Error Detection in Polynomial Basis Multipliers over Binary Extension Fields

Arash Reyhani-Masoleh[1] and M.A. Hasan[2]

[1] Centre for Applied Cryptographic Research,
Department of Combinatorics and Optimization,
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.
areyhani@math.uwaterloo.ca
[2] Department of Electrical and Computer Engineering,
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.
ahasan@ece.uwaterloo.ca

Abstract. In many of cryptographic schemes, the most time consuming basic arithmetic operation is the finite field multiplication and its hardware implementation may require millions of logic gates. It is a complex and costly task to develop such large finite field multipliers which will always yield error free outputs. In this effect, this paper considers fault tolerant multiplication in finite fields. It deals with detection of errors of bit-parallel and bit-serial polynomial basis multipliers over finite fields of characteristic two. Our approach is to partition the multiplier structure into a number of smaller computational units and use the parity prediction technique to detect errors.

Keywords: Finite fields, fault tolerant computing, polynomial basis multiplier, error detection.

1 Introduction

Among the basic arithmetic operations over finite fields $GF(2^m)$, multiplication is the one which has received most attention in the literature [7,4,11,9]. This is mainly because the implementation of a multiplier is much more complex compared to a finite field adder and using multiplication operation repeatedly one can perform other difficult field operations, such as inversion and exponentiation, which are extensively used in cryptographic systems [1,10].

Finite field multiplication is quite different from its counterparts in integer and floating point number systems. For todays cryptographic applications, the field size can be very large and each input of the multiplier can be 160 to 2048 bits long. Such a multiplier may require millions of logic gates and it is a challenging task to implement it free of faults. If one can have a multiplier which is capable of detecting error on-line at the presence of certain faults, cryptographic schemes can be operated more reliably. The importance of eliminating errors in cryptographic computations has been pointed out in some recent articles, for examples [2,5]. The presence of faults in cryptosystems can lead to an active

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 515–528, 2003.

attack and the simplest way to prevent such an attack is to ensure that the computational device verifies the values it computes before sending them out.

In an attempt to detect errors in finite field multipliers, the authors of [3] have considered bit-serial multipliers in $GF(2^m)$ and have presented error detection schemes for four types of multipliers using a parity prediction technique. Their polynomial basis scheme for error detection is applicable to a special class of fields. These fields are defined using irreducible all-one polynomials that are available for certain values of m only. Additionally, when an all-one polynomial is irreducible, the corresponding m is not a prime. This makes many designers to avoid such a value of m and the corresponding irreducible all-one polynomial that define the underlying field for certain cryptosystems, such as those based on elliptic curve cryptography.

In this paper, we consider $GF(2^m)$ multipliers of both bit-parallel and bit-serial types. The polynomial basis is used for representing the field elements. We investigate error detection techniques for such multipliers and develop parity prediction based error detection schemes for both bit-serial and bit-parallel multipliers. The new schemes can be used for any field defining irreducible binary polynomial.

2 Preliminaries

2.1 Multiplication Using Polynomial Basis

Let

$$F(z) = z^m + \sum_{i=0}^{m-1} f_i z^i \tag{1}$$

be a monic irreducible polynomial over $GF(2)$ of degree m, where $f_i \in GF(2)$ for $i = 0, 1, \cdots, m-1$. Let $\alpha \in GF(2^m)$ be a root of $F(z)$, *i.e.*, $F(\alpha) = 0$. Then the set $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$ is known as the polynomial (or standard) basis and each element of $GF(2^m)$ can be written with respect to (w.r.t.) this basis, *i.e.,* if A is an element of $GF(2^m)$, then

$$A = \sum_{i=0}^{m-1} a_i \alpha^i,\ a_i \in \{0,\ 1\}, \tag{2}$$

where a_i's are the coordinates of A w.r.t. polynomial basis (PB). For convenience, these coordinates will be denoted in vector notation as

$$\mathbf{a} = [a_0,\ a_1,\ a_2,\ \cdots\ a_{m-1}]^T, \tag{3}$$

where T denotes the transposition of a vector.

Let C be the product of any two elements A and B of $GF(2^m)$. Then, C can be represented w.r.t. PB as follows:

$$A \cdot B = A \cdot \sum_{i=0}^{m-1} b_i \alpha^i = \sum_{i=0}^{m-1} b_i \cdot (A\alpha^i),$$

$$C = A \cdot B \mod F(\alpha) = \sum_{i=0}^{m-1} b_i \cdot ((A\alpha^i) \mod F(\alpha)) \tag{4}$$

$$= \sum_{i=0}^{m-1} b_i \cdot X^{(i)}, \tag{5}$$

where

$$X^{(i)} = \alpha \cdot X^{(i-1)} \mod F(\alpha), \quad 1 \le i \le m-1 \tag{6}$$

and

$$X^{(0)} = A.$$

A bit-parallel architecture for $GF(2^m)$ multiplication using (5) is shown in Figure 1. It mainly consists of three types of modules, namely, sum, pass-thru and α modules. The sum module (denoted as a double circle with a plus inside) is to simply add two $GF(2^m)$ elements and it can be realized in hardware using m two-input XOR gates. The pass-thru module (denoted as a double circle with a dot inside) is to multiply a $GF(2^m)$ element by a $GF(2)$ element, *i.e.*, if $X^{(i)} \in GF(2^m)$ and $b_i \in GF(2)$ are two inputs to a pass-thru module, then its output is

$$b_i X^{(i)} = \begin{cases} X^{(i)} & \text{if } b_i = 1, \\ 0 & \text{if } b_i = 0. \end{cases}$$

In hardware, each pass-thru module consists of m two-input AND gates.

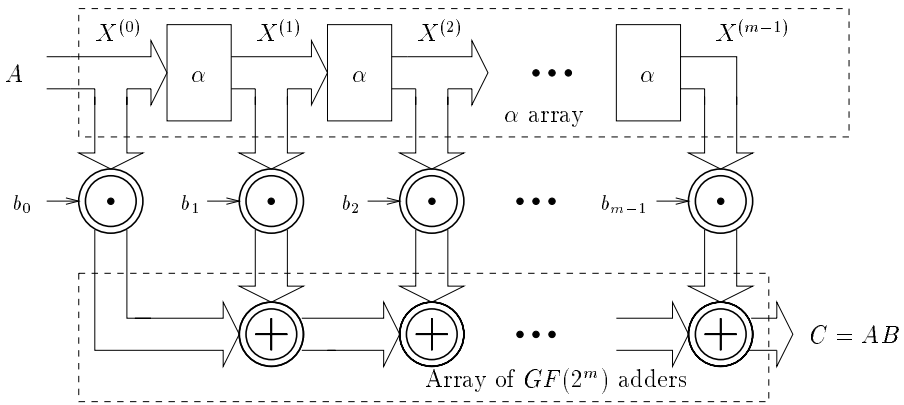

Fig. 1. Multiplication of two elements in $GF(2^m)$

In Figure 1, the third module (*i.e.*, the rectangular shape α module) multiplies its input, which is an element of $GF(2^m)$, by α and reduces the result modulo $F(\alpha)$. Thus, this module is to essentially realize equation (6) in hardware.

Since α is a root of $F(z)$,

$$F(\alpha) = \alpha^m + \sum_{i=0}^{m-1} f_i \alpha^i = 0. \tag{7}$$

Then multiplication of an arbitrary element $A \in GF(2^m)$ by α gives

$$A \cdot \alpha = (\sum_{i=0}^{m-1} a_i \alpha^i) \cdot \alpha = \sum_{i=0}^{m-1} a_i \alpha^{i+1} = \sum_{i=1}^{m-1} a_{i-1} \alpha^i + a_{m-1} \alpha^m. \tag{8}$$

Using (7) and (8), one can write

$$\begin{aligned} X &\triangleq A \cdot \alpha \bmod F(\alpha) \\ &= a_{m-1} \cdot f_0 + \textstyle\sum_{i=1}^{m-1} (a_{i-1} + a_{m-1} \cdot f_i) \alpha^i, \end{aligned} \tag{9}$$

where x_i's are in $GF(2)$ and are the coordinates of X w.r.t. the PB. For any irreducible polynomial over $GF(2)$, $f_0 = 1$. Thus from (9), we write the coordinates of X as

$$x_i = \begin{cases} a_{i-1} + a_{m-1} \cdot f_i & 1 \le i \le m-1, \\ a_{m-1} & i = 0. \end{cases} \tag{10}$$

If ω is the Hamming weight of the irreducible polynomial $F(z)$, then the realization of (10) requires $\omega - 2$ XOR gates, and so does an α module. Thus, unlike the sum and pass-thru modules, the α module has a space (or circuit) complexity which depends on $F(z)$. The space complexity is minimum when $F(z)$ is a trinomial and maximum when $F(z)$ is an all-one-polynomial (AOP).

In vector notations, the coordinates of the $GF(2^m)$ multiplication can be calculated by the well-known formulation [7] as

$$\mathbf{c} = [c_0, c_1, \cdots c_{m-1}]^T = \mathbf{M} \cdot \mathbf{b}, \tag{11}$$

where $\mathbf{M} = [m_{i,j}]$, $m_{i,j} \in GF(2)$, is the $m \times m$ product matrix and $\mathbf{b} = [b_0, b_1, \cdots b_{m-1}]^T$. Note that in Figure 1, the α array generates $m_{i,j}$'s and another part of the multiplier which consists of all pass-thru and sum modules realizes matrix-vector multiplication of (11).

2.2 Error Detection Strategy

In the following sections, we investigate error detection schemes for $GF(2^m)$ multiplication operation that relies on the architecture shown in Figure 1. Towards this effort, the parity prediction method is used. This method is shown in Figure 2 where the CUT (circuit under test) block can be either a complete finite field multiplier or a part of it with A and B as inputs and Y as output, where $A, B, Y \in GF(2^m)$. In this figure, the parity generation (PG) block produces the actual parity of Y, i.e., $p_Y = \sum_{i=0}^{m-1} y_i$, where y_i's are the coordinates of Y w.r.t. the PB. The actual parity p_Y is then compared with the predicted parity $\hat{p}_Y$

using a single XOR gate as shown in the figure. This comparison is monitored by an error indicator flag $\hat{e}_{\mathrm{CUT}}$ where $\hat{e}_{\mathrm{CUT}} = 0$ indicates that no error has been detected and $\hat{e}_{\mathrm{CUT}} = 1$ flags the detection of errors. The parity prediction (PP) block predicts the parity of the output Y using a PP function which depends only on the inputs of the CUT as

$$\hat{p}_Y = \mathit{\Gamma}_{\mathrm{CUT}}(A, B).$$

We assume that the parity of A and B (i.e., p_A and p_B, respectively) are available or they can be reliably pre-computed while loading the coordinates of A and B into the multiplier. We also assume that the PP and PG blocks can be made fault free or any fault in them can be detected using a suitable mechanism since these blocks are simple and/or regular (for example PP can be as simple as an XOR gate and PG is a modulo 2 adder). In the following sections, we derive the function $\mathit{\Gamma}_{\mathrm{CUT}}$ for each of the modules of the multiplier of Figure 1.

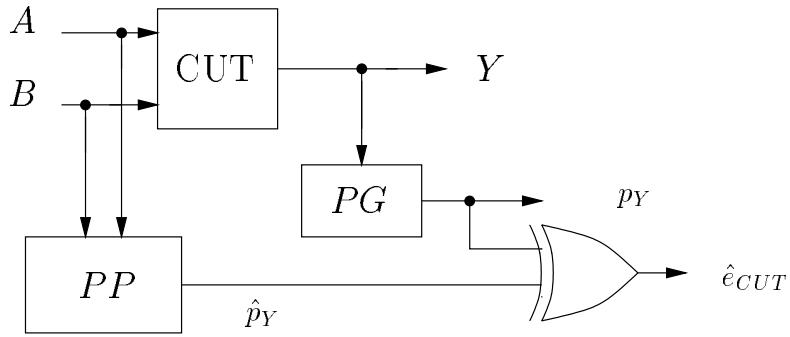

Fig. 2. Error indication of the circuit under test (CUT) using parity prediction method.

For the purpose of this investigation, we consider $GF(2^m)$ multiplier circuit with a single fault. The single fault case provides simplicity in our analysis. Although various types of multiple faults in the multiplier can be detected, we first consider the single fault case and then we show how multiple faults can be detected. This fault is modeled as a stuck-at fault, which appears to be the most common model used for logical faults. For this model, a fault in a logical gate (i.e., XOR, AND, OR, etc.) results in one of its inputs or the output being fixed to either a logic 0 (stuck-at-0, or s-a-0 in short) or a logic 1 (stuck-at-1, or s-a-1), respectively [6].

3 Parity Predictions of Individual Module

In the following, we obtain the parity prediction functions of the modules of the bit-parallel multiplier of Figure 1. PB multipliers (both bit-parallel and bit-serial) that are capable of detecting errors are considered in the next section.

3.1 Parity Prediction in α Module

Let ω be the Hamming weight of the irreducible polynomial $F(z)$. Then, (1) can be written as

$$F(z) = 1 + \sum_{j=1}^{\omega-2} z^{\rho_j} + z^m, \tag{12}$$

where ρ_j's are powers of z in (1) with $f_{\rho_j} = 1$, $1 \le j \le \omega - 2$. Then we have

$$1 \le \rho_1 < \rho_2 < \rho_3 \cdots < \rho_{\omega-2} \le m - 1$$

and (10) can be written as

$$x_i = \begin{cases} a_{\rho_j - 1} + a_{m-1} & i = \rho_j,\ 1 \le j \le \omega - 2, \\ a_{i-1 \bmod m} & \text{otherwise.} \end{cases} \tag{13}$$

Using (13), a circuit diagram for the α module is shown in Figure 3. Note that a stuck-at fault in one of the $(\omega - 2)$ XOR gates of this module causes at most one error at the output.

For $j \in \{\rho_1, \rho_2, \cdots, \rho_{\omega-2}\}$, assume that the j-th gate in Figure 3 is faulty. Then all the output coordinates, except x_j, are error free. If the upper input of the j-th gate is stuck, then the erroneous j-th coordinate is

$$\dot{x}_j = \begin{cases} a_{m-1} \text{ for s-a-0}, \\ \overline{a}_{m-1} \text{ for s-a-1}, \end{cases} \tag{14}$$

where $\overline{x}$ indicates complement of x. On the other hand, if the lower input is stuck, then

$$\dot{x}_j = \begin{cases} a_{j-1} \text{ for s-a-0}, \\ \overline{a}_{j-1} \text{ for s-a-1}. \end{cases} \tag{15}$$

Detection of such errors are discussed below.

Assume that A and X are the input and output of the α module, respectively. Then we have the following lemma.

Lemma 1. *Let $p_A = \sum_{i=0}^{m-1} a_i$ and $p_X = \sum_{i=0}^{m-1} x_i$ be the parity bits of A and X, respectively. Then, the predicted parity of X is*

$$\hat{p}_X = \Gamma_\alpha = p_A + a_{m-1}, \tag{16}$$

where a_{m-1} is the $(m-1)$-th coordinate of input A.

Proof. Using (10), $\hat{p}_X$ can be written as

$$\hat{p}_X = a_{m-1} + \sum_{i=1}^{m-1} (a_{i-1} + a_{m-1} f_i) = a_{m-1} \sum_{i=1}^{m-1} f_i + \sum_{i=0}^{m-1} a_i \,.$$

Since $F(z)$ is irreducible over $GF(2)$, $F(z)$ is not divisible by $z + 1$, and $F(1) = 1$. Then from (1), one obtains $\sum_{i=1}^{m-1} f_i = f_0 = 1$ and the proof is complete.

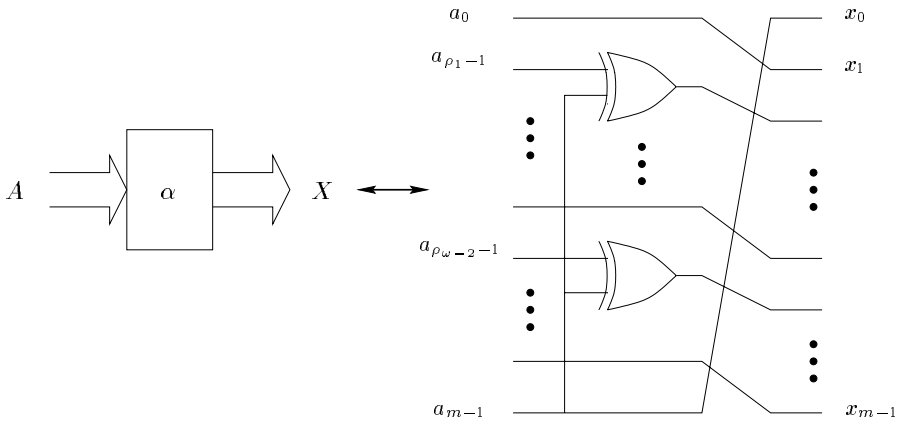

Fig. 3. The original circuit of α module.

Using (16), we can obtain the relation between X and A in the fault free α module as

$$p_X + p_A = a_{m-1}. \tag{17}$$

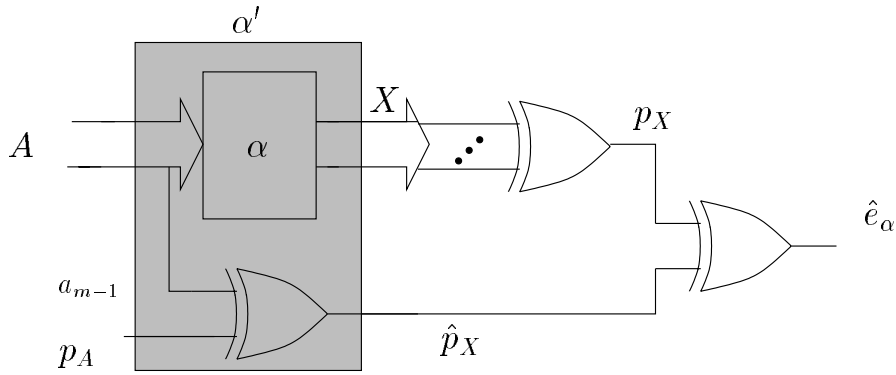

Fig. 4. The circuit for detecting a single fault.

In the α module, a stuck-at fault in one of its gates will result in an output which is different from X and (16) will not hold. Thus, equation (16) can be used for detecting an error in the output of the α module. Circuit for detecting such errors is shown in Figure 4, where $\hat{e}_\alpha = 0$ indicates that no error has been detected and $\hat{e}_\alpha = 1$ flags the detection of an error. Since (16) is over $GF(2)$, the values of $\hat{e}_\alpha$ would detect not only a single error, but also any odd number of errors. With a similar argument, it is clear that even number of errors are not detected by $\hat{e}_\alpha$.

3.2 Parity Predictions of Sum and Pass-Thru Modules

The sum module of Figure 1 is a finite field adder which produces sum of the two elements of $GF(2^m)$ at its output. Let $A = (a_0, a_1, \cdots, a_{m-1})$ and $B = (b_0, b_1, \cdots, b_{m-1})$ be two inputs to this module. Then the output is $D = A+B = (d_0, d_1, \cdots, d_{m-1})$ where $d_i = a_i + b_i$ for $0 \leq i \leq m-1$. The architecture of this module uses m two-input XOR gates. Let $p_A = \sum_{i=0}^{m-1} a_i$ and $p_B = \sum_{i=0}^{m-1} b_i$ be the parity bits of A and B, respectively. Then the parity bit of the output, $p_D = \sum_{i=0}^{m-1} d_i$, is predicted by

$$\hat{p}_D = \Gamma_{sum} = p_A + p_B \tag{18}$$

using one extra XOR gate. Let us denote this $m+1$ XOR gates as the new sum module.

The pass-thru module of Figure 1 multiplies an element $A \in GF(2^m)$ by a single bit $b \in GF(2)$ which can be implemented using m two-input AND gates. Let $G \in GF(2^m)$ be the output of such a module with inputs of A and b. Thus, the output of this module G is zero when $b = 0$ and A when $b = 1$.

Detection of an odd number of errors is accomplished by using a single parity bit similar to the α and sum modules. Let A and $p_A = \sum_{i=0}^{m-1} a_i$ be the input of the pass-thru module and its parity bit respectively. Then, the parity bit of the output $G = bA$ is found as $p_G = \sum_{i=0}^{m-1} g_i$, where $g_i = b \cdot a_i$, $0 \leq i \leq m-1$, are the coordinates of G. Thus, the predicted parity bit of the output can be expressed as

$$\hat{p}_G = \Gamma_{pass} = b \cdot p_A \tag{19}$$

which requires only one AND gate for its implementation. Let us denote the original pass-thru module together with this AND gate as the new pass-thru module similar to the new sum module. These new modules are used in the next section.

4 Error Detections in Polynomial Basis Multipliers

The discussions of the previous section deals with the parity prediction functions of individual modules of the multiplier of Figure 1. Using these parity functions, below we attempt to detect errors in the entire multiplier.

4.1 Bit-Parallel PB Multiplier

Let us generalize (1) for the cascading of j, $1 \leq j \leq m-1$, α modules as follows:

Lemma 2. *As defined earlier $x_{m-1}^{(k)}$ and b_j are the $(m-1)$-th and j-th coordinates of $X^{(j)} = A\alpha^j \mod F(\alpha)$ and B w.r.t. the polynomial basis, respectively. Then*

$$\hat{p}_{X^{(j)}} = p_A + \sum_{k=0}^{j-1} x_{m-1}^{(k)}, \qquad j = 1, 2, \cdots, m-1, \tag{20}$$

Thus, the parity bit of the output of the polynomial basis multiplier can be predicted using the following theorem.

Theorem 1. *Let C be the product of two arbitrary elements A and B of $GF(2^m)$. Let p_A, p_B and p_C be the parity bits of A, B and C respectively. Then,*

$$\hat{p}_C = \sum_{j=0}^{m-1} b_j \hat{p}_{X^{(j)}}, \tag{21}$$

A proof of the above theorem is not included here for lack of space. Note that the theorem is not restricted to any particular irreducible polynomials. When $F(z)$ is an all-one polynomial, the expression for $\hat{p}_C$, which can be obtained from Theorem 1, matches the corresponding result reported in [3].

To detect only one error (in general any odd number of errors) at the output of the multiplier, equation (21) can be realized easily by replacing all the α, pass-thru and sum modules in Figure 1 with the α' module and the new pass-thru and sum modules as shown in Figure 5 (these three new modules are shaded in this figure to distinguish them from the old ones). The bus width of this multiplier is $m+1$. Since the output of any gate of the shaded pass-thru and sum modules in Figure 5 is connected to only one gate, the single stuck fault at any gate of these modules changes only one coordinate of the output of this multiplier. Therefore, a circuit that compares the actual parity p_C with the predicted $\hat{p}_C$, which is shown at the end of the figure, is capable of detecting any single fault in the shaded sum and pass-thru modules of Figure 5. Also, it is clear that any single fault in any XOR gate in the parity generation circuit p_C and the very last XOR gate can be detected by $\hat{e}$. This circuit, however cannot detect a single stuck-at fault in any of the α' modules with the exception of the rightmost α' module, because such a fault is most likely to change more than one bit of the multiplier output. Then, these errors cannot be detected if an even number of output bits are changed due to a single fault in the α' array. To overcome this problem, the following method is proposed.

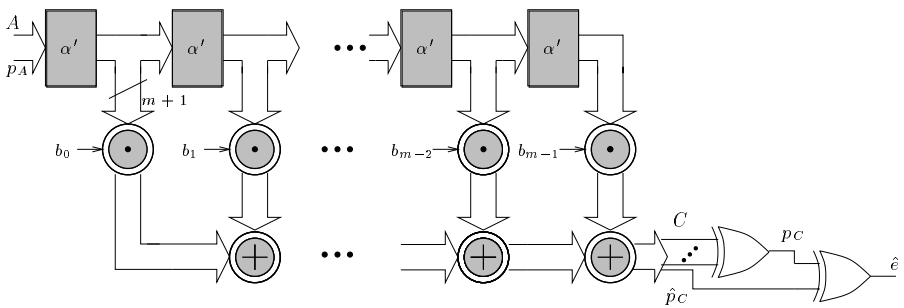

Fig. 5. Multiplication of two elements in $GF(2^m)$ with error detection capability.

For detecting a single fault in the entire multiplier one can change the α array (all α' modules excluding the XOR gates for parity prediction of $\hat{p}_{X^{(j)}}$'s) in such a way so that all $X^{(i)}$'s, $0 \leq i \leq m-1$, are obtained directly from A (instead of $X^{(i-1)}$), i.e., $X^{(i)} = \alpha^i A$ and this is shown in Figure 6. This makes the output of any gate inside the new α array connected to only one gate. In Figure 6, the output of the α^i modules, $X^{(i)}$, $1 \leq i \leq m-1$, are found directly from A. Also, it is noted that the coordinates of $X^{(i)}$'s are obtained using the following matrix equation

$$\mathbf{x}^{(i)} = \mathbf{G}^i \cdot \mathbf{a}, \ \ 1 \leq i \leq m-1, \tag{22}$$

where $\mathbf{x}^{(i)}$ is a vector whose entries are coordinates of $X^{(i)}$ defined by (6) and

$$\mathbf{G} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & f_1 \\ 0 & 1 & \cdots & 0 & f_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & f_{m-1} \end{bmatrix}$$

is the α-multiplication matrix. Using (22), the α^i module in Figure 6 is realized with XOR gates according to the $\mathbf{G}^i$ matrix. As a result, a single stuck-at fault at any logic gate in the multiplier, except in the XOR gates for parity prediction of $\hat{p}_{X^{(j)}}$'s, can affect at most one bit of the output so that it is detected using the parity prediction of (21).

Note that $x_{m-1}^{(k)}$ used in (20) is a function of A and can be calculated and then should be realized separately by using Proposition 4.1 of [7] as follows

$$x_{m-1}^{(k)} = a_{m-1-k} + \sum_{t=0}^{k-1} q_{k-1-t,m-1}\, a_{m-1-t}\ , \quad k = 1, \cdots, m-1, \tag{23}$$

where $q_{i,m-1} \in \{0,1\}$, for $i = 0, 1, \cdots, m-2$, is the i-th entry of the last column of the $m-1 \times m$ binary reduction matrix $\mathbf{Q}$ associated with $F(z)$ as follows:

$$\begin{bmatrix} \alpha^m \\ \alpha^{m+1} \\ \vdots \\ \alpha^{2m-2} \end{bmatrix} = \mathbf{Q} \begin{bmatrix} 1 \\ \alpha \\ \vdots \\ \alpha^{m-1} \end{bmatrix} \mod\ F(\alpha).$$

By substituting (23) into (21), one can realize $\hat{p}_{X^{(j)}}$ as a function of the coordinates of A using XOR gates. Thus any single stuck-at fault in the entire new multiplier results in at most one error and can be detected.

4.2 Bit-Serial PB Multiplier

The PB multiplier of Figure 1 can be realized in a bit-serial fashion as shown in Figure 7. In this figure, both X and Y are m bit registers. Let $X(n)$ and $Y(n)$ be

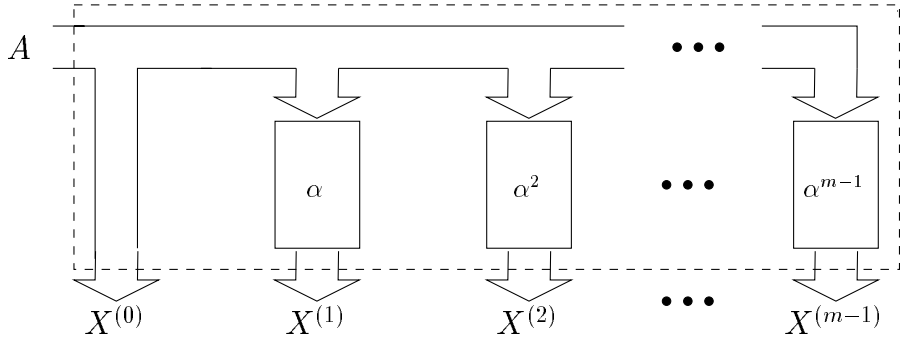

Fig. 6. The architecture of new α array to have a detection capability at the output.

the contents of X and Y registers, respectively, at nth, $1 \leq n \leq m$, clock cycle. Suppose the X register is initialized by A, i.e., $X(0) = A$, then the content of this register at the nth clock cycle is $X(n) = X^{(n)}$, where $X^{(n)} \in GF(2^m)$ is defined in (6). Also, suppose that the register Y is cleared at the initial step, i.e., $Y(0) = 0$. Then one can obtain the content of Y at the first clock cycle as $Y(1) = b_0 A$ and in general at the nth clock cycle as $Y(n) = b_0 A + \sum_{i=1}^{n-1} b_i X(i)$, $1 < n \leq m$. It is easy to verify that after m clock cycle Y contains $C = AB \in GF(2^m)$, i.e., $Y(m) = C$.

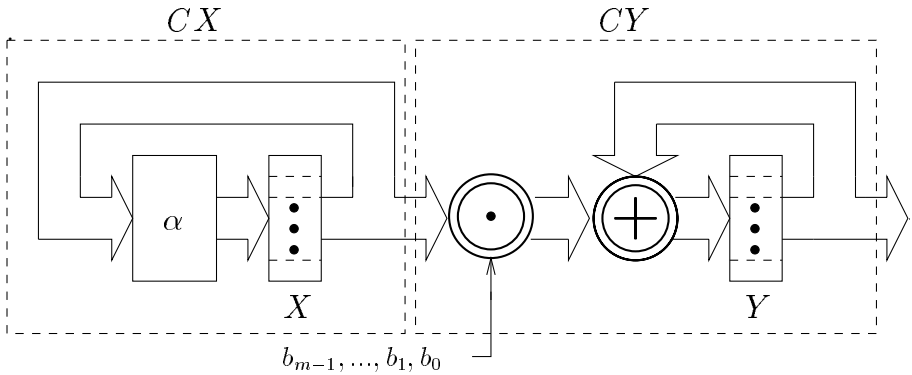

Fig. 7. Bit-serial PB multiplier.

In order to detect errors in the bit-serial multiplier of Figure 7, we check the contents of two registers in every clock cycle. Consider Figure 7 before the triggering of the nth clock cycle when the input and output of the X register are $X(n)$ and $X(n-1)$, respectively and using Lemma 1, we have

$$\hat{p}_{X(n)} = p_{X(n-1)} + x_{m-1}(n-1) = \sum_{i=0}^{m-2} x_i(n-1),$$

where $x_i(n-1) \in GF(2)$ is the ith coordinate of $X(n-1)$. In order to compare $\hat{p}_{X(n)}$ with the actual value of $p_{X(n)}$, we store $\hat{p}_{X(n)}$ into a 1 bit register D_X as shown in Figure 8. Then, after the nth clock cycle, $X(n)$ appears at the output of the X register and the actual value of $p_{X(n)}$ is evaluated and compared with the value of D_X, i.e., $\hat{p}_{X(n)}$ using the last XOR gate of Figure 8. Similar expression can be obtained for the Y register. Since $Y(n) = Y(n-1) + b_{n-1}X(n-1)$, then

$$\hat{p}_{Y(n)} = p_{Y(n-1)} + b_{n-1}p_{X(n-1)},$$

and can be implemented and compared with the actual value of $p_{Y(n)}$ as shown in Figure 8. As a result, after the first clock cycle, both $\hat{e}_{CX}$ and $\hat{e}_{CY}$ should be 0 during the next m clock cycles if there are no single errors.

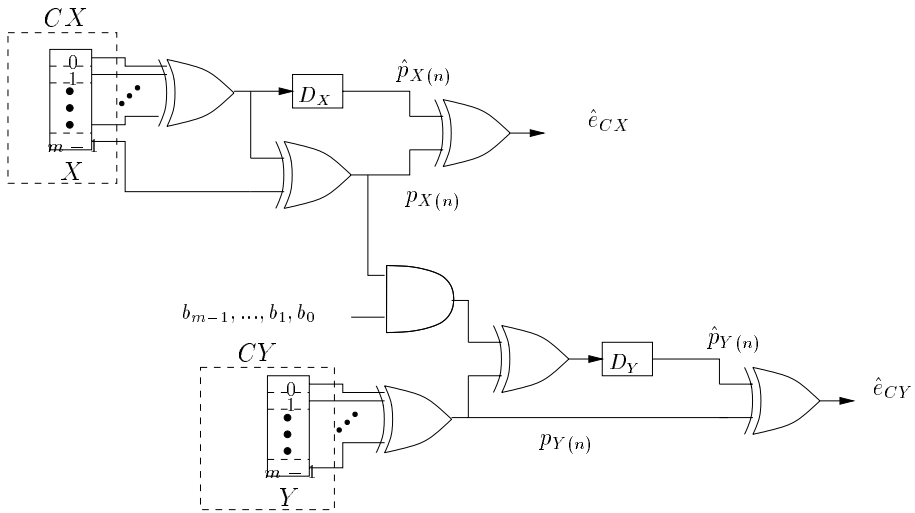

Fig. 8. Detection of errors in the bit-serial PB multiplier.

5 Conclusions and Future Work

In this article, we have considered detection of errors in polynomial basis multipliers. We have used a multiplier structure where a single stuck-at fault causes only odd number of errors at the output. Towards the detection of this type of errors, necessary theoretical results have been presented. Compared to the previously published results [3], the work presented here is quite generic in the sense that it can be applied to any irreducible polynomial defining the field. The

parity prediction method of [3] is only for bit-serial multipliers and based on the prediction of the output parity after the final clock cycle and then comparing it with the actual parity. Although, it reduces the cost of overhead, but its probability of error detection is only about 50% or less. This is because a single fault in their bit-serial multiplier produces multiple errors after m clock cycles and the number of effective errors resulting from the single fault is either odd or even and only the odd number of errors can be detected. The proposed circuit in Figure 8 overcomes this problem. It compares the predicted parity of the storage registers with the actual ones at every clock cycle. Although it costs extra hardware, the probability of error detection of our bit-serial multiplier is about 100%. This result has been verified using a simulation program for a prototype multiplier with $F(z) = z^4 + z + 1$. Using VHDL, we have injected single faults at different nodes of the bit-serial multiplier for all elements of A and B. The probability reaches unity as m increases.

The proposed error detection schemes are not limited to the multiplier architectures discussed in this article. They can be easily extended and applied to other $GF(2^m)$ multipliers. For example, we have considered the bit-serial multiplier introduced by Peterson [7] and have made it capable of detecting single faults. Furthermore, although our discussions have centered around bit-parallel and bit-serial multipliers over $GF(2^m)$, by combining the error detection schemes for serial and parallel multipliers, one can develop an error detection scheme for hybrid multipliers over composite fields [8].

More research is needed to reduce the overhead cost of the proposed multiplier. For example, hardware implementation of the architecture shown in Figure 6 appears to be expensive. Currently we are trying to develop an architecture that can alleviate this problem.

References

1. G. B. Agnew, T. Beth, R. C. Mullin, and S. A. Vanstone. "Arithmetic Operations in $GF(2^m)$". *Journal of Cryptology*, 6:3–13, 1993.
2. D. Boneh, R. A. DeMillo, and R. J. Lipton. "On the Importance of Eliminating Errors in Cryptographic Computations". *Journal of Cryptology*, 14:101–119, 2001.
3. S. Fenn, M. Gossel, M. Benaissa, and D. Taylor. "On-Line Error Detection for Bit-Serial Multipliers in $GF(2^m)$". *Journal of Electronic Testing: Theory and Applications*, 13:29–40, 1998.
4. A. Halbutogullari and C. K. Koc. "Mastrovito Multiplier for General Irreducible Polynomials". *IEEE Transactions on Computers*, 49(5):503–518, May 2000.
5. M. Joye, A. K. Lenstra, and J. J. Quisquater. "Chinese Remaindering Based Cryptosystems in the Presence of Faults". *Journal of Cryptology*, 12:241–245, 1999.
6. P. K. Lala. *Fault Tolerant and Fault Testable Hardware Design*. Prentice Hall, 1985.
7. E. D. Mastrovito. *VLSI Architectures for Computation in Galois Fields*. PhD thesis, Linkoping Univ., Linkoping Sweden, 1991.
8. C. Paar, P. Fleishmann, and P. Soria-Rodriguez. "Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents". *IEEE Transactions on Computers*, 48(10):1025–1034, Oct. 1999.

9. A. Reyhani-Masoleh and M. A. Hasan. "A New Efficient Architecture of Mastrovito Multiplier over $GF(2^m)$". In 20^{th} *Biennial Symposium on Communications*, pages 59–63, Kingston, Ontario, Canada, May 2000.
10. H. Wu and M. A. Hasan. "Efficient Exponentiation of a Primitive Root in $GF(2^m)$". *IEEE Transactions on Computers*, 46(2):162–172, Feb. 1997.
11. T. Zhang and K. K. Parhi. "Systematic Design of Original and Modified Mastrovito Multipliers for General Irreducible Polynomials". *IEEE Transactions on Computers*, 50(7):734–748, July 2001.