Fault-Tolerant In-Memory Crossbar Computing using Quantified Constraint Solving

Alvaro Velasquez
EECS Department
University of Central Florida, Orlando, FL
Email: velasquez@eecs.ucf.edu

Sumit Kumar Jha
EECS Department
University of Central Florida, Orlando, FL
Email: http://www.sumitkumarjha.com/contact.html

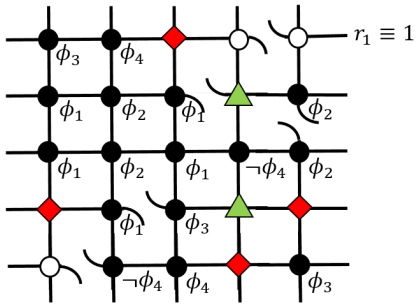
***Abstract*—There has been a surge of interest in the effective storage and computation of data using nanoscale crossbars. In this paper, we present a new method for automating the design of fault-tolerant crossbars that can effectively compute Boolean formula. Our approach leverages recent advances in Satisfiability Modulo Theories (SMT) solving for quantified bit-vector formula (QBVF). We demonstrate that our method is well-suited for fault-tolerant computation and can perform Boolean computations despite stuck-open and stuck-closed interconnect defects as well as wire faults. We employ our framework to generate various arithmetic and logical circuits that compute correctly despite the presence of stuck-at faults as well as broken wires.**

I. Introduction

The continued scaling of the CMOS device has been largely responsible for the increase in computational power and consequent technological progress over the last few decades. However, the end of Dennard scaling has interrupted this era of sustained exponential growth in computing performance. The lithographic processes required to fabricate ever-smaller transistors result in higher process variations and more unpredictable structures when working at the atomic scale [4]. Furthermore, the alignment of these devices poses an even greater challenge as we traverse deeper into the nanoscale [21].

In response to these difficulties, substantial research has been devoted to novel patterning methods that can extend traditional optical lithography. This research effort has produced methods such as extreme ultra-violet (EUV) lithography, nanoimprint lithography (NIL), maskless lithography (ML2), and bottom-up directed self-assembly (DSA). The last of these methods shows promise as a potential successor to top-down lithography [15]. DSA synthesizes relatively basic primitives with precise shapes that may be used for computation. However, it is not suitable for fabricating complex device topologies that frequently occur in modern circuits [4]. The use of these basic primitive structures necessitates a paradigm shift in our circuit model. Motivated by the success of assembling aligned nanowires [10] and the ease of reconfiguration, attention has shifted to the use of crossbars [20], which can be easily assembled using this approach [13].

This material is based upon work supported by the National Science Foundation under Grant No. CCF-1438989 and CCF-1422257. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.



$$r_5 \equiv (\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \neg\phi_4) \vee (\phi_1 \wedge \phi_2 \wedge \neg\phi_3 \wedge \phi_4) \vee (\phi_1 \wedge \neg\phi_2 \wedge \phi_3 \wedge \phi_4) \vee (\neg\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4) \vee (\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4)$$

Fig. 1: A nanoscale crossbar computing the 4-bit majority function in the presence of known defects. Rhombuses represent stuck-open devices and triangles indicate stuck-closed devices. The crossbar also includes broken nanowires. If a flow of current is injected into the topmost nanowire, it reaches the lowest nanowire if and only if the 4-bit majority function evaluates to true.

The use of crossbars has been explored extensively in memory architectures, with the recently revealed Intel 3D XPoint™ [9] memory as a noteworthy example. Such crossbar architectures open up the possibility of developing a computation-in-memory paradigm using crossbars, which is the main contribution of this paper. The unified in-memory computation architecture presented in this paper eliminates the much maligned memory-processor bottleneck [16] that plagues the von Neumann architecture.

However, the fabrication of nanoscale crossbars via bottom-up self-assembly suffers from high defect rates [4]. The faults in nanoscale crossbars can occur in the form of stuck-open and stuck-closed devices as well as broken wires. The high defect rates make it uneconomical to simply reject defective nanoscale crossbars [24]. Hence, there is a pressing need to develop fault-tolerant computing approaches [25], [35], [36] that can ensure the fruitful use of even defective nanoscale crossbars. To this end, we propose a new methodology for mapping logical and arithmetic operations onto a defective crossbar that exploits the occurrence of the faults instead of being hindered by it.

In this paper, we present a new framework based on auto-

mated synthesis [26] that maps Boolean logic computations onto a nano-crossbar. We leverage the power of Satisfiability Modulo Theory (SMT) solvers and exploit recent advances in quantified bit-vector formula solving capabilities to generate our computation-in-memory designs. We demonstrate that our approach is especially well-suited for the critical post-fabrication fault-tolerant logic mapping in the presence of various common defects. We validate our approach by synthesizing arithmetic and logical circuits using highly defective crossbars.

II. Crossbar Computing

A crossbar consists of two sets of mutually perpendicular wires. Each wire from one set is connected to every wire from the other set through an electronic component. This interconnection can be a two-terminal device such as a diode or memristor, or a three-terminal device such as the CMOS transistor. In the remainder of this section, we will briefly discuss a few popular crossbar computing architectures.

The NanoFabric [7] consists of an array of interconnected and reprogrammable nano-crossbars, called nanoBlocks, with corresponding switch blocks. The NanoFabric is similar to an FPGA *sans* the look-up tables. A number of NanoFabric designs have been proposed that include AND, OR, XOR, and a half-adder. The NanoPLA [5] uses reconfigurable diodes or rectifying diode-like elements in tandem with NOR-NOR logic in order to perform logical operations. The Nanoscale Application-Specific Integrated Circuit (NASIC) architecture [31] is an ASIC that utilizes field-effect transistors and crossbars. An extension of this framework that interfaces with a 3D CMOS stack has also been proposed [17].

Crossbar-computing frameworks using traditional CMOS-like logics have also been proposed. In one such methodology [18], crossbars of p- and n-type FETs and programmable switches are used to evaluate logical conjunctions, disjunctions, and inversions. A hybrid approach, referred to as CMOL [14], utilizes CMOS and molecular computing in the form of crossbars. The crossbars are fabricated on top of the CMOS die and the two architectures interface via a set of pins that connects the CMOS die to the top and bottom nanowires. Bottom-up self-assembly is perfectly suited for this by allowing the crossbars to grow regardless of substrate used, thus not requiring the high temperatures that restricts the fabrication of CMOS circuitry on top of another CMOS stack [21]. The crossbars could function as memory or as reconfigurable computing fabrics. A variation of CMOL [19] uses a field programmable nanowire interconnect (FPNI) to facilitate communication between the crossbar and the CMOS stack. Some additional constraints are enforced, such as confining logic to the CMOS stack and restricting routing to the crossbar. The resulting architecture trades off speed and fault-tolerance for ease of fabrication when compared to traditional CMOS. A 3D extension of CMOL was introduced in [28]. In this architecture, two CMOS stacks are used and the crossbar is sandwiched between the two stacks. This allows each stack to communicate with only one set of wires, thereby mitigating the pin interfacing difficulties of traditional CMOL.

Promising emerging technologies, such as the memristor [22] and spin-transfer-torque (STT) devices, have also been introduced to the crossbar-computing domain both as memory and as computation nodes. A 3-D crossbar architecture [6] uses memristors together with an interfacing methodology similar to field programmable nanowire interconnect (FPNI). Another architecture [33] uses Null Convention Logic (NCL) to develop an asynchronous lookup table using a memristor crossbar. Sneak paths [37] pose a problem in these architectures by causing OFF nodes to be read as ON nodes due to the flow of current through sequences of ON nodes that run in parallel to the current being used to measure the OFF node. This problem is analogous to the crosstalk problem in RRAM memories. Solutions have been proposed via the use of 1M1D memristor-diode structures or diode-like rectifying memristors [11] [12]. In earlier work, we have explored the application of formal methods to the problem of designing memristor crossbars that can exploit these sneak paths as a means for evaluating Boolean formula [30] instead of trying to suppress the occurrence of sneak paths. We have also shown how sneak paths can be used to perform digital computations using networks of nanoscale memristor crossbars [29].

One omission often made in the literature, including our earlier work, lies in addressing the robustness of crossbar computing designs in the presence of stuck-open and stuck-closed nodes as well as broken wires, which are common defects resulting from self-assembly [4]. Given a faulty $n \times n$ crossbar, one solution [27] determines the $k \times k$ ($k < n$) maximum defect-free subset of the crossbar which can then be used in the application-mapping stage of the fabrication process. However, the computational power of the resulting crossbar can be diminished due to loss of significantly many computation nodes. Another algorithm for finding defect-free subcrossbars from a defective crossbar is proposed in [34]. Other methods include simply avoiding the defective devices altogether by providing redundancy in the form of spare parts [3].

In light of the high defect rates associated with self-assembled crossbars, we propose a new approach to fault-tolerant crossbar computing that exploits the stuck-at interconnects and broken wires to design nanoscale crossbars for computing Boolean formula. As our method does not try to identify a sub-crossbar of defect-free interconnects and instead leverages defects during the design phase, it is capable of performing useful computations even on highly defective crossbars. See Figure 1 for an example. Our approach is independent of the underlying architecture and only requires the use of a crossbar consisting of digital switches. As such, this framework can be easily implemented using memristor crossbars, CMOL, and programmable logic arrays (PLAs), among others.

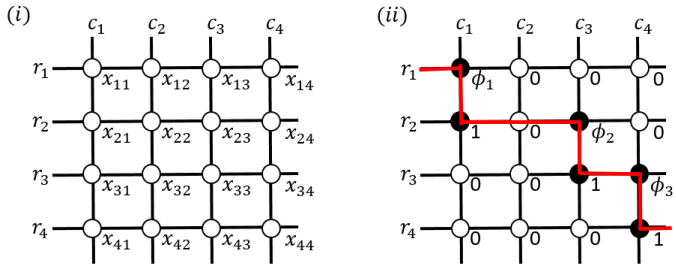


Fig. 2: (i) 4×4 crossbar X with rows r_i, columns c_j, and nodes x_{ij}. (ii) Mapping of the formula $\phi = (\phi_1 \wedge \phi_2 \wedge \phi_3)$ on to X, where r_1 is the source wire and r_4 is the destination.

III. Irreducible Paths in Crossbars

We begin by formalizing the rather intuitive notion of a crossbar. A crossbar can be represented as an ordered triple $\mathbb{C} = (X, R, C)$, where $X = (x_{ij}) \in \{0,1\}^{m \times n}$ denotes the state matrix of the interconnects. $x_{ij} = 0$ indicates that the interconnect located at the junction of row i and column j is open, and $x_{ij} = 1$ denotes a closed interconnect. R is the set of rows such that $r_i \in R$ denotes the i^{th} row. Similarly, C is the set of columns, or vertical wires, such that $c_j \in C$ denotes the j^{th} column. In this abstraction, the values of these wires are Boolean in order to represent the presence or absence of electrical current flow. A pictorial representation of this definition can be seen in Figure 2.

The basis of our approach lies in computing Boolean formula on a crossbar by mapping the variables of the formula to individual interconnects in the crossbar such that there will be a path between two wires if and only if the formula we wish to compute evaluates to $true$. In Figure 2, ϕ_1, ϕ_2, and ϕ_3 are mapped to x_{11}, x_{23}, and x_{34}, respectively, while x_{21}, x_{33}, and x_{44} are in the ON state. Note that this mapping creates the path $\pi^{r_1 \to r_4} = \{x_{11}, x_{21}, x_{23}, x_{33}, x_{34}, x_{44}\}$ which directs flow from wire r_1 to wire r_4 iff ϕ is $true$ given an initial flow of current on r_1.

Axiom 1 (Flow Axioms)**.** *Let $\mathbb{C} = (X, R, C)$ be an $m \times n$ crossbar. Then the following always hold:*

- $\forall i \leq m, j \leq n, (r_i \wedge x_{ij}) \implies c_j$.
- $\forall i \leq m, j \leq n, (c_j \wedge x_{ij}) \implies r_i$.

It is clear that if there is an electric potential difference between two wires r_α and r_β, then there must exist a sequence of interconnects that, when closed, would generate a current from r_α to r_β. We refer to this sequence of nodes as a path. Since there can be many such paths, we denote the ordered set $\pi_k^{r_\alpha \to r_\beta} = \{x_{\alpha j_1}, x_{i_1 j_1}, x_{i_1 j_2}, \ldots, x_{i_k j_l}, x_{\beta j_l}\}$ as the k^{th} path from r_α to r_β, where each $x \in \pi_k^{r_\alpha \to r_\beta}$ is a constituent node in the path. It is obvious that the first node resides on r_α and the last node must reside on r_β. For the sake of simplicity, we will assume that the source and destination of a path are both row wires.

Definition 1 (Path)**.** *A path $\pi^{r_\alpha \to r_\beta} = \{x_{\alpha j_1}, x_{i_1 j_1}, x_{i_1 j_2}, \ldots, x_{i_k j_l}, x_{\beta j_l}\}$ is an ordered set of nodes containing a source node $x_{\alpha i}$ and a destination $x_{\beta j}$.*

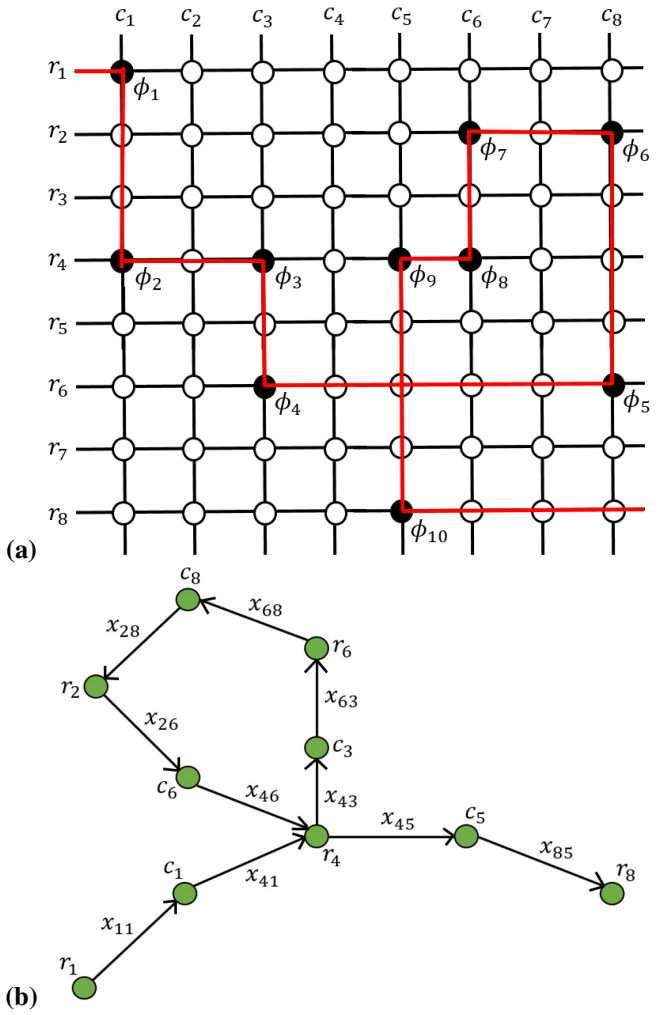


Fig. 3: (a) 8×8 crossbar mapping the formula $\phi = (\phi_1 \wedge \phi_2 \wedge \phi_9 \wedge \phi_{10})$. The red bars represent the path $\pi^{r_1 \to r_8} = \{x_{11}, x_{41}, x_{43}, x_{63}, x_{68}, x_{28}, x_{26}, x_{46}, x_{45}, x_{85}\}$ corresponding to $\bigwedge_{i=1}^{10} \phi_i$. (b) Equivalent representation of $\pi^{r_1 \to r_8}$ as a graph given by $\pi_G^{r_1 \to r_8} = \{(r_1, c_1), (c_1, r_4), (r_4, c_3), (c_3, r_6), (r_6, c_8), (c_8, r_2), (r_2, c_6), (c_6, r_4), (r_4, c_5), (c_5, r_8)\}$, where $(r_i, c_j) \in \pi_G^{r_1 \to r_8}$ denotes a directed edge from r_i to c_j.

Definition 2 (Irreducible Path)**.** *Let $\Pi^{r_\alpha \to r_\beta}$ denote the set of all paths from r_α to r_β in a crossbar. $\pi_j \in \Pi^{r_\alpha \to r_\beta}$ is an irreducible path if and only if $\forall \pi_i \in \Pi^{r_\alpha \to r_\beta}, \pi_i \not\subset \pi_j$.*

According to *Definition 2*, a path is irreducible if no other path with the same source and destination nodes is contained within it. See Figure 3 for an example. The path $\pi^{r_1 \to r_8} = \{x_{11}, x_{41}, x_{43}, x_{63}, x_{68}, x_{28}, x_{26}, x_{46}, x_{45}, x_{85}\}$ is not an irreducible path because the path $\pi' = \{x_{11}, x_{41}, x_{45}, x_{85}\}$ is contained within it. That is, $\pi' \subset \pi^{r_1 \to r_8}$, thereby violating the definition of an irreducible path.

Graphs can be used to represent sneak paths in

nanoscale crossbars. The vertices in this graph represent the row and column wires while the edges correspond to the interconnects connecting these vertices. The unweighted connected digraph $G = (V, E)$ corresponding to $\pi^{r_1 \to r_8}$ is shown in Figure 3b. We define $\pi_G^{r_1 \to r_8} = \{(r_1, c_1), (c_1, r_4), (r_4, c_3), (c_3, r_6), (r_6, c_8), (c_8, r_2), (r_2, c_6), (c_6, r_4), (r_4, c_5), (c_5, r_8)\}$ as a path in the graph. Note that $\pi_G^{r_1 \to r_8}$ and $\pi^{r_1 \to r_8}$ are equivalent representations of the same sneak path. We distinguish between the interconnect and edge representations $\pi = \{x_{ij}\}_{i,j}$ and $\pi_G = \{(r_i, c_j), (c_j, r_k)\}_{i,j,k}$ of an arbitrary path π by adding the subscript G. We observe that the graph formed by $\pi_G^{r_1 \to r_8}$ has a cycle consisting of the edges in $\pi_G^{r_1 \to r_8} \setminus \pi'_G$. Furthermore, the irreducible path $\pi'_G = \{(r_1, c_1), (c_1, r_4), (r_4, c_5), (c_5, r_8)\}$ contains no cycles. That is, π'_G is a simple walk.

Theorem 1. *A path $\pi^{r_\alpha \to r_\beta}$ in an $m \times n$ crossbar $\mathbb{C} = \{X, R, C\}$ is reducible if and only if $\pi_G^{r_\alpha \to r_\beta}$ contains a cycle.*

Proof. ($\implies$) We argue the contrapositive. Let $\pi_G^{r_\alpha \to r_\beta} = \{(r_\alpha, c_{j_1}), (c_{j_1}, r_{i_1}), \ldots, (r_{i_k}, c_{j_l}), (c_{j_l}, r_\beta)\}$. Since there are no cycles, $\pi_G^{r_\alpha \to r_\beta}$ is a minimally connected graph i.e. the out- and in-degrees of all nodes are at most 1. Thus, by definition of minimally connected graphs, there is only one path between any two nodes. It follows that there is only one path from r_α to r_β in the graph formed by $\pi_G^{r_\alpha \to r_\beta}$. Therefore, the path is irreducible.

($\impliedby$) Suppose, without loss of generality, that $\pi = \{x_{i_1 j_1}, x_{i_2 j_1}, x_{i_2 j_2}, \ldots, x_{i\alpha_1}, x_{i\alpha_2}, \ldots, x_{i\alpha_3}, x_{i\alpha_4}, \ldots, x_{i\alpha_{2k-1}}, x_{i\alpha_{2k}}, \ldots\}$ is an irreducible path containing a cycle(s). We can construct $\pi' = \pi \setminus \{x_{i\alpha_2}, \ldots, x_{i\alpha_{2k-1}}\} \subset \pi$. Thus, since $\pi' \subset \pi$, π violates the irreducible paths property, a contradiction. Thus, paths that contain cycles are reducible. □

A reducible path will carry a flow of current from its source to its destination whenever the irreducible path contained in it can carry a flow of current from the source to the destination. Thus, the presence of reducible paths in crossbar designs does not permit the crossbar to compute new Boolean formula.

Corollary 1. *A crossbar designed only using irreducible paths and a crossbar designed using all paths can evaluate the same set of Boolean formula.*

Proof. It follows from *Theorem 1* that reducible paths have cycles and irreducible paths do not. Suppose we are given an n-ary Boolean formula $\phi(\phi_1, \ldots, \phi_n)$ and paths $\pi^{r_\alpha \to r_\beta}$ and $\pi'^{r_\alpha \to r_\beta}$, such that $\pi' \subset \pi$. In the context of our framework, we say that ϕ evaluates to true if and only if there is a path π from r_α to r_β or, equivalently, if π_G contains a connected walk from r_α to r_β. Since the inclusion or exclusion of edges in a cycle does not affect the connectedness of the walk in the graph, it follows that $\pi^{r_\alpha \to r_\beta}$ is a connected walk if $\pi'^{r_\alpha \to r_\beta}$ is a connected walk from r_α to r_β. □

Since reducible paths do not add to the expressive power of nanoscale crossbars, our synthesis technique relies on searching through the space of irreducible paths only. This allows us to prune the search space of our approach substantially by ignoring the redundant reducible paths.

Theorem 2. *A path $\pi^{r_\alpha \to r_\beta}$ forms a cycle if and only if there are more than two nodes in the path that reside on the same wire in the crossbar.*

Proof. ($\implies$) Assume that $\pi_G^{r_\alpha \to r_\beta} = \{(r_i, c_j), (c_j, r_k)\}_{i,j,k}$ contains a cycle. It follows from *Definition 1* that (r_α, c_{i_1}) is the only outgoing edge from r_α and (c_{i_k}, r_β) is the only incoming edge to r_β. Since $\pi_G^{r_\alpha \to r_\beta}$ forms a cyclic connected digraph, and r_α and r_β cannot be nodes in a cycle, there must exist some node r_i that belongs to a cycle as well as to a simple walk from r_α to r_β. It follows that there exist edges (c_{j_1}, r_i) and (r_i, c_{j_2}) connecting r_i to the simple walk from r_α to r_β and edges (c_{j_3}, r_i) and (r_i, c_{j_4}) connecting r_i to the cycle. See node r_4 in Figure 3b for an example. These edges correspond to interconnects x_{ij_1}, x_{ij_2}, x_{ij_3}, and x_{ij_4} in $\pi^{r_\alpha \to r_\beta}$, leading to more than two nodes on the same wire.

($\impliedby$) Without loss of generality, assume that $\pi^{r_\alpha \to r_\beta} = \{x_{\alpha j_1}, x_{i_1 j_1}, x_{i_2 j_1}, x_{i_2 j_2}, \ldots, x_{i' p_1}, x_{i' p_2}, \ldots, x_{i' p_3}, x_{i' p_4}, \ldots, x_{i' p_{2k-1}}, x_{i' p_{2k}}, \ldots, x_{\beta c_l}\}$ has more than two nodes on $r_{i'}$. Note that for all $r_i \neq r_{i'}$, the out- and in-degrees are 1, while the out- and in-degrees of $r_{i'}$ are greater than or equal to 2. Since r_α is the only node with no incoming edges, r_β is the only node with no outgoing edges, and all paths form connected digraphs, it follows that node $r_{i'}$ must be in a cycle. □

Corollary 2. *A path π is irreducible if and only if there are no more than two nodes on any wire.*

We now have the foundations necessary to enumerate irreducible paths in a crossbar. The length and number of such paths is discussed below.

Lemma 1. *The maximum length $|\pi^{r_\alpha \to r_\beta}|_{max}$ of an irreducible path $\pi^{r_\alpha \to r_\beta} = \{x_{\alpha j_1}, x_{i_1 j_1}, x_{i_1 j_2}, \ldots, x_{i_k j_l}, x_{\beta j_l}\}$ in an $m \times n$ crossbar $\mathbb{C} = (X, R, C)$ is $2(min\{m-1, n\})$.*

Proof. It follows from *Definition 1* that $x_{\alpha j_1}$ and $x_{\beta j_l}$ are in the path and are the only nodes on r_α and r_β, respectively. Let $R' = R \setminus \{r_\alpha, r_\beta\}$ denote the set of remaining rows where nodes of $\pi^{r_\alpha \to r_\beta}$ reside. Note that $|R'| = m - 2$. It follows from *Corollary 1* that there can only be up to two nodes per wire. Two cases arise:

- *Case 1* ($n \geq m - 1$): Each row in R' can have two nodes. Thus, $|\pi^{r_\alpha \to r_\beta}|_{\max} = 2|R'| + 2 = 2(m-2) + 2 = 2(m-1)$.
- *Case 2* ($n < m-1$): Assigning two nodes to each column yields $|\pi^{r_\alpha \to r_\beta}|_{\max} = 2n$ and it follows from *case 1* that assigning two nodes to each row yields $|\pi^{r_\alpha \to r_\beta}|_{\max} = 2(m-1)$. However, since $n < m - 1$, it is not possible to have an irreducible path of length $2(m-1)$ as this would require some column to contain more than two nodes, thereby violating the irreducible paths property. Thus, $|\pi^{r_\alpha \to r_\beta}|_{\max} = 2n$. □

Theorem 3. *There are* $\frac{n!(m-2)!}{(n-k/2)!(m-1-k/2)!}$ *irreducible paths of length* k *from* r_α *to* r_β *in an* $m \times n$ *crossbar.*

Proof. Let $f^{(k)}$ denote the number of k-length paths and Π^k denote the set of said paths. For $k = 2$, $\Pi^k = \{(x_{\alpha j}, x_{\beta j})\}_{j=1}^{n}$ for source r_α and destination r_β. Thus, $f^{(2)} = n$. For $k = 4, 6, \ldots, |\pi^{r_\alpha \to r_\beta}|_{\max}$, there are $k-1$ possibilities for choosing row and column coordinates of the nodes in the path. Since our initial and final choices of row coordinate are fixed to the source and destination wires r_α and r_β, there will be one more choice of column coordinate than row coordinate. There are $\left\{ {m-2 \atop k/2-1} \right\}$ ways to choose row coordinates and $\left\{ {n \atop k/2} \right\}$ ways to choose column coordinates, where $\left\{ {n \atop k} \right\}$ denotes the permutation of k elements from an n-element set. Thus, we have $f^{(k)} = \left\{ {n \atop k/2} \right\} \left\{ {m-2 \atop k/2-1} \right\} = \frac{n!(m-2)!}{(n-k/2)!(m-1-k/2)!}$.

Base case: Paths of length 2.

$$f^{(2)} = \frac{n!(m-2)!}{(n-1)!(m-2)!} = n$$

Inductive hypothesis: Paths of length k.

$$f^{(k)} = \frac{n!(m-2)!}{(n-k/2)!(m-1-k/2)!}$$

Inductive step: Note that, given the value of $f^{(k)}$ for any k, there will be $(m-2)-(k/2-1) = m-1-k/2$ possible row choices and $n-k/2$ possible column choices for a $(k+2)$-length path. Therefore,

$$\begin{aligned} f^{(k+2)} &= f^{(k)}(n-k/2)(m-1-k/2) \\ &= \frac{n!(m-2)!}{(n-k/2)!(m-1-k/2)!}(n-k/2)(m-1-k/2) \\ &= \frac{n!(m-2)!}{(n-k/2-1)!(m-k/2-2)!} \\ &= \frac{n!(m-2)!}{(n-\frac{k+2}{2})!(m-1-\frac{k+2}{2})!} \end{aligned}$$

Hence, proved by mathematical induction. □

Corollary 3. *The number of irreducible paths from* r_α *to* r_β *in an* $m \times n$ *crossbar is* $T(m,n) = \sum_{k=1}^{min\{m-1,n\}} \left\{ {n \atop k} \right\} \left\{ {m-2 \atop k-1} \right\}$, *where* $\left\{ {n \atop k} \right\}$ *denotes the permutation of* k *elements from an* n*-element set.*

Proof. Recall that $|\pi^{r_\alpha \to r_\beta}|_{\max}$ denotes the maximum length of a path from r_α to r_β. It follows directly from *Theorem 3* and *Lemma 1* that the total number of irreducible paths is

$$\begin{aligned} \sum_{k=2,\ k \text{ even}}^{|\pi^{r_\alpha \to r_\beta}|_{\max}} f^{(k)} &= \sum_{k=2,\ k \text{ even}}^{|\pi^{r_\alpha \to r_\beta}|_{\max}} \frac{n!(m-2)!}{(n-k/2)!(m-1-k/2)!} \\ &= \sum_{k=1}^{|\pi^{r_\alpha \to r_\beta}|_{\max}/2} \frac{n!(m-2)!}{(n-k)!(m-1-k)!}. \\ &= \sum_{k=1}^{\min\{m-1,n\}} \left\{ {n \atop k} \right\} \left\{ {m-2 \atop k-1} \right\} = T(m,n) \end{aligned}$$

□

The number of paths established in the preceding corollary grows rapidly in tandem with the size of the crossbar. While this enormous number of potential paths of current through the crossbar presents a great opportunity for the design of compact computing crossbars, it also poses a considerable obstacle to the invention of spatially-efficient designs via human reasoning. Hence, we suggest the use of automated synthesis techniques for designing nanoscale crossbars for computing Boolean formula.

IV. Automated Synthesis of Crossbars

Let $\Pi^{r_\alpha \to r_\beta}$ denote the set of all irreducible paths between r_α and r_β. Recall from *Corollary 1* that reducible paths do not add to the expressive power of nanoscale crossbars. Thus, we enforce the irreducible paths property in order to significantly reduce the search space of the synthesis procedure. Let π_{γ_i} denote the i^{th} node in path π_γ. A path between r_α and r_β exists if and only if there is a satisfying assignment to following formula:

$$\bigvee_{\pi_\gamma \in \Pi^{r_\alpha \to r_\beta}} \left(\bigwedge_{\pi_{\gamma_i} \in \pi_\gamma} \pi_{\gamma_i} \right) \tag{1}$$

From Theorem 1, we can make the bounds on the Boolean formula (1) explicit as follows:

$$\bigvee_{\gamma=1}^{T(m,n)} \left(\bigwedge_{i=1}^{|\pi_\gamma|} \pi_{\gamma_i} \right) \tag{2}$$

We define a relationship between interconnects in the crossbar and the variables in the formula ϕ being computed by the crossbar through an auxiliary mapping matrix $P = (p_{ij})$, such that $p_{ij} \in \{\text{OFF}, \text{ON}, 1, 2, \ldots, |\phi|, |\phi|+1, |\phi|+2, \ldots, 2|\phi|\}$, where $|\phi|$ denotes the arity of ϕ and p_{ij} denotes the value being mapped to the node connecting row i and column j. The set of possible values for p_{ij} is rather intuitive. OFF and ON values denote that the interconnect is invariantly OFF or ON while an integer value $k \leq |\phi|$ indicates that the interconnect will be ON if and only if the k^{th} Boolean variable in ϕ is $true$. Conversely, if $k > |\phi|$, then the device will be closed if and only if the $(k-|\phi|)^{th}$ Boolean variable is $false$. Thus, given a mapping matrix $P = (p_{ij})$ and a Boolean formula ϕ, we define the following mapping function to an $m \times n$ crossbar $X = (x_{ij})$, where π_γ denotes the γ^{th} path from the source wire r_α to the destination wire r_β.

$$\begin{aligned} f(x_{ij} \in \pi_\alpha) \triangleq\ & (p_{ij} \equiv \text{ON}) \vee \bigvee_{k=1}^{|\phi|} \Big((p_{ij} \equiv k) \wedge \phi_k \Big) \vee \\ & \bigwedge_{k=1}^{|\phi|} \Big((p_{ij} \equiv k + |\phi|) \wedge \neg \phi_k \Big) \end{aligned} \tag{3}$$

Here, $A \equiv B$ is *true* if and only if A and B have the same value. The Boolean formula (3) is *true* if the node corresponding to its argument is always closed or if the

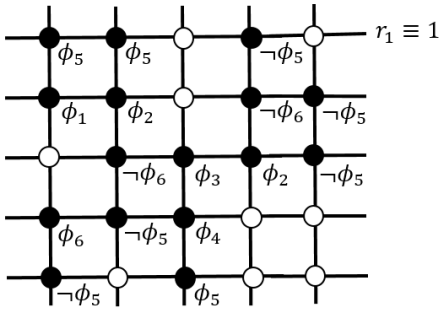


Fig. 4: A crossbar implementing the 4-to-1 multiplexer function $f_{mux} : \{0,1\}^6 \mapsto \{0,1\}$. Here, ϕ_5 and ϕ_6 are the control bits. The result is obtained at r_5 i.e. the lowest row.

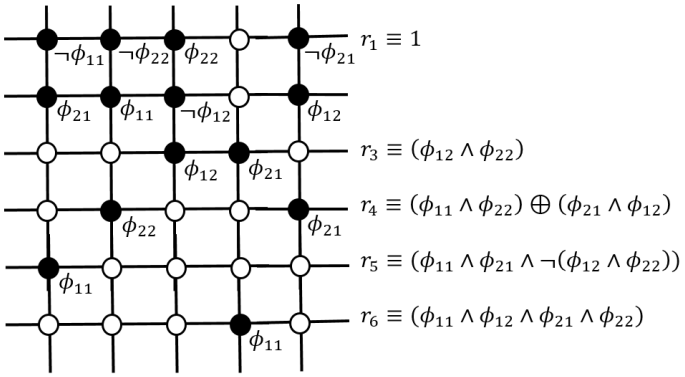


Fig. 5: A crossbar implementing the 2-bit multiplier. There are four outputs: the least significant bit, the second bit, the most significant bit, and the carry-out bit. They are obtained from rows 3, 4, 5, and 6 respectively.

(negated) variable that was mapped to the node is $true$. We can now combine the Boolean formula (2) and (3) to formulate the fundamental constraint used to synthesize crossbars using our approach.

$$\begin{aligned} \psi \triangleq\ & \forall \phi_1, \phi_2, \ldots, \phi_{|\phi|} \in \phi, \\ \phi \iff & \bigvee_{\gamma=1}^{T(m,n)} \left(\bigwedge_{i=1}^{|\pi_\gamma^{r_\alpha \to r_\beta}|} f(\pi_{\gamma_i}^{r_\alpha \to r_\beta}) \right) \end{aligned} \tag{4}$$

The quantified bit-vector formula (QBVF) (4) states that for all values of $\phi_1, \phi_2, \ldots, \phi_{|\phi|}$, ϕ will evaluate to *true* if and only if there is at least one irreducible path from r_α to r_β. Such a path would cause a flow of current to travel from r_α to r_β if and only if ϕ is *true*, given that $r_\alpha = 1$ i.e. there is an initial flow of current on the source node. Recall that $\pi_{\gamma_i}^{r_\alpha \to r_\beta}$ denotes the i^{th} node in path π_γ from r_α to r_β.

We observe that (4) is a quantified bit-vector formula (QBVF). An alternative representation of this formula would be to iterate through the $2^{|\phi|}$ possible Boolean values explicitly using a bounded conjunction. This approach, however, would disregard advances in efficient QBVF solving that are implemented in modern Satisfiability Modulo Theories (SMT) solvers. QBVFs have been handled in SMT solvers through various methods, including E-matching [2], formula expansion routines [1], and word-level simplifications [32].

Examples of multi-output nanoscale crossbars synthesized using this approach are shown in Figures 4 and 5. A 4-to-1 multiplexer is presented in Figure 4 while Figure 5 shows a 2-bit multiplier with four different outputs computed on the same crossbar.

V. Fault Tolerance

When compared to lithographic CMOS device manufacturing, self-assembled nanodevices have high defect rates in the form of stuck-open and stuck-closed devices, as well as broken or partitioned nanowires [8]. This necessitates fault-tolerant mapping schemes at the post-manufacturing stage [24] since rejecting faulty chips becomes unacceptable due to the high probability of defects per chip [23].

Our synthesis procedure is particularly well-suited for mitigating these types of faults at the post-fabrication stage of the design process. We will demonstrate how our approach can still provide valid crossbar configurations in the presence of highly defective devices. We account for the following faults:

- Stuck-open: A node x_{ij} has a stuck-open fault if $x_{ij} = 0$ and the node cannot be programmed to another state. That is, the interconnect is permanently open.
- Stuck-closed: A node x_{ij} has a stuck-closed fault if $x_{ij} = 1$ and the node cannot be programmed to another state. That is, the interconnect is permanently closed.
- Broken wire: This fault corresponds to wires that are segmented into multiple pieces. The affected write r_i is represented by its constituent segments, where r_i^k denotes the k^{th} segment of row i.

We add the stuck-open and stuck-closed faults as constraints to the decision procedure by instantiating an ancillary matrix $S = (s_{ij}) \in \{0, 1, *\}^{m \times n}$, where $s_{ij} = 0$ and $s_{ij} = 1$ denote a stuck-open and a stuck-closed fault, respectively, on the device located at the junction of row i and column j.

$$\begin{aligned} \psi_{\text{stuck}} \triangleq \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{n} \big(& (s_{ij} \equiv 1 \implies p_{ij} \equiv \text{ON}) \ \wedge \\ & (s_{ij} \equiv 0 \implies p_{ij} \equiv \text{OFF}) \big) \end{aligned} \tag{5}$$

A broken, or segmented, wire may still be used for computation by utilizing its remaining segments. Let $r_i^{\{k_1,k_2,\ldots,k_p\}}, k_t \in \{1, \ldots, n-1\}$ denote a break in the i^{th} row wire between columns k_t and $k_t + 1$, $t = 1, \ldots, p$. Similarly, $c_j^{\{k_1,k_2,\ldots,k_p\}}, k_i \in \{1, \ldots, m-1\}$ denotes a break in the j^{th} column wire between rows k_t and $k_t + 1$, $t = 1, \ldots, p$. We treat a broken wire r_i of p breaks as $p+1$ separate wires r_i^1, r_i^2, $\ldots$, r_i^{p+1}.

Corollary 1 holds for each wire segment when enumerating the paths in a faulty crossbar. As seen in Figure 6,

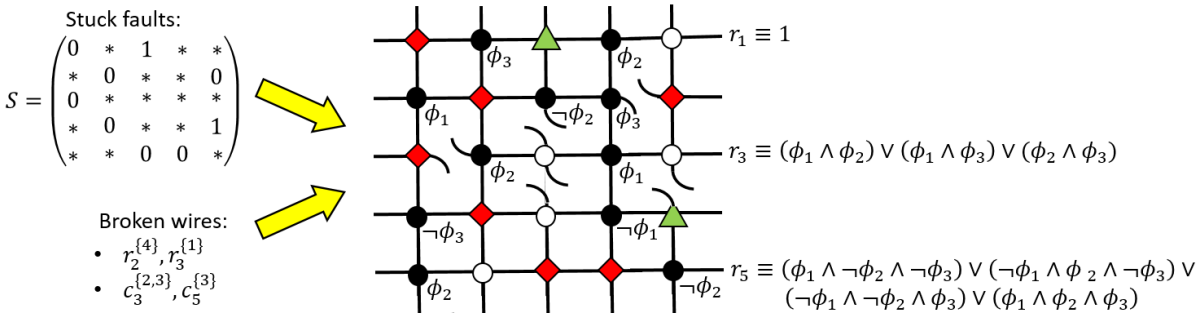


Fig. 7: Synthesized full adder crossbar in the presence of stuck-open/stuck-closed faults and broken wires, where the carry-out bit is produced as an output at row 3 and the sum bit is produced at row 5. Rhombuses and triangles represent stuck-open and stuck-closed devices, respectively. $r_i^{\{k_1,k_2,\ldots,k_p\}}, k_t \in \{1,\ldots,n-1\}$ denotes a break in the i^{th} row wire between columns k_t and k_t+1, $t=1,\ldots,p$. Similarly, $c_j^{\{k_1,k_2,\ldots,k_p\}}, k_i \in \{1,\ldots,m-1\}$ denotes a break in the j^{th} column wire between rows k_t and k_t+1, $t=1,\ldots,p$.

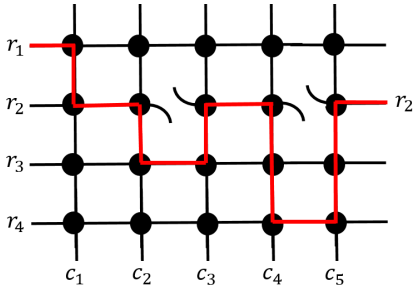


Fig. 6: $\pi^{r_1 \to r_2}$ demonstrated by the red path is an irreducible path in this faulty crossbar.

$\pi^{r_1 \to r_2} = \{x_{11}, x_{21}, x_{22}, x_{32}, x_{33}, x_{23}, x_{24}, x_{44}, x_{45}, x_{25}\}$ is an irreducible path in the crossbar. Yet, there are 5 nodes on row 2, which would seem to violate the irreducible paths property. However, looking at the graph representation $\pi_G^{r_1 \to r_2} = \{(r_1, c_1), (c_1, r_2^1), (r_2^1, c_2), (c_2, r_3), (r_3, c_3), (c_3, r_2^2), (r_2^2, c_4), (c_4, r_4), (r_4, c_5), (c_5, r_2^3)\}$, it can be seen that there are in fact no more than two nodes on any wire in $\pi_G^{r_1 \to r_2}$ when segments of a wire are considered as individual wires themselves.

The results of our fault-tolerant crossbar synthesis are shown in Figures 1 and 7. Figure 1 computes the 4-bit majority function on a crossbar with 4 stuck-open interconnects, 2 stuck-closed interconnects, and 5 breaks in nanowires. Hence, our algorithm is able to synthesize a nanoscale crossbar where as many as 24% of the crossbar interconnects have stuck-at faults. Figure 7 implements a full adder on a crossbar where 36% of the interconnects are stuck-open or stuck-closed.

VI. Conclusion and Future Work

We have presented a new framework for the synthesis of multi-output nano-crossbar computation-in-memory designs that can implement the evaluation of Boolean formula. We have demonstrated the effectiveness of our approach by generating a 2-bit multiplier with four outputs. We have also implemented a full adder on a crossbar with 36% stuck-at faults and a 4-bit majority function on a crossbar with 24% stuck-at faults.

We plan to extend our approach to the implementation of data-intensive programs on nanoscale crossbars. We will also investigate the relationship between different Boolean formula and the fraction of faulty interconnections in a crossbar such that the Boolean formula can be implemented on the crossbar despite these stuck-at faults. We will study experimental data on faults in nanoscale crossbars and design fault-tolerant synthesis algorithms tailored to the experimental distribution of faults in nanoscale crossbars.

References

[1] Armin Biere. Resolve and expand. In *Theory and Applications of Satisfiability Testing*, pages 59–70. Springer, 2005.

[2] Leonardo De Moura and Nikolaj Bjørner. Efficient e-matching for smt solvers. In *Automated Deduction–CADE-21*, pages 183–198. Springer, 2007.

[3] André DeHon. Array-based architecture for fet-based, nanoscale electronics. *Nanotechnology, IEEE Transactions on*, 2(1):23–32, 2003.

[4] Andre DeHon and Benjamin Gojman. Crystals and snowflakes: building computation from nanowire crossbars. *Computer*, 44(2):0037–45, 2011.

[5] Andre DeHon and Michael J Wilson. Nanowire-based sublithographic programmable logic arrays. In *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 123–132. ACM, 2004.

[6] Wei Fei, Hao Yu, Wei Zhang, and Kiat Seng Yeo. Design exploration of hybrid cmos and memristor circuit by new modified nodal analysis. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(6):1012–1025, 2012.

[7] Seth Copen Goldstein and Mihai Budiu. Nanofabrics: Spatial computing using molecular electronics. *ACM SIGARCH Computer Architecture News*, 29(2):178–191, 2001.

[8] Sezer Gören, H Fatih Ugurdag, and Okan Palaz. Defect-aware nanocrossbar logic mapping through matrix canonization using two-dimensional radix sort. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 7(3):12, 2011.

[9] Intel PR. Intel and micron produce breakthrough memory technology. `http://newsroom.intel.com/community/intel_newsroom/blog/2015/07/28/intel-and-micron-produce-breakthrough-memory-technology`. *Accessed* : 28 *July*, 2015.

[10] Seong Jun Kang, Coskun Kocabas, Taner Ozel, Moonsub Shim, Ninad Pimparkar, Muhammad A Alam, Slava V Rotkin, and John A Rogers. High-performance electronics using dense, perfectly aligned arrays of single-walled carbon nanotubes. *Nature nanotechnology*, 2(4):230–236, 2007.

[11] Kuk-Hwan Kim, Sung Hyun Jo, Siddharth Gaba, and Wei Lu. Nanoscale resistive memory with intrinsic diode characteristics and long endurance. *Applied Physics Letters*, 96(5):053106, 2010.

[12] Eero Lehtonen, Jussi H Poikonen, and Mika Laiho. Applications and limitations of memristive implication logic. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on*, pages 1–6. IEEE, 2012.

[13] Bao Liu. Advancements on crossbar-based nanoscale reconfigurable computing platforms. In *Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on*, pages 17–20. IEEE, 2010.

[14] Xialong Ma, Dmitri B Strukov, Jung Hoon Lee, and Konstantin K Likharev. Afterlife for silicon: Cmol circuit architectures. In *Nanotechnology, 2005. 5th IEEE Conference on*, pages 175–178. IEEE, 2005.

[15] Mark Neisser and Stefan Wurm. Itrs lithography roadmap: 2015 challenges. *Advanced Optical Technologies*, 4(4):235–240, 2015.

[16] Andreas Nowatzyk, Fong Pong, and Ashley Saulsbury. Missing the memory wall: The case for processor/memory integration. In *Computer Architecture, 1996 23rd Annual International Symposium on*, pages 90–90. IEEE, 1996.

[17] Pavan Panchapakeshan, Pritish Narayanan, and Csaba Andras Moritz. N3asics: designing nanofabrics with fine-grained cmos integration. In *Nanoscale Architectures (NANOARCH), 2011 IEEE/ACM International Symposium on*, pages 196–202. IEEE, 2011.

[18] Greg Snider, Philip Kuekes, and R Stanley Williams. Cmos-like logic in defective, nanoscale crossbars. *Nanotechnology*, 15(8):881, 2004.

[19] Gregory S Snider and R Stanley Williams. Nano/cmos architectures using a field-programmable nanowire interconnect. *Nanotechnology*, 18(3):035204, 2007.

[20] Mircea R Stan, Paul D Franzon, Seth Copen Goldstein, John C Lach, and Matthew M Ziegler. Molecular electronics: From devices and interconnect to circuits and architecture. *Proceedings of the IEEE*, 91(11):1940–1957, 2003.

[21] Dmitri B Strukov and Konstantin K Likharev. Reconfigurable nanocrossbar architectures. *Nanoelectronics and Information Technology*, pages 543–562, 2012.

[22] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *Nature*, 453(7191):80–83, 2008.

[23] Yehua Su and Wenjing Rao. Defect-tolerant logic mapping on nanoscale crossbar architectures and yield analysis. In *Defect and Fault Tolerance in VLSI Systems, 2009. DFT'09. 24th IEEE International Symposium on*, pages 322–330. IEEE, 2009.

[24] Yehua Su and Wenjing Rao. Defect-tolerant logic hardening for crossbar-based nanosystems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 1801–1806. IEEE, 2013.

[25] Yehua Su and Wenjing Rao. An integrated framework toward defect-tolerant logic implementation onto nanocrossbars. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 33(1):64–75, 2014.

[26] Susmit Jha. *Towards Automated System Synthesis Using SCIDUCTION*. PhD thesis, University of California at Berkeley, 2011.

[27] Mehdi B Tahoori. Application-independent defect tolerance of reconfigurable nanoarchitectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 2(3):197–218, 2006.

[28] D Tu, M Liu, Wei Wang, and S Haruehanroengra. Three-dimensional cmol: Three-dimensional integration of cmos/nanomaterial hybrid digital circuits. *Micro & Nano Letters, IET*, 2(2):40–45, 2007.

[29] Alvaro Velasquez and Sumit Kumar Jha. Parallel computing using memristive crossbar networks: Nullifying the processor-memory bottleneck. In *Design & Test Symposium (IDT), 2014 9th International*, pages 147–152. IEEE, 2014.

[30] Alvaro Velasquez and Sumit Kumar Jha. Automated synthesis of crossbars for nanoscale computing using formal methods. In *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, pages 130–136. IEEE, 2015.

[31] Teng Wang, Zhenghua Qi, and Csaba Andras Moritz. Opportunities and challenges in application-tuned circuits and architectures based on nanodevices. In *Proceedings of the 1st conference on Computing frontiers*, pages 503–511. ACM, 2004.

[32] Christoph M Wintersteiger, Youssef Hamadi, and Leonardo De Moura. Efficiently solving quantified bit-vector formulas. *Formal Methods in System Design*, 42(1):3–23, 2013.

[33] Jun Wu and Minsu Choi. Memristor lookup table (mlut)-based asynchronous nanowire crossbar architecture. In *Nanotechnology (IEEE-NANO), 2010 10th IEEE Conference on*, pages 1100–1103. IEEE, 2010.

[34] Bo Yuan and Bin Li. A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 10(3):25, 2014.

[35] Bo Yuan, Xin Yao, Bin Li, and W Thomas. A new memetic algorithm with fitness approximation for the defect-tolerant logic mapping in crossbar-based nano-architectures. 2014.

[36] Masoud Zamani and Mehdi Baradaran Tahoori. Variation-aware logic mapping for crossbar nano-architectures. In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, pages 317–322. IEEE, 2011.

[37] Mohammed Affan Zidan, Hossam Aly Hassan Fahmy, Muhammad Mustafa Hussain, and Khaled Nabil Salama. Memristor-based memory: the sneak paths problem and solutions. *Microelectronics Journal*, 44(2):176–183, 2013.