Crossbar Demultiplexers for Nanoelectronics Based on n-Hot Codes

Greg S. Snider and Warren Robinett

***Abstract*—Demultiplexers are expected to be key components in interfacing submicrometer-scale and nano-scale electronic circuits. Designing them is challenging because most nanoarchitectures are limited to simple regular structures, such as crossbars, and nanoelectronic circuits in general are likely to be plagued with relatively high hard-defect and soft-error rates. Previous work has shown how linear codes can be used to design defect-tolerant demultiplexers using resistor or diode crossbars. We extend those results with nonlinear codes, constructing resistor and diode crossbar-based demultiplexers that have better electrical characteristics and defect tolerance for a given area of the nano substrate, at the cost of more complex address encoding circuitry.**

***Index Terms*—Demultiplexing, encoding, fault tolerance, nanotechnology.**

I. Introduction

DEMULTIPLEXERS have been proposed as a mechanism for interfacing conventional, submicrometer-scale electronic circuits to nanoelectronic circuits [1]–[4]. They are appealing since they allow a small number of submicrometer-scale wires (microwires) to control a larger number of nano-scale wires (nanowires), thus providing a mechanism for a CMOS circuit on a silicon substrate to interrogate and configure a nanoelectronic "chip" residing on its surface. Demultiplexers fit in well with many proposed nanoelectronic architectures since they can be implemented in simple crossbar structures.

A crossbar implementation of a demultiplexer uses a single type of component, such as a field-effect transistor (FET), diode, or resistor, configured in a subset of the crosspoints within the crossbar [5]–[8]. Although appropriately configured FET and diode crossbars might, in principle, provide excellent approximations to ideal demultiplexer behavior, they carry certain disadvantages with existing fabrication technology: limited current density, forward-biased voltage drops, high impedance of semiconductor nanowires (compared to metal nanowires), difficulty of configuration, and so on. Resistor crossbars are currently easier to fabricate and configure and offer the potential of building demultiplexers with lower output impedances since they can be built using metal nanowires, but present the challenge of approximating the nonlinear demultiplexer function with linear components. In any case, nanoelectronic circuits will likely contain a higher rate of hard defects and be more susceptible to soft errors than CMOS circuits implemented with current technology, so we desire a certain amount of defect tolerance in our demultiplexers as well.

Linear coding theory [9] has been used as a foundation for creating defect-tolerant demultiplexers out of both diode crossbars [10] and resistor crossbars [11], [12]. This paper explores using *nonlinear* error-correcting codes as a foundation for building both diode and resistor-based crossbar demultiplexers. We find that such codes yield demultiplexers that better approximate ideal demultiplexer behavior and that have a higher degree of defect tolerance; the downside is that they require more complex address encoding circuitry. While this encoding complexity might make the approach unattractive in some scenarios, such as memory bank design, there are other scenarios, particularly chip configuration, where the complexity can be amortized by sharing the address encoding circuitry among several demultiplexers. All of the demultiplexer designs, whether based on linear or nonlinear codes, require the designer to make trade-offs. Which one is "better" for a specific application depends, of course, on the application's objectives. The nonlinear demultiplexers presented here require more address encoding logic than those based on linear codes, but provide better electrical performance, particularly in the presence of defects.

II. Ideal Demultiplexer

A demultiplexer is a logic component that uses a small number of input address lines to select exactly one of N output data lines. For each output line, there must be at least one pattern of voltages applied to the address lines that will cause it to be selected while leaving the remaining lines unselected. The properties of an ideal demultiplexer include the following.

1) A fixed output impedance (preferably zero) for all lines when they are in the "selected" state; similarly, one would hope for a single output impedance value for unselected lines as well.
2) Defect tolerance, such that a few defective components within the demultiplexer have no impact on correct functioning.
3) Simple address encoding. Defect-tolerant, crossbar demultiplexers require simple, consecutive binary addresses to be encoded into a larger, sparser address space. Since this encoding requires additional circuitry, one desires this to be as simple as possible to minimize area.
4) Fixed "selected" and "unselected" output voltages that are widely separated so that they are easily distinguished and that do not vary with different input addresses.

For simplicity, we will assume that our ideal demultiplexer drives a voltage of 1.0 V on the selected output line and 0.0 V on the remaining unselected lines. To measure the deviation of

Manuscript received June 21, 2004; revised August 24, 2004..

The authors are with Hewlett-Packard Laboratories, Palo Alto, CA 94304 USA (e-mail: snider.greg@hp.com; warren@warrenrobinett.com).

Digital Object Identifier 10.1109/TNANO.2004.837853

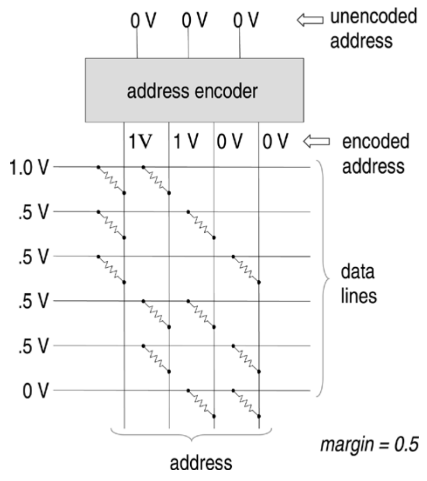

Fig. 1. Small resistor crossbar demultiplexer that selects one of six output lines. The unencoded address is transformed to a larger encoded address which drives the four address lines. The selected output data line (top horizontal line) has the largest output voltage, while the remaining unselected lines have a range of voltages spanning 0–0.5 V. The difference between the selected output voltage and the largest unselected output voltage is called the margin. In an ideal demultiplexer, the margin would be 1, while here it is only 0.5.

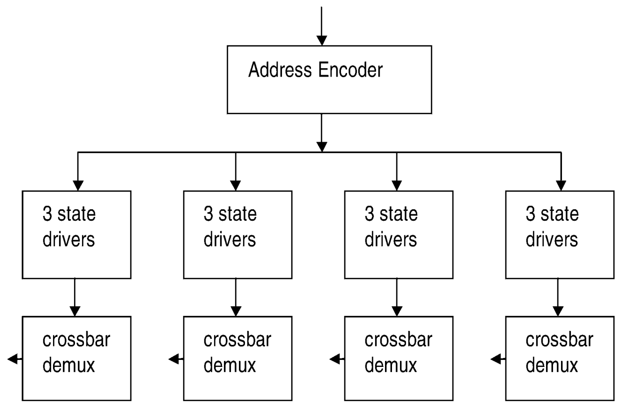

Fig. 2. Sharing an address encoder among several crossbar demultiplexers. This is useful for configuring nanoelectronic chips, which will likely contain a large number of regions that need to be separately addressed and configured.

any particular demultiplexer from this ideal, we use the concept of *margin*, which is defined to be the difference between the worst (lowest) voltage ever driven on any selected output line and the worst (largest) voltage ever driven on any unselected line. For an ideal demultiplexer the margin would be 1; for the demultiplexers we consider here, this value will range between 0 and 1. Fig. 1 illustrates a small, nonideal demultiplexer using a resistor crossbar that has a margin of 0.5. We desire a high margin; a circuit with zero margin would be unusable since one could not distinguish between selected and unselected lines.

Depending on the nature of the target technology and the usage envisioned for the demultiplexer, the relative importance of the ideal multiplexer properties will vary. Insufficient margin, for example, might cause burn-out or misconfiguration of cross-point cells in some nanoelectronic technologies but not others. Simple address encoding might be very important for implementing the "row select" lines in memory banks so as to minimize support circuitry overhead, while encoding complexity may not matter much for a chip configuration application if the encoding hardware can be shared by multiple configuration blocks within a chip (Fig. 2).

III. Coding Theory for Demultiplexer Design

This section gives the briefest of overviews of the parts of coding theory relevant to demultiplexer design, and ties in the theory with some concrete examples. We start with a few definitions.

- A *codeword* is an ordered sequence of bits with a fixed length. For example, "0001" and "1011" are both codewords of length 4.
- The *weight* of a codeword is simply the number of 1's in that codeword. The codeword "1001" has weight 2, the codeword "0000" has weight 0, and the codeword "1111" has weight 4.
- The *distance* between equal-length codewords (known as the Hamming distance) is the number of corresponding bit positions in which the two codewords differ. The codewords "1001" and "1000" are a distance 1 apart since they differ only in their final bit, while "1100" and "1011" are separated by a distance of 3.
- A *code* is a set of codewords. The *length* of a code is the number of bits in each codeword—all codewords in a code must have the same length. An example of a code with length 4 is $\{1100, 0110, 0011, 1001\}$. We will call the code *dense* if it consists of all possible codewords for that length, e.g., the code $\{000, 001, 010, 011, 100, 101, 110, 111\}$ is dense.
- The *distance* of a code (which we denote by d) is the smallest distance between any two codewords in the code. If all pairs of codewords have exactly the same distance, the code is "equidistant." The code $\{1000, 0100, 0010, 0001\}$ has a distance of 2 and also happens to be equidistant. Dense codes always have a distance of 1.
- A *constant weight code* consists of codewords that all have exactly the same weight. The code $\{1100, 0110, 0011, 1001\}$ is a constant weight code, while $\{1000, 1100, 1110, 1111\}$ is not.

To be useful, nondense codes are generally constructed with a distance d greater than one. This makes it possible to detect small numbers of bit errors which might occur when transmitting a codeword from one place to another. For example, the equal weight code $\{1100, 0110, 0011, 1001\}$ has a distance of two; taking any codeword from the code and altering one of its bits produces a new codeword that is clearly not in the code (since its distance from the original codeword is only 1)—and we can therefore detect the fact that an error has occurred. If the distance of a code is d, we can detect up to $(d-1)$ bit errors in a single corrupted codeword. Note that, for a dense code, the distance is 1, and it is not possible to detect any bit errors. We continue with definitions.

- An *encoder* maps the elements of a dense code onto the elements of a sparser code whose distance is usually greater than 1. For example, the encoder $\{00, 01, 10, 11\} \rightarrow \{1000, 0100, 0010, 0001\}$ maps the codeword 00 onto 1000, 01 onto 0100, and so on. The mapping is one-to-one and onto.

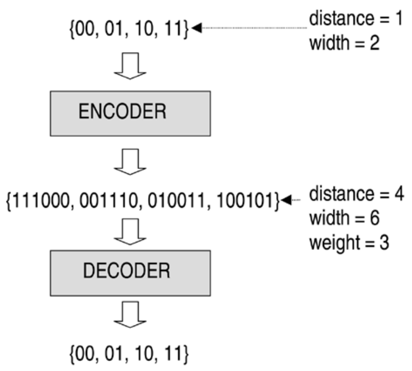

Fig. 3. Encoding a dense code to a sparser code of greater length and distance. The decoder recovers the original unencoded values.

- A *decoder* does the inverse operation, mapping a sparse code onto a denser one, e.g., $\{1000, 0100, 0010, 0001\} \rightarrow \{00, 01, 10, 11\}$.

One important difference between an encoder and decoder is that a decoder can be a many-to-one mapping. It is this property that allows us in certain cases to detect and correct errors. A typical scenario (Fig. 3) involves the encoding of data into a sparser redundant code transmitting the encoded values over a noisy channel and then decoding the possible corrupted received values. If the noisy channel is not too noisy, and we are clever in the way we construct the code and implement the decoder, the decoder can correct a small number of errors introduced by the transmission.

A. Relevance to Demultiplexers

An ideal demultiplexer maps a dense code onto a sparse, weight-1 code, such as $\{00, 01, 10, 11\} \rightarrow \{1000, 0100, 0010, 0001\}$. We want to build such an encoder, but we are forced to do so using a limited palette of components—either resistors or diodes—a fraction of which may be defective. We break the demultiplexer into two parts: a front-end address encoder that produces an intermediate code, and a back-end (built out an array of "recognizers") that transforms the intermediate code into the final sparse code.

Each recognizer will compute a single output bit of the overall encoder, mapping from the intermediate code onto the code $\{0, 1\}$. Even though we want the recognizers to be independent to make their implementation easier, we still want them to collectively act like a demultiplexer, namely that exactly one output wire will have a 1 output while the remaining wires have a 0 (Fig. 4).

We assume that the front-end encoder can be built out of perfect components (implemented in CMOS on a silicon substrate, for example) and that output recognizers are built out of possibly defective nanocomponents. This is a slightly different model than usual one found in coding theory (perfect encoder, noisy channel, perfect decoder) since we are lumping the noisy channel and recognizers together to create "defective recognizers." The idea is to be clever in the design of the intermediate code, introducing enough redundancy into it that the output recognizers are *self-correcting*, compensating for bit corruption that they themselves introduce.

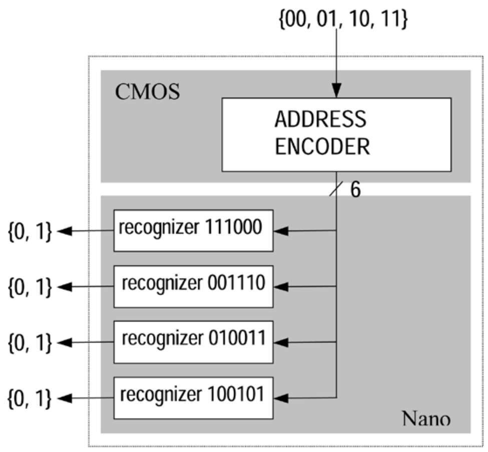

Fig. 4. Split-implementation of a demultiplexer using CMOS and nanocrossbars. The CMOS is used to implement an address encoder which maps a dense code to a sparser, redundant code that is fed to the inputs of an array of (possibly defective) "recognizers." Each recognizer recognizes a single codeword of the intermediate code produced by the address encoder. Even though each recognizer is independent, collectively the recognizers implement the demultiplexer outputs, preserving the single-active-output property of the demultiplexer as long as the defect rate of the nano crossbar is not too high.

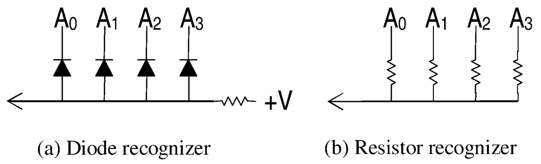

Fig. 5. Diode and resistor recognizers. The diode recognizer (a) implements an AND gate that outputs a logic 1 when all address inputs are 1. The resistor recognizer (b) averages the address inputs (ignoring output loading) to approximate an AND gate.

B. Defective Recognizers

We wish to design demultiplexers which can be fabricated on nano-crossbars using simple processes, and this limits our implementation possibilities for the recognizers. The two designs that we consider here, one for diode crossbars and one for resistor crossbars, are shown in Fig. 5.

The diode recognizers implement AND gates. Each recognizer connects to a subset of "encoded address lines" (carrying the intermediate code from the front-end address encoder) and "triggers" a 1 output when all of the address lines carry a logical 1 value. But note that a logic 0 output voltage of a diode recognizer is offset from zero output voltage by the forward bias voltage drop across one of the diodes. The resistor recognizers are simple summing networks that take a subset of the address lines and compute an average of the voltages applied to those lines (this neglects any loading on the output of the demultiplexer)—they also provide an approximation of AND gates.

These recognizers can fail from transient glitches in both address and data lines. But here we'll only consider static defects, where a diode or resistor that we had expected to be present in the implementation is missing. We call this a "stuck-open" defect. It is also possible to have static defects that result from additional resistors or diodes appearing in the crossbar that we didn't want ("stuck-closed" defects), but we will not consider those in this paper.

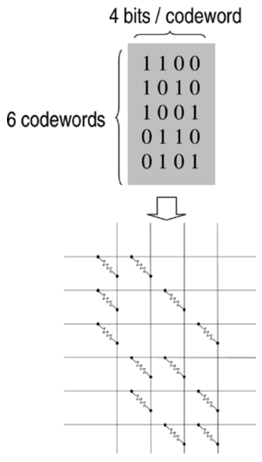

Fig. 6. Using a code to create a resistor-crossbar demultiplexer. Each codeword corresponds to a row in the resistor crossbar. A "1" in a codeword signifies the presence of a resistor at the corresponding crosspoint, while a "0" signifies the absence of a resistor.

C. Mapping Codes to Crossbars

The intermediate code that we will design can easily be mapped onto a demultiplexer implementation as illustrated in Fig. 6. Each codeword corresponds to a single recognizer in the recognizer array. The significance of this is that a single recognizer can be viewed as a pattern matcher for that particular codeword: if the recognizer senses that the address lines equals its pattern (or is close to it in a Hamming distance sense) then it outputs a 1, otherwise it outputs a 0. Note that for resistor decoders, the voltage levels corresponding to logic 1 and logic 0 are necessarily voltage ranges, and we will need to find an appropriate threshold separating the two.

IV. Resistor Demultiplexers

Designing a demultiplexer from a resistor crossbar entails determining a desired size for the crossbar (in terms of the number of output data lines to be driven and the desired number of encoded address lines) and then figuring out which crosspoints within the crossbar to configure with resistors. Next, an address encoder needs to be designed to generate the voltages on the address lines. Once that is all done, the demultiplexer can be evaluated for its cost, defect-tolerance, and deviation from ideal behavior, for example, and compared with other possible designs. The evaluation function is strongly application-dependent. In this section, we will simplify matters by focusing our evaluation primarily on margin and defect-tolerance and ignore address encoder complexity and output impedance considerations (actually, the output impedance for each of the resistor-based designs explored here is constant in the absence of defects).

A. Strategy

The design space for resistor crossbar-based demultiplexers is enormous. In principle, any pattern of resistor populations in the crossbar could be evaluated as a potential candidate for building a demultiplexer (Fig. 6). However, even for a small crossbar of, say, dimensions 8×8, the number of possible patterns is 2^{64}, so exhaustive searching is not feasible. One must somehow severely limit the search to areas of the design space that are "likely to be good," or at least "not likely to be too bad."

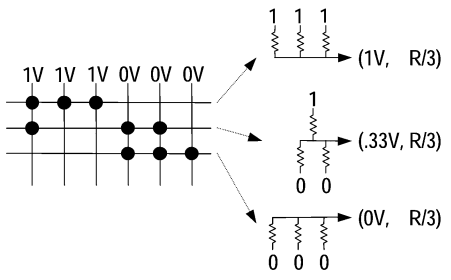

Fig. 7. Constant-weight codes create demultiplexers with a fixed output impedance for all output lines, selected or unselected.

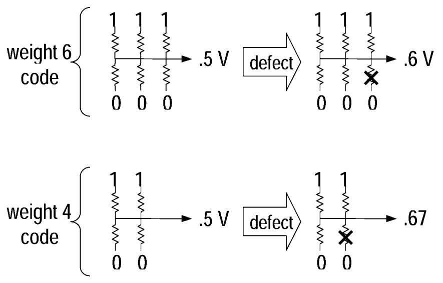

Fig. 8. High-weight codes generate demultiplexers with better defect tolerance. A single failing resistor ("stuck-open") has less impact on the output voltages on data lines.

The heuristics we use to narrow the search are derived from the ideal demultiplexer properties mentioned in Section II. The desire for constant output impedance (along with the desire for some symmetry in the behavior of the outputs) suggests that *constant-weight* codes (where the number of 1's in each codeword is the same) might be a good region to explore (see Fig. 7). These are also known as "n-hot" codes [3]. Since, as we shall see, margin is determined by the minimum Hamming distance between all codewords within a code, we can simply skip evaluating codes where that distance would yield a margin falling below a minimum acceptable threshold. The desire for maximal margin also suggests that the number of resistors on any output data line not exceed half the total number of address lines (otherwise encoded addresses will exists that produce at least one selected line that has an output voltage less than 1.0 V). Defect tolerance suggests that the weight of the codes be as high as possible (Fig. 8). Address encoding complexity, though, is difficult to evaluate without actually synthesizing and optimizing an encoder design, so we do not use that to prune the search space, but instead use it to evaluate the designs produced by that search.

It is possible that these search-space pruning heuristics miss potentially good demultiplexer designs and that cleverer pruning would rout them out. It is also possible that some of the codes found with this strategy could actually be linear—it would certainly be fortuitous if that occurred since it would imply a simple address encoder implementation.

B. Search Algorithm

The search algorithm requires the following input parameters:

- minimum number of data lines to be driven by the demultiplexer outputs ($D_{\min}$);
- maximum number of address lines which the user wishes to consider ($A_{\max}$);
- smallest value of acceptable margin ($M_{\min}$).

From these inputs, additional constraints on the search space can be deduced. Since we are addressing $D_{\min}$ output lines, we must have at least $\lceil \log_2(D_{\min}) \rceil$ address lines in order to be able to distinguish each output line. This implies a lower bound on the number of address lines, $A_{\min}$.

The number of codewords in a full constant-weight code of weight W with A bits per codeword is $\binom{A}{W} = A!/(W!(A-W)!)$. This places a lower bound on acceptable values for W, $(W_{\min})$, since, if this value is less than $D_{\min}$, the code cannot drive all of the desired output lines and therefore cannot possibly be a candidate for the demultiplexer.

The lower bound on acceptable margin also constrains the set of acceptable codes. Since the margin for a given code is $d/(2 \cdot \text{weight})$ and since the user constraint requires this to be greater than or equal to $M_{\min}$, we then only need examine codes where d satisfies the equation $d >= 2 \cdot \text{weight} \cdot M_{\min}$. Furthermore, the distance between any pair of codewords in an equal weight code is even, so that we may also ignore odd values of d. The distance between any two codewords in a constant weight code can never exceed $2 \cdot \text{weight}$, which places an upper bound on d. These constraints prune a large number of codes from consideration.

Searching then proceeds by generating a sequence of candidate codes that fit within the constraints of $(A_{\min}, A_{\max}, W_{\min}, D_{\min}, d)$. Given a number of address lines A and code weight W, we generate the corresponding constant-weight code. Each codeword in the code contains A bits, one for each address line, with exactly W bits within that word set to 1. Each codeword represents a single row in the resistor crossbar, such that a "1" in the codeword represents the presence of a resistor at a crosspoint while a "0" represents the absence of a resistor (as was shown in Fig. 6).

This generated code will likely not meet the constraints on d derived from the margin, so we look for a subset of the generated code that does. This is accomplished using a "sieve" technique that compares every codeword in the code with every other, eliminating codewords that have insufficient distance from others. This algorithm produces a subset of codewords constituting a smaller code that does meet the distance constraints, but it may not be an optimal subset in the sense that there may be other subsets also meeting the constraints that contain more codewords. Since the number of codewords in this smaller code must at least be equal to $D_{\min}$ in order to implement a demultiplexer, we can eliminate the code from further consideration if it does not.

C. Resistor Demultiplexer Results

As an example, the search algorithm was used to explore demultiplexers capable of driving at least 64 data lines ($D_{\min} = 64$) with no more than 22 address lines ($A_{\max} = 22$) and a margin of at least 0.5 ($M_{\min} = 0.5$). The results, shown in Table I, illustrate some of the tradeoffs available to the designer. If minimizing address lines is essential, it is possible to meet the margin requirement with as few as 12 address lines (the first code listed at the top of the table). Note, however, that the small distance for that particular code implies that the resulting demultiplexer would not be defect-tolerant—a single defective resistor would cause the data line on which it appeared to malfunction.

TABLE I
CODES WITH ($M_{\min} = 0.5, D_{\min} = 64, A_{\max} = 22$)

address lines	data lines	weight	d	margin
12	66	2	2	0.5
13	78	2	2	0.5
14	91	2	2	0.5
14	77	4	4	0.5
15	105	2	2	0.5
15	105	4	4	0.5
16	120	2	2	0.5
16	140	4	4	0.5
17	136	2	2	0.5
17	140	4	4	0.5
17	65	6	6	0.5
18	153	2	2	0.5
18	148	4	4	0.5
18	88	6	6	0.5
19	171	2	2	0.5
19	164	4	4	0.5
19	118	6	6	0.5
19	78	8	8	0.5
20	190	2	2	0.5
20	189	4	4	0.5
20	73	5	6	0.6
20	155	6	6	0.5
20	130	8	8	0.5
21	210	2	2	0.5
21	221	4	4	0.5
21	85	5	6	0.6
21	191	6	6	0.5
21	210	8	8	0.5
22	231	2	2	0.5
22	263	4	4	0.5
22	100	5	6	0.6
22	237	6	6	0.5
22	77	6	8	0.666
22	66	7	8	0.571

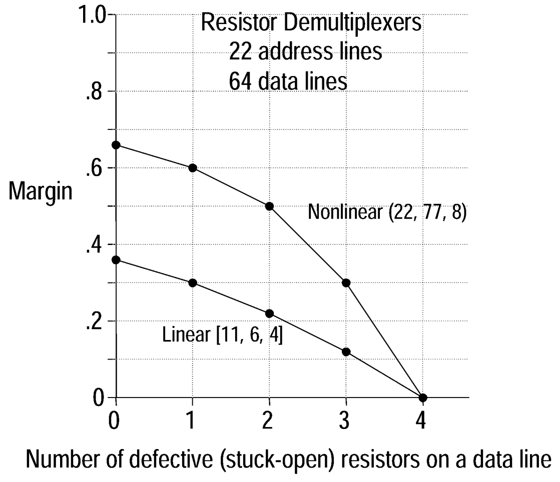

Fig. 9. Comparing "linear" and "nonlinear" encoded demultiplexers. Defects are "stuck-open" (missing resistor) only. The demultiplexers generated from nonlinear codes have better margin that that produced by the best linear code.

A much more robust code utilizing 22 address lines and driving up to 77 data lines appears near the bottom of the table. Not only does the margin of this particular code exceed that of the others (0.666), but its distance of eight implies a high degree of defect tolerance.

Fig. 9 compares the robust "nonlinear code" demultiplexer just described with the best "linear code" demultiplexer found to date using linear coding theory [11]. Both demultiplexers are capable of driving 64 data lines, and both require 22 address lines as inputs, but the "nonlinear" demultiplexer has significantly better margin and tolerance of "stuck-open" defects, while having increased address encoding complexity.

V. Diode Demultiplexers

Due to their nonlinearity, diode-based demultiplexers do not have a "margin" that needs to be considered; the search algorithm for resistor-based demultiplexers can be used here as well. When diode demultiplexers fail (due to defective, stuck-open diodes), it is because a diode decoder lacks enough inputs to distinguish the codeword with which it is associated. The number of stuck-open diodes that any given decoder can tolerate is $(d/2) - 1$, where d is the distance of the code used to design the demultiplexer. For example, using the same 22 address line demultiplexer considered in the last section, and replacing the resistors with diodes, we can tolerate up to 3 diode failures on each data line without losing any demultiplexer functionality. The simplest diode demultiplexer that can be built for addressing 64 data lines can be implemented with only 12 address lines and only two diodes per data line, but would lack any defect tolerance at all since the distance of the generating code is only two.

VI. Summary

Nonlinear error-correcting codes can be used to design defect-tolerant resistor or diode demultiplexers for nanoelectronics, and they produce more compact crossbar implementations than linear codes, although at the cost of increased address encoder complexity. Resistor crossbars require the designer to consider the effects of margin in the presence of defectss, but have the advantage of being easier to implement with existing technology and might have lower output impedances since they can be fabricated using metal nanowires. Diodes are more robust in the presence of stuck-open defects, but are currently more difficult to fabricate; they also have the problem of forward bias voltage drop across the diodes that offset the demultiplexer output voltages from their ideal values.

References

[1] S. Williams and P. Kuekes, "Demultiplexer for a molecular wire crossbar network," U.S. Patent 6 256 767, Jul. 3, 2001.

[2] Y. Chen, G. Jung, D. Ohlberg, X. Li, D. Stewart, J. Jeppesen, K. Nielsen, J. Stoddart, and R. Williams, "Nanoscale molecular-switch crossbar circuits," *Nanotechnol.*, vol. 14, pp. 462–468, 2003.

[3] Z. Zhong, D. Wang, Y. Cui, M. Bockrath, and C. Lieber, "Nanowire crossbar arrays as address decoders for integrated nanosystems," *Science*, vol. 302, pp. 137–139, 2003.

[4] A. DeHon, "Array-based architecture for FET-based nanoscale electronics," *IEEE Trans. Nanotechnol.*, vol. 2, no. 2, pp. 23–32, Mar. 2003.

[5] M. Stan, P. Franzon, S. Goldstein, J. Lach, and M. Ziegler, "Molecular electronics: From devices and interconnect to circuits and architecture," *Proc. IEEE*, vol. 91, no. 11, pp. 1940–1957, Nov. 2003.

[6] S. Goldstein and M. Budiu, "NanoFabrics: Spatial computing using molecular electronics," in *Proc. 28th Int. Symp. Computer Architecture*, 2001, pp. 178–191.

[7] P. J. Kuekes, R. S. Williams, and J. R. Heath, "Molecular wire crossbar memory," U. S. Patent 6 128 214, Oct. 3, 2000.

[8] ——, "Molecular-wire crossbar interconnect (MWCI) for signal routing and communications," U. S. Patent 6 314 019, Nov. 6, 2001.

[9] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*. New York: North-Holland, 1990.

[10] W. Robinett, P. Kuekes, G. Seroussi, and R. Williams, "Defect-tolerant interconnect to nanoelectronic circuits: Internally redundant demultiplexers based on error-correcting codes," *Nanotechnol.*, to be published.

[11] P. Kuekes, W. Robinett, G. Seroussi, and R. Williams, "Resistor-logic demultiplexers for nanoelectronics based on error-correcting codes," , submitted for publication.

[12] ——, "Stuck-closed and stuck-open defects: Defect-tolerance in resistor-logic demultiplexers," , submitted for publication.

Greg S. Snider is currently a consultant to Hewlett-Packard Laboratories, Palo Alto, CA, where he investigates nanoelectronic architectures, circuits, compilation, and simulation. He was previously involved with communications, processor design, medical instrumentation and imaging, networking, operating systems, computer security, memory systems, compilers, hardware and software synthesis, e-services, simulation, and programmable hardware. In the early 1990s, he was the architect and compiler designer of the Teramac simulation system, a defect-tolerant computer built from hundreds of custom field programmable gate arrays.

Warren Robinett is currently a Computer Scientist, working as a contractor with the Quantum Science Research Group, Hewlett-Packard Laboratories, Palo Alto, CA, where he is involved with defect-tolerant computing architectures for nanoelectronics. In the 1990s, while with the University of North Carolina, he co-invented the NanoManipulator, a virtual-reality interface to a scanning-probe microscope, which allows a scientist to be virtually present on the surface of a microscopic sample within the microscope. In the mid-1980s, while with the National Aeronautics and Space Administration (NASA) Ames Research Center, he designed the software for the Virtual Environment Workstation, NASA's pioneering virtual reality project. In 1980, he co-founded The Learning Company, currently a major publisher of educational software, where he designed *Rocky's Boots*, a computer game, which taught digital logic design to upper grade-school children, using interactive visual simulation. In 1978, he designed the Atari video game *Adventure*, the first action-adventure game.

Mr. Robinett was the recipient of Software of the Year awards from three magazines in 1983 for *Rocky's Boots*.