Logic Synthesis and Defect Tolerance for Memristive Crossbar Arrays

1st Onur Tunali
Nanoscience and Nanoengineering
Istanbul Technical University, Istanbul, Turkey
onur.tunali@itu.edu.tr

2nd Mustafa Altun
Electronics and Communication Engineering
Istanbul Technical University, Istanbul, Turkey
altunmus@itu.edu.tr

***Abstract*—Contrary to abundant memory related studies of memristive crossbar structures, logic oriented applications are only gaining popularity in recent years. In this paper, we study logic synthesis, regarding both two-level and multi level designs, and defect aspects of memristor based crossbar architectures. First, we introduce our two-level and multi-level logic synthesis techniques. We elaborate on advantages and disadvantages of both approaches with experimental results regarding area cost. After that, we devise a defect model in alignment with the conventional stuck-at open and closed paradigm. In addition, we determine the effects of defects to the operational capacity of the crossbar. Furthermore, we propose a preliminary defect tolerant Boolean logic mapping approach. In order to evaluate our approach, we conduct extensive Monte Carlo simulations with industrial benchmarks. Finally, we discuss future directions concerning both existing two-level and prospective multi-level logic designs as well as defect tolerance with area redundancy.**

***Index Terms*—memristive crossbar, memristor, logic synthesis, defect tolerance,**

I. Introduction

Due to the scaling issues of current CMOS based technologies, researchers have explored novel approaches as computing elements such as the focus of this paper "memristors" [1] [2]. Even though theoretical manifestation of memristor goes back to 1970s [3], physical realization of an actual circuit component is very recent and established by HP [4]. After this initial step, a variety of memristor based logic circuit designs are proposed such as Boolean logic, implication logic, and threshold logic. A comprehensive review and references of memristor based logic circuits can be found in [5]. In this paper, we focus on logic and defect aspects regarding crossbar arrays using memristors as switching elements.

First study using hysteric resistors as memristive component is demonstrated in [6]. An integrated design approach is devised in [7] showing the implementation of arbitrary Boolean logic functions and their mapping on crossbar array. As a following study, an elaborated mapping method considering larger logic functions is presented in [8] by the same author. Mentioned work adopts a two level logic design using NAND - AND planes obtaining the negation of a logic function with a final inversion. However, authors overlooked the fact that the crossbar is able to produce the logic function and its negation as outputs, so considering both cases during mapping process would generate a potential optimization in terms of area cost. Furthermore, by modifying the computing states we show that it is possible to achieve multi-level logic design which uses the outputs of NAND gates computed in the previous level as inputs. This also reveals a novel area cost prospect.

Regarding defect issues of crossbar arrays, to our knowledge no other study focuses on the defects and their affects to logic circuits realized with memristor crossbar arrays. Certain works concentrate on the robustness of memristor based logic circuits [9] and logic gates [10], but disregards any operational or switch defects. However, defects occur in crossbar degrade operational capacity of switches and complicate the logic mapping process severely similar to memory defects examined in [11] [12]. To tackle this challenge, we devise a defect model concentrating on faulty switches and their affects to logic mapping. Borrowing the common terminology, stuck-at open and closed type defects are defined. Next, we formalize the defect tolerant logic mapping process and propose a hybrid algorithm utilizing the combination of a heuristic matching and an assignment method. Indeed, this problem is very similar to the logic mapping of reconfigurable nano-crossbar arrays (using AND-OR logic) for which a quite mature literature exists [13] [14]. However, the most of mentioned studies focus on a single plane of crossbar particularly AND and neglect OR plane. Furthermore, the related algorithms operate using 1.5 times larger size crossbars and show poor performance for optimum size crossbars. Motivated by these shortcomings, we propose a hybrid algorithm, combination of heuristic and exact algorithms, for optimum size crossbars. Our algorithm covers the whole crossbar array by applying a heuristic approach for the NAND and AND plane with an exact assignment technique for the output connections of given logic functions which is more critical since a single defect might discard a whole output. As a summary, the main contributions of this paper are as follows:

- A multi-level logic design is demonstrated and area cost comparison with existing two-level design is conducted;
- A defect model is established and a preliminary hybrid defect tolerant logic mapping algorithm is proposed;
- Area optimization with considering both the logic function and its negation during mapping is shown; and

This work is part of a project that has received funding from the European Union's H2020 research and innovation programme under the Marie Skodowska-Curie grant agreement No 691178. This work is supported by the TUBITAK-Career project #113E760.

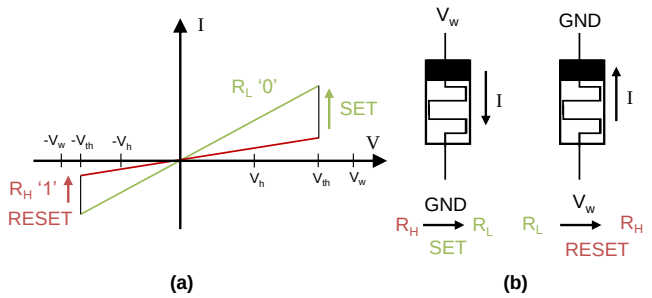

Fig. 1. I-V characteristics (a) and switching operations of a memristor (b).

- Future aspects of both the two-level and multi-level logic synthesis with area cost and redundancy optimization (yield analysis) according to defect rates are discussed.

The rest of this paper is organized as follows. Section II describes memristor model and crossbar based logic circuits. Section III demonstrates the two-level and multi-level logic designs with simulation results. Section IV proposes a defect model, effects of defects, and the defect tolerance algorithm. Section V presents the experimental results for defect tolerant logic mapping and area cost. Finally, Section VI discusses conclusions and future aspects.

II. Background and Previous Works

A. Memristor Model

Memristor is a non-linear electrical component which shows resistive switching properties. Depending on the physical characteristics, switching can be smooth or abrupt. In short, a memristor preserves its sate without exterior influence. So when it is SET or RESET, memristor keeps its state unless voltage difference between the terminals of component is inside the defined constraints. Due to this inherent feature, it is a likely candidate for a variety of applications such as non-volatile memory [15], dynamic load [16], neuromorphic systems [1] and etc. Fig. 1 shows the I-V characteristics and switching behavior of an ideal memristor. In this paper, we use Snider Boolean Logic model which regards a lower resistance R_{ON} as logic 0 and a high resistance R_{OFF} as logic 1 [6].

B. Memristive Crossbar Array

Briefly, a crossbar array is constructed from two layers of orthogonal wires/lines. Every crosspoint/junction acts as a switching element which is a memristor in this study. Memristor based crossbar array for logic circuits proposed by Xie in [7] is able to implement arbitrary logic function in Sum-of-Products (SOP) form. Two types of memristors are necessary for primary operations: active memristors which can switch, and disabled memristors which are permanently in the high resistance state that is used as an assumed component in both [6] [7]. There are four distinct regions of the crossbar reserved for specific operations: Input Latch (IL), NAND plane, AND plane and Output Latch (OL). If a given logic function is denoted with $f = m_1 + ... + m_k + ... + m_n = \overline{m_1}...\overline{m_k}...\overline{m_n}$, a diagram of the crossbar with plane annotations is given in Fig. 2(a). Note that, minterm and product concepts are used

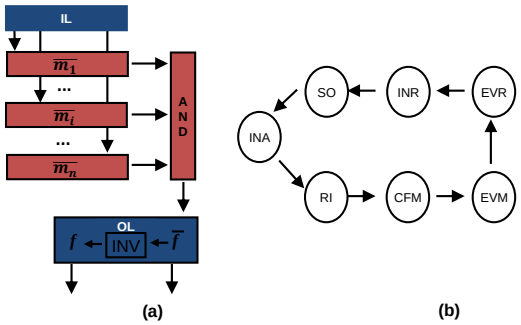

Fig. 2. Two-level logic design of memristor crossbar arrays (a) and state machine diagram (b).

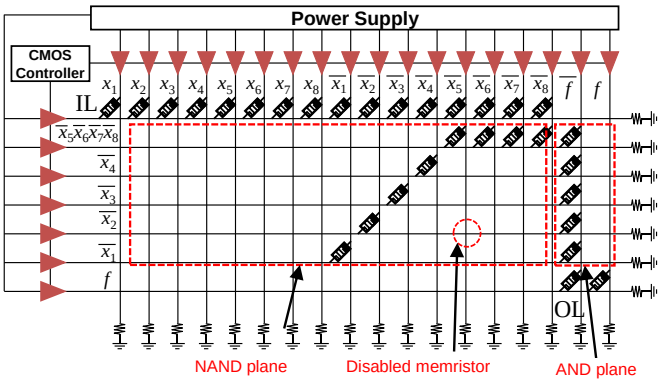

Fig. 3. Logic mapping of a Boolean function on a crossbar with two-level design.

interchangeably in the literature (although they are fundamentally different), so we follow the same tendency in this paper as well. In order to compute a given input, step by step process is as follows: 1) INA: Initialize all the memristors to R_{OFF}; 2) RI: IL block receives inputs from CMOS controller or a previous OL; 3) CFM: All minterms (products) are configured by copying values of IL simultaneously; 4) EVM: Evaluate all minterms configured in NAND plane and write to AND plane; 5) EVR: Evaluate the results of AND plane which calculates the $\overline{f}$; 6) INR: Invert the results to obtain f from $\overline{f}$; and 7) SO: Send outputs to OL.

State machine diagram of computation process is given in Fig. 2. For every computation step, certain voltage values are applied to horizontal and vertical lines according to CMOS controller circuit. Reader is encouraged to refer [7] for further information.

C. Logic Synthesis

Logic synthesis process of a crossbar is consisted of choosing which switches to activate and disable in order to implement a given Boolean function. A memrsitor switch can be programmed into two operational range:

- *active*: Memristor can switch between two resistive states (low R_{ON} and high R_{OFF})
- *disabled*: Memristor always stays in R_{OFF} sate regardless of voltage difference.

Given a Boolean function $f = \overline{x_1} + \overline{x_2} + \overline{x_3} + \overline{x_4} + \overline{x_5}\ \overline{x_6}\ \overline{x_7}\ \overline{x_8}$, implementation is given in Fig. 3. Horizontal and vertical lines represent the minterms and inputs respectively.

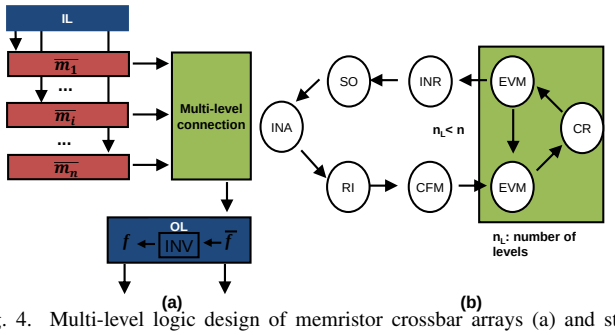

Fig. 4. Multi-level logic design of memristor crossbar arrays (a) and state machine diagram (b).

If an input is present in the minterm, corresponding switch is activated, otherwise disabled. In addition, we introduce two parameters used in both multi-level design and defect tolerance performance of crossbars:

- *Area cost*: The size of the crossbar used to implement a given logic function
- *Logic Inclusion Ratio* (IR): The ratio of the number of switches (memristors) used to realize a logic function to area cost

Using the example in Fig. 3, crossbar has 7 horizontal lines and 18 vertical lines so area cost is 126. There are 31 memristors used in implementation, so IR = $\frac{31}{126}$ = 25%.

III. Multi-Level Logic Synthesis

So far proposed architectures use two-level NAND-AND design to implement a given Boolean function [7] [8]. Certain studies such as [17] [18] explore the multi-level logic design, however rather than using the connection capabilities of a single crossbar they use multiple crossbars to generate intermediate values and cascade them to obtain outputs. Instead of the explained approach, we show that it is possible to modify a crossbar to obtain multi-level design by introducing minterm dependent computation cycles.

A. Proposed Multi-level Design

A diagram of multi-level design is given in Fig. 4 (a). We use multi-level connections in place of AND plane utilized in two-level approach. By activating corresponding memristors, a minterm evaluation (EVM) result can be fed as input to another minterm (horizontal line). Key point is to evaluate minterms one-by-one instead of evaluating all of them simultaneously. Computation steps are shown as a state machine in Fig. 4 (b). Extra state CR (copy result) writes the result of the minterm evaluation to next level minterm as input. Conditional constraint $n_L < n$ (n_l number of levels and n computation step) ensures that computation steps proceed to next state unless all minterms are evaluated.

As an example we use the same $f = \overline{x_1} + \overline{x_2} + \overline{x_3} + \overline{x_4} + \overline{x_5}\ \overline{x_6}\ \overline{x_7}\ \overline{x_8}$ function in Fig. 3 and synthesis it with a multi-level design. Physical implementation shown in Fig. 5 demonstrates a crossbar with 3 horizontal lines and 19 vertical lines, so area cost is 59 . By using multi-level design, we are able to reduce the area cost less than half of the two-level design which has an area cost of 126.

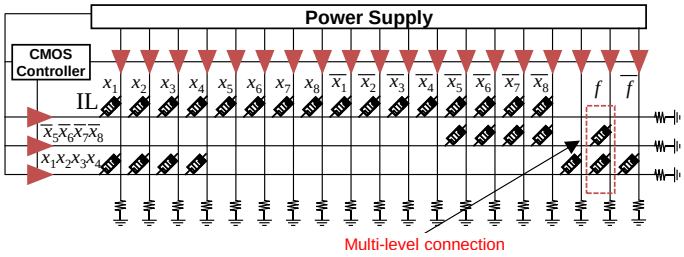

Fig. 5. Logic mapping of a Boolean function on a crossbar with multi-level design.

B. Results of Area Cost Simulations

To further examine the cost advantages of multi-level, we conduct Monte Carlo simulations by randomly generating Boolean functions and using Berkley ABC [19] logic synthesis tool to acquire a gate level technology mapping. We force ABC to use a set of NAND gates (which have fan-in sizes 2 to n that is determined according to input size of a given logic function), so implementation of a logic function is achievable with a memristor based crossbar array. Using a MATLAB™ script, we are able to obtain the area cost of random Boolean functions expressed with n-input NAND gates by ABC.

Results of our simulations are shown in Fig. 6. We use a sample size of 200 for each simulation and input size of 8 through 15. Cost results are sorted according to number of products increasingly. Success rate indicates the percentage of sample size which have a smaller muti-level area cost than two-level design. It is clear from the graphs that, two trends emerge regarding input size and product number parameters of the Boolean functions. When the input size increases, it is more challenging to find a superior multi-level design so success rate drops. When the product size increase, it is easier to find a superior multi-level design. As can be seen from the graphs, the number of samples staying under the two-level area cost line increases as progressed towards the right side meaning larger number of products.

As for the disadvantages, so far we use only single output logic functions and exclude multi-output logic functions such as benchmark functions. To secure a fair assessment, we conduct an area cost comparison of benchmark circuits most of which has multi-output logic circuits presented in [20]. We present a portion of our results in Table I. Even though we choose the examples with most moderate results, area cost difference is still drastic when multi-output functions are considered. Only exceptions are *t481* and *cordic* which have single output and 2 outputs respectively. Since conventional EDA tools are used for technology mapping, satisfactory results cannot be obtained for larger and multi-output benchmark functions. Field of memristor specific technology dependent mapping tools are open to further research.

IV. Defect Aspects of Memristor Crossbar Array

Due to stochastic nature of nano-fabrication, a number of variations might occur during production depending on used materials, alignment issues, integration, etc.. Mentioned physical alterations might cause variations in the operational

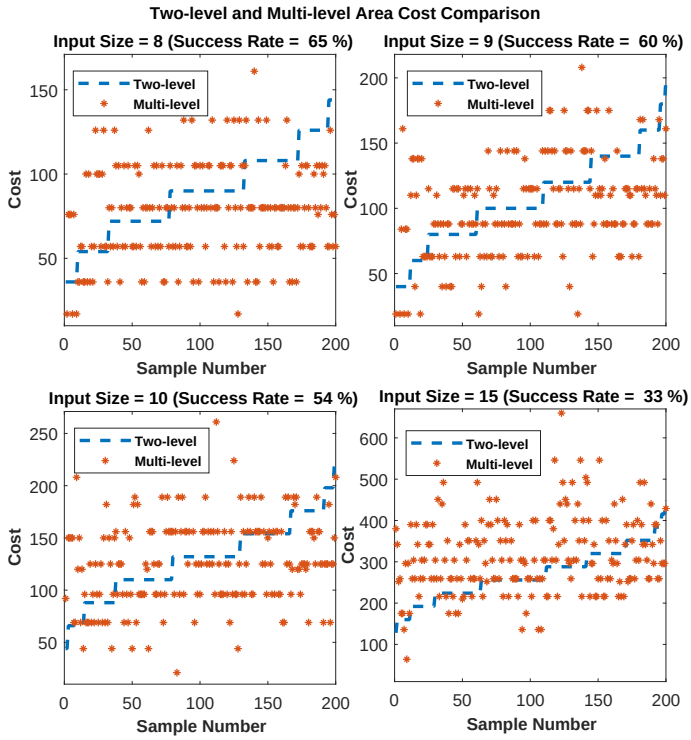

Fig. 6. Area cost comparison of two-level and multi-level designs. The results are sorted in ascending order according to number of products in a sample. Flat lines shows the equal number of products for the samples, so the area cost of two-level design is constant, while multi-level design results fluctuate.

TABLE I
TWO-LEVEL AND MULTI-LEVEL AREA COST COMPARISON OF BENCHMARK FUNCTIONS

Bench Name	Original Circuit		Negation of Circuit	
	Two-level	Multi-level	Two-level	Multi-level
rd53	544	3000	560	2000
con1	198	480	198	527
misex1	570	4836	1590	4161
bw	3300	52875	3564	53110
sqrt8	1008	2745	792	3300
rd84	6216	48124	7128	20276
b12	2496	7800	2064	2691
t481	16388	**5760**	12274	**8034**
cordic	45800	**9594**	59650	**10668**

capacity of memristors, permanent defects or transient faults in wires and switches of memristive crossbar array. However for the sake of simplicity, we only explore the switching defects in this section.

A. Defect Model

A defective switch cannot operate properly meaning no switching between different states, so a switch defect of memristors might be modeled as stuck-at open and stuck-at closed type. Physical resistance equivalence of defect types can be defined as follows:

- *stuck-at open*: Memristor is always in R_{OFF} mode which means high resistance
- *stuck-at closed*: Memristor is always in R_{ON} mode which means low resistance

Stuck-at open defects show the same characteristics with the disabled memristors used in mapping phase and avoiding

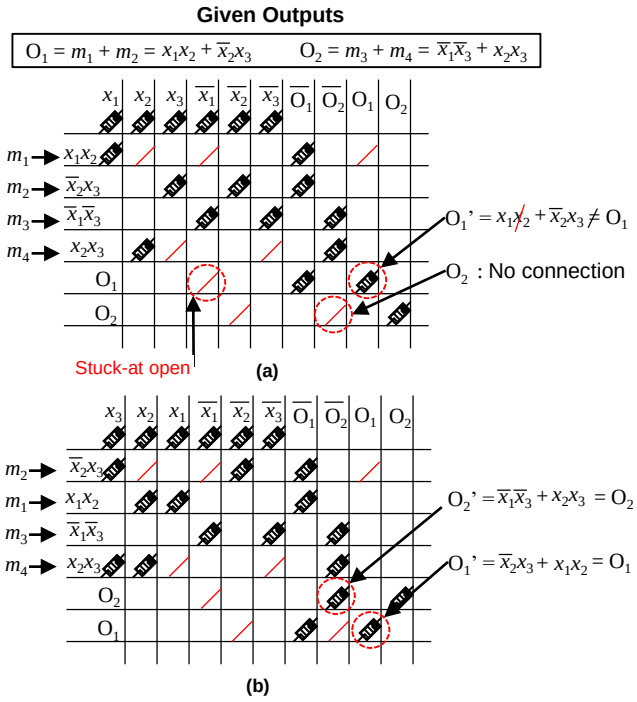

Fig. 7. Logic mapping of a given Boolean function. Red diagonal lines represent a defect on the crosspoint. First mapping is applied by disregarding defects without a valid mapping (a) and second mapping is applied by considering defects with a valid mapping (b).

them during logic mapping is adequate for a valid mapping. However, stuck-at closed defects are always R_{ON}, so they disrupt the operational capacity of both horizontal and vertical lines. Since every computation step starts with initializing all the memristors to R_{OFF} and then copying the input values to minterms, vertical line belonging to the defective switch cannot be used. Furthermore, R_{ON} is equivalent to a logic 0 in Snider Boolean Logic model [6], so every horizontal line which computes a NAND gate outputs a logic 1 independent of the other inputs. For this reason when a stuck-at closed defect is present, horizontal line belonging to the defective switch cannot be used as well. Because of the mentioned challenges, tolerance of stuck-at closed defects is not possible without any redundant crossbar lines. Yield analysis concerning the relationship between area cost with redundant lines and defect tolerance performance is open for future research.

B. Defect Tolerant Logic Synthesis Method

In a defective crossbar, certain switches cannot be programmed as desired (switching or disabled), so logic synthesis process must consider defective switches. In Fig. 7(a), a naive mapping approach disregarding defects is applied for given logic functions and an invalid implementation is obtained. However, in Fig 7(b), after careful consideration a valid mapping is produced with activating correct switches. Before explaining our methodology, we introduce the following concepts used in our algorithm:

1) *Function matrix* (FM) is a representation of a logic function in sum-of-products form. If an input occurs in a

minterm, it is denoted with 1; otherwise 0 is assigned. Fig. 8(a) shows an example of an FM.

2) *Crossbar matrix* (CM) is a representation of a crossbar showing either defective or functional switches. Fig. 8(b) shows a CM that can be also referred as a defect map. *Functional switches* are denoted with 1's that can be matched with 1's and 0's in an FM. *Stuck-open switches* are denoted with 0's that can only be matched with 0's in an FM.

3) *Row matching* checks a row of FM and CM element-by-element. If every element of rows complies with the matching rules given above, then a minterm can be matched to horizontal line of crossbar.

4) *Matching matrix* shows valid row matchings of FM and CM. This is similar to a cost matrix used in assignment problems having an objective of minimizing the total cost. Fig. 8(c) shows a matching matrix of function and crossbar matrices in Fig. 8(a) and (b), respectively. A 0 and 1 elements of the matrix respectively show that a matching is possible and there is no matching.

To generate a valid mapping we are using a hybrid algorithm due to runtime issues. Constructing a matching matrix and applying an assignment method to all rows as shown in Fig. 8(d) would increase computational load of the algorithm excessively for larger logic functions. We will show drastic runtime differences in experimental results.

In short, our algorithm is composed of three parts: First, area cost of the logic function and its negation is calculated. Smaller case is chosen for implementation. Second, a heuristic matching is applied to all minterm (product) rows of FM (denoted with FM_m Fig. 8(a)) which performs row by row matching between FM_m and CM from top to bottom. During the process, matched rows of the CM are traced with an array showing which rows of the FM_m are assigned to them. At first, the matching searches only unmatched rows. If an FM_m row can not be matched with the unmatched rows of the CM, then backtracking starts by considering the matched rows of the CM from top to bottom. If a matching is found, the previously assigned row of the FM_m is checked once whether it can be assigned to an unmatched row of the CM. If this check results in a mismatch then the algorithm continues with the next matched row of the CM and repeats the same process to find a valid matching. As a final step, a matching matrix for output rows of FM (denoted with FM_o Fig. 8(a)) and unmatched rows of CM (denoted with CM_u) is constructed. By using an assignment algorithm choosing which O_i is mapped to H_k yielding a zero cost, we ensure every output has valid row matching. We use Munkres' algorithm [21] for finding an assignment producing zero cost. This is an exact algorithm which means if a zero cost is possible, it will be found by it.

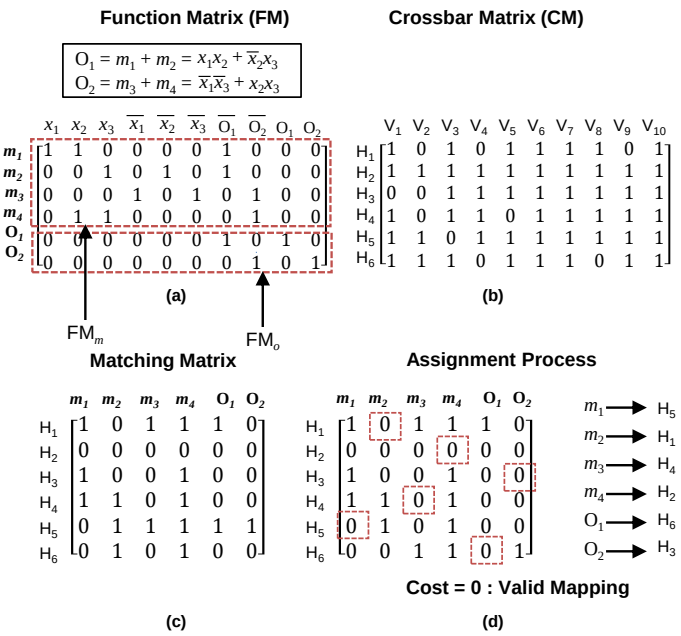

Fig. 8. Function matrix (a) crossbar matrix (b) matching matrix (c) and a valid assignment (d).

Algorithm 1 Hybrid Mapping Methodology

```
1:  Input : FM, CM
2:  Output: row_assignment
3:  minterm_mapping = false
4:  valid_mapping = false;
5:  matched_rows ={}
6:  for i = 1 to N do
7:      flag = false;
8:      F_i ← i^th row of FM_m
9:      for t = 1 to N and C_t ∉ matched_rows do
10:         Try matching F_i with C_t
11:         if a matching exist then
12:             flag = true;
13:             break
14:         end if
15:     end for
16:     if ¬ flag then
17:         flag = BACKTRACKING(Row_i)
18:     end if
19:     if ¬flag then
20:         no possible row matching
21:         break
22:     end if
23: end for
24: if all rows of FM_m are matched then
25:     minterm_mapping = true
26: end if
27: if minterm_mapping then
28:     matching_matrix = MATCHINGCHECK(FM_o, CM_u)        ▷ Matching matrix construction
29:     [cost, assignment] = MUNKRES(matching_matrix)      ▷ Assignment algorithm [21]
30:
31:     if cost = 0 then
32:         row_assignment = assignment
33:         valid_mapping = true;
34:     end if
35: end if
```

V. Experimental Results

To obtain experimental results, we generate defective crossbars with assigning an independent defect probability/rate to each crosspoint that shows a uniform distribution. As opposed the common tendency of using 1.5 times larger crossbars for logic mapping [13] [14], we utilize optimum size crossbars for mapping meaning no redundant lines are present. We only include stuck-at open defects since our simulation regards a minimum area cost and defect tolerance is not possible in terms of stuck-at closed types as mentioned in previous section. Monte Carlo simulations are performed for assessment with a sample size of 200. We observe that fluctuating of

TABLE II
SUCCESS RATE (%) AND RUNTIME VALUES (IN SECONDS) OF PROPOSED HYBRID ALGORITHM (**HBA**) AND EXACT ALGORITHM (**EA**) FOR OPTIMUM AREA CROSSBARS WITH 10% DEFECT RATE

Name	I	O	P	Area Cost	IR	HBA		EA	
						Psucc	Time	Psucc	Time
rd53	5	3	31	544	33%	98%	0.001	98%	0.001
squar5	5	8	25	**858**	16%	100%	0.001	100%	0.001
bw	5	8	22	330	12%	100%	0.002	100%	0.003
inc	7	9	30	1248	17%	100%	0.001	100%	0.002
misex1	8	7	12	570	19%	100%	0.001	100%	0.001
sqrt8	7	4	29	**792**	21%	100%	0.001	100%	0.002
sao2	10	4	58	1736	29%	94%	0.001	97%	0.003
rd73	7	3	127	2600	34%	78%	0.002	92%	0.013
clip	9	5	120	3500	23%	76%	0.005	79%	0.082
rd84	8	4	255	6216	33%	82%	0.006	89%	0.093
ex1010	10	10	284	11760	23%	100%	0.003	100%	0.062
table3	14	14	175	10584	25%	100%	0.004	100%	0.032
misex3c	14	14	197	11856	13%	100%	0.003	100%	0.035
exp5	8	63	74	19454	10%	65%	0.006	80%	0.024
apex4	9	19	436	25480	21%	100%	0.008	100%	0.173
alu4	14	8	575	25652	19%	100%	0.008%	100%	0.284

I,O,P: Input, Output and Product. Implementation with dual is shown in bold.

parameter values stabilize nearly after this threshold value. All algorithms are implemented in MATLAB™. Standard benchmark circuits presented in [20] are used for mapping algorithms. All experiments run on a 3.30GHz Intel Core i7 CPU (only single core used) with 8GB memory.

To evaluate the proposed hybrid algorithm (**HBA**), we consider success rate and runtime values compared to those of the exact algorithm (**EA**). Contrary the our algorithm which applies assignment method only to output rows, the exact algorithm constructs the matching matrix for all minterms and output rows of FM and then applies the assignment method as shown inf Fig. 8(d). Table II shows the results. In terms of runtime, **HBA** is superior for all cases, at least one order of magnitude and at most two order of magnitude for circuits such as *apex4* and *alu4*. However, regarding the success rate (Psucc) showing the percentage of valid mapping found for 200 samples, there is up to 15% difference that is a small trade-off for our runtime gain.

VI. DISCUSSION

In this paper, we study logic synthesis and defect tolerance of memristor based crossbar arrays. We propose two-level and multi-level logic synthesis techniques. We show that for certain single output logic functions, multi-level synthesis reduces the area cost drastically. However, since conventional EDA tools are used for technology mapping satisfactory, results cannot be obtained for larger and multi-output benchmark functions. Field of memristor specific technology dependent mapping tools are open to further research.

In addition, we devise a defect model and propose a hybrid defect tolerant logic mapping method. We show that in spite of defective components, securing a valid mapping is achievable with an appropriate algorithm. Nevertheless, we only employ optimum size crossbars meaning minimum sized crossbar array to realize the given logic function during our mapping simulation. Because of that, exploring redundant crossbar areas might improve the defect tolerance performance especially regarding stuck-at closed type defects which prevents the usage of an entire horizontal and vertical lines. As an another future direction, we plan to integrate multi-level logic design with our defect tolerant logic mapping methods.

REFERENCES

[1] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa *et al.*, "A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications," *Nano letters*, vol. 12, no. 1, pp. 389–395, 2011.

[2] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, no. 1, pp. 13–24, 2013.

[3] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[4] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, p. 80, 2008.

[5] I. Vourkas and G. C. Sirakoulis, "Emerging memristor-based logic circuit design approaches: A review," *IEEE Circuits and Systems Magazine*, vol. 16, no. 3, pp. 15–30, 2016.

[6] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A: Materials Science & Processing*, vol. 80, no. 6, pp. 1165–1172, 2005.

[7] L. Xie, H. A. Du Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "Fast boolean logic mapped on memristor crossbar," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 335–342.

[8] L. Xie, H. A. Du Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, "A mapping methodology of boolean logic circuits on memristor crossbar," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 99, 2017.

[9] L. Xie, "Towards robust implementation of memristor crossbar logic circuits," in *Ph. D. Research in Microelectronics and Electronics (PRIME), 2016 12th Conference on*. IEEE, 2016, pp. 1–4.

[10] L. Xie, H. A. Du Nguyen, J. Yu, M. Taouil, and S. Hamdioui, "On the robustness of memristor based logic gates," in *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2017 IEEE 20th International Symposium on*. IEEE, 2017, pp. 158–163.

[11] S. Kannan, N. Karimi, R. Karri, and O. Sinanoglu, "Detection, diagnosis, and repair of faults in memristor-based memories," in *VLSI Test Symposium (VTS), 2014 IEEE 32nd*. IEEE, 2014, pp. 1–6.

[12] S. Hamdioui, M. Taouil, and N. Z. Haron, "Testing open defects in memristor-based memories," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 247–259, 2015.

[13] O. Tunali and M. Altun, "Permanent and transient fault tolerance for reconfigurable nano-crossbar arrays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 5, pp. 747–760, 2017.

[14] O. Tunali and M. Altun, "A survey of fault-tolerance algorithms for reconfigurable nano-crossbar arrays," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 79:1–79:35, Nov. 2017.

[15] M.-J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur *et al.*, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric ta2o5-x/tao2-x bilayer structures," *Nature materials*, vol. 10, no. 8, p. 625, 2011.

[16] Y. V. Pershin and M. Di Ventra, "Practical approach to programmable analog circuits with memristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 8, pp. 1857–1864, 2010.

[17] M. Traiola, M. Barbareschi, and A. Bosio, "Formal design space exploration for memristor-based crossbar architecture," in *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2017 IEEE 20th International Symposium on*. IEEE, 2017, pp. 145–150.

[18] M. Traiola, M. Barbareschi, A. Mazzeo, and A. Bosio, "Xbargen: A memristor based boolean logic synthesis tool," in *Very Large Scale Integration (VLSI-SoC), 2016 IFIP/IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[19] B. L. Ssynthesis, "Verification group," *ABC: A system for sequential synthesis and verification*, 2013.

[20] K. McElvain, "Iwls93 benchmark set: Version 4.0," in *Distributed as part of the MCNC International Workshop on Logic Synthesis*, vol. 93, 1993.

[21] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.