

# Resistor-Logic Demultiplexers for Nanoelectronics Based on Constant-Weight Codes

Philip J. Kuekes, Warren Robinett, Ron M. Roth, Gadiel Seroussi,  
Gregory S. Snider, and R. Stanley Williams\*

*Abstract*— The voltage margin of a resistor-logic demultiplexer can be improved significantly by basing its connection pattern on a constant-weight code. Each distinct code determines a unique demultiplexer, and therefore a large family of circuits is defined. We consider using these demultiplexers for building nano-scale crossbar memories, and determine the voltage margin of the memory system based on a particular code. We determine a purely code-theoretic criterion for selecting codes that will yield memories with large voltage margins, which is to minimize the ratio of the maximum to the minimum Hamming distance between distinct codewords. For the specific example of a 64×64 crossbar, we discuss what codes provide *optimal* performance for a memory.

*Indexing Terms*— demultiplexing, error correction coding, nanotechnology, resistive circuits, Hamming distance.

---

\* Manuscript received Sept. 20, 2005. This research was supported in part by the Defense Advanced Research Projects Agency.

P. J. Kuekes is with Hewlett-Packard Laboratories, Palo Alto, California 94304 USA (email: [philip.kuekes@hp.com](mailto:philip.kuekes@hp.com))

W. Robinett is with Hewlett-Packard Laboratories, Palo Alto, California 94304 USA (email: [warren.robinett@hp.com](mailto:warren.robinett@hp.com))

R. M. Roth is with the Computer Science Department, Technion, Haifa 32000, Israel. This work was done while visiting Hewlett-Packard Laboratories, Palo Alto, California 94304, USA (email: [ronny@cs.technion.ac.il](mailto:ronny@cs.technion.ac.il))

G. Seroussi is with the Mathematical Sciences Research Institute, Berkeley, CA 94720, USA. This work was done while the author was with Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, USA. (email: [gadiel@msri.org](mailto:gadiel@msri.org))

G. S. Snider is with Hewlett-Packard Laboratories, Palo Alto, California 94304 USA (email: [snider.greg@hp.com](mailto:snider.greg@hp.com))

R. S. Williams is with Hewlett-Packard Laboratories, Palo Alto, California 94304 USA (phone: 650-857-6586; fax: 650-852-8359; email: [stan.williams@hp.com](mailto:stan.williams@hp.com))

## I. INTRODUCTION

A nanoelectronic *demultiplexer* (demux) circuit [1] can be laid out on a *crossbar* [2][3] using configurable resistors [4] or configurable diodes [1] at the crosspoint junctions. In both cases, error-correcting codes can be used to improve the circuit performance. In comparison with diode-based circuits, demux circuits based on resistors have an inherent problem in achieving an adequate voltage margin to distinguish between their activated and non-activated output lines. On the other hand, resistor-based demux circuits have better defect-tolerance properties, in some respects, than diode-based demuxes. This defect tolerance arises from the redundancy (redundant signal paths) introduced by the coding (not from "error correction" in the traditional sense, since there is in fact no decoder in the circuit to perform correction). These defect-tolerance properties of resistor-based demuxes are described elsewhere [5]; in this paper, we focus on an improved means of handling the resistor-demux's primary problem: achieving an adequate voltage margin. Further motivation for choosing resistor-based demuxes is that nano-scale crossbars containing resistors (rather than diodes) are far more feasible to manufacture with current processes. In [6], it was shown that error-correcting codes of a certain type can be employed to construct resistor-based demuxes with superior analog circuit properties – specifically, good voltage margins. In this paper, we improve on the performance of the circuit construction in [6] by extending it to a broader class of codes – the constant-weight codes (also called "*N*-hot codes" or "*m*-of-*n* codes").

Constant-weight codes have been previously presented in a nano-scale demux design [7] (called a "decoder" in that paper), using a specific code with weight  $w=2$  and size  $M=6$ . In this paper, we expand the analysis of constant-weight codes. We (a) present the entire family of demux circuit designs based on constant-weight codes; (b) present a quantitative model for the voltages occurring on each of the output lines of a resistor-logic demux based on an arbitrary constant-weight code; (c) analyze the use of these demuxes in a nano-scale crossbar memory to develop a performance measure; and (d) characterize the best possible codes for 1-of-64 demuxes.

We briefly review the terminology of *coding theory* [8][9] relevant to this paper. The Hamming weight  $w = wt(\mathbf{b}_1)$  of a bitstring (or bit-vector or word)  $\mathbf{b}_1$  is the number of ones in the bitstring. For equal-length bitstrings, the operations  $\mathbf{b}_1$  AND  $\mathbf{b}_2$  and  $\mathbf{b}_1$  XOR  $\mathbf{b}_2$  are performed bit-wise, producing bitstrings of the same length as the input bitstrings. The Hamming distance between two equal-length bitstrings  $\mathbf{b}_1$  and  $\mathbf{b}_2$  is  $d = dist(\mathbf{b}_1, \mathbf{b}_2) = wt(\mathbf{b}_1 \text{ XOR } \mathbf{b}_2)$ , which is the number of bit-positions in which the two bitstrings differ. A binary *code* of size  $M$  and length  $n$  is a set consisting of  $M$  length- $n$  bitstrings (or *codewords*) that we use as nanowire addresses in our demux circuits. The *minimum Hamming distance* between any pair of distinct codewords is denoted by  $d_{min}$ . Any code may be described using these parameters as an  $(n, M, d_{min})$  code. The *maximum Hamming distance* between any pair of codewords is denoted by  $d_{max}$ . The *dimension* of a code is defined as  $\log_2 M$ . An important type of code is the binary *linear code*, which consists of a set of  $M = 2^k$  length- $n$  codewords that is closed under component-wise addition modulo 2, and thus forms a linear vector space over the integers modulo 2. A binary linear code is referred to as an  $[n, k, d_{min}]$  code, or alternatively as an  $(n, 2^k, d_{min})$  code. Another important type of code is the *constant-weight code*, in which all codewords have the same weight  $w$ ; such a code is described as an  $(n, M, d_{min}, w)$  code. A subtype of the constant-weight code is the *balanced*

code, in which each codeword is "balanced" by having an equal number of ones and zeroes, and which therefore has the property that  $w = n/2$ .

## II. DEMUX CIRCUIT

The fundamental design pattern of these code-based demuxes is that each codeword defines the connection pattern for one nanowire in the array of output lines. The code has  $M$  codewords and the demux circuit has  $M$  output lines. Fig. 1 shows the circuit diagram for an example demux based on a particular  $M=4$  code.

Fig. 1. Demux circuit for a simple 1-of-4 demux based on a (6,4,3) code. (a) The analog circuit realizing the demux, including the layout onto the crossbar. The thick vertical lines represent CMOS-scale wires (conventional wires), and the thin horizontal lines represent nanowires. The crossing nanowires and conventional wires constitute a mixed-scale crossbar, and resistors are shown at the *configured junctions*. (b) The digital circuit implemented by (a), with specific groups of wires identified as named signal vectors. The  $k=2$  bit input address is  $\mathbf{a}$ . The output of the encoder is the  $n=6$  bit codeword  $\mathbf{u}$ . All the nanowire output lines of the demux collectively form the  $M=4$  bit output vector  $\mathbf{s}$ .

The encoder circuit and configuration pattern must match one another, since they are determined by the same code. The code can be read directly from the configuration pattern of the crossbar, one codeword per output line – in Fig. 1, the code used is  $\{000111, 011100, 101010, 110001\}$ , and each output line is labeled with its encoded address  $\mathbf{h}$ . The encoding function  $E$ , implemented by the encoder circuit, must produce codewords  $\mathbf{u}$  matching those defined in the configuration pattern. Thus the specific configuration pattern shown for the crossbar in Fig. 1a implies a particular code which must be used in the encoder.

The design pattern illustrated above may be stated formally as follows. First, the length  $k$  of address vector  $\mathbf{a}$  must be a whole number of bits long,

$$k = \text{ceiling}(\log_2 M), \quad (1)$$

where  $\text{ceiling}(r)$  is the smallest integer  $i$  with  $i \geq r$ . The encoding function  $E$  takes the length- $k$  vector  $\mathbf{a}$  as input, and produces a length- $n$  codeword  $\mathbf{u}$  as output.

$$\mathbf{u} = E(\mathbf{a}) \quad (2)$$

We enumerate the demux's output lines using a  $k$ -bit index vector  $\mathbf{i}$  (the unencoded address of each output line) and we may then express the encoded address  $\mathbf{h}$  of each output line as

$$\mathbf{h} = E(\mathbf{i}) \quad (3)$$

Since in general  $M \neq 2^k$ , valid addresses  $\mathbf{a}$  fall in the range  $[0, M-1]$ , with  $\mathbf{a}$  interpreted as a  $k$ -bit integer. Likewise, valid values of  $\mathbf{i}$  fall into the same range. We will assume the values of  $\mathbf{a}$  and  $\mathbf{i}$  are valid in the rest of the paper.

For an arbitrary output line  $S_i$  associated with codeword  $\mathbf{h} = E(\mathbf{i})$ , each bit of the  $n$ -bit vector  $\mathbf{h}$  specifies whether a connection is configured at each of the  $n$  junctions where the  $n$  CMOS address lines cross this particular nanowire (See Fig. 1a). A "1" specifies a configured resistor at the junction, and a "0" specifies no connection.

In Fig. 1, for example, the top output line  $S_{00}$  has index  $\mathbf{i}=00$  and encoded address  $\mathbf{h}=000111$ . It is useful to label each nanowire output line with an encoded address  $\mathbf{h}$ , since the operation of the AND gate on an output line  $S_i$  may be thought of as recognizing a particular encoded address  $\mathbf{h} = E(\mathbf{i})$  when it appears on the CMOS address lines (as signal vector  $\mathbf{u} = E(\mathbf{a})$ ). Output line  $S_i$  thus turns on when it recognizes the condition  $\mathbf{h} = \mathbf{u}$ . This condition occurs if and only if  $\mathbf{i} = \mathbf{a}$ , showing that each input address  $\mathbf{a}$  causes exactly one output line ( $S_a$ ) to turn on.

#### A. Demux Output Voltages: Demux Based on General Code

We consider a single arbitrary demux output line, similar to those shown in Fig. 1, in a demux based on an arbitrary code. We assume that all the resistors in the crossbar are linear and have the same resistance values  $R$ . The current input address signal  $\mathbf{a}$  produces an output vector  $\mathbf{u} = E(\mathbf{a})$  from the encoder, which drives all the CMOS wires in the crossbar. However, for a particular nanowire output line, not all of the junctions on that nanowire are configured, and so only a subset of these address bits affects the output voltage of the nanowire. The voltage on an arbitrary output line may be calculated by counting up the number of ones ( $n_{ones}$ ) and zeroes ( $n_{zeroes}$ ) driving this output line's *configured* resistors, and analyzing the circuit as a voltage divider as shown in Fig. 2.

Fig. 2. The voltage divider formed by the input signals on the configured resistors for a single arbitrary demux output line in a demux based on a (6,4,4,3) code (Fig. 1). The ones in  $\mathbf{h}$  specify which resistors are connected; this subset of the signals in  $\mathbf{u}$  forms a voltage divider. Within that subset, the ratio of ones to total connections determines the output voltage. The resistors connected to the "logic 1" voltage form the *upper parallel bundle*, and the resistors connected to "logic 0" form the *lower parallel bundle*.

We assume normalized driving voltages in which a "zero" bit in the address is at GROUND=0 V, and a "one" bit is at  $V_{DD}=1$  V, with the output voltage measured with respect to GROUND. The normalized output voltage  $v$  of the voltage divider may be calculated as a ratio of the equivalent resistance of the lower parallel bundle of resistors  $R_{lower}$  to the total resistance of both parallel bundles  $R_{lower} + R_{upper}$ . This simplifies to a ratio of the number of ones to the total number of connections (which is equal to the total number of ones and zeroes).

$$v = \frac{R_{lower}}{R_{lower} + R_{upper}} = \frac{R/n_{zeroes}}{R/n_{zeroes} + R/n_{ones}} = \frac{n_{ones}}{n_{ones} + n_{zeroes}} \quad (4)$$

We consider the output line  $S_i$  labeled by encoded address  $\mathbf{h} = E(\mathbf{i})$ . The total number of configured resistors on this output line ( $n_{ones} + n_{zeroes}$ ) is the number of ones in  $\mathbf{h}$ , that is,  $wt(\mathbf{h})$ .

Letting  $\mathbf{u} = E(\mathbf{a})$  be the codeword driving the CMOS address lines, the ones in  $\mathbf{u}$  represents ones on the CMOS lines, and ones in  $\mathbf{h}$  represent configured resistors. Therefore, the *intersection*  $\mathbf{u} \text{ AND } \mathbf{h}$  gives the ones driving configured resistors. The number of ones among the configured connections of output line  $S_i$  is therefore  $n_{ones} = wt(\mathbf{u} \text{ AND } \mathbf{h})$ . Substituting into (4) yields

$$v_h = \frac{wt(\mathbf{u} \text{ AND } \mathbf{h})}{wt(\mathbf{h})} \quad (5)$$

This quantitatively characterizes the normalized voltage  $v_h$  of all output lines of a demux based on an arbitrary code (in particular, also a constant-weight code), for any valid addresses  $\mathbf{a}$  that may occur as input signals to the demux. By (2) and (3),  $\mathbf{u}$  and  $\mathbf{h}$  are codewords of the code. Equation (5) is valid for any code whatsoever, and is not limited to constant-weight codes.

### B. Demux Output Voltages: Demux Based on Constant-Weight Code

We now derive a version of (5) specialized to constant-weight codes. We denote the complement of a codeword  $\mathbf{c}$  by  $\bar{\mathbf{c}}$ . For any two codewords  $\mathbf{u}$  and  $\mathbf{h}$  of a constant-weight  $(n, M, d_{min}, w)$  code, considered together as shown in Fig. 3, we partition the 2-element columns into four subsets, based on the four possible values of the corresponding bit-pairs; thus the weights of these subsets are  $w_{11} = wt(\mathbf{u} \text{ AND } \mathbf{h})$ ,  $w_{10} = wt(\mathbf{u} \text{ AND } \bar{\mathbf{h}})$ ,  $w_{01} = wt(\bar{\mathbf{u}} \text{ AND } \mathbf{h})$ , and  $w_{00} = wt(\bar{\mathbf{u}} \text{ AND } \bar{\mathbf{h}})$ .

Fig. 3. The relationship between distance and intersection of codewords of constant-weight codes, using the (6,4,4,3) code from Fig. 1 as an example. Since we are concerned only with weights, we chose codewords  $\mathbf{h}$  and  $\mathbf{u}$  in which the columns corresponding to  $w_{00}$ ,  $w_{01}$ ,  $w_{10}$ , and  $w_{11}$  happened to form contiguous blocks, so that the width of the blocks graphically indicate the weights. (In general, the bits of these four groups would be scattered throughout the codewords.) The same values of  $\mathbf{h}$  and  $\mathbf{u}$  from Fig. 2 are used here.

Since this is a constant-weight code,  $w = wt(\mathbf{h}) = w_{10} + w_{11}$  and  $w = wt(\mathbf{u}) = w_{01} + w_{11}$ , which implies  $w_{10} = w_{01}$ . The distance between  $\mathbf{u}$  and  $\mathbf{h}$  is thus

$$d = dist(\mathbf{u}, \mathbf{h}) = wt(\mathbf{u} \text{ XOR } \mathbf{h}) = w_{10} + w_{01} \quad (6)$$

and so  $w_{10} = w_{01} = \frac{d}{2}$ . We denote the weight of the intersection as  $a = wt(\mathbf{u} \text{ AND } \mathbf{h}) = w_{11}$ .

Therefore, for constant-weight codes with weight  $w$ , and any pair of codewords  $\mathbf{u}$  and  $\mathbf{h}$ , it is always true that

$$w = a + \frac{d}{2} . \quad (7)$$

The distance  $d$  between codewords in a constant-weight code is always even, and so  $\frac{d}{2}$  is always an integer. For a constant-weight code with weight  $w$ , we may combine (5), (6) and (7) to get

$$v_{\mathbf{h}} = \frac{a}{w} = \frac{w - \frac{d}{2}}{w} = 1 - \frac{d}{2w} \quad \text{with} \quad d = \text{dist}(\mathbf{u}, \mathbf{h}) . \quad (8)$$

This characterizes the normalized voltage at any demux output line (labeled with its encoded address  $\mathbf{h}$ ), for any input signal (represented by the encoded input address  $\mathbf{u}$ ), valid for a demux based on any constant-weight code with weight  $w$ . The distance  $d$  between codewords of a constant-weight code must fall in the interval  $[0, 2w]$ , and therefore the normalized output voltage remains in the unit interval:  $v_{\mathbf{h}} \in [0, 1]$ .

We denote the *distance profile* of a code (the distinct distances that occur between pairs of codewords, sorted into ascending order) as integer vector  $\mathbf{d}$ , and the (normalized) distinct voltages that appear on the output lines of the demux as real vector  $\mathbf{v}$ . From (8) we get

$$\mathbf{v} = \mathbf{1} - \frac{\mathbf{d}}{2w} , \quad (9)$$

where  $\mathbf{1}$  denotes the all-one vector.

The normalized output voltage  $v_{\mathbf{h}}$  in (8) assumes normalized driving voltages  $v_0^{in} = 0$  (for logic-0) and  $v_1^{in} = 1$  (for logic-1) on the CMOS address lines. The demux circuit is assumed to be composed entirely of linear resistors, and for arbitrary address line voltages  $v_0^{in}$  and  $v_1^{in}$ , the (non-normalized) output voltage is

$$v_{\mathbf{h}}^{out} = (v_1^{in} - v_0^{in})v_{\mathbf{h}} + v_0^{in} . \quad (10)$$

Thus, for a demux based on a given code, there remain two degrees of freedom – choosing the logic-1 and logic-0 voltages  $v_1^{in}$  and  $v_0^{in}$ . Since these driving voltages originate in the CMOS circuitry (see Fig. 1a), they can be freely specified by the circuit designer. Combining (9) and (10) gives

$$\mathbf{v}^{out} = (v_1^{in} - v_0^{in}) \left( \mathbf{1} - \frac{\mathbf{d}}{2w} \right) + v_0^{in} \cdot \mathbf{1} . \quad (11)$$

This characterizes the distinct output voltages of the demux in terms of the weight  $w$  and distance profile  $\mathbf{d}$  of the code, and the two driving voltages  $v_1^{in}$  and  $v_0^{in}$  on the CMOS input lines to the demux circuit. The set of distinct output voltages in  $\mathbf{v}^{out}$  includes all the output voltages that occur for any valid input address vector  $\mathbf{a}$ . As an example, Fig. 4 shows how  $\mathbf{d}$ ,  $\mathbf{v}$ , and  $\mathbf{v}^{out}$  are related in a demux constructed from a particular code.

Fig. 4. Distance distribution for an (11,66,4,5) constant-weight code. This shows the relative number of codeword pairs with a given distance between the codewords (vertical axis) vs. the various distances (horizontal axis). The distinct distances between codewords that actually occur define the code's distance profile  $\mathbf{d} = [0 \ 4 \ 6 \ 8]$ . This shows that the largest distance is  $d_{max}=8$ . The minimum distance is  $d_{min}=4$ . This code has weight  $w=5$  and therefore maximum possible distance  $2w=10$ . The normalized vector

of distinct voltages (9) is thus  $\mathbf{v} = [10/10 \quad 6/10 \quad 4/10 \quad 2/10]$ . From (9) and (11), the distinct voltages  $\mathbf{v}^{out}$  that occur on demux output lines are linearly related both to  $\mathbf{d}$  and to  $\mathbf{v}$ . Supplying two arbitrary voltages  $v_0^{in}$  and  $v_1^{in}$  as driving voltages linearly maps the voltage pattern  $\mathbf{v}$  in the unit interval to the voltage pattern  $\mathbf{v}^{out}$  in the voltage interval  $[v_0^{in}, v_1^{in}]$ .

### III. NANO-SCALE CROSSBAR MEMORY APPLICATION

Our application is a pair of demux circuits that drive the rows and columns of a nano-scale crossbar memory array [6]. In this system, the two demuxes are implemented using mixed-scale crossbars (conventional wires cross nanowires) whereas the memory array itself is a pure nano-scale crossbar (Fig. 5a). The crosspoint junctions of the memory array are assumed to be hysteretic resistors [10], which can be dynamically reconfigured by applying certain voltage drops across them. The idea is to realize a memory system by storing bits in the individual junctions of the memory array, with the convention that high resistance represents a zero, and low resistance represents a one. The demuxes are configured once, at manufacturing time, and are assumed to be stable thereafter; their function is to get the proper voltages to the junctions of interest in the memory array. To make such a system work, we must be able, for an arbitrary junction in the memory array, to (a) write a one or (b) write a zero or (c) read the current state of the bit.

The behavior of a hysteretic resistor, which is a two-terminal device, is that it is controlled by the voltage drop that is put across it [10]. Putting large-magnitude voltage drops (either positive or negative) across the device will destroy it; putting a moderate-magnitude voltage drop across it will write either a one or a zero, depending on the polarity; and putting a low-magnitude voltage drop across it will not change its resistance. Specific voltage thresholds for recently-manufactured devices are given in [10]; however, these values vary as the manufacturing process changes.

Because of the nature of the hysteretic resistors, the circuits used to perform READ and WRITE operations have quite different requirements. They must both access the memory array through the row and column demux circuits, but otherwise may be completely independent CMOS circuits, that are alternately enabled, depending on whether reading or writing is currently occurring. In this paper, we focus on the WRITE operation, in which the primary problem is to deliver to the selected junction a voltage drop which is reliably above the write-threshold voltage, and below the destruction voltage (while guaranteeing that all the non-selected junction voltage drops are well below the write threshold). Briefly, the main problem faced by the READ circuit is one of discrimination. It makes sense to use lower-magnitude voltages in reading to make sure that accidental writes do not occur. With a low-magnitude voltage drop across the a hysteretic resistor, it will act as a normal resistor and its resistance can be measured to read the state of the bit stored in it. The problem is to design a circuit that can discriminate the resistance of the addressed junction from the other  $M^2 - 1$  junctions in the memory array. Designing an

effective READ circuit therefore requires entirely different techniques than those needed for designing a WRITE circuit, and READ circuit issues are not considered further in this paper.

In order to analyze the voltages that can be applied to memory junctions by demuxes driving the rows and columns of a crossbar, we make some assumptions:

- the resistance of the nanowires and conventional wires composing the system are negligible compared with the resistances of the configured junctions, and that the non-configured junctions are effectively open circuits.
- the loading of the demuxes by the memory array is negligible, so that the voltages that appear on the demuxes in isolation are very close to the voltages that actually appear when driving the memory array.
- the CMOS driving circuits can be treated as ideal voltage sources, which do not sag with increasing load. This enables the voltage-divider circuit model expressed in (8), in which the voltage on each output line can be calculated independently of all other output lines.

Given these assumptions, the voltage drop across a junction in the memory array is simply the difference between the voltages on the two crossing wires (see Fig. 5a),

$$\Delta v_{junc} = v_{row} - v_{col}. \quad (12)$$

By our assumptions, the voltages on each row or column wire are constant across the entire memory array, and thus the wires convey the demux output voltages (8), without distortion, to either side of the junctions.

Fig. 5. (a) The circuit geometry of a nano-scale crossbar memory system. The row demux activates the selected row, and the column demux activates the selected column, which intersect at the selected junction. However, every demux output line has an output voltage, and so each junction (where a row and column cross) has a voltage drop determined by the voltages on these two crossing wires. (b) Voltage-drop diagram showing the distinct voltages appearing on row wires  $\mathbf{v}_{row}^{out}$  (left) and column wires  $\mathbf{v}_{col}^{out}$  (right). The vertical dimension represents absolute voltage. The filled circles represent the selected line voltages, and the hollow circles non-selected line voltages (See Fig. 4 for an example of the distinct voltages appearing on the output lines of a single demux circuit.). The dotted lines linking a row voltage to a column voltage represent distinct voltage drops that can occur (across junctions in the memory array), with vertical distance representing the magnitude of the voltage drop. The polarity of each voltage drop is shown by whether the row voltage is above or below the column voltage. This diagram shows the magnitude of *all* the distinct voltage drops that can occur across junctions in the memory array. (c) The *splay*  $s$  and *offset voltage*  $t$  between the row voltages and column voltages. These variables use the normalization (14) in which the output-voltage range of each demux has unit width (labeled "1"). This shows how the distribution of distinct demux output voltages and the voltage offset  $t$  between the two demuxes interact to determine the voltage drops that occur.

We write one bit at a time in the memory by using the two demuxes to activate a *selected row* and a *selected column*, which cross at a *specified junction*. The address input signals to the row demux and column demux determine which output line is selected, for each of the two demuxes. However, all the non-selected junctions are still active, and could be accidentally written to or

destroyed if over-threshold voltage drops happen to be unintentionally applied to them. Thus our goal is to set up the demux driving circuits and voltages such that we deliver the intended voltage drop to the selected junction, and deliver voltage drops as small as possible to all the other non-selected junctions in the memory array.

*Design Goal: We wish to minimize the ratio  $q_v$ ,*

$$q_v = \frac{\max\{|\Delta v_{non\_selected}|\}}{|\Delta v_{selected}|}, \quad (13a)$$

*of the largest magnitude of the voltage drops delivered to any non-selected junction with respect to the magnitude of the voltage drop delivered to the selected junction.*

It is natural to define the normalized voltage margin  $v_m$  as

$$v_m = 1 - q_v, \quad (13b)$$

since this is the (normalized) difference between the voltage drop we wish to deliver to the selected junction, and the worst-case voltage drop incidentally delivered to a non-selected junction.

Two versions of (11) are required for the row and column demuxes, with two separate pairs of driving voltages, producing one set of voltages  $\mathbf{v}_{row}^{out}$  driving the rows, and another set of voltages  $\mathbf{v}_{col}^{out}$  driving the columns. We use (12) to combine these absolute voltages to obtain a matrix specifying all voltage drops that can occur in the memory array. We use a graphical technique (Fig. 5b) to visualize the distinct voltage drops that occur.

We must first choose the four driving voltages for the row and column demuxes (logic-1 and logic-0 voltages for the row demux, and logic-1 and logic-0 voltages for the column demux). To keep things simple, it was valuable to use the digital circuit abstraction as much as possible, and to discuss the resistor-logic demux circuit as a demultiplexer (a *bona fide* digital circuit); that is why we used the names "logic-1" and "logic-0" for the two voltages driving the demux input lines. However, at this point, as we combine the outputs of two resistor-logic demux circuits, we leave the digital abstraction. The four voltages are no longer considered to be digital signals, but are four independently-adjustable voltages, which we need to control in order to deliver the desired voltage drops to certain junctions in the memory array. For simplicity, we assume that the row and column demuxes are identical (same code, same number of output and input lines, same encoder circuitry). This restriction is not necessary, since, for example, there may be a reason to implement a rectangular rather than a square crossbar memory.

There are only three degrees of freedom, since adding a constant to all four voltages has no effect on the circuit behavior. We can arbitrarily choose one of these voltages to be GROUND. Because the hysteretic resistors have fixed voltage-drop thresholds governing their behavior, the voltage difference between "logic-1" and "logic-0" for both demuxes is constrained; this consumes two more degrees of freedom. One degree of freedom remains: the voltage offset between the row voltages and the column voltages. The situation is diagrammed in Fig. 5c.

In Fig. 5c, the output voltages of both demuxes are normalized to the output voltage range. This is a different normalization than the one used earlier. It is a code-dependent normalization, which employs knowledge of the code parameter  $d_{\max}$  to scale the demux's two driving voltages

$$v_1^{in} = 1 \text{ and} \quad (14a)$$

$$v_0^{in} = 1 - \frac{2w}{d_{\max}}, \quad (14b)$$

such that the maximum and minimum voltages that occur on the output lines completely fill the unit interval. This can be verified by calculating  $\mathbf{d}$  and  $w$  for a particular code, extracting  $d_{\max}$

from  $\mathbf{d}$ , and substituting (14a) and (14b) into (11), yielding  $\mathbf{v}^{out} = \mathbf{1} - \frac{\mathbf{d}}{d_{\max}}$ . The output line

voltages corresponding to the distances 0,  $d_{\min}$  and  $d_{\max}$  contained in  $\mathbf{d}$  are, respectively, the voltage on the selected line ( $v^{ON}$ ), the highest voltage on a non-selected line ( $v_{\max}^{OFF}$ ), and the lowest voltage on a non-selected line ( $v_{\min}^{OFF}$ ).

$$v^{ON} = 1 - \frac{0}{d_{\max}} = 1. \quad (14c)$$

$$v_{\max}^{OFF} = 1 - \frac{d_{\min}}{d_{\max}}. \quad (14d)$$

$$v_{\min}^{OFF} = 1 - \frac{d_{\max}}{d_{\max}} = 0. \quad (14e)$$

When  $v_1^{in} > v_0^{in}$ , then  $v_{\max}^{OFF} > v_{\min}^{OFF}$ , as the names suggest. However, if  $v_1^{in} < v_0^{in}$ , then the voltages are flipped in polarity. In Fig. 5, the column voltages are flipped in this way. This is necessary to get the maximum voltage drop across the selected junction.

Our goal is to maximize the variable  $t$  in Fig. 5c, which represents the normalized offset voltage between the row and column voltages. The variable  $s$  in Fig. 5c represents the *splay* (how spread out the non-selected output voltages of a demux are – specifically, the relative width of the voltage range occupied by the non-selected output lines); the splay  $s$  is determined by the choice of code, and can be calculated as

$$s = \frac{|v_{\max}^{OFF} - v_{\min}^{OFF}|}{|v^{ON} - v_{\min}^{OFF}|} = 1 - \frac{d_{\min}}{d_{\max}} = 1 - \frac{1}{q_d} \text{ where } q_d = \frac{d_{\max}}{d_{\min}}. \quad (15)$$

The diagram of Fig. 5c provides a visual metaphor. To consider various voltage offsets  $t$  between the row and column voltages, we imagine that there are two solid structures holding the row and column voltages, which may be slid up and down vertically with respect to one another, while the dotted lines (representing voltage drops) are rubber bands that stretch. We slide the two structures back and forth until we have found an offset  $t$  where the ratio  $q_v$ , (13a) is at a minimum. Excluding the selected-junction voltage drop in order to focus on non-selected junctions, we consider the two largest-magnitude voltage drops of each polarity. The dotted line between the lowest row voltage (lowest hollow circle on the left) and the highest column voltage (highest hollow circle on the right) represents the maximum-magnitude voltage drop with

negative polarity (using the polarity convention of (12)). As shown in Fig. 5c, this voltage drop has magnitude  $1-t$ . For the positive polarity, since the selected-junction voltage drop is excluded, we must choose either the voltage drop between the highest row voltage and the second-lowest column voltage, or between the lowest column voltage and the second-highest row voltage. But by the symmetry of the system, these two voltage drops are equal. We arbitrarily choose the latter, which is shown in Fig. 5c as the dotted line between the highest hollow circle on the left, and the bottom filled circle on the right. The magnitude of this voltage drop is  $t+s$ . The quantity  $\max(1-t, t+s)$  is equal to the numerator of (13a). The denominator of (13a) represents the magnitude of the voltage drop across the selected junction, which can be seen in Fig. 5c to be  $1+t$ . We wish to minimize  $q_v = \frac{\max(1-t, t+s)}{1+t} = \max\left(\frac{1-t}{1+t}, \frac{t+s}{1+t}\right)$ . This

minimum occurs when

$$\frac{1-t}{1+t} = \frac{t+s}{1+t}. \quad (16a)$$

Fig. 6 shows why the minimum of  $q_v$  occurs when these two expressions are equal, and that the maximum of the voltage margin  $v_m = 1 - q_v$  also occurs at the same value of  $t$ .

Fig. 6. Plot of  $\frac{1-t}{1+t}$ ,  $\frac{t+s}{1+t}$ ,  $q_v = \max\left(\frac{1-t}{1+t}, \frac{t+s}{1+t}\right)$  and  $v_m = 1 - q_v$  vs. voltage offset  $t$  for the (11,66,4,5) example code, which has  $s = \frac{1}{2}$ . The minimum of  $q_v$  and maximum of the voltage margin  $v_m$  occur when  $t = \frac{1}{4}$ .

We can solve (16a) for  $t$ , the optimal voltage offset between demuxes.

$$t = \frac{1-s}{2} \quad (16b)$$

In the normalization (14) of Fig. 5c, the voltage drop across the selected junction is  $1+t$  and the worst-case voltage drop across a non-selected junction is  $1-t$ . Hence, from (15) and (16b),

$$q_v = \frac{1-t}{1+t} = \frac{2q_d - 1}{2q_d + 1}. \quad (17a)$$

The circuit's normalized voltage margin is therefore

$$v_m = 1 - q_v = \frac{2t}{1+t} = \frac{2}{2q_d + 1}. \quad (17b)$$

The values of  $s$  and  $q_d$  are determined by the code used. Since the function at the right-hand side of (17b) is monotonically decreasing in  $q_d$ , minimizing  $q_d$  minimizes the quantity  $q_v$ , and maximizes the circuit's voltage margin  $v_m$ .

Three different normalizations have been used in the preceding analysis: (a) normalized *input* voltages were used to analyze an individual demux circuit (4)(5)(8)(9); (b) for a demux based on a specific code, knowledge of the code parameter  $d_{\max}$  was used to normalize the demux's *output*

voltage range (14) (the parameters  $s$  and  $t$  use this normalization in Fig. 5c); and (c) the voltage drop across the selected junction is the normalization unit for both  $q_v$  and  $v_m$  (17). This last normalization is dependent on both  $d_{\max}$  and the optimal voltage offset  $t$ . These three normalizations were inconvenient for analyzing the memory circuit; to apply them to a real circuit, a final voltage scaling is needed to make the delivered voltage drop at the selected junction match the write thresholds of the hysteretic resistors in the junctions.

To achieve our design goal of maximizing  $v_m$ , we must minimize  $q_d = \frac{d_{\max}}{d_{\min}}$  in (17b), which is a purely code-theoretic quantity. This gives us a simple criterion for evaluating codes for this crossbar memory application.

$$\text{Code selection criterion for the memory application: minimize } q_d = \frac{d_{\max}}{d_{\min}}. \quad (18)$$

Codes that have low values of  $q_d$  will produce memory systems with low values of  $q_v$ , which will have higher voltage margins  $v_m$  than a code with a higher  $q_v$ .

For a given memory size (or code size)  $M$ , very low  $q_v$  can be achieved if the code length  $n$  is allowed to grow without limit. However, increasing  $n$  incurs a cost in chip area. Therefore there is a trade-off between  $q_v$  and  $n$ , or, in other words, given a specification for the code size  $M$  and the voltage margin  $v_m$ , we shall seek the shortest possible code satisfying the specification.

#### A. Example of Constant-Weight Code Demuxes Used in a Memory System

We will use an (11,64,4,5) code (also referred to in Figures 4, 5 and 6) to illustrate the use of constant-weight codes for constructing demux circuits for memory systems. This code has  $n=11$  and yields a memory system with a  $v_m = 0.4$  (voltage margin of 40%).

Fig. 7 shows the voltage drop diagram for a memory system based on (11,64,4,5) demuxes, with the delivered voltage drop to the selected junction scaled to 2.5 V. We will here assume the threshold voltages of  $v_{\text{destruction}} = \pm 4$  V and  $v_{\text{write}} = \pm 2$  V. Thus 2.5 V exceeds the write-threshold voltage, but is below the destruction threshold voltage.

Fig. 7. The four scaled driving voltages for a nano-scale crossbar memory system based on a (11,64,4,5) constant-weight code. The vertical distances show the voltage drops that can occur at various junctions in the memory array.

The 40% voltage margin of this system is a 1.0 V difference between the 2.5 V delivered to the selected junction, versus the worst-case 1.5 V delivered to any other junction. This margin is allocated in this example so that we exceed the 2.0 V write threshold by 0.5 V for the selected junction, and the selected junction is 1.5 V below the destruction threshold of 4.0 V. For all the non-selected junctions, the worst-case delivered voltage drop is 1.5 V, which is 0.5 V below the write threshold.

The three different normalizations used in the analysis can be seen in Fig. 7. The input voltage range for each demux is 2.5 V. The range of output voltages for each demux is 2.0 V. The value of  $t = \frac{1}{4}$  is calculated as the optimal normalized voltage offset for this code, which is thus an offset of 0.5 V. The voltage drop delivered to the selected junction is 2.5 V. In this example, there are only two distinct voltages among the four driving voltages, but in general the four voltages may be distinct.

#### IV. WHAT ARE THE BEST CONSTANT-WEIGHT CODES FOR 64×64 MEMORIES?

For the nano-scale crossbar memory application, in the foreseeable future, practical circuit issues limit the size  $M$  of the demuxes we might build to the range  $64 \leq M \leq 1024$ , where  $M$  is not required to be a power of 2. As an example, we will choose a specific code size  $M=64$ . Codes of this size determine 1-of-64 demux circuits, which could be used to make 64×64 memory systems. With the number of output lines  $M$  held constant, and the number of CMOS address lines  $n$  allowed to vary, we ask: what are the best constant-weight codes (we can find) for each value of  $n$ ? The area of the mixed-scale crossbar implementing an  $(n, M, d_{min}, w)$  demux is proportional to  $nM$ , so codes with low  $n$  are desirable to minimize chip area requirements.

In contrast to the usual situation in coding theory, we are not concerned with getting high values of  $d_{min}$  – rather it is the relationship between  $d_{min}$  and  $d_{max}$  that is of interest. In particular, we desire low values of  $q_d = \frac{d_{max}}{d_{min}}$ , or equivalently, that  $d_{min}$  and  $d_{max}$  are "near" one another.

In [11], for  $M=64$  constant-weight codes, some lower bounds for  $q_d$  were established for certain ranges of  $n$ . Some specific codes were also given, for which the  $q_d$ -values equal the lower bounds. These are the codes in Table I; the actual codes are explicitly described in the appendix. For codes shown in the table with  $M > 64$ , some codewords can be eliminated to reduce the code size to  $M = 64$  without changing the code's parameter  $q_d = \frac{d_{max}}{d_{min}}$ . The codes in all rows of the table are optimal in  $n$  and  $q_d$ . For example, we consider the (15,70,6,6) code in row 4 of Table I. For  $M=64$  constant-weight codes with  $n \in [15,18]$ , the lowest possible  $q_d$  value for any such code is  $q_d = \frac{5}{3}$ , which is realized by this code. Furthermore, it is an  $n=15$  code, on the bottom end of the  $[15,18]$  range for  $n$  for this value of  $q_d$ . This code is optimal because the lower bound shows that (a) it is impossible to find some other  $n=15$  code with a lower value of  $q_d$ , and (b) there are no shorter codes with an equal or smaller  $q_d$ .

Table I. Table of lower bounds on  $q_d$  for  $M=64$  constant-weight codes, and codes at these bounds.

Range of $n$	Lower Bound for $q_d$	Known $M=64$ Constant-Weight Codes with low $q_d$						
		Code	$d_{\max}$	$d_{\min}$	$q_d$	$n$	$q_v$	$v_m$
8	4	(8,70,2,4)	8	2	4	8	$\frac{7}{9}$	$\frac{2}{9} \cong 0.222$
[9,10]	3	(9,84,2,3)	6	2	3	9	$\frac{5}{7}$	$\frac{2}{7} \cong 0.286$
[11,14]	2	(11,66,4,5)	8	4	2	11	$\frac{3}{5}$	$\frac{2}{5} = 0.400$
[15,18]	$\frac{5}{3}$	(15,70,6,6)	10	6	$\frac{5}{3}$	15	$\frac{7}{13}$	$\frac{6}{13} \cong 0.462$
[19,23]	$\frac{3}{2}$	(19,70,8,9)	12	8	$\frac{3}{2}$	19	$\frac{1}{2}$	$\frac{1}{2} = 0.500$

Previously, a constant-weight code was found through a brute-force search among (constrained) codes [12]. This (22,64, 8, 6) code with  $d_{\max}=12$  has parameters  $n=22$  and  $q_d = \frac{3}{2}$ ; the  $n=19$  code of row 5 of Table I is more efficient (shorter) than this code.

Constant-weight codes appropriate for other demultiplexer sizes (e.g.,  $M = 128$  or  $256$ ) can be obtained similarly to those presented above for  $M = 64$ . The starting point is usually a “good” constant-weight code in the conventional sense of coding theory (e.g., with a large value of  $M$  for given  $n$  and  $d_{\min}$ ), to which coding-theoretic operations of expurgation, shortening, and augmentation [7] are applied to optimize the code for our requirements (i.e., a low  $q_d$  ratio). The mathematical tools from [10] still provide lower bounds on the best achievable value of  $q_d$  given  $M$  and  $n$ . However, as the code size increases, there might be gaps between the best parameters achieved by the constructions, and the bounds of [10], which might leave the question of code optimality open. Nevertheless, from a practical point of view, the codes obtained still provide excellent performance for the application at hand.

In [6], we presented a demux circuit construction that utilized *complementary repeated codes* based on *linear codes*. They are constructed (see Eq. 8 of [6]) by taking a starting linear code and to each codeword appending its complement; which was called “balancing” in [6]. When starting from an  $(n, 2^k, d)$  linear code, this operation produces an  $(2n, 2^k, 2d, n)$  constant-weight code. These are balanced codes, since they have equal numbers of ones and zeroes in each codeword. They are therefore a specialized subclass of the constant-weight codes. However, they cannot be efficient in terms of length  $n$ , because of the doubling of length inherent in their construction, and general constant-weight codes will be shorter. For example, the (11,66,4,5) code from Row 3 of Table I is optimal with  $n=11$  and  $v_m = \frac{2}{5}$ . A directly comparable code from [6] is a (22,64,8,11) complementary repeated code, which has the same voltage margin  $v_m = \frac{2}{5}$ , but with  $n=22$  it is twice as long as the optimal code.

## V. ENCODER AREA

The complexity of the encoder circuit for a general constant-weight code may be greater than that of a complementary repeated code, resulting in a cost of increased chip area. However, there are several reasons why this may be irrelevant for code-based demux circuits. (a) A large number of memory arrays may be serviced by a single encoder, so that the encoder cost is amortized over many memory blocks [1]. (b) Even without amortization, the “good” codes described for this application are often constructed by starting from a good linear code (e.g., the Golay code [8]) and modifying it. Thus, the structure of the original linear code can still be exploited to design encoders that are significantly simpler than a full lookup table. (c) The area of the possibly larger encoder is counterbalanced by the reduced area of the demux's mixed-scale crossbar (half the area, in the above example of the  $n=11$  vs.  $n=22$  codes). Therefore, a larger encoder area is not likely to be a significant disadvantage when using constant-weight codes in designing demux circuits for the nano-scale memory application.

## VI. CONCLUSIONS

We analyzed demuxes based on general constant-weight codes and the entire memory circuit controlled by them. We were able to set lower bounds on the shortest codes (smallest demuxes) that could provide a particular voltage margin in a memory, and then we identified codes that satisfied these bounds for the case of a  $64 \times 64$  crossbar. We made a number of simplifying assumptions in order to obtain a quantitative criterion (18) for evaluating the efficiency of the codes:

- The resistors in the crossbar are linear.
- The configured resistors in the crossbar all have the same value of resistance.
- The non-configured resistors in the crossbar are open (infinite resistance).
- The conventional wires and nanowires of the crossbar have negligible resistance.
- The CMOS voltage sources driving the demuxes do not sag under load.
- The loading by the memory array has a negligible effect on the demux output voltages.
- Two identical demuxes are used to drive the rows and columns.
- The hysteretic resistors in the crossbar all have identical write-thresholds and destruction thresholds, and the magnitudes of these thresholds are equal for both polarities.

Although not all of these assumptions will hold for real nano-scale crossbar memories, the constant-weight codes defined by the criterion of (18) will still be preferable in most such systems because the shorter codes will still be more efficient for memories in which the analog circuit properties deviate from our assumptions. We have therefore disentangled the issue of choosing a code from the complicated analog circuit issues that arise in designing a resistor-based demux.

## ACKNOWLEDGMENT

We gratefully acknowledge J. Straznicky for valuable discussions, and the Defense Advanced Research Projects Agency (DARPA) of the United States for partial support.

APPENDIX: EXPLICIT DESCRIPTION OF THE CODES IN TABLE I

Code 1: (8, 70, 2, 4),  $d_{\max}=8$ . All binary words of length 8 and Hamming weight 4.

Code 2: (9, 84, 2, 3),  $d_{\max}=6$ . All binary words of length 9 and Hamming weight 3.

Code 3: (11, 66, 4, 5),  $d_{\max}=8$ . Constructed from the Steiner system  $S(4,5,11)$  [7, p. 70].

```
{00100011101, 10011011000, 01010010110, 11000001011, 11100110000, 10101000110,
00110101010, 00001110011, 01111000001, 10010100101, 01001101100, 01000111010,
00110110001, 10100101100, 10000010111, 11001100001, 01010001101, 01101010100,
00011100110, 11110000010, 00101001011, 10001110100, 01101100010, 01001011001,
00000101111, 10011000011, 10100011010, 11010101000, 00111001100, 11100000101,
00011101001, 11011000100, 10010110010, 00001011110, 00110000111, 01000110101,
10101010001, 01110011000, 00111010010, 10110001001, 00101100101, 00010111100,
01100001110, 10001101010, 01010100011, 01110100100, 01100010011, 01011001010,
00101111000, 11000011100, 00011010101, 11101001000, 11000100110, 10110010100,
01011110000, 10000111001, 11010010001, 10001001101, 01100101001, 10111100000,
10100100011, 00010011011, 11001010010, 01001000111, 00100110110, 10010001110}
```

Code 4: (15, 70, 6, 6),  $d_{\max}=10$ . Codewords of Hamming weight 6 in the (15, 128, 6) shortened Nordstrom-Robinson code [7, p. 74].

```
{101100000001011, 011010000010101, 011001000011010, 000101000010111, 110011000001001,
101111000000100, 110100100000101, 001100100011100, 000010100011011, 111010100000010,
100001100001110, 101001100010001, 010111100010000, 110000010010011, 111000010001100,
000110010001101, 001110010010010, 100101010011000, 011101010000001, 010011010000110,
001000110000111, 010100110001010, 100010110010100, 001011110001000, 010100001011001,
011100001000110, 100010001000111, 101010001011000, 110001001010100, 001001001001101,
000111001001010, 100100101010010, 010010101001100, 001110101000001, 010001101000011,
000000011011110, 110110011000000, 101001011000010, 000011011010001, 100000111001001,
011000111010000, 000101111000100, 1000000011101, 111100000110000, 001010000101110,
010110000100011, 010101000101100, 100011000110010, 010000100110110, 011000100101001,
100110100101000, 001101100100010, 000011100100101, 100100010100110, 010010010111000,
101010010100001, 000001010101011, 001001010110100, 000100110110001, 110001110100000,
110000001101010, 001000001110011, 000110001110100, 100101001100001, 011011001100000,
101000101100100, 000001101111000, 010000011100101, 001100011101000, 000010111100010}
```

Code 5: (19, 70, 8, 9),  $d_{\max}=12$ . Constructed by shortening, expurgating, and augmenting [7, Ch. 1] the subset of codewords of Hamming weight 12 in the extended binary Golay code [7, p. 64].

```
{0110011001101001001, 1000011100101101010, 0001110111100010100, 1011010011100001010,
0010111011010001100, 1010110011001010001, 1101010000010111010, 0110001101010000111,
0111001000011011100, 1111100101100000001, 1001110001010000111, 1110000110110100010,
0011011000000101111, 0110110010100000111, 1010000010010011111, 0000010111001001111,
0110100110011001001, 0010000111011110100, 1101001001111000010, 0111010001110100100,
0101000010011100111, 1110010010001101100, 110010010000101111, 1000101101110100100,
0100100010110111100, 0100110110001110010, 0110011010010110001, 0000101011000110111,
1011001010001110010, 1001001110100000111, 0001000110111011010, 1100101010101100001,
1001111100000110001, 1001011010011001001, 1011000100100111100, 1101100001001110100,
0101000001100011111, 0011100100001010111, 1011111000101000100, 1110100000101011010,
0001111010110100010, 1000011111010010010, 1001000011110110001, 0100011101000111100,
1101000111010001100, 1011101100010001010, 0111000101001101010, 0100101111100001010,
0000010100110110111, 1100011000001010111, 0100011110111000100, 1010000001101100111,
0010110101111000010, 0010001001110111010, 0000101000111001111, 0000100101101111001,
1000100011011101010, 0111111001000010010, 11101101000100100100, 1110001011100010100,
1000010001111011100, 0001101110001101100, 1100101001010011001, 1100010110100011001,
0101110011000101001, 0011001100111100001, 1100010101011100001, 1010110000110101001,
0101110000111010001, 0011001111000011001}
```

## REFERENCES

- [1] P. J. Kuekes, W. Robinett, G. Seroussi and R. S. Williams, "Defect-tolerant interconnect to nanoelectronic circuits: internally-redundant demultiplexers based on error-correcting codes," *Nanotechnology*, **16**, 869, 2005.
- [2] A. DeHon, "Array-based architecture for FET-based, nanoscale electronics," *IEEE Transactions on Nanotechnology*, vol. 2, pp. 23-32, 2003
- [3] G. Snider, P. J. Kuekes, T. Hogg and R. S. Williams, "Nanoelectronic architectures," *Appl Phys A*, **80**, 1183 2005.
- [4] P. J. Kuekes, R. S. Williams, and J. R. Heath "Molecular wire crossbar memory," *US Patent #6,128,214*, Oct. 3, 2000.
- [5] P. J. Kuekes, W. Robinett, and R. Stanley. Williams, "Defect tolerance in resistor-logic demultiplexers for nanoelectronics," *IEEE Tr. Nanotechnology*, submitted for publication, 2005.
- [6] P. J. Kuekes, W. Robinett, and R. Stanley. Williams, "Improved voltage margins using linear error-correcting codes in resistor-logic demultiplexers for nanoelectronics," *Nanotechnology*, **16**, 1419-32, 2005.
- [7] Z. Zhong, D. Wang, Y. Cui, M. W. Bockrath and C. M. Lieber, "Nanowire crossbar arrays as address decoders for integrated nanosystems," *Science*, vol. 302, pp. 1377-79, 2003
- [8] F. J. MacWilliams and N. J. A. Sloane *The Theory of Error-Correcting Codes*, (North-Holland, New York), 1990.
- [9] S. B. Wicker, *Error Control Systems for Digital Communication and Storage* (Prentice Hall, Upper Saddle River), 1995.
- [10] G. Snider, "Computing with hysteretic resistor crossbars," *Appl Phys A*, **80**, 1165, 2005.
- [11] Ron M. Roth and Gadiel Seroussi, "On the second moment of the distance distribution of binary codes," *Proc. IEEE International Symposium on Information Theory*, Adelaide, Australia, September 2005, pp. 931--935.
- [12] Greg S. Snider and Warren Robinett, "Crossbar demultiplexers for nanoelectronics based on  $n$ -hot codes," *IEEE Tr. Nanotechnology*, vol. 4, no. 2, March 2005.

Fig. 1a.

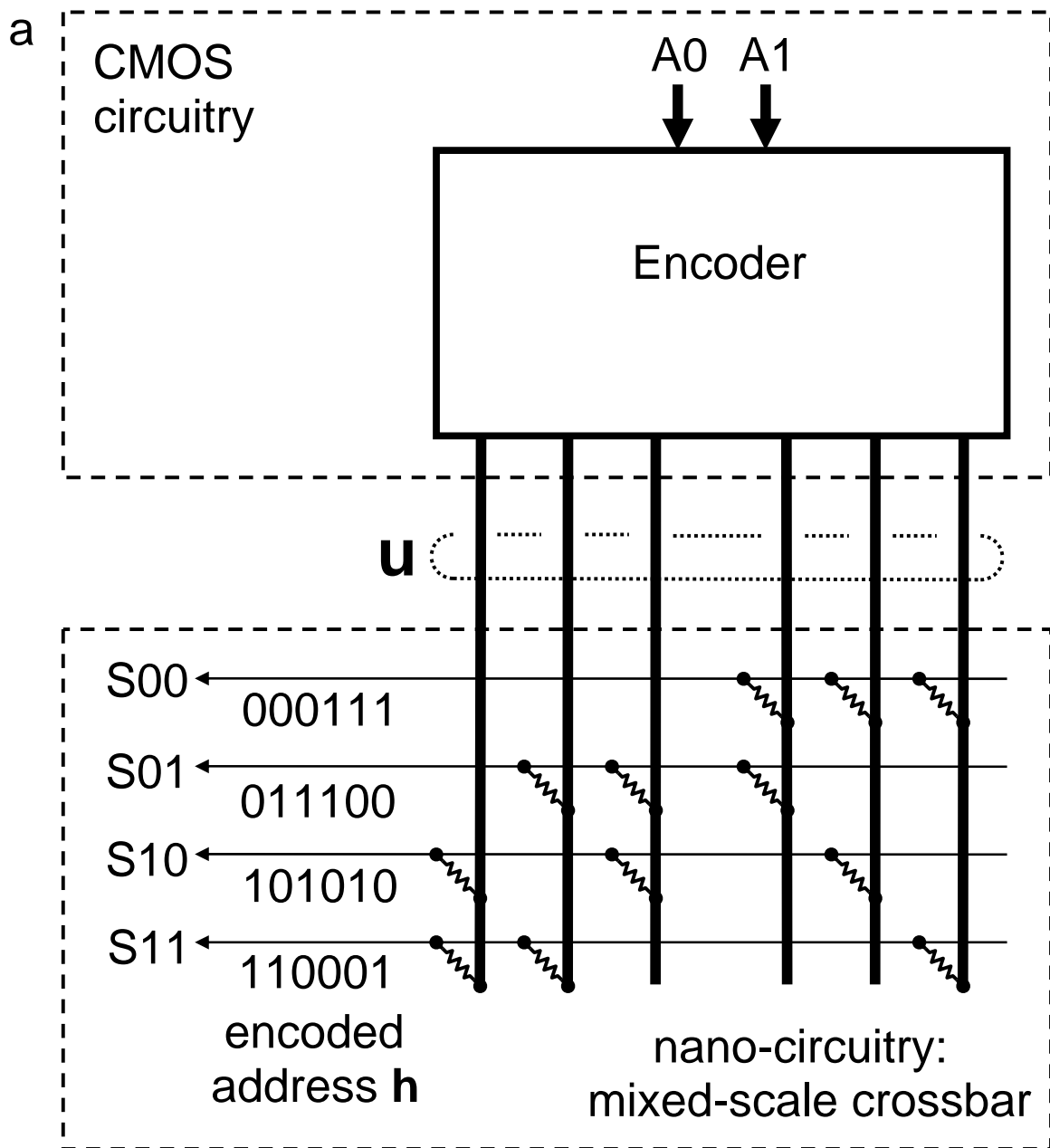


Fig. 1b.

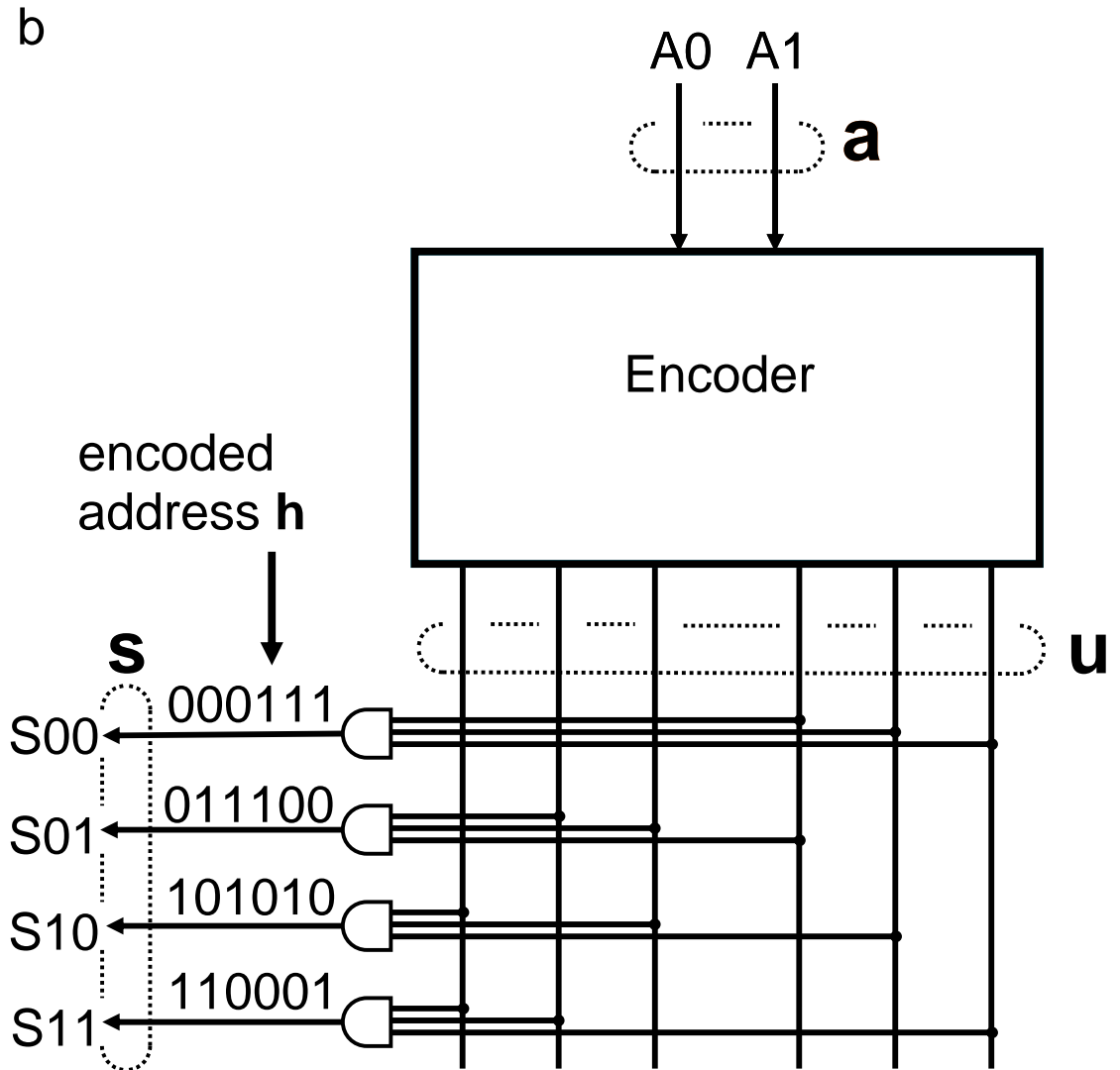


Fig. 2.

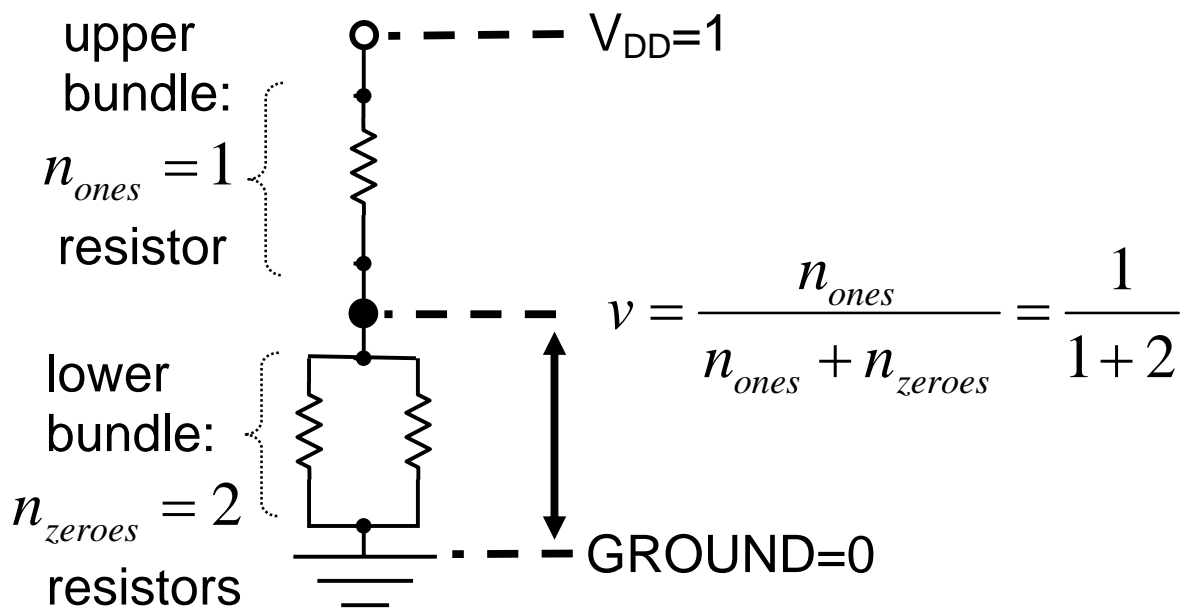
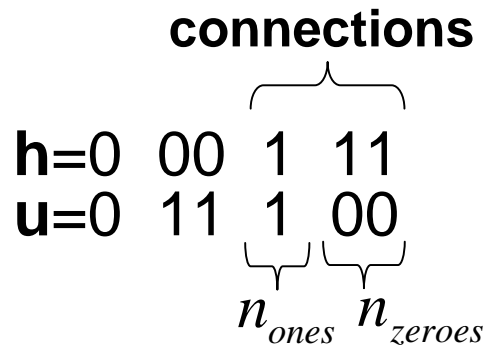


Fig. 3.

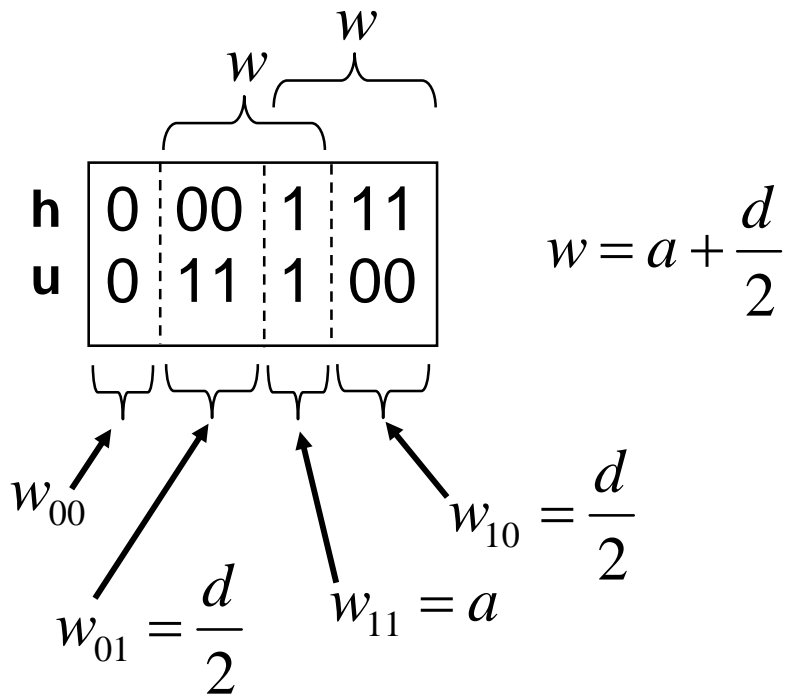


Fig. 4

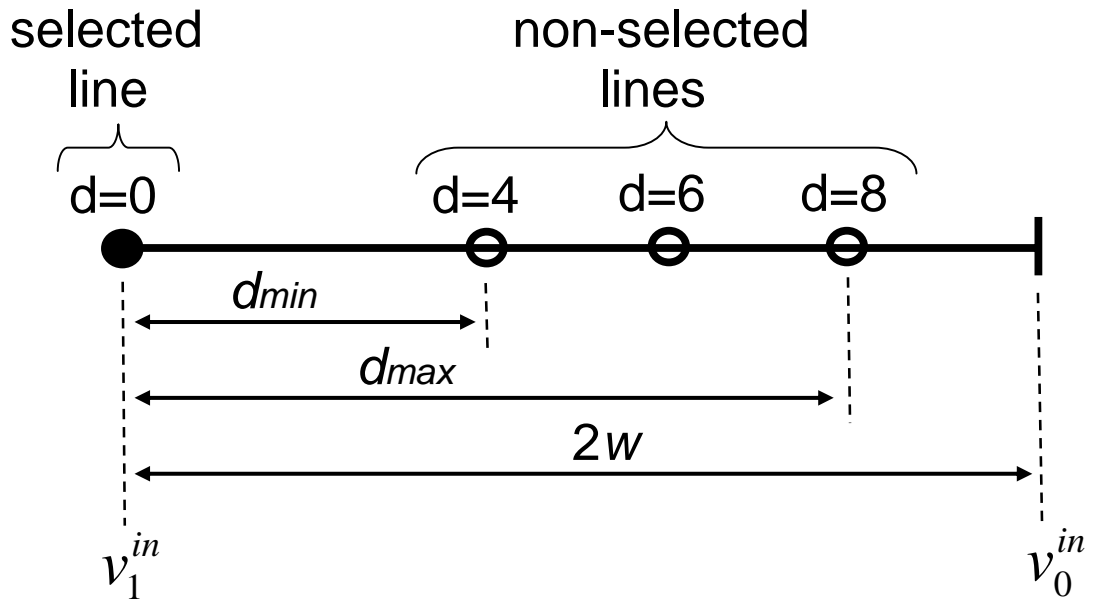
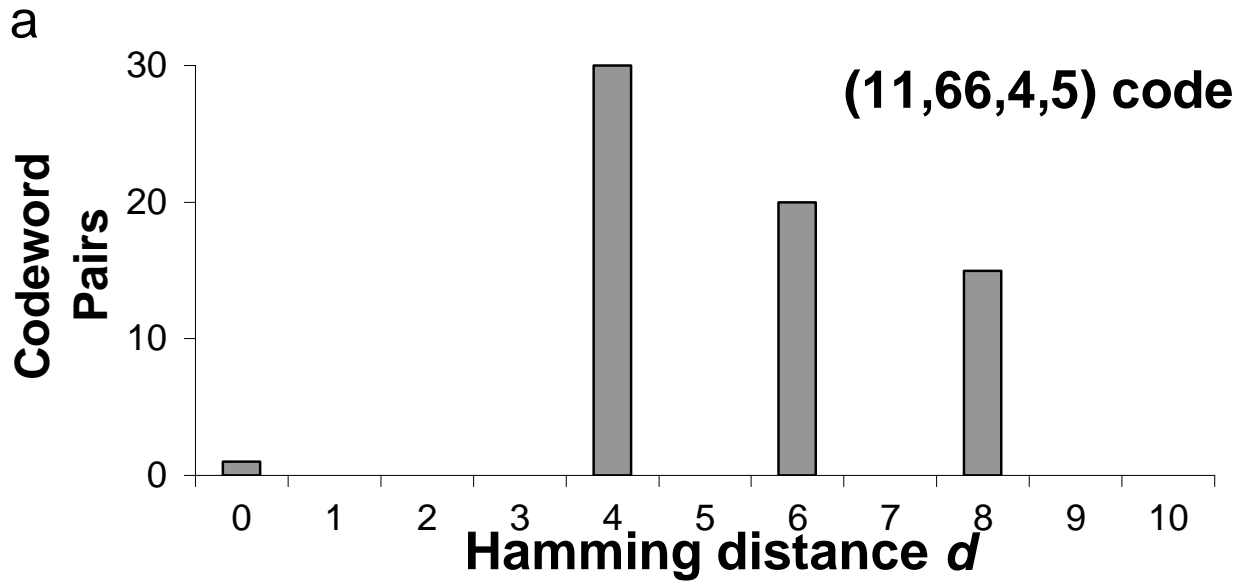


Fig. 5a.

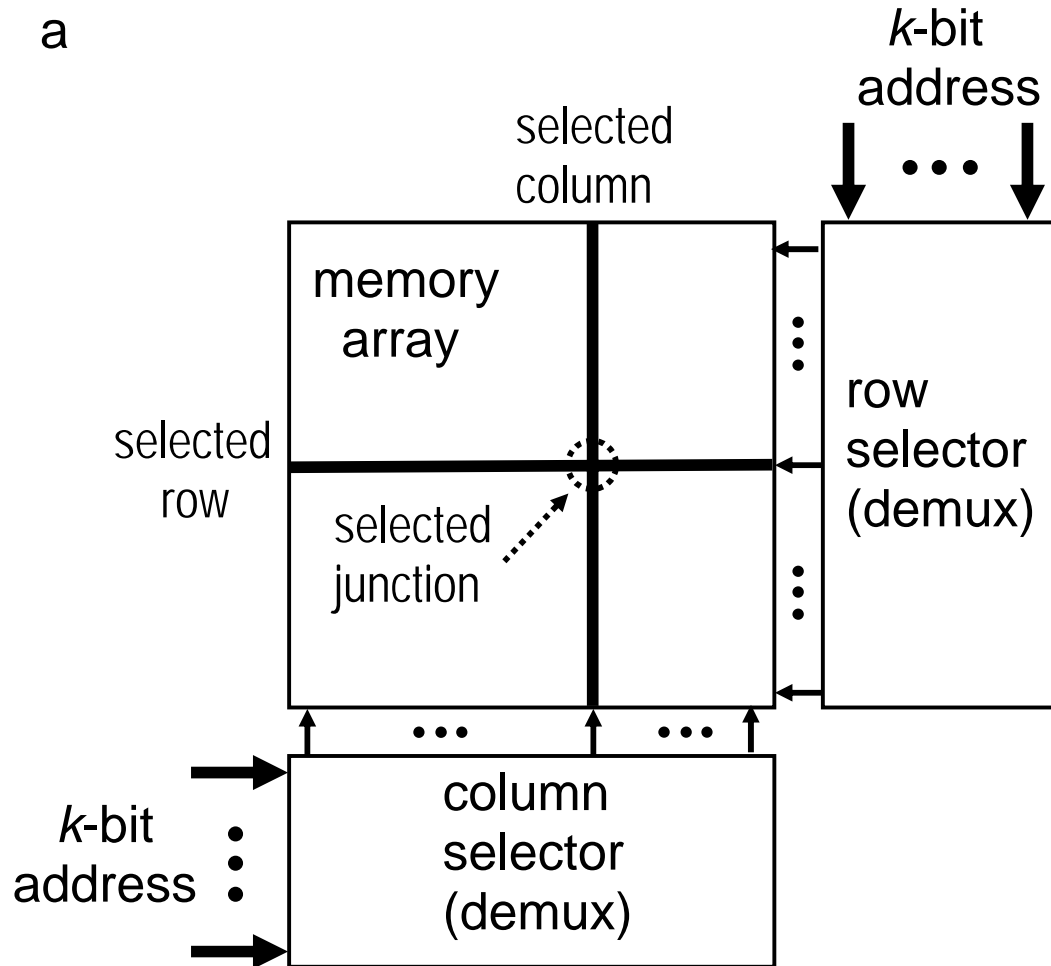
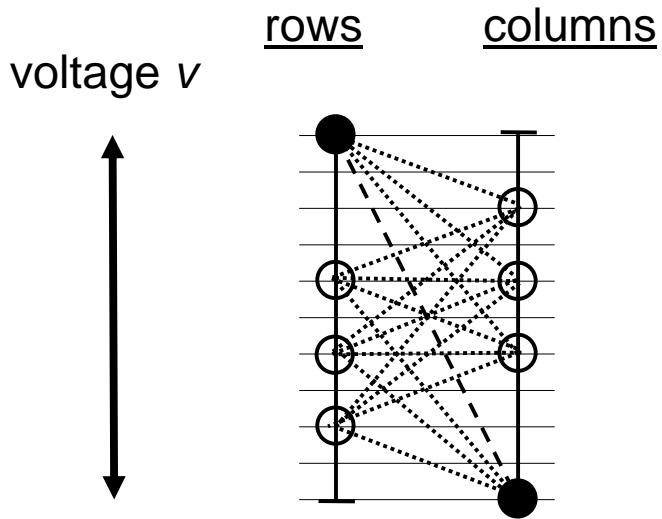


Fig. 5b.

b



- - - selected junction
- ..... non-selected junctions
- selected-line voltage
- non-selected-line voltage

Fig. 5c.

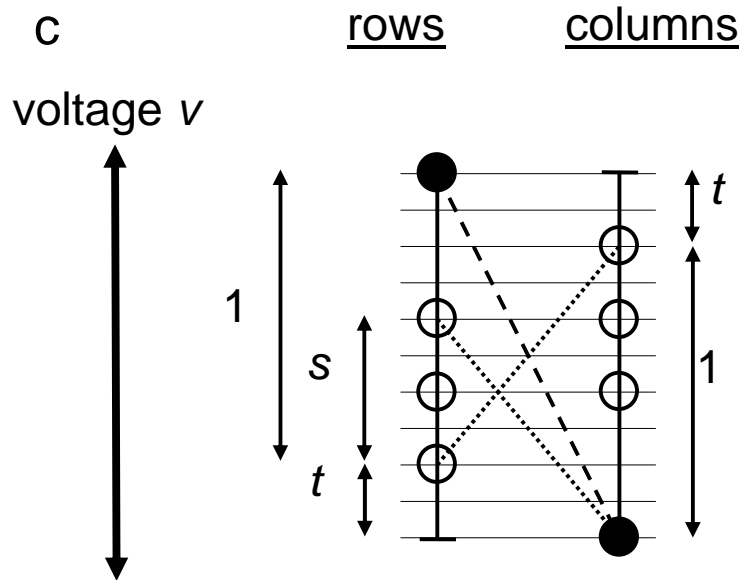


Fig. 6.

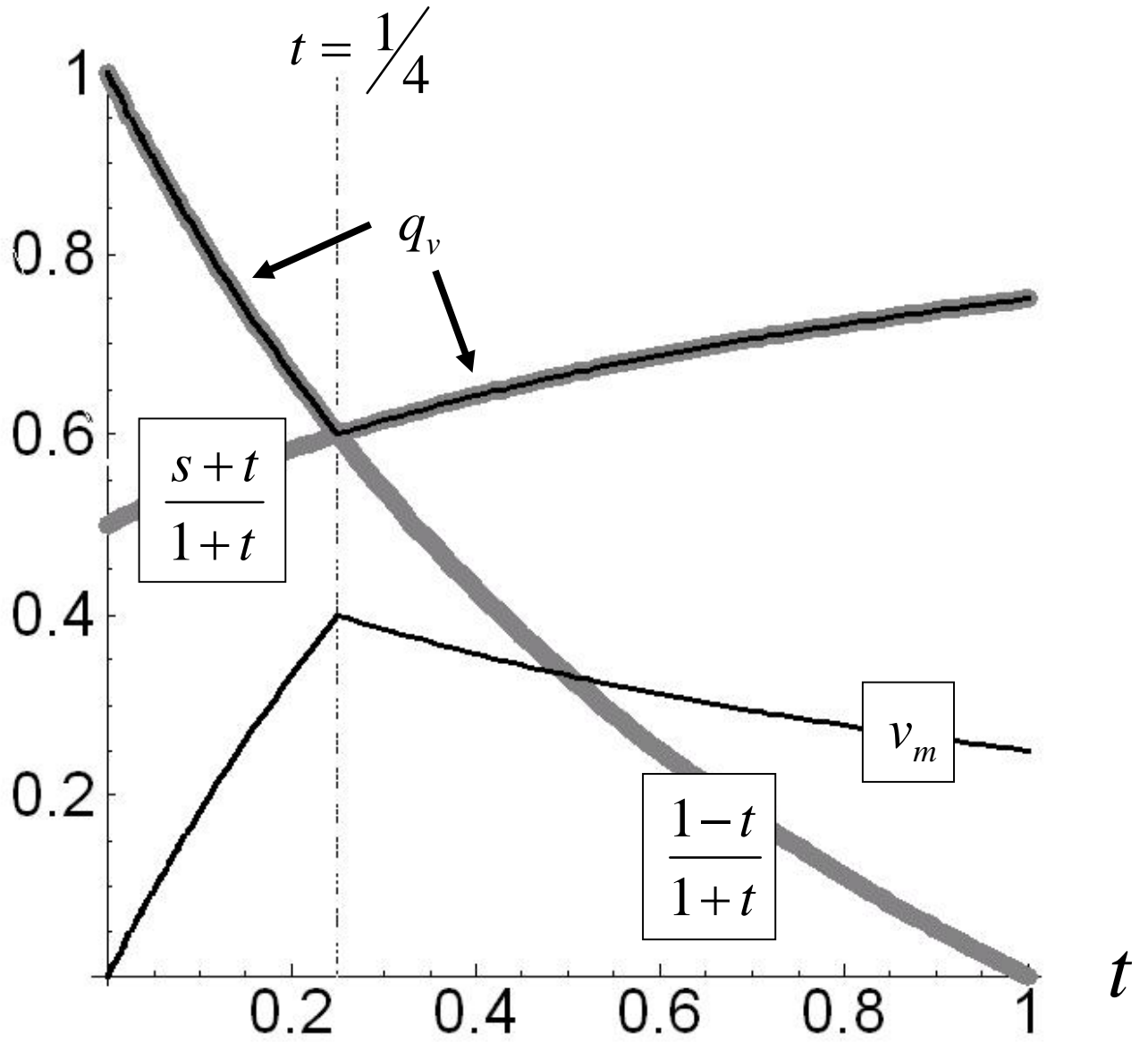


Fig. 7.

