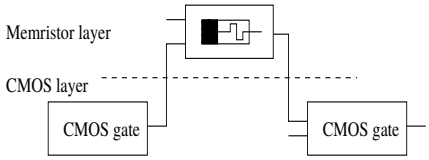


Figure 83: Hybrid Memristor-CMOS logic

Memristor ratioed logic (MRL)

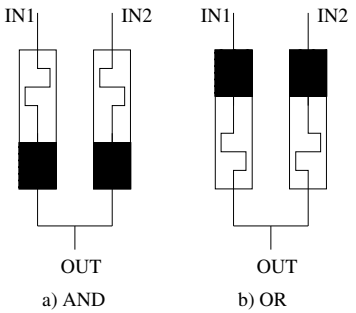


Figure 84: MRL

Similar to CMOS logic.

The output node is the common node of the memristive devices
while the signals on the other terminal of each memristive device are the inputs of logic gate.

Due to the polarity of the memristive devices, in an OR logic gate
when current flows into the logic gate through one of the inputs
the resistance of this memristive device decreases.

Similarly, in an AND logic gate, the opposite polarity is used, and
resistance of the memristive device increases when current flows into the device.

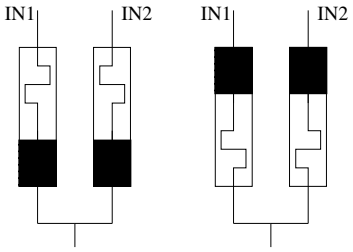
Both OR and AND logic gates react similarly to identical inputs
where either both inputs are logical 1 or both are logical 0.

For identical inputs, the voltage drop between inputs is zero
hence no current flows within the circuit.

The output voltage is, therefore, equal to the input voltage.

For the case where both inputs are logical zero (one)
the ground (supply) voltage is at the inputs, the output voltage is ground (supply) and
the logical state of output is logical zero (one).

For the case where inputs are different -that is, one is logical one and the other input is logical zero
current flows from high voltage (terminal of memristive device where the input is logical one)
to the low voltage (the terminal of the memristive device where the input is logical zero)
thus changing the resistance of both memristive devices.



This case for an OR logic gate is illustrated in Figure 84(b).

The resistance of the memristive device connected to the logical one input is lower and resistance of the other memristive device is higher.

At the end of computational process, resistance of both devices is R_{ON} and R_{OFF}.

As $R_{OFF} >> R_{ON}$, output voltage is determined by voltage divider across both of the devices. That is, $V_{out} = \frac{R_{off}}{R_{off}+R_{on}} V_{high} = V_{high}$.

In the AND logic gate, when the opposite polarity is used

The output voltage is, therefore

$V_{out} = \frac{R_{on}}{R_{off}+R_{on}} V_{high} = 0$.

MRL delay

Need for amplification

MRL uses CMOS for inversion and amplification.

N-Input MRL

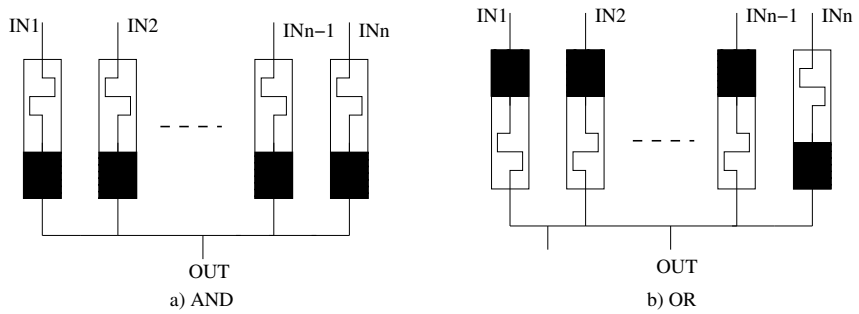


Figure 85: N-input MRL

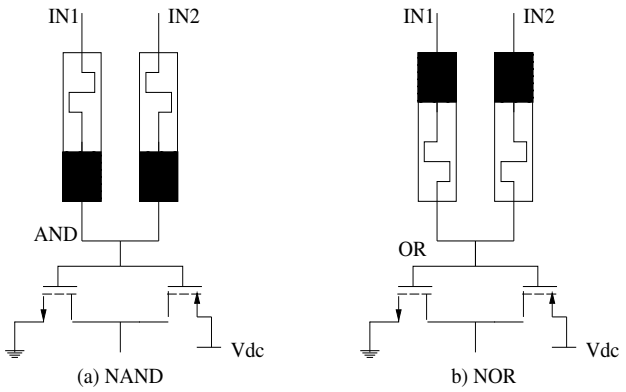


Figure 86: Integration with CMOS inverters

Memristors can be fabricated with CMOS.

Input/output are voltages as in standard CMOS logic.

CMOS gates have gain - amplifier.

MRL standard cells NAND and NOR (Figure 86).

Robust - no current leakage between stages.

Two CMOS-memristor cross-layer connections per cell consume area and power.

!h

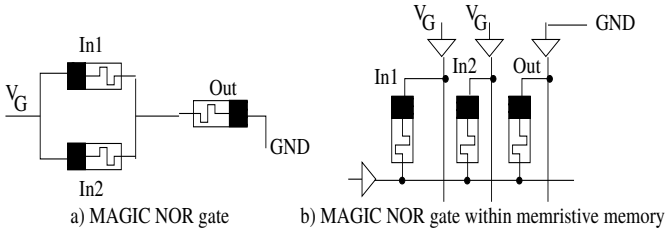


Figure 87: MAGIC NOR

Stateful logic IMPLY, MAGIC

Stateful logic family: resistance is the state variable.

Memristor-Aided loGIC (MAGIC)

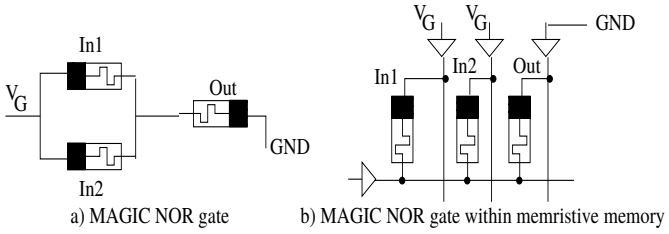
In-Memory Computing: MAGIC

MAGIC is a stateful logic family

where only a single voltage V_G is used to perform a NOR operation.

MAGIC gates do not require additional devices to perform the operation.

!h



Two input NOR gate of Figure 87(a) consists of 2 input memristors (In1, In2) connected in parallel. An additional memristor (Out) as the output.

Initial execution step includes writing a low resistance into output memristor (logical one).

If necessary, also writing the input value into memristors In1 and In2.

In final execution step, the evaluation is achieved by applying a voltage pulse V_0 at the gateway V_G.

V_0 produces a current that passes through the circuit and appears at memristor Out.

For the case where both input memristors are logical zero (high resistance)
voltage/current of the output memristor is lower than the memristor threshold voltage/current.

Hence, the logical state of the output memristor does not change and remains at logical one.

For all other input combinations, voltage/current is greater than the memristor threshold voltage/current.

The logical state of the output memristor for these input combinations switches to logical zero.

Assume a memristor with voltage thresholds of $V_{T,ON}$ and $V_{T,OFF}$.

For correct circuit behavior, voltage at Out is lower than $V_{T,OFF}$ when both inputs are logical zero.

For all other input combinations, voltage across the Out should be greater than $V_{T,OFF}$.

Minimum voltage greater than $V_{T,OFF}$ at Out is achieved when one input is 1 and the other is 0.

Where voltage at Out is to be above and below the threshold - leads to a design constraint on V_0.

The constraint is, assuming $R_{off} >> R_{on}$
$2V_{T,OFF} < V_0 < \frac{R_{OFF}}{2R_{ON}}.V_{T,OFF}$.

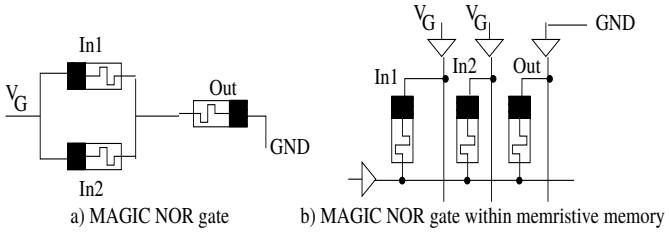
When an input memristor is logical zero, the operation of a MAGIC NOR can be destructive
changing the input to logical one during execution.

To eliminate destroying input, voltage across input needs to be below the threshold voltage $V_{T,ON}$.

Multiple-input (three or more) NOR logic gates can also be produced in a similar manner.

MAGIC NOR operation is sufficient to execute any Boolean operation.

!h



a) MAGIC NOR gate b) MAGIC NOR gate within memristive memory

MAGIC NOR may be used as the basis for performing all desired processing within memory by dividing the desired function into a sequence of MAGIC NOR operations.

The basic NOR operations are executed one after other using memory cells as computation elements.

Schematic of a MAGIC gate operation, performed over gates arranged in different rows and aligned in columns within a memristive memory is shown in Figure 87(b).

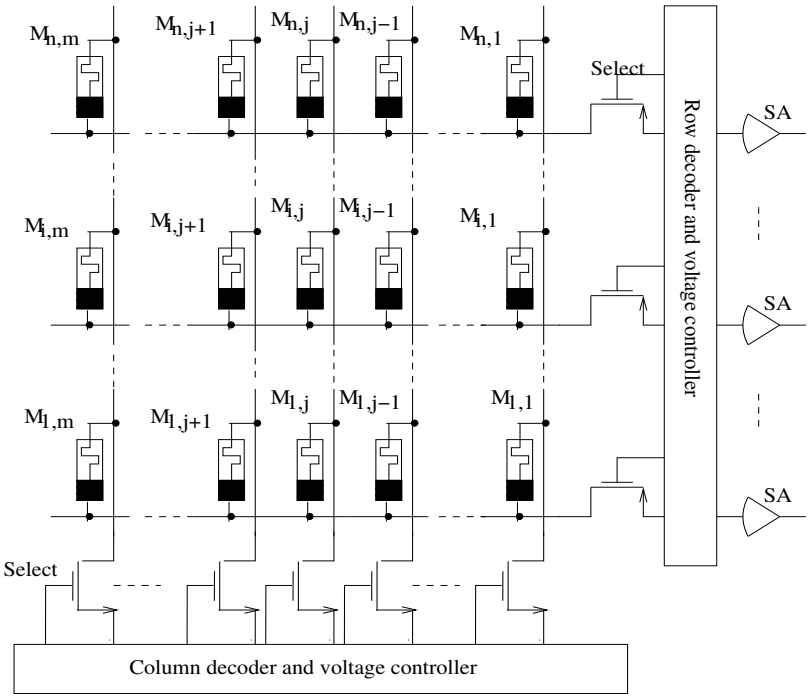


Figure 88: Memristive crossbar structure

Resistive RAM (RRAM or ReRAM) commonly utilizes a crossbar structure.

The crossbar structure enables dense memory of $4F^2$, where F is the feature size.

Memristive-only logic gates within a memristive crossbar array reduce power and provide an opportunity for novel non-von Neumann architectures where logical operations are executed within the memory.

When performing logic within the memory, the input is the stored data within the memristors and output is the stored data after execution.

Initialization of input and output is achieved as a regular memory write operation and sensing the result is achieved as a regular memory read operation.

To integrate a memristive-only logic within a crossbar array, two requirements need to be satisfied: the structure and connections of logic gate should be placed within a crossbar array and logical state of the logic gate is represented as a resistance, as in a memristive memory.

A MAGIC NOR gate fulfills both of these requirements (Figure 88).

Structure of memristive crossbar and a 2-input MAGIC NOR gate within crossbar is in Figure 88.

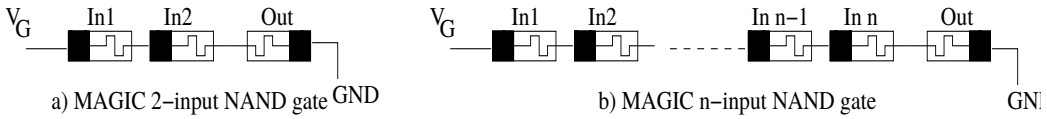


Figure 89: MAGIC NAND

MAGIC NAND/AND/OR/NOT

These additional MAGIC gates are not placed within a crossbar array (except for the NOT gate).

Can can be used as standalone logic.

MAGIC NAND

Connecting the input memristors in series produces a NAND gate (Figure 89).

NAND gate Out is also initialized to logical one.

During execution, a voltage V_0 is applied at the gateway of the circuit.

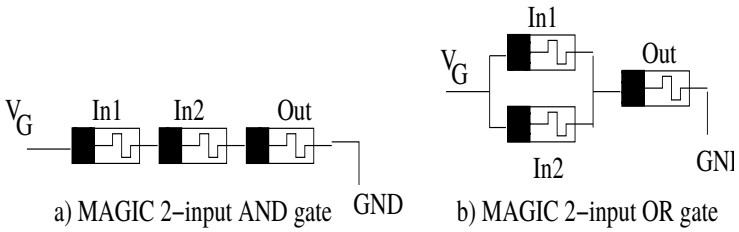


Figure 90: MAGIC AND and OR

MAGIC AND and OR

Figure 90(a) - AND

Figure 90(b) - OR

The output memristor Out is connected with the same polarity as the input memristors.

The Out is initialized to logical zero prior to execution.

Similar to NOR/NAND gates, multi-input logic gates are also possible for MAGIC OR/AND gates.

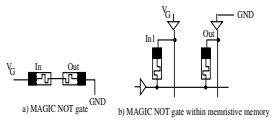


Figure 91: MAGIC NOT gate

MAGIC NOT gate

A MAGIC NOT gate consists of an input memristor *In* and an output memristor *Out*.

These are connected in series with opposite polarity in a complementary structure (Figure 91).

In the first stage of execution, Out is initialized to logical one.

When applying V_0 at V_G

voltage divider between In and Out determines whether resistance of Out changes.

For the case, where In is logical zero

voltage across Out is below the threshold voltage and state of Out remains logical one.

NOT operation can be destructive to In unless applied voltage at memristor in is below $V_{T,ON}$.

For the case where in is logical one

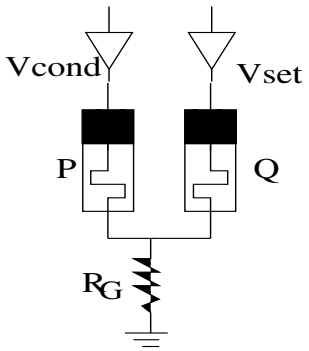
voltage across Out is sufficient to switch its state (greater than the threshold voltage) to zero.

Memristor-Based Material Implication (IMPLY) Logic

IMPLY logic was first proposed by HP Labs.

The logic function P→Q or P IMPLY Q (if P then Q) is described in Figure 92.

The implication logic (IMPLY) design executes the material implication as a universal logic function.



P(t)	Q(t)	Q(t+t∅)
0	0	1
0	1	1
1	0	0
1	1	1

Figure 92: IMPLY logic gate

The initial state of memristors P and Q is the input of the logic gate.

Output is the final state of the memristor Q after applying the voltages V_{cond} and V_{set}.

Load resistor R_G ($R_{ON} < R_G < R_{OFF}$ is connected to both memristors.

Since IMPLY function can be integrated within a memristor crossbar

IMPLY logic provides a basic logic element for a memristor-based circuit.

R_{on} is low Ohmic state and presents logic 1, and R_{off} is high Ohmic state of and presents logic 0.

Note that the memristance of both memristors changes during operation, i.e., the computation is destructive to both inputs.

Assuming that the devices P and Q are already programmed

execution of IMPLY is done by applying two control voltages V_{cond} and V_{set} (Figure 92).

$V_{set} > V_{Th} > V_{cond}$.

V_{Th} is the threshold voltage for switching of the device.

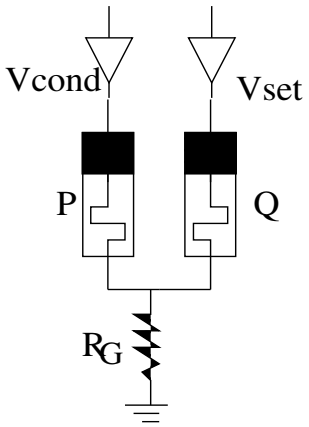
Due to polarity shown in the figure, the memristance of memristor Q can only be reduced.

For any initial state, the memristor state Q tends to drift towards the ON state.

That is, for digital operation, state of Q should either stay unchanged or switch fully ON.

Magnitude of V_{set} across a device is sufficient to switch the state of the device from R_{off} to R_{on}.

Whereas, V_{set} - V_{Th} is not sufficient enough to switch the device state.



P(t)	Q(t)	Q(t+t_0)
0	0	1
0	1	1
1	0	0
1	1	1

Case 1: If P = Q = 0 (high resistance), the applied voltage on Q is approximately V_{set} and Q is switched from R_{off} to R_{on} (Q is 1 now).

Therefore, this logic design is input destructive.

Case 2: If P = 1 (low resistance), the voltage on the common terminal is approximately V_{cond}
(no switching as $V_{cond} < V_{Th}$)
and voltage on Q is approximately V_{set} - V_{cond}.

This is sufficiently small to maintain the logic state of Q.

Although when Q is 0, the voltages tend to change internal state of Q towards ON (logical one).

This phenomenon is ‘state drift’.

Zero state of Q, which is also output of the gate, is electrically ‘weaker‘ than the input state of Q
(the memristance of Q after applying the voltages is lower than the initial memristance).

Case 3: If P = 0 and Q = 1, the logic state of Q is maintained.

That is, P IMPLY Q means - Q = P’ + Q.

State drift may require refreshing the state
otherwise, repeated or prolonged sensing action may incorrectly switch the logic state of Q.

The state drift phenomenon is a deterministic phenomenon.

Stochastic switching even change the logical state of the memristors.

Widely used linear ion drift memristor model is incompatible for logic design.

Note: V_{cond} and V_{set} are fixed.

IMPLY + FALSE $\Rightarrow$ Complete logic

Complete Logic with IMPLY

Sequential operation of IMPLY and FALSE.

NAND:

- Step 1 FALSE(S) -that is, S $\leftarrow$ 0 ($V_S = V_{CLEAR}$)
- Step 2 P IMPLY S, where $V_P = V_{cond}$ and $V_S = V_{set}$, that is. S = P' + 0 = P'
- Step 3 Q IMPLY S, where $V_Q = V_{cond}$ and $V_S = V_{set}$, that is, S = Q' + P'

These three result in S = Q' + P' -that is, S = P NAND Q.

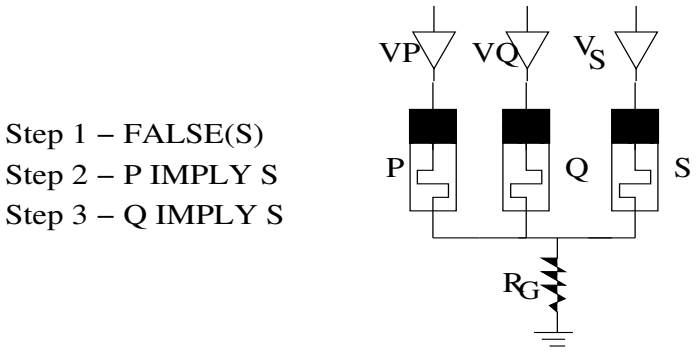


Figure 93: IMPLY NAND

The device may be cleared (assigned a logic 0; the operation FALSE) by applying a voltage V_{CLEAR}.

Many different optimizations for various Boolean functions.

IMPLY NOR

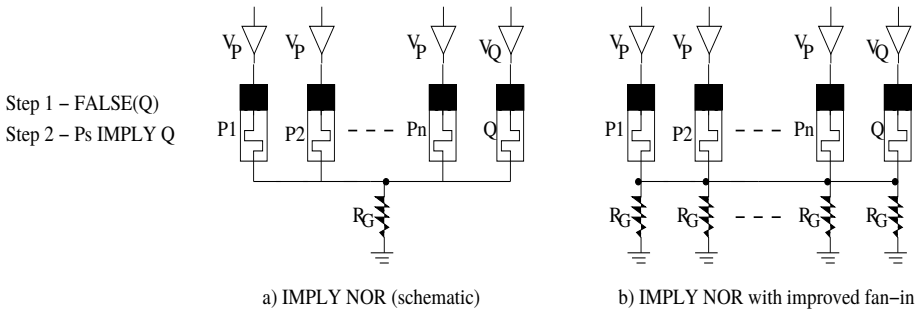


Figure 94: IMPLY NOR

The IMPLY logic gate can also be extended to a multiple input NOR logic gate (Figure 94(a)).

Figure 94(a) shows n input memristors P_1, P_2, $\cdots$, P_n, and the output memristor Q.

The operation of NOR gate requires two computational stages

- Step 1 FALSE(Q) -that is, Q $\leftarrow$ 0 ($V_Q = V_{CLEAR}$)
- Step 2 Ps IMPLY Q, where V_Ps = V_{cond} and $V_Q = V_{set}$, that is, Q = (P1 + P2 + ... + Pn)'.

Analysis: Let assume two imputs P1 and P2 and output is Q.

Then in Step 2, Q = P1, P2 IMPLY 0 means

Case 1: P1 = 0, P2 = 0, Q = 0 means output Q = 1

as if Ps = Q = 0 (high resistance), the applied voltage on Q is approximately V_{set} and
Q is switched from R_{off} to R_{on} (Q is 1 now).

Case 2: If any one of Ps = 1 (low resistance), the voltage on common terminal is V_{cond} (approx)

(no switching as $V_{cond} < V_{Th}$)
and voltage on Q is approximately V_{set} - V_{cond}.

This is sufficiently small to maintain the logic state of Q.

But as Q is 0, the voltages tend to change internal state of Q towards ON (logical one) - state drift.

However, 0 state of Q, which is also output of NOR gate, is electrically weaker than input state of Q

(the memristance of Q after applying the voltages is lower than the initial memristance).

NOR of Figure 94(a) suffers from low fan-in

since R_G needs to be scaled to all possible number of inputs.

Figure 94(b) solves this issue

where a load resistor R_G is connected to every memristor and the load resistance varies.

IMPLY XOR

FALSE(M1), FALSE(Z), X IMPLY Z, Z IMPLY M
FALSE(M2), FALSE(Z), Y IMPLY Z, Z IMPLY M
Y IMPLY M1, FALSE(Z), M1 IMPLY Z
X IMPLY M2, M2 IMPLY Z

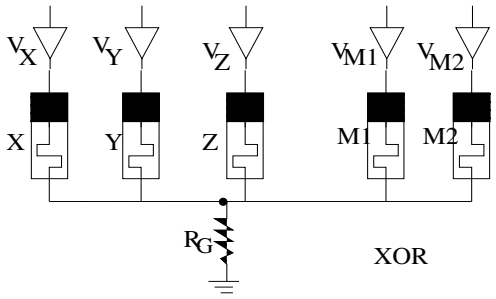


Figure 95: IMPLY XOR

Z = X XOR Y (Figure 95).

Input memristors are X and Y; Z is the output emmristor.

Two functional memristors are used : M1 and M2.

Now the following sequence of steps are to be taken to realize Z = X XOR Y.

Step 1: FALSE(M1) ⇒ M1 = 0
Step 2: FALSE(Z) ⇒ Z = 0
Step 3: X IMPLY Z ⇒ Z = X'
Step 4: Z IMPLY M1 ⇒ M1 = X (X is now copied to M1)
Step 5: FALSE(M2) ⇒ M2 = 0
Step 6: FALSE(Z) ⇒ Z = 0
Step 7: Y IMPLY Z ⇒ Z = Y'
Step 8: Z IMPLY M2 ⇒ M2 = Y (Y is now copied to M2)
Step 9: Y IMPLY M1 ⇒ M1 = Y' + X
Step 10: FALSE(Z) ⇒ Z = 0
Step 11: M1 IMPLY Z ⇒ Z = (Y' + X)'
Step 12: X IMPLY M2 ⇒ M2 = X' + Y
Step 13: M2 IMPLY Z ⇒ Z = (X' + Y') + (Y' + X)' = XY' + X'Y

The X and Y are copied to M1 and M2 since the IMPLY operation destroys both the inputs.

IMPLY within a Crossbar

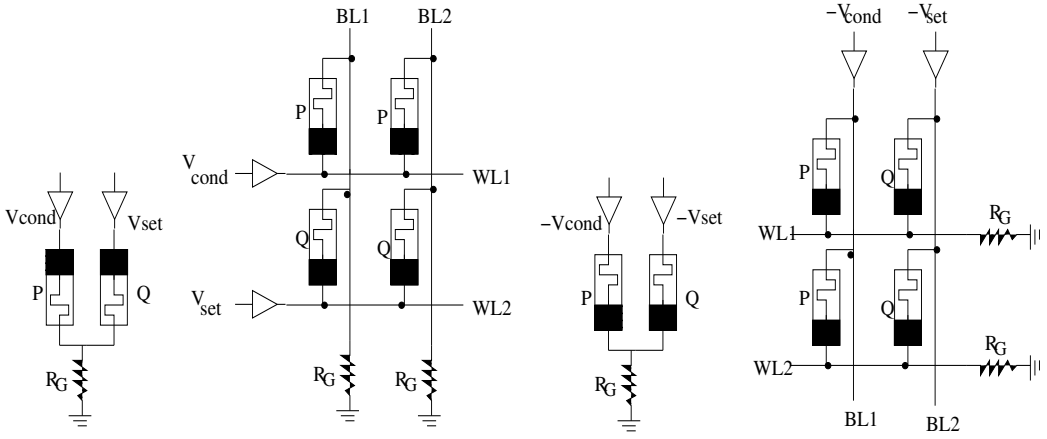
Natural method within memristive crossbar

- Input and output are resistance
- Basic structure is a crossbar-like
- Additional resistor per wordline

Figure 92 is a positive bias IMPLY scheme (PB-IMPLY).

The PB-IMPLY within memristive crossbar is shown on Figure 96(a).

Similar negative bias IMPLY (NB-IMPLY) and its memristive crossbar is shown in Figure 96(b).



P(t)	Q(t)	Q(t+t0)
0	0	1
0	1	1
1	0	0
1	1	1

c) Truth table for both (a) and (b)

Figure 96: PB-IMPLY and NB-IMPLY within crossbar

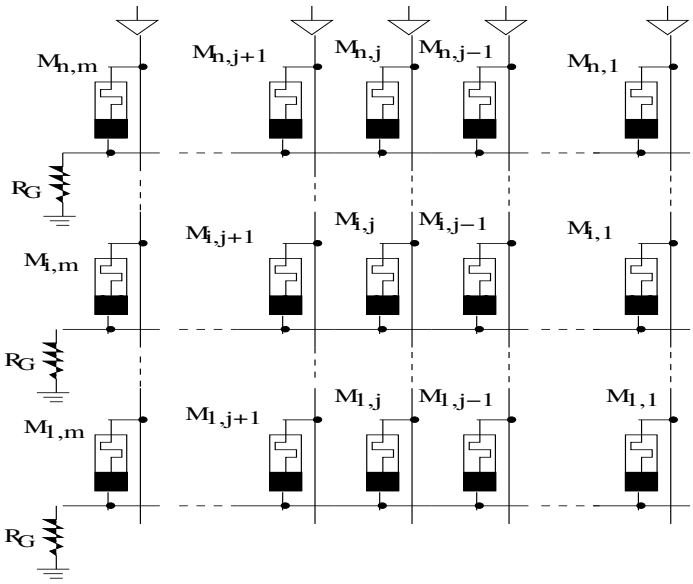


Figure 97: IMPLY within a crossbar

Counts of IMP operations (sequencial) for Boolean logic

Taking into account the possible need for copying one or both of the primary input logic values.

S ← P NAND Q requires 2 IMP Operations (latency)

S ← P AND Q requires 3 IMP Operations (latency)

S ← P NOR Q requires 5 IMP Operations (latency)

S ← P OR Q requires 4 IMP Operations (latency)

S ← P XOR Q requires 8 IMP Operations (latency)

S ← P requires 1 IMP Operations (latency)

Imply adders

IMPLY Full Adder

Design based solely on IMPLY and FALSE operations.

The Sum S_i = $(A_i \oplus B_i) \oplus C_{i-1}$ is computed following the XOR realization described earlier.

The carry is realized as

C_i =$(A_i \rightarrow (B_i \rightarrow$'0'$)) \rightarrow ((C_{i_1} \rightarrow ((A_i \oplus B_i) \rightarrow$'0')) $\rightarrow$'0').

A Memristor Ripple Carry Adder

Figure 98 shows the block diagram of a traditional n-input ripple carry adder.

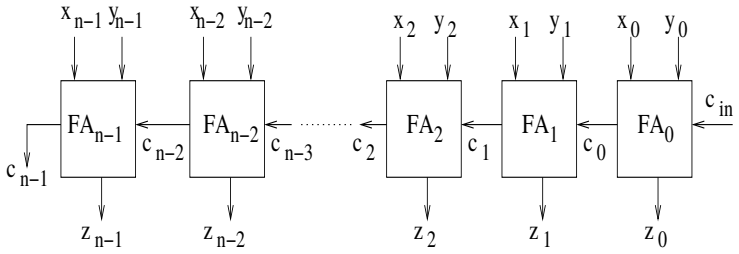


Figure 98: Ripple carry adder

The first and second full adders, producing sum z_0 and z_1 by material implication, are shown in Figure 99.

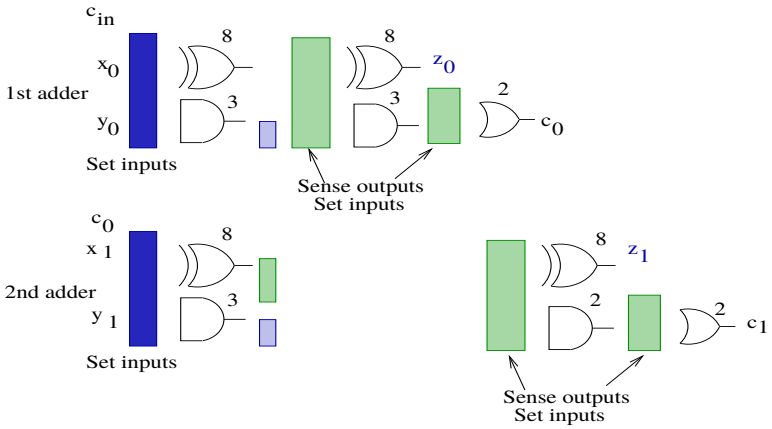


Figure 99: First and second full adders of a ripple carry adder

z_0 can be generated in 19 cycles and c_0 in 18 cycles.

Since the generation of the c_0 takes significantly longer than the 8 cycles of the first XOR operation one implication cycle of the 2^{nd} AND operation can be performed early.

This reduces the 2^{nd} AND operation from 3 cycles to 2 cycles.

A Memristor Carry Lookahead Adder

Two types of carry lookahead adders using the material implication logic is proposed.

The first is a conventional carry lookahead adder
outputs is based on creating propagation and generation terms for inputs A and B.

The sum and carry are calculated based on carry propagate and carry generate signals.

The second adder realization is based on a simplified carry lookahead adder design
based on close examination of gate delays.

In this design, sum and carry are calculated based on carry propagate and negated carry generate.

Memristive Realization for MUX

Figure 100 shows the IMP-based realization for the 2-to-1 multiplexer (MUX).

The implementation requires six computational steps and five RRAM devices.

Three of which, named S, X, and Y store the inputs.

Other two, A and B, are required for operations - thus their initial values change during operations.

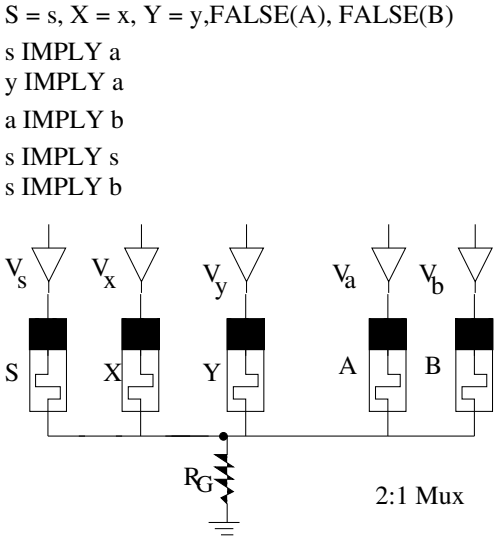


Figure 100: IMPLY realization of 2:1 mux

The implication steps of the MUX are

1: S = s, X = x, Y = y, A = 0, B = 0

2: a ← s IMP a = s'

3: a ← y IMP a = y' + s'

4: b ← a IMP b = y.s

5: s ← x IMP s = x' + s

6: b ← s IMP b = x s' + y s

In the first step, devices keeping the input variables and the two extra work switches are initialized.

The remaining steps are performed by sequential IMP operations.

That are executed by applying simultaneous voltage pulses V_{cond} and V_{set}.

Memristive Realization for Majority Gate

IMP-based realization of majority gate is similar to the circuit shown in Figure 101 with six RRAM.

Three RRAM devices denoted by X, Y , and Z keep the input variables.

Remaining 3 RRAMs A, B, and C are required for retaining intermediate results and the final output.

In the first step, the input variables are loaded and
other RRAMs are assigned to FALSE to be used later for the next operations.

Another FALSE operation is also performed in step 8.
It is to clear an RRAM which is not required anymore for inverting an intermediate result.

Finally, the majority logic is executed by implying results from the seventh and ninth step.

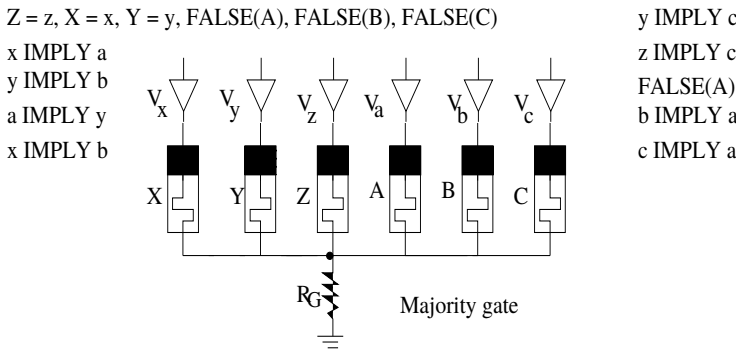


Figure 101: IMPLY realization of majority gate

The realization requires 10 sequential steps to execute the majority function.

The steps for executing the majority function are as follows

01: X = x, Y = y, Z = z, A = 0, B = 0, C = 0
02: a ← x IMP a = x'
03: b ← y IMP b = y'
04: y ← a IMP y = x + y
05: b ← x IMP b = x' + y'
06: c ← y IMP c = x + y
07: c ← z IMP c = x z + y z
08: a = 0
09: a ← b IMP a = x y
10: a ← c IMP a = x y + y z + x z.

Parallelism ⇒ SIMD

Follow Figure 102.

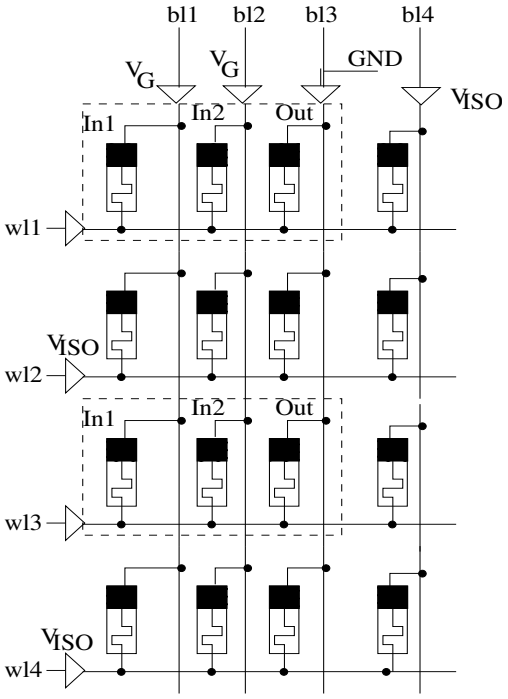


Figure 102: MAGIC SIMD

MAGIC has the ability to perform logic operations in parallel on sets of data.

Applying V_G on any two selected columns and grounding a third column (for output) will result in NOR operations on all rows that are not isolated by applying V_{ISO}.

Extended Logic Functions based on IMPLY

Any general Boolean function f: $B^n \rightarrow$ can be implemented with only n+3 memristors.

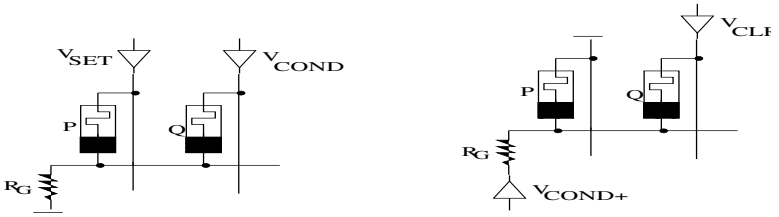
Three/two additional memristors carry out the computation.

Only two memristors are required for up to three inputs.

Computation of the function is performed in steps.

In each step, either FALSE is applied to one memristor, or an IMPLY is applied to two memristors.

This process requires a long sequence of operations depending upon the number of inputs.



P	Q	Q'
0	0	0
0	1	0
1	0	0
1	1	1

a) AND

P	Q	Q'
0	0	0
0	1	1
1	0	1
1	1	1

b) OR

Figure 103: IMPLY AND and OR

More complex logic functions can be implemented by cascading IMPLY sequentially.