Optimizing Performance and Energy Overheads Due to Fanout in In-Memory Computing Systems

Md Adnan Zaman, Rajeev Joshi, Srinivas Katkoori

Optimizing Performance and Energy Overheads Due to Fanout In In-Memory Computing Systems

Md Adnan Zaman, Rajeev Joshi, and Srinivas Katkoori

Department of Computer Science & Engineering,
University of South Florida, Tampa, FL
{mdadnanz,rajeevjoshi,katkoori}@mail.usf.edu

Abstract. For NOR-NOT based memristor crossbar architectures, we propose a novel approach to address the fanout overhead problem. Instead of copying the logic value as inputs to the driven memristors, we propose that the controller reads the logic value and then applies it in parallel to the driven memristors. We consider two different cases based on the initialization of the memristors to logic-1 at the locations where we want keep the first input memristor of the driven gates. If the memristors are initialized, it falls under case 1, otherwise case 2. In comparison to recently published works, experimental evaluation on ISCAS85 benchmarks resulted in average performance improvements of 51.08%, 38.66%, and 63.18% for case 1 and 50.94%, 42.08%, and 60.65% for case 2 considering three different mapping scenarios (average, best, and worst). In regards to energy dissipation, we have also obtained average improvements of 91.30%, 88.53%, and 74.04% for case 1 and 86.03%, 78.97%, and 51.89% for case 2 considering the aforementioned scenarios.

Keywords: Memristor, In-memory computing, Fanout, MAGIC, crossbar, logic synthesis, resistive memory

1 Introduction

Von Neumann architecture suffers from *memory wall* problem due to bandwidth mismatch between slower memory and faster CPU [1]. To overcome *memory wall* problem, non von-Neumann architecture is being actively considered where storage and computing can be performed in the same location. This computing inside memory is known as *in-memory computing*. Emerging non-volatile resistive memory technology such as memristor can enable such non von-Neumann computing paradigm. A Memory Processing Unit (MPU) has been proposed [2], where memristive memory is used as storage in conjunction with logical operations.

Due to high speed, low power consumption, scalability, data retention, endurance, and compatibility with conventional CMOS, many memristor based

logic families and circuits have been proposed [3]. Based on logic state variable, memristor based logic families can be classified into stateful (logic value represented with memristor resistance) and non-stateful logic families [4]. In this work, we employ a stateful logic family, known as Memristor-Aided loGIC (MAGIC) [5]. In this logic style, for a given logic gate, input values and output value(s) are stored as memristor states. Memristors can be fabricated on a crossbar array, which offers high storage density and low power consumption [6]. With MAGIC, only NOR and NOT gates can be directly mapped to crossbar array.

In recent years, researchers have proposed a few in-memory logic synthesis approaches based on MAGIC logic style, where a given circuit netlist consisting of only NOR and NOT gates is mapped to a memristor crossbar. In [5], a detailed procedure to map NOR/NOT logic gates to crossbar has been discussed and also transpose crossbar concept has been introduced to allow gates to be mapped along the rows as well as columns in a crossbar architecture. In [7], a synthesis tool has been proposed that maps arbitrary logical functions within the memristive memory in an optimal manner. In [8], a scalable design flow for in-memory computing has been proposed that allows a given circuit netlist to be implemented in transpose crossbar. In both of these approaches, a given gate netlist is first converted into a netlist of NOR/NOT gates using an existing logic synthesis tool [9] and mapped to the crossbar architecture. While mapping, we come across fanout where a single output (driving) memristor of a logic gate has to be used as input (driven) memristors of multiple gates connected to it. For a fanout of two or more leaf memristors, current methods (previous approaches) perform the *copy* operation for a number of times equal to the number of driven memristors that are not on the same row or column as of the driving memristor. Such copy style requires two NOT operations which in turn requires two extra cycles. With the increment of fanouts in a given netlist, the number of extra cycles increases hence energy as well. To the best of our knowledge, no other previous works attempted to reduce the additional cycle count of a copy operation inherent to a fanout.

In this work, we propose a novel approach that will reduce the performance and energy overheads originated from fanout in a given circuit netlist. Instead of copying the value, the proposed controller can read the value and apply in parallel to the driven memristors. We consider two different cases based on the initialization of the memristors at the copy locations. The locations on the crossbar where we want to keep the first input memristors of the driven gates can be defined as the copy locations. In our prior work [10] which discusses case 1, like the previous works [7, 8], we consider that the memristors at the copy locations are initilaized to logic-1 which provides an added advantage to decide whether to write the read value or not to the driven memristors. As proposed in MAGIC logic style, the output memristors are initialized to logic-1 prior to logic execution and it allows the controller to skip write cycle if the read value is one. We have compared our work with a recently published work [8] for three different mapping scenarios. We obtain average improvements of 51.08%, 38.66%, and

63.18% and 91.30%, 88.53%, and 74.04% in performance and energy dissipation respectively. For case 2, we consider that the memristors at the copy locations are not initialized to logic-1 which dictates the use of initialization cycles with the previous approach. We obtain average improvements of 50.94%, 42.08%, and 60.65% and 86.03%, 78.97%, and 51.89% in performance and energy dissipation respectively.

The rest of the chapter is organized as follows: Section 2 presents background and related work. Section 3 describes the proposed approach to reduce the number of cycles and energy dissipations related to fanout. Section 4 reports experimental results. Section 5 draws conclusions.

2 Background and Related work

In this section, we discuss the basic working principle of a memristor, relevant logic design styles, crossbar architecture, and in-memory computing. We also review some of the recent works on memristor based in-memory logic synthesis for a given gate-level netlist with particular concentration on fanout.

2.1 Memristor

Memristor is a two terminal device that can remember its previous state and change its resistance based on a given potential across the device. Chua [11] first proposed memristor that links flux (ϕ) and charge (q) to its memristance (M) according to,

$$d\phi = Mdq \tag{1}$$

Structurally, memristor can be thought of as a thin semiconductor film with thickness (D) sandwiched between two contacts. We can change the overall resistance by changing its doped region width, w [12].

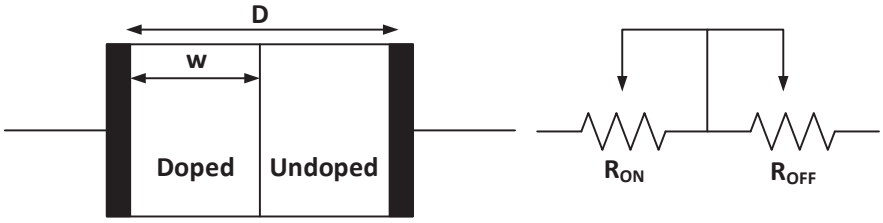

Fig. 1. Physical and circuit model of memristor [2].

2.2 Memristor Aided LoGIC (MAGIC)

Our proposed methodology is based on MAGIC logic style [5]. As shown in Fig. 2, memristors' resistances of the IN_1 and IN_2 are considered as input values and we

can determine the output logic value by measuring OUT memristor resistance. An execution voltage V_G is applied to both input memristors and the output value is stored in the output memristor.

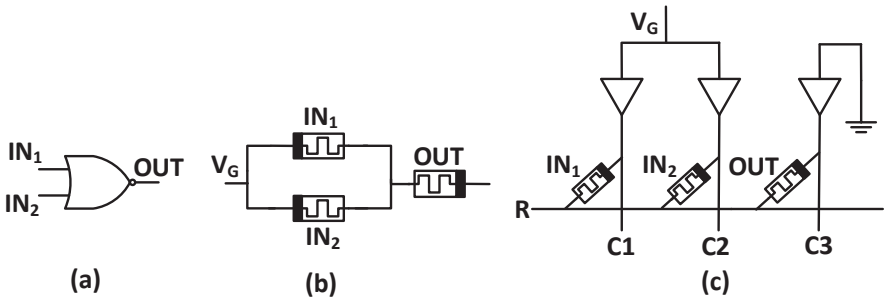

Fig. 2. NOR gate implementation using MAGIC on crossbar.

2.3 In-memory Computation Using Memristor Crossbar

A Memory Processing Unit (MPU) based on *in-memory computing* architecture is as shown in Fig. 3 [2]. It consists of a controller, crossbar memory, and analog multiplexers. The analog multiplexers' outputs are connected to the bitlines and wordlines of the memristive memory and the voltage select lines of the multiplexers are connected to the controller. To carry out a regular read or write operation, controller sends suitable signals to the addressed memristor cells through multiplexers. We have also shown two signals, V_{SET} and V_{RESET} that can enable writing logic-1 and logic-0 to the memristor cells respectively.

The suitability of the crossbar architecture can be better explained by NOR operation as shown in Fig. 2. We, first, initialize the output memristor (OUT) to logic-1 and connect it to the ground (0V) and then we apply execution voltage V_G to the input memristors (IN_1 and IN_2). This voltage may corrupt data on other memristors on the same row/column, to avoid this, an isolation voltage V_{iso} needs to be applied to the columns and rows that we want to unselect. Parallel execution of NOR/NOT gates on the crossbar requires the alignment of the inputs and outputs of the respective gates. Since we are considering transpose memory here, gates can be aligned either by rows or columns.

2.4 Fanout

Fanout occurs when an output memristor at any circuit depth (excluding primary output) has to drive multiple memristors. It can degrade performance as circuit depth increases and incurs additional energy overheads by introduction of extra cycles. For a given gate-level netlist, a single memristor cell can only be

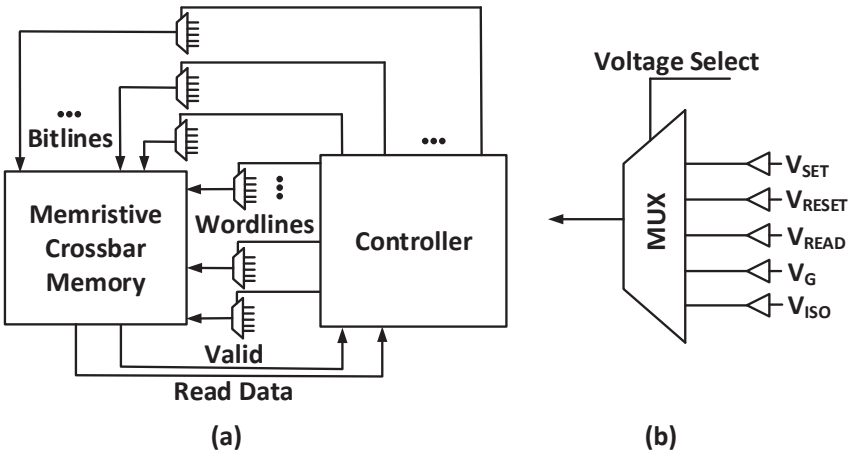

Fig. 3. (a) Memory processing unit (MPU), (b) Analog mux.

used either as an input or output of a gate. Therefore, the value that is stored in the output memristor if needed can work as input to multiple following memristors. A naïve approach can perform this fanout operation by multiple copies of the logic value as equal to number of the driven memristors (on different rows or columns than driving memristor). This copy operation introduces additional cycles, as proposed in MAGIC logic style where each copy operation requires cascade of two NOT operations. Therefore, it will require two extra memristors as well as two extra cycles for a single copy operation. As the number of fanout increases for a specific gate output, the number of copy operations, hence additional cycles also increases linearly. Moreover, the energy dissipation also increases due to the extra NOT operations. Works [7, 8] based in-memory logic synthesis did not address these performance related issues.

3 Proposed Approach

In this work, we propose a novel approach that will reduce the performance and energy overheads due to fanout in memristor based in-memory computing. The controller will apply relevant signals in proper sequence to the rows and columns in a crossbar architecture. Here, we consider two different cases based on the initialization of the output memristors at the copy locations.

- *Case 1:* Here, we maintain the same assumption as of works [8] and [12]. We consider that the output memristors at the copy locations are initialized to logic-1 which allows us to use a multiplexer through which controller can skip write cycle if the read value is one. With our approach, we have assumed on average 50% of the time, the sensed logic value is 1 which allows us to skip

the write cycle. The equations for total cycle counts and energy consumption are formed considering this factor.
- *Case 2:* Here, we consider that the output memristors at the copy locations are not initialized to logic-1. With our proposed approach, we exclude the multiplexer circuit hence removing the ability to skip write cycle (cycle 2) on average 50% of the time. We calculate the total cycle counts and energy requirement and compare with the previous work [8]. Previous works rely on copy operation which requires cascade of two NOT executions which in turn requires initialization of the output memristors of NOT gates to logic-1. Whereas with our approach, we do not need to initialize the output memristors at the copy locations to logic-1. This comparison demonstrates the trade-off between copy operation and direct read-write operation.

3.1 Overall Approach: Case 1

It will carry out the proposed method according to the following steps:

- *Cycle 1:* Controller reads the logic value of the output memristor which has fanout of two or more.
- *Cycle 2:* Controller writes the logic value as inputs to the driven memristors only if the sensed value is zero, otherwise, it can skip the cycle 2.

Previous works [7, 8] utilized a controller that produces a similar state diagram as shown in Fig. 4(a). For regular memory read and write operations, it goes to the read and write states respectively and for executing a logic function, it goes to the execution state. Execution state consists of micro operations where the controller executes the logic function in multiple cycles.

With our proposed approach, the number of state transitions will increase as shown in the Fig. 4(b) but the number of states will remain the same. While executing the micro operations in the execution state, whenever there is a fanout event, the controller will go to the read state and depending on the read value, it will decide whether to go to the write state or not. We can achieve this by modifying the controller circuit. The sensed bit can act as selector for a 2:1 multiplexer and depending on the select bit, controller will determine whether to write to the memory or not. If the read value is 1, the multiplexer will pass the input signal r_{e1} to the controller to go the execution state skipping the write state, otherwise, it will go to the write state and then to the execution state. This allows us to avoid writing logic-1 if the sensed logic value is 1.

3.2 Mapping Scenario Analysis: Case 1

We have made the following assumptions for three different mapping scenarios of a logic function on a memristor crossbar.

We only assume that a single copy of primary input is stored in a single memristor cell and following a fanout scenario from this cell, one can do the copy operation on-demand. Previous works assume that multiple copies of primary

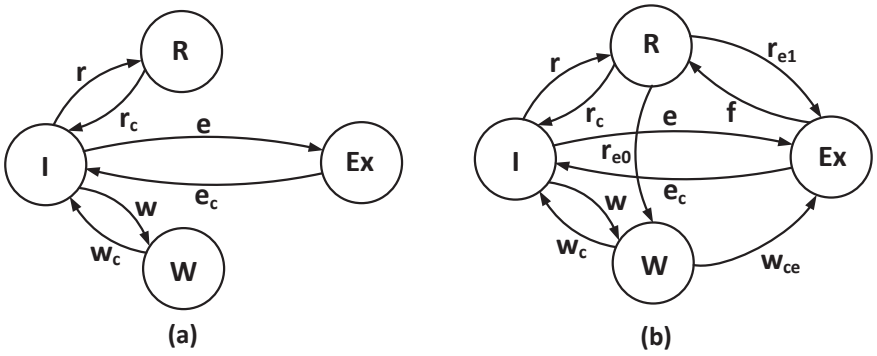

Fig. 4. Controller state diagram (Case 1). (a) Previous approach and (b) Proposed approach. The states are given as: I = Initial state, R = Read state, W = Write state, and Ex = Execution state; The different inputs denote the followings: r = read, r_c = read complete, w = write, w_c = write complete, e = execution, e_c = execution complete, f = fanout event, r_{e1} = read logic-1, r_{e0} = read logic-0, and w_{ce} = write complete and go to execution state.

inputs are already available depending on the number of gates they are driving. Hence, their assumptions underestimate the cycles required as they are able to eliminate the copy cycles produced from the fanouts of primary inputs.

Like the previous works [8, 13], we have considered the output memristors are initialized to logic value one at the beginning of the execution process which is a requirement for MAGIC logic style.

The work [8] has also initialized the output memristors to logic-1 required for copy operations (which eventually become the first input memristors of the driven gates). With our proposed approach, we are only considering fanout event related metrics not the whole synthesis procedure. Therefore, in our result analyses, we are not accounting the initialization cycles and energy consumption generated from initializing the output memristors. For fair comparison, we have maintained the same assumption while comparing with work [8]. In our work, the added consideration is, we can avoid writing logic value one. After reading the logic value of the driving memristor, if the controller finds the value as one, it can skip the write operation as the memristors are already initialized to one.

To demonstrate the efficacy of the approach, we have considered three different mapping scenarios (average, best, and worst) depending on the location of the second input memristors of the driven gates. To explain three scenarios, we consider the gate-level netlist shown in Fig. 5(a). We observe that there is a fanout of 4 from the node h. We also consider the logic value of h to be mapped on the location $(1, 1)$ in the memristor crossbar architecture.

Scenario 1 Here, we consider that all the second input memristors are aligned along the same row or same column. The mapped variable h in Fig. 5(b) needs to be copied 3 times to use as first input memristors of the driven gates and the

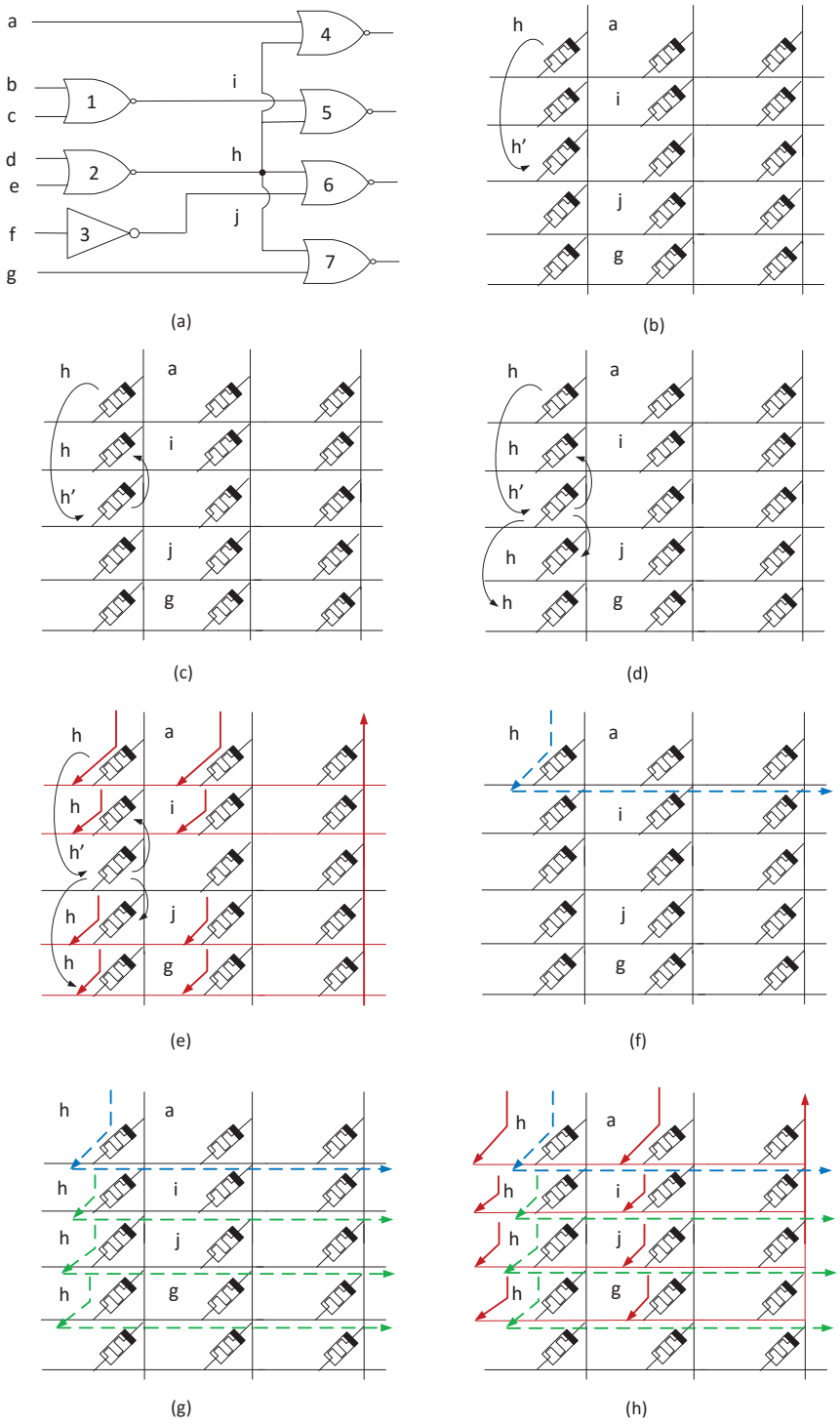

Fig. 5. Scenario 1 with Previous and Proposed Approach. (a) NOR/NOT based synthesized netlist. Regular Approach: (b) 1^{st} cycle: 1^{st} NOT execution to initiate the copy operation, (c) 2^{nd} cycle: 2^{nd} NOT execution to complete the first copy operation, (d) 3^{rd} & 4^{th} cycles: two NOT executions to complete all the copy operations, (e) execution of all the NOR gates in parallel. Proposed Approach: (f) 1^{st} cycle: read operation to sense the logic variable h, (g) 2^{nd} cycle: write operation to write h required number of times in one cycle, (h) 3^{rd} cycle: execution of all NOR gates in parallel. Here, black arrows denote the NOT execution required for copy operation, blue arrow denotes the read operation, green arrows denote the write operations and red arrows denote the execution of NOR gates.

variables a, i, j, and g are mapped as second input memristors on the second column of the crossbar. With the previous approach, it requires 4 cycles to copy logic value of h and now with all the input memristors of the driven gates in aligned position, it is possible to execute the gates in one cycle. The following equation can be used to estimate the total cycles required for copy operations with previous approach:

$$\sum_{n=2}^{N}[n * freq(n)] \tag{2}$$

where, n denotes different fanouts (i.e., number of gates it is driving), N is maximum fanout degree, and $freq(n)$ is the number of times, a fanout of n is found in a netlist.

With the proposed approach, we need 1 cycle to read the logic value and one cycle to write multiple copies of h along the same row or column (given that the read value is logic-0). We consider that on average 50% of the time the read value is 0 which allows us to consider only half of the write cycles to the total cycle count. Therefore, we can estimate the total cycles required with the equation given below:

$$1.5\sum_{n=2}^{N}[freq(n)] \tag{3}$$

It should be noted that with our proposed approach, we require at most two cycles (read and write) for each fanout event and this is true for all scenarios considered here. Therefore, for all three scenarios, we can reuse Equation 13 to estimate the total cycle counts.

We are providing a general formula to estimate the energy dissipation with both the previous and proposed approach due to the fanout only. According to the work [8], 52.49 fJ is the energy required for one NOT execution. Therefore, with previous approach, we can estimate the total energy dissipation by multiplying 52.49 fJ with the total cycle count (same as total NOT executions required for copy operations). The following equation gives us the total energy dissipation with this approach:

$$52.49\sum_{n=2}^{N}[n * freq(n)] \tag{4}$$

Whereas, for the proposed approach, the total energy required for a specific benchmark circuit can be estimated as:

$$\sum_{n=2}^{N}[\{(RE_m + (n-1) * WE)\} * freq(n)] \tag{5}$$

Where, RE_m denotes the total energy needed to read the variable and to decide (using multiplexer) whether to go to the write state or not. For fanout of n, we need to write $(n-1)$ memristors in a single cycle. Here, WE denotes the write energy required for writing a single memristor.

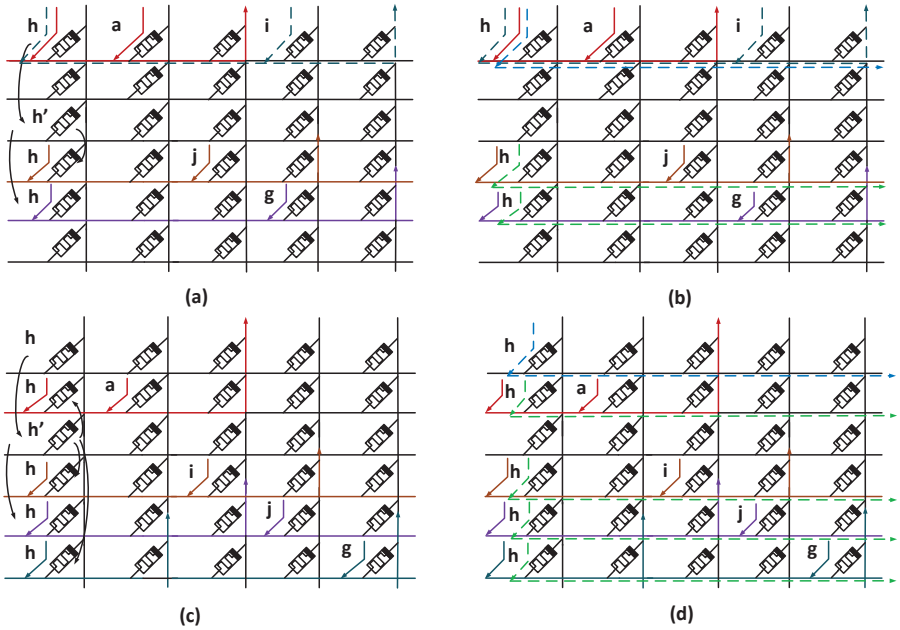

Fig. 6. Scenario 2: (a) previous approach, (b) proposed approach. Scenario 3: (c) regular approach, (d) proposed approach. Here, same color notation as Fig. 5 is maintained, the additional colors, orange, purple, and dark green represent different execution cycles and each composed of all the participating gates in that same cycle.

Scenario 2 We consider here some of the second input memristors are aligned on the same row as of the first input memristor (h) and some are scattered throughout the crossbar. Here, we divide different fanouts (n) according to the number of second input memristors residing on the same row. For example, for fanout of 3, 4, 5, and 6, we have considered 2 second input memristors are on the same row as first input memristor. From Fig. 6(a), it can be seen that the logic variables a and i that are mapped to the second input memristors of the gates 4 and 5 are aligned on the same row and variables j and g are in random locations. With the previous approach, h needs to be copied two times (3 NOT executions) to use it as an inputs of gates 6 and 7. We can evaluate the following equation for the previous approach:

$$2 + \sum_{i=1}^{M} \sum_{n=4i-1}^{4i+2} [\{n - (2i-1)\} freq(n)] \tag{6}$$

where, 2 cycles are required for fanout of 2. Here, i denotes the number of sub-divisions made on the different fanout and M denotes maximum degree of sub-division. For example, fanout of 4 falls under first sub-division (i=1) which requires 3 cycles (given by the term $\{n - (2i-1)\}$) to make two copies of the driving memristor. In this way, we find cycle count for each division and add

together to get the total cycle counts. As discussed earlier, with the proposed approach, we will get the cycle counts from Equation 13.

With the previous approach, as in Scenario 1, by simply multiplying the total cycle count with energy requirement of one NOT execution, we can estimate the total energy dissipation:

$$52.49 * [2 + \sum_{i=1}^{M} \sum_{n=4i-1}^{4i+2} [\{n - (2i-1)\} freq(n)]] \tag{7}$$

and for the proposed approach:

$$(RE_m + WE) + \sum_{i=1}^{M} \sum_{n=4i-1}^{4i+2} [\{RE_m + (n-2i) * WE\} freq(n)] \tag{8}$$

Where, first term $(RE_m + WE)$ is the energy required for fanout of 2. The term $(n - 2i)$ gives us the number of memristors that needs to be written in a single write cycle.

Scenario 3 For the third scenario, we consider that the second input memristors are scattered throughout the crossbar i.e., aligned neither horizontally nor vertically. From Fig. 6(c), we can see that variables a, i, j, and g are mapped on the memristors that are scattered on the crossbar. With the previous approach, it requires $(n + 1)$ copy cycles for a specific fanout of n and the total cycles required can be estimated as:

$$\sum_{n=2}^{N} [(n+1) freq(n)] \tag{9}$$

For the proposed approach, we use Equation 13 which gives us the same cycle counts as of scenario 1 and 2.

By maintaining the same procedure as of Scenarios 1 and 2, the total energy dissipation for the previous approach can be obtained. For the proposed approach, we need to write n memristors in a single write cycle for a fanout of n and the term $(n * WE)$ gives us the write energy required for this operation. For the previous and proposed approach, the energy dissipation can be estimated by the following two equations respectively:

$$52.49 * \sum_{n=2}^{N} [(n+1) freq(n)] \tag{10}$$

$$\sum_{n=2}^{N} [\{RE_m + (n * WE)\} freq(n)] \tag{11}$$

3.3 Overall Approach: Case 2

It will carry out the proposed method according to the following steps:

- *Cycle 1:* Controller reads the logic value of the output memristor which has fanout of two or more.
- *Cycle 2:* Controller writes the logic value as inputs to the driven memristors without considering whether the sensed logic value is one or not.

Previous works [7, 8] utilized a controller that produces a similar state diagram as shown in Fig. 7(a) and maintain the same operation as explained in section 3.1.

With our proposed approach, While executing the micro operations in the execution state, whenever there is a fanout event, the controller will go to the read state and then go to the write state to write the value to the driven memristors without considering whether the read logic value is one or not. Here, we do not use the 2:1 multiplexer, so, controller does not have the capacity to determine whether to skip cycle 2 or not. Controller will always go to the read state and then to the write state irrespective of the read logic value. We do not initialize the output memristors at the copy locations to logic-1 which saves us the initialization cycles and energy required for this operation.

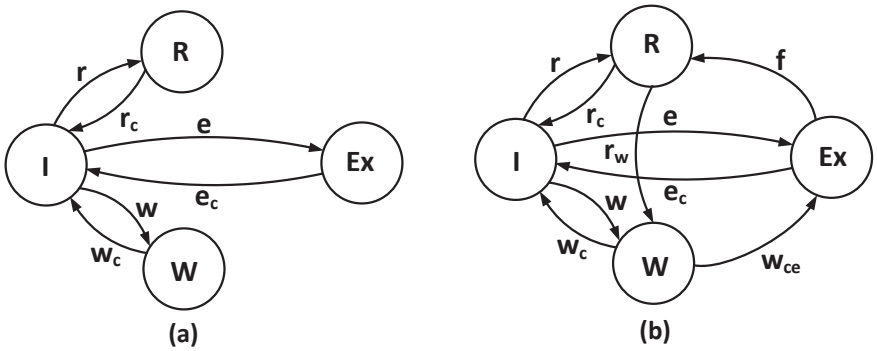

Fig. 7. Controller state diagram (Case 2). (a) Previous approach and (b) Proposed approach. The states are given as: I = Initial state, R = Read state, W = Write state, and Ex = Execution state; The different inputs denote the followings: r = read, r_c = read complete, w = write, w_c = write complete, e = execution, e_c = execution complete, f = fanout event, r_w = go to write state after reading logic value, and w_{ce} = write complete and go to execution state.

3.4 Mapping Scenario Analysis: Case 2

We maintain the same assumption as discussed in the section 3.2 except the third assumption that assume the output memristors at the copy locations are initialized to logic-1. With the exclusion of this assumption, controller now does

not have the ability to skip the write cycle as considered in case 1. Here, we discuss three different mapping scenarios of a logic function on a memristor crossbar.

Scenario 1 Here, we consider that all the second input memristors are aligned along the same row or same column. The mapped variable h in Fig. 5(b) needs to be copied 3 times to use as first input memristors of the driven gates and the variables a, i, j, and g are mapped as second input memristors on the second column of the crossbar. In case 2, we consider that output memristors at the copy locations are not initialized to logic-1. Hence, with the previous approach, it requires at least 1 cycle to initialize the memristors at the copy locations and 4 cycles to copy logic value of h. The following equation can be used to estimate the total cycles required for copy operations with previous approach:

$$\sum_{n=2}^{N}[(n+1)freq(n)] \tag{12}$$

where, n denotes different fanouts (i.e., number of gates it is driving), N is maximum fanout degree, and $freq(n)$ is the number of times, a fanout of n is found in a netlist.

With the proposed approach, we need one cycle to read the logic value and one cycle to write multiple copies of h along the same row or column. Therefore, we can estimate the total cycles required with the equation given below:

$$2\sum_{n=2}^{N}[freq(n)] \tag{13}$$

It should be noted that with our proposed approach, we require two cycles (read and write) for each fanout event and this is true for all scenarios considered here. Therefore, for all three scenarios, we can reuse Equation 13 to estimate the total cycle counts.

We are providing a general formula to estimate the energy dissipation with both the previous and proposed approach due to the fanout only. For case 2, we have to consider both the energy dissipation due to NOT execution and initialization of the output memristors at the copy locations. According to the work [8], 52.49 fJ is the energy required for one NOT execution, we can denote it by NEx and the energy required to set a memristor to logic 1 is 66.09 fJ, we can denote it by $init\_1$. For fanout of n, we need to initialize n number of memristors. The following equation gives us the total energy dissipation with this approach:

$$(NEx + init\_1)\sum_{n=2}^{N}[n * freq(n)] \tag{14}$$

Whereas, for the proposed approach, the total energy required for a specific benchmark circuit can be estimated as:

$$\sum_{n=2}^{N}[\{(RE + (n-1) * WE)\} * freq(n)] \tag{15}$$

Where, RE denotes the energy needed to read the variable. For fanout of n, we need to write $(n-1)$ memristors in a single cycle. Here, WE denotes the write energy required for writing a single memristor.

Scenario 2 We consider here the same mapping scenario as discussed in section 3.2, i.e. some of the second input memristors are aligned on the same row as of the first input memristor (h) and some are scattered throughout the crossbar. For details of this mapping scenario, we can refer to section 3.2 (scenario 2). Referring to Fig. 6(a), with the previous approach, first, an initialization cycle is needed to initialize the three output memristors at the copy locations and then h needs to be copied two times (3 NOT executions) to use it as an inputs of gates 6 and 7. We can evaluate the following equation for the previous approach:

$$3 + \sum_{i=1}^{M} \sum_{n=4i-1}^{4i+2} [\{n - 2(i-1)\} freq(n)] \tag{16}$$

where, 3 cycles are required for fanout of 2. Here, i denotes the number of sub-divisions made on the different fanout and M denotes maximum degree of sub-division. For a specific fanout of n, the term $\{n-2(i-1)\}$ gives us the total cycle count which includes the copy cycles and the initialization cycle. In this way, we find cycle count for each division and add together to get the total cycle counts. As discussed earlier, with the proposed approach, we will get the cycle counts from Equation 13.

With the previous approach, as in Scenario 1, we can estimate the total energy dissipation by the following equation:

$$(NEx + init\_1)[2 + \sum_{i=1}^{M} \sum_{n=4i-1}^{4i+2} [\{n - (2i-1)\} freq(n)]] \tag{17}$$

and for the proposed approach:

$$(RE + WE) + \sum_{i=1}^{M} \sum_{n=4i-1}^{4i+2} [\{RE + (n-2i) * WE\} freq(n)] \tag{18}$$

Where, first term $(RE + WE)$ is the energy required for fanout of 2. The term $(n-2i)$ gives us the number of memristors that needs to be written in a single write cycle.

Scenario 3 For the third scenario, we consider that the second input memristors are scattered throughout the crossbar i.e., aligned neither horizontally nor vertically. From Fig. 6(c), we can see that variables a, i, j, and g are mapped on the memristors that are scattered on the crossbar. With the previous approach, it requires 1 initialization cycle and $(n+1)$ copy cycles for a specific fanout of n and the total cycles required can be estimated as:

$$\sum_{n=2}^{N}[(n+2)freq(n)] \tag{19}$$

For the proposed approach, we use Equation 13 which gives us the same cycle counts as of scenario 1 and 2.

By maintaining the same procedure as of Scenarios 1 and 2, the total energy dissipation for the previous approach can be obtained. For the proposed approach, we need to write n memristors in a single write cycle for a fanout of n and the term $(n*WE)$ gives us the write energy required for this operation. For the previous and proposed approach, the energy dissipation can be estimated by the following two equations respectively:

$$(NEx + init\_1)\sum_{n=2}^{N}[(n+1)freq(n)] \tag{20}$$

$$\sum_{n=2}^{N}[\{RE + (n*WE)\}freq(n)] \tag{21}$$

4 Experimental Results

To validate the proposed approach, we performed a set of experiments on ISCAS'85 benchmark circuits and compared our results with a recently published work [8]. While doing so, we maintained the assumptions as stated in Section 3. In a given NOR/NOT based synthesized netlist, total number of gate executions consist of regular execution of gates pertaining only to the netlist and the extra NOT executions required for copy operations produced by fanout. We estimated the total number of extra NOT executions over the total number of gate executions for a specific benchmark circuit and then averaged over all the benchmark circuits. An average of 47.41%, 41.82%, and 54.40% additional cycle count due to NOT operations has been measured for scenario 1, scenario 2, and scenario 3 respectively. With our proposed approach, we are able to reduce performance and energy overheads produced by these aforementioned additional cycles.

- *Case 1:* Previous works such as [7, 8] have only considered energy dissipation due to gate executions, but as energy consumptions from read cycle, write cycle, and 2:1 multiplexer are inherent in our proposed approach, we have included these in our analysis. Performance and energy overheads for the multiplexer are obtained for a 45nm CMOS process technology from PTM using HSPICE and our circuit level evaluation shows that the delay introduced by a multiplexer is in the picosecond range. The work [8] has used the VTEAM model [14] to find the maximum latency for MAGIC NOR operations (when either of the inputs is at logic-1) and also to find the read time. We consider these times as the cycle time and read time of the system respectively. According to [15], the latency introduced by MAGIC operations are much higher than that of read and write operations. Hence, the effective

delay produced by the sense circuit and the multiplexer still stays below 1 cycle time. Therefore, in our analysis, we will not consider any performance overhead due to this added multiplexer.

– *Case 2:* In case 2, we do not initialize the output memristors at the copy locations to logic-1. This removes the 2:1 multiplexer from the system, as the controller is not aware of the logic value stored in the output memristors at the copy locations. In our result analysis, for the proposed approach, we exclude the energy consumed by the multiplexer and in case of previous approach, we add the energy required for initializing the memristor to logic-1. From [8], the energy required to write logic-1 to a memristor is 66.09 fJ. Therefore, with the previous approach, the total cycle count comes from the addition of initialization cycle and the copy cycles. With the proposed approach, we always need two cycles, one to read the logic value of the driving memristor and other to write the logic value to the driven memristors.

4.1 Experimental Setup

The experimental methodology we follow to validate our approach is as follows:

1. A given gate-level netlist is first synthesized with ABC tool [9] and target library of NOR and NOT gates.
2. For three different scenarios, number of fanout events has been found and cycle counts and energy dissipations estimation have been performed with the proposed method and compared with [8].

4.2 Results and Analysis

Table 1, Table 2, and Table 3 report the improvements over recently published work with respect to the number of cycles due to fanout in average, best, and worst scenarios respectively. For case 1, we observe average improvements of

Table 1. Comparison of Number of Cycles (Average Scenario) with the Regular and Proposed Approach.

Benchmark	Fanout Events	Number of Cycles Due to Fanout Only					
		Case 1: Scenario 1 (Average)			Case 2: Scenario 1 (Average)		
		Approach [8]	Proposed	Improvement (%)	Approach [8]	Proposed	Improvement (%)
c432	80	260	120	53.85	340	160	52.94
c499	215	602	323	46.35	817	430	47.37
c880	159	448	239	46.65	607	318	47.61
c1355	215	602	323	46.35	817	430	47.37
c1908	215	610	323	47.05	825	430	47.87
c2670	328	1024	492	51.95	1352	656	51.48
c3540	352	1343	528	60.69	1695	704	58.47
c5315	596	2193	894	59.23	2789	1192	57.26
c6288	1223	3544	1835	48.22	4767	2446	48.69
c7552	966	2924	1449	50.44	3890	1932	50.33
Average Improvements				51.08			50.94

Table 2. Comparison of Number of Cycles (Best Scenario) with the Regular and Proposed Approach.

Benchmark	Fanout Events	Number of Cycles Due to Fanout Only					
		Case 1: Scenario 2 (Best)			Case 2: Scenario 2 (Best)		
		Approach [8]	Proposed	Improvement (%)	Approach [8]	Proposed	Improvement (%)
c432	80	207	120	42.03	287	160	44.25
c499	215	499	323	35.27	714	430	39.78
c880	159	357	239	33.05	516	318	38.37
c1355	215	499	323	35.27	714	430	39.78
c1908	215	492	323	34.35	707	430	39.18
c2670	328	832	492	40.87	1160	656	43.45
c3540	352	1015	528	47.98	1367	704	48.50
c5315	596	1703	894	47.50	2299	1192	48.15
c6288	1223	2672	1835	31.32	3895	2446	37.20
c7552	966	2373	1449	38.94	3339	1932	42.14
Average Improvements				38.66			42.08

Table 3. Comparison of Number of Cycles (Worst Scenario) with the Regular and Proposed Approach.

Benchmark	Fanout Events	Number of Cycles Due to Fanout Only					
		Case 1: Scenario 3 (Worst)			Case 2: Scenario 3 (Worst)		
		Approach [8]	Proposed	Improvement (%)	Approach [8]	Proposed	Improvement (%)
c432	80	340	120	64.71	420	160	61.91
c499	215	817	323	60.47	1032	430	58.34
c880	159	607	239	60.63	766	318	58.49
c1355	215	817	323	60.47	1032	430	58.34
c1908	215	825	323	60.85	1040	430	58.65
c2670	328	1352	492	63.61	1680	656	60.95
c3540	352	1695	528	68.85	2047	704	65.61
c5315	596	2789	894	67.95	3385	1192	64.79
c6288	1223	4767	1835	61.51	5990	2446	59.17
c7552	966	3890	1449	62.75	4856	1932	60.22
Average Improvements				63.18			60.65

51.08%, 38.66%, and 63.18% in cycle reduction considering three mapping scenarios.

In case 2, we consider that the memristors at the copy locations are not initialized to logic-1 which dictates the need of adding initialization cycle to the total cycle counts. With the proposed approach, we need two cycles to read the driving memristor and then write the logic variable to the driven memristors in parallel. We observe average improvements of 50.94%, 42.08%, and 60.65% in cycle reduction.

Table 4, Table 5, and Table 6 report the improvements in energy consumption. For estimating the energy requirement, we have taken the same parameter values as discussed in [8]. The energy required for NOT execution is 52.49 fJ. The energy required to read logic-0 is 0.03 fJ, while that for logic-1 is 3.34 fJ. For our estimation purpose, we have averaged these two values and considered the read energy as 1.685 fJ. The energy required to write logic-1 to a memristor

Table 4. Comparison of Energy Dissipation (Average Scenario) With the Regular and Proposed Approach.

Benchmark	Fanout Events	Energy Dissipation in pJ Due to Fanout Only					
		Case 1: Scenario 1 (Average)			Case 2: Scenario 1 (Average)		
		Approach [8]	Proposed	Improvement (%)	Approach [8]	Proposed	Improvement (%)
c432	80	28.82	2.32	91.95	65.11	8.00	87.71
c499	215	52.60	4.63	91.20	118.83	16.90	85.78
c880	159	43.46	3.62	91.67	98.19	12.76	87.00
c1355	215	52.60	4.63	91.20	118.83	16.90	85.78
c1908	215	54.69	4.76	91.30	123.57	17.29	86.00
c2670	328	106.19	8.77	91.74	239.91	30.77	87.17
c3540	352	133.85	11.82	91.17	302.41	43.28	85.68
c5315	596	219.20	19.19	91.25	495.23	69.82	85.90
c6288	1223	265.02	25.86	90.24	598.761	99.76	83.34
c7552	966	268.38	23.51	91.24	606.351	85.59	85.88
Average Improvements				91.30			86.03

Table 5. Comparison of Energy Dissipation (Best Scenario) With the Regular and Proposed Approach.

Benchmark	Fanout Events	Energy Dissipation in pJ Due to Fanout Only					
		Case 1: Scenario 2 (Best)			Case 2: Scenario 2 (Best)		
		Approach [8]	Proposed	Improvement (%)	Approach [8]	Proposed	Improvement (%)
c432	80	10.87	1.28	88.22	24.55	5.34	78.25
c499	215	26.19	2.93	88.81	59.18	12.01	79.71
c880	159	18.74	2.06	89.01	42.34	8.39	80.18
c1355	215	26.19	2.93	88.81	59.18	12.01	79.71
c1908	215	25.83	2.87	88.89	58.35	11.72	79.92
c2670	328	43.67	5.09	88.34	98.67	21.22	78.49
c3540	352	53.28	6.54	87.73	120.37	27.78	76.92
c5315	596	89.39	10.93	87.77	201.96	46.39	77.03
c6288	1223	140.25	15.19	89.17	316.87	61.47	80.60
c7552	966	124.56	14.30	88.52	281.41	59.31	78.92
Average Improvements				88.53			78.97

Table 6. Comparison of Energy Dissipation (Worst Scenario) With the Regular and Proposed Approach.

Benchmark	Fanout Events	Energy Dissipation in pJ Due to Fanout Only					
		Case 1: Scenario 3 (Worst)			Case 2: Scenario 3 (Worst)		
		Approach [8]	Proposed	Improvement (%)	Approach [8]	Proposed	Improvement (%)
c432	80	17.85	5.57	68.80	40.32	23.13	42.63
c499	215	42.88	10.04	76.59	96.89	42.12	56.53
c880	159	31.86	8.36	73.76	71.98	34.86	51.57
c1355	215	42.88	10.04	76.59	96.89	42.12	56.53
c1908	215	43.30	10.46	75.84	97.84	43.81	55.22
c2670	328	70.97	20.44	71.20	160.33	85.18	46.87
c3540	352	88.97	25.54	71.29	201.01	107.18	46.68
c5315	596	146.40	41.88	71.39	330.75	175.56	46.92
c6288	1223	250.22	49.86	80.07	565.32	211.61	62.57
c7552	966	204.19	51.28	74.89	461.32	214.95	53.41
Average Improvements				74.04			51.89

which is termed as SET operation is 66.09 fJ and through RESET operation, we can write logic-0 to a memristor and the energy required is 17.60 fJ. For case 1, as discussed in Section 3, we do not need to write logic-1 with our proposed approach, thus, no energy is consumed. Therefore, the average energy considered for SET and RESET operation is 8.8 fJ. For a specific benchmark circuit, we find all different fanout occurrences and add the energies required to estimate the total energy dissipation. We observe average improvements of 91.30%, 88.53%, and 74.04% by experimental evaluation on ISCAS'85 benchmark circuits. This improvement can be primarily attributed to the fact that we were able to eliminate the need for writing logic-1.

For case 2, with the previous approach, the memristors at the copy locations need to be initialized to logic-1. Therefore, this initialization energy needs to be accounted for in the total energy calculation. With the proposed approach, we exclude energy consumption produced by the multiplexer that is required in case 1. The only energies required are the energy to read the driving memristor and energy to write the driven memristors. We observe average improvements of 86.03%, 78.97%, and 51.89% in energy saving.

5 Conclusions

In this work, we outline an effective approach that would significantly reduce the performance and energy overheads due to fanout while mapping a logic function in memristor based crossbar architecture. Comparison has been made with a recently published work and shows significant average improvements in average, best, and worst mapping scenarios in performance and energy dissipation. Future research direction would be to implement the entire logic synthesis process utilizing the fanout optimization discussed here.

References

1. Wulf, W. A. and McKee, S. A.: Hitting the memory wall: implications of the obvious.: ACM SIGARCH computer architecture news, vol. 23(1), 20–24 (1995)
2. Hur, R. B. and Kvatinsky, S.: Memristive memory processing unit (MPU) controller for in-memory processing. In: 2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE), pp. 1–5 Nov (2016)
3. Kvatinsky, S. and Friedman, E. G. and Kolodny, A. and Weiser, U. C.: The Desired Memristor for Circuit Designers. : IEEE Circuits and Systems Magazine, vol. 12(2), pp. 17–22 Secondquarter (2013)
4. Lehtonen, E. and Poikonen, J. H. and Laiho, M. : Memristive stateful logic. In: Memristor Networks, pp 603–623. Springer (2014)
5. Talati, N. and Gupta, S. and Mane, P. and Kvatinsky, S.: Logic Design Within Memristive Memories Using Memristor-Aided loGIC (MAGIC).: IEEE Transactions on Nanotechnology, vol. 15(4), 635-650 July (2016)
6. Nair, R.: Evolution of Memory Architecture.: Proceedings of the IEEE, vol. 103(8), pp. 1331-1345 Aug (2015)

7. Hur, R. B. and Wald, N. and Talati, N. and Kvatinsky, S.: Simple magic: Synthesis and in-memory Mapping of logic execution for memristor-aided logic.: 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 225-232 Nov (2017)
8. Gharpinde, R. and Thangkhiew, P. L. and Datta, K. and Sengupta, I.: A Scalable In-Memory Logic Synthesis Approach Using Memristor Crossbar.: IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26(2), 355-366 Feb (2018)
9. Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 90215. `http://www.eecs.berkeley.edu/~alanmi/abc/`
10. Zaman, M. A. and Katkoori, S.: Minimizing Performance and Energy Overheads Due to Fanout In Memristor based Logic Implementations.: 2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pp. 7-12 October (2018)
11. Chua, L.: Memristor-The missing circuit element.: IEEE Transactions on Circuit Theory, vol. 18(5), 507-519 September (1971)
12. Strukov, D. B. and Snider, G. S. and Stewart, D. R. and Williams, R. S.: The missing memristor found.: nature, vol. 453(7191), 80–83 (2008)
13. Ali, A. H. and Hur, R. B. and Wald, N. and Kvatinsky, S.: Efficient Algorithms for In-Memory Fixed Point Multiplication Using MAGIC. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 May (2018)
14. Kvatinsky, S. and Ramadan, M. and Friedman, E. G. and Kolodny, A.: VTEAM: A General Model for Voltage-Controlled Memristors.: IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 62(8), 786–790 Aug (2015)
15. Talati, N. and Ali, A. H. and Hur, R. B. and Wald, N. and Ronen, R. and Gaillardon, P. E. and Kvatinsky, S.: Practical challenges in delivering the promises of real processing-in-memory machines.: 2018 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1628-1633 March (2018)