BCH Code Based Multiple Bit Error Correction in Finite Field Multiplier Circuits

M. Poolakkaparambil[1], J. Mathew[2], A. M. Jabir[1], Dhiraj K. Pradhan[2] and S. P. Mohanty[3]
[1]Dept. of Comp. Sci. & Electronics, Oxford Brookes University, UK
[2]Department of Computer Science, University of Bristol, UK
[3]Department of Comp. Sci. & Engineering, University of North Texas

***Abstract*— This paper presents a design methodology for multiple bit error detection and correction in Galois field arithmetic circuits such as the bit parallel polynomial basis (PB) multipliers over GF(2^m). These multipliers are crucial in most of the cryptographic hardware designs and hence it is essential to ensure that they are not vulnerable to security threats. Security threats arising from injected soft (transient) faults into a cryptographic circuit can expose the secret information, e.g. the secret key, to an attacker. To prevent such soft or transient fault related attacks, we consider fault tolerance as a method of mitigation. Most of the current fault tolerant schemes are only multiple bit error detectable but not multiple bit error correctable. Keeping this in view, we present a multiple bit error correction scheme based on the BCH codes, with an efficient bit-parallel Chien search module. This paper details the design procedure as well as the hardware implementation specs. Comparison with existing methods demonstrate improved area, and reduced delay performances.**

I. Introduction

Online error detection and correction has received much attention in recent years as a candidate for attack tolerant cryptographic hardware design and to certain extent fault tolerance as well [1]. With low operating voltage levels and low noise margins, digital designs are often vulnerable to various faults [2]. In present day communication devices, cryptographic hardware is an inevitable part. The cryptographic hardware often needs to store some information hidden in order to ensure high degree of information security. The Faults resulting in incorrect output are mainly due to naturally occurring faults or due to malicious attacks. The former can be detected with various testing techniques, whereas the latter cannot be detected with the testing schemes presently available. Clearly, this can result in catastrophe, if undetected [8]. Attacks against such cryptography hardware are often classified into two categories: invasive and non-invasive. The invasive attacks are based on reverse engineering and hence require costly equipment. The non-invasive method, on the other hand, exploits the implementation weakness of the device and is also known as the side channel attacks [3]. It is also noted that, hacking of information within the chip is possible by introducing hardware Trojans within the process variation allowance of the chip [4]. In these types of attacks, the attacker may try to acquire the hidden information by injecting random events, e.g. through transient faults, into the hardware [7]. To keep our information hidden, we need to mask these injected faults and ensure that the devices keep providing the correct information to the external world. Recent research proposes various schemes to overcome such attacks.

The conventional approach for concurrent error detection may be considered as a one-to-one mapping between the output of the functional unit, which is represented as a binary vector of length *k*, and a codeword which is a binary vector of length *n* [13]. Namely, a codeword represents a Boolean expression that is a minterm. The Hamming distance between the codewords must be greater than one, otherwise, it is impossible to detect an error. This restriction increases the implementation cost of the functional unit and the implementation cost of the checker [15]. The most commonly used fault tolerant schemes for single error detection/correction are the error detection and recalculation methods, and the error correcting designs [11] [9]. In the error detection and recalculation scheme, the concurrent error detection circuitry (CED) monitors for an error occurrence and, in case of an error, it rolls backs and recalculate again. The drawback of this approach is that it often increases the time redundancy. Another commonly used approach is the *N-modular redundancy* (NMR). In NMR, the actual circuitry is replicated *N* times and the decision of whether the resultant output is correct is decided by a voting circuitry. The main pitfall of this method is the increase in space redundancy depending upon the number of bit error corrections we need. In [5], a single error correction (SEC) scheme based on the Hamming codes is proposed, while in [10] this method has been extended to the LDPC codes and the method in [6] used to synthesize efficient circuits.

Motivation and Contribution: From the literature review made, it is evident that efficient error correcting designs for Galois field based arithmetic circuits are not fully explored. Our proposed scheme overcomes most of the design drawbacks mentioned above. Firstly, this is the first known approach for bit parallel multiple error correction based on BCH codes for the multipliers over GF(2^m), where as all the existing techniques considered only single bit error. As a part of the design, we also propose an efficient bit parallel implementation of the Chein search module within the error correcting circuitry. Secondly, most of the discussed error correction schemes impose much higher time redundancy where as our proposed multiple bit error correction scheme runs in parallel with the logic, hence requires less time overhead. The added delay is only due to the decoding part of the proposed scheme. In the parity based error correction schemes [1], [5],

the error in the parity blocks can not be detected, where as in the proposed scheme, the errors in the parity blocks are detected as well as the final outputs corrected. We have also derived the closed form expressions for the parity prediction block.

The remainder of this paper is organized as follows. Section II explains the basic fundamentals of the Galois field arithmetic and Polynomial basis multiplication. In Section III, we present the design methodologies of the BCH code based multiple error correction scheme. Section IV presents the experimental results, and finally Section V concludes the paper with future extensions of our research.

II. Preliminaries

For completeness, this section presents the preliminaries of Galois field (GF) arithmetic, which is necessary for both the multiplier as well as the BCH coding scheme.

For every prime number p, there exists a Galois field, also known as the finite fields, over the set GF(p) having p elements with special elements 0 and 1 as the additive and multiplicative identities respectively. It is possible to extend the fields over GF(p) to a field that consists of p^m elements, where m is a nonzero positive integer. This extended field over the set GF(p^m) is known as the extension of the field over GF(p). Let '+' and '$\cdot$' represent the addition and multiplication operations on the field elements. Then GF(p^m) forms a finite field if it forms a commutative ring with identity over these two operations. The finite fields over GF(2), and their extensions over GF(2^m), have particular interest in digital electronics owing to the field elements 0 and 1 only.

The finite fields over GF(2^m) can be generated with monic irreducible polynomials of the form $P(x) = x^{m-1} + \sum_{i=0}^{m-2} c_i.x^i$, where $c_i \in$ GF(2) [14]. Other than elements 0 and 1 the field consists of elements that are multiples of the element α, also known as the primitive element, where α is the root of $P(x)$ i.e. $P(\alpha) = 0$. $P(x)$ is also known as the *primitive polynomial* of the field. To make sure that the operations over the field are finite, any element in the field having power $> 2^{m-1}$ is reduced to an element with power $< 2^{m-1}$ by using the primitive polynomial $P(x)$. The set of elements $\{0, 1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ forms the polynomial basis (PB). Any element $A \in$ GF(2^m) can be represented using the elements in PB. Let $A, B \in$ GF(2^m) with, $A(x) = \sum_{i=0}^{m-1} a_i x^i$, and $B(x) = \sum_{i=0}^{m-1} b_i x^i$, , where $a_i,\ b_i \in$ GF(2). The polynomial basis multiplication of $A(x)$ and $B(x)$ over GF(2^m) is defined as $C(x) = A(x) \cdot B(x) \mod P(x)$. To simplify the classical way of finite field multiplication, Mastrovito proposed an algorithm and equivalent hardware implementation in [16]. Later in [12] an algorithm based on Masterovito's scheme has been presented along with the reduced complexity bit parallel PB multiplier hardware. In our work, we adopt the same multiplier structure as that of [12]. The brief formulation of the PB bit parallel multiplier is explained in the following for the completeness of this paper.

Let A and B are the two multiplicands with $A = [a_0, a_1, a_2, \ldots, a_{m-1}]$ and $B = [b_0, b_1, b_2, \ldots, b_{m-1}]$. The a_is and b_is, where $0 \le i \le m-1$, are the coordinates of A and B respectively. The formulation is based on three matrices namely, an $m \times m$ reduction matrix Q, a lower triangular matrix L and an upper triangular matrix U. The matrix based multiplication is formulated as an inner product (IP) network with two vector outputs $\vec{d}$ and $\vec{e}$ respectively, where,

$$\vec{d} = \mathbf{L}\vec{b} \quad (1)$$

$$\vec{e} = \mathbf{U}\vec{b}, \quad (2)$$

where $\vec{b} = [b_0, b_1, b_2, \ldots, b_{m-1}]^T$, a vector column of the coordinates and x^T represents the x transpose. The matrices **L** and **U** are defined in [12].

The multiplication outputs are given by the equation:

$$\vec{c} = \vec{d} + \mathbf{Q}^T\vec{e}, \quad (3)$$

where the matrix **Q**, which is dependent on the irreducible polynomials, can be derived as shown in [12] and $\vec{c} = [c_0, c_1, c_2, \ldots, c_{m-1}]^T$ is the output bits. For the sake of clarity, we depict a small example below.

Example 1: Let A and B be two multiplicands over GF(2^3) generated with the irreducible polynomial $P(x) = x^3 + x + 1$ with $A = [a_0, a_1, a_2]$ and $B = [b_0, b_1, b_2]$. Then, A and B can also be represented as $A = a_2x^2 + a_1x + a_0$ and $B = b_2x^2 + b_1x + b_0$ in the polynomial form. The product $C(x) = A(x) \cdot B(x)$ mod $P(x)$. Now, $C(x) = (a_2x^2 + a_1x + a_0) \cdot (b_2.x^2 + b_1x + b_0) = (a_2b_2)x^4 + (a_1b_2 + a_2b_1)x^3 + (a_0b_2 + a_1b_1 + a_2b_0)x^2 + (a_0b_1 + a_1b_0)x + (a_0b_0)$. Let us consider the outputs of the IP networks d and e. In the GF(2^3) arithmetic, d and e are column vectors having 3 and 2 elements respectively. Let $d = [d_0, d_1, d_2]$ and $e = [e_0, e_1]$. Then $C(x)$ can be rewritten as,

$$C(x) = e_1x^4 + e_0x^3 + d_2x^2 + + d_1x + d_0. \quad (4)$$

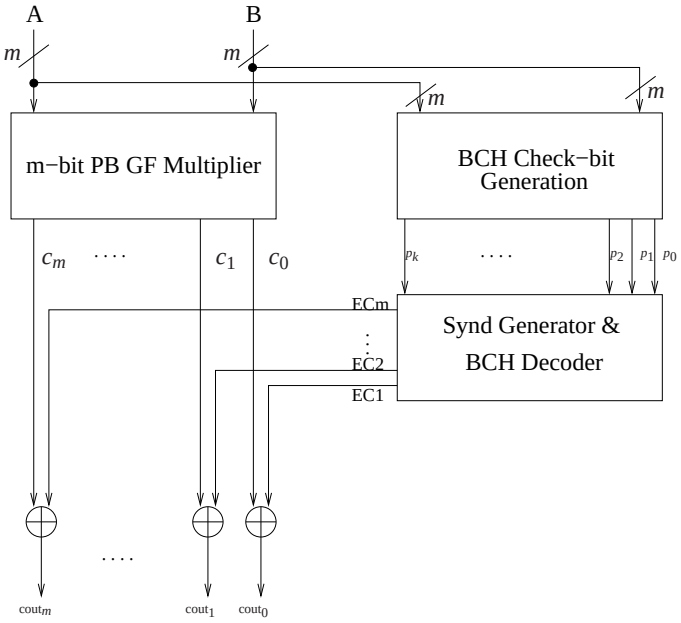

Fig. 1. BCH code based multiple error correction scheme.

It is evident from earlier discussions that the output elements must be closed within GF(2^3). In order to do this, we define the product over $P(x) = x^3 + x + 1$ as $C(x) = A(x) \cdot B(x) \bmod P(x)$. This makes sure that the resulting product terms will be folded back to the elements in GF(2^3). Hence we have, $x^3 = x + 1$, $x^4 = x^2 + x$. Substituting these in Eq. (4) gives, $C(x) = (d_2 + e_1)x^2 + (d_1 + e_0 + e_1)x + (d_0 + e_0)$. The above classical multiplication procedure can be represented in a matrix form using Eq. (3) and, subsequently, the circuit design directly follows from that.

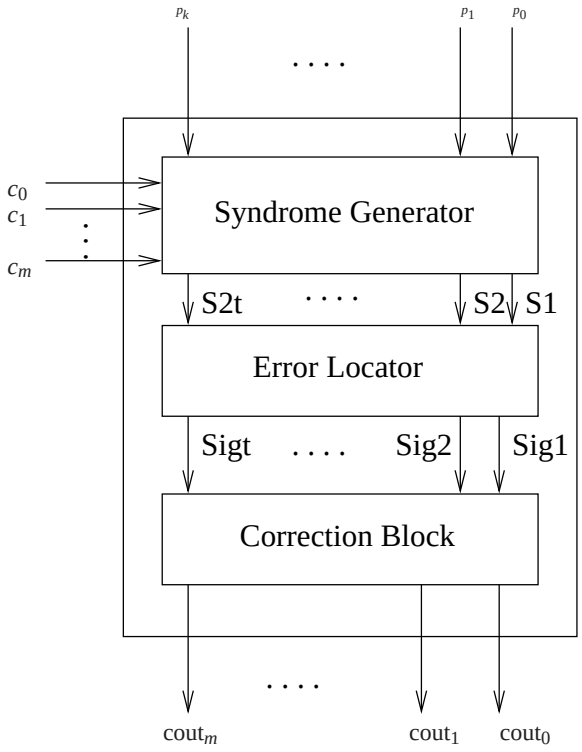

Fig. 2. Syndrome generator and BCH decoder.

III. Multiple Bit Error Correction

A. BCH Code

The Bose-Choudhury-Hocquenghem (BCH) codes belong to the family of cyclic codes in which the message block is encoded using a polynomial $g(x)$, called the *generator polynomial*. The generator polynomial is the least common multiplier (LCM) of the minimal polynomial for the selected powers with respect to GF(2^m), provided that each of the minimal polynomial should appear only once in the product. Here, the message is treated as a whole block and encoded one at a time rather than encoding continuously as in the case of convolution codes. The encoder block possesses no memory, hence no information of the previous message blocks. This style of encoding can be thought of as sliding an encoding window over the message bits. In the conventional BCH codes, the LFSR structure is used to encode incoming message bits one at a time. Hence, the present encoded bit depends on the previous bit, which shows that a memory is being used. In the proposed scheme, we use a parallel BCH encoder which encodes the message as a whole block and uses no memory. The binary BCH codes are generalized Hamming codes and were first proposed by A. Hocquenghem in 1959. In 1960 Bose and Ray Chaudhuri did an independent research and came up with the same idea. Hence the BCH code is named after the three discoverers. BCH codes detect and correct randomly located bit errors in a stream of information bits according to its error correction capability (t). The burst error correcting codes, such as the Reed-Solomon codes, correct multiple errors within a symbol or multiple symbols, but all the bit errors must be within the *same* symbol. The most interesting aspect of the BCH codes over Reed-Solomon codes for our purpose is the simplicity in decoding the codewords. Here, we only need to figure out the bit's location and not the correct value, as in the case of Reed-Solomon codes. The basic block diagram of the generic multiple bit error correction circuit using the binary BCH code is shown in Fig. 1. The overall design contains a parity prediction block, a syndrome generation block, an error-locator polynomial generation block, and a decoder, apart from the bit parallel multiplier circuit.

B. BCH Encoder and Decoder Design

This section details the complete design of a BCH parallel encoder and decoder with an example. The bit parallel multiplier architecture is adopted from [12]. The general representation of BCH code is $BCH(n,k,d)$, where n is the size of the codeword or, in other words, it is the sum of the message length (k), and the number of parity bits (p) used for encoding, and d is the minimum distance (dmin) between the codewords. The possible BCH codes for $m \geq 3$ and $t < 2^{m-1}$ is given by,

$$\text{Block length:} \quad n = 2^{m-1} \tag{5}$$

$$\text{Number of check bits:} \quad n - k \leq mt \tag{6}$$

$$\text{Minimum distance:} \quad d_{\min} \geq 2t + 1 \tag{7}$$

The codeword is formed by adding the remainder after dividing the shifted message block by a special polynomial called the generator polynomial $g(x)$. All the codewords will be a multiple of the generator polynomial. The generator polynomial is not just a minimal primitive polynomial, but a combination of several polynomials corresponding to the powers of the primitive element α in GF(2^m). In other words, $g(x)$ is the least common multiple of the minimal polynomials over the various powers of the primitive element α (powers from $\alpha, \alpha^2, \ldots, \alpha^{2t}$, where t is the error correction capability of the code).

Then,

$$g(x) = \text{lcm}\big(m_1(x), m_2(x), \ldots, m_{2t}(x)\big), \tag{8}$$

where $m_1(x), m_2(x), \ldots, m_{2t}(x)$ are the minimal polynomials corresponding to the various powers of α. It is also noted that

TABLE I
GF(2^4) ELEMENTS IN PB.

GF(2^4) elements	Bit vector
0	0000
1	0001
α	0010
α^2	0100
α^3	1000
α^4	0011
α^5	0110
α^6	1100
α^7	1011
α^8	0101
α^9	1010
α^{10}	0111
α^{11}	1110
α^{12}	0100
α^{13}	1111
α^{14}	1001

every even power of a primitive element has the same minimal polynomial, hence Eq. (8) will be simplified to,

$$g(x) = \mathrm{lcm}\big(m_1(x), m_3(x), \ldots, m_{2t-1}(x)\big). \quad (9)$$

The basic principle and design of the bit-parallel BCH code based multiple error correction scheme is explained with an example as follows. Let us consider a simple case of $BCH(15,5,7)$, where $n = 15$ and $k = 5$. In this fairly small example, we consider bit-parallel PB multiplier over GF(2^5). Let $c = [c_o, c_1, c_2, c_3, c_4]$ be the outputs of the multiplier. Then,

$$\begin{aligned} M(x) &= c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 \quad (10)\\ x^{n-k}M(x) &= x^{n-k}(c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0)\\ &= c_4x^{14} + c_3x^{13} + c_2x^{12} + c_1x^{11} + c_0x^{10}, \quad (11)\end{aligned}$$

since $n = 15$ and $k = 5$ in this case.

The parity check bits are generated by,

$$P(x) = x^{n-k}M(x) \mod g(x). \quad (12)$$

Let α be the primitive element of GF(2^4), as shown in Table I. Here, $P(x) = x^4 + x + 1$ is the primitive polynomial. The three minimal polynomials $m_1(x)$, $m_3(x)$, and $m_5(x)$ are given by,

$$\begin{aligned} m_1(x) &= x^4 + x + 1 \quad (13)\\ m_3(x) &= x^4 + x^3 + x^2 + x + 1 \quad (14)\\ m_5(x) &= x^2 + x + 1. \quad (15)\end{aligned}$$

For three bit error correction $(t = 3)$, the generator polynomial for constructing the codeword is then given by,

$$g(x) = \mathrm{lcm}\big(m_1(x), m_3(x), m_5(x)\big). \quad (16)$$

Substituting values of Eq. (13), Eq. (14) and Eq. (15) in Eq. (16) we get,

$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1 \quad (17)$$

Substituting Eq. (17) in Eq. (12) gives,

$$\begin{aligned} P(x) &= p_9x^9 + p_8x^8 + p_7x^7 + p_6x^6 + p_5x^5 + p_4x^4 + p_3x^3\\ &\quad + p_2x^2 + p_1x^1 + p_0 \quad (18)\end{aligned}$$

where, $p_0 = c_0 + c_2 + c_4$, $p_0 = d_0 + d_2 + d_4 + e_0 + e_1 + e_2 + e_3$, $p_1 = c_0 + c_1 + c_2 + c_3 + c_4$, $p_1 = d_0 + d_1 + d_2 + d_3 + d_4$, $p_2 = c_0 + c_1 + c_3$, $p_2 = d_0 + d_1 + d_3 + e_1 + e_2 + e_3$, $p_3 = c_1 + c_2 + c_4$, $p_3 = d_1 + d_2 + d_4 + e_0 + e_2 + e_3$, $p_4 = c_0 + c_3 + c_4$, $p_4 = d_0 + d_3 + d_4 + e_0 + e_2$, $p_5 = c_0 + c_1 + c_2$, $p_5 = d_0 + d_1 + d_2 + e_2$, $p_6 = c_1 + c_2 + c_3$, $p_6 = d_1 + d_2 + d_3 + e_0 + e_3$, $p_7 = c_2 + c_3 + c_4$, $p_7 = d_2 + d_3 + d_4 + e_1$, $p_8 = c_0 + c_2 + c_3$, $p_8 = d_0 + d_2 + d_3 + e_0 + e_1 + e_3$, $p_9 = c_1 + c_3 + c_4$, $p_9 = d_0 + d_3 + d_4 + e_0 + e_2$.

Hence, the final BCH encoded codeword for the bit parallel GF multiplier circuit is given as,

$$\begin{aligned} E(x) &= c_4x^{14} + c_3x^{13} + c_2x^{12} + c_1x^{11} + c_0x^{10} + p_9x^9\\ &+ p_8x^8 + p_7x^7 + p_6x^6 + p_5x^5 + p_4x^4\\ &+ p_3x^3 + p_2x^2 + p_1x + p_0. \quad (19)\end{aligned}$$

The parity bits (check bits) are generated by a parallel check bit generation unit as shown in Fig. 1. The resulting parity bits along with the multiplier outputs are passed to the syndrome generation blocks as shown in Fig. 2. For three bit error correction capability $(t = 3)$, we need six $(2 \times t)$ syndromes to be generated. The syndromes help us to determine whether the computed multiplication results are error free or not. In case of error free computation, the syndromes will be evaluated to zero. If the syndromes are nonzero, then that flags us the erroneous computation.

The syndromes are calculated as follows,

$$Si(x) = E(x)|_{x=1,\alpha,\ldots,\alpha^{2t}} \quad (20)$$

The syndrome decoding is done by using the well known Peterson-Gorenstein-Zierler algorithm. Peterson noticed that we need only a few of the syndromes to effectively correct the bit errors. In our case for three bit error correction we need to calculate only syndromes $S1$, $S3$, and $S5$. The generalized equation for syndromes for the given example of $BCH(15,5,7)$ are given as follows,

$S1 = s13\alpha^3 + s12\alpha^2 + s11\alpha + s10$, $S3 = s33\alpha^3 + s32\alpha^2 + s31\alpha + s30$, $S5 = s53\alpha^3 + s52\alpha^2 + s51\alpha + s50$, $s10 = c_4 + c_3 + c_2 + c_0 + p_8 + p_7 + p_4 + p_0$, $s11 = c_2 + c_1 + c_0 + p_9 + p_7 + p_5 + p_4 + p_1$, $s12 = c_3 + c_2 + c_1 + c_0 + p_8 + p_6 + p_5 + p_2$, $s13 = c_4 + c_3 + c_2 + c_1 + p_9 + p_7 + p_6 + p_3$, $s30 = c_4 + c_0 + p_9 + p_5 + p_4 + p_0$, $s31 = c_4 + c_3 + p_9 + p_8 + p_4 + p_3$, $s32 = c_4 + c_2 + p_9 + p_7 + p_4 + p_2$, $s33 = c_4 + c_3 + c_2 + c_1 + p_9 + p_8 + p_7 + p_6 + p_4 + p_3 + p_2 + p_1$, $s50 = c_4 + c_2 + c_1 + p_9 + p_8 + p_6 + p_5 + p_3 + p_2 + p_0$, $s51 = c_4 + c_3 + c_1 + c_0 + p_8 + p_7 + p_5 + p_4 + p_2 + p_1$, $s52 = c_4 + c_3 + c_1 + c_0 + p_8 + p_7 + p_5 + p_4 + p_2 + p_1$, $s53 = 0$.

Determining whether the computation is error free is not sufficient, and we also need to correct these errors in case if they are present. For this, we need to compute the error positions or error locations of the erroneous bits. To determine the error positions effectively, we need to decode the syndromes. The syndrome decoding block of the BCH based error correction technique contains an error locator polynomial generator block that finds the root of the error locator polynomial and a decoder that eventually corrects the erroneous bits based on the computed error position. For this purpose computed syndrome values are passed on to the error

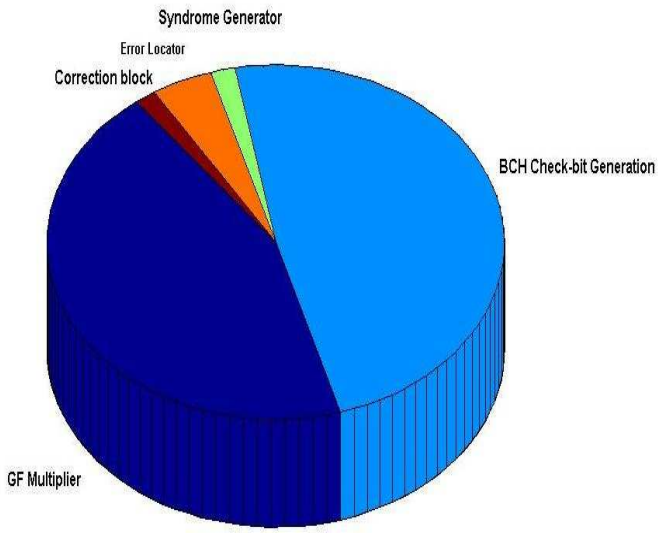

Fig. 3. Block with area analysis of a 45-bit GF multiplier with 3-bit error correction.

locator polynomial computation block, as shown in Fig. 2. For the three $(t = 3)$ bit error correction, we have three $(t = 3)$ coefficients for the error locator polynomial. Let $\sigma 1$, $\sigma 2$, and $\sigma 3$ be the three coefficients of the error locator polynomial.

Then, $\sigma_1 = S1$, $\sigma_2 = \big((S1^2 S3) + S5\big)/(S1^3 + S3)$, $\sigma_3 = (S1^3 + S3 + S1S2)$.

The above three equations give the coefficients $\sigma 1$, $\sigma 2$, and $\sigma 3$ of the error locator polynomial.

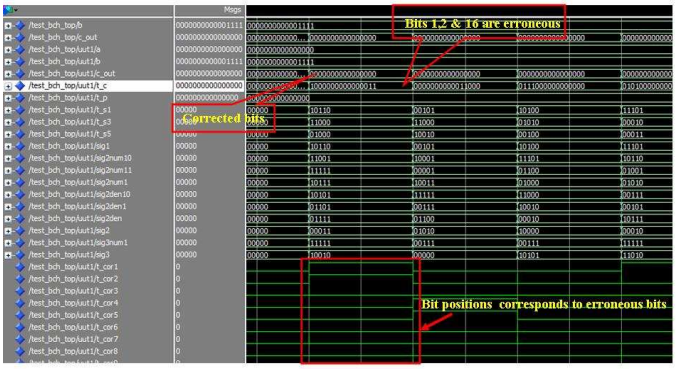

Fig. 4. Modelsim^{TM} simulation results of BCH code based multiple error correction.

Improved Error Locator Design: Once we have the error locator polynomial, the roots of the polynomial will give the error locations. The traditional algorithms for finding the roots of the error locator polynomial are based on exhaustive search methods. A widely known algorithm for finding the roots is the Chien search algorithm, in which all the possible values of the primitive element α, ranging from $\alpha^0, \alpha, \ldots, \alpha^{2m-1}$, are induced into the error locator polynomial to check if they satisfy the polynomial. In the proposed design, a bit parallel implementation of the Chien search algorithm is implemented. In particular, we proposed a scheme in which the root of the error locator polynomial is checked only among the powers of the primitive element α corresponding to the bit positions of the message bits, i.e. the multiplier output bits. The roots of the error locator polynomial corresponding to the parity bits are omitted in order to reduce the hardware complexity. For a 5-bit multiplier, we check whether $\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5$ are roots of the error locator polynomial, which in turn corresponds to the bit positions c_4, c_3, c_2, c_1 and c_0 in the output of the multiplier. In other words, if α is a root of the error locator polynomial, it says that the bit c_4 of the multiplier is erroneous, etc. The decoder corrects the erroneous bit(s) corresponding to the information provided by the parallel root search block. Based on this design principle, we have also extended the design to a 16-bit bit parallel PB multiplier over GF(2^{16}) and to a 45-bit bit parallel PB multiplier over GF(2^{45}).

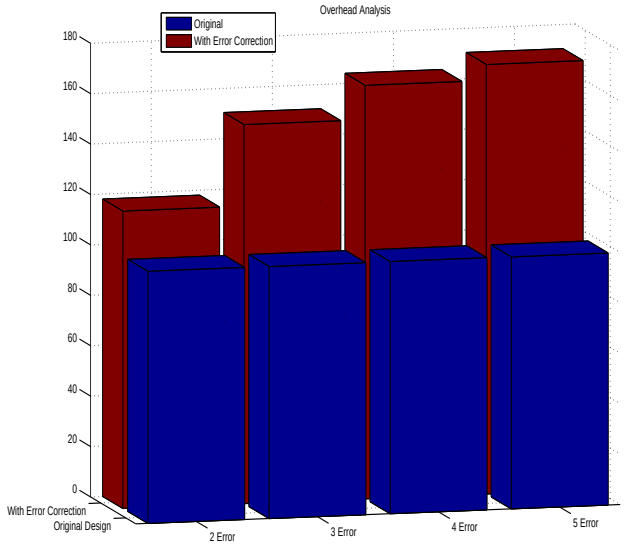

Fig. 5. Overhead analysis of BCH based error correction scheme.

IV. Experimental Results

We have designed the BCH based error correction scheme in VHDL. For simulation and validation of the error correction technique, we have considered 16-bit and 45-bit bit parallel PB multipliers as design examples. Since the error correction logic is independent of the multiplier logic, this scheme can be extended for bit parallel multipliers of any size. The design was simulated using Modelsim^{TM} and was synthesized using

TABLE II

COMPARISON WITH OTHER APPROACHES.

Property	[12]	[10]	Proposed	Proposed	Proposed
#errors correction	single	single	3 Errors	4 Errors	5 Errors
Coding technique	Hamming	LDPC	BCH	BCH	BCH
Overhead	>100%	100%	150.4%	164.04%	170.4%

the Synopsys^{TM} (180nm technology) design compiler. Fig. 5 shows the area overhead for the various designs with 2, 3, 4, and 5 error correction for a 45-bit multiplier. Fig. 3 shows the area of the various blocks in our proposed multiple error correction scheme. Fig. 4 shows the snapshot of a typical Modelsim^{TM} simulation result. During the simulations, we have introduced faults into the multiplier outputs randomly for checking the error correction capability of the proposed scheme. The highlighted parts in Fig. 4 show one among the many testing values. We have introduced errors in the intermediate stages of the multiplier, which in turn gave multiple bit errors at the multiplier output. In this case we have errors at bit positions 1, 2, and 16, however the c_{out} values show the corrected final output from the BCH decoder. Although the example designs considered 2 to 5 bit error correction capability, based on the theory presented in this paper, we can easily extend its capability to more than five bits. The back end process, place and route, is done for a 45-bit GF multiplier with three error correction capability using the Cadence Encounter^{TM} tool set. The final layout of the design is shown in Fig. 6.

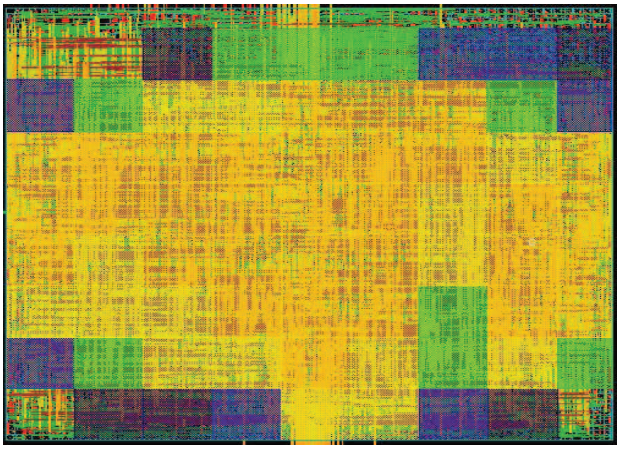

Fig. 6. Layout of 45-bit GF Multiplier with Multiple Error Correction Feature.

Further, for a given error correction capability, the extra hardware comes down significantly for larger and more practical designs. For example, for the 5-, 16-, and 45-bit multipliers we observed that the extra hardware is 600%, 240%, and 150.4% respectively for 3-bit error correction capability.

V. CONCLUSIONS AND FUTURE WORK

This paper proposed a technique for designing bit parallel multiple bit error correctable multipliers over GF(2^m) based on the BCH codes. We also presented an efficient bit parallel structure of the iterative Chien search algorithm for finding the roots of the error locator polynomial, comprising less area and time complexity. The experimental results showed that the proposed scheme has a lower complexity in terms of area and delay compared with the NMR based techniques. Also, compared to SEC techniques the hardware overhead is well within acceptable margins despite its enhanced capability. Future extensions of the proposed work will include fully testable versions of the proposed scheme for higher complexity bit parallel and digit serial architectures. Power and delay specs will be explored with advanced testable designs.

REFERENCES

[1] A. Reyhani-Masoleh, and M. Anwar Hasan, "Fault Detection Architectures for Field Multiplication Using Polynomial Bases", IEEE Trans. Computers, vol. 55, No. 9, Sept. 2006.

[2] Mitra S., Seifert N., Zhang M., Shi Q and Kim K, "Robust System Design with Built-In Soft Error Resilience", IEEE Computer, Vol. 38, Number 2, pp. 43-52, Feb. 2005.

[3] C. R. Moratelli, E. Cota, M. S. Lubaszewski "A Cryptography Core Tolerant to DFA Fault Attacks", Journal Integrated Circuits and Systems, pp-14-21, 2007.

[4] Y. Jin, Y. Makris, "Hardware Torjans in Wireless Cryptographic ICs", IEEE Design & Test Computers, January/February 2010.

[5] J. Mathew, A. M. Jabir, H. Rahaman, D. K. Pradhan, "Single error Correctable Bit Parallel Multipliers over $GF(2^m)$", IET Comput. Digit. Tech., Vol. 3, Iss.3, pp. 281-288,2009.

[6] A. Jabir, D. Pradhan, J. Mathew "GfXpress: A Technique for Synthesis and Optimization of GF(2^m) Polynomials", IEEE Trans. CAD 27(4), 690–711, 2008.

[7] G. B. Ratanpal, R. D. Williams,and T. N. Blalock, "An On-Chip Signal Suppression Countermeasure to Power Analysis Attacks", IEEE Trans. on dependable and secure computing, vol. 1, no. 3, July 2004.

[8] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations", Journal of Cryptology, 14, pp. 101–119, 2001.

[9] M. Ciet and M. Joye. Dueck, "Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults", Designs, Codes and Cryptography, 36(1), July 2005, pp. 33–43.

[10] J. Mathew, J. Singh, A. M. Jabir, M. Hosseinabady and D. K. Pradhan, "Fault Tolerant Bit Parallel Finite Field Multipliers using LDPC Codes", IEEE, 2008.

[11] G. Gaubatz and B. Sunar, "Robust finite field arithmetic for fault-tolerant public-key cryptography", 2nd Workshop on Fault Tolerance and Diagnosis in Cryptography (FTDC), 2005.

[12] A. Reyhani-Masoleh, and M. Anwar Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over GF(2^m)", IEEE Trans. on Computers, Vol.53, No.8, pp.945-959, August 2004.

[13] K. Wu, R. Karri, G. Kuznetsov and M. Goessel: "Low Cost Concurrent Error Detection for the Advanced Encryption Standard", International Test Conference, pp.1242-1248, 2004.

[14] D. K. Pradhan, "A Theory of Galois Switching Functions", IEEE Trans. Computers, vol. 27, no. 3, pp.239-248, Mar. 1978.

[15] O. Keren "One-to-Many: Context-Oriented Code for Concurrent Error Detection" Journal of Electron Test, vol. 26, pp. 337-353, 2010

[16] E. D. Matrovito, "VLSI Achitectures for Computation in Galois Fields." Ph.D. thesis, Linkoping University, Linkoping Sweden, 1999.