SCIENCE CHINA
Information Sciences

• **Supplementary File** •

A synthesis method for logic circuits in RRAM array

Xiaole CUI[1], Xiao MA[1], Feng WEI[1] & Xiaoxin CUI[2*]

[1]*1) Key Lab of Integrated Microsystems, Peking University Shenzhen Graduate School, Shenzhen* 518055*, China;*
[2]*2) Institute of Microelectronics, Peking University, Beijing* 100871*, China*

Appendix A Introduction

The RRAM (Resistive Random Access Memory) is a nano-scale non-volatile memory device, which consists of the metal oxide layer and the electrodes. It is firstly implemented by the researchers at Hewlett-Packard Labs using the Titanium dioxide (TiO2) films with a conductive doped region and an insulating undoped region [1]. From that on, various materials are applied to the RRAM for better device performances, and the RRAM device becomes one of the competitive emerging memory devices in recent years, for it has good characteristics such as small device size, high integration density, low power, high endurance, and technology compatibility with CMOS process.

Generally speaking, the various RRAM devices can be classified into two categories according to the existence of the threshold voltages. The RRAM device without threshold voltages is popular as the synapse in the neuromorphic circuits to support the machine learning applications [2–4], and it is also used as the building blocks of some analog circuits [5, 6], because the resistance values of the device can be regulated continuously. And the RRAM device with threshold voltages is often used in the memory and logic applications. The basic RRAM device with threshold voltages has two stable resistance states to represent logic states [1], as presented in Fig. A1. If a positive voltage V_{set}, which is stronger than the threshold voltage V_{on}, is applied to the set end of the RRAM device, the device is set to the low resistance state (LRS). Whereas, the RRAM device turns to the high resistance state (HRS), if a negative voltage V_{reset}, which is lower than the threshold voltage V_{off}, is applied to the device. The read operation does not change the resistance state because of the weak readout voltage. The ideal I-V curve of the basic RRAM device with threshold voltages is illustrated in Fig. A1, where the set end of the RRAM device is labeled with the black thick line.

In recent years, the RRAM based logic circuit has attracted wide attention, because it is useful for the computation in memory (CIM), which is regarded as one of the feasible solutions for the Non von Neumann computer architecture. Several RRAM based logic gate families, such as IMPLY [7], CRS [8], LTG [9], MRL [10], MAGIC [11], MAD [12], Scouting logic [13], four-step logic [14], have been proposed. However, only few of them are implementable in the RRAM array. The IMPLY logic gate, i.e. material implication, is one of the most famous array implementable RRAM based logic gates [2]. It is logic complete, i.e. all of the logic functions can be realized by the IMPLY gates. Lots of logic circuit synthesis methods based on the IMPLY logic gates have been proposed and discussed [15–21]. Lehtonen E. et al. [15] have proofed that the arbitrary Boolean functions can be implemented by the IMPLY gate based circuit with only two working RRAM cells. [16] used the Karnaugh map to aid the synthesis of the IMPLY gate based circuit. For the purpose of performance improvement, the Majority-Inverter graph based algorithm [17] and the multi-stage evolutionary algorithm [18] are proposed, and [19] found out that the discovery of more dont care bits is helpful for the reduction of cycle counts of the circuit. [20, 21] have revealed that the fanout in the IMPLY based circuit may result in the wrong output in the branch nodes, and have solved the fanout problem by duplicating or re-computing the fanout values, and by applying the INVERSED IMPLY operations, respectively. However, the IMPLY logic circuits consume relatively long computation time, for the circuits work serially in essence.

The motivation of this work is to improve the performance of the IMPLY gate based logic circuits. The main contributions of our work include:

(1) The upper bounds on the input count of the IMPLY gate and the IMPLY compatible OR gate are analyzed, respectively. The analysis results show that these two types of multi-input gate have high usability in the design of logic circuits.

(2) It is found out that the IMPLY gate and the IMPLY compatible OR gate have the equivalent circuit, respectively, if the polarities of the RRAM devices and the applied voltages are reversed at the same time. It provides the flexibility to implement the IMPLY and the OR functions in row and column, respectively.

* Corresponding author (email: cuixx@pku.edu.cn)

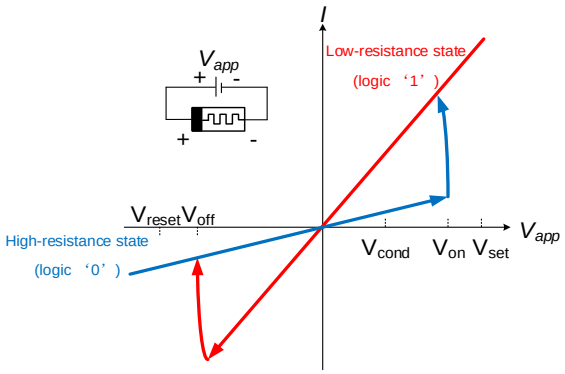

Figure A1 The ideal I-V curve of the RRAM device with threshold voltages.

Figure B1 The IMPLY gates. (a) The original IMPLY gate [7]; (b) The n-input IMPLY gate [22].

(3) We propose a circuit synthesis method of logic circuit in the RRAM array based on the multi-input IMPLY gates and OR gates. The synthesis results show that, for most cases, the RRAM based logic circuits generated from the proposed design methods outperform those original IMPLY gate based counterparts.

Appendix B The upper bound on input count of the in-array IMPLY gate and OR gate

Appendix B.1 The upper bound on input count of the IMPLY gate

The original IMPLY gate [7], which realizes the function $Q = \overline{P}+Q$, is presented in Figure B1(a), where $R_{on} << R << R_{off}$, R_{off} and R_{on} are the resistance values of the RRAM cell if the device is in the HRS and LRS, respectively. The HRS represents the logic 0, and the LRS represents the logic 1. And the multi-input IMPLY gate is designed by directly extending the number of input cells [22], as shown in Figure B1(b). Conceptually, if one of the input cells in $A_1 \sim A_n$ is in LRS, the voltage across the output cell Q is so weak to maintain the resistance state of Q. And if all the input cells are in HRS, the voltage across the output cell is strong enough to set Q to LRS. It realizes the logic function $Q = \overline{A_1 + ... + A_n} + Q$. Particularly, this circuit is the n-input NOR gate if the output cell Q is in HRS.

However, the number of inputs of the circuit in Figure B1(b) has upper bound, because the circuit output is obtained by the voltage division between the RRAM cells and the resistor. For the n-input IMPLY gate, assuming that the equivalent resistance of the input cells is $R_{eq} = R_{A1}//R_{A2}//...//R_{An}$, the voltage across the output cell is calculated by eq. (B1) according to the Kirchhoff's Law:

$$V = V_{set} - \frac{\frac{V_{cond}}{R_{eq}} + \frac{V_{set}}{R_{off}}}{\frac{1}{R} + \frac{1}{R_{eq}} + \frac{1}{R_{off}}}. \tag{B1}$$

If Q is in LRS, i.e. logic 1 state, Q remains LRS regardless of the states of $A_1 \sim A_n$. If Q is in HRS, in order to implement the IMPLY function, the maximum voltage of V should weak enough to maintain the resistance state of the output cell if there is at least one input cell in LRS, i.e. $V_{max} < V_{on}$; and the minimum voltage of V should strong enough to set the output cell to LRS if all the input cells are in HRS, i.e. $V_{min} > V_{on}$. The simulations are conducted with the RRAM device model [23]. In the simulation, we set $R = 4 \times 10^4\Omega$, $V_{on} = 0.8V$, $V_{set} = 1V$, $V_{cond} = 0.5V$, $V_{off} = -1.1V$, $V_{reset} = -1.2V$, $R_{off} = 2.8 \times 10^6\Omega$, $R_{on} = 5 \times 10^3\Omega$. According to the conditions above, the maximum number of inputs is 43, showing that the multi-input IMPLY gate is applicable in the logic circuit. Actually, the maximum input of the IMPLY gate varies if we change the simulation conditions. However, the multi-input IMPLY gate circuit is still useful, because the large input IMPLY operation can be partitioned into several IMPLY operations with less number of inputs. Actually, the circuit designers seldom use the high fan-in gate circuit in the real design practices.

Table B1 compares the area and performance of the NOR gates implemented by the multi-input IMPLY gate and the original IMPLY gates, respectively. It is seen that the multi-input NOR gates outperform the original IMPLY based counterparts, and the multi-input NOR gates constantly consume 2 cycles regardless of the number of inputs.

Table B1 Comparison on the performance and area of the RRAM based NOR gates

NOR gates	Cycles	RRAM cells	Resistors
Original IMPLY(two inputs)	5	3	1
Multi-input IMPLY(two inputs)	2	3	1
Original IMPLY(n inputs)	3n-1	n+1	1
Multi-input IMPLY(n inputs)	2	n+1	1

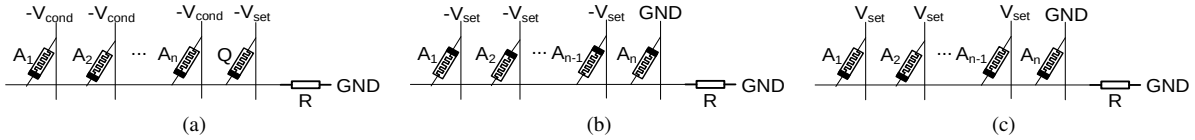

Figure B2 The in array gates. (a) The n-input IMPLY gate with reversed device polarity; (b) The proposed n-input OR gate; (c) The n-input OR gate with reversed device polarity [22].

Table B2 Comparison on the performance and area of the RRAM based OR gates

OR gates	Cycles	RRAM cells	Resistors
Original IMPLY(two inputs)	3	3	1
Multi-input OR(two inputs)	1	2	1
Original IMPLY(n inputs)	3n-3	n+1	1
Multi-input OR(n inputs)	1	n	1

Appendix B.2 The in-array IMPLY gate and OR gate

Assume that the IMPLY gates presented in Figure B1 are deployed in a RRAM row. From the view of column in the same RRAM array, the polarities of the RRAM cells are opposite to that in Figure B1. Fortunately, if the polarities of the RRAM cells and the applied voltages in Figure B1(b) are reversed simultaneously, as presented in Figure B2(a), it implements the same function. This circuit also realizes the n-input IMPLY function if it is deployed in the column of the RRAM array. And these two equivalent IMPLY circuits have the same limitation of the input counts, because they share the same voltage constraints.

It is convenient to realize complex logic functions if the OR gate is available. We propose the n-input OR gate in Figure B2(b). The input states of the OR gate are represented by the resistance states of $A_1 \sim A_n$, and the output state is represented by the resistance state of A_n. The voltage $-V_{set}$ is applied to the set ends of $A_1 \sim A_{n-1}$, respectively, and both the output cell A_n and the resistor R are connected to ground. If A_n is in LRS, i.e. logic 1 state, A_n remains LRS regardless of the states of $A_1 \sim A_{n-1}$. Conceptually, if $A_1 \sim A_n$ are all in HRS, the voltage across A_n is so weak to keep A_n in HRS because $R_{on} << R << R_{off}$; whereas, if A_n is in HRS and at least one input cell in $A_1 \sim A_{n-1}$ is in LRS, the voltage across A_n is strong enough to set A_n to LRS. The OR function is implemented. The circuit of the proposed OR gate is equivalent to the multi-input OR gate in [22] as presented in Figure B2(c). The polarities and the applied voltages of the RRAM cells are opposite in these two circuits.

Actually, the multi-input OR gate has upper bound of input count. If the gate output is logic 1, the resistance state of A_n is independent of the number of inputs. Whereas, if the gate output is logic 0, $A_1 \sim A_n$ are all in HRS, the voltage across the output cell A_n is calculated by eq. (B2).

$$V = V_{set} \frac{R_{off}//R}{\frac{R_{off}}{n-1}+R_{off}//R} \approx V_{set} \frac{R}{\frac{R_{off}}{n-1}+R} \tag{B2}$$

The maximum voltage V_{max} should be weaker than V_{on} to implement the OR function. The maximum number of inputs is 279, by the simulation with the same tool and parameters in section 2.1. It manifests that the multi-input OR gate has high usability in the logic circuits.

Table B2 compares the multi-input OR gates presented in Figure B2(b) and the original IMPLY logic based counterparts. It is seen that the multi-input OR gates in Figure B2(b) are superior to the original IMPLY based OR gates both on the performance and area consumption. The OR gates can be used together with the IMPLY gates in the RRAM array, because the applied voltages are compatible, and the input and output states of the gates are all represented by the resistance states of the corresponding RRAM cells.

Appendix C The two-dimensional synthesis algorithm of logic circuits

In fact, the RRAM devices are usually organized into arrays, and the size of memory array is limited due to some specific problems, such as IR drop, crosstalk, etc. [24–27]. If the required number of RRAM cells of the complex Boolean function is

greater than that in one RRAM row or column, the circuit is not able to be implemented by the one-dimensional synthesis method proposed in section 2. It demands for the RRAM array oriented synthesis method of logic circuits.

For a given logic function, the proposed synthesis method of logic circuit in a RRAM array consists of two phases. The first phase maps the input, output, and intermediate variables into the RRAM array, under the constraints of the array size. The second phase computes the result with logic operations.

Phase 1. Circuit mapping.

The mapping result significantly affects the circuit performance and area consumption. We propose 3 principles for the better circuit mapping results in this phase. (1) Take the performance advantage of the multi-input IMPLY gate and OR gate in the circuit design. (2) If it is possible, map the variables in the same NOR cube to the same row (or column). (3) The different cubes may share some variables. The shared variables are redundant variables, if the cubes containing them are mapped to the same row (or column). The cubes with the most redundant variables are mapped to the same row (or column) if it is possible. It helps to reduce the RRAM counts of the circuit.

The mapping process works as follows.

Step 1. Transform each minterm in the standard SOP(Sum Of Product) expression of the logic function into the NOR form by the De Morgan's law.

Step 2. Sort the NOR cubes in the order of the number of variables. The NOR cubes are partitioned into two groups. The group A contains the NOR cubes that cannot be mapped into a single RRAM row, and the other NOR cubes are in the group B.

Step 3. Map the NOR cubes in group A. Select the NOR cube with the maximum number of variables in group A, and partition the variables of the selected NOR cube into several subgroups. The variables in each subgroup can be mapped into a single RRAM row. Map the variables in each subgroup into the RRAM array row by row, and delete the current NOR cube in group A. Then select the NOR cube with the maximum number of variables in the current group A, and perform the same operations. This process continues until the group A becomes empty or there is no available RRAM row anymore.

Step 4. Map the NOR cubes in group B. The NOR cubes in group B are mapped into the rows which are not occupied by the NOR cubes in group A. The mapping of each row in the RRAM array consists of 3 sub-steps.

Sub-step 1. Select the NOR cube with the maximum number of variables in the current group B, and map its variables into an empty row.

Sub-step 2. Map the NOR cubes containing redundant variables with respect to the mapped NOR cubes in the current row.

Firstly, it selects the unmapped NOR cube containing the maximum number of redundant variables with respect to the mapped NOR cube in the current row.

If there are enough available RRAM cells in the current row, it maps the variables other than the redundant variables in the selected NOR cube into the current row. Then it selects the unmapped NOR cube containing the maximum number of redundant variables with respect to the mapped NOR cubes in the current row, and maps the variables other than the redundant variables in this cube into the current row, if the available RRAM cells in the current row are enough.

Else, it selects the unmapped NOR cube containing the sub-maximum number of redundant variables with respect to the mapped NOR cubes in the current row, and maps the non-redundant variables in this cube into the current row, if the available RRAM cells in the current row are enough.

This process continues until the current row is full or none of the NOR cubes in group B contains the redundant variables with respect to the mapped NOR cubes in the current row. Delete these mapped NOR cubes in group B.

Sub-step 3. Fill in the rest available RRAM cells in the current row.

If there are still available cells in the current row, it selects the unmapped NOR cube with the maximum number of variables that can be mapped into the current row in the current group B one by one, and maps their variables to the current row. This process continues until all the available cells in the current row are filled. Delete these mapped NOR cubes in group B.

Select and map the NOR cubes in the current group B with the 3 sub-steps above row by row, if the empty rows are available. This process continues until the current group B is empty, or none of the row in the RRAM array is available. The circuit mapping is completed successfully if the process stops with the former condition, whereas it fails if the process stops with the latter condition. This process generates a relatively dense mapping result, because the redundant variables are mapped to the same RRAM cell.

The circuit with the logic function $F = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E}FGHIJ + BC\overline{D}\,\overline{E}\,\overline{F}\,\overline{G}\,\overline{H}P\overline{Q} + CD\overline{E}FG + B\overline{E}FG + \overline{A}\,\overline{D}E\overline{G} + \overline{A}\,\overline{C}E\overline{G} + \overline{A}\,\overline{B}E\overline{G} + A\overline{E}FG + E\overline{F}$ is taken as an example to illustrate the mapping process. The target is to map the logic function $F3$ into an 8×8 RRAM array.

Step 1. F is transformed as $F' = \overline{A + B + C + D + E + \overline{F} + \overline{G} + \overline{H} + \overline{I} + \overline{J}} + \overline{\overline{B} + \overline{C} + D + E + F + G + H + \overline{P} + Q} + \overline{\overline{C} + \overline{D} + E + \overline{F} + \overline{G}} + \overline{\overline{B} + E + \overline{F} + \overline{G}} + \overline{A + D + \overline{E} + G} + \overline{A + C + \overline{E} + G} + \overline{A + B + \overline{E} + G} + \overline{\overline{A} + E + \overline{F} + \overline{G}} + \overline{\overline{E} + F}$, with the De Morgan's law.

Step 2. In the 8×8 RRAM array, at most seven variables can be mapped into one row, because the rightmost cell in each row is reserved for the intermediate value. For the transformed F', the NOR cubes $\overline{A + B + C + D + E + \overline{F} + \overline{G} + \overline{H} + \overline{I} + \overline{J}}$ and $\overline{\overline{B} + \overline{C} + D + E + F + G + H + \overline{P} + Q}$ belong to the group A, and the other NOR cubes belong to the group B.

Step 3. The NOR cube $\overline{A + B + C + D + E + \overline{F} + \overline{G} + \overline{H} + \overline{I} + \overline{J}}$ is selected firstly, for it contains the maximum number of variables in the current group A. The variables A, B, C, D, E, $\overline{F}$, $\overline{G}$ are mapped to $Cell(1,1) \sim Cell(1,7)$, the variables

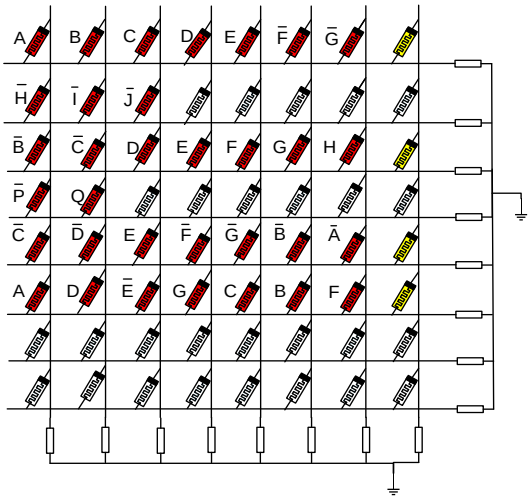

Figure C1 The result of the circuit mapping for the example circuit.

Table C1 The selection process of the NOR cubes in the example circuit

Mapping-Step	1	2	3	4	5	6	7	8	9
Row Number	1-2	3-4	5	5	6	6	6	6	6
Selector of the NOR term	Number of required cells			Saved cells from redundancies/required cells		Number of required cells	Saved cells from redundancies/required cells		
$\overline{A+B+C+D+E+\overline{F}+\overline{G}+\overline{H}+\overline{I}+\overline{J}}$	10	-	-	-	-	-	-	-	-
$\overline{\overline{B}+\overline{C}+D+E+F+G+H+\overline{P}+Q}$	9	9	-	-	-	-	-	-	-
$\overline{\overline{C}+\overline{D}+E+\overline{F}+\overline{G}}$	5	5	5	-	-	-	-	-	-
$\overline{\overline{B}+E+\overline{F}+\overline{G}}$	4	4	4	3/1	-	-	-	-	-
$\overline{A+D+\overline{E}+G}$	4	4	4	0/4	0/4	4	-	-	-
$\overline{A+C+\overline{E}+G}$	4	4	4	0/4	0/4	4	3/1	-	-
$\overline{A+B+\overline{E}+G}$	4	4	4	0/4	0/4	4	3/1	3/1	-
$\overline{\overline{A}+E+\overline{F}+\overline{G}}$	4	4	4	3/1	3/1	-	-	-	-
$\overline{\overline{E}+F}$	2	2	2	0/2	0/2	2	1/1	1/1	1/1

$\overline{H}$, $\overline{I}$, $\overline{J}$ are mapped to $Cell(2,1) \sim Cell(2,3)$, respectively. Then the NOR cube $\overline{\overline{B}+\overline{C}+D+E+F+G+H+\overline{P}+Q}$ is selected, and its variables are mapped to the third and fourth rows respectively, as presented in Figure C1.

Step 4. The NOR cube $\overline{\overline{C}+\overline{D}+E+\overline{F}+\overline{G}}$, which contains the maximum number of variables in the current group B, is selected firstly. The variables $\overline{C}$, $\overline{D}$, E, $\overline{F}$, $\overline{G}$ are mapped to $Cell(5,1) \sim Cell(5,5)$, respectively. Both $\overline{\overline{B}+E+\overline{F}+\overline{G}}$ and $\overline{\overline{A}+E+\overline{F}+\overline{G}}$ have 3 redundant variables, i.e. E, $\overline{F}$ and $\overline{G}$ with respect to the mapped NOR cube, and the variables $\overline{B}$ and $\overline{A}$ are mapped to $Cell(5,6)$ and $Cell(5,7)$, respectively. All the available cells in the fifth row are filled. Following the same process, the rest four NOR cubes are mapped to the sixth row. The selection process of the NOR cubes in F' is illustrated in Table C1. The NOR cube is selected in the corresponding mapping-step if the numbers are shaded in the table. The circuit mapping is completed successfully because the circuit only occupies 6 rows in the 8×8 array.

Phase 2. Computation.

The result of the logic function is obtained by four computation steps.

Step 1. Input. Firstly, all the RRAM cells in the array are reset into HRS simultaneously. Then it sets the corresponding cells into LRS in each row if its input state is logic 1. The set operations consume at most k cycles, if the circuit occupies k rows.

Step 2. Compute the NOR cubes which occupy multiple rows. For each NOR cube occupies multiple RRAM rows, each corresponding bit in different rows are ORed together, and the intermediate results are stored in the RRAM cells in the first row it occupies. It consumes only single cycle. All the rows storing the intermediate results perform NOR operations in the same cycle, and the results of the NOR cubes are stored in the corresponding rightmost RRAM cells, respectively.

Step 3. Compute the NOR cubes which are mapped in the single row. For each row occupied by a complete NOR cube or several NOR cubes, it computes the first NOR cube and stores the result in the rightmost RRAM cell in this row. Then the other NOR cubes are 'ORed' one by one by the IMPLY operations, and the results are stored into the rightmost cell in the row iteratively, until all the NOR cubes in this row are computed.

Step 4. Output. The intermediate results stored in the cells in the rightmost column are ORed together, and the final result is stored in one of these cells.

In the example in Figure C1, the step 1 consumes at most seven cycles, because the set operations have to be executed row by row. In step 2, it executes $Cell(1,i) = Cell(1,i) + Cell(2,i)$ and $Cell(3,i) = Cell(3,i) + Cell(4,i)$, $i = 1 \sim 7$, respectively, and then the results of the two NOR cubes are computed and stored in $Cell(1,8)$ and $Cell(3,8)$, simultaneously. In step 3, for the fifth row in Figure C1, the NOR operation is applied to the cells $\overline{C}$, $\overline{D}$, E, $\overline{F}$, $\overline{G}$ and $Cell(5,8)$, and the result

Table C2 Comparisons of different RRAM based synthesis methods of logic circuits

Logic Function	One-dimensional synthesis method						Proposed two-dimensional syhthesis method		
	RRAM count/Cycle count						Row count/Cycle count		
	[16]	[17]	[18]	[19]	[20]	[21]	8 Columns	16 Columns	32 Columns
rd53f1	-/27	60/64	-/26	-/30	14/22	15/26	1/7	1/7	1/7
rd53f2	-/57	77/77	-/62	-/56	20/37	31/42	8/26	1/18	1/18
rd53f3	-/32	86/66	-/130	-/125	18/26	31/42	4/17	1/13	1/13
con1f1	-/18	70/75	-/17	-/35	12/14	12/15	2/8	1/6	1/6
con1f2	-/19	60/76	-/24	-/17	12/15	12/17	1/6	1/6	1/6
rd73f1	-/238	291/121	-/404	-/142	68/137	133/223	42/86	1/44	1/44
rd73f2	-/46	129/88	-/-	-/257	25/37	66/97	64/130	1/66	1/66
rd73f3	-/104	193/107	-/-	-/210	36/70	72/118	1/37	1/37	1/37
rd84f1	-/351	430/153	-/1102	-/214	67/145	215/361	84/170	2/88	1/86
rd84f2	-/47	172/88	-/-	-/336	29/43	81/117	256/387	2/132	1/130
rd84f3	-/23	90/50	-/-	-/10	10/10	16/16	2/6	1/3	1/3
rd84f4	-/345	473/141	-/-	-/420	79/170	215/363	18/90	2/74	1/72
9sym_d	-/1418	923/175	-/1035	-/420	258/575	459/755	86/175	6/95	1/89
sao2f1	-/102	110/108	-/94	-/67	34/61	55/87	20/33	3/15	1/12
sao2f2	-/112	234/119	-/208	-/49	41/77	65/104	40/63	4/26	1/22
sao2f3	-/380	325/143	-/130	-/20	89/188	122/199	6/30	2/26	1/22
sao2f4	-/252	326/143	-/-	-/34	77/165	121/202	14/43	2/30	1/28

of $\overline{\overline{C}+\overline{D}+E+\overline{F}+\overline{G}}$ is stored in $Cell(5,8)$. Then $\overline{\overline{B}+E+\overline{F}+\overline{G}}+Cell(5,8)$ is computed by the multi-input IMPLY operation, and the intermediate result is stored in $Cell(5,8)$. Finally, $\overline{\overline{A}+E+\overline{F}+\overline{G}}+Cell(5,8)$ is computed and stored in $Cell(5,8)$. Similar computations are performed in the sixth row. In step 4, it computes $Cell(1,8)+Cell(3,8)+Cell(5,8)+Cell(6,8)$, and the final result is stored in $Cell(1,8)$ as the output of the logic function.

Assume a logic function which has m and n NOR cubes in original group A and B, respectively, is mapped to k rows in the RRAM array. It totally requires $m+n+k+3$ cycles, including one initialization operation, k set operations, $m+1$ OR operations, and $n+1$ multi-input IMPLY operations, to obtain the final result.

The proposed array oriented synthesis method of logic circuit is applied to some benchmark circuits, with different number of columns. The synthesis results are collected in Table C2. The circuit performance is represented by the required cycle counts to obtain the logic output. The synthesis results are compared with the counterparts based on the previous one-dimensional synthesis method of RRAM logic circuit. The synthesis results show that, for the most cases, the circuits generated by our proposed two-dimensional synthesis method outperform those counterparts generated by the previous one-dimensional synthesis methods. And the cycle count decreases with the increase of the column count.

This work proposes the synthesis methods for logic circuits in an RRAM array. Thanks for the multi-input IMPLY gate, the proposed synthesis method results in the high performance in-array logic circuits, which outperform those one-dimensional counterparts generated from the original IMPLY gate based methods for the most cases.

References

1 Strukov D B, Snider G S, Stewart D R, et al. The missing memristor found. Nature, 2008, 453: 80-83.
2 Prezioso M, Merrikh-Bayat F, Hoskins B D, et al. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. Nature, 2015, 521: 61-64.
3 Yao P, Wu H Q, Gao B, et al. Face classification using electronic synapases. Nature Communication, 2017, 8, 15199.
4 Tang Y, Cui X L, Liu C L, et al. A spike neuron network prototype based RRAM array. In: International Conference on Solide-State and Integrated Circuit Technology, Hangzhou, China, 2016. 1-3.
5 Li B X, Gu P, Shan Y, et al. RRAM based analog approximate computing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 34: 1905-1917.
6 Gao L G, Chen P CY, Yu S M. Programming protocol optimization for analog weight tuning in resistive memories. IEEE Transactions on Electron Device Letters, 2015, 36: 1157-1159.
7 Borghetti J, Snider G S, Kuekes P J, et al. Memristive switch enable stateful logic operations via material implication. Nature, 2010, 464: 873-876.
8 Yang Y, Mathew J, Pontarelli S, et al. Complementary Resistive Switch-Based Arithmetic Logic Implementations Using Material Implication. IEEE Transactions on Nanotechnology, 2016, 15: 94-108.
9 Gao L, Alibart F, Strukov D B. Programmable CMOS/memristor threshold logic. IEEE Trans. on Nanotechnology, 2013, 12: 115-119.
10 Kvatinsky S, Wald N, Satat G, et al. MRL-Memristor Ratioed Logic. In: International Workshop on Cellular Nanoscale Networks & Their Applications, Turin, Italy, 2012. 1-6.
11 Talati N, Gupta S, Mane P, et al. Logic Design Within Memristive Memories Using Memristor-Aided loGIC (MAGIC). IEEE Transactions on Nanotechnology, 2016, 15(4): 635-650.

12 Guckert L, Swartzlander E. MAD gates- memristor logic design using driver circuitry. IEEE Trans. on Circuits & Systems II Express Briefs, 2017, 64: 171-175.
13 Xie L, Du Nguyen H A, Yu J, et al. Scouting logic: a novel memristor-based logic design for resistive computing. In: IEEE Computer Society Annual Symposium on VLSI, Bochum, Germany, 2017. 176-181.
14 Zhang K, Cui X L, Cui X X. A design of high performance full adder with memristors. In: IEEE International Conference on ASIC, Guiyang, China, 2017. 746-749.
15 Lehtonen E, Poikonen J H, Laiho M. Two memristors suffice to compute all Boolean functions. IET Electronics Letters, 2010, 46: 239-240.
16 Burger J, Teuscher C, Perkowski M. Digital Logic Synthesis for Memristors. Reed-Muller, 2013, 31-40.
17 Shirinzadeh S, Soeken M, Gaillardon P -E, et al. Fast logic synthesis for RRAM-based in-memory computing using Majority-Inverter graphs. In: IEEE Design, Automation & Test in Europe Conference & Exhibition, Dresden, Germany, 2016. 948-953.
18 Wang X X, Tan R, Perkowski M. Synthesis of memristive circuits based on stateful IMPLY gates using an evolutionary algorithm with a correction function. In: IEEE/ACM International Symposium on Nanoscale Architecture, Beijing, China, 2016. 97-102.
19 Ragjivamsjo A, Perkowski M. Logic synthesis and a generalized notation for memristor-realized material implication gates. In: IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, USA, 2014. 470-477.
20 Wang H P, Lin C C, Wu C C, et al. On synthesizing memristor-based logic circuits with minimal operational pulses. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2018, 26: 1-11.
21 Lalchhandama F, Sapui B G, Datta K. An improved approach for the synthesis of Boolean functions using memristor based IMPLY and INVERSE-IMPLY gates. In: IEEE Computer Society Annual Symposium on VLSI, Pittsburgh, PA, USA, 2016. 319-324.
22 Xie L, Nguyen H A D, Taouil M, et al. Boolean logic gate exploration for memristor crossbar. In: International Conference on Design and Technology of Integrated Systems in Nanoscale Era, Istanbul, Turkey, 2016. 1-6.
23 Li H, Huang P, Gao B, et al. A SPICE model of resistive random access memory for large-scale memory array simulation. IEEE Electron Device Letters, 2014, 35: 211-213.
24 Han R, Huang P, Zhao Y, et al. Efficient evaluation model including interconnect resistance effect for large scale RRAM crossbar array matrix computing. SCIENCE CHINA-INFORMATION SCIENCES, 2019, 62: 022401(1)-022401(11).
25 Xu C, Niu D, Muralimanohar N, et al. Overcoming the challenges of crossbar resistive memory architectures. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture, Burlingame, CA, USA, 2015. 476-488.
26 Levisse A, Giraud B, Noel J P, et al. High density emerging resistive memories: what are the limits? In: 2017 IEEE Latin American Symposium on Circuits & Systems, Bariloche, Argentina, 2017.1-4.
27 Zhou J, Kim K H, Lu W. Crossbar RRAM arrays: selector device requirements during read operation. IEEE Transactions on Electron Devices, 2014, 61: 1369-1376.
28 Kvatinsky S, Satat G, Wald N, et al. Memristor-based material implication (IMPLY) logic: design principles and methodologies. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2014, 22: 2054-2066.