Multiple bit Error Detection and Correction in GF Arithmetic Circuits

J. Mathew[1], S. Banerjee[1], Mahesh P.[2], D.K. Pradhan[1], A.M. Jabir[2], S.P. Mohanty[3]
[1]Department of Computer Science, University of Bristol, UK
[2]Dept. of Comp. Sci & Electronics, Oxford Brookes University, UK
[3]University of North Texas, Denton,TX 76203.

***Abstract*— This paper presents a design technique for multiple bit error correctable (fault tolerant) polynomial basis (PB) multipliers over GF(2^m). These multipliers are the building blocks in certain types of cryptographic hardware, e.g. the Elliptic Curve Crypto systems (ECC). One of the drawbacks in the existing techniques is their inability to correct multiple bit errors at the outputs. Also, much attention has been given to error detection, as opposed to error correction. However, owing to possible security threats induced by soft or transient faults in cryptographic hardware, in certain cases multiple bit error correction, as a way of mitigating attacks, is more important than detection. To this end, we use multiple parity predictions to detect multiple errors based on popular error correcting codes. First, we present a multiple error detection scheme using Low Density Parity Check Codes (LDPC). The expressions for the parity prediction are derived from the input operands, and are based on the primitive polynomials of the fields. For multiple bit error correction we use Reed Solomon codes. Comparison with traditional techniques shows improved area and power performance.**

I. Introduction

A variety of techniques have been proposed for increasing the reliability through online error detection, not only for memory, but for logic as well [4]. Furthermore, it has recently been shown that for many digital signature and identification schemes an attacker can inject faults into the hardware and the resulting incorrect outputs can completely expose the secret signatures [1]. Subsequently, [2] presented a technique showing how the presence of faults in the public parameters of an ECC system may expose the secret keys. On chip error error detection and correction, could be one of the options to correct both permanent failures and injected faults.

A number of approaches exists, e.g. [20], [22]. One way to detect errors in finite field multipliers is to use parity prediction techniques [5]. A second approach is to scale the inputs of the multiplier by a factor and at the end of the multiplication, the correctness of the result is checked by one or two divisions [9]. The main techniques that can be used for single error correction are (1) error detection and retry, and (2) error masking. Error detection and retry involves using concurrent error detection (CED) circuitry that monitors the outputs of a circuit for the occurrence of an error. If an error is detected, the system recovers through rollback and retry thereby preventing a failure. Error masking involves using circuitry that masks (i.e. *corrects*) errors using schemes such as the triple modular redundancy (TMR) [8].

In [5] the authors have considered the detection of single stuck at faults in the PB multipliers over GF(2^m). They use simple parity prediction technique for error detection. The main problem with their approach is that for a low complexity bit parallel multiplier the delay overhead is 69.2%. For performance critical applications this delay overhead may be critical. Our technique fundamentally differs from this technique in two critical issues. Firstly, our technique addresses the problem of single error correction. Secondly, and more importantly, since our parity prediction circuit runs parallel with the multiplier, delay penalty comes only in the decoding and correction logic. While most of the previous work provides fault detection techniques, this work proposes error correction techniques and investigates the hardware cost and performance.

Considering the merits and demerits of the existing techniques, this paper presents two aspects of multiple error detection and correction in PB bit-parallel multipliers over GF(2^m). Firstly, we present a multiple error detection scheme based on the LDPC codes, which can detect up to 3 concurrent errors. We also present for the first time a design technique for byte level error detection and correction based on the Reed-Solomon codes. In addition, we derive closed form expressions for parity prediction circuits for the error correction schemes. The area, power and delay overheads for the proposed design techniques are analyzed. Further, the proposed designs can be made C-testable with 100% fault coverage using technique presented in [6].

The remainder of this paper is organized as follows. In Section II, we review the basics of Galois field multiplication. In Section III, we present the proposed error correction scheme based on LDPC and RS codes. Section presents the experimental results.

II. Preliminaries

Let GF(p^m) denote a set of p^m elements, where p is a prime number and m a nonzero positive integer, with two special elements 0 and 1 representing the additive and multiplicative identities respectively. In addition let the two symbols '$+$' and '$\bullet$' represent addition and multiplication operations respectively. Then GF(p^m) forms a finite field, also known as the Galois Field, if it forms a commutative ring with identity over these two operators, in which every element has a multiplicative inverse.

Finite fields over the set 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) [19]. It is conventional to represent the elements of GF(2^m) as a power of the primitive element denoted by α, where α is the root of $P(x)$, i.e. $P(\alpha) = 0$. The set $\{1, \alpha, \alpha^2 \ldots, \alpha^{m-1}\}$ is referred to as the polynomial basis. Each element $A \in$ GF(2^m) can be expressed with respect to the PB as a polynomial of degree m over GF(2), i.e. as $A(x) = \sum_{i=0}^{m-1} a_i x^i$, where $a_i \in$ GF(2). Given any two elements A and B over GF(2^m), the PB multiplication of A and B over GF(2^m) is defined as $W(x) = A(x) \cdot B(x)$ mod $P(x)$. Details can be found in [10], [12]. Mastrovito has proposed an algorithm, along with its hardware architecture, for PB multiplication [12], popularly known as the Mastrovito algorithm/multiplier. In [10], based on the Mastrovito algorithm, a new formulation for PB multiplication and generalized bit-parallel hardware architecture has been presented. Their formulation is summarized below for completeness, which we have used in the rest of the paper.

Consider a multiplier with a and b inputs where $a = [a_0, a_1, a_2, \ldots, a_{m-1}]$ and $b = [b_0, b_1, b_2, \ldots, b_{m-1}]$. The a_i and b_i, where $0 \leq i \leq m-1$, are the coordinates of a and b respectively. The formulation is based on the three matrices: (i) an $m-1$ by m reduction matrix $\mathbf{Q}$, (ii) the $\mathbf{L}$ matrix, and (iii) the $\mathbf{U}$ matrix. The $\mathbf{L}$ and $\mathbf{U}$ matrices are formed for implementation of this multiplication scheme. The $\mathbf{L}$ is a lower triangular matrix and the $\mathbf{U}$ is an upper triangular matrix. The outputs (d's and e's) of the IP-network are defined by the following two vectors, which are functions of A and B.

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

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

where $\vec{b} = [b_0, b_1, b_2, \ldots, b_{m-1}]^T$, a vector column of the coordinates. The matrices $\mathbf{L}$ and $\mathbf{U}$ are defined in [10].

The multiplication outputs are given by the equation:

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

where the matrix $\mathbf{Q}$, which is dependent on the irreducible polynomials, can be derived as shown in [10].

Example 1: Consider a multiplier structure over GF(2^3) defined by the primitive polynomial $P(x) = x^3 + x + 1$.

The two inputs of the multiplier are $A = (a_0, a_1, a_2)$ and $B = (b_0, b_1, b_2)$. The polynomial form of these elements are: $A(x) = a_0 + a_1 x + a_2 x^2$, and $B(x) = b_0 + b_1 x + b_2 x^2$, where $A, B \in$ GF(2^3). The product $C(x) = A(x) \cdot B(x)$.

Now, $C(x) = (a_0 + a_1 x + a_2 x^2) \cdot (b_0 + b_1 x + b_2 x^2) = a_0 b_0 + (a_0 b_1 + a_1 b_0)x + (a_0 b_2 + a_1 b_1 + a_2 b_0)x^2 + (a_1 b_2 + a_2 b_1)x^3 + a_2 b_2 x^4$

Let us denote the lower order 3 coefficients as d_0, d_1, and d_2 and the higher order 2 coefficients as e_0 and e_1. Then, $C(x) = d_0 + d_1 x + d_2 x^2 + e_0 x^3 + e_1 x^4$. Here, we define product over the primitive polynomial $P(x) = x^3 + x + 1$ as $W(x) = A(x) \cdot B(x)$ mod $P(x)$. Hence, we have, $x^3 = x + 1$ and $x^4 = x^2 + x$.

Substituting for x^3 and x^4, and then simplifying we get: $W(x) = C(x) = (d_0 + e_0) + (d_1 + e_0 + e_1)x + (d_2 + e_1)x^2$.

The above polynomial multiplication and modulo reductions can be represented in the matrix form as follows.

$$\vec{c} = \vec{d} + \mathbf{Q}^T \vec{e}, \quad (4)$$

where $\mathbf{Q} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$.

III. Multiple Error Detection

In this section, a technique for designing bit parallel multipliers with multiple error detection is proposed. The basic structure of the multiplier is similar to that of [10]. The classical bit-parallel multiplier is designed by the method described in Section II. The modified single error correcting architecture is shown in Fig. 1. Apart from the functional unit of the multiplier, it consists of a parity prediction unit, output parity generation unit, and the comparison and error detection logic. For Multiple Error Detection (MED), we use parity prediction, which is based on the LDPC codes. The advantage of the LDPC codes is that the complexity of the encoding and decoding logic is simple. We give more emphasis on the error detection, rather than correction, of fault. The reason is that the number of faults detectable $(d-1)$ is larger than the number of correctable faults $(\lfloor \frac{d}{2} \rfloor)$, where d is the minimum distance of the error correction code used. Furthermore, due to the nature of the faults in the cryptographic context and their potential effects on the security of the systems, the detection of the errors is more important than correction. The sizes of the parity prediction circuits depend on the number of input bits. Next, we derive the closed form expressions for the predicted parity bits.

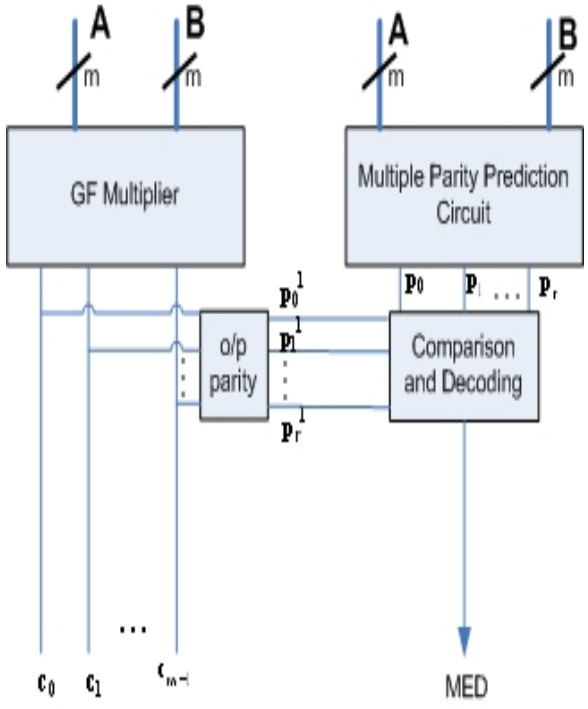

Fig. 1. Proposed Multiple Error Detection Multiplier Over GF(2^m).

For the LDPC codes proposed in the previous chapter, the hamming distance is 3. Now if we add an overall parity the overall hamming distance will be 4. Hence we could detect any three errors. Let $\vec{c} = [c_0, c_1, c_2, \ldots, c_{m-1}]^T$ be the output of the multiplier and the corrected output. Also let r be the number of parity bits, and $\vec{p} = [p_0, p_1, \ldots, p_{r-1}]^T$ and $\vec{p}^1 = [p_0^1, p_1^1, \ldots, p_{r-1}^1]^T$ respectively be the predicted and the parity bits generated from the output bits. Let **H** be the parity check matrix associated with the proposed single error correction scheme.

Design Procedure

- Determine the number of parity bits (r) required to satisfy the required hamming distance and hence the number of detectable errors .
- Construct the **H** matrix based on the approach proposed in the previous chapter with an overall parity check bit added.
- A column vector with a single 1 is assigned to parity P_i.
- The remaining m columns are assigned the output bits c_i, without any constraints.
- Generate predicted parity expressions in terms of a_is and b_is.

Example 2: Consider the multiplier structure over GF(2^6), with primitive polynomial $P(x) = x^6 + x + 1$. Here we have $m = 6$. Therefore, we need 4 parity bits to detect three errors. We have,

$$\mathbf{H} = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & p_0 & p_1 & p_2 & p_3 & p_4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The predicted parity of outputs based on the above H matrix is given by

$$p_0 = c_0 + c_1 + c_2 + c_3 + c_4 + c_5 \quad (5)$$
$$p_1 = c_0 + c_3 + c_5 \quad (6)$$
$$p_2 = c_0 + c_1 + c_4 \quad (7)$$
$$p_3 = c_1 + c_2 + c_3 \quad (8)$$
$$p_4 = c_2 + c_4 + c_5 \quad (9)$$

(Here, the '+' sign represents XOR operation.)

$$p_0 = d_0 + d_1 + d_3 + d_4 + d_5 \quad (10)$$
$$p_1 = d_0 + d_3 + d_5 + e_0 + e_2 + e_3 + e_4 \quad (11)$$
$$p_2 = d_0 + d_1 + d_4 + e_1 + e_3 + e_4 \quad (12)$$
$$p_3 = d_1 + d_2 + d_3 + e_0 + e_3 \quad (13)$$
$$p_4 = d_2 + d_4 + d_5 + e_1 + e_2 + e_3 \quad (14)$$

where, the $\vec{d}$ and $\vec{e}$ outputs of the network are given below.

$d_0 = a_0b_0$; $d_1 = a_1b_0 + a_0b_1$; $d_2 = a_2b_0 + a_1b_1 + a_0b_2$; $d_3 = a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3$; $d_4 = a_4b_0 + a_3b_1 + a_2b_2 + a_1b_3 + a_0b_4$; $d_5 = a_5b_0 + a_4b_1 + a_3b_2 + a_2b_3 + a_1b_4 + a_0b_5$;

$e_0 = a_5b_1 + a_4b_2 + a_3b_3 + a_2b_4 + a_1b_5$; $e_1 = a_5b_2 + a_4b_3 + a_3b_4 + a_2b_5$; $e_2 = a_5b_3 + a_4b_4 + a_3b_5$; $e_3 = a_5b_4 + a_4b_5$; $e_5 = a_5b_5$.

Table III shows the comparison of the area overhead for various multiplier sizes. On average the overhead is slightly over 100%.

TABLE I

HARDWARE OVERHEAD FOR DIFFERENT MULTIPLIERS FOR MULTIPLE ERROR DETECTION

Field Size	Multiplier area in μm^2	PP and other Circuitry area in μm^2	% Overhead
GF(2^{10})	3844.20	4154.448	108.0
GF(2^{11})	4656.90	4941.5	106.1
GF(2^{12})	5547.00	5805.968	104.6
GF(2^{13})	6514.50	6747.852	103.5
GF(2^{14})	7559.40	7767.152	102.7
GF(2^{15})	8681.70	8863.868	102.0

IV. Multiple bit Error Correction Using Reed-Solomon code

A. Reed-Solomon Code

The Reed-Solomon (RS) codes belong to a family of Forward Error Correction (FEC) codes known as the linear block codes. The linear block codes encode a message as a block and the redundant information is unique per block, i.e. the whole message block is passed into the encoder one block at a time and the encoder has no memory of any information from the previous block. This is different from the convolutional codes, which encode continuously. One can think of this style of encoding as a window sliding over the information bits. In the conventional RS encoder, the bits inside the window are encoded, and therefore the encoded bits depend on previous bits, i.e. the encoder has memory. In the proposed technique a different approach is taken, i.e. a parallel encoder which does not have any memory is used.

The RS codes were first proposed by Reed and Solomon in 1960 [14]. They are known to be very efficient algebraic codes, i.e. can correct a large number of errors with a low overhead. By the very nature of their structure, RS codes are well suited to FEC in bursty noise environments [15], [16], [17]. The codes have the power to correct errors that occur in a cluster. Decoding RS codes is a non-trivial task. There are two fundamental types of decoding. The Hard Decision Decoding (HDD) first thresholds the received data, effectively making a hard decision for each bit (0 or 1). Then, using the properties of the code, the decoder detects and corrects the bits that are in error. In contrast, the Soft Decision Decoding (SDD) uses all the information received from the channel, i.e. there is no thresholding of the received data.

The RS codes operate over an extended binary field GF(2^m) and each symbol in the RS code word is an element from the corresponding Galois field. The RS codes have the following parameters: $n = 2^m - 1$ symbols in a code word $k =$ the number of message symbols n - k = 2t redundant symbols. t =the number of errors to be corrected. d_{min} = 2t + 1.

To construct the generator for a Reed Solomon code, we need to construct the appropriate finite field and choose roots. Let the roots be β^i to β^{i+2t-1}, the generator polynomial will be $g(X) = (X+\beta^i)(X+\beta^{i+1})\cdots(X+\beta^{i+2t-2})(X+\beta^{i+2t-1})$ where t is number of symbol errors that can be corrected.

Here, we represent the original output bits and the RS code word using polynomials where the coefficients of the polynomial are output symbols and the redundant symbols and the power of X represents the position of the symbol in the code word. The original output symbols/bits will be denoted c(X) and the code word will be denote o(X). The code word polynomial is related to the output polynomial by the generator polynomial

$$g(X)=\prod_{1}^{2t}(X+\alpha) = g_0 + g_1X + \cdots + g_{2t-1}X^{2t-1} + X^{2t}$$

A Reed-Solomon output word could be generated directly using the generator polynomial o(X)=c(X)g(X)

If m(X) is of degree k-1 and g(X) is of degree 2t, then the resultant code word will be of degree 2t+k-1 = n-1. However, for practical purposes, systematic encoding has the advantage of simplifying the retrieval of the original bits after decoding. Systematic encoding means that the redundant information is appended to the original message.

B. Parallel Encoder

The classical bit-parallel multiplier is designed by the method described in Section II. The modified multiple error correcting architecture for a $GF(2^{15})$ multiplier is shown in Fig. 2. Apart from the functional unit of the multiplier, it consists of a reed Solomon check bits generation, syndrome generator, and correction logic. The sizes of the parity prediction circuits depend on the number of input bits. Next, we derive the closed form expressions for the predicted parity bits.

TABLE II
FIELD ELEMENTS GF(8) WITH $P(x) = x^3 + x + 1$

Element	Representation
0	000
1	001
β	010
β^2	100
$\beta^3 = \beta+1$	011
$\beta^4=\beta^2+\beta$	110
$\beta^5=\beta^2+\beta+1$	111
$\beta^6=\beta^2+0+1$	101
$\beta^7= 1$	001

The basic principle of multiple bit error correction is explained by considering the following motivating example. Here we consider a $GF(2^{15})$ bit multiplier. Let the roots be β and β^2, the generator polynomial will be $g(X) = (X+\beta)(X+\beta^2)$. i.e. $g(X) = X^2+\beta^4X+\beta^3$ The symbols encoded with above g(x) could correct one symbol error.

Let $\vec{c} = [c_0, c_1, c_2, \ldots, c_{14}]^T$ be the output of the multiplier. The 15 bit output is divided into 5 three bit symbols over GF(2^3). The five 3 bit symbols are C_4, C_3,, C_2, C,*and*C_0.

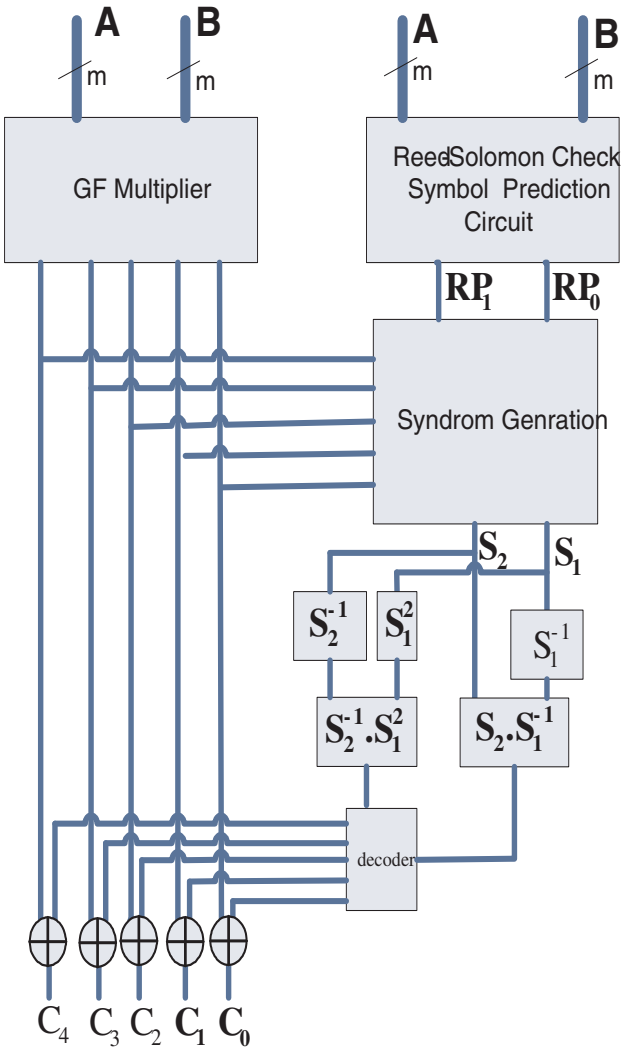

Fig. 2. Proposed Multiple Error Correcting Multiplier Over GF(2^{15}).

Table IV-B shows the field elements of GF(2^3) with $P(x) = x^3 + x + 1$.

Let RP_1 and RP_0 denotes the two Reed Solomon check symbols which is generated from the input operand. Using systematic encoding we can derive a closed close expression for RP_1 and RP_0. That is,

$RP_0 = \beta C_4 + \beta C_3 + \beta^3 C_2 + C_1 + \beta^3 C_0$
$RP_1 = \beta^4 C_4 + \beta^5 C_3 + \beta^5 C_2 + C_1 + \beta^4 C_0$
$C_4 = (c_{14},\ c_{13},\ c_{12})$
$C_3 = (c_{11},\ c_{10},\ c_9)$
$C_2 = (c_8,\ c_7,\ c_6)$
$C_1 = (c_5,\ c_4,\ c_3)$
$C_0 = (c_2,\ c_1,\ c_0)$
$\beta C_4 = (c_{13},\ c_{14} + c_{12},\ c_{14})$
$\beta C_3 = (c_{10},\ c_{11} + c_9,\ c_{11})$
$\beta^3 C_2 = (c_8 + c_7,\ c_8 + c_7 + c_6,\ c_8 + c_6)$
$C_1 = \quad (c_5,\ c_4,\ c_3)$

$\beta^4 C_0 = (c_2 + c_1 + c_0,\ c_1 + c_0,\ c_2 + c_1)$

$\mathrm{RP}_0 = (rp_{02}, rp_{01}, rp_{00})$

$rp_{02} = c_{13} + c_{10} + c_8 + c_7 + c_5 + c_2 + c_1 + c_0$ $rp_{01} = c_{14} + c_{12} + c_{11} + c_9 + c_8 + c_7 + c_6 + c_4 + c_1 + c_0$ $rp_{00} = c_{14} + c_{11} + c_8 + c_6 + c_3 + c_2 + c_1$

$rp_{02} = d_{13} + e_{13} + e_{12} + d_{10} + e_{10} + e_9 + d_8 + e_8 + d_7 + e_6 + d_5 + e_5 + e_4 + d_2 + d_1 + d_0 + e_0$

$rp_{01} = d_{14} + e_{13} + d_{11} + e_{11} + e_{10} + d_9 + e_9 + d_8 + + d_7 + d_6 + e_5 + d_4 + e_4 + e_3 + d_1 + e_1 + d_0$

$rp_{00} = d_{14} + e_{14} + e_{13} + d_{11} + e_{11} + e_{10} + d_8 + e_8 + e_7 + d_6 + e_6 + e_5 + d_3 + e_3 + d_2 + e_1 + d_1 + e_1 + e_0$

Deriving the bit level details of the RP_1, we have $\mathrm{RP}_1 = \beta^4 C_4 + \beta^5 C_3 + \beta^5 C_2 + C_1 + \beta^4 C_0$

$\beta^4 C_4 = (c_{14} + c_{13} + c_{12},\ c_{13} + c_{12},\ c_{14} + c_{13})$

$\beta^5 C_3 = (c_{10} + c_9,\ c_9\ ,\ c_{11} + c_{10} + c_9)$

$\beta^5 C_2 = (c_8 + c_7,\ c_6\ ,\ c_8,\ c_7 + c_6)$

$\mathrm{C}_1 = (c_5,\ c_4,\ c_3)$

$\beta^4 C_0 = (c_2 + c_1 + c_0\ ,\ c_1 + c_0,\ c_2 + c_1)$

$\mathrm{RP}_1 = (rp_{12}, rp_{11}, rp_{10})$

$rp_{12} = c_{14} + c_{13} + c_{12} + c_{10} + c_9 + c_8 + c_7 + c_5 + (c_2 + c_1 + c_0$ $rp_{11} = c_{13} + c_{12} + c_9 + c_6 + c_4 + c_1 + c_0$ $rp_{10} = c_{14} + c_{13} + c_{11} + c_{10} + c_9 + c_8 + c_7 + c_6 + c_3 + c_2 + c_1)$

$rp_{12} = d_{14} + e_{13} + d_{12} + e_{10} + e_9 + d_9 + e_8 + e_7 + d_7 + e_6 + e_7 + d_5 + e_4 + + e_5 d_2 + d_1 + d_0 + e_0$ $rp_{11} = d_{13} + e_{13} + e_{12} + d_{12} + e_{10} + e_9 + d_9 + e_8 + e_7 + d_6 + e_6 + e_5 + d_4 + e_4 + + e_3 d_1 + d_1 + d_0$ $rp_{10} = d_{14} + e_{14} + d_{13} + e_{12} + d_{11} + e_{11} + d_{10} + d_9 + e_8 + d_7 + e_7 + e_6 + d_3 + e_3 + e_2 + d_1 + e_1 + e_0$

The above expressions for RP_1 and RP_0 are used for implementing RS check symbol prediction block in Figure 2

C. Decoding

The decoding algorithm is more complicated and is divided into the following steps:

- Detection of errors .
- Identification of the corrupted symbol
- Error magnitude computation
- Apply correction

Here we consider single symbol errors only(i.e. t= 1) and uses the Peterson-Gorenstein-Zierler (PGZ) algorithm. Considering various RS decoding algorithms, the Peterson-Gorenstein-Zierler in general has the least computational complexity for small t values. According to PGZ algorithm C(x) denote the transmitted code word. Then the received code word, r(x), can be represented as

$$r(x) = C(X) + e(x) \tag{15}$$

where e(x) represents the error pattern. The syndrome values, denoted by S_i,are obtained by evaluating the received polynomial r(x) at β_i That is, equation can be written as

$$S_i = r(\beta^i) = \sum_{j=0}^{n-1} r_j (\beta^i)^j, \qquad 1 \preceq i \leq 2t \tag{16}$$

In our case we denote syndrome as S_1 and S_2, it is computed by

$$S_i = r(\beta^i) = \sum_{j=1}^{2} r_j (\beta^i)^j, 1 \preceq i \leq 2 \tag{17}$$

Let the output of the multiplier be

$C(X) = C_4 X^6 + C_3 X^5 + C_2 X^4 + C_1 X^3 + C_0 X^2 + RP_1 X + RP_0$

Syndrome $S_1 = (s_{12}, s_{11}, s_{10})$ is generated by

$S_1 = C_4 \beta^6 + C_3 \beta^5 + C_2 \beta^4 + C_1 \beta^3 + C_0 \beta^2 + RP_1 \beta + RP_0$

$\beta^6 C_4 = (c_{12}, c_{14}, c_{13} + c_{12})$ $\beta^5 C_3 = (c_{10} + c_9, c_9, c_{11} + c_{10} + c_9)$ $\beta^4 C_2 = (c_8 + c_7 + c_6, c_7 + c_6, c_8 + c_7)$ $\beta^3 C_1 = (c_5 + c_4, c_5 + c_4 + c_3, c_5 + c_3)$ $\beta^2 C_0 = (c_2 + c_0, c_1 + c_0, c_1)$

$\beta^1 RP_1 = (rp_{11}, rp_{12} + rp_{10}, rp_{12})$

$RP_0 = (rp_{02}, rp_{01}, rp_{00})$

$s_{12} = c_{12} + c_{10} + c_8 + c_7 + c_6 + c_5 + c_4 + c_2 + c_0 + rp_{11} + rp_{02})$ $s_{11} = c_{14} + c_9 + c_7 + c_6 + c_5 + c_4 + c_3 + c_1 + c_0 + rp_{12} + rp_{10} + rp_{01})$ $s_{10} = c_{13} + c_{12} + c_{11} + c_{10} + c_9 + c_8 + c_7 + c_5 + c_3 + rp_{10} + rp_{12} + rp_{00})$

Syndrome $S_2 = (s_{22},\ s_{21},\ s_{20})$ is generated by evaluating the output polynomial at β^2

$S_2 = C_4 \beta^1 2 + C_3 \beta^1 0 + C_2 \beta^8 + C_1 \beta^6 + C_0 \beta^4 + RP_1 \beta^2 + RP_0$

$S_2 = C_4 \beta^5 + C_3 \beta^3 + C_2 \beta^1 + C_1 \beta^6 + C_0 \beta^4 + RP_1 \beta^2 + RP_0$

$\beta^5 C_4 = (c_{13} + c_{12}, c_{12}, c_{14} + c_{13} + c_{12})$ $\beta^3 C_3 = (c_{11} + c_{10}, c_{11} + c_{10} + c_9, c_{11} + c_9)$ $\beta^1 C_2 = (c_7, c_8 + c_6, c_8)$ $\beta^6 C_1 = (c_2, c_5, c_4 + c_3)$ $\beta^4 C_0 = (c_2 + c_1 + c_0, c_1 + c_0, c_2 + c_1)$ $\beta^2 RP_1 = (rp_{12} + rp_{10}, rp_{12} + rp_{11}, rp_{11})$ $RP_0 = (rp_{02}, rp_{01}, rp_{00})$

$s_{22} = c_{13} + c_{12} + c_{11} + c_{10} + c_7 + c_2 + c_1 + rp_{12} + rp_{10} + rp_{02})$ $s_{21} = c_{12} + c_{11} + c_{10} + c_9 + c_8 + c_6 + c_5 + c_1 + c_0 + rp_{12} + rp_{11} + rp_{01}$ $s_{20} = c_{14} + c_{13} + c_{12} + c_{11} + c_9 + c_8 + c_4 + c_3 + c_2 + c_1 + rp_{11} + rp_{00}$

The above are used for generating the syndrome.

By PGZ method, the error locator is given by

$$X = \frac{S_2}{S_1} \tag{18}$$

and error magnitude is given by

$$Y = \frac{S_1^2}{S_2} \tag{19}$$

The complete block level implementation is shown in Figure 2.

D. Simulation Results

Both original and error correcting architectures described in the previous section have been coded in VHDL. The results reported here are for 15bit multiplier however the design is generic, it can be extended to any multiplier size . The design was simulated using modelsim^{TM} and was tested for functionality by giving various inputs. The outputs from the VHDL coded architecture are validated against a standard multiplier function. The architectures were synthesized using the Synopsys tools. Synopsys Power compiler^{TM} was used to estimate the power consumption. Figure 3 shows the simulated waveform in a multiple bit error correction case using Reed Solomon code. Here, one of the error injected produces multiple bit errors in symbol C_3 which corresponds to 6^{th}

symbol in the output code. Recall that the 7 symbols are C_4, C_3,C_2, C_1,C_0, RP_1 and RP_0. Further the 6^{th} symbol has changed from 000 to 101. In Figure 3 the error magnitude is 101 and position is 111. The position 111 corresponds to 6^{th} symbol because 111 means β^6 (see Table IV-B).

Table IV-D shows area analysis of the above design example. The overhead is about 185 percentage, however it is much less than a N modular error correction. The above design can correct three errors (one symbol error).

TABLE III
HARDWARE OVERHEAD FOR GF(2^{15}) MULTIPLIER WITH MULTIPLE ERROR CORRECTION

Field size	**Multiplier area in μm^2**	**PP and Other Circuitry area in μm^2**	**% Overhead**
GF(2^{15})	8681.70	15958.868	184.82

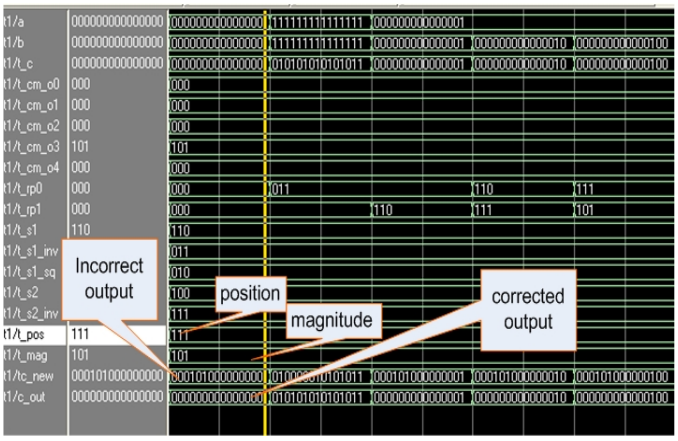

Fig. 3. Simulation Results

V. CONCLUSIONS

This paper proposed multiple bit error detection and correction techniques for PB multipliers over GF(2^m). The LDPC code has been extended for three error detection by adding a overall parity check bit.For multiple error correction, the experimental results suggest that this technique can significantly reduce area, delay, and power compared to N modular redundancy. One of the future works is to make C-testable by applying the techniques described in [6].

ACKNOWLEDGMENTS

The work was partially funded by the Engineering and Physical Science Research Council (EPSRC), United Kingdom, under Grant No: EP/G032904/1 and EP/F030991/1.

REFERENCES

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

[2] 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.

[3] W. Hamming, "Error Detecting and Error Correcting Codes", Bell Systems Tech. Journal, pp. 147-160, 1950.

[4] 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.

[5] A. Reyhani-Masoleh, and M. Anwar Hasan, "Fault Detection Architectures for Field Multiplication Using Polynomial Bases", IEEE 91(11), ",IEEE Trans. Computers, vol. 55, No. 9, Sept. 2006.

[6] H. Rahaman, J. Mathew, D. K. Pradhan and A. M. Jabir, "C-Testable Bit Parallel Multipliers Over GF(2^m)", ACM Transactions on Design Automation of Electronic Systems, Jan, 2008.

[7] J. Mathew, A. Costas, A. M. Jabir, H. Rahaman and D.K. Pradhan, "Single Error Correcting Finite Field Multipliers Over GF(2^m) ", 21^{st} International Conference on VLSI Design, Jan 2008.

[8] M. Nicolaidis and Y. Zorian, "Online Testing for VLSI - A Compendium of Approaches", J. Electronic Testing: Theory and Applications, Vol. 12, Nos. 1/2, pp. 7-20, Feb.-Apr. 1998.

[9] 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.

[10] 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.

[11] R. Gallager, "Low-Density Parity-Check Codes", MIT press, Cambridge,MA, 1963.

[12] E.D. Mastrovito, "VLSI Architectures for Computation in Galois Fields", PhD thesis, Linkoping Univ., Linkoping, Sweden, 1991.

[13] B. Sunar and C.K. Koc, "Mastrovito Multiplier for All Trinomials", IEEE Trans. Computers, vol. 48, no. 5, pp. 522-527, May 1999.

[14] S. Reed and G. Solomon. Polynomial codes over certain finite fields. SIAM Journal of Applied Mathematics, 8, 1960.

[15] S. Lin and D.J Costello. Error Control Coding: Fundamentals and Applications. Prentice-Hall, Englewood Cliffs, N.J, 1983.

[16] Mario Blaum. A course on error correcting codes. IBM Research Division IBM Corp, 1997.

[17] A.Vardy and Y. Beery. Bit level soft-decision decoding for reedsolomon codes. IEEE trans. On Commn, 39:440-444, March 1991

[18] A. Halbutogullari and C.K. Koc, "Mastrovito Multiplier for General Irreducible Polynomials", IEEE Trans. Computers, vol. 49, no. 5, pp. 503-518, May 2000.

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

[20] S. Fenn, M Gossel, M Benaissa, and D. Taylor, "Online Error Detection for Bit-seial Multipliers in GF(2^m)", J. Electronic Testing: Theory and Applications, vol. 13, pp.29-40, 1998.98.

[21] C.Y. Lee, C.W. Chiou and J.M. Lin, "Concurrent error detection in a bit-parallel systolic multiplier for dual basis of GF(2^m)", Journal of Electronic Testing: Theory and Applications, vol.21, pp.539-549,Sep. 2005.

[22] J. Mathew, A.M. Jabir, H. Rahaman and D.K. Pradhan, "Single error correctable bit parallel multipliers over $GF(2^m)$ IET Computers & Digital Techniques,pp. 281-288, May 2009.