Embedded ECC Solutions for Emerging Memories (PCMs)

M. Ferrari, A. Tomasoni
IEIIT, Consiglio Nazionale delle Ricerche
Via Ponzio 34/5, 20133 Milano, Italy
{marco.ferrari,alessandro.tomasoni}@ieiit.cnr.it

S. Bellini
DEIB, Politecnico di Milano
P.za L. da Vinci, 20133 Milano, Italy
sandro.bellini@polimi.it

P. Amato, M. Sforzin, C. Laurent
Micron Semiconductor Italia s.r.l.
Via Torri Bianche 24, 20871 Vimercate (MB), Italy
{pamato,claurent,msforzin}@micron.com

***Abstract*—Emerging Memories (EMs) could benefit from Error Correcting Codes (ECCs) able to correct few errors in a few nanoseconds. The low latency is necessary to meet the DRAM-like and/or eXecuted-in-Place requirements of Storage Class Memory devices. The error correction capability would help manufacturers to cope with unknown failure mechanisms and to fulfill the market demand for a rapid increase in density. This paper shows the design of an ECC decoder for a shortened BCH code with 256-data-bit page able to correct three errors in less than 3 ns. The tight latency constraint is met by pre-computing the coefficients of carefully chosen Error Locator Polynomials, by optimizing the operations in the Galois Fields and by resorting to a fully parallel combinatorial implementation of the decoder. The latency and the area occupancy are first estimated by the number of elementary gates to traverse, and by the total number of elementary gates of the decoder. Eventually, the implementation of the solution by Synopsys topographical synthesis methodology in $54\,\mathrm{nm}$ logic gate length CMOS technology gives a latency lower than $3\,\mathrm{ns}$ and a total area less than $250 \cdot 10^3\ \mu\mathrm{m}^2$.**

***Index Terms*—Error Correction Codes, Flash Memories, DRAM, Emerging Memories, Storage Class Memories.**

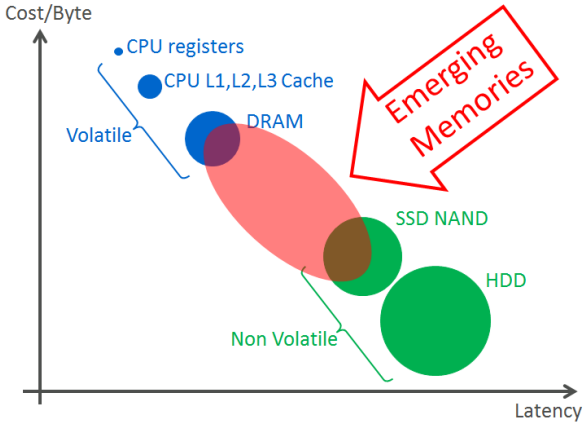

Fig. 1. The gap between Volatile and Non-Volatile memories may be filled by Emerging Memories.

I. Introduction

In the last four decades memory technologies have evolved and consolidated in two mainstreams: DRAM and NAND flash memories. In DRAMs information is stored accumulating electrons in capacitors whereas in NAND flash memories electrons are stored in a floating gate or in an oxide layer in case of *charge trap* NAND. As a consequence, DRAMs have low latency read/write operations, they are volatile and much closer to the CPU in the memory hierarchy. NAND flash memories are high density, non-volatile and more suitable to storage application. Both technologies suffer from the continuous scaling of their electron containers that weakens the memory reliability.

On one side new materials and geometries, such as 3D memories or Cross Point architectures, are investigated to extend the life of DRAM and NAND. On the other, new memory concepts are emerging [1] where the storage mechanisms are different for each type of Emerging Memory (EM). In Phase Change Memory (PCM, [2]) the state is stored in the structure of the material. In metal oxide resistive RAM (Ox-RAM, [3]) the state is stored in the oxygen location. In copper resistive RAM (Cu-RAM, [4]) it is stored in copper location. In Spin Transfer Torque Magnetic RAM (STTMRAM, [5]) it is stored in the electron spin. In Ferroelectric RAM (Fe-RAM, [6]) it is stored in the ion displacement. In correlated electron RAM (Ce-RAM or Mott memories [7]) the state is stored in the resistive state of Mott insulators.

To displace mainstream technologies an EM should show overwhelming advantages. Moreover, EMs based on innovative physical principles are hardly able to reach a high reliability. Yet, emerging technologies can play a fundamental role in improving the memory hierarchy. A classification of consolidated technologies, based on *Cost/byte* and performance (*latency*), shows a wide gap between DRAM and NAND (see Fig. 1). The DRAM *Cost/byte* is about 10 times larger than the NAND one, while the NAND *latency* is 1000 times higher than the DRAM one. The class of storage devices that can fill this gap, referred to as *Storage Class Memories* (SCM) must be both non-volatile/high density and low latency, and offers many opportunities to EMs.

High performance, DRAM-like SCM devices need to be fast and reliable, thus they could benefit from embedded algebraic Error Correction Codes (ECCs) able to correct a few errors just in a few nanoseconds (Section II). DRAM is already adopting binary Hamming codes to correct one error per page [8]. For high performance SCM devices it looks natural to extend to 2, 3, or more bits the protection against errors. For example, in [9] for HfO_X-based resistive memory a bit-error-

rate of $\sim 10^{-8}$ is reported. By applying a BCH3 to such bit-error-rate it is possible to achieve DRAM reliability target. As another example, the authors of [10] describe some techniques to improve STTMRAM reliability such that a triple-error-correcting code is enough to achieve the target block failure rate of 10^{-9}.

The 2-error correction case has already been treated in [11],[12]. In this paper we aim at fast decoding of a 3-error correcting BCH code, with latency in the order of few nanoseconds to fill the gap between DRAM-like and NAND-like memories. BCH codes are already adopted in NAND flash memories, where they are applied to large pages of thousands of data bits, to correct tens of errors. The mild latency constraints of NAND allow traditional BCH decoding. The iterative Berlekamp-Massey (BM) algorithm computes the coefficients of the Error Locator Polynomial (ELP), and the sequential Chien algorithm finds its roots, i.e., the error positions. This classical approach is not compatible with high speed SCM devices. One single iteration of the BM algorithm, implemented even in High-Throughput decoders [13] with error correction capability higher than 2, would require the same latency of the whole decoding process we propose in our design (see [14]). On the other hand, fully parallel solutions for low latency decoding proposed so far such as [15] and [16] cannot guarantee full correction of any 3-bits error pattern. To achieve this challenging latency and error correction target, in this paper all the iterative and sequential processes of the decoding algorithm are replaced by full parallel and combinatorial implementations of the same functions (Section III). When processing the syndromes of the code, great care is to be taken to avoid time-consuming operations in the Galois Field (GF). A careful optimization of all aspects that can speed up the decoding process is carried on across all stages of the decoding algorithm. The solution shown in this paper (Section IV) works for any codeword size, but for the sake of clarity the discussion is focused on the management of a 256-data-bit page.

II. Code definition and shortening

The ECC error correction capability t directly impacts the device reliability. In memory jargon, the fraction of bits that contains incorrect data before applying ECC is called the *raw bit error rate* (RBER), while the error rate after applying ECC is called the *uncorrectable bit error rate* (UBER). Instantaneous UBER is defined as the probability that a codeword will fail, divided by the number of user bits in the codeword. For instance, Fig. 2 shows UBER as a function of RBER for $t = 1, 2, 3$, applying the ECC to a 256-data-bit page. If the target UBER is 10^{-15}, an RBER equal to $6 \cdot 10^{-6}$ can be tolerated with $t = 3$ that is our goal. This RBER is one decade higher than that with $t = 2$, and three decades high than that with $t = 1$. At RBER $= 10^{-8}$ each additional unity of t reduces UBER by six decades.

The minimum Hamming distance of a binary code $\mathcal{C}$ able to correct $t = 3$ errors is $d = 7$. The generator polynomial of a BCH code designed in $GF(2^m)$ with distance 7 has 6

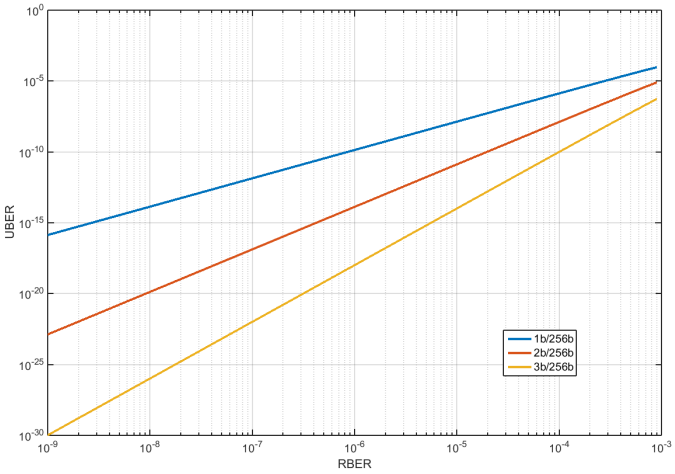

Fig. 2. UBER as function of RBER for 256-data-bit page ECC and different correction capabilities.

consecutive roots in the exponential representation of α^i, plus the conjugate roots [17]

$$g(x) = \prod_{i=0}^{m-1} \left(x - \alpha^{2^i}\right)\left(x - \alpha^{3\cdot 2^i}\right)\left(x - \alpha^{5\cdot 2^i}\right). \quad (1)$$

To encode a data page of $K = 256$ bits, we need a BCH code designed in a larger field, for example in $GF(2^9) = GF(512)$. The final code $\mathcal{C}$ can be obtained by shortening a primitive code of length $N' = 511$, with $N' - K' = mt = 27$ parity bits, $K' = 484$ information bits and generator polynomial

$$g(x) = x^{27} + x^{26} + x^{24} + x^{22} + x^{21} + x^{16} + x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1.$$

The shortened $(283, 256)$ BCH code is obtained deleting 228 selected data bits. In the following $i_n \in \{0, 1, \ldots, 510\}$, with $n \in \{0, 1, \ldots, 282\}$, indicates the degree of the nth element of the shortened codewords. The choice of 256 data bits out of 484 gives many degrees of freedom that can be spent to achieve particular features of the final code. For instance, in the original parity check matrix H' of size 27×511, the 256 information positions can be chosen so that one parity bit is always zero. Therefore it is possible to reduce also the number of parity check bits. This feature has the highest priority in our design, to save memory for the parity bits. The final code $\mathcal{C}$ has only 26 parity bits, i.e. it is a shortened $(282, 256)$ BCH code. The choice of the shortened parity bit and data positions still leaves a residual degree of freedom that is spent to speed up the computation latency of one block, for the reasons explained later in Section IV.

III. Ultra fast decoding of BCH codes

To minimize the decoding latency, the decoder has been carefully designed. The iterative BM algorithm, usually implemented in NAND memories with ECC of high error correction capability, is not affordable with tight latency constraints. Instead, it can be replaced by the parallel evaluation of the ELP

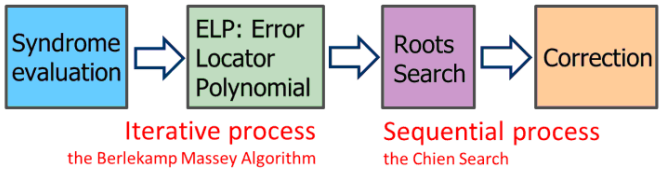

Fig. 3. Block diagram of the standard BCH decoding process.

symbolic expressions, selected through a decision tree [18]. It is important to deal with a limited number of ELP expressions. Any time consuming computation must be avoided whenever possible. Complex evaluations must be carried in parallel with the other terms, to avoid bottlenecks in the decoder.

The decoding algorithm is composed of the four classical stages shown in Fig. 3, namely syndrome evaluation, ELP computation, exhaustive search of the roots of the ELP, and final correction. The latency of these four steps is carefully optimized, both independently and jointly, as outlined in the next subsections.

A. Syndrome evaluation

Starting from the received sequence $\mathbf{y} = y_{i_0} \dots y_{i_{281}}$, with $y_{i_n} \in GF(2)$, the syndromes $S_b \in GF(2^9)$, with $b \in \{1, 3, 5\}$, can be computed by

$$S_b = y(\alpha^b) = \sum_{n=0}^{281} y_{i_n} \alpha^{bi_n}.$$

These operations can be rewritten in matrix form as

$$S_b = \mathbf{y} W_b \tag{2}$$

where each W_b is a 282×9 binary matrix whose rows are given by the polynomial representations of α^{bi_n} with $n = 0 \dots 281$. The three W_b matrices can be combined in a single 282×27 matrix $W = [W_1 W_3 W_5]$.

As no column of W_b has Hamming weight w larger than 256, the depth of the logic computing (2) is $\lceil \log_2(w) \rceil = 8$. The total latency of this stage is thus $8T_X$ where T_X is the latency of a single XOR gate[1].

B. Error Locator Polynomial analysis

When three errors occur ($\nu = 3$), say in positions n_1, n_2, n_3, the ELP with roots in α^{-n_i} reads

$$\Lambda(x) = (1 - x\alpha^{n_1})(1 - x\alpha^{n_2})(1 - x\alpha^{n_3}) = \\ = 1 + \Lambda_1 x + \Lambda_2 x^2 + \Lambda_3 x^3.$$

The coefficients of $\Lambda(x)$, evaluated running the BM algorithm in symbolic form, are [18]

$$\Lambda_1 = S_1, \ \Lambda_2 = \frac{S_5 + S_1^2 S_3}{S_3 + S_1^3}, \ \Lambda_3 = \frac{S_5 S_1 + S_1^6 + S_3^2 + S_1^3 S_3}{S_3 + S_1^3}$$

[1]Unless explicitly specified, gates have only 2 inputs.

and can be computed once S_1, S_3, S_5 are available. Among the elementary operations the division is too demanding, and must be avoided. In case of three errors $S_3 + S_1^3 \neq 0$. An ELP with the same roots α^{-n_i} is

$$\Lambda(x) = A + Bx + Cx^2 + Dx^3 \tag{3}$$

where

$$\begin{aligned} A &\triangleq S_3 + S_1^3 \\ B &\triangleq S_1 A = S_1 S_3 + S_1^4 \\ C &\triangleq S_5 + S_1^2 S_3 \\ D &\triangleq S_5 S_1 + S_1^6 + S_3^2 + S_1^3 S_3. \end{aligned} \tag{4}$$

This form requires no divisions.

If $S_3 + S_1^3 \neq 0$ but $S_5 S_1 + S_1^6 + S_3^2 + S_1^3 S_3 = 0$ we are in the case of two errors ($\nu = 2$). The symbolic BM algorithm delivers the ELP (multiplied by S_1)

$$\Lambda(x) = A' + B'x + C'x^2 \tag{5}$$

where

$$A' \triangleq S_1, \ B' \triangleq S_1^2, \ C' \triangleq S_3 + S_1^3. \tag{6}$$

The choice between two different ELPs (3) and (5) is driven by the value of D, whose computation is the most time consuming (as will become clear at the end of the section). This increases the decoding latency.

If $D = 0$ and also $C' = A = 0$, the ELP computed by the BM algorithm has degree one and reads

$$\Lambda(x) = A'' + B''x \tag{7}$$

with $A'' = 1$ e $B'' = S_1$. Finally, if also $S_1 = 0$ we are in the error free case ($S_1 = S_3 = S_5 = 0$). Note that the ELP (5) reduces to (7) when $S_3 + S_1^3 = 0$, whereas the error free case requires a separate test because if also $S_1 = 0$, the ELP (5) is null for any x.

Following [18] we thus need three different ELPs, whose choice is driven by the most time-consuming term D.

C. ELPs from the Key equation

An alternative to the BM algorithm is the symbolic solution of the Key Equation. The same ELP is obtained when $\nu = t = 3$. If $\nu < 3$ we have more equations than unknowns, and we can pick arbitrarily ν of them.

In particular, let $D = 0$. Then $\nu < 3$, and the unknowns Λ_1 and Λ_2 obey four equations

$$S_5\Lambda_1 + S_4\Lambda_2 = S_6 \tag{8}$$
$$S_4\Lambda_1 + S_3\Lambda_2 = S_5 \tag{9}$$
$$S_3\Lambda_1 + S_2\Lambda_2 = S_4 \tag{10}$$
$$S_2\Lambda_1 + S_1\Lambda_2 = S_3. \tag{11}$$

From (10) and (11) we get $\Lambda_1 = S_1$. From (9) and (11) we obtain

$$\Lambda_2 = \frac{S_5 + S_1^2 S_3}{S_3 + S_1^3}$$

that is the same Λ_2 of (3). The two expressions of $\Lambda(x)$ match when $\nu = 2$. In fact, let $\alpha^{n_1} = u$ and $\alpha^{n_2} = v$. We have

$$S_1 = u + v, \ S_3 + S_1^3 = uv(u + v), \ S_5 + S_1^2 S_3 = u^2 v^2 (u + v)$$

and finally

$$\frac{S_3 + S_1^3}{S_1} = uv = \frac{S_5 + S_1^2 S_3}{S_3 + S_1^3}.$$

Therefore (3)-(4) hold both in case of two or three errors. When $D = 0$ the coefficients A, B, C of (4) correctly identify the two error positions.

From (9) and (10) we can infer the condition for $\nu = 2$ that reads

$$S_4 S_2 + S_3^2 \neq 0 \quad \Leftrightarrow \quad S_1^6 + S_3^2 \neq 0 \quad \Leftrightarrow \quad S_3 + S_1^3 \neq 0$$

and thus the condition for $\nu = 2$ or 3 (and for the use of the ELP (3)) is simply $A \neq 0$. This is a great advantage because the computation of A is much faster than the computation of D, as we will show in Section IV.

When $A = 0$ and $S_1 \neq 0$, we have $\nu \leq 1$ and five equations for Λ_1. They all lead to

$$\Lambda_1 = S_1.$$

Finally, when $S_1 = 0$ the sequence is error free, and the correction must be disabled.

In conclusion, the decision tree has just two ELPs to choose from:

- when $A \neq 0$, $\Lambda(x) = A + Bx + Cx^2 + Dx^3$, with A, B, C, D given in (4)
- when $A = 0$, $\Lambda(x) = 1 + S_1 x$, that includes as a special case the ELP with no roots $\Lambda(x) = 1$ when $S_1 = 0$.

D. Exhaustive Search of the ELP roots

The fastest search for the ELP roots is the parallel test $\Lambda\left(\alpha^{-i_n}\right) = 0$, $\forall i_n$. When $\nu \geq 2$ ($A \neq 0$), the test can be run checking the expression

$$A\alpha^{3i} + B\alpha^{2i} + C\alpha^i + D = 0 \tag{12}$$

which is obtained multiplying each side of the equation $\Lambda\left(\alpha^{-i}\right) = 0$ by α^{3i}.

Since the evaluation of D is the most time-consuming, expression (12) enables the fastest test. The computation of $A\alpha^{3i}, B\alpha^{2i}$ and $C\alpha^i$ can be carried in parallel with D as soon as A, B and C get available.

In the simplified ELP case of $\nu \leq 1$ ($A = 0$), the search can be run by the same hardware. We simply replace the coefficient C with 1 and D with S_1, to obtain the correct ELP. In fact, in this case, $A = 0$, and $B = S_1 A = 0$.

IV. Decoder architecture

In this Section we analyze the overall architecture of the proposed decoder shown in Fig. 4.

1) Syndrome Evaluation: The first step is realized by the blue blocks in Fig. 4. These blocks implement the linear combinations (2) of the received vector $\mathbf{y}$ producing the three 9-bit syndromes $S_1, S_3, S_5 \in GF(512)$. Each block requires about 1200 gates. As anticipated in Section III the longest chain requires $8T_X$ for each block. Hence the values of S_1, S_3, S_5 get available $8T_X$ seconds after reading the data.

2) Error Locator Polynomial: The second step is run in parallel for the four coefficients of the ELP, by the blocks colored with light green in Fig. 4. The fastest to compute is A, that requires a cube and one addition, and takes in our implementation 4 XOR levels and one AND level (see the Appendix). The value of A is thus ready $T_A + 12T_X$ seconds after data reading. The result of the test $A = 0$ is conveyed at the output of the blocks computing C and D, because their values need to be replaced if the test is true. The values of B and C are available T_X seconds later than A (both require a product operation) and are conveyed to the *Root Search* stage. The approximate gate count of each block is in the order of 200 gates for B and C and 150 gates for A. The computation of D requires about 600 gates and a latency $T_A + 10T_X$, including the change $D \to S_1$ if $A = 0$. The various addends of D are carefully combined. S_3^2 is the first term available and it is added to S_1^6 as soon as it is computed. The result gets available at the same time as $S_1 S_3$ and their sum can be inserted during the computation of $S_1^3 S_3$ without latency penalty. The value of D is thus ready $2T_A + 16T_X$ after data reading. No linear combinations of D are needed.

3) Root Search: The root search computes $A\alpha^{3i} + B\alpha^{2i} + C\alpha^i + D$ for each i survived after shortening. This computation requires 9 linear combinations of the bits of A, B and C, for each i. The overall number of possible linear combinations is 511, and we need almost all of them (1.8 kgates for each block). We have a last degree of freedom left in the choice of the data positions surviviving the shortening stage (see Section II). We spend this choice excluding from the linear combinations of one coefficient the sum of all its 9 bits, that would take 4 levels of XOR. We spend this feature on the coefficient B. This way, the linear combinations of B require only 3 XOR levels and get available after $T_A + 16T_X$ from data reading, exactly as the linear combinations of A that require 4 XOR levels. For each position i this choice allows to add the terms coming from A and B at the same time, having the result ready when $C\alpha^i$ comes, T_X seconds later. This is shown for a single position i in the gray cloud in Fig. 4. When the addition $A\alpha^{3i} + B\alpha^{2i} + C\alpha^i$ is ready, T_X seconds later, the value of the coefficient D is also ready, and the overall ELP computation is completed $T_A + 19T_X$ seconds after data reading.

Finally, the ELP root test requires two levels of OR gates with three input (latency T_{3O} each) and the correction $b_i = y_i + e_i$ is completed $T_A + 20T_X + 2T_{3O}$ seconds after data reading.

A first-order approximation of the area occupancy as a function of m comes from the formulae in Table I. These provide an over-estimate of the actual area, because they do not take into account any optimization. In $\mathrm{GF}(512)$ and for a $(282, 256)$ code, the estimates are $1260\,\textsc{Xor}$ for each S_i, $193\,\textsc{Xor} + 36\,\textsc{And}$ for A, $198\,\textsc{Xor} + 81\,\textsc{And}$ for B (and for C too), $715\,\textsc{Xor} + 234\,\textsc{And}$ for D, $1788\,\textsc{Xor}$ for each linear combination, and $8192\,\textsc{Xor}$ for the correction blocks. Through a careful optimization the blocks A, B, C, D can be almost halved even though most of the overall complexity

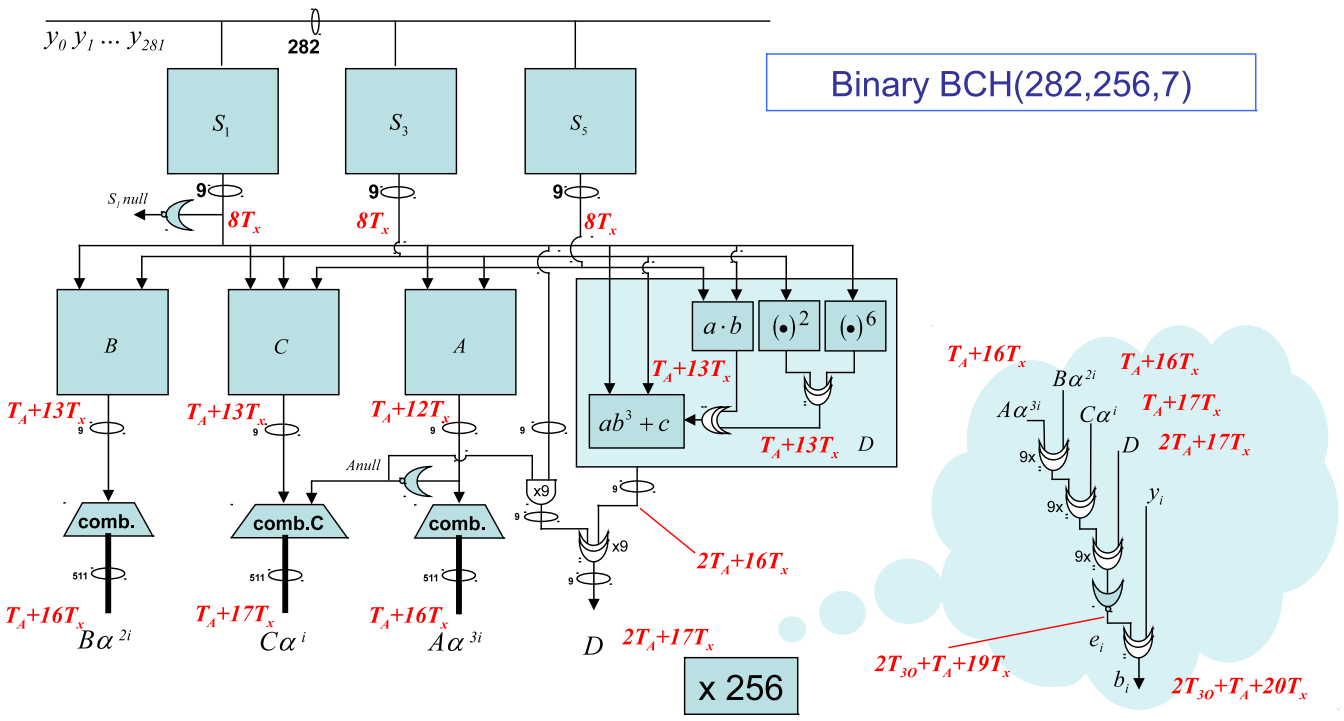

Fig. 4. Conceptual architecture and latency block-by-block of the 3-error BCH Ultra-Fast decoder.

(approximately 18 $kgate$) is concentrated outside these blocks.

TABLE I
THEORETICAL ESTIMATES OF AREA OCCUPANCY FOR THE BCH3 DECODER BLOCKS IMPLEMENTED IN GF(2^m).

Block	Area
S_i	$m\left(\frac{n}{2}-1\right)$ XOR
A	$\frac{1}{4}m(m^2+m-4)$ XOR + $\frac{1}{2}m(m-1)$ AND
B, C	$\frac{1}{2}(5m^2-m)$ XOR + m^2 AND
D	$\frac{1}{2}m^3+5m^2-6m$ XOR + $3m^2-m$ AND
Comb	$(2^m-1)(\frac{m}{2}-1)$ XOR
Correction	$\frac{k}{2}(7m+1)$ XOR

A. Synthesis example

The solution proposed has been implemented following the Synopsys topographical synthesis methodology using a 54 nm logic gate length CMOS technology. In such implementation the decoding latency is smaller than 3 ns, and the area occupancy of the decoder is about $250 \cdot 10^3$ µm^2. Unlike the theoretical estimates in terms of number of elementary gates, these results take into account the buffering of the longer interconnections as well as the routing of signals which may increase the total decoder area occupancy. A trade-off between latency and area occupancy is possible, depending on the specific application.

V. CONCLUSIONS

This paper presents the theoretical derivation and implementation of an Ultra-Fast algebraic BCH decoder with error correction capability $t = 3$ and 256-data-bit page, for emerging memories. Embedding such a decoder into emerging memory devices pushes these technologies towards the market, managing the effects of unknown failure mechanisms that may arise in technologies not yet implemented on a large scale. Although all the description has been given for a 256-data-bit page, the proposed solution is technology independent and can be applied to any block size.

Of course the technology node of the CMOS logic impacts the area and the final latency of the decoder. The solution implemented by Synopsys topographical synthesis methodology in 54 nm logic gate length CMOS technology results in a latency lower than 3 ns with an area occupancy smaller than $250 \cdot 10^3$ µm^2.

APPENDIX. ELEMENTARY OPERATIONS IN $GF(2^m) = GF(512)$

A. Multiplication of two variables

Different structures have been proposed to compute $c = ab$. In this project, we have adopted the Mastrovito multiplier [19], that allows for different latencies of the factors a and b. We can write $c = ab$ as

$$c = ab = a\sum_{j=0}^{m-1} b_j\alpha^j = \sum_{j=0}^{m-1} b_j(a\alpha^j).$$

The products

$$a\alpha^j = \sum_{i=0}^{m-1} a_i^{(j)}\alpha^i, \qquad j = 0, \ldots, m-1$$

can be prepared in advance. The m products and $m-1$ additions in $c_i = \sum_{j=0}^{m-1} b_j a_i^{(j)}$ start when also b is available. The total latency depends on the binary representation of the powers α^j to α^{m-1+j} ($j = 1 \dots m-1$). For instance in GF(512) the maximum latency for the terms $a_i^{(j)}$ is $2T_X$. Some terms require only T_X and can be multiplied in advance by the corresponding b_j (if available). With this precaution, the computation of each bit c_i can be completed within $3T_X$, despite nine addends, and the total latency of the operation is $T_A + 5T_X$ (with a maximum allowed delay T_X for b). Even an additional sum, i.e., $ab + d$ or $ab + d^2$ or even $ab + d^4$, can be completed in parallel during the computation of ab. An equivalent implementation can be done also for $c = a^2 b$. Powers a^{2^k} (see next subsection) are linear combinations of a whose evaluation can be embedded in the terms $a^{2^k}\alpha^j$ with no additional delay.

B. Powers a^n

The computation of powers as $c = a^{2^k}$ is simple, as they require only linear combinations of the bits a_i for each bit c_j. The latency depends on the actual field. For example, in GF(512) α^2 requires one single level of XOR, whereas α^4 requires $2T_X$.

The computation of powers $c = a^n$ with $n \neq 2^k$ is complicated by the fact that non linear terms are required. To minimize the latency we can separate a linear part from a non-linear one as done, e.g., for a^3 in [12]

$$\left(\sum_{i=0}^{m-1} a_i \alpha^i\right)^3 = \sum_{i=0}^{m-1} a_i \alpha^{3i} + \sum_{i=0}^{m-2} \sum_{j=i+1}^{m-1} a_i a_j (\alpha^{2i+j} + \alpha^{i+2j}).$$

By carefully optimizing the sequence and order of sums and products, the computation of a^3 can be completed within $T_A + 4T_X$. A similar implementation, with the same latency, can be done also for a^6.

As to the computation of $c = ab^3$, the minimum latency is achieved computing b^3 as described above, and the terms $a\alpha^j$ ($j = 0 \dots m-1$) in the meantime. The products require a single level of m^2 AND, and the final sum of products can be optimized. In a similar manner, the evaluation of terms like $ab^3 + d + e + \dots$ can be realized without additional latency. Table II summarizes the operations latency.

TABLE II
SUMMARY OF THE COMPUTATION LATENCY (AND ALLOWED DELAYS OF THE VARIOUS TERMS) FOR THE OPERATIONS IN GF(512).

operation	latency	b delay	c delay
a^2	T_X		
a^4	$2T_X$		
$ab + c^2$	$T_A + 5T_X$	T_X	T_X
$a^2 b + c$	$T_A + 5T_X$	T_X	$3T_X$
$a^3 + b$	$T_A + 4T_X$	$3T_X$	
a^6	$T_A + 4T_X$		

REFERENCES

[1] G. Atwood, "Current and emerging memory technology landscape," in *in 2011 Flash Memory Summit*, Santa Clara, CA, Aug. 2011.

[2] C. Villa, D. Mills, G. Barkley, H. Giduturi, S. Schippers, and D. Vimercati, "A 45nm 1Gb 1.8V phase-change memory," in *2010 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, S. Francisco, CA, Feb. 2010, pp. 270–271.

[3] Tz-yi Liu *et al.*, "A 130.7mm2 2-layer 32Gb ReRAM memory device in 24nm technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, S. Francisco, CA, Feb. 2013, pp. 210–211.

[4] W. Otsuka *et al.*, "A 4Mb conductive-bridge resistive memory with 2.3GB/s read-throughput and 216MB/s program-throughput," in *2011 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, S. Francisco, CA, Feb. 2011, pp. 210–211.

[5] K. Tsuchida *et al.*, "A 64Mb MRAM with clamped-reference and adequate-reference schemes," in *2010 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, S. Francisco, CA, Feb. 2010, pp. 258–259.

[6] K. Udayakumar *et al.*, "Low-power ferroelectric random access memory embedded in 180nm analog friendly CMOS technology," in *2013 5th IEEE International Memory Workshop (IMW)*, Monterey, CA, May 2013, pp. 128–131.

[7] N. F. Mott, *Metal-Insulator Transitions, Second Edition*. 4 John St, London, GB: Taylor & Francis, Inc., 1990.

[8] JEDEC, "Low power double data rate 4 (LPDDR4)," Aug. 2014, JESD209-4. [Online]. Available: https://www.jedec.org/

[9] B. Ji *et al.*, "In-line-test of variability and bit-error-rate of hfox-based resistive memory," in *Memory Workshop (IMW), 2015 IEEE International*, May 2015, pp. 1–4.

[10] Y. Emre, C. Yang, K. Sutaria, Y. Cao, and C. Chakrabarti, "Enhancing the reliability of stt-ram through circuit and system level techniques," in *Signal Processing Systems (SiPS), 2012 IEEE Workshop on*, Oct 2012, pp. 125–130.

[11] W. Xueqiang, P. Liyang, W. Dong, H. Chaohong, and Z. Runde, "A high-speed two-cell BCH decoder for error correcting in MLC nor flash memories," *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 56, no. 11, pp. 865–869, Nov. 2009.

[12] P. Amato, C. Laurent, M. Sforzin, S. Bellini, M. Ferrari, and A. Tomasoni, "Ultra fast, two-bit ECC for emerging memories," in *Proc. of 6th IEEE International Memory Workshop (IMW)*, Taipei, Taiwan, May 2014, pp. 79–82.

[13] W. Liu, "Low-Power High-Throughput BCH Error Correction VLSI Design for Multi-Level Cell NAND Flash Memories," in *Proc. of Workshop on Signal Processing Systems Design and Implementation (SIPS)*, Banff, Canada, Oct 2006, pp. 303–308.

[14] D. Strukov, "The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories," in *Proc. of 40th Asilomar Conference on Signals, Systems and Computers (ACSSC)*, Pacific Grove, CA, Oct 29th - Nov 1st 2006, pp. 1183–1187.

[15] X. Wang *et al.*, "An On-Chip High-Speed 4-bit BCH Decoder in MLC NOR Flash Memories," in *Proc. of IEEE Asian Solid-State Circuits Conference*, Taipei, Taiwan, Nov 16-18 2009, pp. 229–232.

[16] C. Badack, T. Kern, and M. Gossel, "Modified DEC BCH codes for parallel correction of 3-bit errors comprising a pair of adjacent errors," in *Proc. of IEEE 20th International On-Line Testing Symposium (IOLTS)*, Platja dAro, Catalunya, Spain, Jul 7-9 2014, pp. 116–121.

[17] S. Lin and D. J. Costello, *Error Control Coding, Second Edition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004.

[18] C. Kraft, "Closed solution of Berlekamp's algorithm for fast decoding of BCH codes," *IEEE Trans. Inf. Theory*, vol. 39, no. 12, pp. 1721 – 1725, Dec. 1991.

[19] E. Mastrovito, "VLSI designs for multiplication over Finite Fields GF(2^m)," *Lecture Notes in Computer Science, Springer-Verlag*, vol. 357, pp. 297–309, Mar. 1989.