Review of memristive technology for binary computations

# Introduction

# Background

## Memristors

The memristor, short for memory-resistor, was proposed as the fourth fundamental passive circuit element after the resistor, capacitor and inductor by Leon Chua [1] in 1971. In 2008, researchers at HP labs unveiled a two-terminal nanoscale memristive device. It consisted of a thin layer (50 nm) of Titanium Dioxide sandwiched between two nanowires, one made of Platinum and another made of Titanium. The memristor is a voltage-controlled device whose resistance can be varied by the application of a voltage across its terminals. The memristor is a non-volatile device, i.e. when the voltage is turned off, it retains its present resistance state until a subsequent voltage application changes its resistance state. The non-volatility and nano-scale of memristive devices make them ideal candidates for building memories.

More importantly, the memristor can be operated as a switch. The application of certain voltage levels VTRUE and VCLEAR across the memristor turns on and turns off the switch abruptly. This switching behavior makes the memristor a good candidate to replace CMOS circuits for logic applications. This paper focuses only on digital applications of memristors. Therefore, here, we consider the memristor as a linear bistable switch which exhibits either low resistance, indicating logic state ‘1’, or high resistance, indicating logic state ‘0’. When the voltage across the memristive switch goes above the switching threshold VCLOSE, it either keeps the device in the low resistance state, *CLOSED switch*, or changes its state to low resistance. Similarly, when the voltage across the memristive switch goes under the switching threshold VOPEN, it either keeps the memristor in the high resistivity state, *OPEN switch*, or changes its state to high resistance. Keeping the voltage across the memristor between VOPEN and VCLOSE does not change the resistance state of the memristor.

Fig. 1 illustrates the zero-crossing non-linear resistive characteristic of the memristor in the current-voltage (I-V) plane



Fig1- Idealized hysteretic behavior of the binary memristor. VCLOSE and VOPEN are voltage thresholds for switching the resistance state of the memristor from high-resistance, denoted by logic ‘0’, to low-resistance, denoted by logic ‘1’ and vice versa [ref].

Fig2- Idealized hysteretic behavior of the memristor. VCLOSED and VOPEN are voltage thresholds for switching the state resistance of the memristor from high-resistance (off-state) denoted by logical ‘0’) to low-resistance (on-state). The off-state and on-state are denoted by logical ‘0’ and ‘1’, respectively. [ref]

## Memristor-based Stateful Logic Operations

By combining the ability of memristors to act as switches with their ability to act as non-volatile memories, we can construct a system of stateful logic operations. In this system, the next logic state is a function of the present logic state and the applied voltages [2-3]. A single switch can be operated to perform only primitive functions TRUE and FALSE. For example, function TRUE unconditionally closes a switch by the application of VTRUE across the device, function FALSE unconditionally opens the switch by the application of VFALSE across the device. However, with a sufficient number of switches an arbitrary Boolean function can be realized. A configurable circuit to perform stateful logic is illustrated in Fig. 2. The circuit is comprised of two memristive switches, labeled P and Q, electrically connected through the common horizontal nanowire BL, which is itself grounded by the load resistor RG. The input of the logic function is the logic state encoded by the resistance of switches P and Q, whereas the output of the circuit is the new logic state of switch Q. Logic functions are realized by performing two simultaneous operations, READ and WRITE. The READ operation is performed by applying VCOND to P, where VOPEN <VCOND <VCLOSE. The WRITE operation is performed by applying VSET to Q, where VCLEAR ≤VSET ≤VTRUE, see Fig. 1. The READ operation drives the voltage of nanowire BL to either ≈ VCOND or to ≈ 0V, as determined by the state value of P. The WRITE operation conditionally toggles the state of Q, based on the initial state of Q and the voltage on BL. The state of P remains unchanged during the operations.

Let RCLOSED and ROPEN denote the resistance of a memristive switch in logic state ‘1’ and ‘0’, respectively, where $ R\_{OPEN}\gg R\_{CLOSED}$. Since the correct operation of the circuit requires BL to be either ≈ 0V or ≈ VCOND, care must be taken to ensure $R\_{OPEN}$/$R\_{G}$ >>1 and $R\_{G}$/$R\_{CLOSE}$>>1. Assuming these inequalities are satisfied, we use $R\_{OPEN}$/$R\_{G}$ = $R\_{G}$/$R\_{CLOSE}$ from which $R\_{G}$ = $\sqrt{R\_{OPEN}R\_{CLOSE}}$ is derived [4].

A flow of current through Q could disturb the WRITE operation. If the current raises the voltage of BL in a time interval shorter than the switching time of Q, the WRITE operation is disturbed. Therefore, the RC time constant of BL should be larger than the switching time of the memristive device, where R = RBL+RG, combined resistance of nanowire BL and load resistor RG, C denotes the stray capacitance of the nanowire BL. However, a large RC slows down the circuit operation. The use of external CMOS circuitry—a keeper circuit— can help improve the speed and accuracy of circuit operation [5]; however the area overhead would present another challenge.

A better solution to the designing of the WRITE operation would be to use the rectifying memristive switches introduced by Kim, *et al*., [6-7], elaborated in the next section, with the circuit configuration of 0V < VCOND <VCLOSED and VOPEN <VSET < 0V, i.e., the target memristor is reverse biased. This architecture blocks the flow of current through Q during the WRITE operation, and thus, enables the inhibition operation (material non-implication) with no external CMOS circuitry [8]. For the rest of the paper, the term *architecture* refers to a particular specification of voltage configuration, type of memristors, and circuit structure.



Fig. 2. A 1×2 crosspoint array. The application of two simultaneous voltage pulses VCOND and VSET to switches P and Q, respectively, conditionally toggles the state logic of Q depending on the initial state of P and Q, and the magnitude and polarities of VCOND and VSET.

 Fig. 3 illustrates the idealized hysteretic behavior of the rectifying memristor. The destructive voltage pulses VTRUE and VCLEAR *set* and *clear* the switch. The switching voltage thresholds VCLOSE and VOPEN are chosen to be symmetric, $V\_{OPEN}= -V\_{CLOSE},$ for simplicity. Let $I\_{+}$ be the current flowing through the forward biased memristor and $I\_{-}$ be the current through the reverse biased memristor. This current shown for negative voltages is nearly horizontal line because of very high resistance when reverse biased.

The current ratio $\frac{I\_{+}}{I\_{-}}$ is of the order of magnitude $10^{6}$[6]. The resistance ratio $\frac{R\_{OPEN}}{R\_{CLOSED}}$ is of the order of magnitude $10^{3}-10^{6}$[6-7] demonstrating the robustness of the circuit shown in Fig. 2 when constructed with rectifying memristive switches.

Fig. 3. Idealized hysteretic behavior of the rectifying memristor. VCLOSED and VOPEN are the voltage switching thresholds for conditionally toggling the state resistance of the device between ROPEN and RCLOSED. ROPEN and RCLOSED encode logic ‘0’ and ‘1’, respectively. The abrupt voltage thresholds are symmetric: VOPEN= -VCLOSED.


## Stateful logic gates with standard memristors

Now that the basic circuit design concepts have been discussed, the question becomes what stateful logic gates can be realized given specific design choices. In this section, designs based on standard memristors are discussed, and in Section D designs based on rectifying memristors will be discussed. We apply all polarity combinations of Vcond and Vset to the 1×2 crosspoint array with standard memristors to examine all possible logic gates that can be realized in the circuit. The same procedure will be applied to the 1×2 crosspoint array with rectifying memristors, see Section D.

### Stateful AND gate

 The stateful AND gate was proposed in [9]. However, a 1×2 crosspoint array alone is not sufficient to implement the gate; a keeper circuit is required. The stateful AND gate requires VSET < VOPEN <VCOND < 0V and VOPEN <VSET-VCOND. With this voltage combination, the use of a keeper circuit is necessary and it changes the logical operation of the crosspoint array from no operation to AND. Take, for example, the case where P and Q are both logic ‘0’, denoted as *p, q* = [0, 0] where *p* is the state of memristor P and *q* is the state of memristor Q. The READ operation drives nanowire BL to ≈0V. After the WRITE operation, the voltage across Q is below VOPEN so Q remains ‘0’, as expected. Similarly, when P is ‘1’ the crosspoint array behaves as expected for the logic AND, regardless of the state of Q. However, in the case where *p, q* = [0, 1], the crosspoint array does not exhibit the desired behavior. The READ operation keeps BL at ≈0V. Since RG >>RCLOSE of Q, the WRITE operation drops VSET across RG and thus the voltage difference across Q is almost 0V. This is insufficient to change the state of Q, therefore the AND function would operate incorrectly. Laiho *et al*. [5] proposed splitting the simultaneous READ WRITE operation into consecutive operations. During the READ operation the voltage drop on BL is sampled by the keeper circuit. During the WRITE operation, the keeper circuit forces the sampled voltage on to BL. Consequently, VSET drops entirely across Q. That voltage drop is sufficient to toggle the state of Q to ‘0’ which is the desired behavior of AND gate. The area overhead of the keeper circuit is not the only obstacle that needs to be overcome; AND by itself is logically incomplete. In order to have a logically complete system, inversion is required as well. Inversion can be realized as a special case of the stateful IMP gate, explained in C.2.

### Stateful IMP gate

The stateful IMP gate was proposed in [2-3] and is defined as *q: = p→q ≡* $\overbar{p}$*+q* where ‘*→*’ is the IMP operator (short for IMPLY operator). IMP gate can be implemented with a keeper circuit, or without a keeper circuit if sufficient capacitance on BL is assumed. With a keeper circuit, the crosspoint array runs at full speed with the cost of increased area and power; in contrast, without a keeper circuit, the crosspoint array runs slow, but it consumes less area and power. The stateful IMP gate requires 0V< VCOND <VCLOSE < VSET, and VSET-VCOND < VCLOSE. Take, for example, the case where *p, q*= [0, 0]. The READ operation drives BL to ≈0V. The WRITE operation causes a voltage drop sufficient to toggle the state of Q, which is the expected behavior as shown in Table 1. The circuit operates similarly for all combinations of *p*, *q*. Implication logic is functionally complete when coupled with the FALSE operation. Recall that the FALSE operation is implemented by VSET =VCLEAR.

### Stateful OR gate

Much like IMP, the stateful OR gate [proposed by Phil Kuekes] can be implemented with or without a keeper circuit and the same power, speed and area tradeoff applies. In practice, the use of a keeper circuit in parallel with voltage divider ensures the correct operation of the circuit. The stateful OR gate requires VOPEN < VCOND < 0V < VSET < VCLOSE, and VCLOSE < VSET -VCOND. With these control voltages, the circuit operates correctly. OR gate by itself is logically incomplete. In order to have a logically complete system, inversion is required as well. Inversion can be realized as a special case of the stateful IMP gate.

### Additional stateful gate

As discussed earlier in this section, different polarity combinations of VCOND and VSET implement different logic operations. There remains one voltage combination to consider: VOPEN < VSET < 0V < VCOND < VCLOSE, and VCOND -VSET > VCLOSE. Assuming sufficient capacitance on BL, this configuration implements the OR gate with the output on P instead of Q. However, the use of a keeper in parallel with RG produces the INH gate. The stateful INH gate was proposed in [8] and is defined as *q: = p*$↛$*q ≡* $\overbar{p}$*q* where ‘$↛$’ is the INH operator. The INH truth table is shown in Table 2. Take, for example, the case where *p, q*= [1, 1]. The READ operation drives BL to ≈VCOND. During the READ operation the voltage on BL is sampled by the keeper circuit. During the WRITE operation, the keeper circuit forces the sampled voltage onto BL. Consequently, the voltage across Q becomes VSET –VCOND. This voltage difference is sufficient to toggle the state of Q to ‘0’ which is the desired behavior of the INH gate. Inhibition logic is functionally complete when coupled with the TRUE operation. Recall that the TRUE operation is implemented by VSET =VTRUE. Similar to its use in the AND gate, the keeper circuit here splits READ/WRITE into two consecutive operations.

In summary, a simple 1×2 crosspoint array with standard memristors is sufficient to implement several useful logic gates as long as care is taken in the selection of load resister RG. Which particular logic gate is implemented depends on the magnitude and polarity of the applied control voltages and on whether or not a keeper circuit is used. The circuit that provides these control voltages is called driver and it can be built from hybrid CMOS-memristor devices.

|  |  |  |
| --- | --- | --- |
| In | In | Out |
|  P | Q | $p↛$q |
|  0 | 0 | 0 |
|  0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| In | In | Out |
| P | Q | p$⟶$q |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

 Table 1. The truth table for IMP operation q’: p$\rightarrow $q≡$\overbar{p}+q$ Table 2. The truth table for INH operation q’: p$↛$q≡$\overbar{p}q$

As already discussed, Table 3 summarizes the logic gates realized by all possible voltage polarity and keeper/no-keeper circuit combinations. Concluding in standard memristor without keeper circuit we can realize not only IMPLY gate but also with both Q (line 2) and P (line 3) as stateful memristors. This dual realization on OR allows to minimized some designs when compared with those realize with only IMP gates. When the keeper circuit is available, standard memristor allows to realize all four operations: IMP, INH, OR, and AND which can decrease the cost of several function implementations. In Section D we follow a similar pattern to discuss the logic gates possible with rectifying memristors.

|  |  |
| --- | --- |
| Control Polarity | Logic Function |
| VCOND | VSET | $\uparrow $ RC on BL | keeper circuit |
| + | + |  q':= p$\rightarrow $q (IMP) | q':= p$\rightarrow $q (IMP) |
| - | + |  q':= OR (p, q) | q':= OR (p, q) |
| + | - |  p':= OR (p, q) | q':= p$↛$q (INH) |
| - | - | No logic operation | q':= AND(p, q) |

Table 3. The logic gates realized by all possible voltage polarities and keeper/n0-keeper circuit combinations.

## Stateful logic gates with rectifying memristors

In this section, all stateful gates that can be realized in a 1×2 crosspoint array constructed with rectifying memristors are studied. The application of various polarity combinations of Vcond and Vset results in IMP and INH logic gates as shown in Table 4.

Table 4. The logic gates realized by all possible polarity combinations

|  |  |
| --- | --- |
| VCOND, VSET Polarities  | Executed gate |
| +, + | q:= p$\rightarrow $q  |
| +, - | q:= q$↛$p |
| -, + | p:= q$↛$p |
| -, - | No logic operation |

### Stateful IMP gate

A stateful IMP gate is realized with the same polarity combinations of Vcond and Vset described in a crosspoint array constructed with standard memristors, i.e. it requires 0V< VCOND <VCLOSE < VSET, and VSET-VCOND < VCLOSE. An IMP gate can be implemented with a keeper circuit or without a keeper circuit assuming sufficient capacitance on BL. The same speed/power/area tradeoff imposed by a keeper circuit in a crosspoint array with standard memristors also hold for a crosspoint array with rectifying memristors.

###  Stateful INH gate

A stateful INH gate is realized in a 1×2 crosspoint array **with NO keeper circuitry** with the same polarity combination of Vcond and Vset described in a crosspoint array constructed with standard memristors, i.e. VOPEN < VSET < 0V < VCOND < VCLOSE, and VCOND -VSET > VCLOSE. This polarity combination (Vset <0) always keeps target memristor Q reverse biased and thus no current flows through Q due to its rectifying property. As a result, the crosspoint array can be seen as a voltage divider comprised of the source memristor and load resistor RG that controls the state resistance of target memristor Q, i.e. no electrical current flows between the voltage divider and the target memristor and thus the state of the target memristor is a function of its initial state and the output of the voltage divider, VBL. Note that the rectifying property of the memristors simplifies the circuit operation and eliminates the need for a keeper circuitry.

The stateful INH gate can be also realized with the same polarity combination of Vcond and Vset described in a crosspoint array constructed with standard memristor for q: = OR (p, q), i.e. VOPEN < VCOND < 0V < VSET < VCLOSE, and VCLOSE < VSET -VCOND. This configuration implements the INH gate with the output on P instead of Q, i.e. P is the target memristor whereas Q is the source memristor. No keeper is required with this circuit configuration.

There remains one voltage combination to consider: VSET < VOPEN <VCOND < 0V and VOPEN <VSET-VCOND. This circuit configuration produce no logic operation.

The immediate stateful gate is defined as one step operation which requires a single combination of a controlling voltages to realize a single stateful logic gate such as AND or OR.

Note that immediate stateful OR and AND gate cannot be realized with a crosspoint array constructed with rectifying memristors. However, unlike stateful OR/AND gate, the INH gate with TRUE gate create a functionally complete logic set. Moreover, in practice, the realization of OR or AND stateful gate requires a keeper circuit, but the realization of INH gate with rectifying memristors does not require any keeper circuitry. In terms of area and power consumption, a crosspoint array constructed with rectifying memristors consumes less area and power, i.e. no keeper circuitry is imposed for its correct operations. In terms of circuit complexity, the rectifying property of memristors simplifies the circuit operation, as mentioned in previous paragraph. As will be discussed in Section X, the use of rectifying memristor in crossbar arrays suppresses the sneak path. However, the use of standard memristors in a crossbar array requires extra cost to suppress the undesired sneak path.

Table 5 summarizes all the tradeoffs discussed above for implementing logic gates in a crosspoint array constructed with standard and rectifying memristors. For the rest of this paper, we consider a rectifying memristor as a basic element used in constructing a crosspoint or a crossbar array and implement any Boolean function with INH gates.

In summary, realization of stateful AND gate and stateful INH gate imposes the use of a keeper circuit. In practice, stateful OR gate and stateful IMP gate require keeper circuits for reliable performance. The immediate implementation of stateful OR gate and stateful AND gate requires an advanced keeper circuit. In this process VCOND is always a positive voltage whereas the voltage on the common nanowire during the WRITE operation can be either 0V, ±VCOND depends on an implemented logic gate.

|  |  |  |
| --- | --- | --- |
| memristors | Standard | Rectifying  |
| Stateful logic | AND | IMP | OR | INH | IMP | INH |
| Functionally complete logic set? | No | +FALSE | No | +TRUE | +FALSE | No |
| Need a keeper circuit? | Must | PracticallyYes | Must | Practically Yes | No |
| Area consumption | +keeper | +keeper | +keeper | +keeper | +keeper | -keeper |
| Power consumption | High | High | High | High | High | Less |
| Circuit complexity | High | High | High | High | High | less |
| Sneak Path | Yes | No |
| Multi-Input  | +keeper | Practically +keeper | -keeper |
| Multi-output | +keeper | +keeper | -keeper |

Table 5. Properties of crosspoint array constructed by standard memristors and rectifying memristors.

Add schematic of a keeper circuit

##  Implementation of multi-input multi-output logic gates in a generalized crosspoint array

In this section, the implementation of a multi-input multi-output stateful logic gates in crosspoint arrays will be discussed. Multi-input multi-output stateful gates can be executed in a crosspoint array scaled to a 1×*n* or an *n*×1array, i.e. a vector of rectifying memristors.

Fig. 4 shows a 1×5 crosspoint array. Our goal is to implement a two-input single-output stateful INH F: = OR {a, c}$↛${e}, i.e. $F= \left(\overbar{a+c}\right)∙e=\overbar{a}\overbar{c}e$. Target memristor M5 must be initially closed, i.e. it must be set to low resistance state RON encoded by logic ‘1’*, e=1*. Therefore,$ F= \left(\overbar{a+c}\right)=\overbar{a}\overbar{c}$, a two-input stateful NOR whose output updates the logic state of target memristor M5, i.e. *e=F*. Memristors {M1, M3} are driven by VCOND and thus act as source memristors. The source memristors store logic variables $a$ and$ c$. Memristor {M5} is driven by VSET and thus acts as target memristor. Target memristor M5 stores logic variable$ e$. Memristors {M2, M4} do not take part in logic operation. Nonparticipating memristors are terminated to high-impedance, HZ. Function F can be implemented by executing three consecutive gates: one TRUE gate and two stateful INH gate as follows.

1. SET target memristor M5. Gate TRUE can be implemented by driving M5 with VTRUE.
2. Implement F1:=$ a↛e$. This operation change the state resistance of M5 to$ \overbar{a}$.
3. Implement F2:= c$↛e$. This operation will update the state resistance of M5 to$ \overbar{a}\overbar{c}$.

The second gate, stateful INH F1, is implemented by simultaneously driving M1 and M5 to VCOND and VSET, respectively. Source memristor M1 together with load resistor RG construct a voltage divider whose output VBL drives target memristor M5. If the resistance state of M1 is low, the voltage on the common nanowire will be ≈ VCOND, and the resistance state of the target memristor will toggle. In other words, if the voltage drop across the target memristor is more negative than VOPEN, the state resistance of the target memristor will toggle, otherwise it remains unchanged. Note that a special case of an INH gate is a stateful NOT when the target memristor of the gate is initialized to low resistance state. The last gate, stateful INH F2, is implemented similar to F1 by simultaneously driving M3 and M5 to VCOND and VSET. The result of F2 updates the state resistance of M5 to$ \overbar{a}\overbar{c}$. Note that gate F1 and F2 can be simultaneously implemented by driving M1, M3, and M5 to VCOND, VCOND, and VSET, respectively.

Concluding that 1) a primitive INH gate can be used for immediate implementation of stateful NOR/NOR gate. This implementation requires initializing the target memristor of the first INH gate being executed to low-resistance state. 2) A multi-input stateful NOR can be sequentially implemented by cascading stateful INH gates or can be simultaneously implemented by executing all stateful INH gates.



Fig. 4. Two-input single-output NOR gate.

A multi-input single-output NOR gate can be extended to multi-input multi-output NOR gate by simultaneously driving several memristors to VSET. For instance, driving memristor M4 and M5 by VSET and M1 and M3 by VCOND would extend the stateful NOR gate to two-input two-output stateful NOR gate. The simulation results demonstrate that stateful NORs constructed with rectifying memristors have no fan-out restriction as CMOS gates have. For instance, in a 1×64 crosspoint array, we demonstrated the correct operation of a two-input sixty two-output stateful NOR gate, see Fig. 5.

a

b

Fig. 5. A two-input sixty two-output stateful NOR implemented in a 1×64 crosspoint array. The circuit parameters are VCOND=0.6V, VTRUE=1.2V, TD=0, TR=0, TF=0,and PWD=6ns where TD, TF, and PWD are offset delay, rising delay, and falling delay of signal drivers(a) When both input memristors are in low-resistance states, the average voltage on common nanowire BL is 457.83mV and the switching delay of the gate is 5.153ns. (b) When one of the input memristor is in a low-resistance state while the other one in a high-resistance state, the average voltage on common nanowire BL is 423.96mV and the switching delay of the gate is 5.153ns.

Fig. 6 illustrates an analogous circuit to the crosspoint array constructed with rectifying memristors. The circuit shows the immediate implementation of multi-input stateful NAND by primitive IMP gates. A reliable performance of a stateful NAND may impose the use of a keeper circuit to maintain VBL, the manifestation of logical OR of input logic variables encoding the resistance state of source memristors driven by VC. Fig. 6 shows a voltage divider whose output VBL drives the negative terminal of target memristor T. The circuit configuration for stateful IMP requires target memristor T to be forward biased, VSET-VCOND> 0. In addition, immediate implementation of stateful NAND with primitive stateful IMPs requires the initialization of target memristor T to high-resistance state, ROFF. Being in forward bias, an electrical current flows through target memristor T. This flow of current raises VBL and consequently drops the voltage across T, VSET-VBL. The output of the stateful NAND is a function of the current state of the target memristor and the input variables decoded as the voltage difference across T. Since this voltage difference changes during the WRITE operation, the NAND operation may be disturbed.

This disturbance can be overcome utilizing nanowires with sufficiently large substantial capacitance. This substantial capacitance must keep the voltage on the common nanowire during the WRITE operation, the application of VSET. The WRITE operation must be performed in a time interval shorter than the time constant RC. However, this solution is not efficient for multi-output NAND gate since the flow of electrical current through the target memristors largely increases VBL. In this case, a keeper circuit would be a solution which imposes large area.

Remark: In implementation of a stateful NOR with basic stateful INHs, the circuit configuration keeps the target memristors in reverse bias and thus blocks the flow of current through the memristors. Therefore, the voltage on the common nanowire is maintained without the need for a keeper circuit or utilizing nanowires with large substantial capacitance.

VC

VS

RG

ROFF

RON

ROFF

VBL

*i*

Fig. 6. An analogous to crosspoint array implementing multi-input single-output stateful NAND gate with primitive stateful IMP gates.

An N-input stateful NAND gate can be implemented in a crosspoint array with N+1 consecutive stateful gates in two consecutive time slops (pulses):

1. CLEAR gate: Applied to the target memristor to clear the resistance state of the memristor, i.e. program the memristor to high resistance state.
2. N primitive stateful IMP gates: Applied simultaneously.

In short, implementing a multi-input multi-output stateful NOR gate in a crosspoint array constructed with rectifying memristors utilizing stateful INH gates has the least requirement (cost) among other stateful logic gate e.g. multi-input multi-output NAND, OR, and AND gate.

## Logic synthesis in a crosspoint array

In this section, our goal is to synthesize a given Boolean function utilizing only stateful NORs The first step in logic synthesis base on NORs is obviously rewriting a given Boolean function based on NOR gates. Thus it is beneficial to recall De Morgan laws:$ \overbar{ac}=\overbar{a}+\overbar{c}$, and$ \overbar{a+c}=\overbar{a}\overbar{c}$. We start with implementing a parity function of two inputs in a crosspoint array constructed with rectifying memristors. Function $F=a⨁c=\left(a+c\right)(\overbar{a}+\overbar{c})$ can be rewritten as$ F=\overbar{\overbar{\left(a+c\right)}+\overbar{(\overbar{a}+\overbar{c})}}=\overbar{A+B}$, i.e. F is presented as two level NOR network: $A= \overbar{a+c}$, $B= \overbar{\overbar{a}+\overbar{c}}$, and $\overbar{A+B}$. Note that implementation of F requires producing the complements of inputs, $\overbar{a}$ and$ \overbar{b}$. Implementing F requires six consecutive stateful logic gates: one TRUE gate, two NOT gates, and three NOR gates as summarized in Table 6. Note that stateful NOT is a special case of stateful INH gate where the output of the gate is initialized to low resistance state, logic ‘1’, e.g. $G≔m↛n$ is $G=\overbar{m}$ if initial state of the target memristor is *n* =1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Gate | M1 | M2 | M3 | M4 | M5 | M6 | M7 | BL | V1 | V2 | V3 | V4 | V5 | V6 | V7 |
| TRUE | *a* | *c* | 1 | 1 | 1 | 1 | 1 | VSET | HZ | HZ | VCOND | VCOND | VCOND | VCOND | VCOND |
| NOR | *a* | *c* | 1 | 1 | *A* | 1 | 1 | RG | VCOND | VCOND | HZ | HZ | VSET | HZ | HZ |
| NOT | *a* | *c* | $$\overbar{a}$$ | 0 | *A* | 1 | 1 | RG | VCOND | HZ | VSET | HZ | HZ | HZ | HZ |
| NOT | *a* | *c* | $$\overbar{a}$$ | $$\overbar{c}$$ | *A* | 1 | 1 | RG | HZ | VCOND | HZ | VSET | HZ | HZ | HZ |
| NOR | *a* | *c* | $$\overbar{a}$$ | $$\overbar{c}$$ | *A* | *B* | 1 | RG | HZ | HZ | VCOND | VCOND | HZ | VSET | HZ |
| NOR | *a* | *c* | $$\overbar{a}$$ | $$\overbar{c}$$ | *A* | *B* | *F* | RG | HZ | HZ | HZ | HZ | VCOND | VCOND | VSET |

Table 6. An implementation of a parity function with two inputs in a 1×7 crosspoint array.

Computing F requires implementing a sequence of a six stateful gate as shown in the Table.

As shown in Table 6, a given Boolean function is implemented sequentially. This serial computation may not be a limitation, if Boolean functions are implemented in a pipelined memristive circuits.

An *m*×*n* crossbar array is a simple pipelined circuit. It allows to uniquely access each of $m×n$ memristors within the array with only $n+m$ CMOS drivers. However, this dense circuitry with few voltage controllers imposes a limited form of parallelism in the circuit e.g. one type of logic operation can be implemented at each time slot. Fig. 7 shows a 3×3 memristive crossbar array constructed with perpendicular nanowires (electrodes); in each crosspoint or junction— an area between two perpendicular nanowires— one rectifying memristor is constructed. This two dimensional array can be seen as a three row memristors, each of the size of 1×3, electrically connected by vertical nanowires. The vertical connections also form three column memristors, each of the size of 3×1. In this crossbar array, vertical memristors are connected to the positive polarity of memristors.

|  |  |  |
| --- | --- | --- |
| m11 | m12 |  m13 |
| m21 | m22 |  m23 |
| m31 | m32 |  m33 |

C1 C2  C3

R1

R2

R3

**+**

**-**

Fig. 7. A 3×3 memristive crossbar array. In each junction one memristor (one arrow) is constructed. Vertical nanowires C*i* and horizontal nanowires R*i* are connected to CMOS drivers.

Function F can be executed in a 3×3 crossbar array in a four time slots (pulse). The logic states of crossbar memristors after each operation are represented in a 3×3 matrix, as shown in Fig. 8. The row and column memristors of the crossbar array are connected to voltage drivers whose values are shown on top and on left side of each matrix. Note that a column memristor refers to a crosspoint array with common vertical nanowire and a row memristor refers to a crosspoint array with common horizontal nanowire.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | *a* | - | - |
|  | *c* | - | - |
|  | - | - | - |

 a

|  |  |  |  |
| --- | --- | --- | --- |
|  | HZ  | VC | VC |
| VS | *a* | 1 | 1 |
| VS | *c* | 1 | 1 |
| VS | - | 1 | 1 |

 b

|  |  |  |  |
| --- | --- | --- | --- |
|  | VC | VS | HZ |
| RG | *a* | $$\overbar{a}$$ | 1 |
| RG | *c* | $$\overbar{c}$$ | 1 |
| VS | 1 | 1 | 1 |

 c

|  |  |  |  |
| --- | --- | --- | --- |
|  | RG | RG | 0V |
| VC | *a* | $$\overbar{a}$$ | 1 |
| VC | *c* | $$\overbar{c}$$ | 1 |
| VS | *A* | *B* | 1 |

 d

|  |  |  |  |
| --- | --- | --- | --- |
|  | VC | VC | VS |
| 0V | *a* | $$\overbar{a}$$ | 1 |
| 0V | *c* | $$\overbar{c}$$ | 1 |
| RG | *A* | *B* | *F* |

 e

Fig. 8. Realization of two-input parity function in 3×3 crossbar array. (a) Initial state of the crossbar array. (b) Implementation of TRUE gate. (c) Implementation of NOT gate. (d)

Fig. 8a shows the initial state of each memristor of the crossbar array. The state values of input memristors are *a* and *c*. The state logic of the rest of memristors are unknown, shown with ‘-’. Fig. 8b show implementation of a column-wise TRUE gate, i.e. setting the resistance states of second and third column memristor to logic ‘1’, low resistance state. This column-wise TRUE gate is realize by driving the second and third vertical nanowire to VC, the same VCOND, and all horizontal nanowire to VS, the same VSET. Fig. 8c shows implementation of a column-wise NOT gate. This column-wise gate produces the inversion of input logic values *a* and *c*. While executing column-wise NOT, a TRUE gate can also be implemented to set the resistance state of memristor M31 to logic ‘1’ for next logic implementation. Memristor M31 is located between the first vertical and the third horizontal nanowires of the crossbar array. Note that executing more than one type of logic gate requires a certain data arrangement in a crossbar array to avoid any undesired operations on crossbar data. Fig. 8d shows implementation of a column-wise NOR gate that produces A and B. Fig. 8e shows implementation of a NOR gate applied to A and B to produce the parity function, F.

In summary, the memristive crossbar structure allows implementation of column-wise and row-wise stateful logic gates. However, with crossbar array usually one type of stateful logic gate can be implemented at a time. Also, there exists another potential problem in a crossbar array called sneak path.

Fig. 9 shows the phenomenon of the sneak path. The goal is to measure the resistance state of memristor M11. The resistance of M11 can be determined from the measurement of the current across the memristor. However, leakage pathways from the surrounding memristors can substantially increase the current and perturb the READ operation. In Fig. 9a, a sold line shows a desired current path, whereas a dash line shows a leakage current path. The leakage path adds resistance in parallel to load resistor RG. The equivalent circuit to the crossbar array in Fig. 9a is depicted in Fig. 9b. The voltage on nanowire BL, VBL, must be ≈VC, but the leakage path reduces this voltage value to less than$ \frac{2}{3}V\_{C}$ and causes an undistinguishable logic state for M11.

VC

M11

RON

RG

M12

M32

VBL

RON

RON

VC HZ HZ

RG

0V

0V

1

1

0

0

1

0

0

VBL

Fig. 9. Sneak path phenomenon. (a) Leakage path way perturb the READ operation. (b) The circuit equivalent to the crossbar array in Fig 9a.

One way to suppress the possible leakage pathways is to fabric a crossbar array utilizing rectifying memristors. A rectify memristor suppress a leakage current when reverse biased. There are other approaches proposed to deal with the sneak path problem which adds more complexity to the circuit [refs]. This is one additional reason that encouraged us to choose rectifying memristors for circuit design.

## An extention to ISD notation

As demonstrated in Section E and F, the process of logic synthesis based on stateful logic gates in a crosspoint array and a crossbar array is sequential. The sequential process can be visualized with ISD notation [ref]. ISD (IMPLY Sequential Diagram) notation visualizes the process of logic synthesis based on IMPY logic system, i.e. IMP and FALSE gates. However, we use INHIBIT logic system for logic synthesis, i.e. INH and TRUE gates, and thus INHIBIT Sequential Diagram (ISD) for visualizing a logic synthesis. In ISD notation, each horizontal line represents a resistance state of one memristor over the course of synthesis. Fig. 10 shows symbols used for stateful logic gates in ISD notation for IMPLY logic system. We extend ISD to include logic synthesis based on INHBIT logic set. Moreover, multi-input multi-output logic gate are included in this notation. We generalize ISD to represent hierarchical blocks, i.e. the generalized ISD visualizes blocks e.g. crosspoint array and crossbar array and their connections.

|  |  |
| --- | --- |
| Logic System | Stateful Gates |
| IMPLY | **(a)**00 | **(b)**PQ$\overbar{p}$+q+ |
| INHIBIT | **(c)**11 | **(d)**PQ$\overbar{p}$q× |
| Multi-input INH | **(e)**PQR$\overbar{r}\overbar{p}$q× | **(f)**PRQ$\overbar{r}\overbar{p}$q× |
| INH with fan-outs | **(f)**QPRS$\overbar{p}\overbar{s}$q×p$\overbar{p}\overbar{s}$r×$$s$$ | **(g)**PQRS$\overbar{p}\overbar{s}$qp××$\overbar{p}\overbar{s}$r$$s$$ |

Fig. 10. Extended ISD symbols. Symbols represent stateful logic gates in extended ISD notation. (a) FALSE gate. (b) Two-input IMP gate. (c) TRUE gate. (d) Two-input INH gate. (e) and (f) Three-input INH gate. Fig. (f) and (g) are examples of INH gate with two fan-outs.

In IMPLY logic system, False gate is denoted by a square with label ‘0’, see Fig 10a. The application of stateful FALSE clears a memristor that is shown by ‘0’ on the horizontal line, a memristor. The symbol of two-input IMP gate is shown in Fig 10b. Line P represents the source memristor, the inverted input, whereas line Q represent the target memristor, the non-inverted input. The output of the gate$,\overbar{p}+q,$ updates the logic state of target memristor Q. In IMP symbol, the ‘+’ sign refers to logic OR, and the ‘black circle’ refers to logic NOT, i.e. the essential logic gates used to define IMPLY logic.

In INHIBIT logic system, TRUE gate is denoted by a square with label ‘1’, see Fig. 10c. The application of stateful TRUE sets a memristor that is shown by ‘1’ on the horizontal line, a memristor. The symbol of two-input INH gate is shown in Fig 10d. Line P represents the source memristor, the inverted input, whereas line Q represent the target memristor, the non-inverted input. The output of the gate$, \overbar{p}q,$ updates the logic state of target memristor Q. In INH symbol, the ‘×’ sign refers to logic AND, and the ‘black circle’ refers to logic NOT, i.e. the essential logic gates used to define INHIBIT logic.

Fig 10e and Fig 10f show symbols of multi-input INH gates, i.e$. OR(p,r)↛q$. Fig 10f and Fig 10g show symbols of INH gates with two fan-outs, i.e. simultaneous implementation of $OR(p,s)↛q$ and$ OR(p,s)↛r$.

As deductive example, consider the parity function discussed in Section F. The process of realization function F in a crossbar array is depicted in Fig 11.

×

1

1

1

1

1

1

×

1

1

1

1

1

1

1

1

×

×

×

$$a$$

M11 M21

M31

M12

M22

M32

M13

M23

M33

$$c$$

$$\overbar{a}\overbar{c}$$

$$\overbar{a}$$

$$\overbar{c}$$

$$ac$$

$$a⨁c$$

$$t\_{1}$$

$$t\_{2}$$

$$t\_{3}$$

$$t\_{4}$$

Fig. 11. ISD representation of two-input parity function in a 3×3 crossbar array.

Remark: In ISD representation shown in Fig. 11, memristor M31 cannot be set at moment t1 with the rest of memristors. Setting M13 at t1requires driving the common vertical nanowire C1 by VCOND, see Fig. 7. Driving C1 by VCOND at t1 forces all elements of column memristor C1 to be set, i.e. M11, M21, and M31. In other word, setting M31 at t1 will also set source memristors, see Fig 8b.

Spread memristors in a crossbar array can be a constraint in logic synthesis, i.e. memristors located in different row/column memristor of a crossbar array can raise the complexity of logic synthesis. This constraint does not exist when a Boolean function is executed in a single crosspoint array. As a result, ISD representation of a Boolean function executed in a crosspoint array does not require a synthesis of the Boolean function. However, if executed in a crossbar array, the map synthesis of the Boolean function is required for ISD representation of the function. For instance, that the map synthesis of function F, shown in Fig. 8, is used for its ISD representation (Fig. 11).

Generalization of ISD will be discussed in Section XX.

In summary, ISD notation is a tool that allows circuit designers to implement their memristive circuit in a crosspoint array more efficiently. This tool clearly shows the tradeoffs between the number of memristors and the number of stateful gates required to implement a memristor-based circuit

# Memristor-Based Volistor Gates Compute Logic

References

[1] Chua LO. Memristor-the missing circuit element. Circuit Theory, IEEE Transactions on. 1971 Sep; 18(5):507-19.

[2] Kuekes, P.: Material implication: digital logic with memristors. In: A Presentation the Memristor and Memristive Systems Symposium at UC Berkeley (2008)

[3] Borghetti J, Snider GS, Kuekes PJ, Yang JJ, Stewart DR, Williams RS. ‘Memristive’ switches enable ‘stateful’ logic operations via material implication. Nature. 2010 Apr 8; 464(7290):873-6.

[4]

[5] Laiho M, Lehtonen E. Cellular nanoscale network cell with memristors for local implication logic and synapses. In Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on 2010 May 30 (pp. 2051-2054). IEEE.

[6] Kim KH, Hyun Jo S, Gaba S, Lu W. Nanoscale resistive memory with intrinsic diode characteristics and long endurance. Applied Physics Letters. 2010 Feb 1; 96(5):053106.

[7] Kim KH, Gaba S, Wheeler D, Cruz-Albrecht JM, Hussain T, Srinivasa N, Lu W. A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications. Nano letters. 2011 Dec 9; 12(1):389-95.

[8] Lehtonen E, Poikonen JH, Laiho M. Applications and limitations of memristive implication logic. InCellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on 2012 Aug 29 (pp. 1-6). IEEE.

[9] Kim K, Shin S, Kang SM. Field programmable stateful logic array. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on. 2011 Dec; 30(12):1800-13.

[10]