PESEC – A Simple Power-Efficient Single Error Correcting Coding Scheme for RRAM

Shlomo Engelberg
Dept. of Electrical and Electronics Engineering
Jerusalem College of Technology, Jerusalem, Israel

Osnat Keren
Faculty of Engineering
Bar-Ilan University, Ramat Gan, Israel

***Abstract*—The power consumed when writing to Resistive Random Access Memory (RRAM) is significantly greater than that consumed by many charge-based memories such as SRAM, DRAM and NAND-Flash memories. As a result, when used in applications where instantaneous power consumption is constrained, the number of bits that can be set or reset must not exceed a certain threshold. In this paper, we present a power-efficient, single error correcting (PESEC) code for memory macros, which, when combined with bus encoding, ensures low-power operation and reliable data storage. This systematic, multiple-representation based single-error correcting code provides a relatively high rate, with a marginal increase in implementation cost relative to that of a standard Hamming code, and it can be used with any bus encoder.**

***Index Terms*—Emerging non-volatile memory, Resistive RAM (RRAM), Write after Read, Bus encoding, Single error correction (SEC), Power-efficient coding.**

I. Introduction

There are many types of memory, and each has strengths and weaknesses. (See, for example, Table 1 of [1] and [2].) When choosing what type of memory to use, tradeoffs must be made. When choosing non-volatile memory, if minimizing the area taken up by the memory and the time to read from or write to the memory is critical, users are likely to consider technologies such as Resistive Random Access Memory (RRAM) which do not require much real estate but may consume markedly more energy when the value of a bit is changed [1]–[4].

When changing bits in a memory consumes substantial amounts of energy and the *available instantaneous power* is limited, it is critical to change as few bits as possible when writing to the memory. This is the job of a processor's *bus-encoder*. In order to ensure the reliability of the data stored in the memory, it is important to add error correction capability to the RRAM macro. To keep the "cost" of the error protection to a minimum, it is *crucial* that the error correcting code (ECC) implemented in the RRAM macro be designed in a *power-efficient* fashion. In this paper, we present a new power-efficient ECC-encoder that can be placed in memory macros that make use of memory, such as (certain types of) RRAM, for which writing consumes substantial amounts of energy.

Our contribution: While most research in this area focuses on minimizing average power consumption by reducing the *average* number of bit transitions, relatively few address the problem of limiting the number of bit transitions in a single write operation. We focus on this problem and introduce a power-efficient single error correction (PESEC) coding scheme that is compatible with all bus encoders. Our code is based on multiple representations of each data word by codewords from a conventional Hamming code. The key advantage of our scheme is its very low implementation cost, which is negligible compared to the cost of the best-known solutions and which remains close to that of a conventional low-cost Hamming code. The ideas presented can be applied to the design of power-efficient SEC-DED codes.

The remainder of this paper is organized as follows. Section II reviews related work. Section III formulates the coding problem, and Section IV introduces the ECC-coder. The efficiency, cost and performance of our coding scheme are presented in Section V, and Section VI concludes the paper.

This work was supported by the Israel Science Foundation under Grant 773/24.

II. Preliminaries and Related work

A. Redundancy and multiple representations

In order to protect data from errors, one generally introduces redundancy. It is often convenient to employ systematic codes; i.e., to introduce the redundancy as bits that are "tacked on" to the data being protected (rather than mapping each possible data-word to a codeword that does not contain a copy of the data). In this way, the (possibly correct) data can be viewed without dealing with the redundant bits.

In what follows, we make use of systematic codes and associate multiple codewords of length n (the memory width) with each k-bit data word, $k < n$. Each codeword has $r = n - k$ redundant bits.

We start with a simple example in order to demonstrate how multiple representations of each data word can be useful. Assume that one wants to store one of four messages: 'A','B','C', or 'D.' The possible messages can be represented by $k = 2$ bits, $00, 10, 01$ and 11, respectively. To protect the messages from errors, we must append at least three redundant bits, and to reduce the power consumed by using the code, we choose multiple representations for each message. Consider the following codewords, each of length $n = 6$:

$$\begin{array}{rcll} A & = & \{(\mathbf{00}\ 0000), (\mathbf{00}\ 1011)\} & C = \{(\mathbf{01}\ 0110), (\mathbf{01}\ 1101)\} \\ B & = & \{(\mathbf{10}\ 1110), (\mathbf{10}\ 0101)\} & D = \{(\mathbf{11}\ 1000), (\mathbf{11}\ 0011)\}. \end{array}$$

Here, 'A' has two possible representations, and in both, the first two digits are 00. (Both representations have four bits

978-3-9826741-0-0/DATE25/© 2025 EDAA

of redundancy.) Similarly 'B','C', and 'D' have two possible representations. It is easy to verify that this code is linear, and the smallest distance between any two of the eight codewords is 3. Therefore, it is a single error correcting code.

To see how multiple representations reduce the amount of energy used when writing, assume that initially the memory held an 'A' in the form $(000\ 000)$. Suppose one would like to write a 'B' to this location. Then one can write either $(100\ 101)$ or $(101\ 110)$. To reduce the number of bits that must be changed, one chooses the first representation and changes 3 bits. The memory location now contains $(100\ 101)$. Suppose one would like to write a 'D' to this location. There are two choices – to write a $(111\ 000)$ or a $(110\ 011)$. In order to minimize the cost in energy, the second option is preferred, and it requires changing three bits. In fact, *because of the multiple representations for each symbol* it is possible to keep the number of bits changed to three – which is the best one could hope for in a one-error correcting code (whose minimum distance must be at least 3).

When using the *single*-representation code given by the left-hand word of each pair, the maximal number of bits that must be changed when going from one message to another is four and the arithmetic mean is $3\frac{1}{3}$. Thus, when the instantaneous power used when writing must be kept to a minimum, the code using multiple representations is preferred.

Making use of multiple representations is an important component of many bus-encoding techniques which are techniques designed to reduce the number of wires on a bus that experience a change in value when data are moved. Some of the same power-efficient bus encoding techniques designed for use with communication links may be employed to reduce the power consumed when writing to locations in the memory by minimizing the number of bit transitions during this operation, and such units are still referred to as bus-encoders [5], [6]. Power-efficient bus-encoders are generally classified into encoders that utilize probability-based codes, permutation-based codes, or algebraic codes [5]. Some bus encoding techniques leverage the unique properties of the data values to decrease power consumption. For example, in applications where errors can be tolerated without significant loss of information, introducing small changes in the data can substantially reduce the number of transitions [7]. In addition, when there is a temporal correlation between successive words written to the same address, algebraic differential encoders can exploit this correlation to minimize the number of bit transitions [8]. For instance, in applications where the information represents numbers or addresses and the difference between two words (addresses) is small, permutation codes such as the Gray code have been employed [9], [10].

A recent article by Giraud et al. [3] discusses the performance and reliability of a 128-kb RRAM macro implemented in 130-nm CMOS technology. It is a particularly relevant example of the advantages of using bus-encoding techniques (bus inversion) in combination with other power reduction and time-saving techniques. They showed that employing read-before-write reduces the power consumption during programming operations by 47%. By combining this technique with current-limitation techniques, write-termination techniques, write verification, and error correction code mechanisms, they achieve an 83% reduction in energy consumption and a 55% decrease in access time.

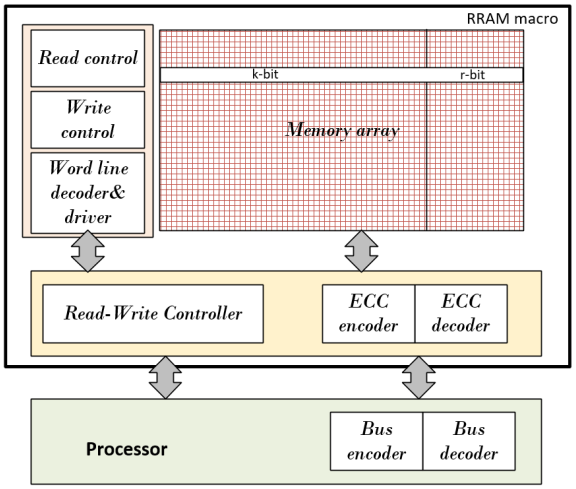

Fig. 1. Schematic diagram of a memory (RRAM) architecture. The memory width is $n = k+r$. The memory macro includes an ECC-encoder and decoder. The processor contains the bus-encoder and decoder.

Most bus-encoding techniques aim to minimize the average number of bit-transitions. In this paper we are interested in developing bus-encoding techniques for application in which *the restriction is on the maximal power (i.e., the maximal number of bit transitions) that can be consumed during one write operation.*

B. WEMs – Write Efficient Memories

The problem of storing data in memory for which certain transitions in the memory are assumed to be "expensive" is not new [11], and such memories are known as Write-Efficient-Memories (WEMs). The problem of designing codes to ensure the reliability of WEMs where a read operation is permitted before the write and there is a restriction on the maximal number of bit-transitions was dealt with in [12]. Following Heegard, who studied the problem of improving the reliability of message storage in memory susceptible to stuck-at defects and noise by using partitioned linear codes [13], Fu and Yeung presented *nested coding schemes* for WEMs [12]. These coding schemes are inherently not modular; the error control and bus-encoding mechanisms are intertwined. Moreover, the cost of the encoder is significant, as a real-time implementation requires a lookup table whose size grows exponentially with the number of bits at the encoder's input.

Initial efforts to reduce the size of the lookup table in power efficient codes were made in [14]. Here, we eliminate the need for a lookup table entirely, by presenting a systematic encoder that employs a conventional Hamming code followed by several small logic decoders.

III. The Formulation of the Design Problem

A. Error model

Writing to an RRAM involves changing the resistivity of the material from which the RRAM is made, and this activity is energy intensive. If it is unsuccessful, values are wrongly assigned. In addition, sometimes memories suffer from stuck bits [3, §II]). Moreover, random telegraph noise can cause read errors [1, p. 13].

B. Assumptions

Figure 1 shows the RRAM's architecture in schematic form. In this work, we assume that the bus-encoder and decoder, shown in Figure 1 as external to the memory macro, as well as their properties are neither known nor of interest to the designer of the RRAM macro. Our focus is on the ECC's encoder and decoder for the error correcting code, which are internal to the memory macro. Because writing data to an RRAM is energy intensive, our ECC *encoder* will be power efficient. As we know nothing about the distribution of the output of the bus encoder (if one is present), when designing the encoder, we treat all data words as equally likely. Because reading is not energy intensive, we do not require our ECC decoder to be power efficient.

In what follows, we assume that the energy used to "flip" bits grows linearly with the number of bits flipped. Furthermore, we assume that the PESEC encoder is allowed to cause no more than τ bit-flips to the r-bit redundant part. This parameter is provided to the processor's designer so that the designer can make an informed choice of the number of bit flips the bus encoder can cause in each write operation.

C. The design problem

In what follows, a binary code $\mathcal{C}$ of length n refers to a subset of the set of n-bit binary vectors ($\mathbb{Z}_2^n$), and a codeword in $\mathcal{C}$ is an n-bit vector $(c_{n-1}, \ldots, c_1, c_0)$. We require that the encoding be *systematic*: that it map each k-bit word $u = (u_{k-1}, \ldots, u_0)$ into a codeword of the form

$$\begin{aligned} c &= (c_{n-1}, \ldots, c_1, c_0) = \\ &= (\underbrace{u_{k-1}, \ldots, u_1, u_0}_{k\text{-bit inf. word } u}, \underbrace{v_{r-1}, \ldots, v_1, v_0}_{r\text{-bit redundancy word } v}) \in \mathcal{C}. \end{aligned}$$

Remark 1. *In what follows, we refer to the word input to the processor as a data word and to the output of the processor's bus encoder – which is the input to the memory macro – as an information word.*

The information word, u, can be a raw data vector or the output of a bus-encoder. Since our interest is confined to the design of power-efficient, systematic ECC-encoders, optimizing the bus-encoder is beyond the scope of this paper. We require that one should be able to concatenate the PESEC encoder with any type of bus-encoder as long as it generates an output vector of length k. The decoupling of the design and operation of the PESEC-encoder from that of the bus-encoder, allows the system to *change bus-encoders dynamically* according to current (power consumption) requirements *without notifying the PESEC encoder*. Thus, the problem we must address is:

Problem formulation: *For a given τ and k, construct a PESEC encoder that makes use of a systematic, SEC code, $\mathcal{C}$, of minimal length n, such that for every previously stored (and possibly erroneous) word y, $y = c_{\text{prev}} \oplus e$, and any k-bit information word u that enters the SEC encoder, the Hamming distance between the (possibly erroneous) redundant part of y and the redundant part $v = v(u, y)$ associated with u and y is of Hamming weight $\leq \tau$.*

IV. A power-efficient single error correcting coding scheme

A. PESEC code structure

When implementing a single error correcting code, it is natural to start with the classic (primitive) Hamming code. This code is a linear code of length $2^m - 1$, dimension $2^m - 1 - m$, and minimum distance $d = 3$, and it is defined by an $m \times (2^m - 1)$ check matrix, H, whose columns are the $2^m - 1$ nonzero m-bit vectors. A codeword, c belongs to the code if and only if it satisfies $Hc^{\mathrm{T}} = 0$. (Please note that when performing operations using check matrices, all operations are performed modulo 2.)

In some cases there is no need for a full-length Hamming code, and the code can be shortened by deleting columns from the check matrix. A check matrix for a shortened code of length n is denoted by $H_{m\times n}$, where $2^{m-1} \leq n < 2^m - 1$. When it is clear from the context that we are referring to a shortened Hamming code, we omit the word "shortened."

When shortening a full length, conventional Hamming code, whose check matrix is $H_{m\times 2^m-1}$, the columns chosen and their order in the check matrix of the shortened code are generally unimportant. However, when dealing with systematic, power efficient coding schemes, the choice of columns and their order are crucial.

We are interested in designing a code for which each k-bit information word is associated with multiple representations. To this end, we divide the 2^{n-m} codewords of the Hamming code $\mathcal{C}$ into 2^k disjoint subsets in such a way that each information word u is associated with the subset

$$\mathcal{C}_u = \{c = (u, v) \ : c \in \mathcal{C}\}, \ |\mathcal{C}_u| = 2^{n-m-k}.$$

To protect u while keeping the power consumption low, we need to choose $r = n - k > m$ specific columns for the right hand side of the check matrix; we choose and order the columns of $H_{m\times n}$ as follows:

Construction 1 (Check matrix for a PESEC code.)**.** *For a given k, τ and m, define $b \equiv m \mod \tau$ and $a = (m - b)/\tau$. Define τ sub-matrices:*

$$D_i = \left(0_{\nu_i\times\mu_{i-1}}, \ H_{\nu_i\times(2^{\nu_i}-1)} \ , \ 0_{\nu_i\times(r-\mu_i)}\right), 1 \leq i \leq \tau,$$

$$\begin{aligned}\nu_i &= \begin{cases} a+1 & 1 \le i \le b \\ a & b < i \le \tau \end{cases} \\ \mu_i &= \begin{cases} 0 & i = 0 \\ \sum_{j=1}^{i} (2^{\nu_i} - 1) & i > 1 \end{cases} \\ r &= \sum_{i=1}^{\tau} (2^{\nu_i} - 1) = (\tau + b)2^a - \tau,\end{aligned}$$

where $H_{\nu_i \times (2^{\nu_i}-1)}$ is the check matrix of a full-length Hamming code. The parity check matrix of the power efficient code is composed of two parts,

$$H_{m\times n} = (A_{m\times k}, D_{m\times r}) = \left(\begin{array}{c|c} A & \begin{matrix} D_1 \\ \vdots \\ D_\tau \end{matrix} \end{array} \right)$$

where the columns of the matrix D form a generating set of radius τ (a set for which sums of not more than τ elements span $\mathbb{Z}_2^m$, see [15]) and the matrix A consists of k columns of the check matrix of a full-length Hamming code, $H_{m\times(2^m-1)}$, that are not included in D.

The following example clarifies this idea.

Example 1. *Assume that $k = 64$ and at most $\tau = 3$ bit transitions are allowed in the redundant part. In order to protect sixty-four bits, a (primitive) Hamming code must have at least $2^7 - 1$ columns: $m \ge 7$. Taking $m = 7$, we find that $m = 7 = 2 \cdot 3 + 1$. That is, $b = 1, a = 2$, and $\nu_1 = 3, \nu_2, \nu_3 = 2$, there are $r = 13$ redundant bits, and, thus, $n = 77$. The check matrix is*

$$\begin{aligned} H_{7\times(64+13)} &= (A_{7\times 64}, D_{7\times 13}) \\ &= \left(\begin{array}{c|ccc} A_{7\times 64} & \begin{matrix} H_{3\times 7} \\ 0 \end{matrix} & \begin{matrix} 0 \\ H_{2\times 3} \end{matrix} & \begin{matrix} 0 \\ 0 \end{matrix} \\ & 0 & 0 & H_{2\times 3} \end{array} \right) \\ H_{3\times 7} &= \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \\ H_{2\times 3} &= \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}, \end{aligned}$$

and the 64 columns of A are taken from the $(2^7 - 1 - 13)$ columns of $H_{7\times 127}$ that are not contained in D.

B. ECC Decoder structure

Since all of the codewords utilized by our PESEC code are codewords of a Hamming code, a conventional ECC-decoder for a Hamming code can be utilized. For the sake of completeness, we provide a brief description of the decoder. A typical decoder is depicted in Figure 2. The input to the decoder is a possibly corrupted codeword, $y = c \oplus e \in \mathbb{Z}_2^n$.

The decoder is composed of two blocks: a syndrome computation block and an error correction block.

- Syndrome computation: In order to reuse the decoder circuit for encoding, we represent y as $y = (y_u, y_v)$ and compute the r-bit syndrome as follows:

$$s = Hy^T = Ay_u^T \oplus Dy_v^T. \tag{1}$$

  This block can be implemented as a XOR network.

- Error correction: The error (if any) is corrected by flipping the erroneous bit of y. Since only the information part is output, only y_u has to be corrected by XORing it with the relevant bits of the logic-decoder output.
  In "standard" Hamming codes, the columns of the check matrix are the binary representations of the numbers from 1 to $2^m - 1$ in increasing order. Because we permute the columns and omit unnecessary columns, it is necessary to "translate" the value of the syndrome to the appropriate bit of the sequence, which necessitates the "routing" shown in Figure 2.

Example 2. *Let $k = 8$ and $\tau = 2$. For these parameters, we find that $m = 4 = 2{\cdot}2$. Consequently, we need $r = 2(2^2-1) = 6$ redundant bits and the check matrix is*

$$\begin{aligned} H_{4\times 14} &= \left(\begin{array}{c|cc} \overset{8,7\ldots1}{A} & \overset{6\ldots4}{\begin{matrix} H_{2\times 3} \\ 0 \end{matrix}} & \overset{3\ldots1}{\begin{matrix} 0 \\ H_{2\times 3} \end{matrix}} \end{array} \right) \qquad (2) \\ &= \left(\begin{array}{cccc|cc} 11 & 11 & 10 & 00 & 011 & 000 \\ 11 & 00 & 01 & 11 & 101 & 000 \\ 10 & 11 & 01 & 10 & 000 & 011 \\ \underbrace{01}_{A_{4\times 8}} & \underbrace{10}_{} & \underbrace{11}_{} & \underbrace{01}_{} & \underbrace{000}_{D_{4\times 6}} & \underbrace{101}_{} \end{array} \right). \end{aligned}$$

(The numbers above the matrix A give the index of the information bit corresponding to the relevant column. The numbers above the second part of the matrix give the index of the redundant bit corresponding to the relevant column.) The corresponding syndrome computation and error correction blocks are depicted in Figure 2.

C. PESEC-Encoder structure

The structure of the PESEC-encoder is similar to the structure of the ECC-decoder; see Figure 3. It has the same syndrome computation block and a set of τ decoders.

The inputs to the PESEC-encoder are $y = (y_u, y_v)$ and u, and it generates a transition vector $T = (T_u, T_v)$. The part of T that corresponds to the information part, is computed by XORing u and y_u: $T_u = u \oplus y_u$.

The part of T that corresponds to the redundant part, T_v, is computed as follows. First, the m-bit syndrome, S, is calculated:

$$S = Au^T \oplus Dy_v^T. \tag{3}$$

This vector is composed of τ parts, $S = (S_\tau, \ldots S_1)$. Then, each part enters a logic decoder. The i^{th} decoder is a ν_i-bit input, 2^{ν_i}-bit output logic circuit. Its input is S_i and its output is denoted by $D_i = (d_{i,0}, d_{i,1}, \ldots d_{i,2_i^\nu - 1})$. We use all its output bits except $d_{i,0}$, the output associated with the input value $S_i = 0$, to form the i^{th} part of the T_v. Thus, at most τ bits of T_v are ones.

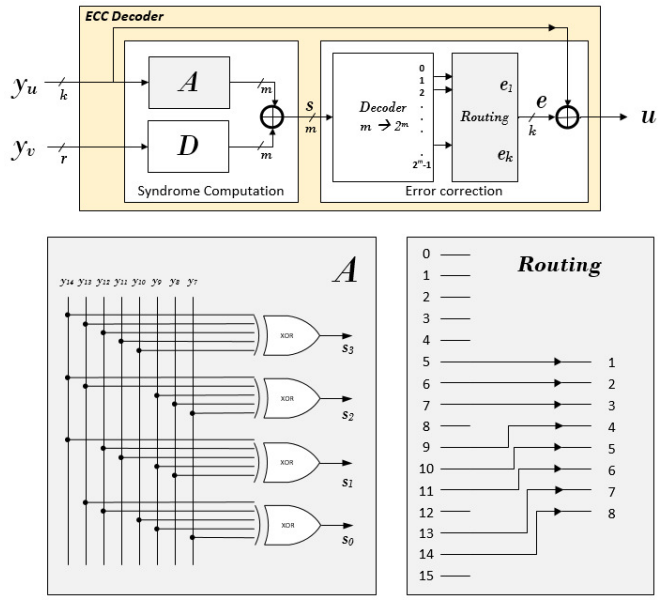

Fig. 2. ECC decoder. Blocks A and D are XOR networks implementing Ay_u^T and Dy_v^T. The lower-left portion is a schematic implementation of multiplication by the matrix A from Example 2, and the lower-right portion depicts the corresponding routing.

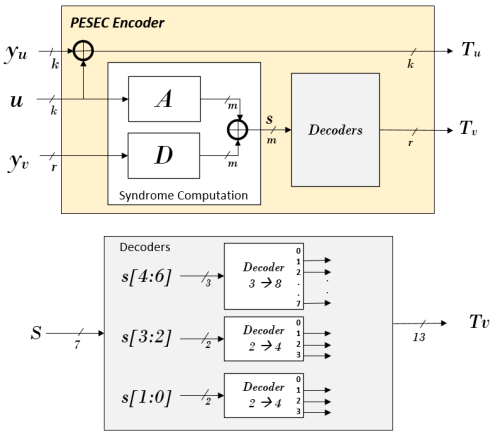

Fig. 3. PESEC encoder. The encoder generates an n-bit transition vector T. The lower box shows a schematic implementation of the logic decoders blocks from Example 1.

Example 3. *The encoder for the code in Example 1 is shown in Figure 3. The syndrome computation block computes a $\tau = 3$ part syndrome, that is*

$$S = (\underbrace{s_6, s_5, s_4}_{S_3}, \underbrace{s_3, s_2}_{S_2}, \underbrace{s_1, s_0}_{S_1}).$$

Each part enters its decoder which outputs the bits of T_v as follows

$$T_v = (\underbrace{d_{3,1}, \ldots d_{3,7}}_{\text{Decoder 3}}, \underbrace{d_{2,1}, d_{2,2}, d_{2,3}}_{\text{Decoder 2}}, \underbrace{d_{1,1}, d_{1,2}, d_{1,3}}_{\text{Decoder 1}}).$$

Remark 2. *Since the check matrix of our PESEC code is a check matrix of a Hamming code with permuted columns, with small changes, our encoder can be made to function as a Hamming code encoder. That is, it can encode a word, say x, of length $k' = n - m > k$ into a Hamming codeword c.*

D. Set and reset of selected bits

The memory is updated by setting or resetting selected bits in order to form the new codeword and, if necessary, to increase the read margin of an unreliable cell. The set and reset operations are performed separately. Denote by SET and RESET the masks indicating the relevant bits for these operations. Their values depend on the PESEC-encoder output.

Two types of errors should be handled during the update procedure:

- **Type I:** A *bit-flip error* in which the value of y_i *stored in the RRAM* is incorrect. The PESEC-encoder corrects this error without exceeding Δ_v bit-flips in the redundancy. To prove this, recall that $T = (T_u = u \oplus y_u, T_v)$, note that by construction T_v has Δ_v ones, and define

  $$c^* = y \oplus T = (y_u, y_v) \oplus (T_u, T_v) = (u, v^*).$$

  Then, the vector c^* is one of the codewords associated with u because $H(c^*)^T = 0$, namely,

  $$\begin{aligned} H(c^*)^T &= Au^T \oplus D(y_v \oplus T_v)^T \\ &= (Au^T \oplus Dy_v^T) \oplus DT_v^T \\ &= (Au^T \oplus Dy_v^T) \oplus S \\ &= (Au^T \oplus Dy_v^T) \oplus (Au^T \oplus Dy_v^T) = 0. \end{aligned}$$

- **Type II:** A *read error* in which the value of y_i read from the (between two writes) is random. In this case, we write in order to increase the read margin. This can be done by modifying the masks if an error has been detected by the ECC-decoder. (See below.)

The following SET and RESET masks update the memory and correct both types of errors:

$$\begin{aligned} SET_i &= T_i\bar{y}_i \vee \underbrace{\bar{T}_i y_i e_i}_{\text{for Type II}} \quad (= c_i(\bar{y}_i \vee e_i)) \quad (4) \\ RESET_i &= T_i y_i \vee \underbrace{\bar{T}_i \bar{y}_i e_i}_{\text{for Type II}} \quad (= \bar{c}_i(y_i \vee e_i)). \end{aligned}$$

Because we have no way to distinguish between Type I and Type II errors, if the designer chooses to use the correction term, it will be applied whenever y_i is in error. If necessary, the application of the correction term in Eq. 4 can be postponed (to avoid exceeding instantaneous power limitations, for example).

V. Evaluation and comparison with other solutions

In this section we compare our PESEC coding scheme to the nested-codes coding scheme presented in [13]. This coding scheme is the best known solution to the problem of writing to a memory in a systematic manner when the *number of bit transitions must not exceed* a certain threshold. We compare the code rates and implementation complexities of

TABLE I
A COMPARISON OF LWSEC, N-BCH AND PESEC CODES

m_b	n	LWSEC code	N-BCH codes [13]	PESEC code				
				n'	k	Δ_u	Δ_v	Δ
16	63	10	≥ 7	57	43	5	2	7
	127	8	≥ 5	82	60	4	2	6
32	63	22	≥ 13	47	33	16	2	18
	127	16	≥ 11	125	103	9	2	11
64	255	28	≥ 19	253	223	17	2	19
Avr.	100%	138.2%		85.4%				100%

TABLE II
AVERAGE AND MAXIMAL NUMBER OF BIT TRANSITIONS

k	n	Δ_u	Δ_v	Δ	$P_e = 0$		$P_e = 1$	
					max	avr	max	avr
32	46	2	2	4	4	3.65	5	4.26
		4	2	6	6	5.60	7	6.13
		8	2	10	10	9.36	11	9.72
		16	2	12	18	15.79	19	15.88
	41	8	3	11	11	9.85	12	10.26
		16	3	19	19	16.28	20	16.38
64	86	2	2	4	4	3.75	5	4.46
		4	2	6	6	5.74	7	6.40
		8	2	10	10	9.66	11	10.22
		16	2	18	18	17.37	19	17.76
	77	8	3	11	11	10.22	12	10.85
		16	3	19	19	17.94	20	18.37

our scheme and the nested coding scheme. Finally, we present simulation results showing the maximal and average number of bit-transitions in different scenarios.

A. Efficiency

Using the notion of partitioned linear codes, Heegard introduced a (semi) systematic nested coding scheme [13]. This scheme inherently combines a bus-encoder with an ECC-encoder, and these two functions cannot be separated. Consequently, it is not suitable for use as an ECC-coder in an RRAM macro (and, in addition, they have very high logic complexity). The same is true of limited weight SEC (LWSEC) codes, [5], codes that consist of all the codewords of length n in a conventional Hamming code whose weight is less or equal to $\Delta/2$ [16, §5.2].

Table I compares PESEC codes with LWSEC codes and with nested BCH (N-BCH) codes. The value in the first column of the table, m_b, is the number of bits to be input to the system, and $n = 2^m - 1$ is the length of the N-BCH code derived from BCH codes of length n and dimension less than $n - (m + m_b)$ [13]. The LWSEC code column give the maximal number of transitions for LWSEC codes, and the N-BCH codes columns uses the super-code lemma [17] to provide a lower bound on the maximal number of transitions. The columns describing the PESEC codes list the code's total length, n', the number of bits output by the bus encoder, k, and the maximum number of bit transitions used by the bus encoder, Δ_u, by the ECC, Δ_v, and by both, $\Delta = \Delta_u + \Delta_v$. Other than when $m_b = 32$ and $\Delta_u = 16$, the bus encoder used in the calculations related to the PESEC codes uses a matrix, D, of dimension $m_b \times k$ whose columns form a generating set of radius τ, as defined in Construction 1.

Considering Table I, we find that for a given m_b and n, where memory widths are required to satisfy $n' \leq n$, the PESEC codes consume somewhat less power than the LWSEC codes The principal advantages of the PESEC codes are, however, the codes' (almost negligible) implementation complexity and the added modularity they provide.

B. Implementation complexity

The nested coding scheme in [13] improves upon earlier solutions by simplifying the *decoder* and unlike other constructions, the code in [13] is systematic. However, the encoding complexity remains high: In addition to the syndrome calculation block, it requires a lookup table with 2^{m_b} entries. Our solution eliminates the enormous lookup table and instead uses τ small decoder circuits. For example, to support $m_b = 32$ and a memory width of $n = 63$, a nested-code-based encoder would require a lookup table with 2^{32} entries, while our PESEC encoder only requires three small logic decoders (one $3 \to 2^3$ decoder and two $2 \to 2^2$ decoders), as shown in the lower portion of Figure 3.

C. Performance

Table II presents simulation results based on one million write operations. The input to the PESEC encoder was a k-bit vector generated by a bus encoder with uniformly distributed inputs. For a given budget of $\Delta = \Delta_u + \Delta_v$, the table reports the total number of bit transitions. The maximum number of bit transitions is denoted by $BT_{\max}$, and the average by BT_{avr}. Two scenarios were simulated: in the first, y was error-free ($P_e = 0$), and in the second, y was corrupted by a single random error ($P_e = 1$). Since we updated the memory contents with the masks defined in Eq. 4, $BT_{\text{avr}}(P_e)$, the average number of transitions when the probability of a single error is P_e, equals

$$BT_{\text{avr}}(P_e) = P_e \cdot BT_{\text{avr}}(1) + (1 - P_e) \cdot BT_{\text{avr}}(0).$$

VI. CONCLUSIONS AND FUTURE WORK

In applications for which the instantaneous power consumption is limited, using RRAM can be challenging due to its high power requirements for writing. We propose a low-overhead, systematic, single-error-correcting, power-efficient coding scheme for RRAM macros that is compatible with any bus encoder. This scheme utilizes a modified Hamming code and is more efficient than and less complex than the system utilizing nested codes presented in [13] and the approach presented in [14].

In future work, we aim to extend these ideas to more robust codes with greater error-correction capabilities, implement the coding scheme, and measure the power consumption of the module.

REFERENCES

[1] F. Zahoor, T. Z. A. Zulkifli, and F. A. Khanday, "Resistive Random Access Memory (RRAM): an overview of materials, switching mechanism, performance, multilevel cell (mlc) storage, modeling, and applications," *Nanoscale Research Letters*, vol. 15, 2020.

[2] M. Hellenbrand, I. Teck, and J. L. MacManus-Driscoll, "Progress of emerging non-volatile memory technologies in industry," *MRS Communications*, vol. 14, no. 6, pp. 1099–1112, 2024.

[3] B. Giraud, S. Ricavy, C. Laffond, I. Sever, V. Gherman, F. Lepin, M. Diallo, K. Zenati, S. Dumas, O. Guille, M. Vershkov, A. Bricalli, G. Piccolboni, J.-P. Noel, A. Samir, G. Pillonnet, Y. Thonnart, and G. Molas, "Smart write algorithm to enhance performances and reliability of an RRAM macro," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 9, pp. 3045–3057, 2024.

[4] A. Lele, S. Jandhyala, S. Gangurde, V. Singh, S. Subramoney, and U. Ganguly, "Disrupting low-write-energy vs. fast-read dilemma in RRAM to enable L1 instruction cache," in *VLSI Design and Test* (A. P. Shah, S. Dasgupta, A. Darji, and J. Tudu, eds.), (Cham), pp. 499–512, Springer Nature Switzerland, 2022.

[5] S. Mittal and S. Nag, "A survey of encoding techniques for reducing data-movement energy," *Journal of Systems Architecture*, vol. 97, pp. 373–396, 2019.

[6] W.-C. Cheng and M. Pedram, "Memory bus encoding for low power: a tutorial," in *Proc. of the IEEE 2001. 2nd Int. Symp. on Quality Electronic Design*, pp. 199–204, 2001.

[7] P. Stanley-Marbell, A. Alaghi, M. Carbin, E. Darulova, L. Dolecek, A. Gerstlauer, G. Gillani, D. Jevdjic, T. Moreau, M. Cacciotti, A. Daglis, N. E. Jerger, B. Falsafi, S. Misailovic, A. Sampson, and D. Zufferey, "Exploiting errors for efficiency: A survey from circuits to applications," *ACM Comput. Surv.*, vol. 53, June 2020.

[8] A. Sarman, A. Shaju, R. G. Kunthara, N. K, R. K. James, and J. Jose, "RIBiT: Reduced intra-flit bit transitions for bufferless NoC," in *2022 IFIP/IEEE 30th Int. Conf. on Very Large Scale Integration (VLSI-SoC)*, pp. 1–6, 2022.

[9] Y. Aghaghiri, F. Fallah, and M. Pedram, "Irredundant address bus encoding for low power," in *Proc. of the 2001 Int. Symp. on Low Power Electronics and Design*, ISLPED '01, (New York, NY, USA), p. 182–187, Association for Computing Machinery, 2001.

[10] S. Komatsu and M. Fujita, "Irredundant address bus encoding techniques based on adaptive codebooks for low power," in *Proc. of the 2003 Asia and South Pacific Design Automation Conf.*, ASP-DAC '03, (New York, NY, USA), p. 9–14, Association for Computing Machinery, 2003.

[11] R. Ahlswede and Z. Zhang, "Coding for write-efficient memory," *Inf. Comput.*, vol. 83, no. 1, pp. 80–97, 1989.

[12] F.-W. Fu and R. Yeung, "On the capacity and error-correcting codes of write-efficient memories," *IEEE Trans. on Information Theory*, vol. 46, no. 7, pp. 2299–2314, 2000.

[13] C. Heegard, "Partitioned linear block codes for computer memory with 'stuck-at' defects," *IEEE Transactions on Information Theory*, vol. 29, no. 6, pp. 831–842, 1983.

[14] S. Engelberg and O. Keren, "Hardening bus-encoders with power-aware single error correcting codes," in *IEEE European Test Symposium, ETS 2024, The Hague, Netherlands, May 20-24, 2024*, pp. 1–4, IEEE, 2024.

[15] R. Brualdi, V. Pless, and R. Wilson, "Short codes with a given covering radius," *IEEE Transactions on Information Theory*, vol. 35, no. 1, pp. 99–109, 1989.

[16] F. MacWilliams and N. Sloane, *The Theory of Error-correcting Codes*. Mathematical Library, North-Holland Publishing Company, 1977.

[17] G. Cohen, M. Karpovsky, H. Mattson, and J. Schatz, "Covering radius—survey and recent results," *IEEE Trans. on Information Theory*, vol. 31, no. 3, pp. 328–343, 1985.