Information-Theoretic Sneak-Path Mitigation in Memristor Crossbar Arrays

Yuval Cassuto, *Senior Member, IEEE*, Shahar Kvatinsky, *Member, IEEE*, and Eitan Yaakobi, *Member, IEEE*

***Abstract*— In a memristor crossbar array, functioning as a memory array, a memristor is positioned on each row-column intersection, and its resistance, low or high, represents two logical states. The state of every memristor can be sensed by the current flowing through the memristor. In this work, we study the sneak path problem in crossbar arrays, in which current can sneak through other cells, resulting in reading a wrong state of the memristor. Our main contributions are modeling the error channel induced by sneak paths, a new characterization of arrays free of sneak paths, and efficient methods to read the array cells while avoiding sneak paths. To each read method we match a constraint on the array content that guarantees sneak-path free readout, determine the resulting capacity, and provide an efficient encoder that achieves the capacity.**

***Index Terms*— Codes for memories, sneak paths, constraint codes, memristors, resistive memories, crossbar arrays, Z channel.**

I. Introduction

The memristor technology [15] allows packing storage cells in an unprecedented density, over a simple crossbar structure. The blessing of high storage density and architectural simplicity comes with a major caveat: *data-dependent behavior* [13]. The read accuracy, speed, and power consumption in memristor storage may all vary significantly depending on the instantaneous data stored in the crossbar array. This is clearly an undesired property for a storage medium, and a motivation for data representations that ensure that the physical content of the array corresponds to a well-behaving device. Memristor storage has already motivated a novel data representation for one instantiation of the data-dependence problem [11]. Here we address another very significant data-dependent phenomenon called *sneak paths* [13], causing the read correctness to depend on the array content. The importance of the sneak-path problem can be sensed by the significant body of research addressing it recently in the device and circuit literature [5]–[7], [9], [12], [13], [18], [19].

To understand the sneak-path problem in memristor arrays, we first show a simplified schematic of a memristor array in Fig. 1(a). Each row-column pair is connected by

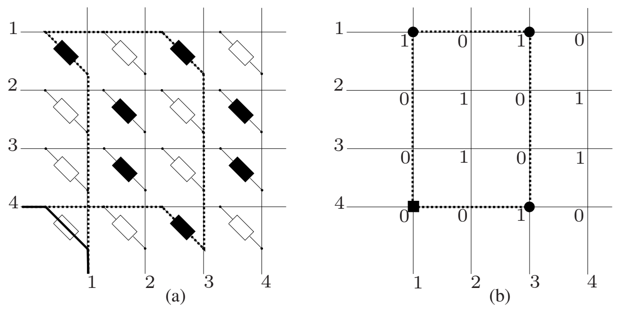

Fig. 1. (a) A memristor array as an array of programmed resistors – white: high resistance, black: low resistance. The high-resistance cell at location $(4,1)$ has a sneak-path in parallel (plotted dashed), causing it to be read as low-resistance. (b) The corresponding logical values of the memristor array. The cell in the square frame has a sneak-path comprising of the three cells marked in circles.

a resistor that can be in either the high-resistance state (marked white) or the low-resistance state (marked black). In Fig. 1(b) appear the corresponding logical values of the cells: logical ”0” for the high-resistance state, and logical ”1” for the low-resistance state. The sneak-path problem occurs when a resistor in the high-resistance state (white) is being read, while a series of resistors in the low-resistance state (black) exists in parallel to it, thereby causing it to be erroneously read as low-resistance. It is shown by the dashed line in Fig. 1(a) that the white resistor in (row, column) location $(4,1)$ has a sneak path that traverses the black resistors in locations $(4,3)$, $(1,3)$ and $(1,1)$. This dashed path is in parallel to the main current path of $(4,1)$ marked by a solid line.

In this paper we seek to combat memristor-array sneak paths using information-theoretic techniques. We first note that such an attempt was already presented in [17] by forcing the number of zeros and ones in every row and column to be the same. While this gives an elegant intuitive solution that can reduce the sneak path effect, our objective in this paper is to give a systematic information-theoretic study of the sneak-path problem

Yuval Cassuto is with the Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa 32000, Israel (email: ycassuto@ee.technion.ac.il).

Shahar Kvatinsky is with the Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa 32000, Israel (email: shahar@ee.technion.ac.il).

Eitan Yaakobi is with the Department of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel (email: yaakobi@cs.technion.ac.il).

The work of Y. Cassuto was supported in part by the European Union Marie Curie CIG grant, by the Intel Center for Computing Intelligence, and by the Israeli Ministry of Science and Technology. The work of S. Kvatinsky was supported by the Intel Center for Computing Intelligence and by a Viterbi computer engineering fellowship. The work of E. Yaakobi was supported in part by the ISEF Foundation and in part by the Lester Deutsch Fellowship. Part of the results in the paper were presented at the IEEE International Symposium on Information Theory, Istanbul Turkey, July 2013, and part at the IEEE International Conference on Signal Processing and Communications, Bangalore India, July 2014.

and its mitigation. We begin the information-theoretic treatment of sneak paths in Section II by calculating the error probability as a function of the array dimensions and the distribution of the written bits. We give precise closed-form expressions for the error probability as a function of these parameters, and also extend the calculations to a model where an error occurs only if at least L distinct sneak paths affect the read cell. This formulation of sneak-path arrays as error channels lays the foundation to coding for mitigating sneak-path errors.

An alternative approach to mitigating sneak paths is to eliminate them altogether by a proper constraint code. Here coding works to restrict the written bits in the array to bit assignments that do not induce any sneak path to any cell. The number of sneak-path free assignments to an $m \times n$ memristor array was calculated by Sotiriadis in [14]. Unfortunately, straightforward elimination of sneak paths in the array implies storage capacity that vanishes with the array dimensions, which is clearly undesirable in practice. Hence our focus here is on generalizing the sneak-path constraint to practical scenarios that do not result in vanishing capacity. In Section III we derive a succinct characterization of sneak-path free arrays that enables the subsequent study of generalized sneak-path constraints. Then Section IV presents methods to obtain high-rate sneak-path free coding by selectively *grounding* array rows. Grounding array rows is a well known technique to eliminate sneak paths [1]. Grounding, however, will increase significantly the readout current [13]. Hence the introduction of grounding methods enables a tradeoff between power consumption (grounding many rows increases read power) and storage capacity (grounding few rows enforces harder constraints and reduces capacity). For each of two proposed grounding methods we match a corresponding constraint, and calculate the storage capacity resulting from the constraint. Both methods are shown to have non-vanishing capacities, and for both we present capacity-achieving encoders. It is an interesting finding from practical standpoint that for the read method that achieves the higher capacity from the two, capacity can be achieved by a 1-dimensional (d, ∞) run-length limited constraint. Because the 1-dimensional constraint is only sufficient for the sneak-path constraint, but not necessary, proving this result requires careful enumeration of words outside the intersection of the constraints. We close Section IV with a general model for trading off read power and storage rate in memristor arrays, which can help future work generalizing the specific models we addressed herein. Finally, we conclude the paper in Section V with some forward-looking remarks on employing error-correcting codes in memristor arrays.

II. Error Probabilities due to Sneak Paths

We start the information-theoretic treatment of sneak paths by examining the errors that they cause, and calculating the resulting error probabilities. In this section and throughout the paper we assume that a resistive cell at location (i, j) programmed to the "0" state (high resistance) is read in error as being at the "1" state (low resistance) when it is affected by at least one sneak path of any length, i.e., there exists a path as defined in Definition 1.

Definition 1. *Given a binary array A of size $m \times n$, we say that there is a **sneak path** of length $2k + 1$ affecting the cell at position (i, j) if $a_{i,j} = 0$ and there exist $2k$ positive integers $1 \leqslant r_1, \ldots, r_k \leqslant m$ and $1 \leqslant c_1, \ldots, c_k \leqslant n$ for some $k \geqslant 1$ such that the following $2k + 1$ cells satisfy*

$$a_{i,c_1} = a_{r_1,c_1} = a_{r_1,c_2} = \cdots = a_{r_{k-1},c_k} = a_{r_k,c_k} = a_{r_k,j} = 1.$$

The sneak path is a closed path originating from and returning to (i, j) and traversing "1"-state cells through alternating vertical and horizontal steps. The integers $r_1, \ldots, r_k$ and $c_1, \ldots, c_k$ are, respectively, the row and column indices of the traversed cells.

A. Calculating the sneak-path bit-error probability

In an $m \times n$ array we want to calculate the probability that a certain bit will be in error due to one or more sneak paths affecting it. To this end we restrict ourselves to sneak paths of length 3 ($k = 1$ in Definition 1), and define that a bit will be in error due to sneak path if the following two conditions are met:

1) The bit value is 0.
2) The bit location (i, j) has at least one combination c_1, r_1 that induces a sneak path defined by

$$a_{i,c_1} = a_{r_1,c_1} = a_{r_1,j} = 1. \tag{1}$$

According to the definition of sneak path given in (1), we only consider in the analysis sneak paths with 3 cells, which is a special case of the $2k + 1$-cell sneak path given in Definition 1. We do so for two reasons. One is that sneak paths with more than 3 (5, 7, etc.) cells are less prone to errors because of their higher resistance. Practical realizations of memristors implement non-linearity in the cell resistance, and so the small increase of the path length from 3 cells to 5 may be sufficient for the sneak current to drop to a level that would not induce a read error. The second reason is that analyzing longer sneak paths is much more difficult. In Section III we show that when determining sneak-path existence in a full $m \times n$ array, it is sufficient to consider 3-cell sneak paths as in (1).

We denote by P the *conditional probability* that given that a cell is programmed to the value 0 it will be read as 1 due to sneak path. To get from P the overall array bit error probability due to sneak paths, we multiply it by the fraction of 0 bits in the array $\Pr(a_{i,j} = 0)$, because all the sneak-path errors occur to 0 bits. Throughout this section all the sneak-path error probabilities are conditional given a 0 bit, but we keep this conditioning implicit for notational convenience. The main challenge in finding P stems from the fact that there are many possible c_1, r_1 combinations creating sneak paths, and multiple of them may exist simultaneously for the same array assignment. We now define the problem formally.

Problem 1. *Given an array of dimensions $m \times n$, where bits are written to array locations such that $\Pr(a_{i,j} = 1) = q$, $\Pr(a_{i,j} = 0) = 1 - q$, i.i.d. for all (i, j). What is the probability P that a 0-written array bit is read in error due to sneak path?*

It is clear that the answer to Problem 1 depends on the parameter q. For example, it is possible to trivially make P identically zero by setting $q = 0$ (hence having no 1s in the array to create sneak paths). However, this would not be a wise choice as the resulting information rate is zero. The sneak-path error probability also depends on the array dimensions, hence we re-denote P as $P(m, n, q)$.

Theorem 2. *The error probability due to sneak paths in an $m \times n$ array with parameter q equals*

$$P(m,n,q) = 1 - \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} \binom{m-1}{u} \binom{n-1}{v} \cdot q^{u+v} (1-q)^{m-1-u+n-1-v+uv}. \tag{2}$$

Proof: The proof proceeds by summing the probabilities of bit assignments for which there is *no* sneak path affecting cell (i, j). Taking the complement yields $P(m, n, q)$.
We consider a location (i, j) where the bit value is 0. Suppose column j has u 1s in row locations taken from $\{1, \ldots, m\} \setminus i$, and row i has v 1s in column locations taken from $\{1, \ldots, n\} \setminus j$. Then in order for cell (i, j) to have no sneak path, each intersection of a 1 in column j with a 1 in row i must have a 0 value. An example for an array with no sneak path for cell $(i, j) = (1, 1)$ is given in Fig. 2.

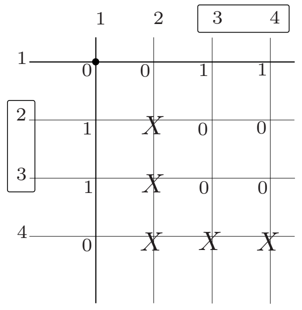

Fig. 2. An example of an array with no sneak path for cell $(1, 1)$. Given 0/1 assignments in row $i = 1$ and column $j = 1$, the intersections of the 1-rows with the 1-columns must be set to zeros. In the rest of the array locations, the value can be arbitrary, marked with X.

The probability that all these intersections have 0s is $(1 - q)^{uv}$. Now all that is needed to obtain the second term of (2) is to sum over all u from 0 to $m - 1$ and all v from 0 to $n - 1$ and weight with their respective probabilities.

■

Theorem 2 gives an exact closed-form expression for the probability that randomly selecting the $m \times n$ array bits i.i.d. with parameter q will result in at least one sneak path affecting location (i, j). Note that the same expression applies to any array location, as (2) does not depend on (i, j). This implies that an encoder with bias q induces information reliability given by (2). It is important to observe that for two locations (i_1, j_1) and (i_2, j_2) on the same array, sneak-path error events are *not independent*. For example, if $i_1 = i_2$, knowing that (i_1, j_1) has a sneak path makes a sneak path for (i_2, j_2) more likely. The dependence between sneak-path errors within the array is discussed in more detail in Section II-D.

B. Error probability due to L or more sneak paths

It may be the case in practice that one sneak path is not sufficient to cause a bit error. Rather, the sensing circuit may have sufficient margins to tolerate up to $L - 1$ sneak paths affecting an array location, in which case a bit error due to sneak paths requires at least L sneak paths affecting the same array location. For this case we derive a generalized expression for the bit-error probability due to L or more sneak paths.

Theorem 3. *The error probability due to **L or more** sneak paths in an $m \times n$ array with parameter q equals*

$$P_L(m,n,q) = \sum_{l=L}^{(m-1)(n-1)} \sum_{u=1}^{m-1} \sum_{v=1}^{n-1} \binom{m-1}{u} \binom{n-1}{v} \binom{uv}{l} \cdot q^{u+v+l} (1-q)^{m-1-u+n-1-v+uv-l}. \tag{3}$$

Proof: Given u 1s in column j and v 1s in row i, an array will induce l sneak paths on location (i, j) if it has exactly l 1s out of the uv cells that intersect a 1-row of column j and a 1-column of row i. There are $\binom{uv}{l}$ combinations to choose these l 1s. To restrict to exactly l 1s in the probability expression, we add (on top of (2)) l to the exponent of q to get q^{u+v+l} and $uv - l$ to the exponent of $1 - q$ to get $(1 - q)^{m-1-u+n-1-v+uv-l}$. Summing for all l greater or equal to L yields (3). ■
One can verify that $P(m, n, q) = P_1(m, n, q)$, hence Theorem 2 is a special case of Theorem 3.

C. Asymptotic analysis

A practically important question is whether reliable readout is possible in the limit of large memory arrays. As it turns out, one can increase the array size without limit while keeping the error probability constant and bounded away from 1. That statement is true if the array grows in one dimension, and is kept constant in the other. If the array grows unbounded in both dimensions, then we show that the error probability does tend to 1. We start with the positive result.

Proposition 4 *Given an integer constant b, the error probability $P(b,n,q)$ is bounded from above by a constant strictly smaller than 1 as n tends to infinity. The upper bound on the error probability equals*

$$P(b,n,q) \leqslant 1-(1-q)^{b-1}. \tag{4}$$

Proof: For a constant number b of rows (which include row i and $b-1$ additional rows), the probability that column j has all 0s equals

$$(1-q)^{b-1}.$$

Since having all 0s in column j guarantees that there are no sneak paths, we get the bound in (4). ■

The way in practice to keep the array-size constant in one dimension is by grounding all rows outside a $b \times n$ sub-array containing the read cells. When using the grounding technique the value of b sets a tradeoff between the error probability and the power consumption during read. The following result shows that it is not possible to grow the array to infinity in both dimensions while maintaining a constant error probability.

Theorem 5. *The error probability $P(m,n,q)$ tends to 1 if m and n tend to infinity, for any constant q.*

Proof: As we did in the proof of Theorem 2, we will look at the complement of $P(m,n,q)$ and prove that it tends to 0 for m and n tending to infinity. Suppose that column j is assigned a particular vector $\boldsymbol{u}$ with weight $u=1$. Then the probability that there is no sneak path conditioned on $\boldsymbol{u}$ in column j is given by

$$1-P(m,n,q|\boldsymbol{u}) = \sum_{v=0}^{n-1}\binom{n-1}{v}q^v(1-q)^{n-1-v}(1-q)^v. \tag{5}$$

The power of q in (5) is the number of 1s in row i and the first power of $1-q$ is the number of 0s in row i. The second power of $1-q$ is the number of zeros required to not have sneak path when the weight of $\boldsymbol{u}$ is 1. Simplification of (5) yields

$$\begin{aligned}1-P(m,n,q|\boldsymbol{u}) &= \\ (1-q)^{n-1}(1+q)^{n-1} &= (1-q^2)^{n-1} \underset{n\to\infty}{\longrightarrow} 0.\end{aligned} \tag{6}$$

It is clear that for any vector $\boldsymbol{u'}$ in column j with weight $u \geqslant 1$, the probability of having no sneak path conditioned on $\boldsymbol{u'}$ similarly tends to 0. (Intuitively, more 1s in column j mean fewer assignments to the remaining bits that do not cause sneak-path.) Hence the probability of no sneak path conditioned on the weight of column j being greater or equal to 1 tends to 0 as n tends to infinity. To prove that the same applies even without conditioning, we observe that the probability that column j has weight 0 ($\boldsymbol{u'}$ is the all-zero vector) tends to 0 as m tends to infinity. This completes the proof. ■

D. Joint distribution of sneak-path errors

The probability to have a sneak-path error is shown in the previous section to be independent of the location in the array, only depending on the array and source parameters. However, it is not hard to see that a sneak-path error in one array location is *not* independent from a sneak-path error in another location given the parameters (m,n,q). For example, a sneak path affecting an array location increases the likelihood of a sneak path affecting another location in the same row or column, as sneak paths affecting locations in the same row/column have many cells in common. One option to mitigate this dependence is to perform sneak-path error-correction coding *across arrays* (i.e., each bit in a codeword is assigned to a different array with the same parameters), so that errors within a code block can be assumed i.i.d. However, restricting coding to be performed cross-array has significant practical drawbacks, mainly increased latency due to the need to read multiple arrays before the decoding of a code block can start. So to avoid this, we study in this section the dependence between sneak-path errors in the same array.

Consider an $m \times n$ array with bias q. Denote by $e_{i,j}$ the event that cell (i,j) is affected by a sneak path, and by $\bar{e}_{i,j}$ the complementary event that cell (i,j) is not affected by a sneak path. In the notation of Section II-A we write $\Pr(e_{i,j}) = P(m,n,q)$ and $\Pr(\bar{e}_{i,j}) = 1-P(m,n,q)$. We now want to examine the probability that *two cells* within the same array column are affected by sneak paths. In other words, we want to calculate the joint probability

$$\Pr(e_{i,j}, e_{i',j}),$$

for some index triple i,i',j. We may similarly be interested in the joint probability $\Pr(\bar{e}_{i,j}, \bar{e}_{i',j})$ (one can be obtained from the other and the individual probability $\Pr(e_{i,j})$). Denote by u the number of 1s in column j, by v the number of 1s in row i, and by v' the number of 1s in row i'. In addition, denote by σ the number of column indices in which both rows i,i' have 1s. Hence $\sigma \leqslant \min\{v,v'\}$.

Theorem 6. *The joint probability $\Pr(\bar{e}_{i,j}, \bar{e}_{i',j})$ in an $m \times n$ array with parameter q equals the expression in (7).*

Proof: Given u,v,v' and the overlap σ, avoiding sneak paths for both cells (i,j) and (i',j) requires 0 assignments to the cells that intersect the u row indices where column j has 1s, with the union of the v and v' column indices where row i or i' (or both) have 1s. The size of this union is $v+v'-\sigma$, and this adds the right

$$\Pr(\bar{e}_{i,j},\bar{e}_{i',j}) = \sum_{u=0}^{m-2}\sum_{v=0}^{n-1}\sum_{v'=0}^{n-1}\sum_{\sigma=0}^{n-1}\binom{m-2}{u}\binom{n-1}{v}\binom{v}{\sigma}\binom{n-1-v}{v'-\sigma} q^{u+v+v'}(1-q)^{m-2-u+n-1-v+n-1-v'+u(v+v'-\sigma)}. \tag{7}$$

term $u(v+v'-\sigma)$ to the exponent of $(1-q)$ in (7). The remaining terms in the exponent of $(1-q)$, as well as the terms in the exponent of q, follow from the assignment of 0s and 1s to column j and rows i, i' prescribed by u, v, v'. The latter two binomial coefficients in (7) count the number of assignments of v' column indices having overlap σ with a given set of v column indices. ■

Thanks to symmetry between rows and columns[1] the formula (7) can be used also for the case of two cells in the same row.

We now use Theorem 6 to compare the joint error probability of two same-column cells to the probability of double bit error assuming independent errors. Intuitively, since two cells in the same column share the same number u of 1s in the column, we expect a higher likelihood of both erring together (when u is high), and also a higher likelihood of both not erring together (when u is low). We validate this intuition quantitatively in the following figures. Fig. 3 compares the probability $\Pr(\bar{e}_{i,j},\bar{e}_{i',j})$ to the square of the probability $1-P(m,n,q)$ from Section II-A. Fig. 4 compares the probability $\Pr(e_{i,j},e_{i',j})$

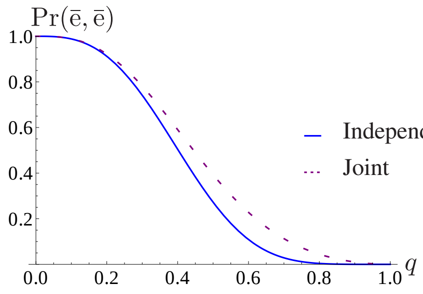

Fig. 3. Comparing the probability $\Pr(\bar{e}_{i,j},\bar{e}_{i',j})$ from Theorem 6 (dashed line) to the square of the individual-cell no-error probability from Theorem 2 (solid line).

(calculated from $\Pr(\bar{e}_{i,j},\bar{e}_{i',j})$ and the individual error probability) to the square of the probability $P(m,n,q)$. Figs. 3 and 4 show that both the joint no-error probability and the joint error probability of cells in the same column (or row) are higher than those probabilities had errors been independent.

It is possible to extend Theorem 6 to obtain the joint error probabilities of more than 2 cells in a row or column. The importance of deriving joint error probabilities is that these probabilities can be used to obtain error models for the analysis and design of error-correcting codes.

[1]When we transpose an $m\times n$ array we obtain an $n\times m$ array where rows and columns switch roles.

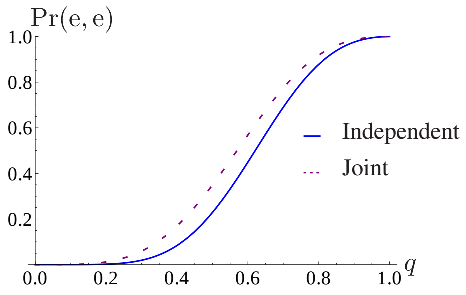

Fig. 4. Comparing the probability $\Pr(e_{i,j},e_{i',j})$ (dashed line) to the square of the individual-cell error probability (solid line).

III. Characterization of Sneak-Path Free Arrays

The approach manifested in the previous section is to regard sneak paths as a source of errors, and using error-correcting codes to combat them. Now in this section we explore an alternative approach of constraining the array to bit assignments that do not have any sneak path for any cell (i,j). Such an approach would give a complete solution to the sneak-path problem without need to employ error-correcting codes.

To obtain sneak-path free arrays, let us first define formally and mathematically the required constraint. Recall that for a cell at location (i,j) that is programmed to "0" a sneak path is defined according to Definition 1. Hence we say that an array A satisfies the **sneak-path constraint** if no cell within it has a sneak path *of any length*. In these cases we call the array A a **sneak-path-free array**.

The sneak-path constraint was already introduced and studied in [14] with application to nanowire resistive crossbar switching networks (R-CSNs). This previous work addressed the same problem of high-resistance cells being "short-circuited" by paths of cells at low-resistance state. The contributions of [14] include an exact count of the number of "0", "1" $m\times n$ arrays that are distinguishable by measuring resistance at the array row/column terminals. This count can be easily seen to be identical to the number of distinct sneak-path-free arrays. However, the more refined characterization of the sneak-path constraint pursued here allows obtaining superior storage information rates for more general sneak-path problems motivated by memristor arrays. For completeness and clarity we include in the presentation results for the simple sneak-path model, which can be implied by results in [14].

For the ability to extend sneak-path-free coding results to more general models, it is useful to represent the sneak-path constraint by a new, more succinct constraint, which is later shown to be equivalent. It turns out that the existence of sneak paths of any length in a memristor array can be perfectly characterized by an abstract constraint, which we call the *isolated zero-rectangle constraint*.

Definition 7. *A binary array A has an* ***isolated zero rectangle*** *if there are four positive integers $i_1 \neq i_2$ and $j_1 \neq j_2$ such that*

$$a_{i_1,j_1} + a_{i_1,j_2} + a_{i_2,j_1} + a_{i_2,j_2} = 3.$$

That is, the value of exactly one out of the four cells in the rectangle formed by these four positions is zero.

Note the similarity between Definition 7 and Definition 1 for the special case of $k = 1$. The difference is that Definition 1 characterizes sneak paths affecting the particular cell at location (i, j), while Definition 7 characterizes the existence of sneak paths affecting *any* cell in the array. An array A satisfies the **isolated zero rectangle constraint** if it has no isolated zero rectangles and then it is called an **isolated zero rectangle free array**.

According to the last definition, a ”0” belongs to an isolated-zero rectangle if it is part of any rectangle in the array, all of whose remaining vertices are ”1”s. For example, the cell in the $(4, 1)$ location in Fig. 1(b) belongs to an isolated zero rectangle because it is part of a rectangle (marked by a dashed line) with three ”1”s at locations $(1, 1)$, $(1, 3)$ and $(4, 3)$. There are no other isolated zero rectangles in the array.

Next we show that a memristor array is free of sneak paths of any length if and only if it has no isolated zero-rectangles. Note that sneak paths may consist of any odd number of cells greater than one, not necessarily three as in the rectangle case. However, this property tells us that rectangles, i.e. sneak paths of length three, provide a complete characterization of the existence of sneak paths.

Theorem 8. *The sneak path constraint and the isolated zero rectangle constraint are equivalent.*

Proof: We will show that an array has a sneak path if and only if it has an isolated zero rectangle. One direction is immediate. Assume that an array A has an isolated zero-rectangle. Then, there are $i_1 \neq i_2$ and $j_1 \neq j_2$ such that exactly one out of the four cells $\{(i_1, j_1), (i_1, j_2), (i_2, j_1), (i_2, j_2)\}$ has value ”0”. Then the remaining three cells have value ”1” and satisfy the condition of Definition 1 with $k = 1$. Thus, we are only left with showing that if an array has a sneak path then it also has an isolated zero rectangle.

Let us assume to the contrary that there exists an array A which has a sneak path affecting the (i, j) cell and yet it satisfies the isolated zero rectangle constraint. First note that $a_{i,j} = 0$ and there is a path as defined in Definition 1 starting at the i-th row and ending at the j-th column. Assume the vertices of this path are the cells at positions $(i, c_1), (r_1, c_1), (r_1, c_2), \ldots, (r_{k-1}, c_k), (r_k, c_k), (r_k, j)$ for some $k \geqslant 1$, and these array cells have value ”1”.

We will show by induction that for all $1 \leqslant h \leqslant k$, $a_{r_h,c_1} = 1$. This property holds for $h = 1$ since the (r_1, c_1) cell is part of the sneak path. Assume the claim is true for some $1 \leqslant h < k$, that is, $a_{r_h,c_1} = 1$. We will show that $a_{r_{h+1},c_1} = 1$ as well. Note that the vertices $(r_h, c_{h+1}), (r_{h+1}, c_{h+1})$ belong to the sneak path and hence $a_{r_h,c_{h+1}} = a_{r_{h+1},c_{h+1}} = 1$. Therefore, in the rectangle formed by the vertices

$$(r_h, c_{h+1}), (r_h, c_1), (r_{h+1}, c_{h+1}), (r_{h+1}, c_1)$$

the first three cells have value one. Therefore, according to the assumption that there is no isolated zero rectangle we conclude that $a_{r_{h+1},c_1} = 1$.

From the last claim we get in particular that $a_{r_k,c_1} = 1$. Since the vertices $(i, c_1), (r_k, j)$ belong to the sneak path, we have $a_{i,c_1} = a_{r_k,j} = 1$ and since the sneak path affects the cell at position (i, j) we also have $a_{i,j} = 0$. Therefore, there exists a sneak-path with three cells $(i, c_1), (r_k, c_1), (r_k, j)$ in contradiction with the assumption that there are no isolated zero rectangles. ■

From the isolated zero rectangle characterization it is implied that for sneak paths to not exist in the array, the ”1” cell locations in any pair of rows (or columns) must have either full overlap or no overlap. For example, rows 2, 3 in Fig. 1(b) have full overlap of ”1”s, rows 2, 4 have no overlap of ”1”s, and thus no sneak paths exist between these row pairs. However, rows 1, 4 have neither full-overlap nor no-overlap, and thus introduce a sneak path.

Lemma 9. *An array A is an isolated zero rectangle free array if and only if the ”1”s in every two rows either completely overlap or are disjoint.*

Proof: It is clear that the condition is sufficient. If ”1”s either completely overlap or have no overlap between every pair of rows, then every rectangle has either 0,1, 2 or 4 ”1”s.

To prove necessity, assume to the contrary that the condition does not hold. That is, there are two rows, say the i-th and j-th rows, such that the ones in these rows neither completely overlap nor are disjoint. Assume without loss of generality that there are more ones in the i-th row and assume that there are $\ell_i \geqslant 2$ ones in positions $1, \ldots, \ell_i$. Since the ones in the two rows are not disjoint, there is $1 \leqslant k \leqslant \ell_i$ such that $a_{j,k} = 1$, and since they do not fully overlap, there is $1 \leqslant h \leqslant \ell_i$, $h \neq k$ such that $a_{j,k} = 0$. Thus, the rectangle formed by the vertices $\{(i, k), (i, h), (j, k), (j, h)\}$ is an isolated-zero rectangle and so the array A does not satisfy the isolated zero rectangle constraint. ■

Let $N(m, n)$ be the number of $m \times n$ arrays satisfying the isolated zero-rectangle constraint. An exact count of $N(m, n)$ (for an equivalent constraint) is derived in [14].

For the sake of completeness and clarity of the results that follow, we provide a proof of the result that uses the isolated zero rectangle constraint and its characterization in Lemma 9.

First, we denote by $S(k,\ell)$ the number of distinct ways that a set of k elements can be partitioned into ℓ nonempty subsets, where it is known that

$$S(k,\ell) = \frac{1}{\ell!}\sum_{t=0}^{\ell}(-1)^{\ell-t}\binom{\ell}{t}t^k = \frac{1}{\ell!}\sum_{t=0}^{\ell}(-1)^{t}\binom{\ell}{t}(\ell-t)^k.$$

This is known as the Stirling number of the second kind [16].

Lemma 10. *The value $N(m,n)$ is expressed by*

$$N(m,n) = 1+ \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\binom{m}{i}\binom{n}{j}\sum_{\ell=1}^{\min\{m-i,n-j\}} S(m-i,\ell)S(n-j,\ell)\ell!.$$

Proof: Assume A is an array that satisfies the isolated zero-rectangle constraint and is not the all zero array. First, let us consider the zero rows and columns in A. Assume that A has i zero rows and j zero columns where $0 \leqslant i \leqslant m-1$ and $0 \leqslant j \leqslant n-1$. There are $\binom{m}{i}$ options to choose these rows and $\binom{n}{j}$ to choose the columns. After removing these i rows and j columns we obtain an $(m-i)\times(n-j)$ array A' with no zero rows or zero columns.

We follow the observation from Lemma 9 in which the "1"s in every row either completely overlap or are disjoint, and note that the same property holds also for the columns (otherwise this property doesn't hold for the rows). Therefore, the rows of A' can be partitioned into some $1 \leqslant \ell \leqslant m-i$ sets such that the rows in every set are identical. A set of identical rows determines also a set of identical columns, and hence the columns of A' can also be partitioned into ℓ sets such that the columns in every set are identical. The main idea in characterizing A' is to define the partition of ℓ sets of rows, ℓ sets of columns, and then match between these two partitions. See Fig. 5 for an illustration of the partitions of sets of rows and columns and then matching between them.

The number of different options to partition the $m-i$ rows into ℓ nonempty sets is $S(m-i,\ell)$. Similarly, the number of different options to partition to the $n-j$ columns into ℓ nonempty sets is $S(n-j,\ell)$. Lastly, there are $\ell!$ options to match between the ℓ sets of rows and ℓ sets of columns, yielding the expression

$$\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\binom{m}{i}\binom{n}{j}\sum_{\ell=1}^{\min\{m-i,n-j\}} S(m-i,\ell)S(n-j,\ell)\ell!,$$

for the number of possible arrays A. Together with the all zero array, we get the result stated in the lemma. ■

The second, more compact, expression for $N(m,n)$ in [14] can similarly be obtained using the isolated zero rectangle constraint. We omit the proof since it is similar to the one of Lemma 10, and another proof of this result can be found in [14].

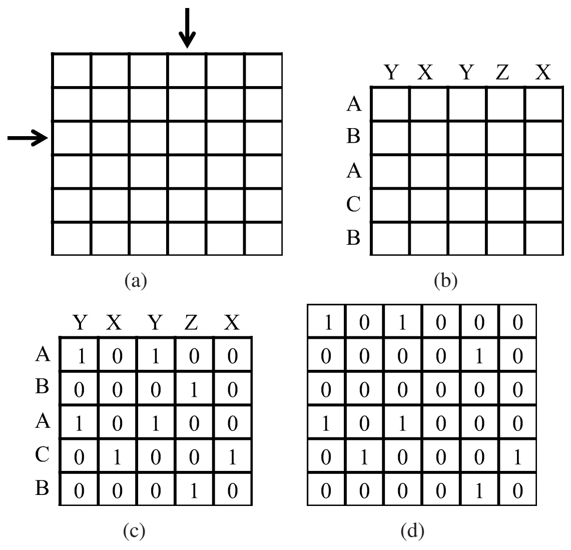

Fig. 5. Fig. 5(a) describes the first step of choosing the third row and fourth column as the zero rows and columns in the array A. In Fig. 5(b), the partition of rows and columns into three sets is determined where rows and columns with the same letter will be identical. In Fig. 5(c), a matching between the three sets of rows and columns, which is fixed by $A \to Y, B \to Z, C \to X$, determines the positions of "1"s in the array A'. Lastly, in Fig. 5(d), the zero rows and columns are brought back to A' to get the encoded array A.

Lemma 11. *The value $N(m,n)$ can be expressed by*

$$N(m,n) = \sum_{\ell=0}^{\min\{m,n\}} S(m+1,\ell+1)S(n+1,\ell+1)\ell!.$$

Unfortunately, the asymptotic behavior of the value $N(m,n)$ for m and n large enough states that $\log_2 N(m,n) \approx (m+n)\log_2(m+n)$ in case both m and n approach infinity and the ratio m/n approaches some positive number [14]. Thus, under these conditions it is derived that

$$\frac{\log_2 N(m,n)}{mn} \longrightarrow 0, \tag{8}$$

which implies a 0 asymptotic storage capacity. In fact, this behavior holds for all values of m and n which approach infinity (that is, the ratio m/n does not have to approach a positive number). This indicates that the sneak path constraint is too strong, and we need to find milder ways to avoid sneak paths without ending up with zero capacity. This will be the topic of the next section.

IV. Representations trading off sneak paths and power consumption

One way to eliminate memristor sneak paths without resorting to any information-theoretic tools is by electrically grounding all rows except the one being read [13]. The problem with grounding all other rows is that it significantly increases the power consumption of the read operation due to lower equivalent resistance through which flows the measurement current. Without information theoretic tools, this suggests a tradeoff between power consumption (from grounded rows) and read errors (from sneak paths). Alternatively, we propose to replace the power-correctness tradeoff with a power-density one, by combining partial grounding with sneak-path constraint codes. The key idea is to specify how many of the rows will be grounded in a read operation, and ensure that no sneak paths exist in the part of the array remaining "active" in the non-grounded rows. By doing that, we can control the power consumption of the read operation while preventing sneak-path errors. Since many of the cells will be deactivated in grounded rows, maintaining sneak-path-free reads will be possible with good storage rates. The grounding schemes proposed in the sequel are especially attractive for practical memories, because they leave ungrounded the rows *closest* to the read cell, which maximizes the read-power savings. In contrast, the power consumed by the farther grounded rows is less significant thanks to the resistance of the wires. In the next two sub-sections we propose two distinct grounding schemes, and explore the resulting information-theoretic problems to guarantee sneak-path free reads.

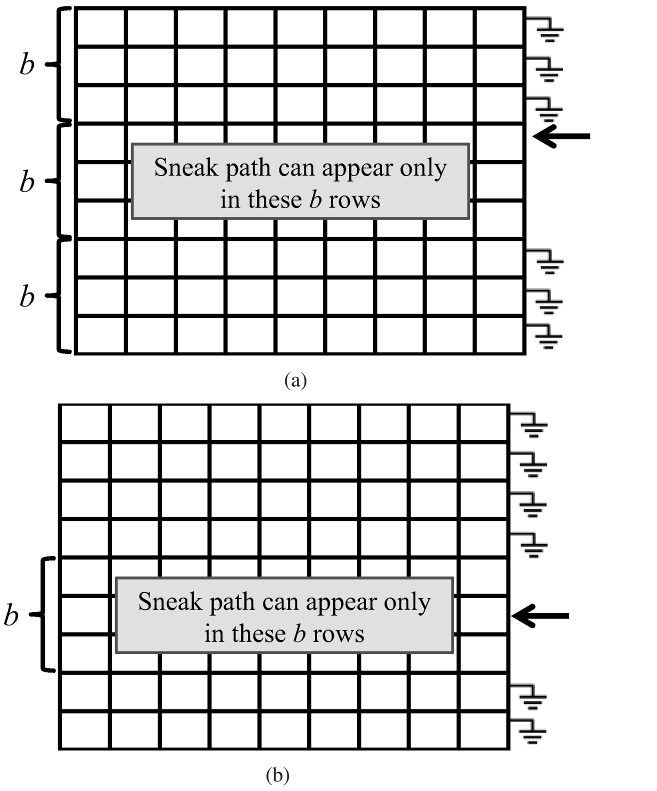
b
b
b
Sneak path can appear only in these b rows
(a)
b
Sneak path can appear only in these b rows
(b)

Fig. 6. (a) depicts the grounding method based upon fixed subsets, and (b) depicts the grounding method based upon the read row. The read row is marked by an arrow in both cases.

A. Grounding based upon fixed subsets

In this sub-section we divide the array rows into disjoint subsets, and choose to ground all the rows *except* those that are in the subset containing the read row. See Fig. 6(a). Given this grounding scheme, we study the coding required to prevent sneak paths. We will show that when the subset size is a constant, the capacity no longer goes to zero as in the full-array case.

Assume the array size is $m \times n$ and let b be some positive integer which is a divisor of m. The m rows are divided into m/b disjoint subsets of consecutive rows. Then, any of the m/b sub-arrays of size $b \times n$ is required to satisfy the isolated zero rectangle constraint. Since all these sub-arrays are disjoint and thus independent, we conclude that the number of arrays will be $N(b,n)^{m/b}$. Let us define the capacity of this constraint by $\mathbb{C}_1(b)$. Then, we get

$$\mathbb{C}_1(b) = \lim_{m,n\to\infty} \frac{\log\left(N(b,n)^{m/b}\right)}{mn} = \lim_{n\to\infty} \frac{\log\left(N(b,n)\right)}{bn}.$$

We first prove lower and upper bounds on the value of $N(b,n)$.

Lemma 12. *For any fixed b and n large enough the following holds*

$$(b+1)^n - b^{n+1} \leqslant N(b,n) \leqslant (b+1)!S(n+1,b+1).$$

Proof: According to Lemma 11

$$N(b,n) = \sum_{\ell=0}^{b} S(b+1,\ell+1)S(n+1,\ell+1)\ell!$$
$$\geqslant S(b+1,b+1)S(n+1,b+1)b!.$$

Since $S(b+1,b+1)=1$ and

$$S(n+1,b+1) = \frac{1}{(b+1)!}\sum_{i=0}^{b+1}(-1)^i\binom{b+1}{i}(b+1-i)^{n+1}$$
$$\geqslant \frac{(b+1)^{n+1}-(b+1)b^{n+1}}{(b+1)!} = \frac{(b+1)^n - b^{n+1}}{b!},$$

we get

$$N(b,n) \geqslant \frac{(b+1)^n-b^{n+1}}{b!}b! = (b+1)^n - b^{n+1}.$$

On the other hand, if b is fixed, let us show that for n large enough the following holds for every $0 \leqslant \ell < b$

$$S(b+1,\ell+1)S(n+1,\ell+1)\ell! \leqslant S(n+1,b+1)b!.$$

First note that

$$S(b+1,\ell+1)S(n+1,\ell+1)\ell!$$
$$\leqslant \frac{(\ell+1)^{b+1}}{(\ell+1)!}\cdot\frac{(\ell+1)^{n+1}}{(\ell+1)!}\cdot \ell! = \frac{(\ell+1)^{n+b}}{\ell!},$$

and we saw that $S(n+1,b+1) \geqslant \frac{(b+1)^n - b^{n+1}}{b!}$. Now,

$$\lim_{n\to\infty} \frac{\frac{(\ell+1)^{n+b}}{\ell!}}{\frac{(b+1)^n-b^{n+1}}{b!}} = \lim_{n\to\infty} \frac{b!(\ell+1)^{n+b}}{\ell!\left((b+1)^n - b^{n+1}\right)}$$
$$\leqslant \lim_{n\to\infty} \frac{b!(\ell+1)^{n+b}}{(b+1)^n - b^{n+1}}$$
$$= \lim_{n\to\infty} \frac{b!(\ell+1)^{n+b}}{(b+1)^n} \cdot \lim_{n\to\infty} \frac{(b+1)^n}{(b+1)^n - b^{n+1}}.$$

Let us evaluate every term independently under the assumption that b is fixed.

$$\lim_{n\to\infty} \frac{(b+1)^n}{(b+1)^n - b^{n+1}} = \lim_{n\to\infty} \frac{1}{1 - b\cdot\left(\frac{b}{b+1}\right)^n}$$
$$= \lim_{n\to\infty} \frac{1}{1 - b\cdot\left(\left(1-\frac{1}{b+1}\right)^b\right)^{n/b}}$$
$$= \lim_{n\to\infty} \frac{1}{1 - b\cdot e^{-n/b}} = 1.$$

Similarly,

$$\lim_{n\to\infty} \frac{b!(\ell+1)^{n+b}}{(b+1)^n} \leqslant \lim_{n\to\infty} \frac{(\ell+1)^{n+b}}{(b+1)^{n-b}}$$
$$\leqslant \lim_{n\to\infty} \frac{(\ell+1)^{n-b}}{(b+1)^{n-b}} \cdot b^{2b} = \lim_{n\to\infty} \left(\frac{\ell+1}{b+1}\right)^{n-b} \cdot b^{2b}$$
$$= \lim_{n\to\infty} \left(1 - \frac{b-\ell}{b+1}\right)^{n-b} \cdot b^{2b}$$
$$\leqslant \lim_{n\to\infty} \left(1 - \frac{1}{b+1}\right)^{n-b} \cdot b^{2b}$$
$$= \lim_{n\to\infty} \left(\left(1 - \frac{1}{b+1}\right)^{b}\right)^{\frac{n}{b}-1} \cdot b^{2b}$$
$$= \lim_{n\to\infty} e^{-\frac{n}{b}+1} \cdot b^{2b} = 0.$$

Therefore, we get that for n large enough

$$N(b,n) = \sum_{\ell=0}^{b} S(b+1,\ell+1)S(n,\ell+1)\ell!$$
$$\leqslant (b+1)S(n+1,b+1)b! = (b+1)!S(n+1,b+1).$$

■

Now we are ready to calculate the capacity $\mathbb{C}_1(b)$ for fixed values of b.

Lemma 13. *For any fixed* b, $\mathbb{C}_1(b) = \frac{\log(b+1)}{b}$.

Proof: According to Lemma 12

$$\lim_{n\to\infty} \frac{\log(N(b,n))}{bn} \geqslant \lim_{n\to\infty} \frac{\log\left((b+1)^n - b^{n+1}\right)}{bn}$$
$$= \lim_{n\to\infty} \frac{\log\left((b+1)^n\left(1 - b\left(\frac{b}{b+1}\right)^n\right)\right)}{bn}$$
$$= \frac{\log(b+1)}{b} + \lim_{n\to\infty} \frac{\log\left(1 - b\left(\frac{b}{b+1}\right)^n\right)}{bn} = \frac{\log(b+1)}{b}.$$

To prove the opposite inequality, again by Lemma 12 we get

$$\lim_{n\to\infty} \frac{\log(N(b,n))}{bn} \leqslant \lim_{n\to\infty} \frac{\log((b+1)!S(n+1,b+1))}{bn}$$
$$\leqslant \lim_{n\to\infty} \frac{\log((b+1)^{n+1})}{bn} = \frac{\log(b+1)}{b}.$$

■

B. Grounding sets based upon the read row

In this sub-section we choose to ground all the rows outside a subset of odd size *centered* at the read row. See Fig. 6(b). Given this grounding scheme, we study the coding required to prevent sneak paths. It turns out that a sufficient (but not necessary) condition to have sneak-path free reads in this case is that each column satisfies some run-length limited (RLL) [4] constraint, which depends on the number of ungrounded rows.

Under this model, we say that there is a ***b-centered-path***, where b is odd, affecting the cell in position (i,j) if $a_{i,j} = 0$ and there is a path as defined in Definition 1 which can be confined between the $(i - \frac{b-1}{2})$-th row and the $(i + \frac{b-1}{2})$-th row. That is, for some $k \geqslant 1$, there exist $2k$ positive integers $\max\{i - \frac{b-1}{2}, 1\} \leqslant r_1, \ldots, r_k \leqslant \min\{i + \frac{b-1}{2}, m\}$, $1 \leqslant c_1, \ldots, c_k \leqslant n$ such that

$$a_{i,c_1} = a_{r_1,c_1} = a_{r_1,c_2} = \cdots = a_{r_{k-1},c_k} = a_{r_k,c_k} = a_{r_k,j} = 1.$$

Thus, we say that an array satisfies the ***b-centered-path constraint*** if it has no b-centered-paths.

For any odd $b \geqslant 1$, we denote by $N_2(m,n;b)$ the number of arrays that satisfy the b-centered-path constraint and we denote the capacity of this constraint by $\mathbb{C}_2(b)$, so

$$\mathbb{C}_2(b) = \lim_{m,n\to\infty} \frac{\log\left(N_2(m,n;b)\right)}{mn}.$$

Furthermore, we say that an array has a ***b-isolated zero rectangle*** if there are four positive integers $i_1 \neq i_2$, $j_1 \neq j_2$, and $|i_2 - i_1| \leqslant b-1$, such that $a_{i_1,j_1} + a_{i_1,j_2} + a_{i_2,j_1} + a_{i_2,j_2} = 3$. An array A satisfies the ***b-isolated zero rectangle constraint*** if it has no b-isolated zero rectangles and then it is called a ***b-isolated zero rectangle free array***. The b-isolated zero rectangle constraint is the same as the isolated zero rectangle constraint from Definition 7 when applied to sub-arrays of A with b rows.

It is a matter of simple observation to get to the following correspondence between the centered path constraint and the isolated zero rectangle constraint:

Lemma 14. *The b-centered-path constraint and the $\frac{b+1}{2}$-isolated zero rectangle constraint are equivalent.*

To proceed, let us recall the one-dimensional RLL constraint. We say that a binary sequence satisfies the (d,k) RLL constraint if the number of zeros between every two consecutive ones is at least d and at most k. The capacity of the one dimensional (d,k) RLL constraint is denoted by $\mathbb{C}_{d,k}$. Next, we show that the capacity of the $(\frac{b-1}{2},\infty)$ RLL constraint is a lower bound on $\mathbb{C}_2(b)$.

Lemma 15. *For any odd b, $\mathbb{C}_2(b) \geqslant \mathbb{C}_{\frac{b-1}{2},\infty}$.*

Proof: This result follows from the observation that if every column satisfies the $(\frac{b-1}{2},\infty)$ RLL constraint then necessarily there are no pairs of ones in the same column at distance less than $\frac{b-1}{2}$ rows. In particular, there is no rectangle confined to $\frac{b+1}{2}$ rows with an isolated zero. ■

The reverse inequality on $\mathbb{C}_2(b)$ is proved in the next lemma. This direction is less immediate than the previous one, because the column $(\frac{b-1}{2},\infty)$ RLL constraint is *not* necessary for an array to satisfy the b-centered-path constraint.

Lemma 16. *For any odd b, $\mathbb{C}_2(b) \leqslant \mathbb{C}_{\frac{b-1}{2},\infty}$.*

Proof: Let $B_{m,n}$ be the number of $m\times n$ arrays where every column satisfies the $(\frac{b-1}{2},\infty)$ RLL constraint. Let A be a b-centered-path-free array. According to Lemma 14, A is a $(\frac{b+1}{2})$-isolated zero-rectangle free array. Thus, as in the proof of Lemma 9, in every $\frac{b+1}{2}$ consecutive rows of A, every two rows are either the same or their ones are located at disjoint locations.

For a positive integer divisor d of m, we define a mapping $F_d : \{0,1\}^{m\times n} \to \{0,1\}^{m\times n}$, which transforms an array A to $F_d(A)$ as follows. Starting with the first d rows of A, if there are identical rows among these d rows, then the first row remains the same and the subsequent identical rows are replaced with all-zero rows. Then the same operation is performed on the new array with the next window of d rows, between the second and $(d+1)$-th row, and so on until reaching the last window consisting of the last d rows.

Let A' be the array resulting under this mapping with $d=\frac{b+1}{2}$ on the array A, that is $A' = F_{\frac{b+1}{2}}(A)$. The array A' holds the property that every column satisfies the $(\frac{b-1}{2},\infty)$ RLL constraint.

We note that this mapping is many to one, as there can be several b-centered-path-free arrays A which will be mapped to the same array A'. Given an array A' we can bound the number of arrays A that are mapped to it. Assuming the array A' has x zero rows, then each row can be identical to any of the $\frac{b-1}{2}$ rows above it, or originally all-zero. Since there are m rows in the array, we can use a loose upper bound here (which will be sufficient for our goal), and say that at most m^m arrays will be mapped to the array A'. Therefore, we get the following relation

$$N_2(m,n;b) \leqslant m^m \cdot B_{m,n}.$$

Now we conclude that

$$\begin{aligned}\mathbb{C}_2(b) &= \lim_{m,n\to\infty} \frac{\log N_2(m,n;b)}{mn}\\ &\leqslant \lim_{m,n\to\infty} \frac{\log(m^m\cdot B_{m,n})}{mn}\\ &= \lim_{m,n\to\infty} \frac{m\log m + \log B_{m,n}}{mn}\\ &= \lim_{m,n\to\infty} \frac{\log B_{m,n}}{mn} = \mathbb{C}_{\frac{b-1}{2},\infty}.\end{aligned}$$

■

From Lemma 15 and Lemma 16, we get that

$$\mathbb{C}_2(b) = \mathbb{C}_{\frac{b-1}{2},\infty}.$$

It turns out that the symmetric grounding set method is better than the one based upon fixed subsets. In other words, we can prove the inequality $\mathbb{C}_2(b) \geqslant \mathbb{C}_1(b)$.

Theorem 17. *For all odd values of b, the following holds*

$$\mathbb{C}_2(b) \geqslant \mathbb{C}_1(b).$$

Proof: We need to show that $\mathbb{C}_{\frac{b-1}{2},\infty} \geqslant \frac{\log(b+1)}{b}$ for odd values of b. For $b < 250$ we numerically calculated the values of $\mathbb{C}_1(b)$ and $\mathbb{C}_2(b)$ to verify this inequality. For $b > 250$, we use a simple block-by-block coding argument giving a capacity lower bound for any t, $\mathbb{C}_{\frac{b-1}{2},\infty} \geqslant \frac{\log(t+1)}{\frac{b-1}{2}+t}$. In particular, we choose $t = \lfloor (b+2)/4 \rfloor$ and get that

$$\mathbb{C}_{\frac{b-1}{2},\infty} \geqslant \frac{\log(\lfloor (b+2)/4\rfloor + 1)}{\frac{b-1}{2} + \lfloor (b+2)/4\rfloor} \geqslant \frac{\log(\lfloor (b+2)/4\rfloor + 1)}{3b/4}.$$

Thus, it is enough to show that

$$\log(\lfloor (b+2)/4\rfloor + 1) \geqslant (3/4)\cdot(\log(b+1),$$

or

$$(b+2)/4 \geqslant (b+1)^{3/4},$$

which holds for $b > 250$. ■

To conclude, we compare in Table I between the numerical values of the capacities of the two grounding methods we introduced here for $b \leqslant 11$.

TABLE I
CAPACITY VALUES AND COMPARISON BETWEEN TWO GROUNDING METHODS.

b	$\mathbb{C}_1(b) = \frac{\log(b+1)}{b}$	$\mathbb{C}_2(b) = \mathbb{C}_{\frac{b-1}{2},\infty}$
2	0.792	-
3	0.667	0.694
4	0.580	-
5	0.517	0.551
6	0.468	-
7	0.423	0.465
8	0.396	-
9	0.369	0.406
10	0.346	-
11	0.326	0.362

C. Encoding of sneak-path free arrays

To complete the discussion of the different grounding schemes, we discuss in this section how to encode information using these methods. First note that encoding and decoding of arrays according to the grounding scheme based upon the read row (Section IV-B) can simply be done by the encoding and decoding of sequences which satisfy the RLL constraint. Therefore, we focus here on the first grounding method which is based upon fixed subsets (Section IV-A). In particular, we will study how to encode information to sneak-path free arrays of size $b \times n$, as well as arbitrary sizes.

Let b be a fixed positive integer number, and we will show how to encode information into $b \times n$ arrays that satisfy the isolated zero rectangle constraint. The idea is to partition the columns into $b+1$ sets of the same size (for simplicity we assume that n is a multiple of $b+1$). The first set of columns is assigned with the first row so the columns in this set have a single one in the first row. Similarly, the second set of columns is assigned with the second row and the columns in this set have a single one in the second row. This principle repeats until the b-th set where the columns in this set have a single one in the last row. Lastly, the columns of the $(b+1)$-th set will be the all-zero columns. The number of options to partition the n columns into $b+1$ sets of the same size is $\frac{n!}{\left(\frac{n}{b+1}\right)!^{b+1}(b+1)!}$ and there are $(b+1)!$ options to match these sets with the b rows and all-zero column. Then, using the approximation $\log x! \approx x \log x$ for x large enough, we get that the number of bits that can be stored this way is approximated to be

$$\log\left(\frac{n!}{\left(\frac{n}{b+1}\right)!^{b+1}}\right) = \log n! - (b+1)\log\left(\frac{n}{b+1}\right)! \approx$$
$$n\log n - (b+1)\cdot\left(\frac{n}{b+1}\right)\log\left(\frac{n}{b+1}\right) = n\log(b+1).$$

Therefore, the rate of this encoder is $\frac{\log(b+1)}{b}$, which equals the capacity $\mathbb{C}_1(b)$ of arrays satisfying the sneak-path free constraint with fixed number of rows b.

We also show how this encoding idea can be extended to an asymptotically optimal encoder for arrays of arbitrary dimensions $m \times n$, where m is not necessarily a constant. Let $S_1(k)$ be the set of all partitions of the numbers $\{1,\ldots,k\}$ into L groups, each consisting of $\frac{k}{L}$ numbers. Alternatively, we can treat $S_1(k)$ as the set of all multipermutations over $\frac{k}{L}$ numbers where each number appears L times. The size of $S_1(k)$ is

$$s_1(k) = |S_1(k)| = \frac{k!}{(\frac{k}{L})!^L \cdot L!}.$$

Assume for now that there is a one-to-one mapping with efficient encoding and decoding maps

$$F_1 : \{0,1\}^{\log s_1(k)} \to S_1(k)$$

between all binary vectors of length $\log s_1(k)$ and $S_1(k)$. Let S_2 be the set of all permutations of L numbers, so $s_2 = |S_2| = L!$, and similarly, assume that there is a mapping with efficient encoding and decoding maps

$$F_2 : \{0,1\}^{\log s_2} \to S_2.$$

Our approach here follows the proof of Lemma 10, which uses the if and only if condition in Lemma 9. We encode only arrays where the rows, columns are partitioned into L sets of $m/L, n/L$, rows, columns, respectively. Thus, every array is represented by: 1) a partition of the rows, that is, an element from $S_1(m)$, 2) a partition of the columns, again, an element from $S_1(n)$, and 3) a mapping between the L sets of rows and L sets of columns, i.e., an element from S_2. The encoding and decoding maps will be clear from the encoding and decoding of the mappings F_1 and F_2.

The number of bits that can be stored by this encoding is $N = \log(s_1(m)\cdot s_1(n)\cdot s_2) = \log s_1(m) + \log s_1(n) + \log s_2$. We use again the approximation $\log x! \approx x\log x$ for x large enough, and get

$$\begin{aligned}
N &= \log s_1(m) + \log s_1(n) + \log s_2 \\
&= \log\left(\frac{m!}{(\frac{m}{L})!^L \cdot L!}\right) + \log\left(\frac{n!}{(\frac{n}{L})!^L \cdot L!}\right) + \log(L!) \\
&= \log m! + \log n! - L\log\left(\left(\frac{m}{L}\right)!\right) - L\log\left(\left(\frac{n}{L}\right)!\right) - \log(L!) \\
&\approx m\log m + n\log n - L\cdot\frac{m}{L}\log\left(\frac{m}{L}\right) - L\cdot\frac{n}{L}\log\left(\frac{n}{L}\right) - L\log(L) \\
&= m\log m + n\log n - m\log\left(\frac{m}{L}\right) - n\log\left(\frac{n}{L}\right) - L\log(L) \\
&= (m+n-L)\log(L).
\end{aligned}$$

If we choose $L = \frac{m+n}{\log(m+n)}$ we have

$$N = \left(m+n-\frac{m+n}{\log(m+n)}\right)\log\left(\frac{m+n}{\log(m+n)}\right),$$

and for m,n large enough we get

$$\lim_{n\to\infty}\frac{N}{(m+n)\log(m+n)} = 1, \tag{9}$$

where $(m+n)\log(m+n)$ is asymptotically the number of bits that can be reliably stored in $m \times n$ arrays which are free of sneak paths, when the ratio m/n approaches some positive number. Thus, under the last assumption this mapping will be asymptotically optimal.

We note that the functions F_1 and F_2 indeed have implementations by applying different methods for the enumerations of permutations and multipermutations; see for example [2] and chapter 5.1 in [8]. However, these schemes are still not attractive enough for memory applications that require high speed. In order to have high encoding speed, a low complexity mapping was presented for $m \times n$ arrays in [14], while the number of information bits that this mapping can carry is $m\log n$. However, the number of bits that can be represented by all sneak-path free arrays is roughly $(m+n)\log(m+n)$. Thus, for square ($m = n$) arrays the mapping in [14] reaches approximately only a half of the number of bits that could be stored.

Lastly, we show another scheme which achieves both encoding and decoding with efficient complexity, and yet asymptotically achieves the number of bits that can be represented in these arrays. Let $1 \leqslant L \leqslant \min\{m, n\}$ be an integer number, and for simplicity assume it is a power of 2. We show how to encode $(m+n-L)\log(L)$ information bits using a mapping

$$H : \{0,1\}^{(m+n-L)\log(L)} \to A(m,n),$$

where $A(m,n)$ is the set of all sneak-path free arrays of size $m \times n$. We consider the input bits as a vector of $(m+n-L)$ length-$\log(L)$ vectors, denoted by $(\boldsymbol{r}_1, \dots, \boldsymbol{r}_m, \boldsymbol{c}_1, \dots, \boldsymbol{c}_{n-L})$, so $\boldsymbol{r}_i, \boldsymbol{c}_j \in \{0,1\}^{\log(L)}$, for $1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n-L$. Let ψ be a function which converts a length-$\log(L)$ binary vector to a number between 1 and L. The input bits are encoded to an array A as follows. The j-th column of A is denoted below by $a_{1:m,j}$.

1) For $i = 1, \dots, m$, set $a_{i,\psi(r_i)} = 1$.
2) For $j = 1, \dots, n-L$, set $a_{1:m,j+L} = a_{1:m,\psi(c_j)}$.
3) All other bits of the array A are set to the value zero.

It is not hard to see that the mapping H is bijective and there are no isolated zero rectangles. As in the first mapping in this section, if we choose $L = \frac{m+n}{\log(m+n)}$ we get the same asymptotic result as in (9). Note also that the encoding and decoding of this encoding map is efficient since it only requires to convert between a binary vector and an integer number.

To conclude the discussion in this section and to compare between the three encoding schemes discussed in this section, we plot in Fig. 7 the number of bits that can be stored by each scheme for the case of $m = n$ while n is a power of 2.

1) The first encoding scheme allows one to encode $2\left\lfloor \log\left(\frac{n!}{(n/L)^L \cdot L!}\right)\right\rfloor + \lfloor \log(L!) \rfloor$ bits, for $L = \lfloor \frac{2n}{1+\log(n)} \rfloor$.
2) The second encoding scheme is by [14] which stores $n\log(n)$ bits.
3) The third encoding scheme is the one given by the encoding map H and stores $(2n-L)\lfloor \log(L) \rfloor$ bits for $L = \lfloor \frac{2n}{1+\log(n)} \rfloor$.

D. Power model and savings

In this sub-section we use a simple electrical model of the crossbar array to show the power benefits of the proposed scheme. When reading cells in some array row, our proposed constraint scheme suggests to leave a small number $(b-1)$ of the nearby rows ungrounded. The impact of these ungrounded rows on power consumption will next be evaluated with a model we now define.

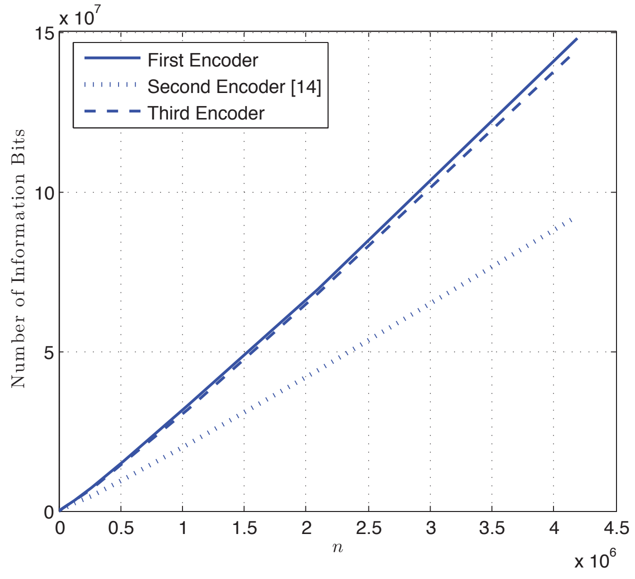

Fig. 7. A comparison between the number of bits stored in the three encoding schemes.

For the sake of clarity, we consider an extremely simple array model, with understanding that a much richer model is needed if practical realization is sought. Let the read cell be any cell in the i-th row. When other rows in the array are grounded, new current paths are created involving cells in the same *column* of the read cell. See Fig. 8 for a circuit-level view. It is sufficient in the figure

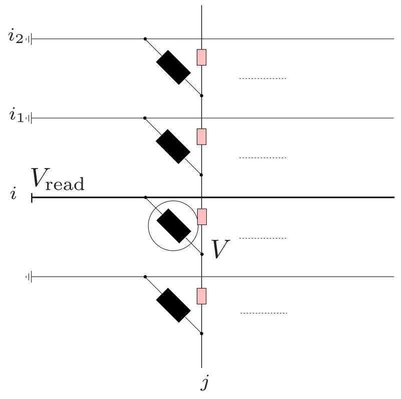

Fig. 8. A simple model for parasitic power when memristors are read with row grounding.

to consider only one column, because from symmetry other cells in the i-th row will see similar current paths in their respective columns (we assume here that multiple cells in the row are read in parallel, thereby amortizing the current paths through other columns over other read cells). When the cell in row i is read (marked in a circle in Fig. 8), a voltage V_{read} is applied on its row, and current is measured in its column. Then a voltage V develops on the column terminal of the read cell. The value of V is immaterial for this analysis, but any non-zero V implies a non-zero power consumed by other cells in the j-th column. In the forthcoming analysis we define power consumed by non-read cells in the column as *parasitic* power. Minimizing parasitic power is important not just for lowering energy costs, but also for reducing the cell disturbances and wearout. Due to wire resistances, the voltage V falls on a non-read cell in series to the wire connecting it to row i. We mark these serial wire resistances as small vertical rectangles in Fig. 8. As a result, the parasitic power consumed by a non-read cell decreases as its row distance to i increases. For example, the cell in row i_2 will consume less parasitic power than the cell in row i_1 because its serial resistance is larger. By this effect it is preferred to have the grounded rows far from the read row i, which is the case for the constraint schemes suggested earlier in this section. We next plot the power savings achieved by the b-centered constraint scheme with $b = 3$ and $b = 5$ in an array with $m = 64$ rows. Fig. 9 shows the *fraction of saved parasitic power* as a function of the cell non-linearity. We chose for the evaluation the weakest non-linearity model: *quadratic non-linearity* [1], in which a quadratic term is added to the linear dependence between current and voltage. Non-linearity value of 0 corresponds to plain linear resistive cells.

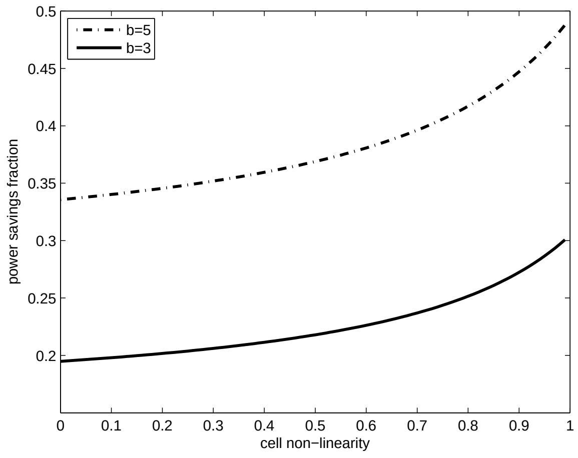

Fig. 9. Savings of parasitic power with b-centered ungrounded sets for $b = 3$ and $b = 5$ in an array with $m = 64$ rows. Savings are plotted as a function of the cells' non-linearity parameter. The wire resistance per row is taken as 0.2 times the low cell-resistance value[2].

It can be seen that without non-linearity the constraint scheme saves roughly 1/5 of the power with $b = 3$, and 1/3 of the power with $b = 5$. These fractions are much higher compared to the fractions of non-grounded rows $(b-1)/(m-1)$, which are 0.0317 and 0.0635 for $b = 3$ and $b = 5$, respectively. With non-linearity value of 1, the power savings are even more significant: around 0.3 for $b = 3$ and around 0.5 for $b = 5$. Higher non-linearity values reduce the parasitic powers for both the fully-grounded and the b-ungrounded schemes; the saving fractions increase with non-linearity because higher non-linearity makes the parasitic power consumed outside the b ungrounded rows more negligible. The results of Fig. 9 are obtained by summing the parasitic powers on all cells $i' \neq i$ in column j. Each parasitic power equals $V_{i'} \cdot I_{i'}$, where $V_{i'}$ is calculated by voltage divider between the cell resistance and the wire resistance to row i', and $I_{i'}$ is obtained as a function of $V_{i'}$ and the non-linearity parameter according to the model of [1] (without non-linearity $I_{i'}$ is obtained from $V_{i'}$ by Ohm's law and the constant resistance of the cell).

[2]While 0.2 seems a high relative resistance, some materials and geometries suffer from wire resistances at that scale [10].

E. Toward other schemes trading off sneak paths and power consumption

The solutions proposed in this section treated only one specific manifestation of the tradeoff between sneak paths and power consumption. Arguably this central tradeoff of memristor storage is much more general, and can benefit from additional models beyond the presently proposed ones. Our objective in this closing sub-section is to sketch the more general problem for future research.

In an $m \times n$ memristor array we define a *read configuration* $T_{i,j}$ as the set of voltages we apply to the rows and columns of the array in order to read cell (i, j). In its most generality, the *read power* consumed for reading cell (i, j) is given as a function

$$W(A, T_{i,j}),$$

where A is the binary matrix holding the values of all the cells in the array. Because the read configuration $T_{i,j}$ depends on (i, j), the power function W may depend on the location of the read cell; for example due to wire resistances that depend upon where the read cell is located with respect to the rest of the cells. A read operation of cell (i, j) using configuration $T_{i,j}$ returns a bit deciding the value of the cell. For the entire array we define these decisions through the binary-output *read function*

$$D(i, j, A, T_{i,j}).$$

The function $D(i, j, A, T_{i,j})$ returns the decision on cell (i, j) based on electrical measurements of array A performed under configuration $T_{i,j}$. This definition of the read function assumes that decision is made independently on each read cell. In real memory usage it may be the case that information from previous measurements of other cells will also be used to decide on a cell value. In such cases we may add this prior information as additional arguments to $D(\cdot)$. Given this notation, we may now define the problem of low-power error-free readout as the following optimization problem.

Problem 2.

$$\text{find } \{T_{i,j}\}_{i=0,j=0}^{m-1,n-1} \text{ and a maximal size code } \mathcal{C}, \text{ s.t}$$

$$\forall A \in \mathcal{C}, \forall i, j,$$

$$D(i,j,A,T_{i,j}) = a_{i,j},$$

and with the power constraint

$$W(A,T_{i,j}) \leqslant W_{\max},$$

where $W_{\max}$ is an upper bound on the measurement power consumption.

In the special case studied in this paper we defined the read function $D(i,j,A,T_{i,j})$ to be 1 if and only if either $a_{i,j} = 1$ or there exists a sneak-path contained in the rows that are not grounded in $T_{i,j}$. In Sections IV-A,IV-B we chose to leave the nearest rows to i ungrounded, which implies that the function $W(A,T_{i,j})$ has the property that the read power consumed by grounded rows in $T_{i,j}$ decays with the distance $|i'-i|$ between the read row i and a grounded row i'. This behavior of $W(\cdot)$ is justified by wire resistances that alleviate the power consumption from remote grounded rows.

Beyond the models considered in this paper, there are many directions toward solving Problem 2 in realistic setups. For example, another typical behavior of $W(\cdot)$ is that the consumed read power depends on the density of 1 values in the array. So designing sneak-path free codes and read configurations with low density of 1 values is an important future work, for which prior work on memristor coding [11] can be a useful tool. Beyond that, there are many other physical properties of crossbar readout that can be accommodated into appropriate functions $D(\cdot)$ and $W(\cdot)$, and then used to solve Problem 2 in increasingly practical scenarios.

V. Conclusion and Discussion

We proposed in this paper two approaches to deal with sneak-path read errors: 1) to regard sneak-path interference as a random error source and use error-correcting codes to correct the errors, and 2) to enforce constraints that completely eliminate sneak paths. For the second approach we have proposed explicit coding schemes and showed their optimality. For the first approach we provided analytical tools to evaluate the sneak-path error incidence, but left the error-correcting code design open for future work. A central issue to handle in code design is the dependence between errors in different array locations. A potential solution is to perform error-correction coding *across arrays* (i.e., each bit in a codeword is assigned to a different array). This coding structure is possible when multiple 2-dimensional arrays are stacked as layers of a 3-dimensional architecture, and then codewords can span multiple layers and see i.i.d. errors within a code block.

The research in the memristor field is only in its beginning. The focus of this work is on sneak paths as error sources and it can be further extended. Possible extensions include both constructing codes for the models proposed in this paper, and extending the problem and proposed methods to additional sneak-path models motivated by real memristor devices. The common research directions to reduce sneak paths are adding non-linearity to the devices by fabricating additional thin-film layers that change the memristor behavior. This method may generate new sneak path constraints and additional interesting possible problems. Memristive crossbar memories have several fundamental differences from conventional technologies. Specifically, the fact that data is represented by resistance (rather than charge in conventional semiconductor memory technologies) and the two-terminal structure of the storing device that prevents disconnecting of unselected cells, raise many additional interesting problems that can be explored by information theory techniques. For example, during a write operation unselected devices can be unintentionally written depending on the stored data. Exploring the properties of memristive memories will lead to new coding schemes and innovative solutions.

VI. Acknowledgments

The authors thank Ron M. Roth for pointing reference [14] to their attention, and also the associate editor and anonymous reviewers for detailed suggestions that significantly improved the results and their presentation.

References

[1] A. Chen, “Accessibility of nano-crossbar arrays of resistive switching devices,” in *11th IEEE International Conference on Nanotechnology*, August 2011.

[2] T.M. Cover, “Enumerative source encoding,” *IEEE Trans. Inf. Theory*, vol. 19, no. 1, pp. 73–77, January 1973.

[3] S.W. Golomb, “The limiting behavior of the Z-channel,” *IEEE Trans. Inf. Theory*, vol.26, no.3, pp.372,372, May 1980.

[4] K.S. Immink. *Coding techniques for digital recorders*. Prentice-Hall, College Div., 1991.

[5] C.-M. Jung, J.-M. Choi, and K.-S. Min, “Two-step write scheme for reducing sneak-path leakage in complementary memristor array,” *Nanotechnology, IEEE Transactions on*, vol. 11, pp. 611–618, May 2012.

[6] S. Kannan, J. Rajendran, R. Karri, and O. Sinanoglu, “Sneak-path testing of memristor-based memories,” in *12th International Conference on VLSI Design and Embedded Systems (VLSID),*, pp. 386–391, Jan 2013.

[7] K.-W. Kim, S. Gaba, D. Wheeler, J. Cruz-Albrecht, H. Tahir, N. Srinivasa, and W. Lu, “A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications,” *Nano Letters*, vol. 12, no. 1, pp. 389–395, 2012.

[8] D.E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, Addison-Wesley, 1998.

[9] J. Liang and H.-S. Wong, “Cross-point memory array without cell selectors – device characteristics and data storage pattern dependencies,” *Electron Devices, IEEE Transactions on*, vol. 57, pp. 2531–2538, Oct 2010.

[10] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, “Complementary resistive switches for passive nanocrossbar memories,” *Nature Materials*, vol. 9, pp. 403–406, April 2010.

[11] E. Ordentlich and R.M. Roth. Low complexity two-dimensional weight-constrained codes. *IEEE Transactions on Information Theory*, 58(6):3892–3899, 2012.

[12] J. Shin, I. Kim, K. Biju, M. Jo, J. Park, J. Lee, S. Jung, W. Lee, S. Kim, S. Park, and H. Hwang, “Tio2-based metal-insulator-metal selection device for bipolar resistive random access memory cross-point application,” *Journal of Applied Physics*, vol. 109, no. 3, 2011.

[13] S. Shin, K. Kim, and S. Kang. Analysis of passive memristive devices array: data-dependent statistical model and self-adaptable sense resistance for RRAMs. *Proceedings of the IEEE*, 100(6):2021–2032, 2012.

[14] P.P. Sotiriadis, "Information capacity of nanowire crossbar switching networks," *IEEE Transactions on Information Theory*, vol. 52, no. 7, pp. 3019–3032, July 2006.

[15] D. Strukov, G. Snider, D. Stewart, and R.S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008.

[16] J. van Lint and R. Wilson, *A Course in Combinatorics, second edition*. Cambridge UK: Cambridge University Press, 2001.

[17] P.O. Vontobel, W. Robinett, P.J. Kuekes, D.R. Stewart, J. Straznicky, and R.S. Williams, "Writing to and reading from a nano-scale crossbar memory based on memristors," *Nanotechnology*, vol. 20, October 2009.

[18] J. Yang, M.-X. Zhang, M. Pickett, F. Miao, J. Strachan, W.-D. Li, W. Yi, D. Ohlberg, B. Choi, W. Wu, J. Nickel, G. Medeiros-Ribeiro, and R.S. Williams, "Engineering nonlinearity into memristors for passive crossbar applications," *Applied Physics Letters*, vol. 100, no. 11, 2012.

[19] M. Zidan, H. H. Fahmy, M. Hussain, and K. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectronics Journal*, vol. 44, no. 2, pp. 176 – 183, 2013.

Yuval Cassuto (S'02-M'08-SM'14) is a faculty member at the Andrew and Erna Viterbi Department of Electrical Engineering, Technion – Israel Institute of Technology. His research interests lie at the intersection of the theoretical information sciences and the engineering of practical computing and storage systems.

During 2010-2011 he has been a Scientist at EPFL, the Swiss Federal Institute of Technology in Lausanne. From 2008 to 2010 he was a Research Staff Member at Hitachi Global Storage Technologies, San Jose Research Center. From 2000 to 2002, he was with Qualcomm, Israel R&D Center, where he worked on modeling, design and analysis in wireless communications.

He received the B.Sc degree in Electrical Engineering, summa cum laude, from the Technion, Israel Institute of Technology, in 2001, and the MS and Ph.D degrees in Electrical Engineering from the California Institute of Technology, in 2004 and 2008, respectively.

Dr. Cassuto has won the 2010 Best Student Paper Award in data storage from the IEEE Communications Society, as well as the 2001 Texas Instruments DSP and Analog Challenge $100,000 prize.

Shahar Kvatinsky is an assistant professor at the Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion Israel Institute of Technology. He received the B.Sc. degree in computer engineering and applied physics and an MBA degree in 2009 and 2010, respectively, both from the Hebrew University of Jerusalem, and the Ph.D. degree in electrical engineering from the Technion Israel Institute of Technology in 2014. From 2006 to 2009 he was with Intel as a circuit designer and was a post-doctoral research fellow at Stanford University from 2014 to 2015. Kvatinsky is an editor in Microelectronics Journal and has been the recipient of the 2015 IEEE Guillemin-Cauer Best Paper Award, 2015 Best Paper of Computer Architecture Letters, Viterbi Fellowship, Jacobs Fellowship, the 2014 Hershel Rich Technion Innovation Award, 2013 Sanford Kaplan Prize for Creative Management in High Tech, 2010 Benin prize, and six Technion excellence teaching awards. His current research is focused on circuits and architectures with emerging memory technologies and design of energy efficient architectures.

Eitan Yaakobi (S'07-M'12) is an Assistant Professor at the Computer Science Department at the Technion — Israel Institute of Technology. He received the B.A. degrees in computer science and mathematics, and the M.Sc. degree in computer science from the Technion — Israel Institute of Technology, Haifa, Israel, in 2005 and 2007, respectively, and the Ph.D. degree in electrical engineering from the University of California, San Diego, in 2011. Between 2011-2013, he was a postdoctoral researcher in the department of Electrical Engineering at the California Institute of Technology. His research interests include information and coding theory with applications to non-volatile memories, associative memories, data storage and retrieval, and voting theory. He received the Marconi Society Young Scholar in 2009 and the Intel Ph.D. Fellowship in 2010-2011.