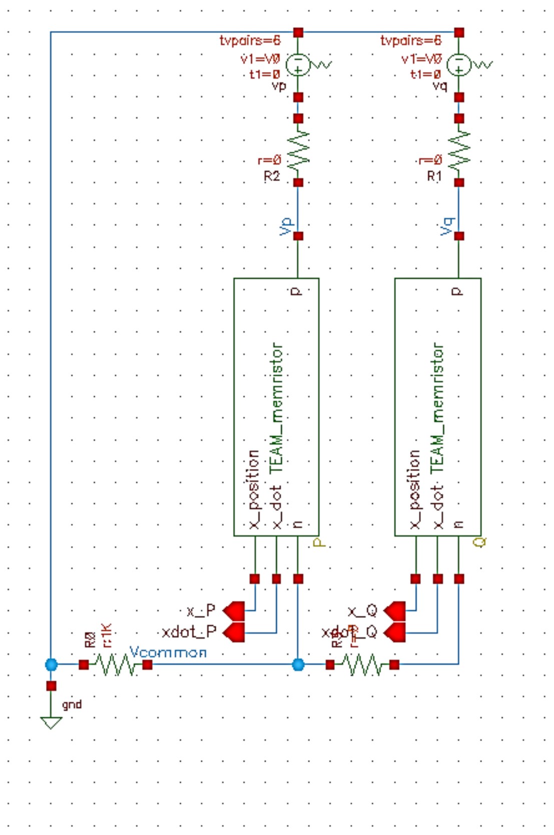


Figure 15 - IMPLY gate circuit

The left memristor is P and the right one is Q, each memristor cell has two additional outputs (x_position and x_dot) which are not used. Each node of a memristor is connected via a resistor, when a resistor is not needed the resistance is set to 0, this is done to prevent connection of two output nodes.

The voltage sources used are Vpwl and the resistor Rg is of 2KΩ.

The voltage sequence used for all simulations is as follows:

Time [ns]	**1-3**	**4-5**	**6-8**	**9-11**	**12-13**	**14-16**	**17-19**	**20-21**	**22-24**	**25-27**	**28-29**
Vp [V]	Vrst	Vcond	Vrst	0	Vcond	Vset	0	Vcond	Vset	0	Vcond
Vq [V]	Vrst	Vset	0	Vset	Vset	0	Vrst	Vset	0	Vset	Vset

Table 2 - IMPLY voltage sequence

This allows us to cover all possible inputs to the gate.

Vcond and Vset were chosen for each memristor according to [8] and minor adjustments were made to receive full switching. The parameters used are:

α	**1**	**1**	**3**	**5**	**10**
Current Threshold [A]	**0.1p**	**20u**	**5u**	**5u**	**10u**
Vset [V]	0.25	2.5	1.6	1.6	2.7
Vcond [V]	0.235	1.6	1.2	1.2	1.7
Vreset [V]	-5	-5	-5	-5	-5
Rg [Ω]	2K				
Tr\Tf of step [sec]	0.1n				

Table 3 - IMPLY gate circuit parameters

Simulation results for all memristors are shown in the next figures.

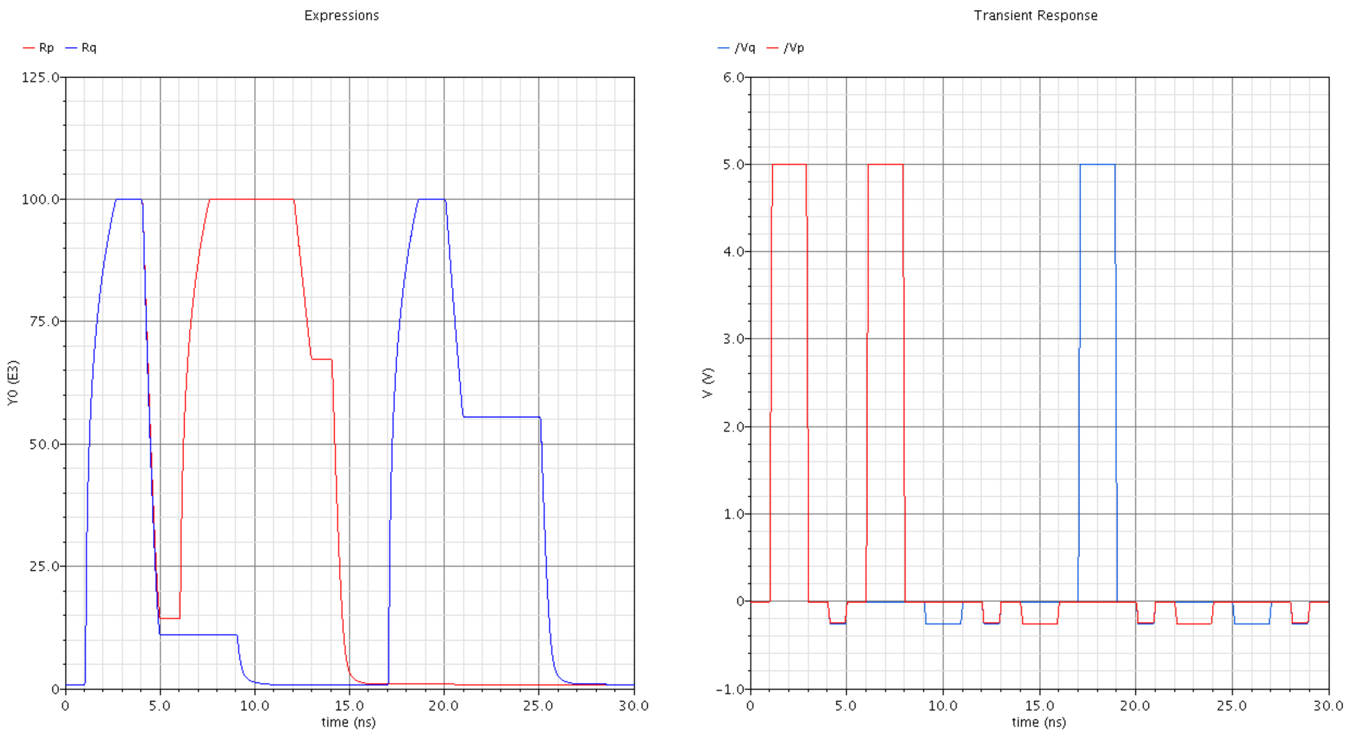


Figure 16 - IMPLY gate results for α=1 w/o threshold

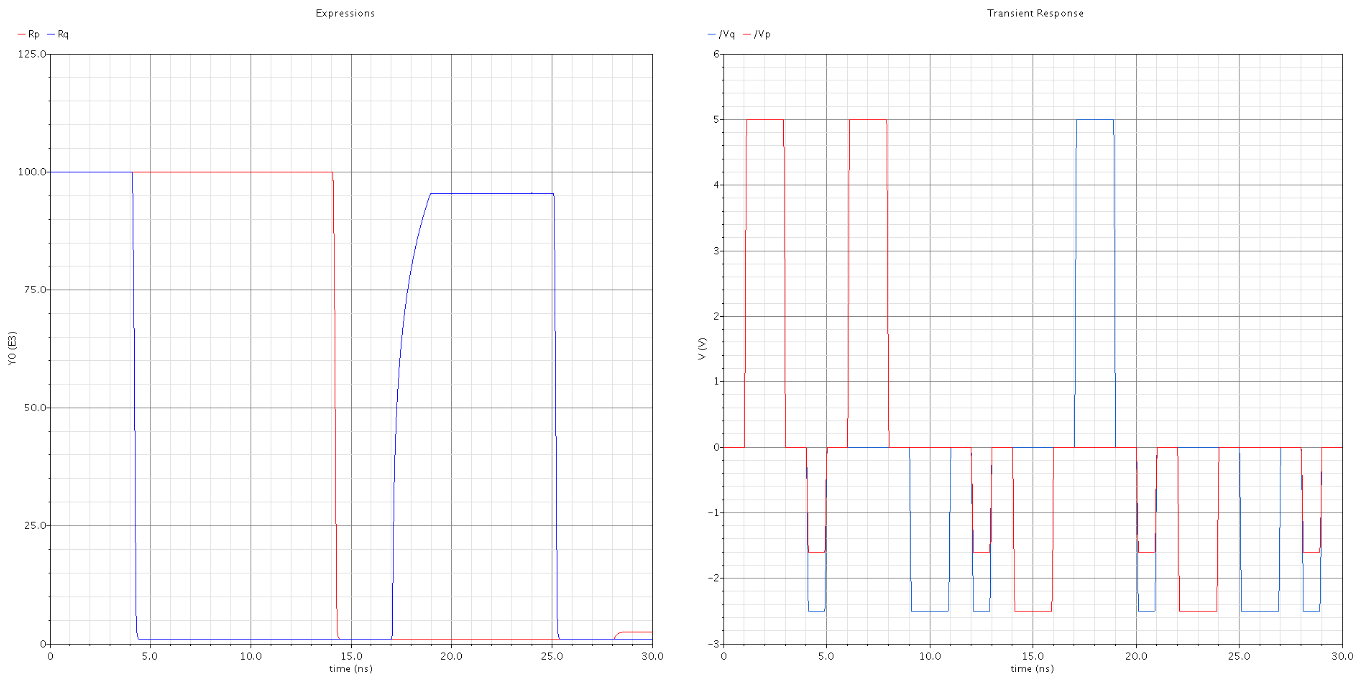


Figure 17 - IMPLY gate results for α=1 with threshold

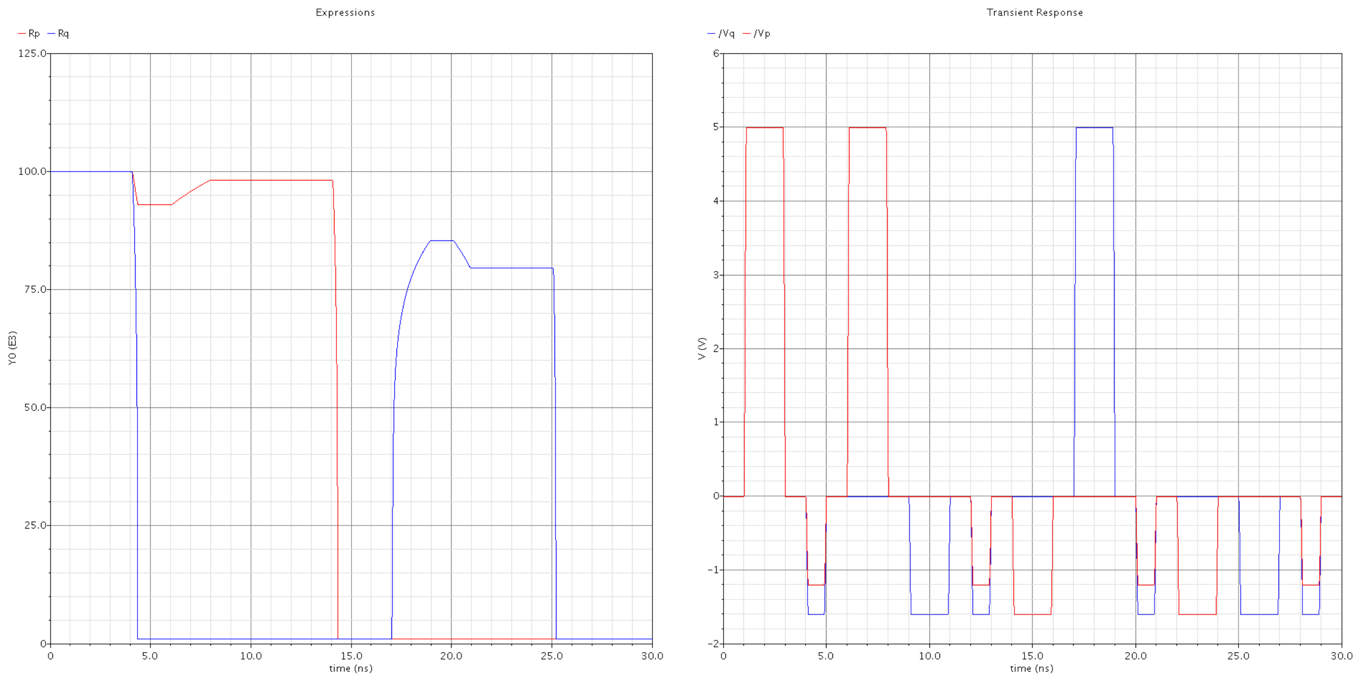


Figure 18 - IMPLY gate results for α=3

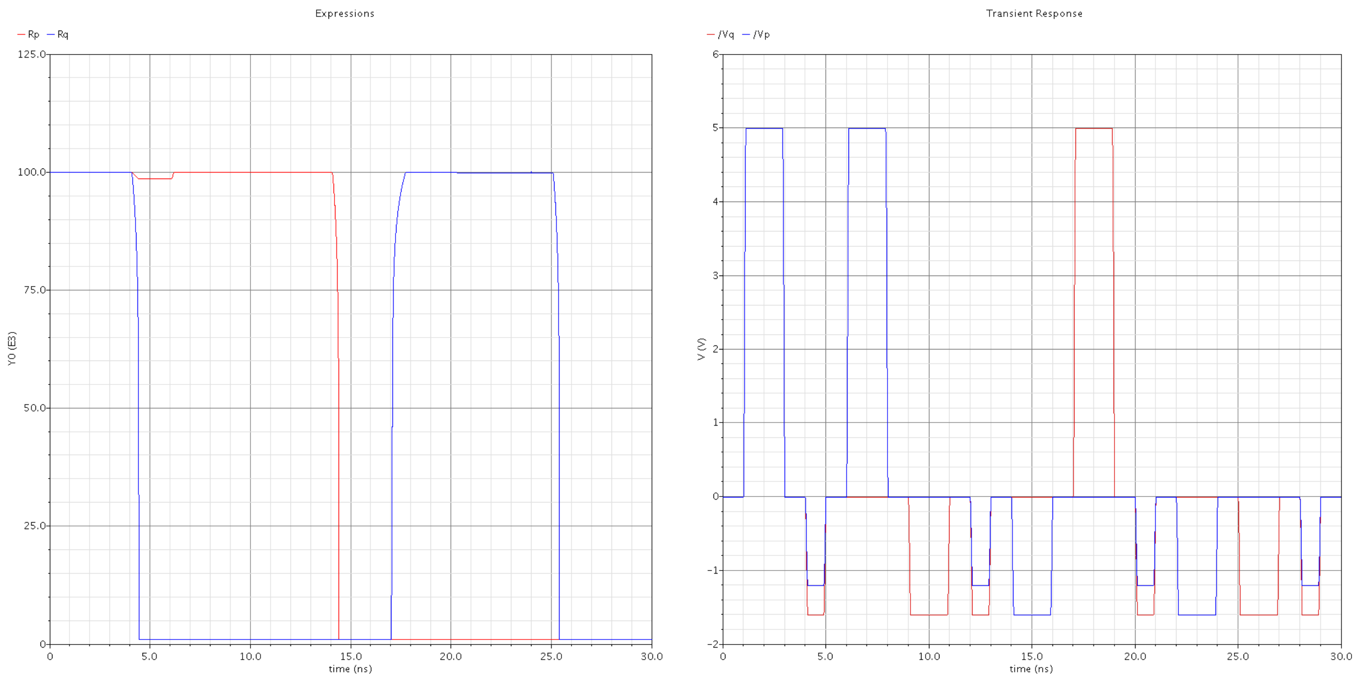


Figure 19 - IMPLY gate results for α=5

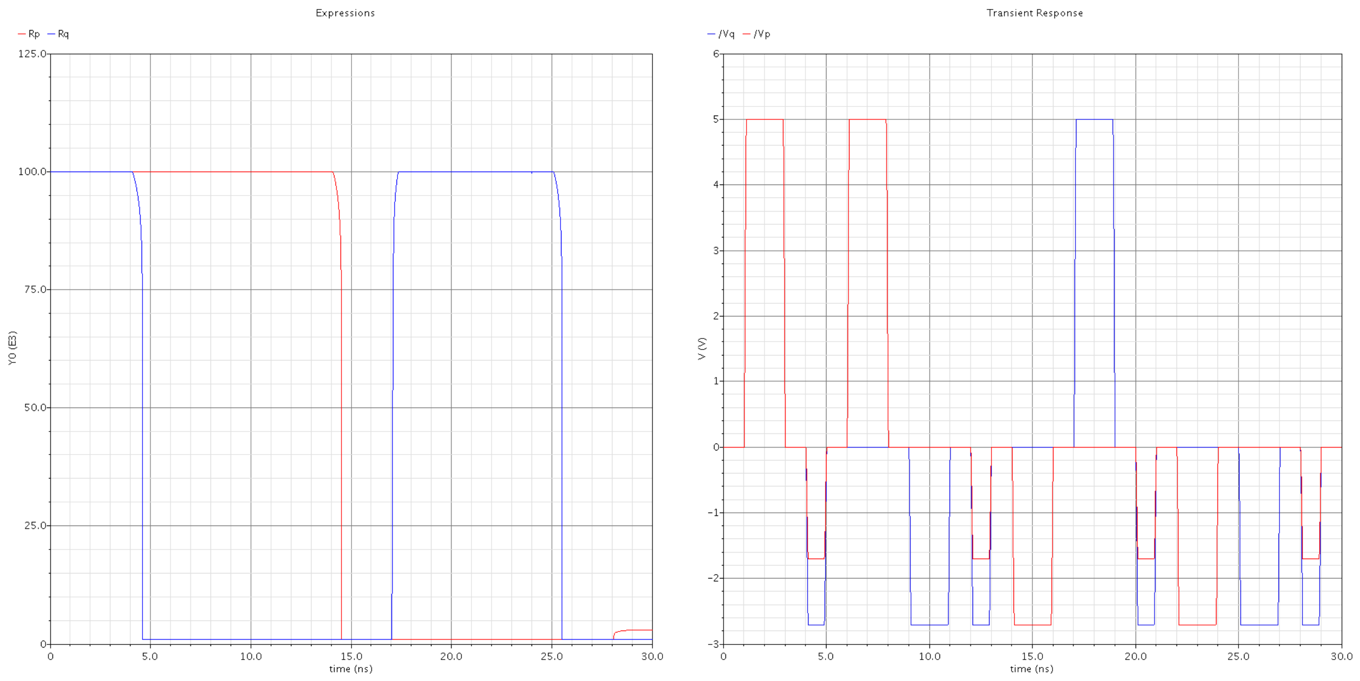


Figure 20 - IMPLY gate results for α=10

A summary of the results is given below:

α	**1**	**1**	**3**	**5**	**10**
Current Threshold [A]	**0.1p**	**20u**	**5u**	**5u**	**10u**
T_{IMPLY} [sec]	1n	0.4371n	0.343n	0.4216n	0.5805n
Treset [sec]	1.65n	2n	2n	0.7124n	0.331n
Value after reset [Ω]	100k	95K	85K	100K	100K
State drift from (p,q)=(0,0)	-86% in P +10% in Q	No	~-7.5% in P	~-2% in P	No
State drift from (p,q)= (0,1)	-33% in P	No	No	No	No
State drift from (p,q)= (1,0)	-44% in Q	No	~-2.5% in Q	~-0.1% in Q	No
State drift from (p,q)= (1,1)	No	~1.5% in P	No	No	~2% in P
Trise from (p,q)=(0,0) For IMPLY [sec]	0.4576n	0.1992n	0.31514n	0.41585n	0.5805n
Tfall for reset [sec]	0.2387n	0.2578n	0.118n	0.063n	0.05527n

Table 4 - IMPLY gate simulation results

The results were extracted under the following considerations:

- Trise is measured for a change from R=100KΩ to R=50KΩ.
- Tfall is measured for a change from R=1KΩ to R=50KΩ.
- Treset was limited by 2ns in order to prevent extremely long reset times. The result is that for α=3, 1 with threshold the reset value is lower than Roff.

Results analysis:

- The state drift for the linear memristor without a threshold is extremely significant and expected to be very problematic for IMPLY logic implementation.
- State drift in the case of (p,q)=(0,0) is only in P as expected and is increased for low current thresholds, after the switch of Q the current rushes from Vset to Vcond and increases P's resistance.
- State drift in the case of (p,q)=(1,0) is only in Q as expected and is increased for low current thresholds.

- State drift in the case of (p,q)=(1,1) is only in P and is decreased for low current thresholds. The reason for this state drift is the large difference $V_{SET} - V_{COND}$ that exists for memristors with high current threshold.
- The timing results are affected by the linearity, current threshold, and applied voltages. Since these parameters vary between the memristors we were not able to find any distinct trend.

The timing results were used as base time units when designing the full adders voltage sequences.

4.2. Implementing Boolean functions using IMPLY gates

In order to implement a 1 bit full adder using IMPLY gates, it was required to understand how to perform various Boolean functions with these gates.
Truth tables that can be realized using imply:

1) $P \rightarrow Q => \; Q = \bar{P} + Q$

P	Q	$\bar{P}$	$\bar{P} + Q$	$P \rightarrow Q$
0	0	1	1	1
0	1	1	1	1
1	0	0	0	0
1	1	0	1	1

2) $'0' \rightarrow Q => Q = '1'$

P	Q	$P \rightarrow Q$
0	0	1
0	1	1

3) $'1' \rightarrow Q => Q = Q$

P	Q	$P \rightarrow Q$
1	0	0
1	1	1

4) $P \rightarrow' 0' => Q = \bar{P}$

P	Q	$\bar{P}$	$P \rightarrow Q$
0	0	1	1
1	0	0	0

Using these it can be seen that:

1. $A \oplus B = (A \rightarrow B) \rightarrow \big((B \rightarrow A) \rightarrow' 0'\big)$

A	B	$A \rightarrow B$	$B \rightarrow A$	$(B \rightarrow A) \rightarrow' 0'$	$(A \rightarrow B) \rightarrow \big((B \rightarrow A) \rightarrow' 0'\big)$	$A \oplus B$
0	0	1	1	0	0	0
0	1	1	0	1	1	1
1	0	0	1	0	1	1
1	1	1	1	0	0	0

2. $AB = \big(A \rightarrow (B \rightarrow' 0')\big) \rightarrow' 0'$

A	B	$B \rightarrow' 0'$	$A \rightarrow (B \rightarrow' 0')$	$\big(A \rightarrow (B \rightarrow' 0')\big) \rightarrow' 0'$	AB
0	0	1	1	0	0
0	1	0	1	0	0
1	0	1	1	0	0
1	1	0	0	1	1

3. $A + B = (A \rightarrow' 0') \rightarrow B$

A	B	$A \rightarrow' 0'$	$(A \rightarrow' 0') \rightarrow B$	$A + B$
0	0	1	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

4. $\bar{A} = A \rightarrow '0'$

A	$A \rightarrow' 0'$	$\bar{A}$
0	1	1
1	0	0

An option to duplicate a given input memristor to another memristor (Duplicate): Duplicating the value of memristor A to memristor B involves a work memristor C, we use function number (4) $\big((P \rightarrow' 0') \rightarrow '0'\big)$. The duplication is done by the following sequence:

$$False(B)$$

$$False(C)$$

$$A \rightarrow C$$

$$C \rightarrow B$$

In order to calculate XOR the values of A and B are needed at least twice, So it is assumed that the value of P remains intact after IMPLY, this is a bad assumption, and its effect will be checked in simulation. To avoid this assumption, it is needed to have inputs duplicated in some other way not using IMPLY.

To calculate XOR two work memristor and results memristor are needed (the input memristor remain intact). The sequence to calculate XOR is:

Duplicate A to M1: $False(M1)\ ,\ False(S)\ ,\ A \to S\ ,\ S \to M1$

Duplicate B to M2: $False(M2)\ ,\ False(S)\ ,\ B \to S\ ,\ S \to M2$

B→A (saved to M1): $B \to M1$

(B→A)→'0' (saved to S): $False(S)\ ,\ M1 \to S$

A→B (saved to M2): $A \to M2$

Final calculation: $M2 \to S$

Thus, to calculate XOR two work memristors are needed, and a total of 13 calculation steps. Once again, this method assumes that the values of the input memristor are kept, and it is assumed that IMPLY operation changes only q, and p remains intact.

4.3. 1 Bit Full Adder

A 1 bit full adder truth table is:

in1	in2	C_in	S	C_out
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Table 5 - 1 Bit Full Adder Truth Table

It can be seen that:

$$S = (A \oplus B) \oplus C \quad , \ C_{out} = \big(AB + C_{in}(A \oplus B)\big)$$

To calculate $S = (A\ XOR\ B)\ XOR\ C_{in}$, calculation is divided to two XOR operations, keeping the intermediate value for later use in Cout calculation. Two working memristors are needed and a total of 26 steps (these steps include 4 steps to

duplicate A XOR B which is needed for the calculation of Cout. Due to this Cout is used as temporarily working memristor during calculation of S).
These steps are sequential, but many operations can be done simultaneously (like false). In order to perform simultaneous IMPLY operations it is needed to separate between the memristors in the cross bar.

To calculate $C_{out} = \left(AB + C_{in}(A \oplus B)\right)$, the intermediate value of A XOR B from calculating S is used.

$$\begin{aligned} C_{out} &= AB + C_{in}(A \oplus B) \\ &= \left(\left(A \to (B \to' 0')\right) \to' 0'\right) + \left(\left(C_{in} \to \left((A \otimes B) \to' 0'\right)\right) \to' 0'\right) \\ &= \left(\left(\left(A \to (B \to' 0')\right) \to' 0'\right) \to' 0'\right) \to \left(\left(C_{in} \to \left((A \otimes B) \to' 0'\right)\right) \to' 0'\right) \\ &= \left(A \to (B \to' 0')\right) \to \left(\left(C_{in} \to \left((A \otimes B) \to' 0'\right)\right) \to' 0'\right) \end{aligned}$$

Numerous computation sequences can be thought of, the considerations that were taken while forming the sequence used were:

- The number of calculation steps can be reduced at the expense of adding more memristors (and vice versa) for saving intermediate calculation results. Two options are shown later that demonstrate this trade-off.
- The calculation of C_{out} utilizes some of the intermediate calculations done to get S and the other way around. This leads to potential reduction of calculation steps if interleaving the two calculations.
- The input memristor can be used as temporary working memristors after the value initially stored in them is used. It was decided that A and B will be preserved (due to their unknown origin) while C_{in} may be run-over to reduce calculation steps and memristors.

The chosen sequence is brought in detail in Appendix C1.

The circuit used for the 1 bit full adder is:

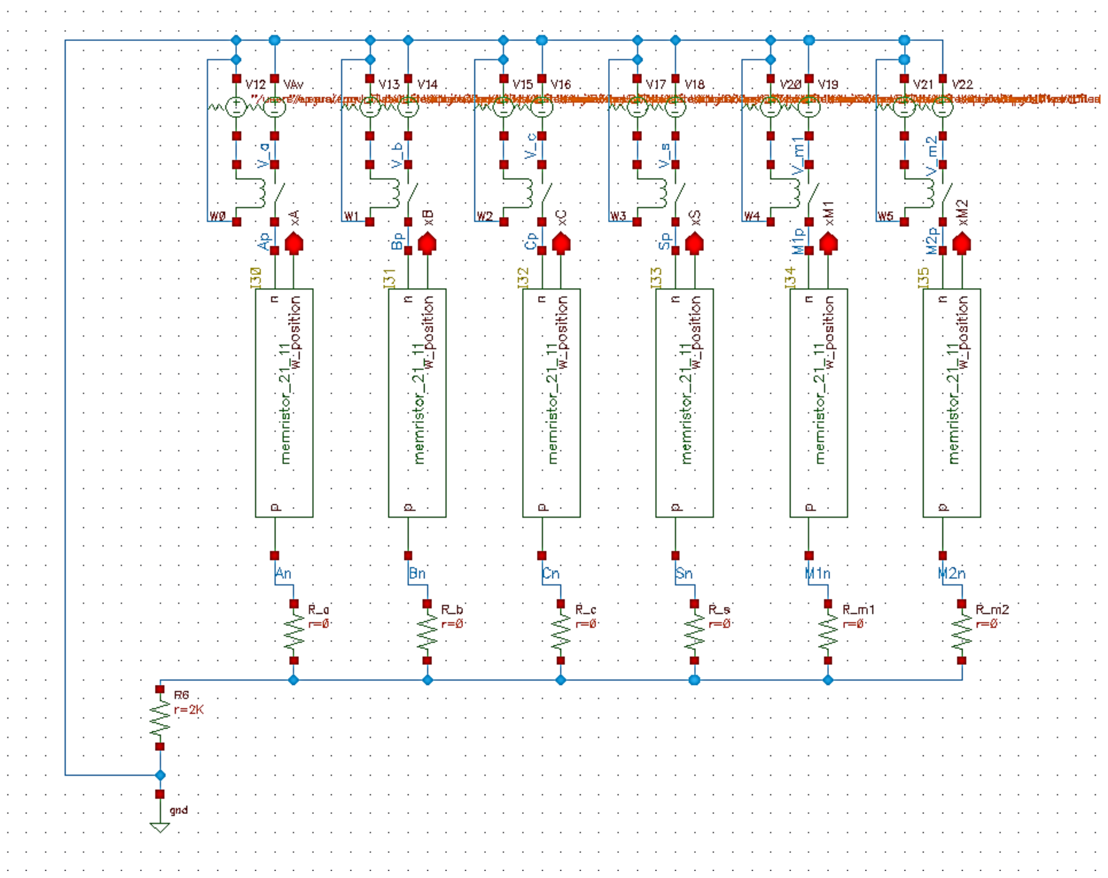


Figure 21 - IMPLY 1 Bit Full Adder Circuit

This implementation requires 6 memristors and 29 computation steps.

An example of simulation results for all 8 input combinations with memristors of α=5:

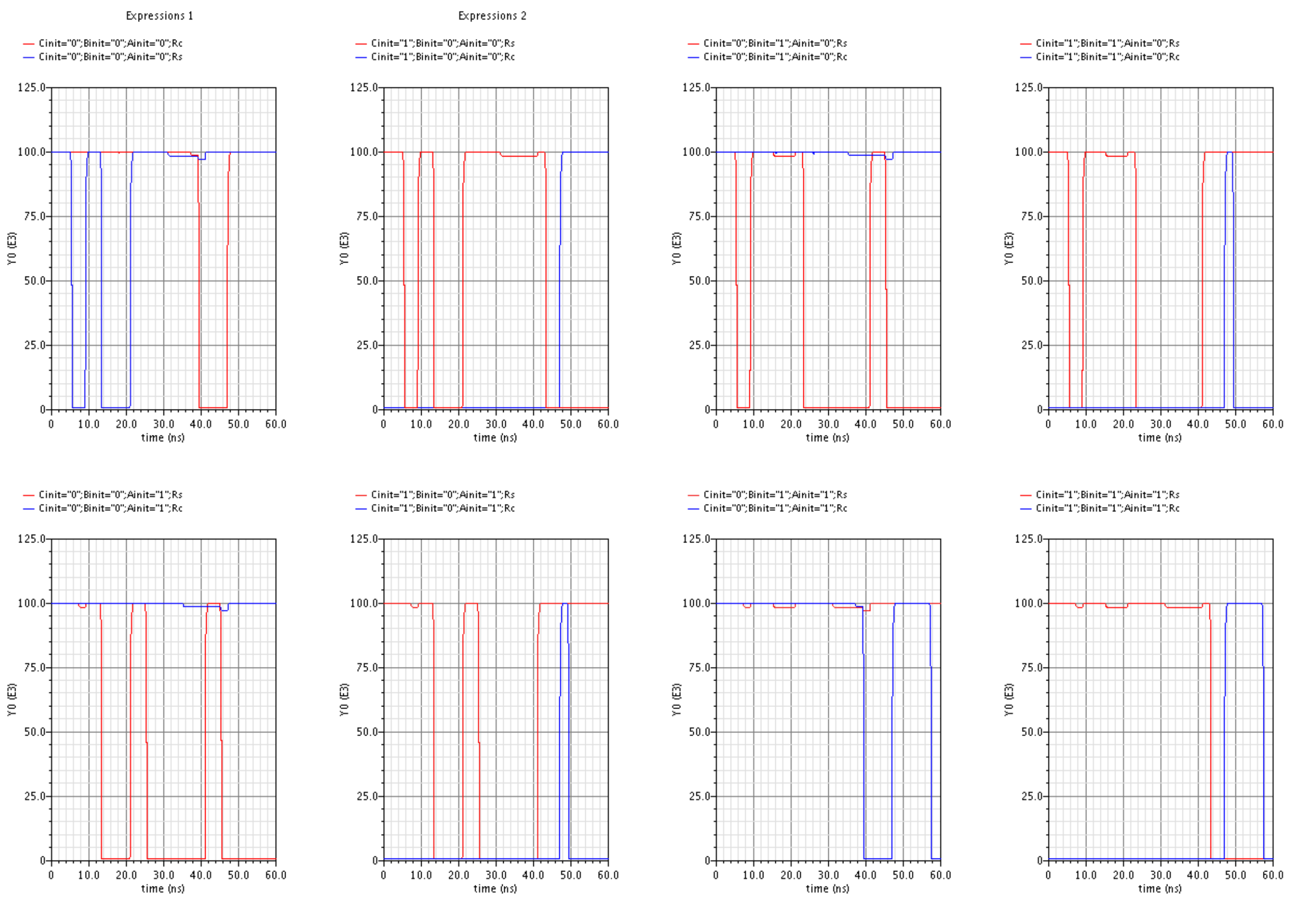


Figure 22 - IMPLY 1 Bit Full Adder Simulation Results

4.3.1. Driver Implementation

As can be seen in the circuit, each memristor is connected to a complex driver. This driver is designed to drive one of five different options: Float, Ground, Vreset, Vcond, Vset. This is achieved by two Vpwlf voltage sources (configured by a text file containing couples of time and voltages). One of the sources controls a switch (responsible for the floating state) and the other one drives the desired voltage when not floating. A schematic of the driver is shown:

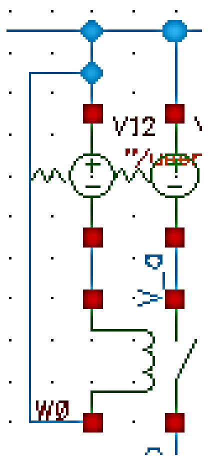


Figure 23 -IMPLY Voltage Driver

In order to create the Vpwlf files a Matlab code was created with an input of the desired sequence and an output of all the needed control files.

The Matlab code is presented in Appendix D1.

4.4. 8 bit Full Adder

A natural way to implement an 8 bit full adder is to concatenate eight 1 bit full adders in series. This method is simple for design yet the computation consumes a significant amount of time that can be reduced. As in carry look ahead systems it is desired to do a parallel computation of the result, this parallelism reduces significantly the computation time at the expense of circuit complexity and adding more memristors. This tradeoff is examined.

4.4.1. Serial Implementation

The circuit used for the serial implementation:

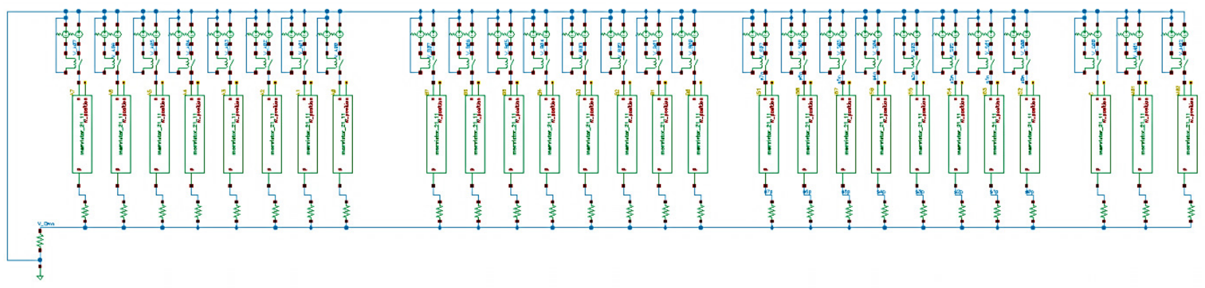


Figure 24 - IMPLY 8 Bit Full Adder Circuit

The sequence used here is the same sequence used for the 1 bit full adder replicated 8 times. Each level uses the memristor ‘C’ as it's carry in and writes its carry out to the same memristor.

This implementation results in a very long computation time. Since memristors are “inexpensive” it has limited advantages over the parallel implementation and for this reason it was decided to focus on the parllel direction.

One full simulation was produced for the memristor with α=5.

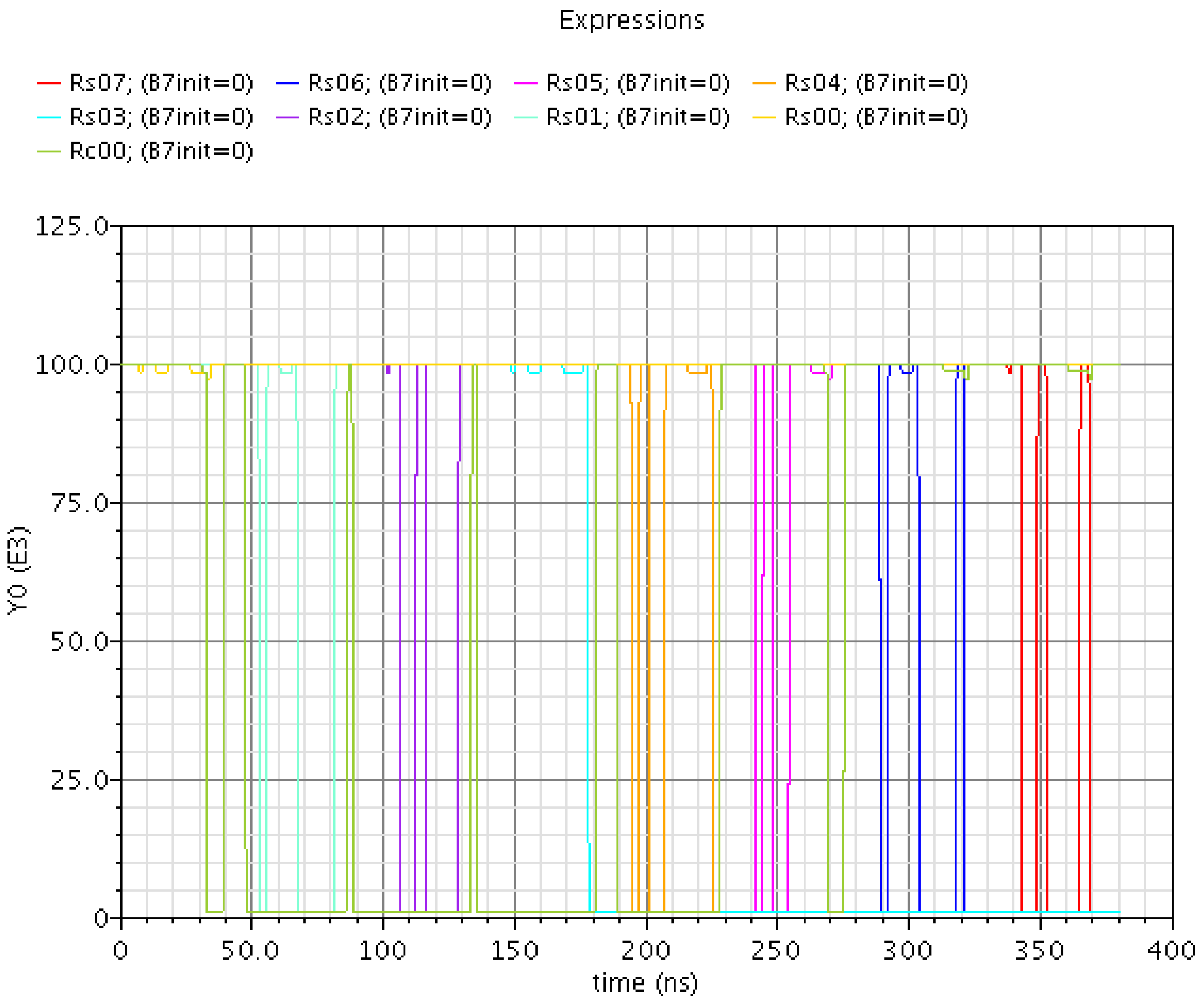


Figure 25 - IMPLY Serial 8 Bit Full Adder Simulation

This method requires 27 memristor and 232 calculation steps.

4.4.2. Parallel Implementation

In order to perform parallel computation a complex circuit is needed with the following features:

- Separate line for each bit (in order to perform logic on multiple bits simultaneously).
- Duplicate operation between bit lines, in order to pass carry.
- Perform simultaneous False on multiple memristors in the same line.

A circuit implementing these features:

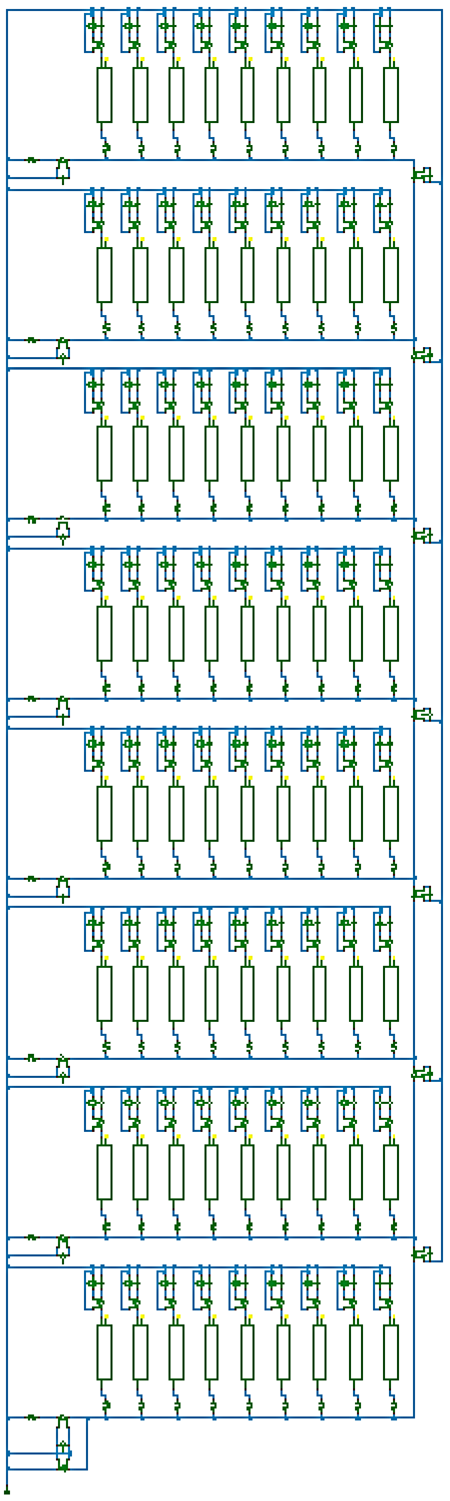


Figure 26 - IMPLY Parallel 8 Bit Full Adder Circuit

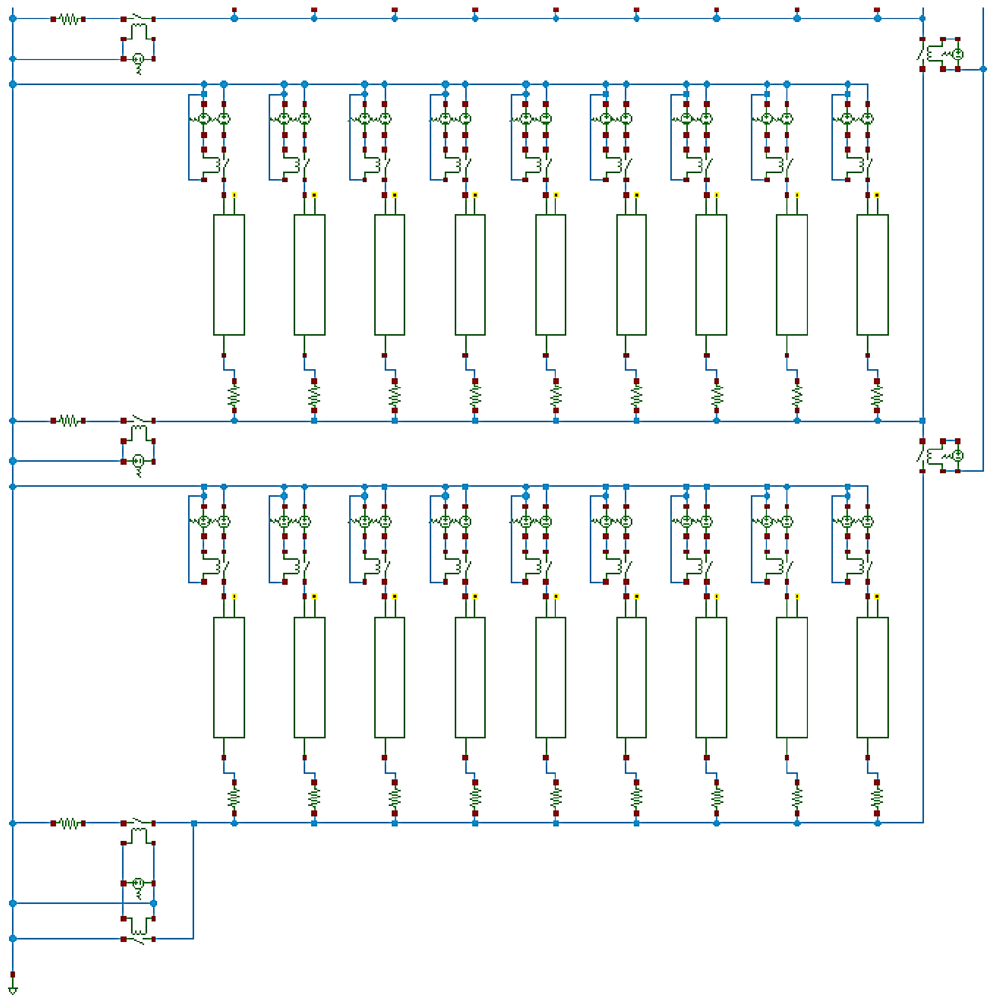


Figure 27 - IMPLY Parallel 8 Bit Full Adder Circuit Zoomed In

The circuit was broken down into bit lines in order to perform calculations simultaneously. The switches on the right and left sides were added to enable transfer of carry between the bit lines (by performing imply operations between memristors in different bit lines) and otherwise separating them and enabling simultaneous operations within all bit lines.

The computation sequence was designed in order to calculate all the possible steps in parallel. The sequence is composed of a parallel part of 13 steps in the beginning, followed by 5 steps that are repeated for each bit line (calculating carry and passing it on), and completed by another 5 steps that are done in parallel in order to calculate the final result for each bit. In order to save computation steps the number of memristors per bit was increased from 6 to 9. The complete sequence is presented in Appendix C2.

This implementation requires $8\,lines \cdot 9\frac{memristors}{line} = \; 72\,memristors$ and $13 + 5 \cdot 8bit + 5 = 58$ calculation steps.

A separate Matlab tool was created in order to prepare the control text files for the voltage sources (needed because additional sources and commands were added). The Matlab code is presented in Appendix D2.

An analysis of the sequence needed to compute an arbitrary Boolean function using IMPLY was suggested in [10], along with a specific N bit full adder sequence. A comparison between the various methods of computing an N bit adder is shown:

		Reference [10]	**Suggested implementation**	
			Serial	**Parallel**
Calculation steps		$89N$	$29N$	$5N + 18$
Memristors	Input	$2N + 1$	$2N + 1$	$2N + 1$
	Output	$N + 1$	$N + 1$	$N + 1$
	Work	3	1	$6N - 2$
	Total	$3N + 5$	$3N + 3$	$9N$
Special functions required	Parallel false needed			V
	Imply between lines			V
	True	V		

Table 6 - IMPLY Full Adder Implementation Methods Comparisson

5. MAGIC Logic Analysis

As described in the background and in Appendix B, the MAGIC family is broken down to 2 types of gates. The NOR and NOT gates can be built using a crossbar form, while all other gates require a unique structure for each gate. The two types are presented.

5.1. Non-Crossbar Gates Analysis

As presented in 3.2, the non-crossbar gates are OR, AND, NAND and NOT (with a resistor). Trying to connect an output of one gate as an input of the other is extremely problematic as shown in the next example.

Considering the simple case of concatenation of two AND gates. One AND gate is simple to produce:

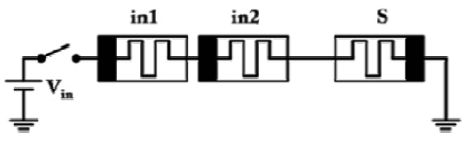


Figure 28 - MAGIC AND Gate

The output memristor S contains the value in the form of memristance. This output is then fed as an input to another AND gate. Considering the polarities of the memristor, the two combined gates are:

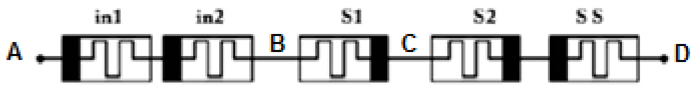


Figure 29 - 2 MAGIC AND Gates Concatenated

In order to perform the calculation the next steps are required:

Step 1:

- Node C: disconnect S1 from S2 and connect S1 to GND.
- Node A: drive V_{AND}.

Step 2:

- Node B: disconnect in2 from S1 and connect S1 to GND.
- Node D: drive V_{AND}.

In order to allow these connections a general node in the circuit should look like:

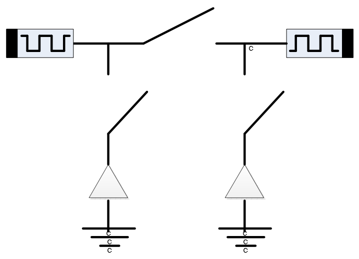


Figure 30 - MAGIC Non-crossbar Circuit Node

This results in every wire in the circuit transforming to at least 3 transistors which are controlled by a controller and 2 additional controlled voltage sources.
In case of a more complex function, more problems can arise. The first is that the polarity of the memristors is not always aligned to suit the gate implemented. The second is that sometimes a single memristor is needed as input of more than one gate and the node connectivity becomes much more complicated.

Due to these design difficulties this way does not seem to be viable in terms of efficiency, and the crossbar method seems to be a better solution.

5.2. Crossbar Gates Analysis

The MAGIC crossbar gates are NOR and NOT (without a resistor), these gates are described in much detail in Appendix B. As described in the last part of the Appendix, these gates suffer from a noise margin problem.

A few unsuccessful trials to overcome this problem are presented.

An example shows the result of a single NOR gate for the 3 types of inputs, for a memristor with α = 5 with an operating voltage of 1V (The maximum voltage allowed according to the design constraints)

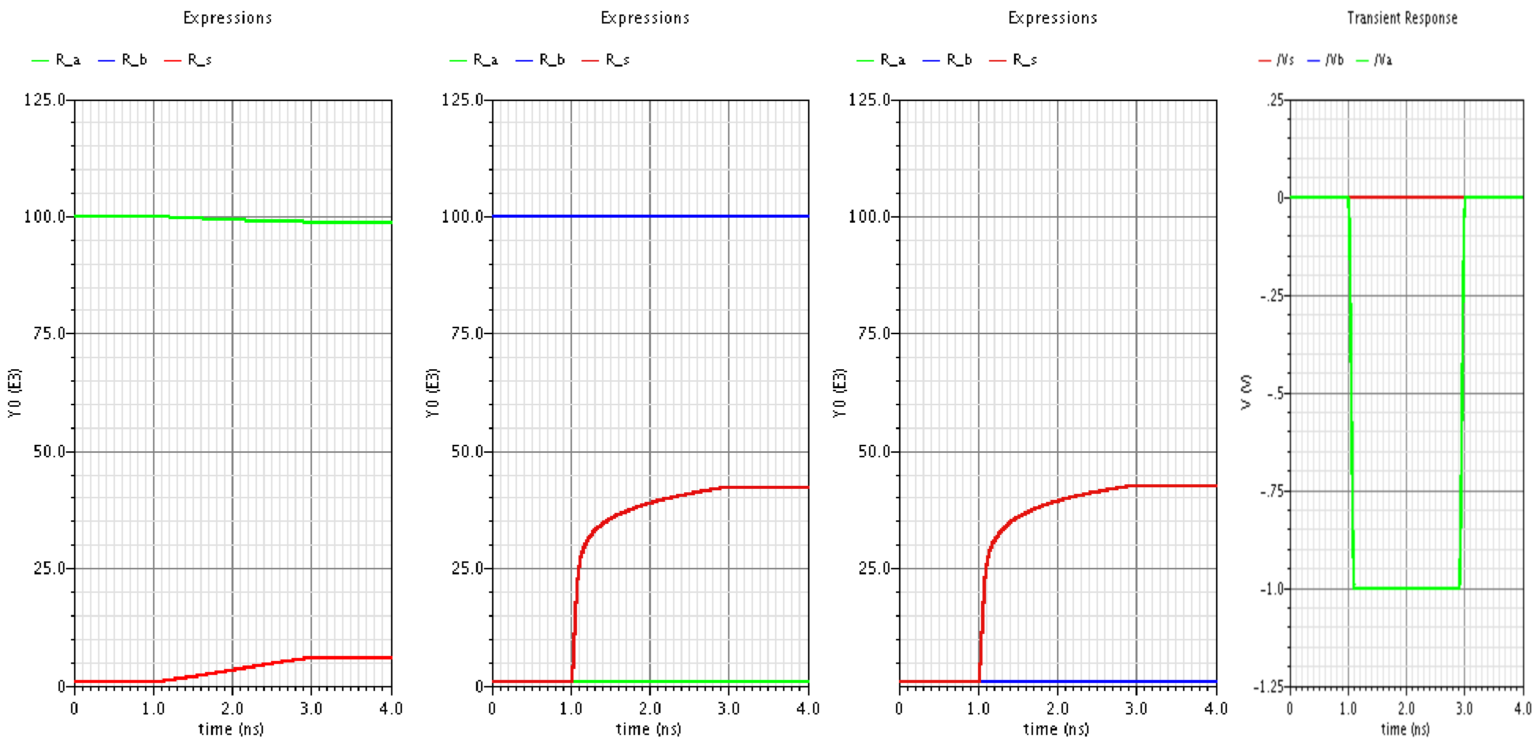


Figure 31 – MAGIC NOR Simulation Results

As can be seen the case of 00 as inputs leads to a small state drift in the output memristor, while the 01 and 11 inputs lead to partial switching. This partial switch is expected and described in Appendix B.

Overlooking the problem while connecting two NOR gates leads to a logical failure as shown in the next example. The graphs describe the result of NOR('0', NOR('1','1')). The expected result is '1' but due to the output of NOR('1','1') the result fails.

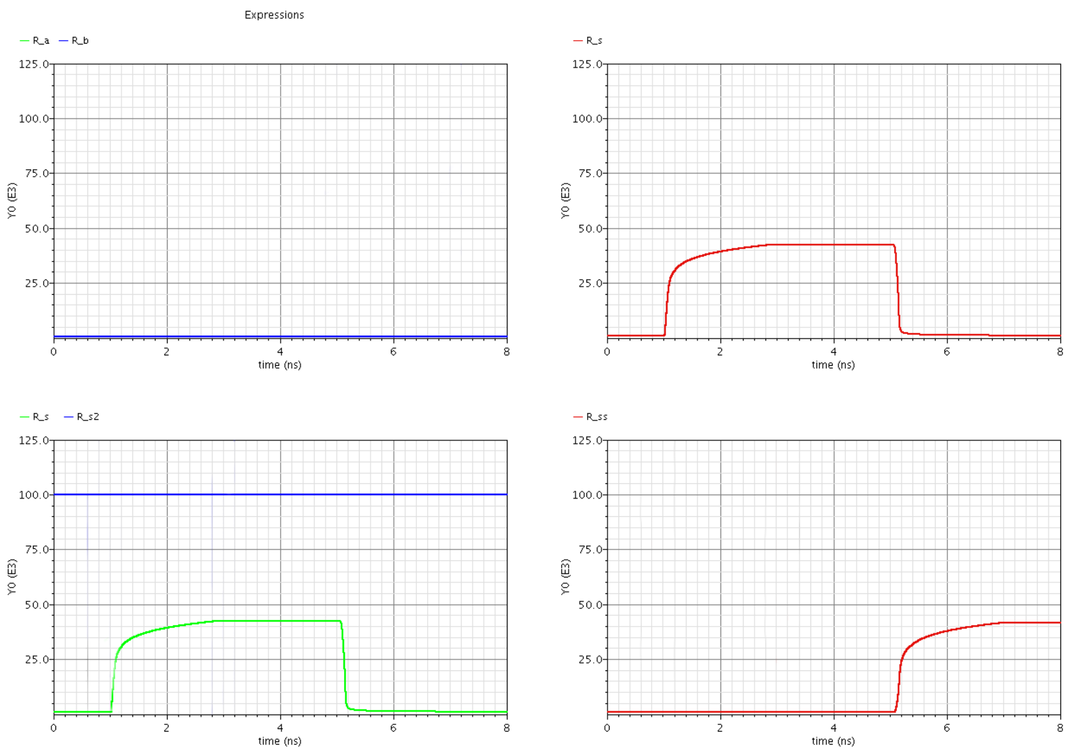


Figure 32 – MAGIC 2 NOR Gates Simulation

The left graphs are the NOR inputs and the right ones are the outputs. The top graphs are for the inner NOR gate and the bottom ones are for the outer gate.

This result demonstrates that a partial switching from '1' to '0' causes problems in cases where no switching should take place. Another visible problem is the change in the input memristance when it is not strong (bottom left green graph).

Using different memristor types and specifically linear memristors does not improve the results. Since the constrain on the operating voltage is of the type $< i_{th} \cdot \frac{R_{off}}{2}$, increasing the threshold or the value of R_{off} does not help since the switching will end anyway according to $i_{th} \cdot R_s = V = i_{th} \cdot \frac{R_{off}}{2}$ which results in $R_s = \frac{R_{off}}{2}$. This result is demonstrated in the next figure:

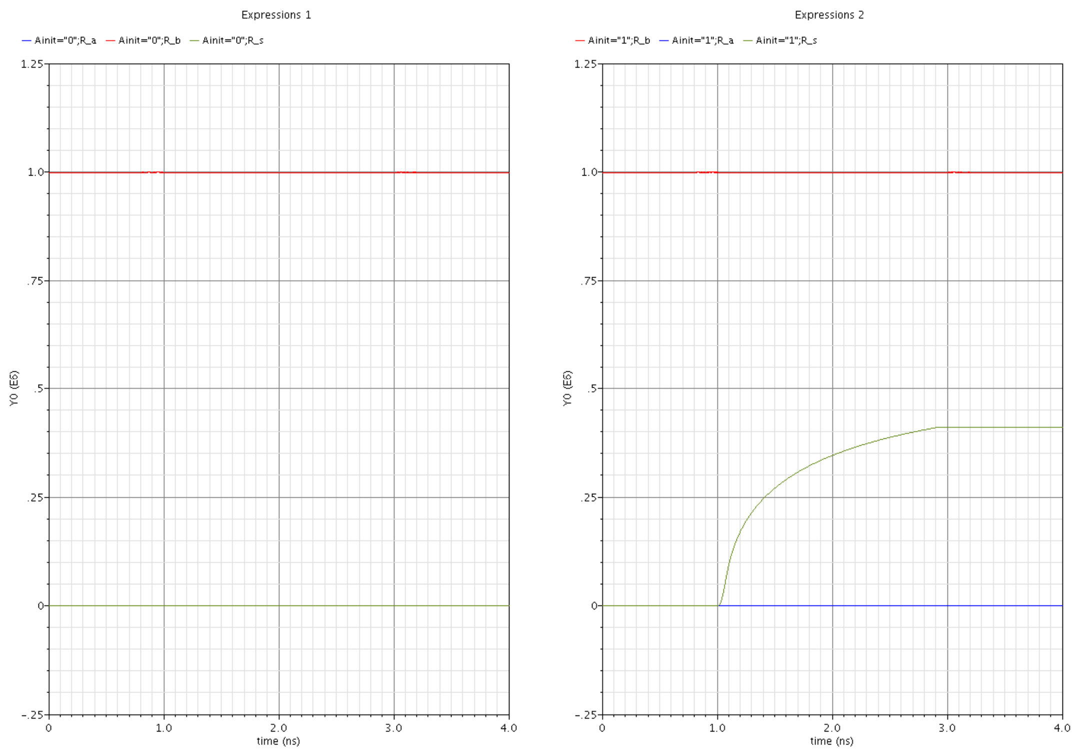


Figure 33 - MAGIC NOR, Inputs 00(left) 01(right), R_{OFF}=1MΩ, V_{NOR}=1V

As can be seen, the left graph shows no switch, as expected. The right graph shows switch to $\sim\frac{R_{off}}{2}$, regardless of the new value of R_{off}.

Trying to increase the applied voltage above the constraint in order to improve the partial switching at the cost of more state drift in no-switch case failed as well. This failure is due to the rate of change which is very fast at the beginning and extremely slow at the end ("self-strangling"). Thus, the aid to the partial switch is very limited, while the hurt in the no switch case is large.

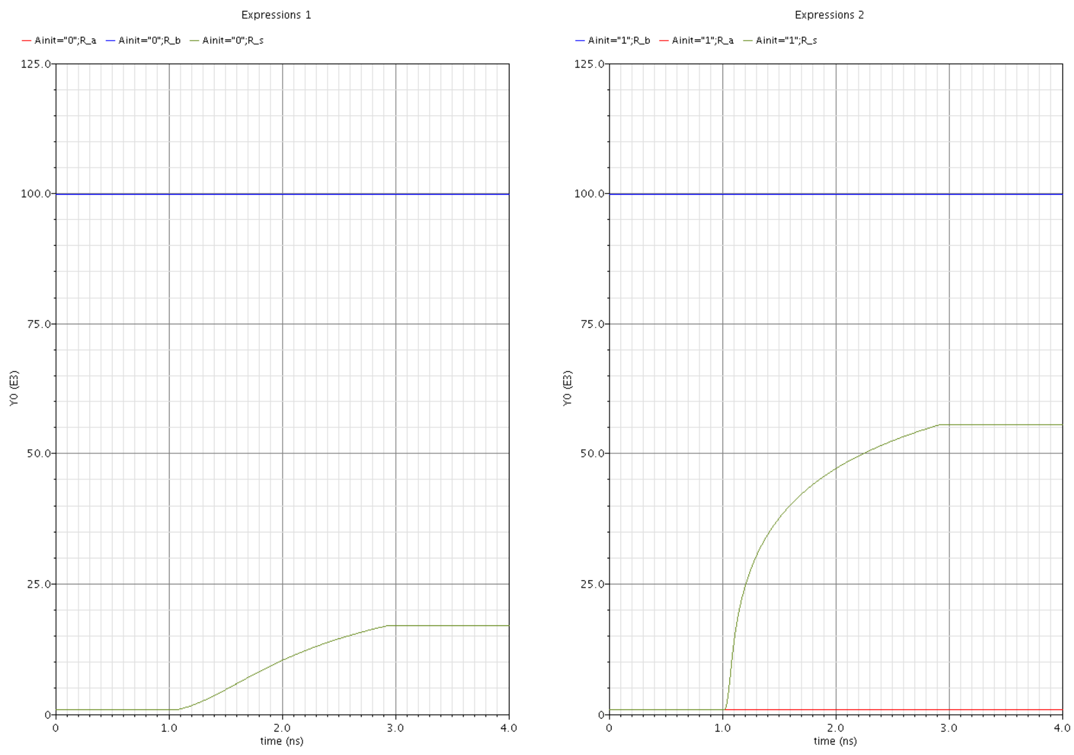


Figure 34 - MAGIC NOR, Inputs 00(left) 01(right), V_{NOR}=1.5V

As can be seen the state drift in the left picture is ~18%, while the partial switch improves only to 55% of the wanted value.

Various trials to strengthen the signal or avoid this partial switching problem with a crossbar type circuit are so far unsuccessful.

6. Hybrid CMOS Memristor logic analysis

This chapter will discuss the method of Hybrid CMOS Memristor logic. This method uses voltages to denote logic states. The AND and OR gates that were introduced in chapter 3.3 do not comprise a full set of operations so additional CMOS inverters (or similar) are needed.

6.1. Single gate analysis

An OR gate is built in the following way:

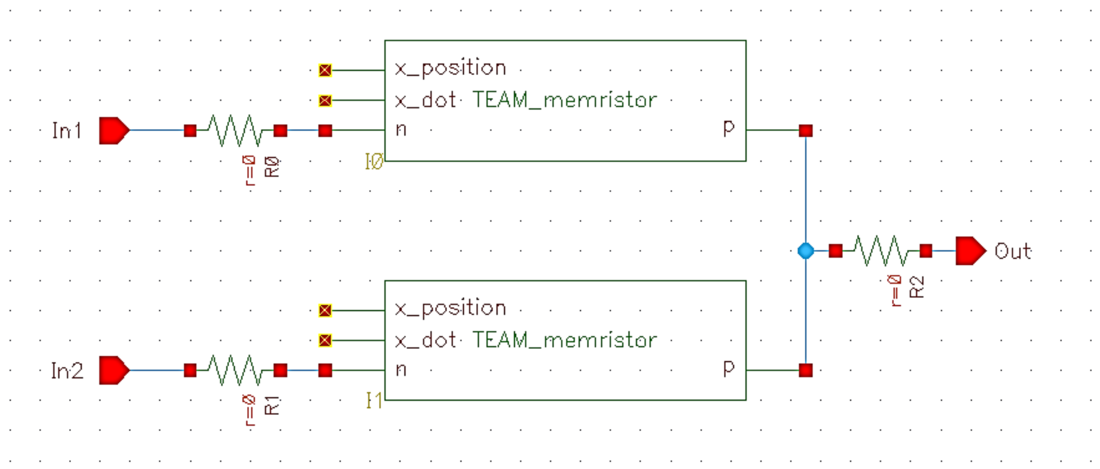


Figure 35 - Hybrid CMOS OR Gate Circuit

As in previous gates, resistors with zero resistance are connected to all nodes to ensure that there are no forbidden connections when the gate is connected to a circuit.

The gate is assigned a symbol for use in future circuits:

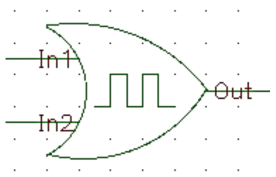


Figure 36 - Hybrid CMOS OR Gate Symbol

Simulation results for a single OR gate with a floating output node (for memristors with α=5 and working voltage of 4 V) :

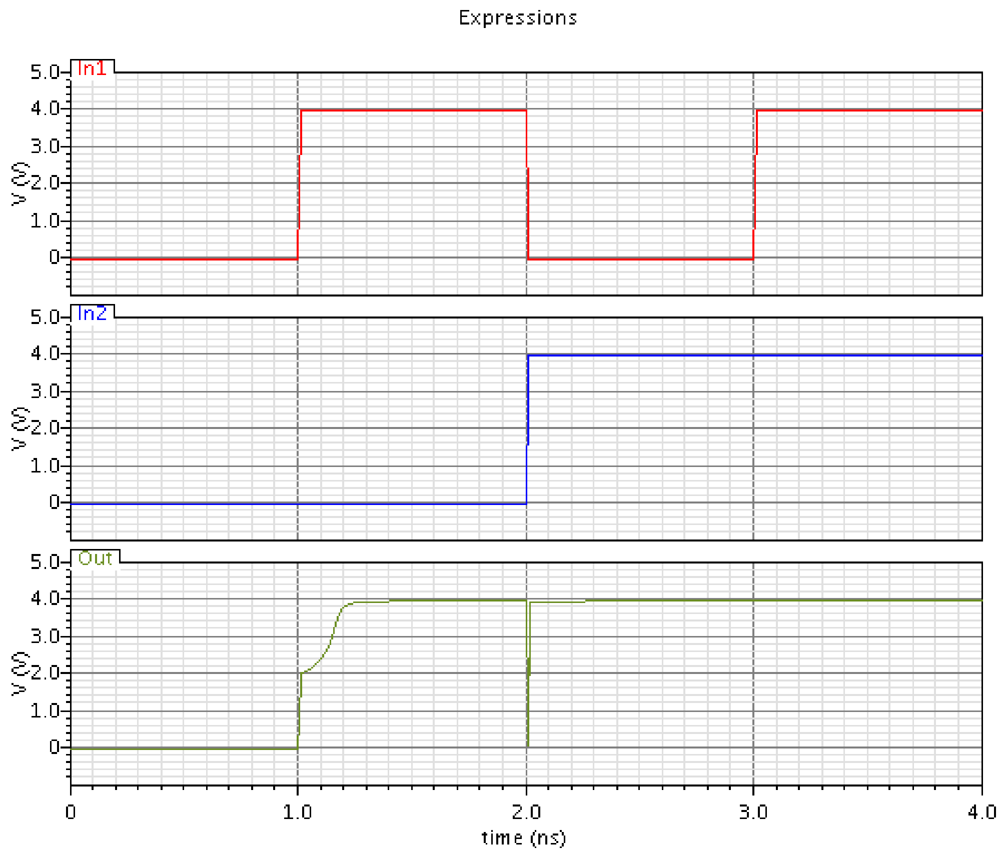


Figure 37 - Hybrid CMOS OR Gate Simulation Results

The circuit used to create an AND gate:

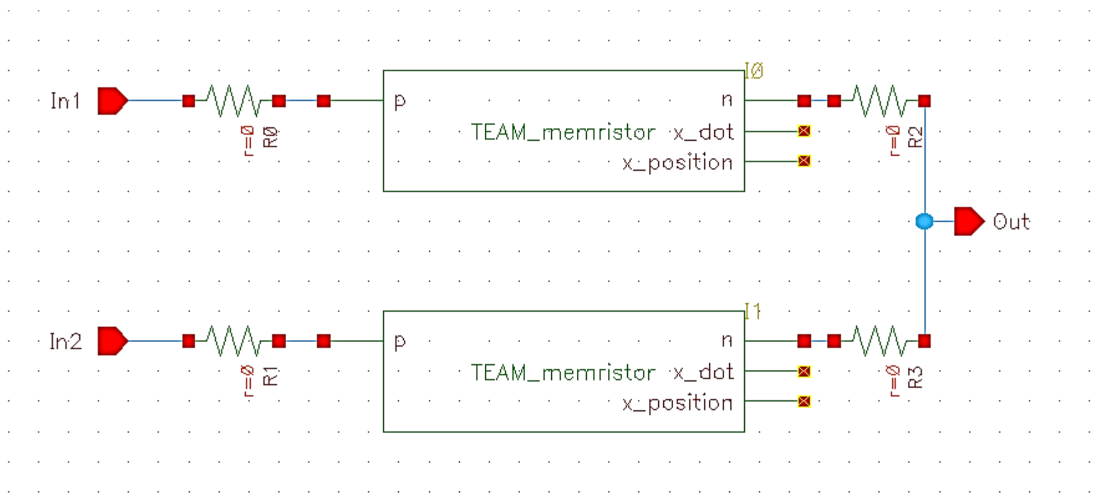


Figure 38 - Hybrid CMOS AND Gate Circuit

As can be seen, this gate is identical to the OR gate with the exception of reverse memristor polarity.

The AND gate was also assigned the appropriate symbol:

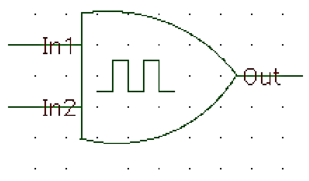


Figure 39 - Hybrid CMOS AND Gate Symbol

Simulation results for a single AND gate with a floating output node (for memristors with α=5 and working voltage of 4 V):

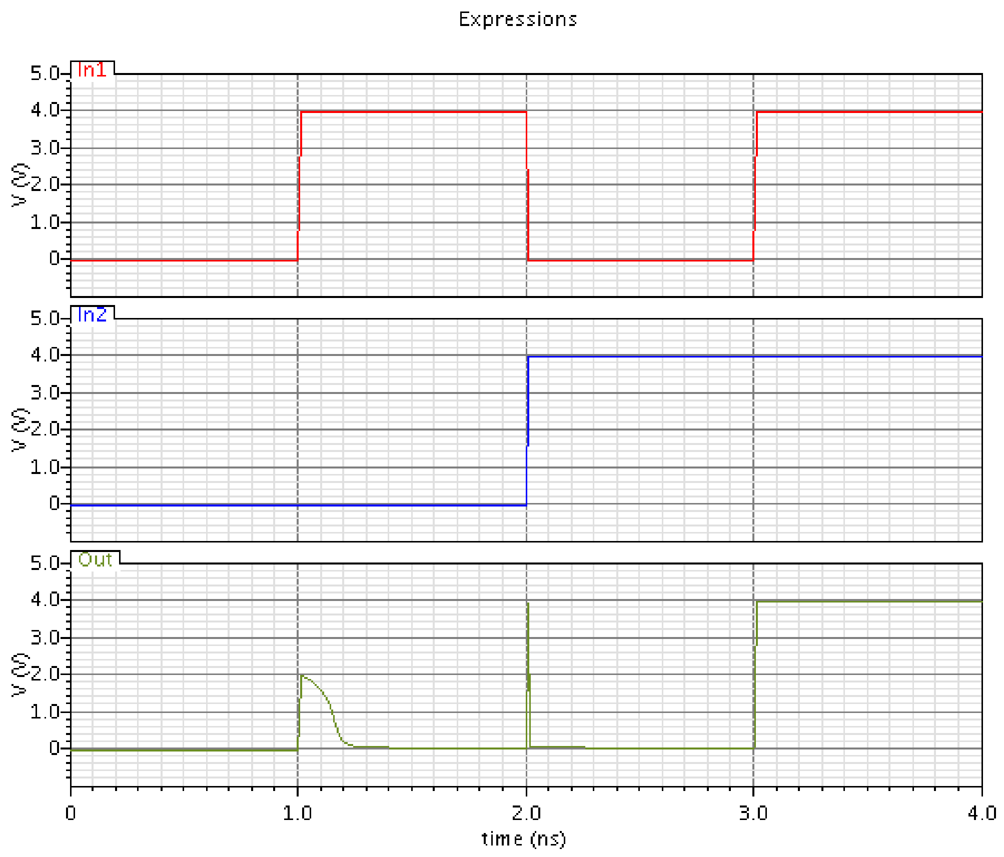


Figure 40 - Hybrid CMOS AND Gate Simulation Results

The simulations above demonstrate a few of the gate's attributes.

In can be seen that the output voltage doesn't always remain in the correct value even if the change in inputs does not call for a change in the output. This happens due to the fact that the output gets it's voltage from the node connected to the memristor with lower memristance, and that for different inputs the memristors might switch values.

The meaning of this is that these gates can cause "Dynamic Hazards" and therefore the output value cannot be sampled shortly after a change in the input values, but only after a measured propagation time. This phenomenon will affect the timing constraints of any logic implemented with this family as can be seen in the following sections.

6.2. 1 bit Full Adder

As mentioned, a CMOS inverter is needed in order to implement any logic function. The inverter used, is implemented using a Tower 0.18µm process:

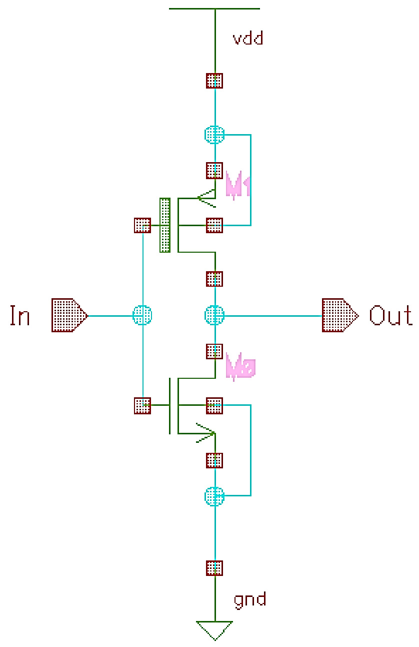


Figure 411 - CMOS Inverter Circuit

The dimensions of the transistors are Wn=0.84µm, Wp=1.8µm, L=0.18µm. The gate behavior is shown for various supply voltages. The following graphs show the output of a single inverter, a buffer made of two inverters in a row, and two serial buffers.

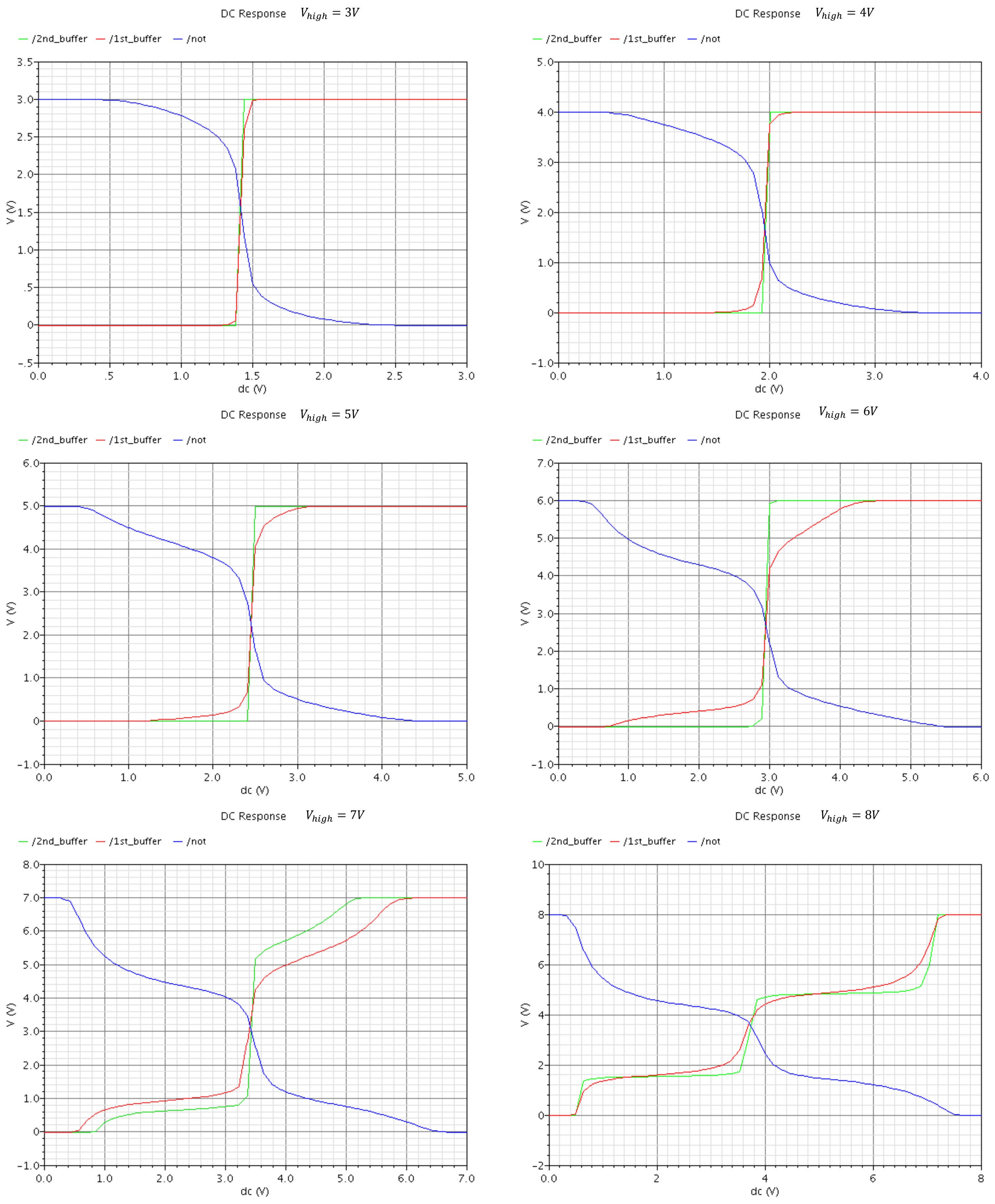


Figure 422 – DC Response for CMOS Inverter, Buffer, and 2 Series Buffers

It can be seen that the NOT gate can work with voltage of up to 6V, and possibly 7V with more than 2 buffers. A supply voltage of 8V causes failure.

Implementing a full adder requires the realization of:

$S = (A \oplus B) \oplus C \quad , \; C_{out} = \left(AB + C_{in}(A \oplus B)\right)$

As can be seen, these functions require the use of a XOR gate multiple times. For this reason a XOR gate was implemented using the gates shown previously and the CMOS inverter.

The XOR gate was implemented by using $A \oplus B = A\overline{B} + \overline{A}B$ in the following circuit:

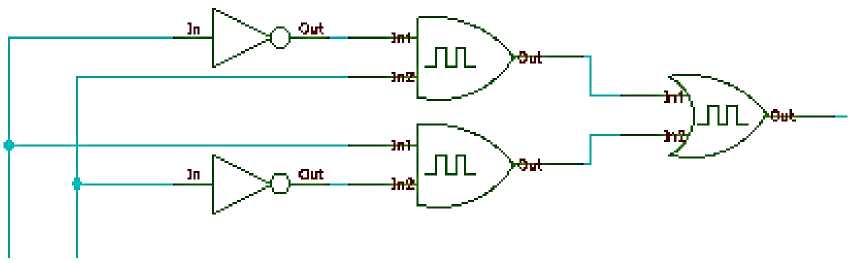


Figure 43 - Hybrid CMOS XOR Gate

Results of this circuit (for memristors with α=5 and working voltage of 4 V) are:

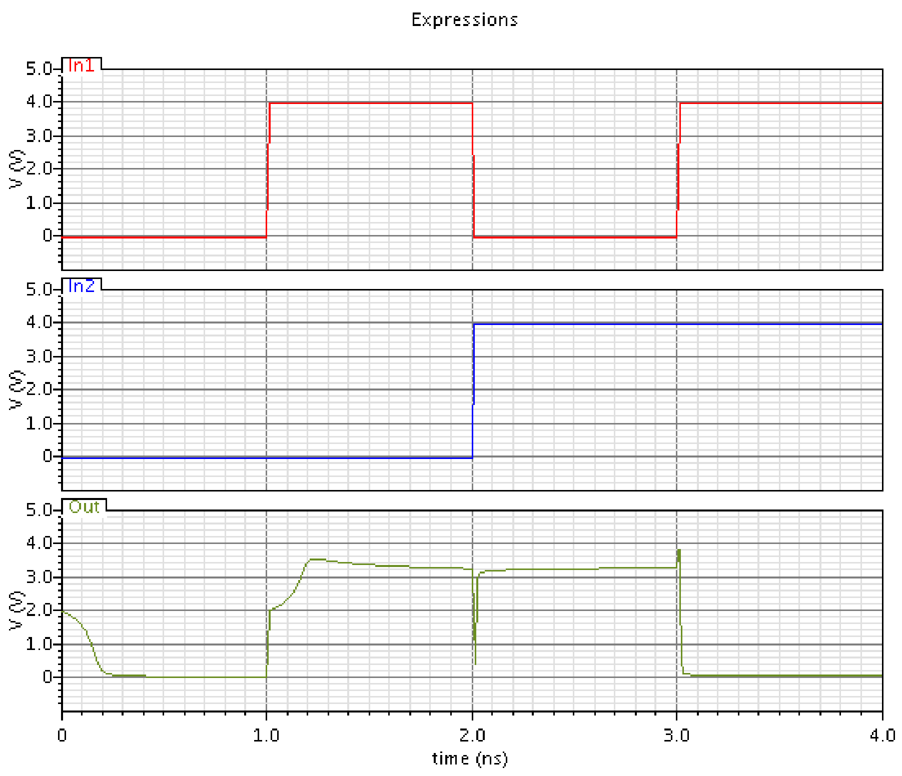


Figure 44 – Hybrid CMOS XOR Gate Simulation Results

A very important problem of this logic family is demonstrated in this graph. In addition to the erosion of the output voltage caused by the existence of a voltage divider (expected to be ~1%). Another, more significant difference is measured between the output voltages and the supply voltages after settling. This difference is explained by the fact that in this circuit, the output of one gate is connected to the input of another. This causes some of the current that flows through an input node to exit the gate from the output node instead of the other input. Because not all the current flows through one of the memristors a full switching might not occur (due to the existence of a threshold) and the output voltage is not always close to one of the edges.
For this reason, memristors with lower threshold voltages are less sensitive to the issue and give improved results compared to memristors with high thresholds.

The effect is highly dependent on the structure of the circuit. Furthermore, for different starting states of the memristors, different results can be measured for the same inputs. Because of these reasons it is very difficult to model the effect and solve it. A very straightforward approach is to buffer every two gates with a CMOS element (a buffer or an inverter), thus solving the problem by eliminating the current that flows through the output node (because the gate of a transistor does not allow the flow of current).

The integration of CMOS gates and memristors forces usage of voltages that are suitable for the process used in order to prevent breaches and other harmful effects. Moreover, the connection between the CMOS level (silicon) and the memristor level (metals) presumably requires the use of area consuming VIAs. To reduce this, the number of buffers used is reduced to the minimum necessary to keep the correct logic behavior of the circuit.

The 1 Bit Full Adder circuit is implemented in the following circuit:

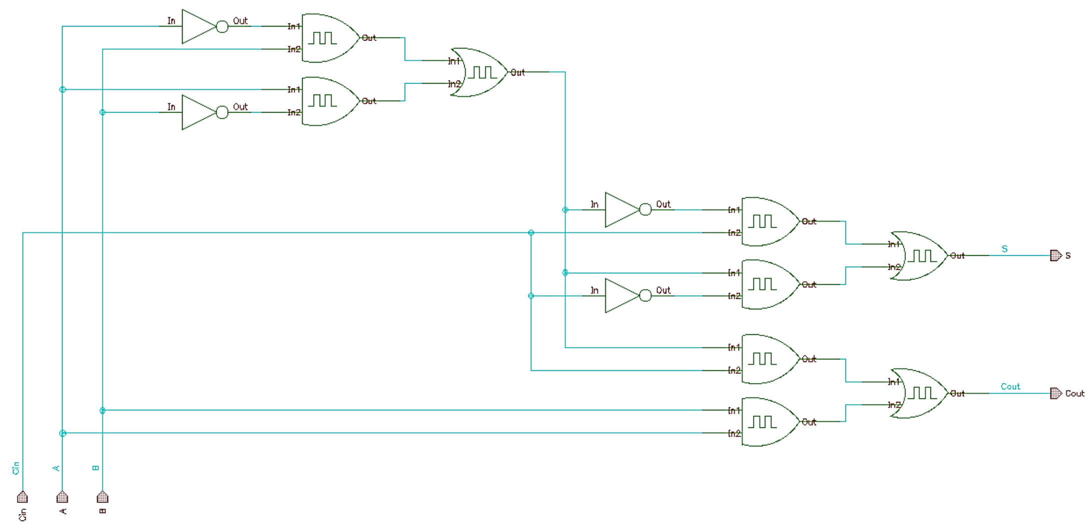


Figure 45 - Hybrid CMOS 1 Bit Full Adder Circuit

For memristors with α=5 the circuit was tested with all possible inputs and with different supply voltages to determine the voltage in which it works best. The results for the worst conditions (Cin=0 A=1 B=0) are shown. Two inverters were placed after each output S is the voltage in the output, S2 is after the first inverter and S3 is after the buffer.

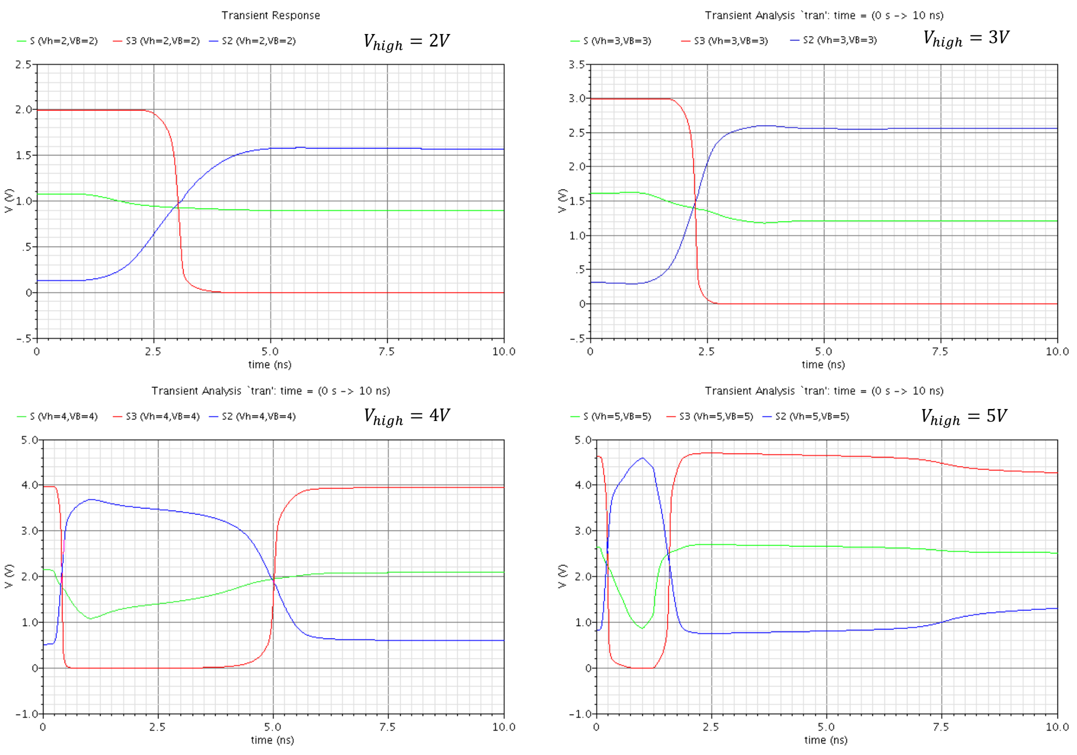


Figure 46 – Hybrid CMOS 1 bit Full Adder Results for Various Voltages

For voltages VDD=2V,3V the results are wrong, there is a logic failure inside the circuit which can be resolved only by adding buffers between the gates.

For VDD=4V the results are satisfactory after the buffer.

For VDD=5V the result after the buffer is logically correct but another buffer is needed in order to get the output voltage to VDD.

The same process is done for each type of memristor. Simulations are done to decide on the appropriate operating voltage of the circuit by checking what voltage gives the most distinct results (voltages closer to Vdd or GND and not in the middle). If possible, a lower voltage is chosen to reduce the currents flowing in the circuit and hence the power dissipated.

For the memristors of α=10 and α=1 with threshold, a different approach had to be taken. To overcome the large threshold current of 20μA a high voltage is needed, yet the inverter can't work at voltages higher than 7V. Buffers must be added in the middle of the 1 bit adder. Adding 2 buffers after the circuit part that performs $A \oplus B$ solves the problem, and allows work with a voltage of 6.5V which is suitable for the CMOS process. Still, 2 buffers are also needed after the S output and for Cout one buffer is needed for α=1 and 2 are needed for α=10.

A summary of the needed buffers for each type of memristor is given in the next part.

The results for a 1 bit Full Adder with memristors of α=3 are shown:

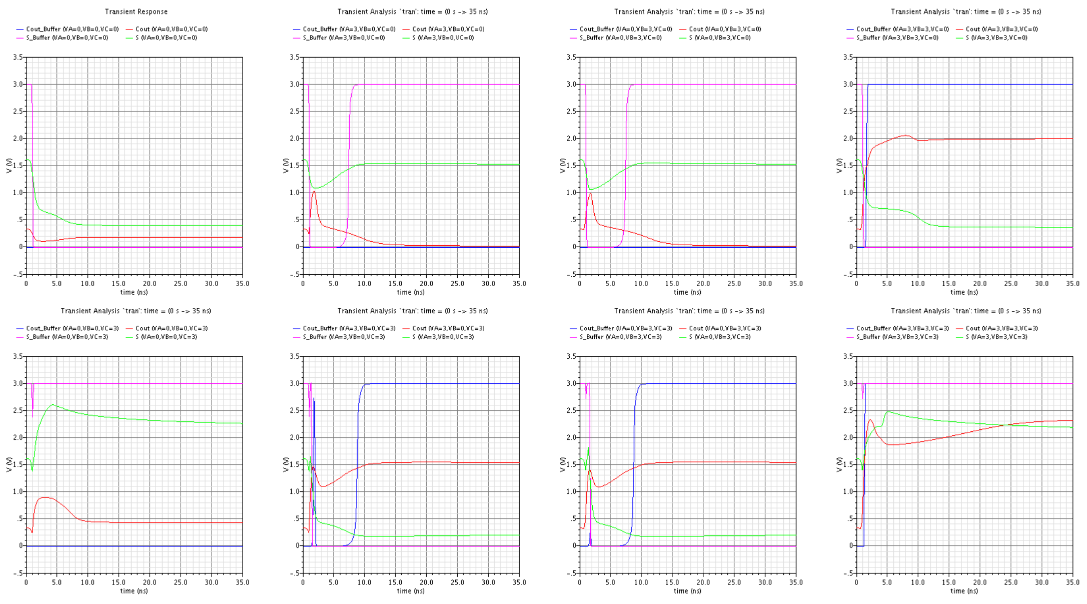


Figure 47 – Hybrid CMOS 1 bit Full Adder Results for All Possible Inputs

For each memristor, the maximum calculation time (output settling) can be found from the graphs.

6.3. 8 bit Full Adder

After figuring out the operating voltages and buffers needed, an 8 bit full adder can be built simply by connecting the circuits that were built in a chain as if they were regular CMOS 1 bit full adders. The 1 bit Full Adder circuit is represented by the following symbol:

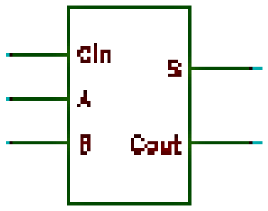


Figure 48 - Hybrid CMOS 1 Bit Full Adder Cell

An example of an 8 Bit Full Adder for the memristor type of α=1 with threshold is shown:

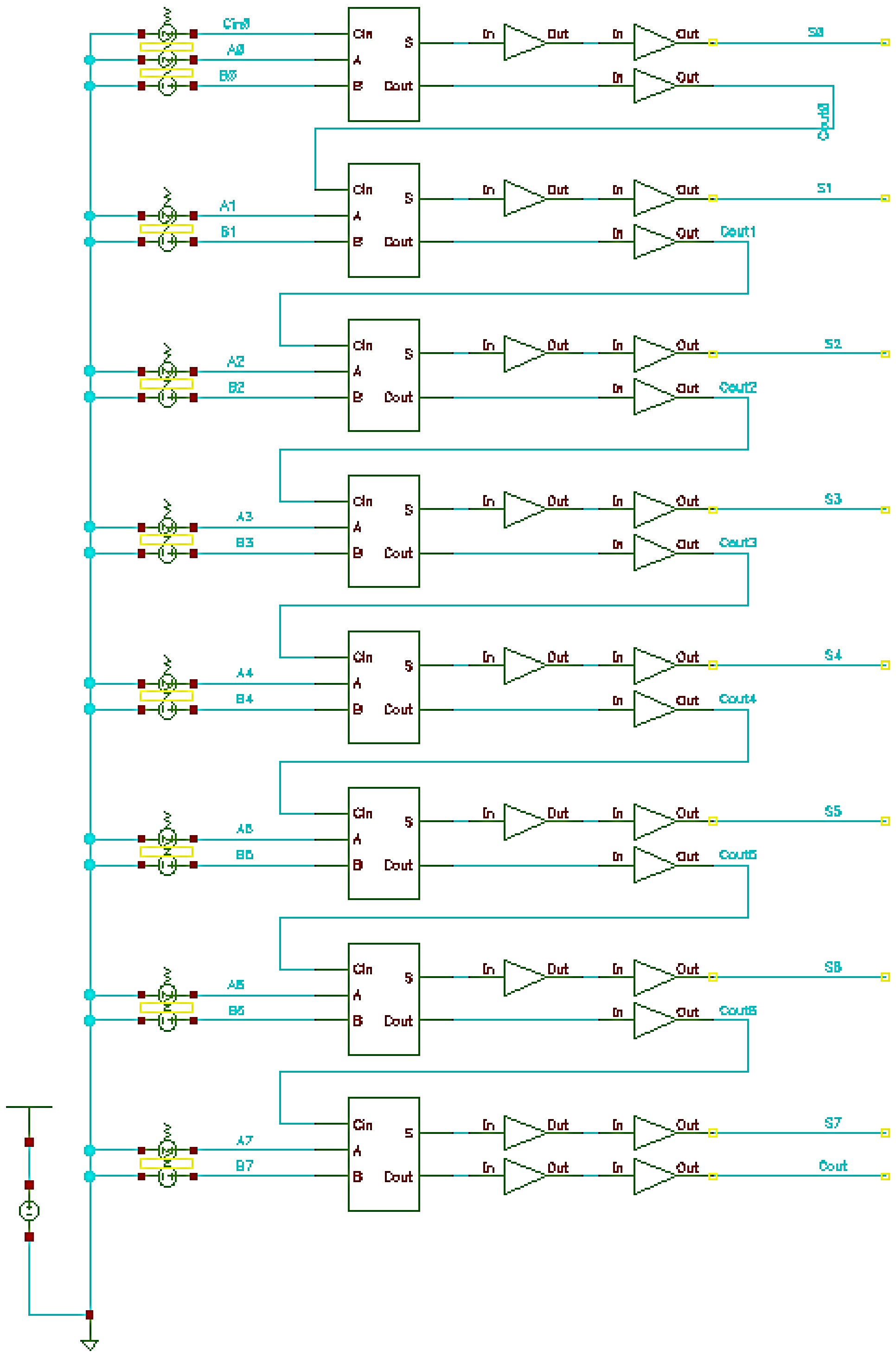


Figure 49 - Hybrid CMOS 8 Bit Full Adder Circuit

Each 1 bit Full Adder cell contains additional 2 buffers inside as mentioned. Note that all "S" outputs contain the same number of buffers while for "Cout" the last output may contain more than all the rest of the intermediate ones. The reason for that is that all the outputs that can be read from outside the circuit must hold a "strong" logic value. Cout nodes that are used only as Cin for the next level should only have values strong enough for the next level to operate.

A summary of the buffers needed for all memristor types:

Memristor Type	Supply Voltage Used	Buffers Count			
		In the middle of 1 bit adder	between Cout->Cin	S	Final Cout
α=1 (w/o threshold)	1V	0	2	1	2
α=1 with threshold	6.5V	2	1	2	2
α=3	3V	0	1	1	1
α=5	4V	0	2	1	2
α=10	6.5V	2	2	2	2

Table 7 - Hybrid CMOS Required Buffers Summary

Example results for an 8 bit Full Adder with memristors of α=3 is shown. The inputs of the circuit are designed for worst possible propagation time (A=11111111, B=00000001):

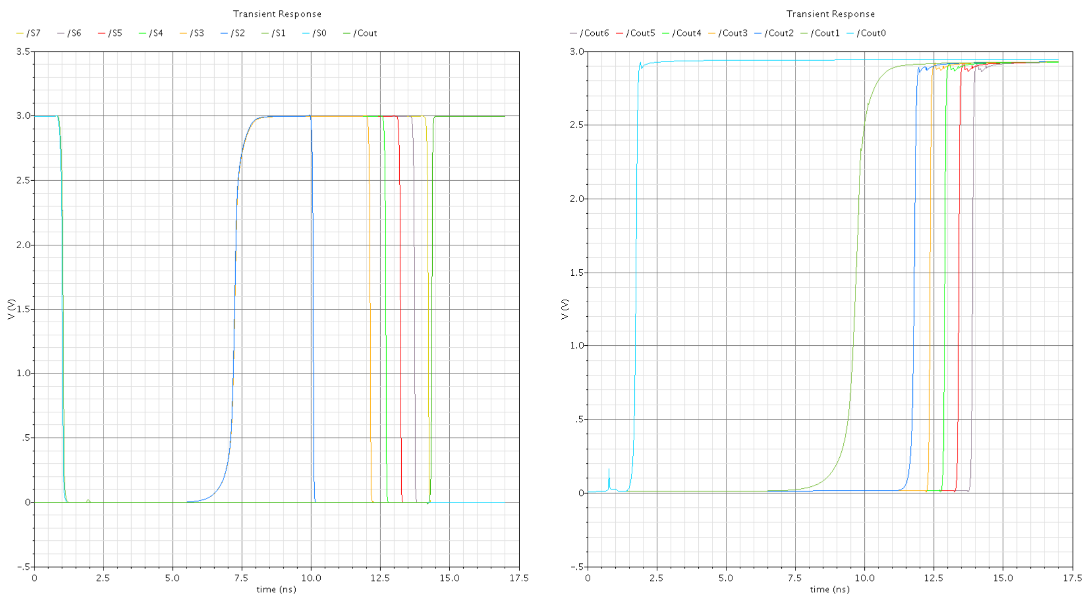


Figure 50 – Hybrid CMOS 8 bit Full Adder Results (Worst Propagation Time)

As can be seen, there is a long initial propagation time from the change in inputs until any change in the outputs is visible. This is due to the fact that the inputs propagate through the long route containing the XOR gate. After this time, the changes become more frequent because they are the result of the carry propagation which is done via a shorter route.

Considerations in designing Hybrid CMOS Memristor logic:

- Current threshold:
  Memristors with low current threshold are wanted to allow fast switching with relatively low voltages.
- Buffers must be inserted in some places to prevent voltage decay. The number of consecutive buffers is not important, but the number of CMOS Memristor transitions should be minimized.

7. Programmable Logic Array (PLA) Analysis

As described in 3.4 memristors can be used to implement programmable logic arrays (PLA). The memristors are used as the connections between the vertical and horizontal wires. A memristor set to R_{off} is assumed to be disconnected and memristor set to R_{on} is assumed to be connected. In order to perform logic the working voltages should be selected so that the currents through the memristors are sub-threshold to prevent state changes.

The PLA to be used here is a PLA in which the vertical wires (left part) create the minterms, and the horizontal wires (right part) create the sum of minterms.

Similar to the Hybrid logic, CMOS gates are needed to implement the PLA. In order to simplify the design a 4-input NAND CMOS gate was created:

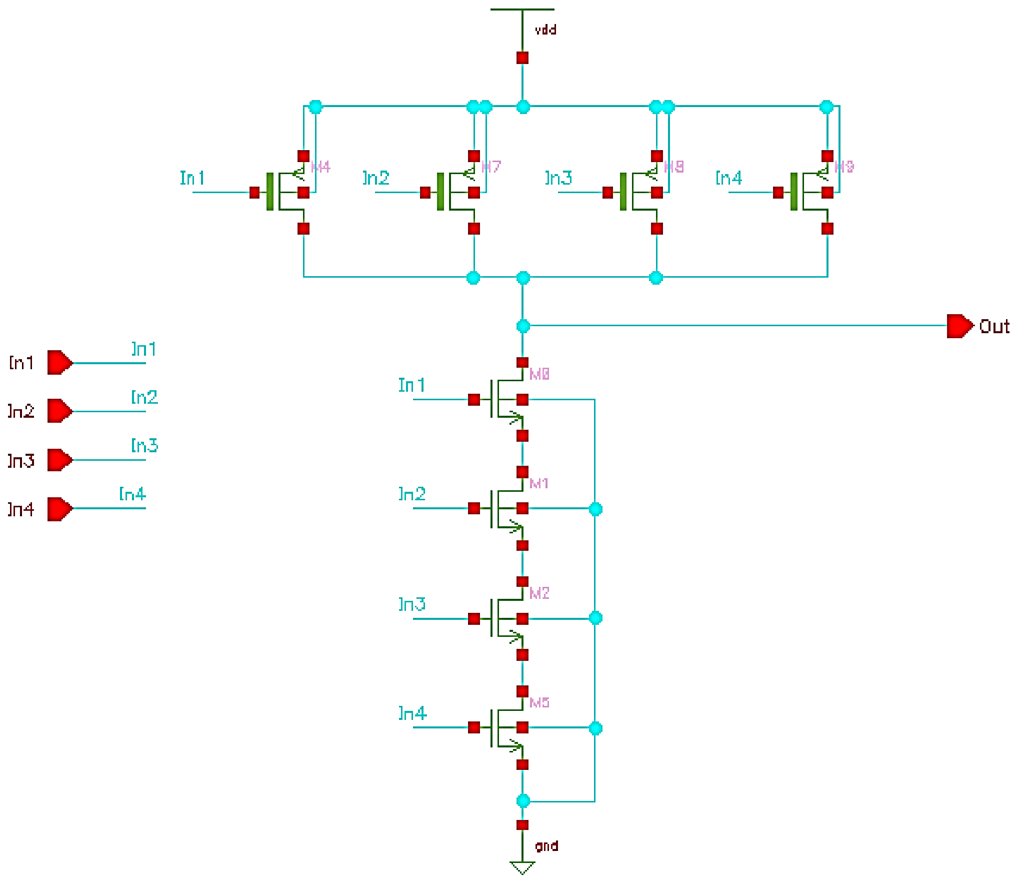


Figure 51 - NAND4 implemented in CMOS for PLA

DC response of this CMOS NAND-4 for various working voltages (V_{high}) is presented:

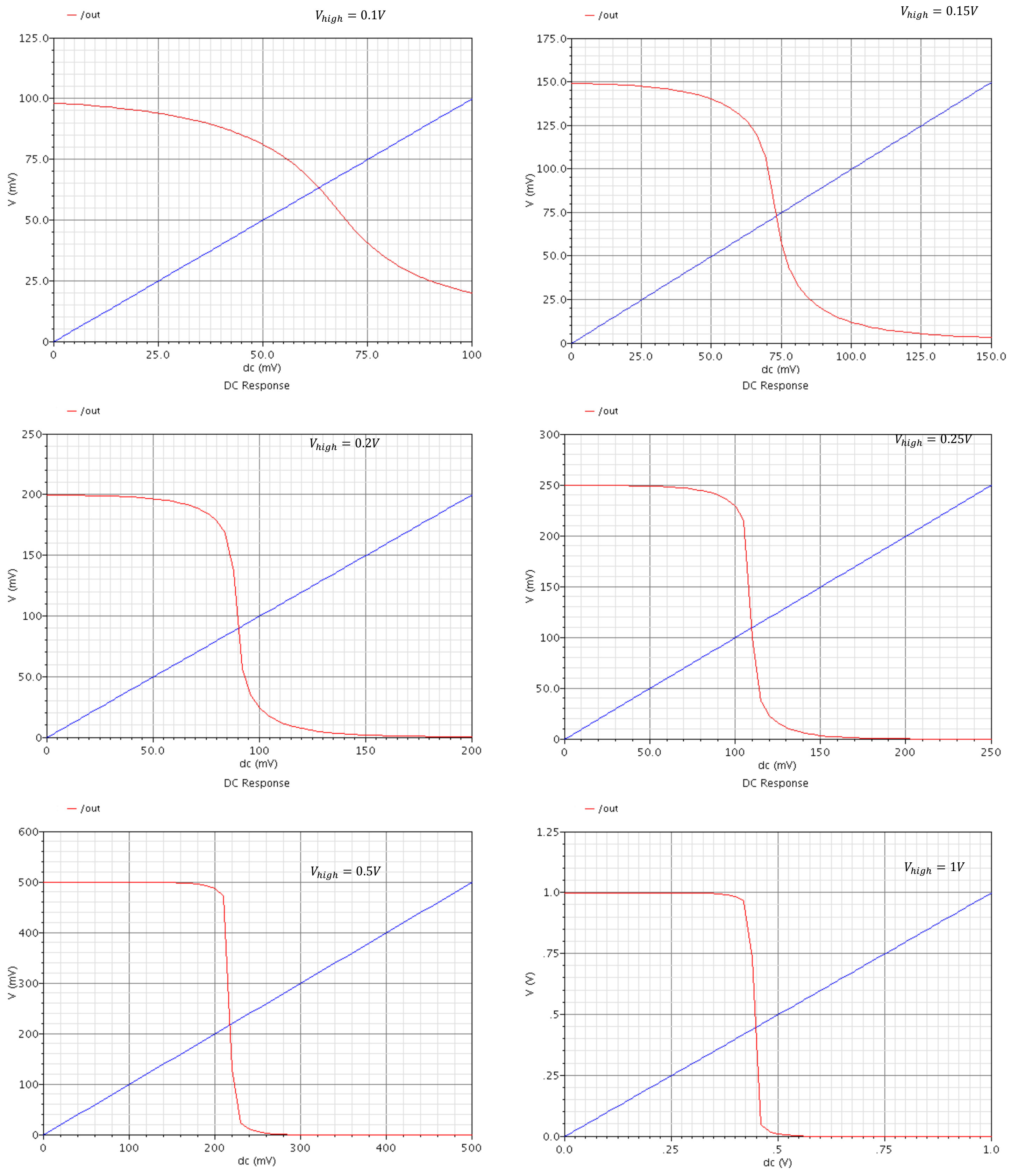


Figure 52 – CMOS NAND4 DC Response for Various Voltages

Selecting the working voltages is the main issue in designing memristor based PLA, since the memristor models used in this project are defined with a current threshold, and the designer controls the working voltages.
In order to be sure to work sub threshold the safest approach is to choose $V_{high} \leq R_{off} i_{th}$. This is a rigorous approach since many current dividers in the circuit can reduce this demand and allow higher working voltage. Since PLA are programmable this fine tuning of selecting the voltages is limited.

7.1. 1 bit Full Adder

A 1 bit full adder contains 3 inputs and 2 outputs.

A general memristor based 3 input, 2 output PLA circuit was designed:

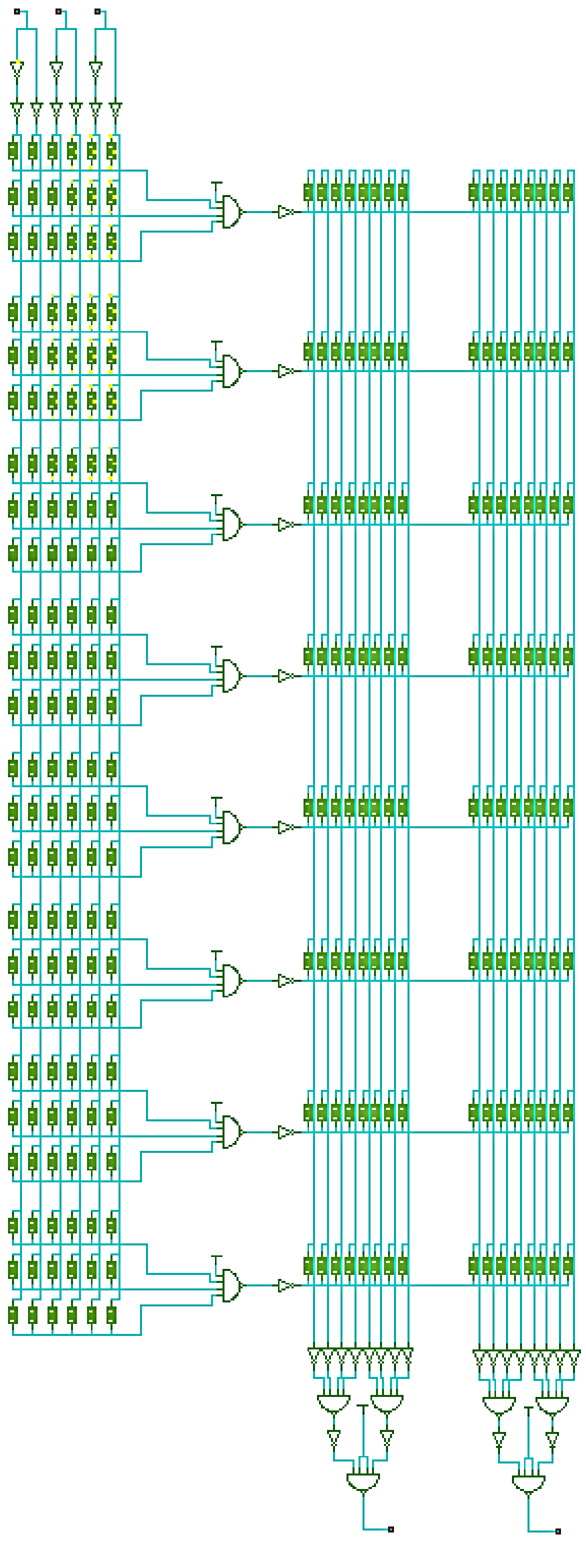


Figure 53 - memristor based 3 input, 2 output PLA array

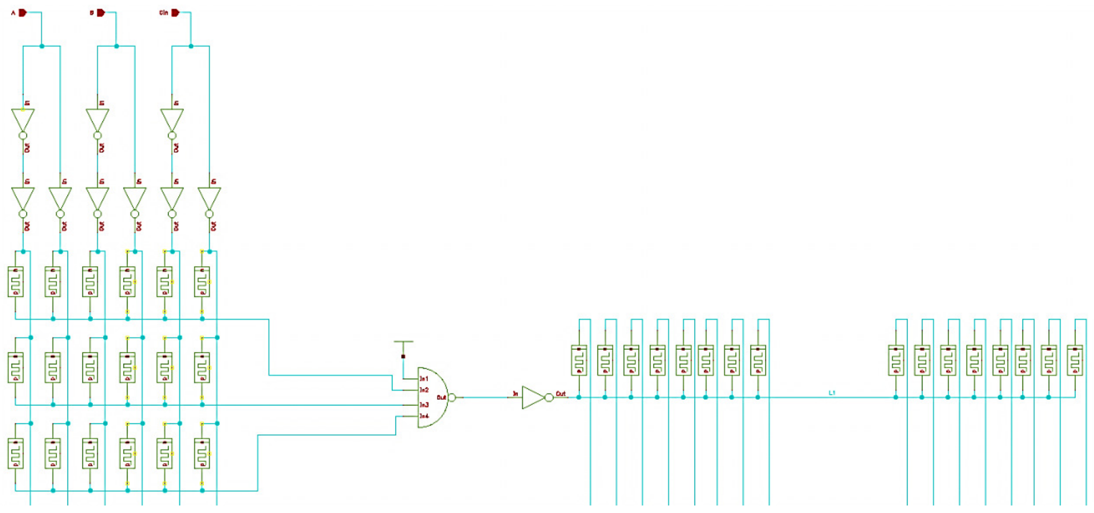


Figure 54 - memristor based 3 input, 2 output PLA array (Zoom in on top part)

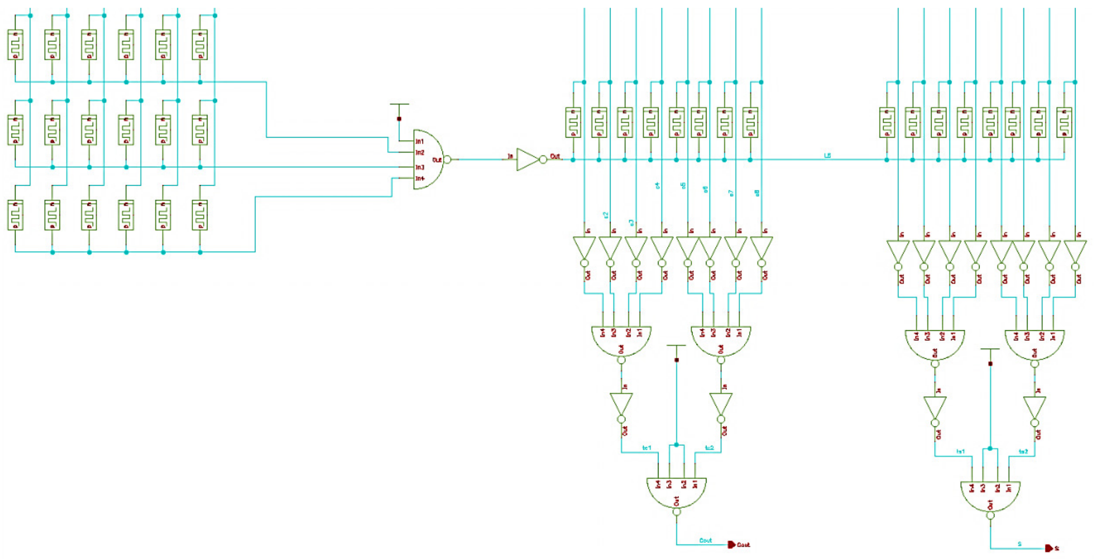


Figure 55 - memristor based 3 input, 2 output PLA array (Zoom in on bottom part)

Circuit design considerations:

- Left part (minterms):
  - The inputs are buffered with a strong NOT gate, this is necessary due to the high fan out of the inputs.

    The dimensions of the used inverter are Wp=1.8µm, Wn=0.84µm.
  - Each input is separated to the input and its complement – total of 6 vertical wires.

    - 3 inputs enable 8 possible minterms, so there are 8 groups of horizontal lines. Each group is composed of 3 lines entering a NAND (followed by NOT – to perform AND).
    - Each line should contain only 1 memristor set to R_{on}, thus creating the minterms.
- Right part (sum of minterms):
    - Each output of an AND gate is connected to vertical wires. On these connections memristor set to R_{on} will connect the minterm to the OR gate at the bottom.
    - It should be noted that 0V voltage passes a memristor as 0V regardless of the memristor state, so the gathering of the minterms should be designed to gather '1's and not '0's. In order to achieve that there is a NOT gate after the minterm creation (left part) and after the minterm selection (right part). These 2 NOT gates could have been saved by selecting different minterms.
    - The bottom part is composed of NANDs and NOTs to create OR-8.

The sum of minterms used for the calculation:

$S = A\bar{B}\overline{C_{in}} + \bar{A}B\overline{C_{in}} + \bar{A}\bar{B}C_{in} + ABC_{in}$

$C_{out} = AB\overline{C_{in}} + A\bar{B}C_{in} + \bar{A}BC_{in} + ABC_{in}$

The last minterm in both functions is identical, so some saving of memristors was possible – yet, for generality, it was decided to proceed with a general 3 input, 2 output array.

The following tables summarize the programming of the arrays.

(The memristors are set to R_{off} by default, only places with R_{on} are indicated.)

Left part (minterms):

Line		A	$\bar{A}$	B	$\bar{B}$	C_{in}	$\overline{C_{in}}$
1		R_{on}					
2	$A\bar{B}\overline{C_{in}}$				R_{on}		
3							R_{on}
4			R_{on}				
5	$\bar{A}B\overline{C_{in}}$			R_{on}			
6							R_{on}
7			R_{on}				
8	$\bar{A}\bar{B}C_{in}$				R_{on}		
9						R_{on}	
10		R_{on}					
11	ABC_{in}			R_{on}			
12						R_{on}	
13		R_{on}					
14	$AB\overline{C_{in}}$			R_{on}			
15							R_{on}
16		R_{on}					
17	$A\bar{B}C_{in}$				R_{on}		
18						R_{on}	
19			R_{on}				
20	$\bar{A}BC_{in}$			R_{on}			
21						R_{on}	
22		R_{on}					
23	ABC_{in}			R_{on}			
24						R_{on}	

Table 8 - PLA array programming (left part)

Right part (sum of minterms):

		1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
1	$A\bar{B}\overline{C_{in}}$									R_{on}							
2	$\bar{A}B\overline{C_{in}}$										R_{on}						
3	$\bar{A}\bar{B}C_{in}$											R_{on}					
4	ABC_{in}												R_{on}				
5	$AB\overline{C_{in}}$	R_{on}															
6	$A\bar{B}C_{in}$		R_{on}														
7	$\bar{A}BC_{in}$			R_{on}													
8	ABC_{in}				R_{on}												
Result		C_{out}								S							

Table 9 - PLA array programming (right part)

The need to use voltages to provide sub-threshold memristor currents requires coordination between the CMOS process and the selected memristor.

Only the memristor with α =1 with threshold yielded the desired results (this is the memristor with highest current threshold). The root cause to the failures led to the CMOS gates. Simulations with ideal logic gates (unrelated to process) yielded good results. This fact led to the conclusion that the CMOS gates used failed to work at the low voltage determined by the memristor current threshold.

Obviously the memristor with α = 1 without threshold didn't work because the memristor states changed during the simulations.

Simulation results (for memristor of α =1 with threshold):

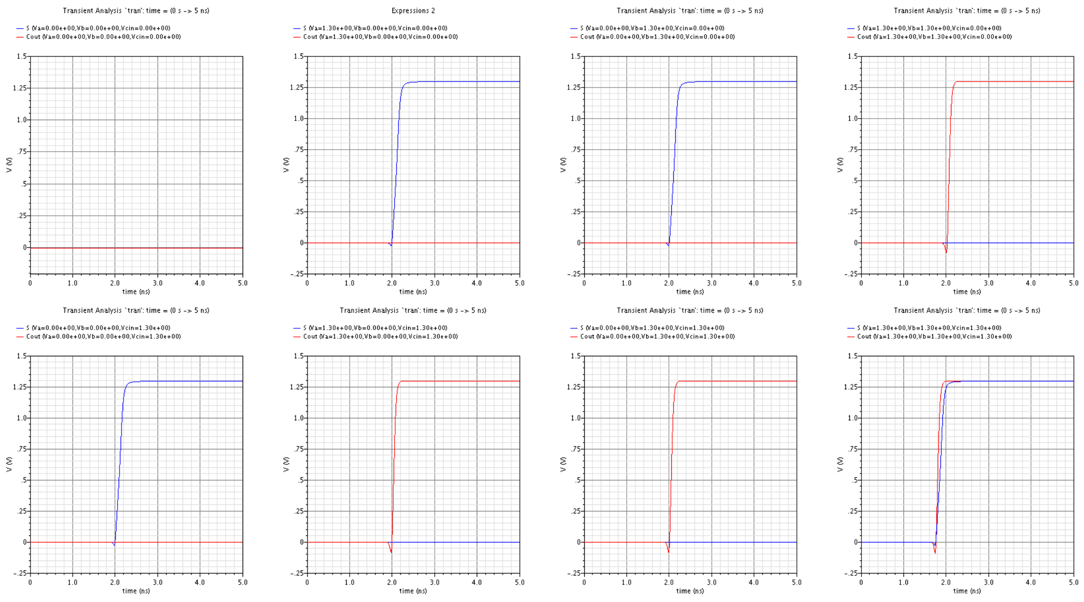


Figure 56 – Memristor Based PLA 1 Bit Full Adder Simulation Results

The PLA array described requires 272 memristors, 186 transistors, and 54 CMOS layer to memristor layer transitions. Calculation time is ~1ns.

7.2. 8 bit Full Adder

The 8 bit Full Adder built is combined of 8 series circuits of 1 bit Full Adder (similar to the Hybrid CMOS logic family).

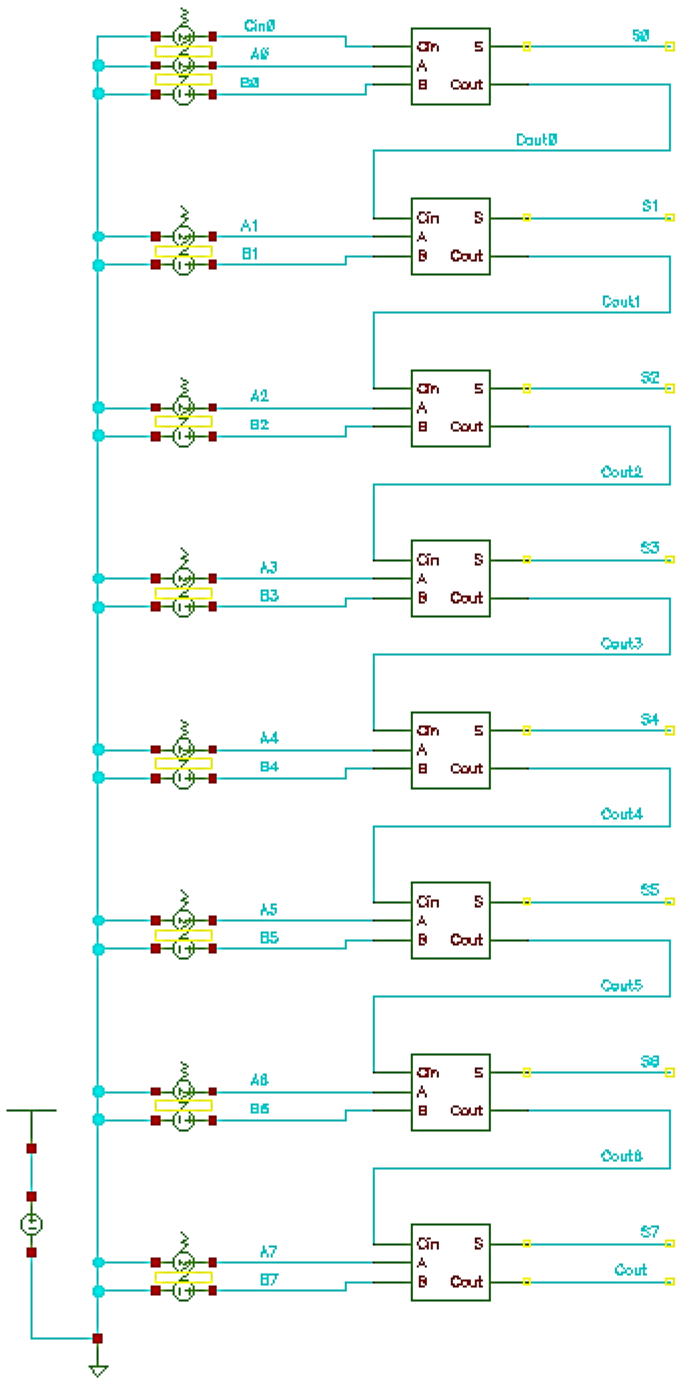
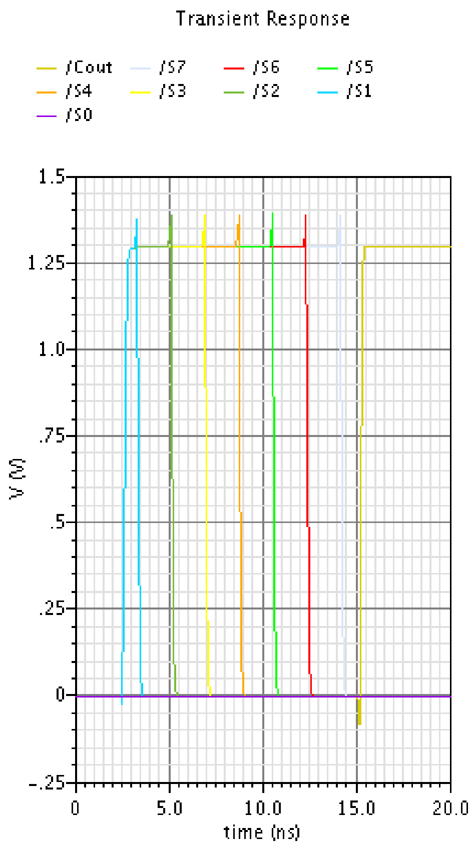


Figure 57 – memristor based PLA array 8 bit Ful Adder

Simulation results with memristors of α = 1 with threshold, for inputs which define the worst propagation of Cout (similar to Hybrid CMOS):



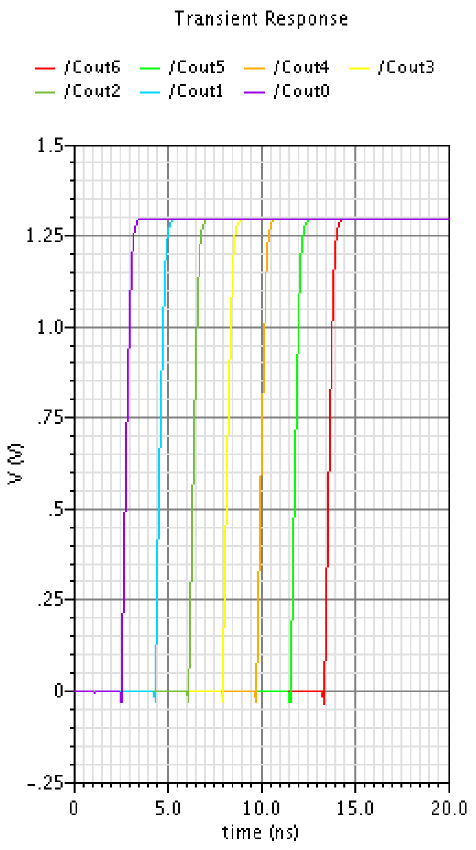


Figure 58 - Memristor based PLA 8 bit Full Adder simulation result

Memristor properties and their effect on designing memristor based PLA:

- Current threshold:
  - Current threshold is necessary.
  - CMOS process voltages, and memristor sub-threshold currents must agree.
  - A memristor with high current threshold is desired.
- Memristor linearity has no effect on logic operation (it will have effect on programming operations, which wasn't analyzed).
- $\frac{R_{on}}{R_{off}}$ ratio will determine the voltage dividers in the circuit, this ratio determines the voltages in the CMOS gates, and so has direct effect on speed. It is desired to have $\frac{R_{on}}{R_{off}} \to 0$.

8. Results Analysis

The results acquired for 8 Bit Adders implemented with all logic families and memristor types are summarized in the following tables (X marks irrelevance and "Can't work" marks memristor types not suitable for use with a certain family). The MAGIC family is not presented due to previously explained reasons.

	Memristor Type		Time [ns]			Maximum Freq. [GHz]
	α	Threshold	Total calculation time	Minimum time interval	Maximum time interval	
IMPLY	1	0	Can't work			
	1	20uA	29.17	0.44	2	0.50
	3	5uA	24.18	0.34	2	0.50
	5	5uA	24.48	0.42	0.71	1.40
	10	10uA	31.76	0.33	0.58	1.72
Hybrid CMOS	1	0	11	x	x	0.09
	1	20uA	5.2	x	x	0.19
	3	5uA	17	x	x	0.06
	5	5uA	40	x	x	0.03
	10	10uA	7.5	x	x	0.13
PLA	1	0	Can't work			
	1	20uA	15	x	x	0.07
	3	5uA	Can't work			
	5	5uA	Can't work			
	10	10uA	Can't work			

Table 10 - Timing Results Summary

	Memristor Type		Area			
	α	Threshold	Number of Memristors	Number of Transistors	Number of Resistors	CMOS -> Memristor layer transitions (VIAs)
IMPLY	1	0	Can't work			
	1	20uA	72	X	8	x
	3	5uA	72	X	8	x
	5	5uA	72	X	8	x
	10	10uA	72	X	8	x
Hybrid CMOS	1	0	144	160	x	80
	1	20uA	144	228	x	96
	3	5uA	144	128	x	80
	5	5uA	144	160	x	80
	10	10uA	144	256	x	96
PLA	1	0	Can't work			
	1	20uA	2176	1488	x	432
	3	5uA	Can't work			
	5	5uA	Can't work			
	10	10uA	Can't work			

Table 11 – Area Results Summary

	Memristor Type		Controller Complexity			
	α	Threshold	Calculation Steps	Voltage Sources	Different Time Intervals	Nodes to Control
IMPLY	1	0	Can't work			
	1	20uA	58	3	2	74
	3	5uA	58	3	2	74
	5	5uA	58	3	2	74
	10	10uA	58	3	2	74
Hybrid CMOS	1	0	1	1	x	x
	1	20uA	1	1	x	x
	3	5uA	1	1	x	x
	5	5uA	1	1	x	x
	10	10uA	1	1	x	x
PLA	1	0	Can't work			
	1	20uA	1	1	x	x
	3	5uA	Can't work			
	5	5uA	Can't work			
	10	10uA	Can't work			

Table 12 - Controller Complexity Results Summary

	Memristor Type		Total dissipation [pJoule]	Power consumed [mWatt]			Static power consumption
	α	Threshold		RMS	Averaged	Maximum	
IMPLY	1	0	Can't work				
	1	20uA	118	3.39	1.28	16.2	no
	3	5uA	71.1	2.53	0.79	14	no
	5	5uA	43.6	1.65	0.484	14	no
	10	10uA	88.4	2.82	1.03	16.5	no
Hybrid CMOS	1	0	11.3	0.311	0.305	1.01	yes
	1	20uA	4590	192	165	531	yes
	3	5uA	379	9.78	9.59	22.8	yes
	5	5uA	1130	28.4	25.9	51.6	yes
	10	10uA	4530	168	151	516	yes
PLA	1	0	Can't work				
	1	20uA	111	6.98	6.96	7.3	no
	3	5uA	Can't work				
	5	5uA	Can't work				
	10	10uA	Can't work				

Table 13 – Power Results Summary

The power results are measured for a "Typical" input that causes the circuit to compute all possible 1 Bit Adder input combinations. The input is:

C	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	
A		*1*	*0*	*0*	*0*	*1*	*1*	*0*	*1*
B		*1*	*1*	*0*	*1*	*1*	*0*	*0*	*0*
S		**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**

Table 14 - Inputs Used for Power Analysis

8.1. Comparison between Logic Families

8.1.1. Speed

- In general, the total calculation time is Hybrid CMOS < PLA < IMPLY.
- Since the IMPLY logic is sequential and divided to a lot of time intervals, it allows the usage of a fast clock. This enables integration of imply in fast systems. The other families conduct the calculation in one cycle. The allowed clock frequencies are IMPLY > Hybrid CMOS > PLA.
- Throughput – IMPLY logic can be easily turned to a pipelined architecture due to its sequential nature (registers are saved due to values stored as resistance). Other families require a design effort and the introduction of registers in order to pipeline.
- IMPLY with parallel architecture is faster than serial architecture.

8.1.2. Area

- Under the assumption that memristors are smaller than transistors and resistors the area is: IMPLY < Hybrid CMOS < PLA.
- IMPLY requires an additional controller logic which is not accounted for in the area calculation.
- IMPLY with parallel architecture requires more area than serial architecture.

8.1.3. Controller Complexity

- The only family that requires an external controller for operation is IMPLY. The controller complexity for this family is dependent on the design architecture.
- PLA requires a controller for programming the circuit before operation. This controller was not analyzed.
- If a circuit was to be implemented using the MAGIC family, it would need a controller of similar complexity to the IMPLY's.

8.1.4. Power

- In general the power used is: IMPLY < PLA < Hybrid CMOS.
- Hybrid CMOS is the only family in which static power is consumed.
- The unexpected difference between PLA and Hybrid CMOS is due to the basic need of PLA for use of low voltages and of Hybrid CMOS for high voltages.
- The Hybrid CMOS circuit with a linear memristor with no threshold consumes the least power. This is due to the fact that for this type of memristor and logic family any working voltage compliant with the CMOS process can be chosen.

8.1.5. Versatility

- PLA and IMPLY (and crossbar based MAGIC) are versatile families and can implement multiple logic functions on a single circuit.
    - Implementation in IMPLY (and MAGIC) is by changing the controller.
    - Implementation in PLA is by reprogramming the array.

8.2. Comparison between Memristor Types

8.2.1. IMPLY

- Speed - dependent mostly on the current threshold, timing decreases as the current threshold decreases.
- Area – The circuit has the same area for all memristor types.
- Power – power is dependent on the voltages Vset and Vcond which are selected according to the memristor type (linearity and current threshold). This complex dependency on the memristor parameters prevents finding a clear trend.

8.2.2. Hybrid CMOS

- Speed - mostly dependent on the working voltages which are affected by the threshold, therefore memristors with high current thresholds are faster.
- Area – as a result of the need for more buffers when the current threshold is high, the area increases with the current threshold.
- Power – is dominated by the working voltage, which is selected according to the current threshold.
    - Power can be reduced significantly at the cost of adding more buffering with CMOS gates, this buffering will reduce both static power and dynamic power (more buffering will reduce the fan out and thus enables reducing the working voltage).
    - Static power can be eliminated completely by adding CMOS buffers after each Hybrid CMOS gate.
- A memristor without current threshold is ideal for this family, it enables almost limitless control of the tradeoff between speed, area and power.

8.2.3. PLA

- This family was successful only with linear memristors with current threshold (the memristor with the highest current threshold checked). The failures were due to incompatibility of working voltage required by the memristors and the CMOS gates.

- For performing logic operations memristors with high current threshold are required, and the linearity has no effect on performance.
- The overall performance is highly affected by the $\frac{R_{on}}{R_{off}}$ ratio. A ratio of 0 is preferred.
- Analysis of programming the logic wasn’t conducted, yet it is expected that the memristor linearity will affect speed and power.

9. Conclusions

This project examined 4 types of ways to implement logic with memristors. This field is in its youth, and many open questions are present. The target of the project was to examine the feasibility of logic with memristors and to compare the various options.

During the project a novel logic was suggested – crossbar based MAGIC gates. It encountered difficulties and is yet operational, yet it seems to hold promise.

The main method considered these days is IMPLY logic. Some of the methodology used in the IMPLY part of this project can be generalized to broader uses (such as the standard Boolean realization with IMPLY, serial vs. parallel approaches and tradeoffs).

The major outcomes from the project are:

- Memristor type has little effect compared to the logic family chosen on most parameters examined.
- IMPLY logic was proven best in terms of area and versatility. It seems to be also the most robust of the examined families.
- The Hybrid CMOS was proven fastest, but is not power efficient. It also requires integration with CMOS gates, which makes the design process sensitive to the working voltages.
- The PLA is the most versatile family, which results in the highest area needed.
- MAGIC family didn't yield the desired results so far.

A visual graph to summarize correlation between memristor types to logic families is shown:

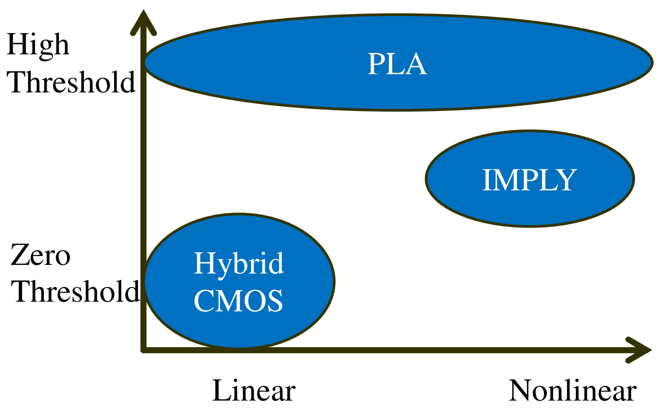


Figure 59 - Memristor and Logic Family Matching

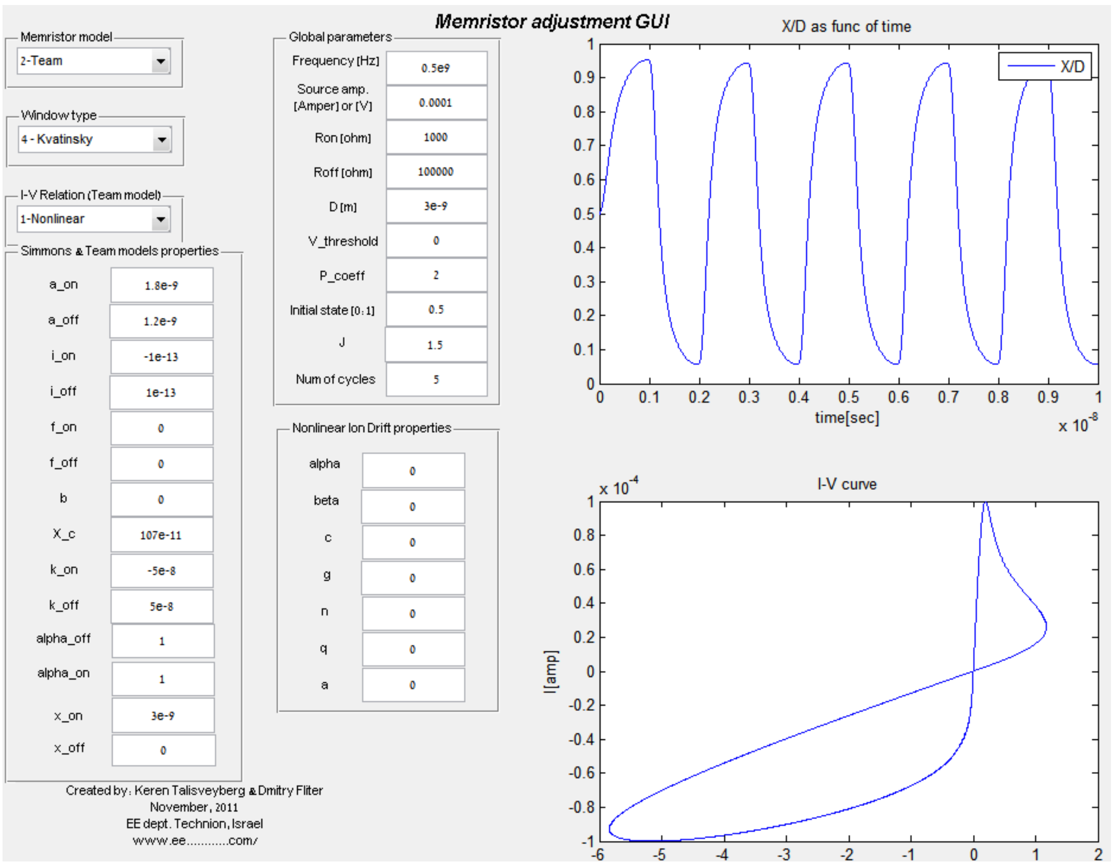
Appendix A – List of TEAM Model Parameters

Fitting of the models was done with the Matlab tool described in [7].

- Linear memristor without threshold

α	1
Kon	-5e-8
Koff	5e-8
Ion	-1e-13
Ioff	1e-13
a_{on}	1.8n
a_{off}	1.2n
Ron	1k
Roff	100k
dt	1e-13

Table 15 - Linear memristor without threshold parameters


Memristor adjustment GUI

Memristor model: 2-Team

Window type: 4 - Kvatinsky

I-V Relation (Team model): 1-Nonlinear

Simmons & Team models properties

a_on	1.8e-9
a_off	1.2e-9
i_on	-1e-13
i_off	1e-13
f_on	0
f_off	0
b	0
X_c	107e-11
k_on	-5e-8
k_off	5e-8
alpha_off	1
alpha_on	1
x_on	3e-9
x_off	0

Global parameters

Frequency [Hz]	0.5e9
Source amp. [Amper] or [V]	0.0001
Ron [ohm]	1000
Roff [ohm]	100000
D [m]	3e-9
V_threshold	0
P_coeff	2
Initial state [0:1]	0.5
J	1.5
Num of cycles	5

Nonlinear Ion Drift properties

alpha	0
beta	0
c	0
g	0
n	0
q	0
a	0

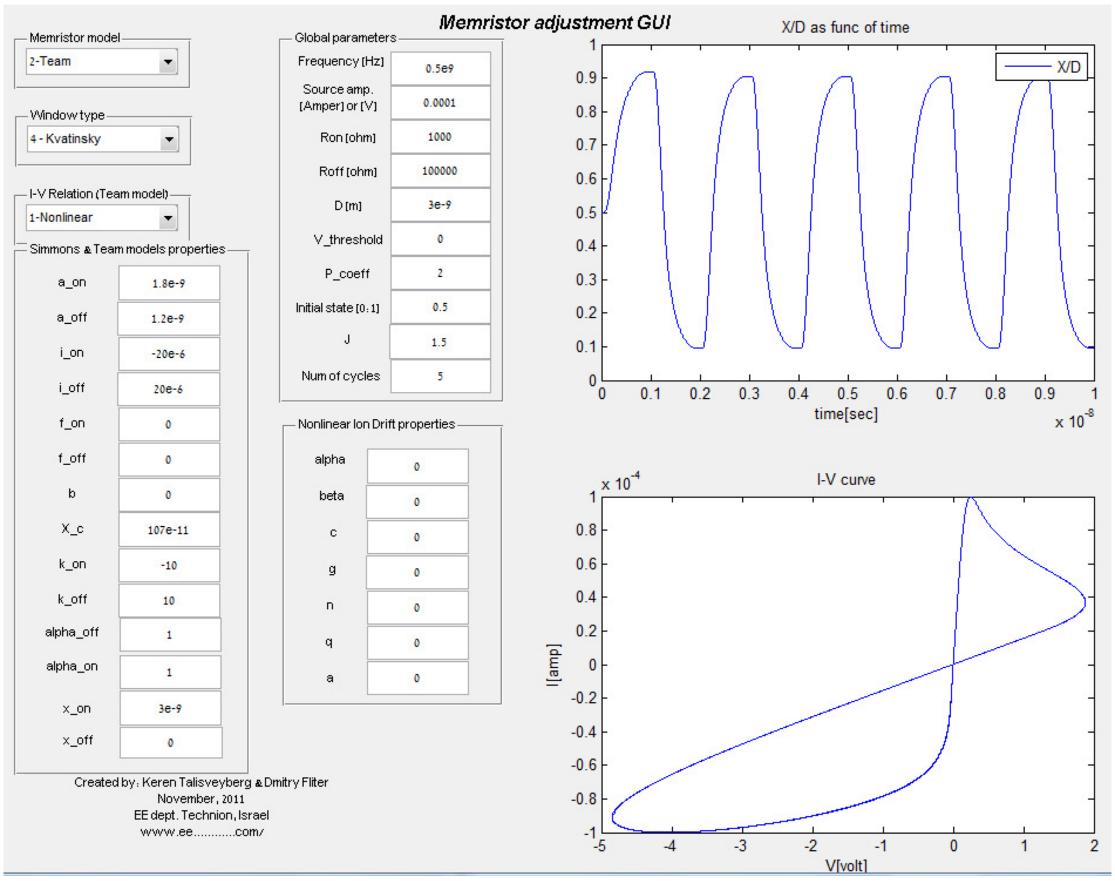
Created by: Keren Talisveyberg & Dmitry Filter
November, 2011
EE dept. Technion, Israel
www.ee..........com/

Figure 60 -Linear memristor without threshold response

- Linear memristor with threshold

α	1
Kon	-10
Koff	10
Ion	-20u
Ioff	20u
a_{on}	1.8n
a_{off}	1.2n
Ron	1k
Roff	100k
dt	1e-13

Table 16 - Linear memristor with threshold parameters


Memristor adjustment GUI

Memristor model: 2-Team

Window type: 4 - Kvatinsky

I-V Relation (Team model): 1-Nonlinear

Simmons & Team models properties

a_on	1.8e-9
a_off	1.2e-9
i_on	-20e-6
i_off	20e-6
f_on	0
f_off	0
b	0
X_c	107e-11
k_on	-10
k_off	10
alpha_off	1
alpha_on	1
x_on	3e-9
x_off	0

Global parameters

Frequency [Hz]	0.5e9
Source amp. [Amper] or [V]	0.0001
Ron [ohm]	1000
Roff [ohm]	100000
D [m]	3e-9
V_threshold	0
P_coeff	2
Initial state [0:1]	0.5
J	1.5
Num of cycles	5

Nonlinear Ion Drift properties

alpha	0
beta	0
c	0
g	0
n	0
q	0
a	0

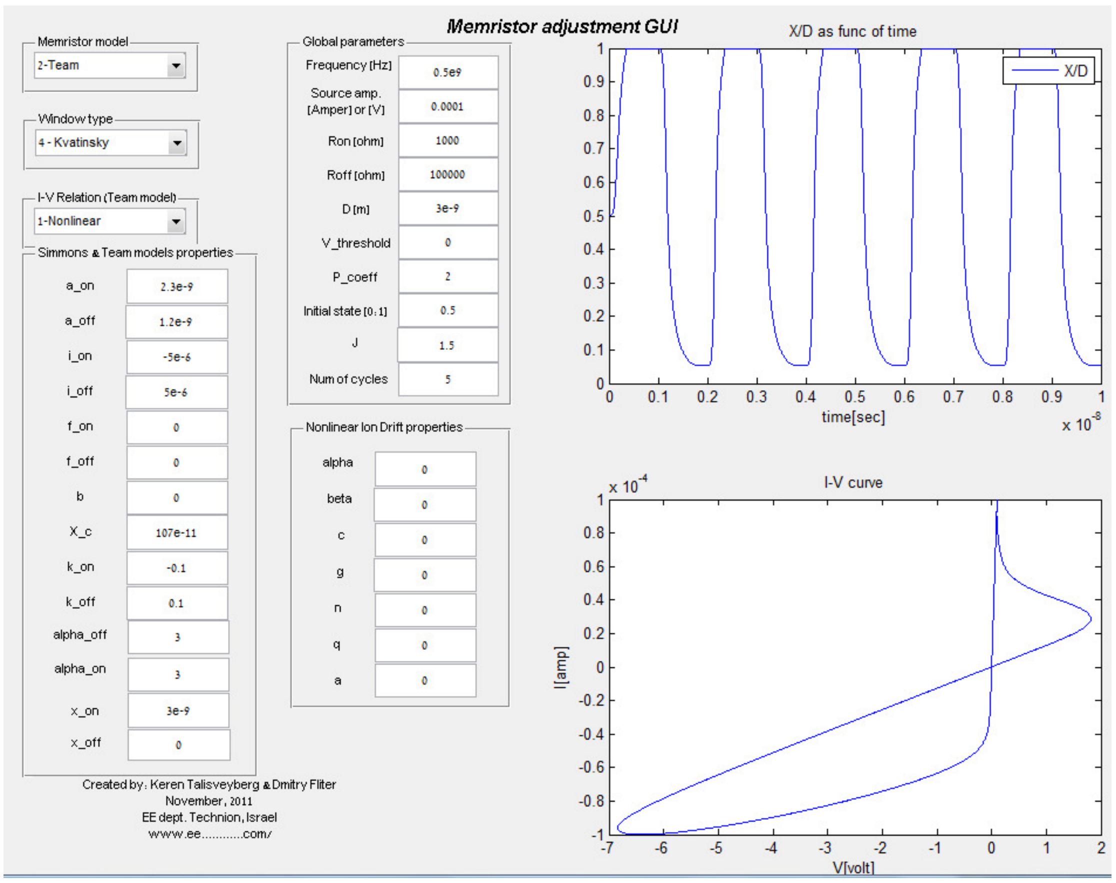
Created by: Keren Talisveyberg & Dmitry Fliter
November, 2011
EE dept. Technion, Israel
www.ee...........com/

Figure 61 – Linear memristor with threshold response

- Nonlinear memristor with threshold (α = 3)

α	3
Kon	-0.1
Koff	0.1
Ion	-5u
Ioff	5u
a_{on}	2.3n
a_{off}	1.2n
Ron	1k
Roff	100k
dt	1e-13

Table 17 - Nonlinear memristor with threshold (α = 3)



Memristor adjustment GUI

Memristor model: 2-Team

Window type: 4 - Kvatinsky

I-V Relation (Team model): 1-Nonlinear

Simmons & Team models properties

a_on	2.3e-9
a_off	1.2e-9
i_on	-5e-6
i_off	5e-6
f_on	0
f_off	0
b	0
X_c	107e-11
k_on	-0.1
k_off	0.1
alpha_off	3
alpha_on	3
x_on	3e-9
x_off	0

Global parameters

Frequency [Hz]	0.5e9
Source amp. [Amper] or [V]	0.0001
Ron [ohm]	1000
Roff [ohm]	100000
D [m]	3e-9
V_threshold	0
P_coeff	2
Initial state [0:1]	0.5
J	1.5
Num of cycles	5

Nonlinear Ion Drift properties

alpha	0
beta	0
c	0
g	0
n	0
q	0
a	0

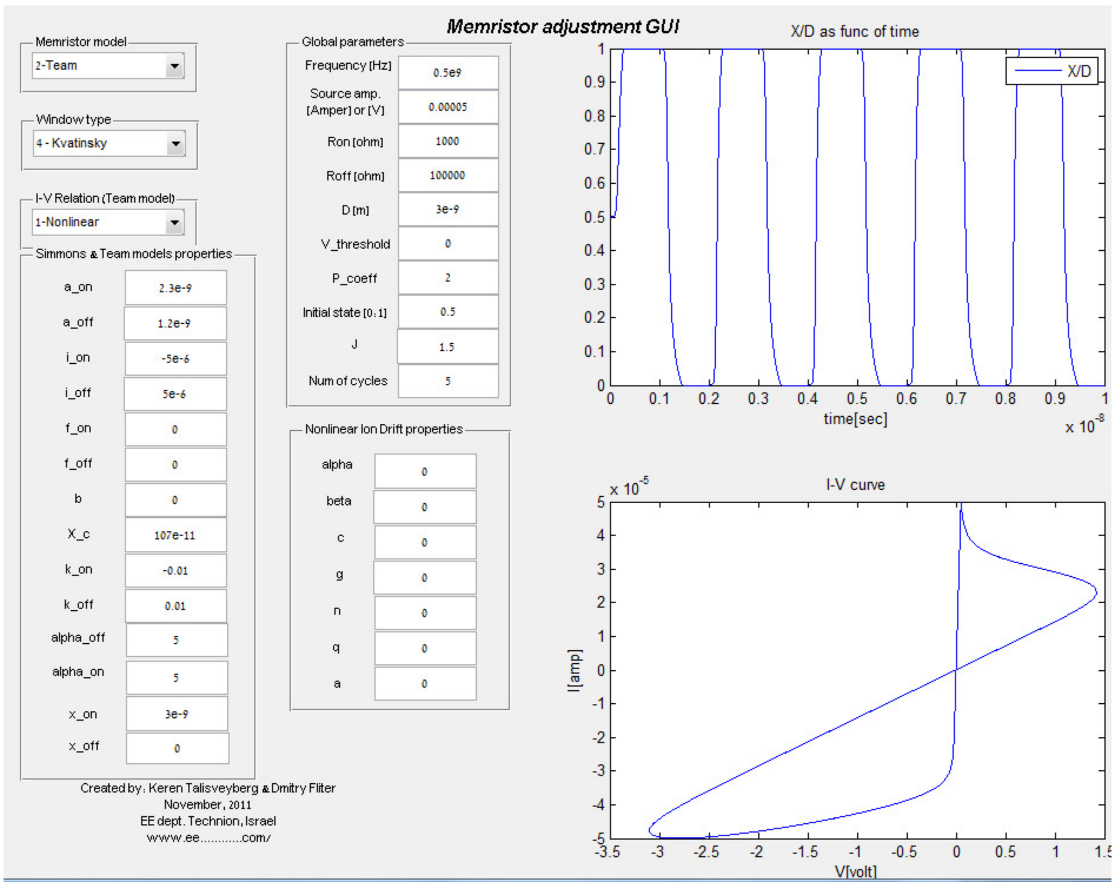
Created by: Keren Talisveyberg & Dmitry Fliter
November, 2011
EE dept. Technion, Israel
www.ee...........com/

Figure 62 - Nonlinear memristor with threshold (α = 3)

- Nonlinear memristor with threshold (α = 5)

α	5
Kon	-0.01
Koff	0.01
Ion	-5u
Ioff	5u
a_{on}	2.3n
a_{off}	1.2n
Ron	1k
Roff	100k
dt	1e-13

Table 18 - Nonlinear memristor with threshold (α = 5)


Memristor adjustment GUI

Memristor model: 2-Team

Window type: 4 - Kvatinsky

I-V Relation (Team model): 1-Nonlinear

Simmons & Team models properties

a_on	2.3e-9
a_off	1.2e-9
i_on	-5e-6
i_off	5e-6
f_on	0
f_off	0
b	0
X_c	107e-11
k_on	-0.01
k_off	0.01
alpha_off	5
alpha_on	5
x_on	3e-9
x_off	0

Global parameters

Frequency [Hz]	0.5e9
Source amp. [Amper] or [V]	0.00005
Ron [ohm]	1000
Roff [ohm]	100000
D [m]	3e-9
V_threshold	0
P_coeff	2
Initial state [0:1]	0.5
J	1.5
Num of cycles	5

Nonlinear Ion Drift properties

alpha	0
beta	0
c	0
g	0
n	0
q	0
a	0

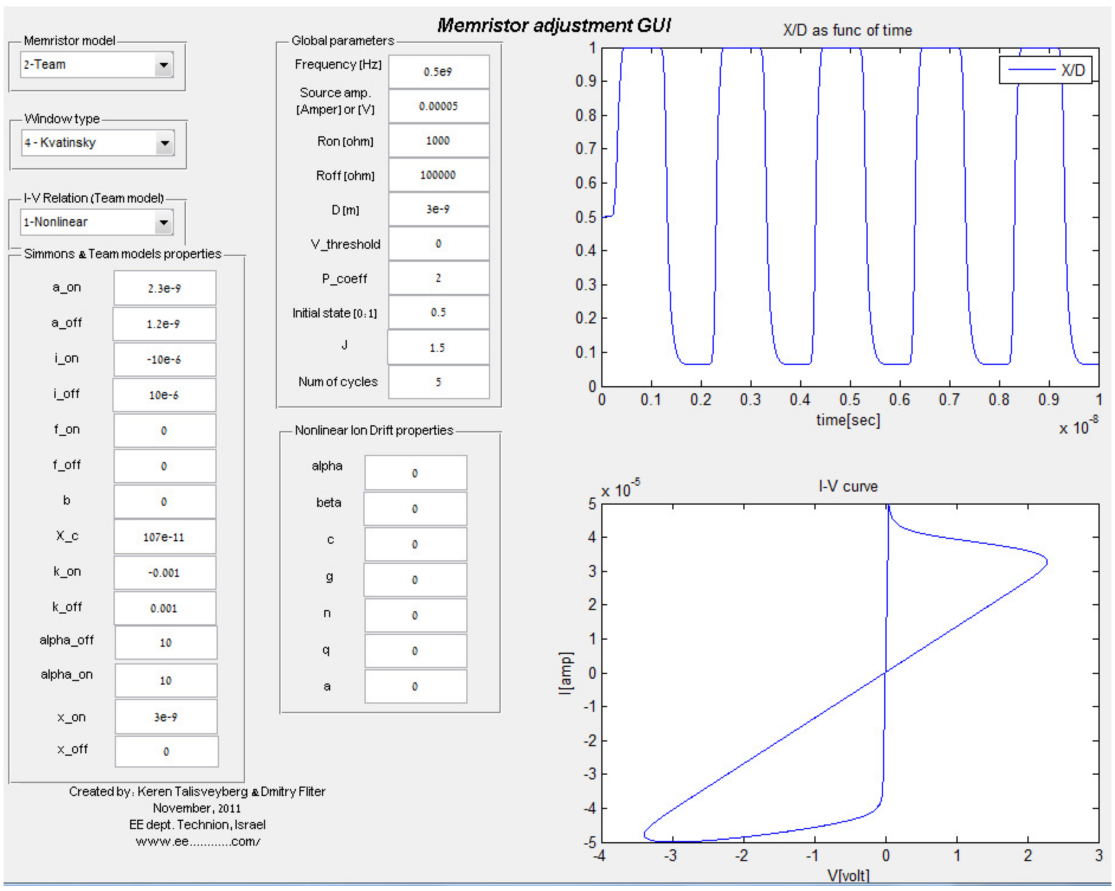
Created by: Keren Talisveyberg & Dmitry Fliter
November, 2011
EE dept. Technion, Israel
www.ee...........com/

Figure 63 - Nonlinear memristor with threshold (α = 5)

- Nonlinear memristor with threshold (α = 10)

α	10
Kon	-0.001
Koff	0.001
Ion	-10u
Ioff	10u
a_{on}	2.3n
a_{off}	1.2n
Ron	1k
Roff	100k
dt	1e-13

Table 19 - Nonlinear memristor with threshold (α = 10)



Memristor adjustment GUI

Memristor model: 2-Team

Window type: 4 - Kvatinsky

I-V Relation (Team model): 1-Nonlinear

Simmons & Team models properties

a_on	2.3e-9
a_off	1.2e-9
i_on	-10e-6
i_off	10e-6
f_on	0
f_off	0
b	0
X_c	107e-11
k_on	-0.001
k_off	0.001
alpha_off	10
alpha_on	10
x_on	3e-9
x_off	0

Global parameters

Frequency [Hz]	0.5e9
Source amp. [Amper] or [V]	0.00005
Ron [ohm]	1000
Roff [ohm]	100000
D [m]	3e-9
V_threshold	0
P_coeff	2
Initial state [0:1]	0.5
J	1.5
Num of cycles	5

Nonlinear Ion Drift properties

alpha	0
beta	0
c	0
g	0
n	0
q	0
a	0

Created by: Keren Talisveyberg & Dmitry Fliter
November, 2011
EE dept. Technion, Israel
www.ee...........com/

Figure 64 - Nonlinear memristor with threshold (α = 10)

Appendix B – Complete analysis of addition to MAGIC

NOT Logic Gate

A NOT logic gate consists of an input memristor “in”, an output memristor “S” and a controlled voltage source V_{in}. The memristors are connected in series and with opposite polarity. A schematic of a NOT logic gate is shown in the figure below. Prior to the calculation the output memristor S must be set to logic '1'.

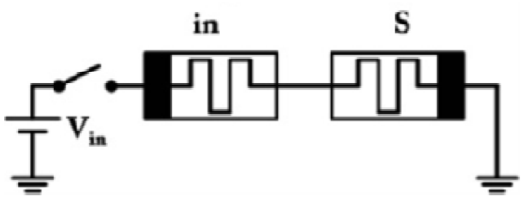


Figure 65 - MAGIC NOT gate

In case that the input is set to '0', the total resistance of the circuit is $R_{off} + R_{on}$, and this must cause the current to be below the current threshold i_{off}. In case that the input is '1', the total resistance is $2R_{on}$, and the current must be above the current threshold i_{off} in order to achieve a change in the output. Combining the 2 conditions gives: $2\ i_{off} R_{on} < V_{in} < i_{off}(R_{off} + R_{on})$.

A memristor might be asymmetric, having different response when switching to on or to off. This asymmetry may cause the current threshold to be different i.e. $i_{off} \neq i_{on}$. In that case, a stricter criterion is enforced to keep the resistance of the input memristor unchanged. In case the input is set to ‘0’, the current should be below the minimum of the current thresholds, i.e. $minimum\{|i_{on}|, |i_{off}|\}$. In case the input memristor is set to ‘1’, the current direction attempts to decrease the resistance, thus not changing it. Combining this criterion to the above equation yields: $2\ i_{off} R_{on} < V_{in} < minimum\{|i_{off}|, |i_{on}|\} \cdot (R_{off} + R_{on})$.

Another aspect to be considered is the maximum change in the output memristor (switching from R_{on} to R_{off}). In this case, the input memristor is set to ‘0’, and as described above remains at ‘0’ during the switch process. The output memristor resistance is initially R_{on}, and increases until the current reaches the threshold i_{off}.

The output resistance at this time is $R_S = \frac{V_{in}}{i_{off}} - R_{on}$. Thus in order to get as close as possible to R_{off}, V_{in} must be as large as possible under the constraint. If $i_{off} = i_{on}$ and V_{in} is set at the upper limit, then $R_{S_{max,\ on\to off}} \approx R_{off}$.

NOR Logic gate

A two input NOR logic gate consists of 2 input memristors in1, in2 connected in parallel, an output memristor with opposite polarity to the input memristors, and a controlled voltage source V_{in}. A schematic of a 2 input NOR gate is shown in the figure below. Prior to the calculation the output memristor must be set to '1'.

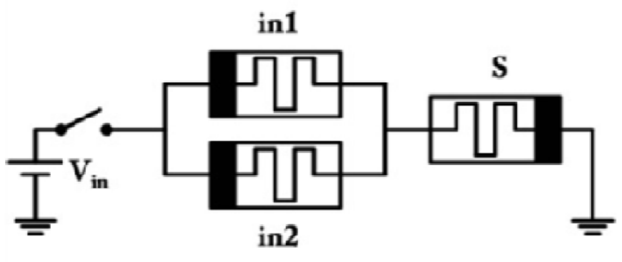


Figure 66 - MAGIC NOR gate

To satisfy NOR truth table, the value of the output memristor should remain at '1' only when the input memristors are set to '0'. In that case the effective resistance of the circuit is $\frac{R_{off}}{2} + R_{on}$ and the current should be smaller than the threshold i_{off}. Other combinations of the input memristors should change the output memristor to '0'. The stricter condition is when only one of the input memristors is set to '0'. In that case the effective resistance is $R_{off}||R_{on} + R_{on}$ and the current should be above the current threshold. Combining the conditions gives:

$$i_{off}(R_{off}||R_{on} + R_{on}) < V_{in} < i_{off}\left(\frac{R_{off}}{2} + R_{on}\right)$$

For a multi input gate with χ inputs, the above condition becomes:

$$i_{off}\left(\frac{R_{off}}{\chi-1}||R_{on} + R_{on}\right) < V_{in} < i_{off}\left(\frac{R_{off}}{\chi} + R_{on}\right)$$

The configuration connecting the memristors guarantees that the values of the input memristors remain intact. In case all input memristors are set to '0', the conditions above guarantees the current is below the threshold. In case all input memristors are set to '1', they are at the minimal resistance R_{on}. Due to their polarity, current passing through them can only decrease their resistance, and thus their logical value remains the same. The complement case is when some of the input memristors are set to '1', and the others to '0'. Due to the ratio between $R_{on}$to R_{off} most of the current will pass thorough the memristors set to R_{on}, and similarly to the previous case their logical value remains the same. The part of the current that passes through the memristors which are set to R_{off} is below the threshold.

If a memristor is asymmetric ($i_{off} \neq i_{on}$) , a stricter criterion is needed in order to keep the input memristor unchanged. This will only affect the case in which all input memristors are set to '0'. In that case the current should be low enough not to change the input memristors and the output memristor, i.e.
$i < minimum\{|i_{off}|, 2|i_{on}|\}$. Thus, the above conditions become:
$i_{off}(R_{off}||R_{on} + R_{on}) < V_{in} < minimum\{|i_{off}|, 2|i_{on}|\}\left(\frac{R_{off}}{2} + R_{on}\right)$
$i_{off}\left(\frac{R_{off}}{\chi-1}||R_{on} + R_{on}\right) < V_{in} < minimum\{|i_{off}|, 2|i_{on}|\}\left(\frac{R_{off}}{\chi} + R_{on}\right)$

To analyze effects on noise margin the maximum change in the resistance of the output memristor should be considered. For example, a 2 input NOR gate. In case one of the input memristors is set to '1', the maximum resistance of the output memristor is given by: $R_{s_{max}} = \frac{V_{in}}{i_{off}} - R_{off}||R_{on}$. So, V_{in} should be set to the maximum possible value, in case of a symmetric memristor the maximum resistance of the output memristor is $R_{s_{max}} = \frac{R_{off}}{2} + R_{on} - R_{off}||R_{on} \approx \frac{R_{off}}{2}$. Similar calculation for the case in which both input memristors set to '1' gives: $R_{s_{max}} = \frac{R_{off}+R_{on}}{2} \approx \frac{R_{off}}{2}$. Since $R_{on} \ll R_{off}$ this effect has small impact on 1 gate, but it effects the maximum number of gates connected in serial before amplifying the signal.

NAND Logic gate

A NAND logic gate consists of 2 input memristor and an output memristor. The memristors are connected in serial, when the polarity of the output memristor is opposite to the polarity of the input memristors. A schematic of a two input NAND logic gate is shown in the figure below. The output memristor S must be set to logic '1' prior to the calculation.

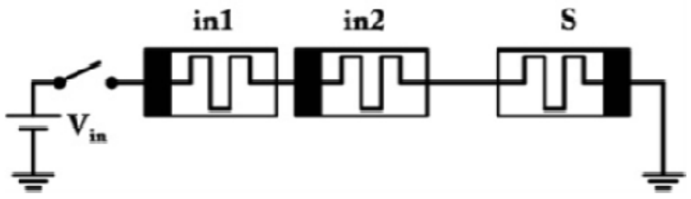


Figure 67 - MAGIC NAND gate

In order to satisfy the NANDs gate truth table, a change in the output memristor is expected only in case that both input memristors are set to '1'. In that case the effective resistance of the 3 memristors is $3R_{on}$, and to achieve the change the current through S must be above the threshold i_{off}. In case that at least one of the input memristors is set to '0' no change is expected, and the current must be below the threshold. In that case the effective resistance of the 3 memristors is $R_{off} + 2R_{on}$. Combining the 2 requirements yields:

$$3\, i_{off}R_{on} < V_{in} < i_{off}\,(R_{off} + 2R_{on})$$

For a multi input gate with χ input memristors the equivalent requirement is:

$$(1+\chi)i_{off}R_{on} < V_{in} < i_{off}\left((\chi-1)R_{off} + 2R_{on}\right)$$

The configuration in which the memristors are connected guarantees that the value of the input memristors remains intact. The only case in which the current is above the switch threshold is when all the input memristors are set to '1'. Due to their polarity with respect to V_{in}, current passing through them can only decrease their resistance, but since it is already R_{on} it remains as is. In other combinations of the input memristors the current is below the threshold thus no change is expected.

If the memristor is asymmetric, a stricter criterion is needed in order to keep the input memristor values unchanged. In case part of the input memristors are set to

'0', the current must be smaller than the minimum of the current thresholds ($minimum\{|i_{on}|, |i_{off}|\}$). Thus, the conditions become:

$$3i_{off}R_{on} < V_{in} < minimum\{|i_{on}|, |i_{off}|\}\left(R_{off} + 2R_{on}\right)$$

$$(1+\chi)i_{off}R_{on} < V_{in} < minimum\{|i_{on}|, |i_{off}|\}\left((\chi - 1)R_{off} + 2R_{on}\right)$$

Unlike the NOR gate, when the NAND gate switches from R_{on} to R_{off} the switch is maximal. The resistance of the output memristor when the current reaches the threshold is: $R_s = \frac{V_{in}}{i_{off}} - \chi\, R_{on}$. When using the largest V_{in} possible, and for a symmetric memristor the maximum value is $R_{S_{max}} \approx R_{off}$.

Comparing between NOR and NAND gates

Since NOR and NAND are both a complete set, it is reasonable to compare between the 2 methods. The table summarizes the differences:

		NOR	NAND
V_{in} – for 2 input and symmetric memristor	Maximum	$i_{off}\left(\frac{R_{off}}{2}+R_{on}\right)$	$i_{off}\,(R_{off}+2R_{on})$
	Minimum	$i_{off}(R_{off}\|\|R_{on}+R_{on})$	$3\ i_{off}R_{on}$
Example for characteristic values: $i_{on}=i_{off}=10\mu A$ $R_{off}=100k\Omega$ $R_{on}=1k\Omega$		$0.019V<V_{in}<0.51V$	$0.03V<V_{in}<1.02V$
Initial set of R_S		'1' = R_{on}	'1' = R_{on}
Probability to switch from initial set value to new value		3/4	1/4
Configuration		Very modular, can be part of crossbar. The output can easily be an input for another gate. Same memristor can be used to calculate NOT or NOR (depends on applied voltage) See "Implementation of NOR and NOT Logic Gates in Crossbar Form" later.	
Maximum switch from R_{on} to R_{off} (effect on noise margin)		The maximum resistance of the output memristor is $\frac{R_{off}}{2}$	The maximum resistance of the output memristor is R_{off}

Table 20 - MAGIC NOR and NAND Comparison

Implementation of NOR and NOT Logic Gates in Cross Bar Form

One of the benefits of using NOR, and NOT gates as presented above, is the ability to implement the gates in a multifunctional crossbar form. A crossbar is presented in the figure below. It should be noted that the polarities of all memristors with respect to the driver is identical, as it is in the NOT and NOR gates.

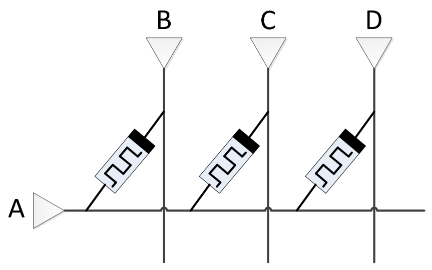


Figure 68 - Cross bar implementation of gate

In order to implement the gates as described, a voltage driver is needed. The driver should be able to drive several voltage levels: Ground, V_{set} (to set a memristor to '1'), V_{NOR} (V_{in} for NOR calculation), V_{NOT} (V_{in} for NOT calculation) and to float. In the next paragraph it is assumed that a driver is floating unless stated otherwise.

Implementing NOR requires these steps: Initialize the output memristor by applying $V_{set}$to driver D and ground to driver A. Calculate the result by applying V_{NOR} to drivers B and C, and ground to driver D.

Implementing NOT requires these steps: Initialize the output memristor by applying $V_{set}$to driver D and ground to driver A. Calculate the result by applying V_{NOT} to driver B, and ground to driver D.

The advantages of using this crossbar form with NOR and/or NOT gates are:

- Any memristor on the crossbar can be part of any calculation.
- The output of any calculation can easily be used as an input for the next one.
- NOT and NOR can be done on the same set of memristors – no structural change.
- Crossbar can be used to compute multiple functions, the only change is in the controller.

- NOR constitutes a complete set of operations, therefore it is possible to implement any logic function on a crossbar large enough.

The disadvantage of using this form is the requirement of different voltages to compute different functions (since NOR is a complete set of operations, this is not necessarily a disadvantage).

Noise Margin Problem of MAGIC NOT and NOR Gates

A NOT gate is considered as an example.

The output memristor S, is assumed to be fully initialized to R_{on}.

Noise margins are defined as follows:

'0' can be $R_{off}(1-N_0) \div R_{off}$

'1' can be $R_{on} \div R_{on}(1+N_1)$

The conditions on V_{in} for switching now become:

$$\frac{V_m}{R_{off}(1-N_0)+R_{on}} < i_{th} < \frac{V_m}{R_{on}(1+N_1)+R_{on}} .$$

This gives: $R_{on}(2+N_1)i_{ith} < V_m < \left(R_{on} + R_{off}(1-N_0)\right)i_{th}$.

For maximal switch, the maximal voltage is needed:

$V_m = \alpha i_{th}\left(R_{on} + R_{off}(1-N_0)\right)$, Where $\alpha \to 1^-$.

Suppose the input is R_{on} . Then:

$$i = \frac{V_m}{R_{on}+R_{on}} = \alpha i_{th}\frac{R_{on}+R_{off}(1-N_0)}{R_{on}+R_{on}} > i_{th} \text{ (for reasonable } N_0, \alpha)$$

But the maximum switch of S is to:

$$i_{th} = i = \frac{V_m}{R_{on}+R_s} = \alpha i_{th}\frac{R_{on}+R_{off}(1-N_0)}{R_{on}+R_s}$$

$$\to R_s = \alpha\left(R_{on} + R_{off}(1-N_0)\right) - R_{on} \to R_{off}(1-N_0)$$

So S never reaches R_{off}.

Same concept of problem rises in NOR when trying to add noise margin. NOR gate can maximum switch is to $\frac{1}{2}R_{off}$. So noise margin problem is critical.

Appendix C

C1. IMPLY 1 Bit Full Adder Sequence

The computation sequence used for 1 bit full adder using IMPLY:

Step	Goal	Operation	Input memristors		In / Out memristor	Work memristors		Output memristor
			A	B	C	M1	M2	S
0	start		A	B	Cin	?	?	?
1	Duplicate A to M2 (Via S)	False(S)						0
2		False(M2)					0	
3		A->S						A'
4		S->M2					A	
5	Duplicate B to M1 (Via S)	False(S)						0
6		False(M1)				0		
7		B->S						B'
8		S->M1				B		
9	A XOR B	B->M2					B->A	
10		A->M1				A->B		
11		False(S)						0
12		M2->S						(B->A)->'0'
13		M1->S						A XOR B
14	Duplicate S = (A XOR B) to M1 via M2	False (M2)					0	
15		False(M1)				0		
16		S->M2					(A XOR B)'	
17		M2->M1				A XOR B		
18	Calculate part of Cout	C->M2					Cin->((A XOR B)->'0')	
19	Back to S calculation	C->M1				C->(A XOR B)		
20		S->C			(A XOR B)->C			
21		False(S)						0
22		M1->S						(C->(A XOR B))->'0'
23		C->S						S
24	Finish Cout calculation	False (C)			0			
25		M2->C			(Cin->((A XOR B) ->'0'))->'0'			
26		False(M1)				0		
27		B->M1				B->0		
28		A->M1				A->(B->0)		
29		M1->C			C			

Table 21 - IMPLY 1 Bit Full Adder Sequence

C2. IMPLY Parallel 8 Bit Full Adder Sequence

The computation sequence used for 8 bit full adder using IMPLY:

step	target	operation	Input memristors		In	Work memristors				Output memristors	
			A	B	C0	M1	M2	M3	T0	S	C1
0	start		A	B	? (for all except bit 0)	?	?	?	?	?	?
1	Duplicate A to M2 AND B to M1	False(7,M1,M2,M3,S,C0, T0,C1)			0 (for all except bit 0)	0	0	0	0	0	0
2		A->T0							A->0		
3		T0->M2					A				
4		B->M3						B->0			
5		M3->M1				B					
6	Part of C calc	A->M3						A->(B->0)			
7	A XOR B	B->M2					B->A				
8		A->M1				A->B					
9		M2->S								(B->A)->0	
10		M1->S								A XOR B	
11	Duplicate S = (A XOR B) to M1 via M2	False (3,M1,M2,T0)				0	0		0		
12		S->M2					(AXORB)->0				
13		M2->M1				AXOR B					
					Cin						
14	Calculate part of Cout	C0->M2					Cin->((AXORB)->0)				
15		M2->C1									(Cin->((AXORB)->0))->0
16		M3->C1									Cout
17	Copy Cout to next stage Cin	IMPBL(C1i,T0i+1)							Cout->0 (from prev)		
18		IMPBL(T0i+1,C0i+1) = IMPLY(T0,C0)LINES(i+1)			Cin						
19	Back to S calculation	C0->M1				C->(AXORB)					
20		S->C0			(AXORB)->C						
21		False(S)								0	
22		M1->S								(C->(AXORB))->0	
23		C0->S								S	

Table 22 - IMPLY 8 Bit Full Adder Sequence

- Steps 1-13 are done in parallel on all lines.
- Steps 14-18 calculate carry of each bit independently, and are repeated for all bit lines.
- Steps 19-23 are then done also in parallel to complete the computation.

Appendix D

D1. IMPLY 1 Bit Full Adder Voltage Control Files MATLAB Code

The MATLAB function that produces the VPWLF files:

```
function VPWL(file,mems,Th,Tr,Timp,Trst,Tf,Td,V0,V1,Vcond,Vset,Vreset)
%% Open and read memristor file, Create pwl files for memristors
memristorfile = fopen(mems);
m=0;
while(~feof(memristorfile))
    memristors(m+1,:)=fgetl(memristorfile);
    vfid(m+1)=fopen(['outputs\',memristors(m+1,:),'v.txt'],'w+');
    sfid(m+1)=fopen(['outputs\',memristors(m+1,:),'s.txt'],'w+');
    fprintf(vfid(m+1),'0 %d\n',V0);
    fprintf(sfid(m+1),'0 %d\n',V0);
    m=m+1;
end
%% Open and read commands file
commandfile = fopen(file);
l=0;
while(~feof(commandfile))
    commands{l+1}=fgetl(commandfile);
    l=l+1;
end
%% Write times and voltages into pwl files
T=0;
for k=1:l
    if findstr(commands{k}(:)','IMPLY')
       fprintf(vfid(strmatch(commands{k}(7:9),memristors)), '%d %d\n%d %d\n%d %d\n%d %d\n', ...
                    T+Th,V0,T+Th+Tr,Vcond,T+Th+Tr+Timp,Vcond,T+Th+Tr+Timp+Tf,V0);
       fprintf(vfid(strmatch(commands{k}(11:13),memristors)), '%d %d\n%d %d\n%d %d\n%d %d\n', ...
                    T+Th,V0,T+Th+Tr,Vset,T+Th+Tr+Timp,Vset,T+Th+Tr+Timp+Tf,V0);
       fprintf(sfid(strmatch(commands{k}(7:9),memristors)), '%d %d\n%d %d\n%d %d\n%d %d\n', ...
                    T+Th-Td,V0,T+Th,V1,T+Th+Tr+Timp+Tf,V1,T+Th+Tr+Timp+Tf+Td,V0);
       fprintf(sfid(strmatch(commands{k}(11:13),memristors)), '%d %d\n%d %d\n%d %d\n%d %d\n', ...
                    T+Th-Td,V0,T+Th,V1,T+Th+Tr+Timp+Tf,V1,T+Th+Tr+Timp+Tf+Td,V0);
        T=T+Th+Tr+Timp+Tf;
    elseif findstr(commands{k}(:)','FALSE')
        for n=1:str2num(commands{k}(7))
           fprintf(vfid(strmatch(commands{k}(5+4*n:7+4*n),memristors)), '%d %d\n%d %d\n%d %d\n%d %d\n', ...
                        T+Th,V0,T+Th+Tr,Vreset,T+Th+Tr+Trst,Vreset,T+Th+Tr+Trst+Tf,V0);
           fprintf(sfid(strmatch(commands{k}(5+4*n:7+4*n),memristors)), '%d %d\n%d %d\n%d %d\n%d %d\n', ...
                        T+Th-Td,V0,T+Th,V1,T+Th+Tr+Trst+Tf,V1,T+Th+Tr+Trst+Tf+Td,V0);
        end
        T=T+Th+Tr+Trst+Tf;
    else
        fprintf('ERROR - Unknown command in line %d\n',k)
        break
    end
end
fclose(commandfile);
fclose(memristorfile);
for i=1:length(vfid)
    fclose(vfid(i));
    fclose(sfid(i));
end
```

The function receives the following arguments:

- File: a file containing the desired sequence. The accepted syntax is:
  IMPLY(M##,M##) where a letter and 2 numbers representing a memristor.
  FALSE(M##).
- Mems: A file listing all the memristors to be used using the M## convention.
- Th: time between consecutive operations.
- Tr: All voltages rise time.
- Timp: IMPLY operation time.
- Trst: Reset operation time.
- Tf: All voltages fall time.
- Td: Delay between switch closing and start of voltage rise.
- V0: 0 reference voltage.
- V1: Control switch open voltage.
- Vcond, Vset, Vreset: Voltages specified for IMPLY and reset operations.

The function produces 2 files for each memristor, one controls the switch (ends with 's') and one controls the voltage driven (ends with 'v').

D2. IMPLY Parallel 8 Bit Full Adder Voltage Control Files MATLAB Code

The MATLAB function that produces the VPWLF files:

```
function VPWLp(file,mems,Th,Tr,Timp,Trst,Tf,Td,V0,V1,Vcond,Vset,Vreset)

%% Open and read memristor file, Create pwl files for memristors
memristorfile = fopen(mems);
m=0;
while(~feof(memristorfile))
    memristors(m+1,:)=fgetl(memristorfile);
    vfid(m+1)=fopen(['outputs\',memristors(m+1,:),'v.txt'],'w+');
    sfid(m+1)=fopen(['outputs\',memristors(m+1,:),'s.txt'],'w+');
    fprintf(vfid(m+1),'0 %d\n',V0);
    fprintf(sfid(m+1),'0 %d\n',V0);
    m=m+1;
end
blffid=fopen('outputs\BLF.txt','w+');
fprintf(blffid,'0 %d\n',V0);
flsfid=fopen('outputs\FLS.txt','w+');
fprintf(flsfid,'0 %d\n',V0);

%% Open and read commands file
commandfile = fopen(file);
l=0;
while(~feof(commandfile))
    commands{l+1}=fgetl(commandfile);
    l=l+1;
end

%% Write times and voltages into pwl files
T=0;
for k=1:l
    if findstr(commands{k}(:)','IMPLY')
        a=19;
        while(commands{k}(a) ~= ')')
            if commands{k}(a) ~= ','
                fprintf(vfid(strmatch([commands{k}(7:8),commands{k}(a)],memristors)), '%d %d\n%d %d\n%d %d\n%d
%d\n', ...
                                T+Th,V0,T+Th+Tr,Vcond,T+Th+Tr+Timp,Vcond,T+Th+Tr+Timp+Tf,V0);
                fprintf(vfid(strmatch([commands{k}(10:11),commands{k}(a)],memristors)), '%d %d\n%d %d\n%d %d\n%d
%d\n', ...
                                T+Th,V0,T+Th+Tr,Vset,T+Th+Tr+Timp,Vset,T+Th+Tr+Timp+Tf,V0);
                fprintf(sfid(strmatch([commands{k}(7:8),commands{k}(a)],memristors)), '%d %d\n%d %d\n%d %d\n%d
%d\n', ...
                                T+Th-Td,V0,T+Th,V1,T+Th+Tr+Timp+Tf,V1,T+Th+Tr+Timp+Tf+Td,V0);
                fprintf(sfid(strmatch([commands{k}(10:11),commands{k}(a)],memristors)), '%d %d\n%d %d\n%d %d\n%d
%d\n', ...
                                T+Th-Td,V0,T+Th,V1,T+Th+Tr+Timp+Tf,V1,T+Th+Tr+Timp+Tf+Td,V0);
            end
            a=a+1;
        end
        T=T+Th+Tr+Timp+Tf;
    elseif findstr(commands{k}(:)','FALSE')
        for n=1:str2num(commands{k}(7))
            a=15+3*str2num(commands{k}(7));
            while (commands{k}(a) ~= ')')
                if commands{k}(a) ~= ','
                    fprintf(vfid(strmatch([commands{k}(6+3*n:7+3*n),commands{k}(a)],memristors)), '%d %d\n%d %d\n%d
%d\n%d %d\n', ...
                                T+Th,V0,T+Th+Tr,Vreset,T+Th+Tr+Trst,Vreset,T+Th+Tr+Trst+Tf,V0);
                fprintf(sfid(strmatch([commands{k}(6+3*n:7+3*n),commands{k}(a)],memristors)), '%d %d\n%d %d\n%d
%d\n%d %d\n', ...
                                T+Th-Td,V0,T+Th,V1,T+Th+Tr+Trst+Tf,V1,T+Th+Tr+Trst+Tf+Td,V0);
            end
            a=a+1;
          end
        end
        fprintf(blffid, '%d %d\n%d %d\n%d %d\n%d %d\n',T+Th-
Td,V0,T+Th,V1,T+Th+Tr+Trst+Tf,V1,T+Th+Tr+Trst+Tf+Td,V0);
        fprintf(flsfid, '%d %d\n%d %d\n%d %d\n%d %d\n',T+Th-
Td,V0,T+Th,V1,T+Th+Tr+Trst+Tf,V1,T+Th+Tr+Trst+Tf+Td,V0);
        T=T+Th+Tr+Trst+Tf;
    elseif  findstr(commands{k}(:)','IMPBL')
        fprintf(vfid(strmatch(commands{k}(7:9),memristors)), '%d %d\n%d %d\n%d %d\n%d %d\n', ...
                        T+Th,V0,T+Th+Tr,Vcond,T+Th+Tr+Timp,Vcond,T+Th+Tr+Timp+Tf,V0);
        fprintf(vfid(strmatch(commands{k}(11:13),memristors)), '%d %d\n%d %d\n%d %d\n%d %d\n', ...
                        T+Th,V0,T+Th+Tr,Vset,T+Th+Tr+Timp,Vset,T+Th+Tr+Timp+Tf,V0);
        fprintf(sfid(strmatch(commands{k}(7:9),memristors)), '%d %d\n%d %d\n%d %d\n%d %d\n', ...
                        T+Th-Td,V0,T+Th,V1,T+Th+Tr+Timp+Tf,V1,T+Th+Tr+Timp+Tf+Td,V0);
        fprintf(sfid(strmatch(commands{k}(11:13),memristors)), '%d %d\n%d %d\n%d %d\n%d %d\n', ...
                        T+Th-Td,V0,T+Th,V1,T+Th+Tr+Timp+Tf,V1,T+Th+Tr+Timp+Tf+Td,V0);
        fprintf(blffid, '%d %d\n%d %d\n%d %d\n%d %d\n', T+Th-
Td,V0,T+Th,V1,T+Th+Tr+Timp+Tf,V1,T+Th+Tr+Timp+Tf+Td,V0);
         T=T+Th+Tr+Timp+Tf;
    else
        fprintf('ERROR - Unknown command in line %d\n',k)
        break
    end
end

fclose(commandfile);
fclose(memristorfile);
fclose(blffid);
fclose(flsfid);
for i=1:length(vfid)
    fclose(vfid(i));
    fclose(sfid(i));
end
```

The function same arguments as described in the previous Appendix.

The syntax of the input sequence file is:

- FALSE(4,M#,M#,M#,S#)LINES(0,1,2,3,5,6,7) – The first number represents the total number of memristors to be reset in each line, followed by the memristors names. The next numbers are the line numbers to be active.
- IMPLY(A0,T0)LINES(0,1,2,3,4,5,6,7) – This similar to the original form, with addition of multiple lines as in the False function.
- IMPBL(M10,M01) – This option performs IMPLY between lines, it receives 2 memristors, Mij and Mkl. the letter (in this case M) and the first index (i and k) are specifiers of the memristors in the line, the second indices (j and l) are line numbers.

Bibliography

[1] D. B. Strukov, G. S. Snider, D. R. Stewart and R. S. Williams, "The Missing Memristor Found," *Nature,* vol. 453, pp. 80-83, May 2008.

[2] L. O. Chua, "Memristor-The Missing Circuit Element," *IEEE TRANSACTIONS ON CIRCUIT THEORY,* Vols. CT-18, no. 5, pp. 507-519, September 1971.

[3] M. D. Pickett, B. D. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart and R. S. Williams, "Switching dynamics in titanium dioxide memristive devices," *JOURNAL OF APPLIED PHYSICS,* vol. 106, no. 074508, 2009.

[4] J. G. Simmons, "Generalized Formula for the Electric Tunnel Effect between Similar," *JOURNAL OF APPLIED PHYSICS,* vol. 34, no. 6, pp. 1793-1803, 1963.

[5] N. R. McDonald, R. E. Pino, P. J. Rozwood and B. T. Wysocki, "Analysis of Dynamic Linear and Non-linear Memristor Device Models for Emerging Neuromorphic Computing Hardware Design," *arXiv,* vol. 1008.5117, 2010.

[6] S. Kvatinsky, E. G. Friedman, A. Kolodny and U. C. Weiser, "TEAM - ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I,* 2012.

[7] S. Kvatinsky, E. G. Friedman, A. Kolodny and U. C. Weiser, "Memristor-based IMPLY Logic Design Procedure," in *Proceedings of the IEEE 29th International Conference on Computer Design*, 2011.

[8] S. Kvatinsky, K. Talisveyberg, D. Fliter, E. G. Friedman, A. Kolodny and U. C. Weiser, "Verilog-A for Memristor Models," *CCIT Technical Report ,* vol. 801, 2011.

[9] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart and R. S. Williams, "'Memristive' switches enable 'stateful' logic," *Nature,* vol. 464, pp. 873-876, April 2010.

[10] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A Materials Science & Processing,* vol. 80, pp. 1165-1172, 2005.

[11] E. Lehtonon and M. Laiho, "Stateful Implication Logic with Memristors," *IEEE/ACM International Symposium on Nanoscale Architectures,* pp. 33-36, 2009.

[12] Y. V. Pershin and M. D. Ventra, "Neuromorphic, Digital and Quantum Computation," *Proceedings of the IEEE,* 2011.

[13] E. Lehtonen, J. H. Poikonen and M. Laiho, "Two memristors suffice to compute all Boolean functions," *Electronics Letters,* vol. 46, no. 3, pp. 239-240, 2010.