Noname manuscript No.
(will be inserted by the editor)

An Efficient Memristive Alternating Crossbar Array and The Design of Full Adder

Meiqi Jiang · Jingru Sun · Chunhua Wang · Ziyao Liao · Yichuang Sun · Qinghui Hong · Jiliang Zhang

Received: date / Accepted: date

Abstract Memristor is one of the most promising emerging technologies to solve the von Neumann bottleneck problem due to its non-volatile and binary characteristics. This paper studies the design method of high-efficiency logic circuit based on memristor. First, a Multiple Input Multiple Output (MIMO) logic circuit design scheme based on IMPLY and AND logic is proposed, which can derive multiple new efficient logic operation methods and complete complex logic with fewer steps and memristors. Second, in order to perform rapid interactive operations between different rows, an alternating crossbar array structure is designed which can quickly complete cross-row logic operations. Finally, a high-efficient Full Adder (FA) based on MIMO logic and alternating crossbar array is proposed. To accomplish 32-bit add operation, the proposed FA needs 160 memristors and only 41 steps. Compared with the state of art FA, our work has faster execution speed and fewer memristors.

Keywords Memristor · IMPLY logic · Crossbar array · Multi-input logic · Multi-output logic · Full adder

M. Jiang
College of Computer Science and Electronic Engineering, Hunan University, Changsha, 410082, People's Republic of China
E-mail: meiqij@hnu.edu.cn

J. Sun
College of Computer Science and Electronic Engineering, Hunan University, Changsha, 410082, People's Republic of China
E-mail: jt_sunjr@hnu.edu.cn

C. Wang
College of Computer Science and Electronic Engineering, Hunan University, Changsha, 410082, People's Republic of China
E-mail: wch1227164@hnu.edu.cn

Z. Liao
College of Electronic Science and Technology, Shanghai Jiao Tong University, Shanghai, 200030, People's Republic of China
E-mail: 1911538379@qq.com

Y. Sun
School of Engineering and Computer Science, University of Hertfordshire, Hatfield, AL10 9AB, United Kingdom.
E-mail: y.sun@herts.ac.uk

Q. Hong
College of Computer Science and Electronic Engineering, Hunan University, Changsha, 410082, People's Republic of China
E-mail: hongqinghui@hnu.edu.cn

J. Zhang
College of Semiconductors, Hunan University, Changsha, 410082, People's Republic of China
E-mail: zhangjiliang@hnu.edu.cn

1 Introduction

Memristor was first proposed by Leon Chua in 1971 [1, 2], and realized in physical form in 2008. The most outstanding characteristic of the memristor is nanoscale dimensions, nonvolatile, low power and multi-state programming. Thus the early application research of the memristor is nonvolatile memory [3–5]. With the development of research, the application has been extended to neuromorphic network [6, 7] and chaotic circuit design [8–12]. Especially in the design of logic circuits, the memristor can realize the processing in memory (PIM) architecture, and is considered to be one of the most promising candidate technologies to break through the von Neumann bottleneck [13–16].

According to the components, the logic circuits based on memristors can be divided into two categories. One is the hybrid CMOS-memristor-based logic, which mixes

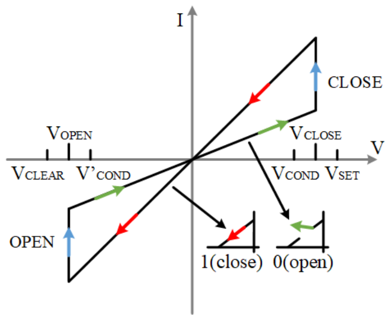


Fig. 1 I-V characteristic curve of ideal memristor model

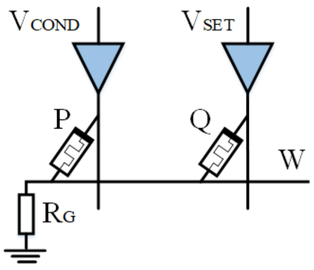


Fig. 2 IMPLY logic operation circuit

Table 1 Truth Table of IMPLY Logic Operation

	P	Q	Result (output to Q)
case 1	0	0	1
case 2	0	1	1
case 3	1	0	0
case 4	1	1	1

the CMOS logic and memristive logic, such as the memristor ratioed logic (MRL) [17] and memristor-based threshold logic gates(TLGs) [18]. The other one is memristor-only logic, which includes stateful logic and sequential logic. The characteristic of sequential logic [19] is that input and output can be represented by different variables. While the logic input and output of stateful logic [20] are both represented by resistance, such as Memristor Aided Logic (MAGIC) [21] and Material Implication (IMPLY) [22, 23], which are both built based on crossbar arrays structure. IMPLY is simple and reliable, and has the ability to implement complete Boolean logic operations [24], and therefore is more widely applied [25–27]. As shown in Fig. 1, it is the I-V characteristic curve of the ideal memristor model. V_{CLOSE} and V_{OPEN} represent the positive threshold voltage and negative threshold voltage of the memristor respectively. V_{SET} is slightly larger than the positive threshold voltage, when acting alone, the resistance state of the memristor will be switched to the low-resistance-state (LRS) R_{ON}, representing a logic 1. V_{CLEAR} is slightly smaller than the negative threshold voltage, when acting alone, the resistance state of the memristor will be switched to the high-resistance-state (HRS) R_{OFF}, representing a logic 1. V'_{COND} and V_{COND} do not reach the threshold voltage, so the resistance state of memristor will not be changed when they act alone. The IMPLY logic circuit can be built by using the resistance variation characteristics of memristors. The circuit and truth table of an IMPLY logic is shown in Fig. 2 and Table I. The resistance of the memristor represents the logical state, where low-resistance-state (LRS) R_{ON} is considered as logic 1, and high-resistance-state (HRS) R_{OFF} is considered as logic 0.

However, there are two problems when IMPLY logic is used for complex logic operations. First, the execution efficiency of IMPLY is low, and more operation steps are required when performing complex logic operations. Second, the input of IMPLY will be overwritten. As shown in Fig. 2, IMPLY logic has only two computational memristors, while logical operations generally have two or more operands. Therefore, we have to input the original data into memristor Q, whose logic state will be overwritten by the operation result. If the input value is required to be used again, it has to be transferred to another place before the operation.

In order to solve the above problems, Shin et al. proposed a high-fan-in NOR gate [28]. It can execute multiple implications simultaneously in one step. Later, [29] proposed a 3M1R NAND logic and AND logic. This structure contains 2 input memristors and 1 output memristor, and its input value will not be overwritten and the execution efficiency will be improved. [30] proposed an ORNOR logic gate, which incluedes 3 input memristors, and 1 as output at the same time. This logic can complete NOR logic and OR logic in 1 step.

The computing system includes a number of fundamental blocks. The efficiency of these fundamental blocks has an important impact on the execution efficiency of the computer system. Among them, the FA is one of the most frequently used blocks. Therefore, the design of FA circuit based on memristor has become a hot research topic.

The parallel approach is a common structure in FA design. As shown in Fig. 3, each bit represents a different row with its related working memristors, and different rows can execute simultaneously, so this approach can effectively save operation steps. However, the number of memristors is greatly increased in parallel structure. For example, the 32-bit FA proposed by Kvatinsky [26] needs 178 operation steps and 288 memristors. 3M1R based 32-bit FA [29] needs 104 operation steps and 288 memristors. ORNOR based 32-bit FA [30] needs 79 operation steps and 198 memristors, better than the above two.

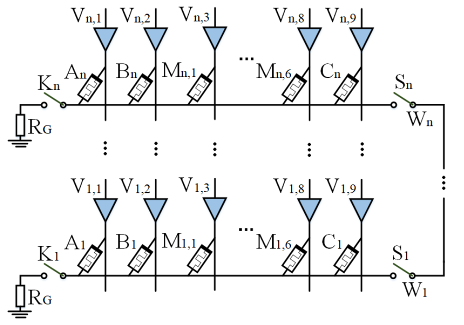


Fig. 3 The n-bit FA using parallel structure. Each row calculates 1 bit. A_i and B_i are addend and summand. $M_{i,j}$ is the register during operations. C_i is a carry. And the standard sum can be stored anywhere except C_i, depending on the specific algorithm of the designer

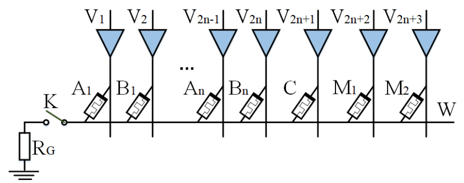


Fig. 4 The n-bit FA using serial structure. All calculations are on the same row. A_i and B_i are addend and summand. M_1 and M_2 are the registers during operations. C is a carry. The out result is saved in A_i or B_i

Serial FA [31] is another common structure, as shown in Fig. 4, all memristors are connected to the ground via a resistor. Voltage V_i is applied to the corresponding single memristor, and each time only one operation can be executed. The advantage of this structure is small area, but at the cost of increasing the total steps. For example, a 32-bit serial FA requires only 67 memristors, but 704 steps. Then a semiparallel structure FA is proposed [27], which has the same number of memristors as the serial method, but the steps of the 32-bit FA are reduced to 544. Another improved FA is the semi-series structure [32]. In this structure, each bit is calculated serially, but the working memristor is arranged in a separate third part, thereby achieving part parallelism, for a 32-bit FA, which needs 70 memristors and 322 steps. Above all, compared with parallel approach, the improved serial method will still require more steps.

To make calculations more efficient, based on Shin's work [28], we propose a Multiple Input Multiple Output (MIMO) logic circuit design scheme. The MIMO scheme contains Multi-input logic and Multi-output logic, which exhibits high computational efficiency and provides data reusability. Then, to improve the efficiency of memristors interaction between different rows in the traditional crossbar array structure, we propose an alternating crossbar array structure, which makes it possible to complete the carry operation in FA in one step. Finally, a highly efficient FA based on MIMO logic circuit design scheme and alternating crossbar array structure is proposed. Compared with the existing FAs, the proposed FA greatly reduces the operation steps in all FAs and the number of memristors in parallel FAs. Because of fewer memristors and operation steps, it consumes less power when calculating the addition operation of the same number of bits.

The main contributions of this work are as below:
1) The proposed MIMO logic circuit design scheme can improve the efficiency of computation, avoid the overwriting of inputs and extend the application of output.
2) The proposed alternating crossbar array structure has higher computational efficiency than traditional crossbar array in cross-row logic operations.
3) The proposed FA is faster than all other designs, requires less area than other parallel designs, and has the reusability of data.

The rest of this paper is organized as follows: In Section 2, we introduce the memristor model and two basic memristive logic. In Section 3, the proposed MIMO logic circuit design scheme is described in detail. The proposed alternating crossbar array structure and FA are presented in Section 4. Section 5 shows the correctness of our design through PSpice simulation. In Section 6, comparisons between different FAs are presented. The paper is summarized in Section 7.

2 Background

2.1 Memristor Model

This paper uses the memristor model as Drift Speed Adaptive Memristor (DSAM) [33]. The DSAM model is based on three main characteristics, namely, linear I-V relationship, drift speed adaptive control, and a voltage threshold, which can well describe the characteristics of you and the ideal memristor. By adjusting the fitting parameters, a variety of state variable curves are provided, which makes it possible to describe different memristors. Moreover, it can simultaneously satisfy boundary validity, scalability, nonlinearity and solve the problem of boundary lock. The DSAM model is very sensitive and accurate to the conduction curve under impulse excitation. This allows us to reach an ideal state in logical operation. Our work is mainly to use the binary characteristics of memristor to carry out logical operations, We adjusted the parameters of this model and found that various logic operations can be realized quickly and stably under the action of excitation voltage. And its $I-V$ relationship is

$$v(t) = (R_{OFF} - x\Delta R) \times i(t), \tag{1}$$

where the state variable x represents the normalized broadband of the conductive area, and its range is $[0, 1]$. In addition, $\Delta R = (R_{OFF} - R_{ON})$, and R_{OFF} and R_{ON} represent the high resistance state (HRS) and low resistance state (LRS) of the memristor. And the corresponding state variable is $x = 1$ and $x = 0$. Therefore, the derivative of the state variable x can be expressed as follows

$$\frac{dx}{dt} = \begin{cases} k_{on} \times \Delta R \times i(t) \times f(x), & v(t) > v_{on}, \\ 0, & v_{off} \leq v(t) \leq v_{on}, \\ k_{off} \times \Delta R \times i(t) \times f(x), & v(t) < v_{off}, \end{cases} \tag{2}$$

where

$$f(x) = \begin{cases} (a \times (1 - x))^p, & v(t) > 0, \\ (a \times x)^p, & v(t) \leq 0. \end{cases} \tag{3}$$

Among them, v_{on} and v_{off} represent the positive and negative threshold voltages, a and p are curve fitting parameters, k_{on} and k_{off} are linear adjustable parameters.

Generally, LRS is considered as logic 1(close), and HRS is considered as logic 0 (open). The I-V characteristic curve of the ideal memristor model is shown in Fig. 1, where $V_{CLEAR} < V'_{COND} < V_{COND} < V_{SET}$. When the applied voltage is greater than the positive threshold voltage V_{CLOSE} of memristor, the memristor switches from state 0 to state 1; when the applied voltage is less than the negative threshold voltage V_{OPEN} of memristor, the memristor switches from state 1 to state 0.

2.2 IMPLY Logic Operation

P IMPLY Q is a logic operation called material implication (IMPLY), which can be realized by memristive IMPLY logic operation circuit. As shown in Fig. 2, P and Q are two memristors, which are connected to the resistor R_G through the horizontal nanowire L. R_G is grounded. In order to understand the principle, we simply assume that the parameters of memristors need to satisfy $R_{ON} \ll R_G \ll R_{OFF}$. By applying two fixed voltages V_{SET} and V_{COND} to memristor P and Q respectively through three-state buffer, IMPLY operation can be achieved, which is $q = \overline{p} + q$. The logical value of memristor Q is replaced by the operation result, and the value of memristor P stays unchanged.

When the state of memristor P is HRS (p=0), the voltage on R_G is almost 0 due to $R_{OFF} \gg R_G$. So the voltage dropped on memristor Q is $V_Q \approx V_{SET} > V_{CLOSE}$. Therefore, no matter what the state of memristor Q is, it will convert to LRS (q = 1). The above analysis corresponds to case 1 and case 2 of the Table I.

When memristor P is LRS (p=1), the voltage on R_G is approximately equal to V_{COND} due to $R_G \gg R_{ON}$. So the voltage dropped on the memristor Q is $V_Q \approx V_{SET} - V_{COND} < V_{CLOSE}$. In this case, Q will remain in its original state. The above analysis corresponds to case 3 and case 4 of the Table I.

Above is a brief introduction to the principle, and next we will strictly deduce the formula. According to Kirchhoff's Current Law, the voltage dropped in the memristors P and Q are

$$V_P = \frac{(R_Q + R_G)\, V_{COND} - R_G V_{SET}}{R_Q + R_G\,(1 + R_Q/R_P)}, \tag{4}$$

$$V_Q = \frac{(R_P + R_G)\, V_{SET} - R_G V_{COND}}{R_P + R_G\,(1 + R_P/R_Q)}. \tag{5}$$

In order to ensure that the function of IMPLY logic operation is correct, in case 1, V_Q must be greater than V_{CLOSE}, and in case 3, V_Q must not be greater than V_{CLOSE}. Then we get two inequalities, for case 1:

$$V_Q = \frac{(R_{OFF} + R_G)\, V_{SET} - R_G V_{COND}}{R_{OFF} + R_G\,(1 + R_{OFF}/R_{OFF})} > V_{CLOSE}, \tag{6}$$

and for case 3:

$$V_Q = \frac{(R_{ON} + R_G)\, V_{SET} - R_G V_{COND}}{R_{ON} + R_G\,(1 + R_{ON}/R_{OFF})} < V_{CLOSE}. \tag{7}$$

In any case, the resistance state of the memristor P remains unchanged. Replace R_P and R_Q in (4) with the correspond state in case 1-4 respectively. Then we can get four inequalities, for case 1:

$$V_P = \frac{(R_{OFF} + R_G)\, V_{COND} - R_G V_{SET}}{R_{OFF} + R_G\,(1 + R_{OFF}/R_{OFF})} < V_{CLOSE}. \tag{8}$$

The inequalities of the other 3 cases can be obtained according to the same principle.

By combining $V_{COND} < V_{CLOSE} < V_{SET}$ with the above inequalities, the expressions of the upper and lower bounds of R_G are obtained as

$$R_G < \frac{R_{OFF}\,(V_{SET} - V_{CLOSE})}{2V_{CLOSE} - (V_{SET} - V_{COND})}, \tag{9}$$

$$R_G \geq \frac{R_{ON}\,(V_{SET} - V_{CLOSE})}{V_{CLOSE}(1 + R_{ON}/R_{OFF}) - (V_{SET} - V_{COND})}, \tag{10}$$

where

$$V_{SET} - V_{COND} < V_{CLOSE}\,(1 + R_{ON}/R_{OFF})\,. \tag{11}$$

Table 2 Truth Table of AND Logic Operation

	P	Q	Result (output to Q)
case 1	0	0	0
case 2	0	1	0
case 3	1	0	0
case 4	1	1	1

Therefore, if inequalities 8, 9 and 10 are satisfied, IMPLY operation can be realized. We notice that IMPLY operation can also be achieved when R_G does not satisfy $R_{ON} \ll R_G \ll R_{OFF}$.

2.3 AND Logic Operation

The principle and circuit structure of AND operation are similar to IMPLY operation. The truth table of AND is shown in Table II. When applying two fixed voltages V'_{COND} and V_{CLEAR} to the memristor P and Q respectively, AND operation (q = p · q) can be achieved. The logical value of memristor Q is replaced by the operating result, and the state of memristor P remains unchanged.

In order to make the function of AND logic operation correct, in case 2, V_Q must be less than V_{OPEN}, and in case 4, V_Q must be not less than V_{OPEN}. The resistance state of memristor P remains unchanged in any circumstances. Similar to IMPLY logic, the value range of R_G is

$$R_G < \frac{R_{OFF}\,(V_{CLEAR} - V_{OPEN})}{V_{OPEN}(1+R_{OFF}/R_{ON}) - (V_{CLEAR} - V'_{COND})}, \tag{12}$$

$$R_G \geq \frac{R_{ON}\,(V_{CLEAR} - V_{OPEN})}{2V_{OPEN} - (V_{CLEAR} - V'_{COND})}, \tag{13}$$

where

$$V_{CLEAR} - V'_{COND} > 2V_{OPEN}. \tag{14}$$

3 MIMO Scheme Based on Memristive Logic

In this section, a systematic and more efficient MIMO memristive logic scheme is proposed. The MIMO scheme consists of two parts: multi-input logic and multi-output logic. Each part contains two improvements to the basic memristive logic. Applying them to the design of complex logic will effectively improve the computational efficiency and save the original input.

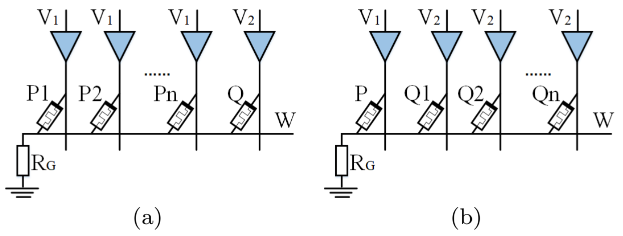


Fig. 5 (a) Circuit model of Multi-input logic operation. (b) Circuit model of Multi-output logic operation.

3.1 Circuit Model of MIMO Logic

IMPLY and AND logic have the same circuit structure but different excitation voltages. In Fig. 2, memristor P and Q are regarded as input memristor and output memristor respectively.

3.1.1 Multi-input Logic

Fig. 5(a) shows the circuit model of multi-input logic, in which the input memristors can be extended to n. When $\{V_1, V_2\} = \{V_{COND}, V_{SET}\}$, corresponding to Multi-input IMPLY logic, the logic operation can be expressed as $q = \overline{(p_1 + p_2 + \ldots + p_n)} + q$. When $\{V_1, V_2\} = \{V'_{COND}, V_{CLEAR}\}$, corresponding to Multi-input AND logic, the logic operation can be expressed as $q = (p_1 + p_2 + \ldots + p_n) \cdot q$. q is output, and is set to 0 or 1 according to the logical operation.

Since Multi-input logic has more than one input memristor, we can complete operations involving multiple data in one step. Using Multi-input logic to design complex logic can effectively reduce computation time and the number of memristors. When we input the original data only to memristors $P_1 \sim P_n$ (memristor Q does not store the input data), the data can avoid being overwritten by operation result as what IMPLY and AND logic do. So the Multi-input logic has data reusability.

3.1.2 Multi-output Logic

Fig. 5(b) shows the circuit model of Multi-output logic, in which the output memristors can be extended to n. The initial logic values of Q_1, Q_2 up to Q_n should be ensured the same before calculation. The Multi-output logic operation results are stored on n memristors. When $\{V_1, V_2\} = \{V_{COND}, V_{SET}\}$, corresponding to Multi-output IMPLY logic, the logic operation can be expressed as $q_1 = q_2 = \ldots = q_n = \bar{p} + q$. When $\{V_1, V_2\} = \{V'_{COND}, V_{CLEAR}\}$, corresponding to Multi-output AND logic, the logic operation can be expressed as $q_1 = q_2 = \ldots = q_n = p \cdot q$.

By using Multi-output logic, the operation results can be stored in multiple memristors, which is convenient for the output data to participate in different operations and improves the execution efficiency. In addition, if the specific conditions of R_G are satisfied, the Multi-input and Multi-output logic operations can be realized in the same structure at the same time.

3.2 Constraints of MIMO Logic

To execute MIMO logic correctly, circuit parameters should satisfy the following constraints.

3.2.1 Multi-input Logic

The parallel resistance value of input memristors is defined as R_i. When the resistance states of all input memristors are HRS, the parallel resistance is defined as logic 0, and the rest cases are all defined as logic 1. The resistance corresponding to the logic value is

$$R_i = \begin{cases} \frac{R_{OFF}}{n}, & logic\ 0, \\ \left[\frac{R_{ON}}{n}, \frac{R_{OFF}R_{ON}}{R_{OFF}+R_{ON}(n-1)}\right], & logic\ 1. \end{cases} \tag{15}$$

Similar to the calculation method of IMPLY logic, the constraints of the Multi-input IMPLY logic can be obtained as follows:

$$R_G < \frac{R_{OFF}\left(V_{SET} - V_{CLOSE}\right)}{(n+1)\,V_{CLOSE} - n\left(V_{SET} - V_{COND}\right)}, \tag{16}$$

$$R_G \geq \frac{\frac{R_{OFF}R_{ON}}{R_{OFF}+(n-1)R_{ON}}\left(V_{SET} - V_{CLOSE}\right)}{\frac{R_{OFF}+nR_{ON}}{R_{OFF}+(n-1)R_{ON}}V_{CLOSE} - \left(V_{SET} - V_{COND}\right)}, \tag{17}$$

where

$$V_{SET} - V_{COND} < \frac{R_{OFF} + nR_{ON}}{R_{OFF} + (n-1)\,R_{ON}}V_{CLOSE}. \tag{18}$$

The constraints of Multi-input AND logic can be obtained by the same method.

3.2.2 Multi-output Logic

The parallel resistance value of output memristors is defined as R_o. Because the logic value of all output memristors should be kept the same before calculation, there are only two cases of R_o value.

$$R_O = \begin{cases} \frac{R_{OFF}}{n} & logic\ 0, \\ \frac{R_{ON}}{n} & logic\ 1. \end{cases} \tag{19}$$

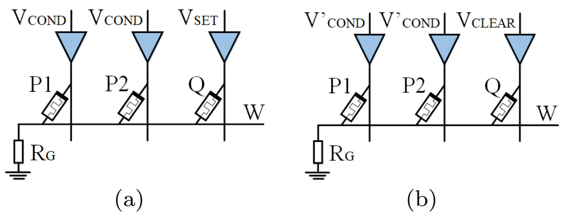


Fig. 6 (a) Circuit implementation of ONO logic operation. (b) Circuit implementation of OA logic operation.

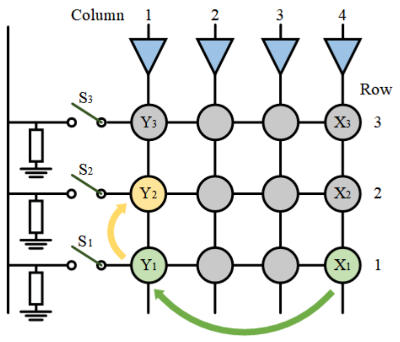


Fig. 7 IMPLY operation of different rows in traditional crossbar array structure.

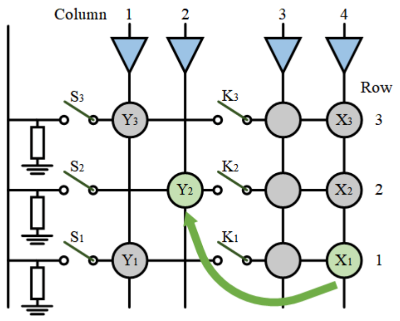


Fig. 8 IMPLY operation of different rows in alternating crossbar array structure.

Similar to the calculation method of IMPLY logic, the constraints of the Multi-output IMPLY logic can be obtained as follows:

$$R_G < \frac{R_{OFF}\left(V_{SET} - V_{CLOSE}\right)}{(n+1)\,V_{CLOSE} - \left(V_{SET} - V_{COND}\right)}, \tag{20}$$

$$R_G \geq \frac{R_{ON}\left(V_{SET} - V_{CLOSE}\right)}{V_{CLOSE}(1+nR_{ON}/R_{OFF}) - (V_{SET} - V_{COND})}, \tag{21}$$

where

$$V_{SET} - V_{COND} < V_{CLOSE}\left(1 + nR_{ON}/R_{OFF}\right). \tag{22}$$

The constraints of the Multi-output AND logic can be obtained by the same method.

Table 3 Truth Table of ONO Logic Operations

	P1	P2	Q	Result (output to Q)
case 1	0	0	q	1
case 2	0	1	q	q
case 3	1	0	q	q
case 4	1	1	q	q

Table 4 Truth Table of OA Logic Operations

	P1	P2	Q	Result (output to Q)
case 1	0	0	q	0
case 2	0	1	q	q
case 3	1	0	q	q
case 4	1	1	q	q

3.3 ONO and OA Logic

For the convenience of later description, we redefine the 2-input IMPLY and 2-input AND as OR-NOT-OR (ONO) and OR-AND (OA) respectively. The ONO logic can be expressed as $q = \overline{p_1 + p_2} + q$. Its circuit implementation is shown in Fig. 6(a), and the true value table of ONO logic is shown in Table III. The OA logic can be expressed as $q = (p_1 + p_2) \cdot q$. Its circuit implementation is shown in Fig. 6(b), and the true value table of OA logic is shown in Table IV.

4 Alternating Crossbar Array FA

4.1 Alternating Crossbar Array

Fig. 7 shows the structure of a traditional crossbar array. In this structure, when the logic operations involve the memristors of different rows, they usually require multiple steps to complete. For example, computing X_1 IMPLY Y_2 takes two steps. First, X_1 is copied into Y_1 by horizontal AND operation, and then Y_1 IMPLY Y_2 by vertical operation.

To achieve rapid data interaction between different rows in a crossbar array structure, as shown in Fig. 8, an alternating crossbar array structure is proposed. It differs from the traditional structure in two points. First, the memristors in columns 1 and 2 are placed alternately to avoid interference from the memristors of adjacent rows. Second, a column of switches is added to isolate the interference of the same row memristors. By controlling switches S_i and K_i, logical operations in different rows and columns can be realized quickly.

For a better understanding, here is an example. In Fig. 8, the IMPLY logical operation is performed on X_1 and Y_2. X_1 is in row 1, column 4, and Y_2 is in row 2, column 2. The following operations need to be performed simultaneously:

- By closing switches S_1 and S_2 and opening other S-Series switches, rows 1 and 2 are selected.
- By closing switch K_1 and opening other K-series switches, the part behind K_1 in row 1 is connected, and the part behind K_2 in row 2 is separated.
- V_{SET} is applied to column 2 and V_{COND} is applied to column 4, so that the memristors selected for the IMPLY operation contain only X_1 and Y_2.

Therefore, the IMPLY operation between adjacent rows in an alternating crossbar array can be completed in one step, and there is no need to move the two operators to the same row or the same column. When the calculation involves the memristors of adjacent rows, the alternating crossbar array is faster than the traditional crossbar array. The application of alternating crossbar array makes it possible to complete the carry operation in FA in one step, which effectively improves computational efficiency.

Table 5 Truth Table of OA

IN C_{i-1}	IN A_i	IN B_i	OUT S_i	OUT C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

4.2 Proposed FA

In this section, the algorithm and structure of FA are proposed based on the MIMO logic and alternating crossbar array mentioned above.

The basic ADD logic is described as follows, A_i is the addend, B_i is the summand, C_{i-1} is the carry-in from the adjacent lower bit, S_i is the sum of the current bit, and C_i is the carry-out to the adjacent higher bit. The truth table of one-bit FA is shown in Table V.

Sum (S_i) and Carry-out (C_i) are calculated by

$$S_i = (A_i \oplus B_i) \oplus C_{i-1} \tag{23}$$

$$C_i = A_i \cdot B_i + C_{i-1} \cdot (A_i + B_i) \tag{24}$$

$$\overline{C_i} = \left(\overline{A_i} + \overline{B_i}\right) \cdot \left(\overline{A_i + B_i} + \overline{C_{i-1}}\right), \tag{25}$$

where

$$A_i \oplus B_i = (A_i + B_i) \cdot \left(\overline{A_i} + \overline{B_i}\right). \tag{26}$$

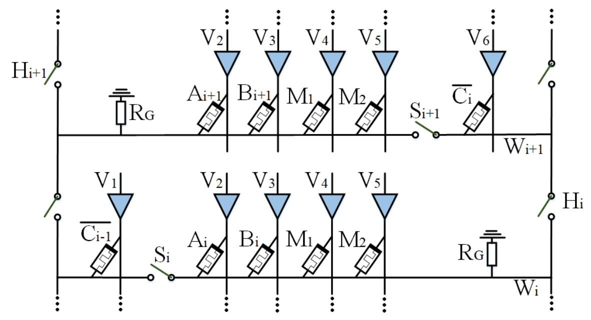


Fig. 9 The proposed FA circuit.

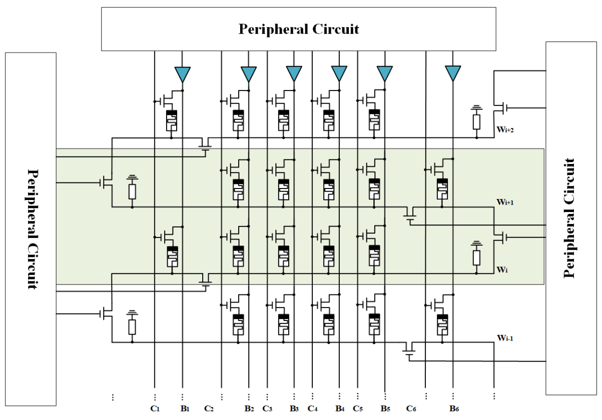


Fig. 10 The implementation of 1T1R crossbar array of the proposed FA.

The proposed FA circuit is shown in Fig. 9. The scale of the circuit can be expanded to n bits. Only two bits of the FA are shown here for the convenience of illustration.

In the i-th bit, memristors A_i and B_i store the input logic values. Memristor $\overline{C_{i-1}}$ stores the reversed carry-out, which comes from the adjacent lower bit. Memristors M_1 and M_2 are used to store the intermediate process. The control circuit, which is not discussed in this paper, enables all inputs to occur simultaneously. Memristor $\overline{C_i}$ is located in the (i+1)-th bit, which is used to store the reversed carry-out of the i-th bit. In our algorithm, the carry-out of each bit is the reversed logical value, and the sum of current bit is stored in the memristor M_2.

The implementation of crossbar array of the proposed FA is shown in Fig. 10. In a large-scale array, we need to ensure that there are memristors placed alternately on both sides of the array and at least four columns of memristors in the middle. The structure in the dotted box corresponds to Fig. 9. The memristors of the first and last columns in the crossbar array are alternating, corresponding to the alternating crossbar array. Resistors R_G and switches S_i are also alternating in order to complete the carry operation correctly. Some switches in Fig. 9 are not marked because they are always open in the algorithm in this paper. We use the 1T1R crossbar arrays [4] to solve the sneak current issue. The detailed connections of word line W, bit line B and control signal C are shown in Fig. 10.

The operations of each step and the state of the memristors after the operations are recorded in Table VI. According to Table VI, one-bit FA calculation needs 10 steps. The excitation voltage not mentioned in each step in the table is 0. Steps 2, 5, 7 and 10 use Multi-input logic, and steps 3 and 4 use Multi-output logic. Note that in steps 1, 3, 4, 9 and 10, both V_1 and V_6 apply the same voltage. Because in parallel operation mode, both columns 1 and 6 can store carry-in and carry-out. And in step 5, in the serial operation mode, the position of the memristor storing the carry-in and carry-out depends on the bit position i of the FA. Therefore, the voltage values of V_1 and V_6 depend on whether i is odd or even.

In order to show the working mode of this work in the case of n-bit, 10 operating steps are divided into 4 phases:

Phase 1: steps 1 and 2. Close switches $S_2 \sim S_n$. Open switches S_1 and $H_0 \sim H_n$. Note that S_1 needs to be open because we need to keep the original carry-in $\overline{C_0}$ unchanged in step 1. The n-bit FA can simultaneously calculate steps 1 and 2 in parallel. In this phase, the n-bit FA needs only 2 steps in parallel calculation.

Phase 2: steps 3 and 4. Close switches $H_1 \sim H_n$. Open switches $S_1 \sim S_n$. In this way, $\overline{C_i}$ can be connected to W_i without affecting other bits, so data transmission can be realized between the i-th and (i+1)-th bit positions. Similar to Phase 1, the n-bit addition can be computed in parallel to complete Phase 2 in parallel calculation with 2 steps.

Phase 3: step 5. Close switches S_i and H_i. Open all other switches. To calculate the carry-out of the i-th bit position, we must first get the carry-in from the i-th. Therefore, the n-bit FA can only complete Phase 3 by serial calculation. In this phase, 1 step is needed for each bit but n steps for n-bit in serial calculation.

Phase 4: from step 6 to step 10. Close switches $S_1 \sim S_n$. Open switches $H_0 \sim H_n$. After completing Phase 3, each FA receives a carry-out from the adjacent lower bit, and each FA can complete all the remaining steps independently. So n-bit FA can complete Phase 4 in parallel with 5 steps.

The operation diagram of the n-bit FA is shown in Fig. 11. In Phases 1 and 2, all bits execute in parallel, which takes 2 steps; In Phase 3, each bit is executed in serial immediately after the previous bit, which takes n steps; In Phase 4, all bits execute in parallel as in Phases 1 and 2, which takes 5 steps.

Table 6 The Implementation of i-th FA

Step	Operation	Voltage	The logical value after operation in:		
			M_1	M_2	$\overline{C_i}$
1	$CLEAR\ (M_1,\ M_2,\ \overline{C_i})$	$V_1 = V_4 = V_5 = V_6 = V_{CLEAR}$	0	0	0
2	$(A_i,\ B_i)\ ONO\ M_1$	$V_2 = V_3 = V_{COND};\ V_4 = V_{SET}$	$\overline{A_i + B_i}$	0	0
3	$B_i\ IMPLY\ (M_2,\ \overline{C_i})$	$V_3 = V_{COND};\ V_1 = V_5 = V_6 = V_{SET}$	$\overline{A_i + B_i}$	$\overline{B_i}$	$\overline{B_i}$
4	$A_i\ IMPLY\ (M_2,\ \overline{C_i})$	$V_2 = V_{COND};\ V_1 = V_5 = V_6 = V_{SET}$	$\overline{A_i + B_i}$	$\overline{A_i} + \overline{B_i}$	$\overline{A_i} + \overline{B_i}$
5	$(\overline{C_{i-1}},\ M_1)\ OA\ \overline{C_i}$	$\begin{cases} V_1 = V_4 = V'_{COND};\ V_6 = V_{CLEAR}\ \ i \in odd \\ V_4 = V_6 = V'_{COND};\ V_1 = V_{CLEAR}\ i \in even \end{cases}$	$\overline{A_i + B_i}$	$\overline{A_i} + \overline{B_i}$	$\overline{C_i}$ (carry-out)
6	$CLEAR\ M_1$	$V_4 = V_{CLEAR}$	0	$\overline{A_i} + \overline{B_i}$	-
7	$(A_i,\ B_i)\ OA\ M_2$	$V_2 = V_3 = V'_{COND};\ V_5 = V_{CLEAR}$	0	$A_i \oplus B_i$	-
8	$M_2\ IMPLY\ M_1$	$V_5 = V_{COND};\ V_4 = V_{SET}$	$\overline{A_i \oplus B_i}$	$A_i \oplus B_i$	-
9	$\overline{C_{i-1}}\ IMPLY\ M_2$	$V_1 = V_6 = V_{COND};\ V_5 = V_{SET}$	$\overline{A_i \oplus B_i}$	$C_{i-1} + A_i \oplus B_i$	-
10	$(\overline{C_{i-1}},\ M_1)\ OA\ M_2$	$V_1 = V_4 = V_6 = V'_{COND};\ V_5 = V_{CLEAR}$	$\overline{A_i \oplus B_i}$	S_i (sum)	-

i represents the bit position of the FA.

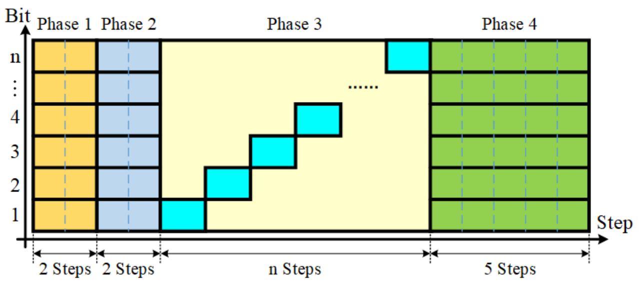


Fig. 11 The operation diagram of the n-bit FA.

To sum up, for the n-bit FA, the proposed design requires $2 + 2 + n + 5 = n + 9$ steps. In this part, we propose the design of FA and make a preliminary analysis. As shown in Table VI, the Multi-input and Multi-output logics are used multiple times in the FA. It can reduce the calculation steps. The application of the alternating crossbar array enables the carry operation to be completed in one step, which greatly improves the computational efficiency of the FA.

In addition to improving the computational efficiency, our algorithm and structure also reduce the number of memristors and make our design have higher integration. Furthermore, instead of performing IMPLY or AND logic operations on the original input memristors, we use Multi-input logic operations with data reusability to retain the original input data. So the proposed FA also has data reusability. Comparisons of speed, area, power consumption, and data reusability are presented in Section VI.

4.3 The Peripheral Circuit

Because of the need of the algorithm, $V_1 \sim V_6$ need to input different voltages in different operations. To realize this operation, this paper has written a peripheral circuit program circuit by Verilog HDL language, which can output different pulse voltages at different times. For example, we can get from Table VI that the voltage on the bit line V_1 has five possibilities -1.2V, -0.8V, 0V, 0.8V, 1.2V at different times. Add five switched voltages to V_1, which are -1.2V, -0.8V, 0V, 0.8V, 1.2V respectively. As long as the clock control switch of Verilog HDL is used, the input of different voltages can be achieved. After many simulation tests, the step time of 10 ns can realize the function of the corresponding steps. Other switches in the circuit are also connected to the peripheral circuits on both sides, and are similarly controlled by Verilog HDL clock.

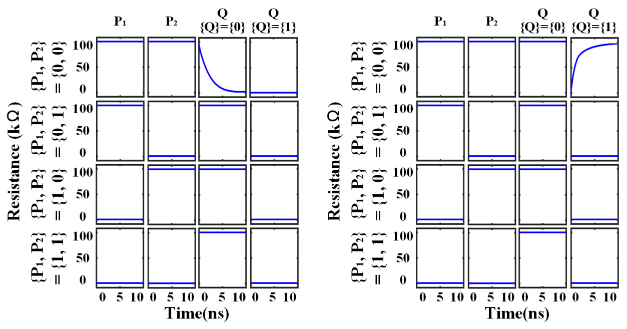
However, there is a small problem with this peripheral circuit. The periphery of adders with different bits cannot be used universally. To solve this problem, we propose two methods. One is that it can be set as a fixed adder and the other is to design multiple peripheral circuits, and choose different peripheral circuits when making corresponding different bits calculations.

5 Simulation

To verify the correctness of MIMO logic and the calculation accuracy of FA, we simulated them with PSpice. The DSAM memristor model is used in the design, and the main parameters that constrain the implementation of logic operations are listed in Table VII. According to (9-14, 16-18, 20-22), the difference between R_{OFF} and

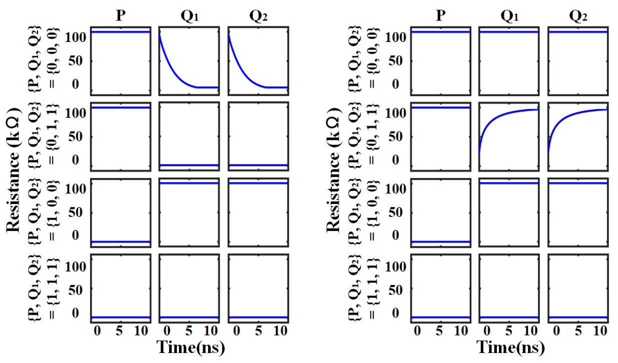
Table 7 Memristors and circuit parameters considered in the simulations

a	p	k_{on}	k_{off}	V_{CLOSE}	V_{OPEN}	R_{ON}	R_{OFF}	V_{SET}	V_{COND}	V_{CLEAR}	V'_{COND}	R_G
2.1	1.8	8000	5000	$1V$	$-1V$	$1k\Omega$	$100k\Omega$	$1.2V$	$0.8V$	$-1.2V$	$-0.8V$	500Ω



(a) (b)

Fig. 12 The change of each memristor resistance in ONO logic (a) and OA logic (b).



(a) (b)

Fig. 13 The change of each memristor resistance in two-output IMPLY logic (a) and AND logic (b).

R_{ON} has a great influence on IMPLY, Multi-input IMPLY, Multi-output IMPLY logic, while the difference between V_{CLEAR} and V'_{COND} has a great influence on AND, Multi-input AND, Multi-output AND. The parameters can be set according to actual needs. There are many combinations of parameter selection, not only the one shown in Table VII. As long as the range of R_G is reasonable, the logic operation can be realized.

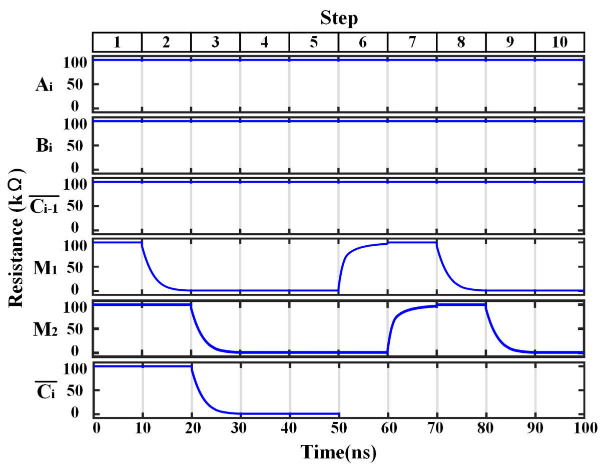
Unlike ideal models, there is resistance drift in real memristors. The resistance varies with the voltage and does not exhibit a perfect threshold characteristic during the transition from LRS to HRS and from HRS to LRS. Compared to the ideal R_G range, the non-ideal characteristic results in a smaller R_G range for IMPLY, ONO, Multi-output IMPLY logic operations and a larger R_G range for AND, OA, Multi-output AND logic operations. Our simulation results show that the two types of logical operations mentioned above function correctly when R_G is in the ranges of $[328, 2000]\Omega$ and $[277, 1518]\Omega$, respectively.

Resistance drift is a serious problem in memristive circuit which can affect the accuracy of the logic operation. In order to ensure that the subsequent operation can be correctly calculated, in some cases, the resistance value of the intermediate result memristor must be refreshed to make it return to R_{ON} or R_{OFF}. The specific implementation measures can be referred to [34].

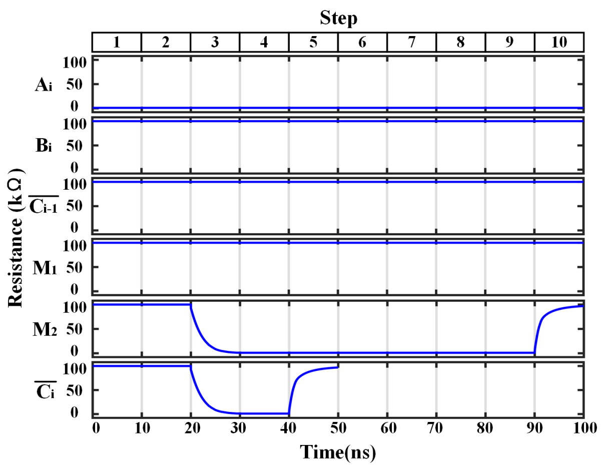
Fig. 12 shows the simulation result of ONO logic and OA logic. In Fig. 12, different rows represent the different combinations of initial states of P_1 and P_2. Columns 1 and 2 represent the changes of resistance values of memristors P_1 and P_2. Columns 3 and 4 represent the change of resistance values of memristors Q at different initial states (0 or 1). We can see that input memristors P_1 and P_2 remain unchanged, but Q changes depending on the inputs. Fig. 13 is the simulation result of two-output IMPLY logic and AND logic. Different rows represent the different combinations of initial states of P, Q_1 and Q_2. The resistance value of P remains unchanged, and the resistance values of Q_1 and Q_2 are always the same. All of the simulations gain the correct results.

The result of one-bit FA is shown in Fig. 14, showing both $\{A_i, B_i, \overline{C_{i-1}}\} = \{0, 0, 0\}$ and $\{A_i, B_i, \overline{C_{i-1}}\} = \{1, 0, 0\}$ cases. Fig. 14 records the resistance changes of all memristors in FA. $\overline{C_i}$ is not displayed after step 5 because it gets the carry-out and will participate in the calculation of the next bit. We can see the result: for Fig. 14(a), M_2 is LRS, which means sum=1, and $\overline{C_i}$ is LRS, which means carry-out=0; For Fig. 14(b), M_2 is HRS, which means sum=0, and $\overline{C_i}$ is HRS, which means carry-out=1. The state of the input memristors does not change, which shows the data reusability.

Fig. 15 is the result of Verilog HDL simulation when we design the peripheral circuit. Our design idea is that each input can be connected to a different voltages through a switch, and then the switch can be turned on and off by a clock, so that different pulse inputs can be realized. Because it is controlled by the clock, the step time of each operation step is fixed in the design, and the step time we set is 10 ns. Fig. 15 is a simulation diagram of the input pulse of V_1. From the operation in Table VI, we can get that the input voltage of V_1



(a)



(b)

Fig. 14 The change of each memristor resistance in the FA. (a): $\{A_i, B_i, \overline{C_{i-1}}\} = \{0,0,0\}$; (b): $\{A_i, B_i, \overline{C_{i-1}}\} = \{1,0,0\}$.

Table 8 Comparisons Between the Proposed FA and Other Works

	Number of Memristors					Number of Steps			Data Reusability
	Input	Output	Total	n=32	Imp.	Total	n=32	Imp.	
Parallel [26]	2n+1	n+1	9n	288	44%	5n+18	178	76%	NO
Serial [31]	2n+1	n+1	2n+3	67	-58%	22n	704	94%	NO
Semiparallel [27]	2n+1	n+1	2n+3	67	-58%	17n	544	92%	NO
Semiserial [32]	2n+1	n+1	2n+6	70	-56%	10n+2	322	87%	NO
Parallel [29]	2n+1	n+1	9n	288	44%	3n+8	104	60%	NO
Parallel [30]	2n+1	n+1	6(n+1)	198	19%	2n+15	79	48%	NO
Our work	2n+1	n+1	5n	160	-	n+9	41	-	YES

may be -1.2V,-0.8V, 0V, 0.8V and 1.2V when different operations are carried out. Assume that the switch K1 is connected with the voltage of -1.2V, and K2, K3, K4 and K5 are connected with the voltages of -0.8V, 0V, 0.8V and 1.2V, respectively. In the first step of the add operation, it can be seen from Fig. 15 that the switch K1 is open, and the input voltage at this time is the pulse voltage of -1.2V, and so on.

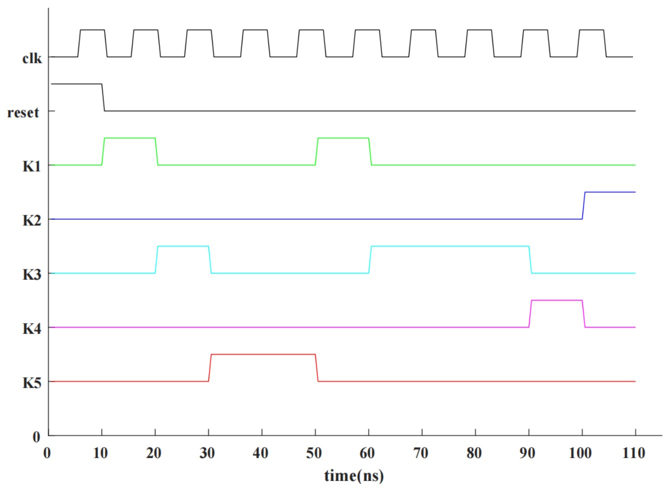


Fig. 15 The peripheral circuit control of V_1.

6 Comparison

To have a better understanding of the advantages and disadvantages of the proposed FA, we compared our design with other existing works. In Table VIII, we list some characteristics and indicators of different FA designs. The percentage improvement (Imp.) is calculated based on $(P_{other} - P_{our})/P_{worse} \times 100\%$, where P_{other} (P_{our}) represents the considered characteristic of the other designs (our design) and P_{worse} is the worse value of the two.

6.1 Calculation speed

Speed plays an important role in determining the merit of a design and its potential for widespread use and implementation. The proposed MIMO logic based FA design needs only 41 steps to implement a 32-bit addition.It is 76%, 94%, 92%, 87% faster than IMPLY based FA in [26] [31] [27] [32], respectively. It is 60% faster than the 3M1R-based parallel FA in [29] and 48% faster than the ORNOR-based parallel FA in [30].

6.2 Number of memristors

The number of memristors reflects integration and cost to some extent. Even if the peripheral circuit accounts for a large proportion of the area, the reduction of the number of memristors will lead to higher integration and less cost as the number of bits of FA increases. Our design, for 32-bit addition, requires 160 memristors, 44% less than the IMPLY based [26] and 3M1R-based [29] parallel design, and 19% less than the ORNOR-based design [30]. The number of memristors in [31], [27] and [32] are nearly 58% less than our design. This is because their designs are serial, semiparallel or semiserial, which are characterized by high integration at the expense of computation speed.

6.3 Data Reusability

Non von Neumann structure is dedicated to processing in memory, which effectively reduces the transmission of data. However, the use of IMPLY logic in [26], [31], [27] and [32] results in the replacement of the original input data. 3M1R and ORNOR logic have data reusability, but designs [29] and [30] do not avoid replacing the original data in the calculation process. The loss of original data means that the data cannot be used again after the calculation. A solution is to retransmit data to another unit [35], but it will increase the time required for data transfer and reduce the efficiency of the operation.

In our design, by using the MIMO logic, the memristors A_i, B_i and $\overline{C_{i-1}}$ do not change state during calculation after input. The input datas can be kept in its original state and be used again. The reusability of input data can reduce the data transmission time, improve efficiency effectively, and promote the combination of storage and computation.

6.4 Power Consumption

Power consumption is a very important index to measure the quality of adders. Although this work increases the number of switches, it greatly reduces the number of memristors and the operation steps of the algorithm. Therefore, compared with other adders, the power consumption of the adder proposed in this paper is relatively small. We select two very representative adders [30] and [32], and when we select the same mos transistor as the switch and perform the addition operation with the same number of bits, our power consumption is 22 % less than that in reference [30] and 39% less than that in reference [32].

7 Conclusion

This paper proposes a MIMO design scheme based on memristive logic and an alternating cross-array structure. The MIMO scheme consists of two parts, Multi-input and Multi-output logic, which have high computational efficiency and provide data reusability, and have a good prospect in the design of complex logic. Alternating cross-array structures can perform rapid interactive operations between adjacent rows, which avoid the burden of operators being in the same row or column.

Then, a fast FA design was proposed based on MIMO logic and alternating crossbar array structure. The proposed FA is superior to the existing parallel design in both area and speed. Although serial, semiparallel, and semiserial designs are smaller in area than ours, our calculation speed is much faster than theirs. In addition, the proposed FA has another advantage, data reusability, which is not available in any of the other designs. Data reusability, which may reduce data loss and data transmission time, is important in the structure of processing in memory. In addition, the alternating cross-array structure can be used for any other complex operations to reduce data movement steps. Although the adder designed in this work increases the number of switches, it greatly reduces the number of memristors and the steps to realize the algorithm operation. Compared with other works, our proposed adder has relatively less power consumption.

Finally, in order to realize the arithmetic operation of the adder, we design a peripheral circuit to control the voltage of the switch and memristor through Verilog HDL. We design the corresponding step time, and control the pulse voltage through the clock when the memristor can reach a stable value. At different times, the voltage that meets the conditions is output, and the arithmetic operation of the adder is realized.

Funding

This work was supported by the National Natural Science Foundation of China(62171182), the Natural Science Foundation of Hunan Province (2021JJ3014) and the Natural Science Foundation Project of Chongqing, Chongqing Science and Technology Commission (CSTB 2022NSCQ-M SX0770).

Data Availability Statement

Data sharing is not applicable to this article, as no datasets were generated or analysed during the current study.

Declaration

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. L. O. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.
2. L. O. Chua and S. M. Kang, "Memristive devices and systems," *Proc IEEE*, vol. 64, no. 2, pp. 209–223, 1976.
3. Y. Ho, G. M. Huang, and P. Li, "Dynamical properties and design analysis for nonvolatile memristor memories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 4, pp. 724–736, 2011.
4. M. Zangeneh and A. Joshi, "Design and optimization of nonvolatile multibit 1T1R resistive RAM," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 8, pp. 1815–1828, 2014.
5. X. Wang, S. Li, H. Liu, and Z. Zeng, "A compact scheme of reading and writing for memristor-based multi-valued memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2017.
6. Chao Zhou, Chunhua Wang, Yichuang Sun, Wei Yao, Hairong Lin. "Cluster output synchronizationfor memristive neural networks," *Information Sciences*, vol. 589, PP. 459-477, 2022.
7. Zhou, Chao and Wang, Chunhua and Yao, Wei and Lin, Hairong. "Observer-based synchronization of memristive neural networks under DoSattacks and actuator saturation and its application to image encryption," *Applicated Mathematics and Computation*, vol. 425, PP. 127080, 2022.
8. Lin, Hairong and Wang, Chunhua and Cui, Li and Sun, Yichuang and Xu, Cong and Yu, Fei, "Brain-like initial-boosted hyperchaos and application in biomedical image encryption," in *IEEE Transactions on Industrial Informatics*, DOI: 10.1109/TII.2022.3155599, 2022.
9. Lin, Hairong and Wang, Chunhua and Xu, Cong and Zhang, Xin and Iu, Herbert HC, "A memristive synapse control method to generate diversified multi-structure chaotic attractors," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, DOI: 10.1109/TCAD.2022.3186516, 2022.
10. Lin, Hairong and Wang, Chunhua and Sun, Jingru and Zhang, Xin and Sun, Yichuang and Iu, Herbert HC, "Memristor-coupled asymmetric neural networks: Bionic modeling, chaotic dynamics analysis and encryption application," in *Chaos, Solitons & Fractals*, vol. 166, pp. 112905, 2023.
11. Q. Zhao, C. Wang, and X. Zhang, "A universal emulator for memristor, memcapacitor, and meminductor and its chaotic circuit," *Chaos*, vol. 29, no. 1, 2019.
12. M. Chen, M. Sun, H. Bao, Y. Hu, and B. Bao, "Flux–charge analysis of two-memristor-based chua's circuit: Dimensionality decreasing model for detecting extreme multistability," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 3, pp. 2197–2206, 2020.
13. A. Haj-Ali, R. Ben-Hur, N. Wald, R. Ronen, and S. Kvatinsky, "Not in name alone: A memristive memory processing unit for real in-memory processing," *IEEE Micro*, vol. 38, no. 5, pp. 13–21, 2018.
14. S. Kvatinsky, "Real processing-in-memory with memristive memory processing unit (mmpu)," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160-052X, 2019, pp. 142–148.
15. R. Yang, H. M. Huang, Q. H. Hong, X. B. Yin, Z. H. Tan, T. Shi, Y. X. Zhou, X. S. Miao, X. P. Wang, and S. B. a. Mi, "Synaptic suppression triplet-stdp learning rule realized in second-order memristors," *Advanced Functional Materials*, p. 1704455, 2017.
16. K. A. Ali, M. Rizk, A. Baghdadi, J. Diguet, J. Jomaah, N. Onizawa, and T. Hanyu, "Memristive computational memory using memristor overwrite logic (mol)," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–13, 2020.
17. S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman, "Mrl — memristor ratioed logic," in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*, 2012, pp. 1–6.
18. G. Papandroulidakis, A. Serb, A. Khiat, G. V. Merrett and T. Prodromakis, "Practical Implementation of Memristor-Based Threshold Logic Gates," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 8, pp. 3041-3051, Aug. 2019.
19. E. Gale, B. de Lacy Costello, and A. Adamatzky, "Boolean Logic Gates from a Single Memristor via Low-Level Sequential Logic," in *Unconventional Computation and Natural Computation*, USA, NY, New York:Springer, pp. 79-89, 2013.
20. N. Xu, L. Fang, K. M. Kim, and C. S. Hwang, "Time-efficient stateful dual-bit-memristor logic," in *physica status solidi (RRL) - Rapid Research Letters*, 2019.
21. S. Kvatinsky, D. Belousov, S. Liman, G. Satat, and U. C. Weiser, "Magic—memristor-aided logic," *Circuits & Systems II Express Briefs IEEE Transactions on*, vol. 61, no. 11, pp. 895–899, 2014.
22. J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.
23. Y. Yang, J. Mathew, S. Pontarelli, M. Ottavi, and D. K. Pradhan, "Complementary resistive switch-based arithmetic logic implementations using material implication," *IEEE Transactions on Nanotechnology*, vol. 15, no. 1, pp. 94–108, 2016.
24. K. M. Kim, N. Xu, X. Shao, K. J. Yoon, H. Kim, R. Williams, and C. S. Hwang, "Single-cell stateful logic using a dual-bit memristor," *physica status solidi (RRL) - Rapid Research Letters*, 2018.
25. L. Guckert and E. E. Swartzlander, "Optimized memristor-based multipliers," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 2, pp. 373–385, 2017.

26. S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (imply) logic: Design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, 2014.
27. S. G. Rohani, N. Taherinejad, and D. Radakovits, "A semiparallel full-adder in imply logic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 297–301, 2020.
28. S. Shin, K. Kim, and S. Kang, "Reconfigurable stateful nor gate for large-scale logic-array integrations," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 7, pp. 442–446, 2011.
29. P. Huang, J. Kang, Y. Zhao, S. Chen, R. Han, Z. Zhou, Z. Chen, W. Ma, M. Li, and L. Liu, "Reconfigurable Nonvolatile Logic Operations in Resistance Switching Crossbar Array for Large-Scale Circuits" *Advanced Materials*, 2019.
30. A. Siemon, R. Drabinski, M.J. Schultis, X. Hu, and J.S. Friedman, "Stateful Three-Input Logic with Memristive Switches[J]," *Scientific Reports*, vol. 9, no. 1, 2019.
31. S. G. Rohani and N. TaheriNejad, "An improved algorithm for imply logic based memristive full-adder," in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–4.
32. D. Radakovits, N. Taherinejad, M. Cai, T. Delaroche, and S. Mirabbasi, "A Memristive Multiplier Using Semi-Serial IMPLY-Based Adder," *IEEE Transactions on Circuits and Systems I: Regular Papers*, no. 99, pp. 1-12, 2020.
33. H. Fu, Q. Hong, C. Wang, J. Sun and Y. Li, "Solving Non-Homogeneous Linear Ordinary Differential Equations Using Memristor-Capacitor Circuit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 11, pp. 4495-4507, Nov. 2021.
34. I. E. Ebong and P. Mazumder, "Self-controlled writing and erasing in a memristor crossbar memory," *IEEE Transactions on Nanotechnology*, vol. 10, no. 6, pp. 1454–1463, 2011.
35. N. Talati, A. H. Ali, R. Ben Hur, N. Wald, R. Ronen, P. Gaillardon, and S. Kvatinsky, "Practical challenges in delivering the promises of real processing-in-memory machines," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1628–1633, 2018.