On Concurrent Detection of Errors in Polynomial Basis Multiplication

Siavash Bayat-Sarmadi and M. Anwar Hasan

Abstract

The detection of errors in arithmetic operations is an important issue. This paper discusses the detection of multiple-bit errors due to faults in bit-serial and bit-parallel polynomial basis (PB) multipliers over binary extension fields. Our approach is based on multiple parity bits. Experimental results presented here show that due to an increase in the number of parity bits, the area overhead tends to increase linearly, but the probability of error detection approaches unity fairly quickly, e.g., for 8 parity bits. In bit-serial implementation of a $GF(2^{163})$ PB multiplier using 8 parity bits, the area overhead and the probability of error detection are 10.29% and 0.996, respectively. This is achieved without any increase in the computation time of the $GF(2^{163})$ PB multiplier.

Index Terms

Polynomial basis multiplication, concurrent error detection, finite fields.

1. Introduction

Digital systems that require large number of circuits for their implementation can be more prone to produce erroneous results simply because of the increase in the probability that one of the circuits may become faulty while in use. As a result, for sensitive or critical applications large digital systems are generally designed with some kind of mechanism to provide correct functionality or to detect errors.

In some hardware based cryptosystems or their arithmetic accelerators, a finite field multiplier can be the most silicon area occupying component [1], [2] and hence can be subject to hardware

Siavash Bayat-Sarmadi and M. Anwar Hasan are with the Department of Electrical and Computer Engineering, and with the Center for Applied Cryptographic Research at University of Waterloo, Ontario, Canada

faults. Depending on the cryptosystems, errors due to such faults in the multiplier can be detected at an upper level operation, e.g., in elliptic curve cryptography (ECC), if a point leaves the curve it can be easily detected by point verification [3], [4]. This is, however, not always possible. In the case of ECC, a fault may move a point to another point without leaving the curve and this has been exploited in the so-called sign change fault attack [5]. As a result, some kind of mechanisms for error detection in the finite field multiplier can be quite important in cryptography as well as other critical applications where finite field multipliers of various sizes are used, for example, in deep space channel coding [6] and VLSI testing [7].

One technique to detect errors in hardware implementation is on-line testing or concurrent error detection (CED). CED is used to concurrently test a system while the system is operating normally [8]. CED can test the circuit at full operating speed without stopping the system or switching it to test mode. Accordingly, CED can detect transient faults, which may not be detected in off-line testing, since they may not occur in test mode (see [9], [10], [11], [12], [13], [14], [15], [16], [17] as CED examples). This paper focuses on the detection of errors in extension field multipliers. The complexity of multiplication is much higher than the field's two basic operations namely addition and subtraction. Other complex finite field arithmetic operations such as inversion and exponentiation over binary extension fields can be preformed by repeated multiplications [18], [19].

In [11], Fenn et al. presented a concurrent error detection scheme for finite field multipliers over binary extension fields. They used a parity bit for detecting errors in bit-serial multipliers, using a number of bases for representation of fields, defined by an irreducible all-one polynomial. Thus, the scheme is not generic in the sense that it cannot be used for other field defining polynomials. In [10], Chiou presented a concurrent error detection for two bit-parallel systolic multipliers for extension fields which the field defining polynomials are irreducible all-one polynomials or irreducible equally spaced ones. In [14], [16], Reyhani-Masoleh and Hasan developed a generic parity based error detection scheme for both bit-serial and bit-parallel polynomial basis multipliers. The scheme can detect any odd number of erroneous bits. In this scheme, input parity is developed through the multiplier, and predicted output parity is compared to actual output parity. In case of inequality of the parities, an error signal is given.

This paper extends the work of [14], [16] by applying multiple parity bits. The concept of multiple parity is already known and used in some other applications [20], [21], but this is the

first time it is being used for the finite field multiplication. Like [16], our work can be applied to any finite field GF(2^m). However, unlike [16], our work can detect all odd parity errors as well as most of the even parity errors. Additionally, our work can detect at least m multiple-bit errors in the multiplier.

The main contributions of this paper are summarized as follows:

- A multiple parity scheme that can detect multiple-bit errors in both bit-serial and bit-parallel polynomial basis multipliers over binary extension fields are presented. The error detection capability of the scheme in the presence of multiple-bit random errors is also investigated. With our proposed frequency of check points, a maximum of one multiple-bit error in each round of the bit-serial operation (or each row of the bit-parallel operation) can be detected. This implies that in a $GF(2^m)$ polynomial basis multiplier, at least m multiple-bit errors can be detected.
- A number of experimental analyses are presented, including the simulation-based fault-injection evaluation of the scheme and the analyses of the area and time overheads. Our experimental results show that the area overhead tends to increase linearly as the number of parity bits increases but the probability of undetected errors decreases quite quickly. Furthermore, the area overhead for the bit-serial implementation is quite low, e.g., for 8 parity bits the area overhead is 10.29% and the error detection probability is 0.996. The area overhead for a bit-parallel implementation of the multiplier is greater than the corresponding bit-serial one, but it is still lower than the conventional *dual modular redundant* systems. The average time overhead due to the use of the scheme in bit-parallel implementations is 25%. For bit-serial implementations, time overheads have been observed to be small to negligible.

The organization of the remainder of this paper is as follows. In Section 2, some preliminaries about polynomial basis multiplication are discussed. A concurrent error detection strategy is presented in Section 3. In Section 4, the error detection capability of the scheme is investigated. Our experimental results for this scheme are reported in Section 5. Finally, Section 6 gives a few concluding remarks.

2. PRELIMINARIES

In this section, first polynomial basis multiplication is briefly explained. Then three main components for the construction of bit-serial and bit-parallel multipliers are introduced.

Let $F(x) = \sum_{i=0}^{m} f_i x^i$ be an irreducible polynomial over $GF(2)$ of degree m. Let $\alpha \in GF(2^m)$ be a root of $F(x)$, i.e., $F(\alpha) = 0$. Polynomial (or canonical) basis is defined as the following set:

$$\left\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\right\}$$

Each element A of $GF(2^m)$ can be represented using the polynomial basis (PB) as $A = \sum_{i=0}^{m-1} a_i \alpha^i = (a_0 a_1 \cdots a_{m-1})$ where $a_i \in GF(2)$.

The multiplication of α and an arbitrary element A of $GF(2^m)$ can be represented with respect to PB as:

$$\begin{aligned} \alpha A &= \alpha \sum_{i=0}^{m-1} a_i \alpha^i \bmod F(\alpha) \\ &= a_{m-1} f_0 + \sum_{i=1}^{m-1} \left(a_{m-1} f_i + a_{i-1}\right) \alpha^i. \end{aligned}$$

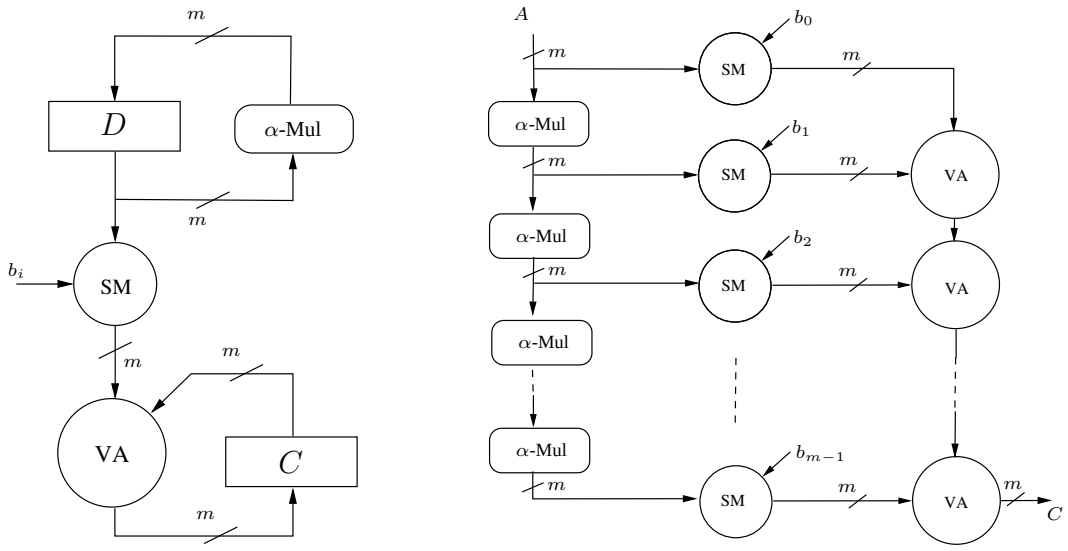
Hereafter, the module that receives $A \in GF(2^m)$ as input and computes αA is called α-Mul module.

Let C be the product of two elements A and B of $GF(2^m)$. Then PB representation of C is as follows:

$$\begin{aligned} C = AB \bmod F(\alpha) &= A \sum_{i=0}^{m-1} b_i \alpha^i \bmod F(\alpha) \\ &= \sum_{i=0}^{m-1} b_i . A^{(i)} = (b_{m-1} . A^{(m-1)} + b_{m-2} . A^{(m-2)} + \\ &\cdots + b_1 . A^{(1)} + b_0 . A^{(0)}). \end{aligned} \tag{1}$$

where $A^{(0)} = A$ and $A^{(i)} = \alpha A^{(i-1)}$. In (1), '.' is a scalar multiplication, since $b_i \in GF(2)$ and $A^{(i)} \in GF(2^m)$, and '+' is a vector addition, since its two operands are the elements of $GF(2^m)$. Modules that perform scalar multiplication and vector addition are hereafter referred to as SM module and VA module, respectively. These two modules and the α-Mul module discussed earlier are the main components of a PB multiplier. In accordance with (1) and using

these three main components, bit-serial and bit-parallel PB multipliers can be constructed as shown in Fig. 1 (see [22] for a similar multiplier architecture).



(a) Bit-serial (b) Bit-parallel

Fig. 1. Polynomial-basis multiplication

3. Concurrent Error Detection Strategy

In this section, an error detection scheme for PB multipliers is presented. Errors may be caused by different types of faults such as open faults, short (bridging) faults, and/or stuck-at faults. Furthermore, the faults can be transient or permanent. The goal of this scheme is to detect as many random errors as possible including single and multiple errors. Towards this goal, we use a parity based method. One-bit parity is able to detect the presence of any odd number of erroneous bits [23]. Here, we use additional parity bits in order to increase error detection capability. In particular, an m-bit input is divided into k parts and for each part one parity bit is used. Thus, the m-bit PB representation of $A \in GF(2^m)$ is divided as follows:

$$A = (A_0, A_1, A_2, \cdots, A_{k-1}).$$

The length of A_j, $0 \leq j \leq k-1$, is

$$l_j = \begin{cases} \lfloor \frac{m}{k} \rfloor + 1 & \text{if } j < m \bmod k; \\ \lfloor \frac{m}{k} \rfloor & \text{otherwise.} \end{cases}$$

For the sake of simplicity, we assume that $k|m$ and the length of each part is $l = \frac{m}{k}$, i.e.,

$$A_j = \alpha^{jk} \sum_{i=0}^{l-1} a_{jk+i}\alpha^i = (a_{jk}, a_{jk+1}, a_{jk+2}, \cdots, a_{jk+l-1}).$$

Parity of A_j is denoted as $P(A_j)$. Using parity bits of A_j's, a k-bit parity of A is formed as follows:

$$P(A) = (P(A_0), P(A_1), P(A_2), \cdots, P(A_{k-1})).$$

Then using the parity $P(A)$, we construct encoded A as follows:

$$E(A) = (A_0, A_1, A_2, \cdots, A_{k-1}, P(A)).$$

Unlike A which is represented with m bits, the field defining irreducible polynomial $F(x)$ requires $m + 1$ bits. In order to have the same length for partitioning, we exclude the leading coefficient of $F(x)$ and divide $F(x) - x^m$ into k parts as follows:

$$F(x) - x^m = (F_0, F_1, \cdots, F_{k-1}).$$

The parity bit of F_j, $0 \leq j \leq k-1$, is denoted as $P(F_j)$.

One of the important issues in detecting errors in the output of a finite field multiplier (or an arbitrary circuit, in general) is parity prediction. The latter refers to the task of determining the parity of the expected outputs by using the corresponding inputs as well as the functionality of the circuit. As mentioned in Section 2, a polynomial basis multiplier consists of three modules: 1) α-Mul module 2) SM module, and 3) VA module. In the following, the parity prediction method for each of these modules will be discussed.

A. Multiple Parity Prediction in α-Mul Module

In the following, the output parity of an α-Mul module is predicted.

Let $A' = \alpha A$, i.e.,

$$A' = \sum_{i=0}^{l-1} a_i \alpha^{i+1} + \alpha^l \sum_{i=0}^{l-1} a_{l+i} \alpha^{i+1} +$$
$$\cdots + \alpha^{(k-1)l} \sum_{i=0}^{l-1} a_{(k-1)l+i} \alpha^{i+1}$$
$$= \left(0 + \sum_{i=1}^{l-1} a_{i-1} \alpha^i\right) + \alpha^l \left(a_{l-1} + \sum_{i=1}^{l-1} a_{l+i-1} \alpha^i\right) +$$
$$\cdots + \alpha^{(k-1)l} \left(a_{(k-1)l-1} + \sum_{i=1}^{l-1} a_{(k-1)l+i-1} \alpha^i\right) + a_{kl-1} \alpha^{kl}.$$

A' must be reduced by $F(\alpha) = \alpha^m + \sum_{j=0}^{k-1} F_j(\alpha)$ as follows:

$$A' \bmod F(\alpha) =$$
$$\left(0 + \sum_{i=1}^{l-1} a_{i-1} \alpha^i\right) + \alpha^l \left(a_{l-1} + \sum_{i=1}^{l-1} a_{l+i-1} \alpha^i\right)$$
$$+ \cdots + \alpha^{(k-1)l} \left(a_{(k-1)l-1} + \sum_{i=1}^{l-1} a_{(k-1)l+i-1} \alpha^i\right)$$
$$+ a_{m-1} \left(\sum_{j=0}^{k-1} F_j(\alpha)\right).$$

Now, we group the expression and obtain

$$A' \bmod F(\alpha) =$$
$$\left(0 + \sum_{i=1}^{l-1} a_{i-1} \alpha^i + a_{m-1} \sum_{i=0}^{l-1} f_i \alpha^i\right)$$
$$+ \alpha^l \left(a_{l-1} + \sum_{i=1}^{l-1} a_{l+i-1} \alpha^i + a_{m-1} \sum_{i=0}^{l-1} f_{l+i} \alpha^i\right)$$
$$+ \cdots + \alpha^{(k-1)l} \left(a_{(k-1)l-1} + \sum_{i=1}^{l-1} a_{(k-1)l+i-1} \alpha^i\right.$$
$$\left. + a_{m-1} \sum_{i=0}^{l-1} f_{(k-1)l+i} \alpha^i\right).$$

Thus, the j^{th} part of A' for $0 \leq j \leq k-1$ can be derived as:

$$A'_j = \alpha^{jl} \left(a_{jl-1} + \sum_{i=1}^{l-1} a_{jl+i-1} \alpha^i + a_{m-1} \sum_{i=0}^{l-1} f_{jl+i} \alpha^i\right) \quad (2)$$

where $a_{-1} = 0$. Fig. 2 shows a circuit diagram implementing A'_j. In practice, many coefficients of $F(x)$ are zero and hence the corresponding XOR gates in Fig. 2 are not needed. By cascading k copies of the circuit shown in Fig. 2, an α-Mul module can be constructed as illustrated in Fig. 3.

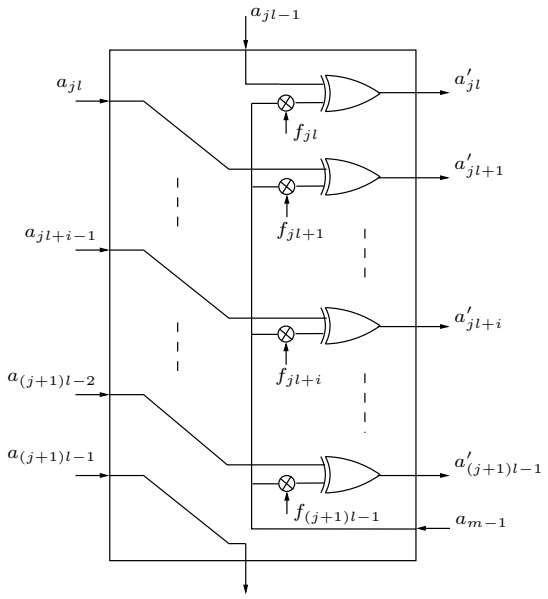


Fig. 2. The j^{th} part of the α-Mul module

Let ω be the Hamming weight of $F(x)$. The total number of two-input XOR gates required in an α-Mul module is $\omega - 2$, since no XOR gate is needed for the first and the last coefficients of $F(x)$.

For parity prediction of the j^{th} part of the α-Mul module, we have the following lemma where $A' = \alpha A$ and $P_{F_j} = \sum_{i=0}^{l-1} f_{jl+i}$.

Lemma 1: Let $P(A_j)$ and $P(A'_j)$ be the parities of the input and the expected output of the j^{th} part of the α-Mul module, respectively. Then,

$$P(A'_j) = a_{jl-1} + P(A_j) + a_{(j+1)l-1} + a_{m-1}P_{F_j}.$$

Proof: Using (2) the proof is immediate. ■

Fig. 4 shows the parity prediction circuit of the j^{th} part of the α-Mul module, where $P(x)$ is predicted parity of x. The parity of the j^{th} part of $F(x)$ is P_{F_j} and is assumed to be known, since it can be pre-computed. Thus, the corresponding AND gate is not really required. On the

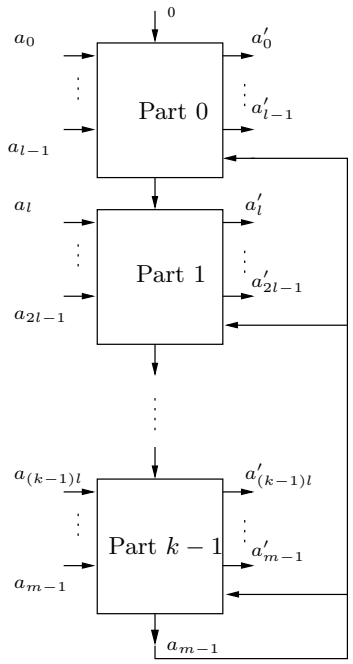


Fig. 3. α-Mul module

other hand, $F(x)$ can be a trinomial or a pentanomial and usually it can be chosen so that the parities of all parts become zero, i.e., $P_{F_j} = 0$ for $0 \leq j \leq k-1$. In this case, the value of $a_{k-1,l-1}$ is not important and one XOR gate is removed. In the worst case the circuit of Fig. 4 can be implemented with 3 two-input XOR gates. The total number of two-input XOR gates for the whole parity prediction circuit is $3k$.

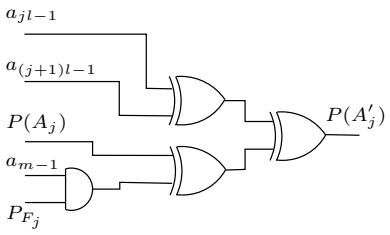


Fig. 4. Parity prediction circuit of the j^{th} part of the α-Mul module

Hereafter, an α-Mul module together with its parity prediction circuit (PPC) is referred to as α-Mul-P module. It should be mentioned that different partitioning of A and F can change the parity prediction circuit of the α-Mul module. Appendix I presents a partitioning of A and

F that reduces the number of XOR gates of each parity prediction circuit by two, i.e., parity prediction circuit can be constructed by only one XOR gate.

B. Parity Prediction in Scalar Multiplication and Vector Addition Modules

In this work, scalar multiplication refers to multiplication of an element of $GF(2)$ by an element of $GF(2^m)$ and vector addition refers to addition of two elements of $GF(2^m)$. For $b_i \in GF(2)$ and $A \in GF(2^m) = (a_0, a_1, \cdots, a_{m-1})$, scalar multiplication of b_i and A is $b_i.A = (b_i a_0, b_i a_1, \cdots, b_i a_{m-1})$. Thus,

$$\begin{aligned} P(b_i.A) &= b_i a_0 + b_i a_1 + \cdots + b_i a_{m-1} \\ &= b_i(a_0 + a_1 + \cdots + a_{m-1}) = b_i P(A). \end{aligned} \tag{3}$$

For $A, B \in GF(2^m)$, vector addition of A and B is:

$$A + B = \sum_{i=0}^{m-1} a_i \alpha^i + \sum_{i=0}^{m-1} b_i \alpha^i = \sum_{i=0}^{m-1} (a_i + b_i)\alpha^i.$$

Thus,

$$\begin{aligned} P(A+B) &= \sum_{i=0}^{m-1} (a_i + b_i) = \sum_{i=0}^{m-1} a_i + \sum_{i=0}^{m-1} b_i \\ &= P(A) + P(B). \end{aligned} \tag{4}$$

The circuit of the parity prediction, as defined in (3) and (4), are shown in Fig. 5 where they need k two-input AND gates and k two-input XOR gates, respectively. These circuits for parity bits are now included with the SM and the VA modules appropriately and the resulting new modules are hereafter referred to as SM-P and VA-P.

C. Parity Checking Circuit

In order to detect errors in the multiple parity scheme, the predicted parity bits should be compared with the corresponding actual parity bits. Actual parity bits are generated by parity generating circuit. Fig. 6 shows the parity generator and the parity checker.

In Fig. 6, Z and $\tilde{Z}$ can be considered as the expected and the actual outputs of one of the three modules discussed earlier. $P(Z)$ and $P(\tilde{Z})$ are k-bit parities of Z and $\tilde{Z}$, respectively. The result of bit by bit comparison of $P(Z)$ and $P(\tilde{Z})$ are ORed to signal any difference which indicates an error. The parity generator is constructed by XOR trees which contain $l-1$ two-input

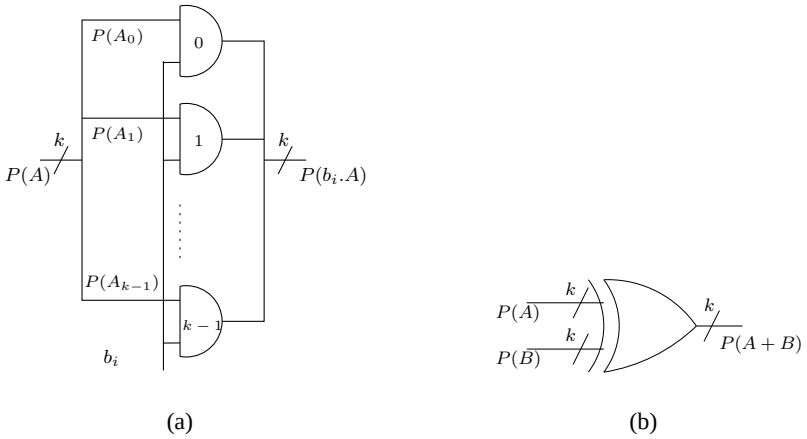


Fig. 5. PPC for a) SM module and b) VA module

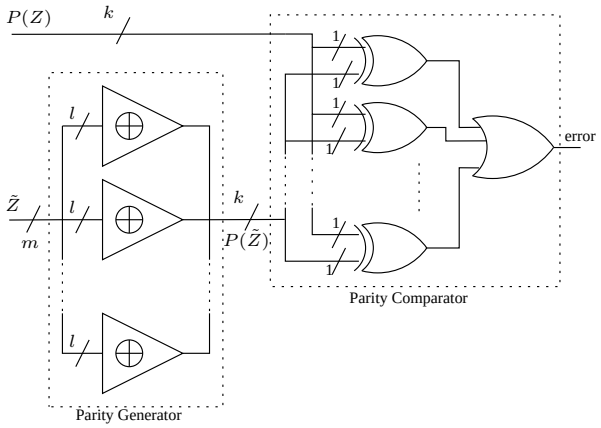


Fig. 6. Multiple-bit parity checker

XOR gates. Furthermore, k two-input XOR gates are required for comparison. Total numbers of two-input XOR and OR gates required for a parity checker are m $(= k(l-1)+k)$ and $k-1$, respectively.

D. Polynomial Basis Multiplier with CED

To construct a bit-serial and a bit-parallel multiplier with concurrent error detection capability, we will use PPC embedded modules α-Mul-P, SM-P, and VA-P. Fig. 7 shows a bit-serial multiplier with PPC. A and B are the inputs of the multiplier. Register D is initialized with A and its k-bit parity $P(A)$. A parity checker can be at each of the three locations: L1, L2 and L3. In the next section, the frequency of check points will be discussed.

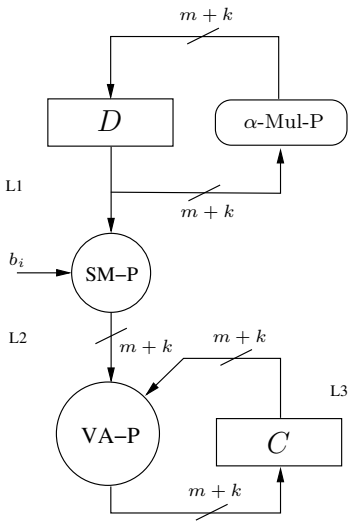


Fig. 7. Bit-serial polynomial basis multipliers with parity prediction circuit

Fig. 8 shows a bit-parallel multiplier with PPC. In the bit-parallel multiplier a parity checker can be placed after each modules. Thus, there can be as many as $3m-2$ error checkers for a bit-parallel multiplier.

4. Error Detection Capability

In this section, first the error model is explained. Then the probability of error detection at the output of the circuit using the multiple parity method is determined. Finally, the frequency of the check points is discussed.

A. Error Modelling

The effect of a fault, such as a transient fault, in one location of the multiplier circuit is modelled by XORing an error vector with the expected correct ”value” of that location. The i^{th} bit of the error vector of a location being one implies that the i^{th} bit of the value of the location has changed from 0 to 1 or vice versa due to a fault. If the location is one of the main components (α-Mul-P, SM-P or VA-P), without loss of generality we can assume that the error vector should be XORed with the output of the component. It is worth mentioning that the parity prediction circuits, parity generators and parity checkers should be fault free or at least self-checking [8]. Since in practice the number of parity bits, k, is much less than the size of the

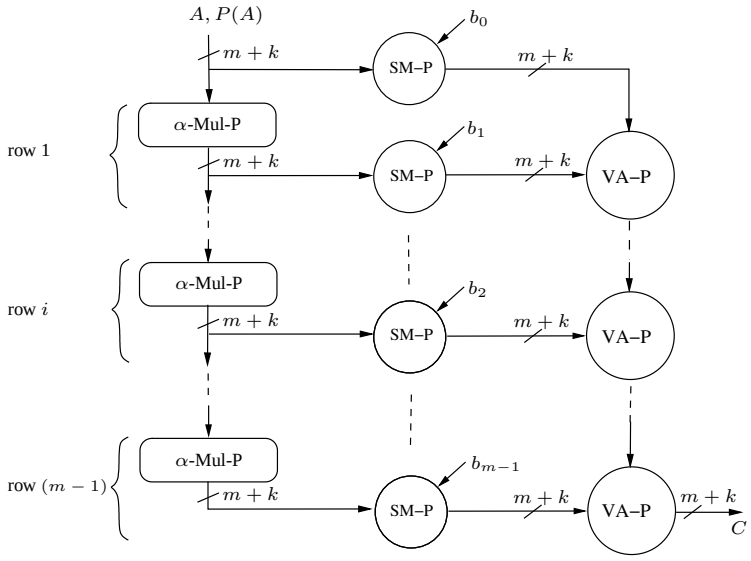


Fig. 8. Bit-parallel polynomial basis multipliers with parity prediction circuit

input operands of the multiplier, m, the self-checking technique is feasible. In this work, these circuits are assumed to be fault free or self-checking. It will be shown in Section 5 that with a moderate number of parity bits the probability of error detection becomes quite close to unity. As an example, for $m = 163$, with 8 parity bits, the error detection probability is approximately 0.996.

Let $e = (e_0, e_1, \cdots, e_{m+k-1})$ be the representation of an error of a location in the multiplier. The first m bits of e correspond to errors in an element, say $A \in GF(2^m)$ that is part of the value of that location. The remaining k bits of e correspond to errors in the k-bit parity vector $P(A)$. Note that although we assume the parity prediction and the parity checking circuits to be fault free or self-checking, an error may occur in the parity bits any where in the remainder of the multiplier circuit such as the registers in the bit-serial implementation of the multiplier or the wires through which the parity signals propagate. If one assumes otherwise, i.e., the parity bits/signals are error free, then all registers and wires through which these signals travel have to be fault free, even though some of these registers and wires are *not* part of the parity prediction and checking circuits.

Since e is an $(m+k)$-tuple vector and the all-zero $e = (0, 0, \cdots, 0)$ corresponds to no error, the number of possible errors is $2^{m+k} - 1$. We logically divide e into k parts each of length

$l+1=\frac{m}{k}+1$ bits where the j^{th} part is

$$(e_{jl}, e_{jl+1}, \cdots, e_{jl+l-1}, e_{m+j}).$$

In the following, we investigate which kind of errors cannot be detected by the k-bit parity scheme.

B. Probability of Error Detection

Let e_O be an odd parity error, i.e., the number of 1's in e_O is odd. Then the parity of at least one of the k partitions is odd. Therefore, e_O can be detected by the proposed CED method and the probability of undetected error is $Pr_U(e_O)=0$.

Let e_E be a nonzero even parity error. Since $k<m$, there is at least one error, e_E, such that all of its partitions have even parity. Then the error cannot be detected. Accordingly, $Pr_U(e_E)\geq 0$.

Theorem 1: Let k be the number of parity bits of the scheme. Suppose p is the probability that $e_i=1$ for $0\leq i\leq m+k-1$. The probability of error detection is given as follows:

$$Pr_D(e)=1-\left[\left(\frac{(1-2p)^{\frac{m}{k}+1}+1}{2}\right)^k-(1-p)^{m+k}\right]. \tag{5}$$

Proof: $Pr_D=1-Pr_U$ where Pr_U is the probability of undetected errors. As it is mentioned, all nonzero errors with even parity in their partitions are undetectable. Thus, considering error vectors are $(m+k)$-bit long and each of them has k partitions, first we need to compute the probability of an $(\frac{m}{k}+1)$-bit number with even parity.

Let E_i and O_i be the probabilities that an i-bit number has even parity and odd parity, respectively. Thus, $E_i=1-O_i$. Moreover, let q be the probability that a bit of the error vector is zero, i.e., $q=1-p$. We proceed in a recursive manner.

$$\begin{aligned} E_{i+1} &= qE_i+pO_i \\ &= (1-p)E_i+p(1-E_i) \\ &= (1-2p)E_i+p. \end{aligned}$$

Let $1-2p=A$ and $p=B$. We determine E_i for some i to find a closed formula:

$$\begin{aligned}
E_0 &= 1 \\
E_1 &= q \\
E_2 &= Aq + B \\
E_3 &= A^2q + AB + B \\
E_4 &= A^3q + A^2B + AB + B \\
&\vdots \\
E_i &= A^{i-1}q + A^{i-2}B + \cdots + AB + B \\
&= A^{i-1}q + B\left(\frac{A^{i-1}-1}{A-1}\right).
\end{aligned}$$

Now, we write the expression only in terms of p:

$$\begin{aligned}
E_i &= (1-2p)^{i-1}(1-p) + p\left(\frac{(1-2p)^{i-1}-1}{(1-2p)-1}\right) \\
&= (1-2p)^{i-1}(1-p) - \frac{(1-2p)^{i-1}-1}{2} \\
&= (1-2p)^{i-1}(1-p-1/2) + 1/2 \\
&= \frac{(1-2p)^i + 1}{2}.
\end{aligned}$$

The probability that an $\left(\frac{m}{k}+1\right)$-bit partition of the error vector has even parity is $E_{i=\frac{m}{k}+1}$. Moreover, the partitions are independent. Thus, the probability of having a vector with even parity in each of its partitions is $\left(E_{i=\frac{m}{k}+1}\right)^k$ or

$$\left(\frac{(1-2p)^{\frac{m}{k}+1}+1}{2}\right)^k.$$

However, the zero vector should be excluded and hence,

$$Pr_U = \left(\frac{(1-2p)^{\frac{m}{k}+1}+1}{2}\right)^k - (1-p)^{m+k}.$$

As a result,

$$Pr_D = 1 - \left[\left(\frac{(1-2p)^{\frac{m}{k}+1}+1}{2}\right)^k - (1-p)^{m+k}\right].$$

■

As mentioned, p is the probability of an error vector bit being one. A reduction of p increases the probability of having an all-zero error vector. This reduction means a reduction in the probability of (nonzero) errors, which in turn means a reduction in the probability of undetectable errors. Thus, with a reduction in p, the probability of error detection increases.

As it can be determined from Equation (5), as the number of parity bits increases, the probability of error detection quickly approaches unity so that it reaches 0.996 for 8 parity bits.

C. Frequency of the Check Points

Suppose that there are several multiple-bit errors in a location of the circuit of a PB multiplier. For having an error detection capability Pr_D as given in Theorem 1, each of the above mentioned locations in Section 3-D should have a parity checker. This causes a very high area overhead especially for bit-parallel multipliers. The following lemma helps us reduce the number of checkers considerably.

Lemma 2: Suppose only a maximum of one multiple-bit error occurs per round of a bit-serial multiplier or per row of a bit-parallel multiplier (see Fig. 7 and Fig. 8). Then any such error can be detected with the probability Pr_D, given in Section 4-B, using a parity checker at L3 of the bit-serial multiplier or a parity checker before the vertical input of every VA-P and one parity checker after the final VA-P in the bit-parallel multiplier.

Proof: It should be verified if a detectable error vector can be changed to an undetectable one after passing through a main component and before reaching one of the check points.

If a detectable error vector passes through an α-Mul-P module, it can be changed to an undetectable one. However, the check points are located so that any error vector can reach one of the check points without passing through any α-Mul-P module. Therefore, one of the following cases should be considered: 1) a detectable error vector passes through an SM-P module or 2) a detectable error vector passes through a VA-P module or 3) both.

In the first case, if $b_i = 0$ then regardless of the other input value, the value of the output vector and parity are zero. This is a correct result and there is no error anymore. If $b_i = 1$ then the input and the output of the SM-P module are equal. Hence, the error vector passes SM-P without any change.

In the second case, if only one of the two inputs of VA-P module has erroneous bits, the error vector can pass the VA-P module without any change. Since a maximum of one multiple-bit error is allowed in a round of a bit-serial multiplier or in a row of a bit-parallel multiplier, only one of the inputs of VA-P can be erroneous.

In the third case, the error must occur before an SM-P module but after the α-Mul-P module (in the corresponding row of a bit-parallel multiplier). Therefore, according to case 1 and case 2, it passes SM-P and VA-P modules and reaches the parity checker. ■

5. RESULTS

Important performance measures for an error detection scheme include error detection capability, area and time overheads. In this section the results of our studies on these measures are presented. The results can guide the choice of a proper number of parity bits for design requirements.

A. Simulation-Based Fault Injection

We have injected stuck-at faults to a $GF(2^{163})$ PB multiplier with $k = 8$ to evaluate the error detection capability of the proposed scheme. The fault injection was performed in a C model of the multiplier. Furthermore, the fault injection was at the gate-level, i.e., stuck-at faults (both stuck-at 1 and stuck-at 0) were injected at the input and output pins of the gates of the multiplier. In the proposed scheme, a checker is placed at the end of a round of a bit-serial multiplier (or at the end of the row of a bit-parallel one). Moreover, the scheme can detect an error if the error can be detected in one round of a bit-serial multiplier (or a row of a bit-parallel one). Fault injection in a complete multiplier of $GF(2^{163})$ is extremely time consuming. In order to reduce the time for completing experiments, faults were injected in only one round of a bit-serial multiplier (and a row of a bit-parallel one). In the following, two phases of our fault injections are presented.

1) Single-Bit Stuck-at Faults: In this experiment, single-bit stuck-at faults were injected at the input or output pins of gates. As shown in Figure 9, to inject a fault at a point, a multiplexer is placed at that point, where the control signal of the multiplexer selects between the original value of that point and the fault. Also the fault can be chosen to be stuck-at 1 or stuck-at 0.

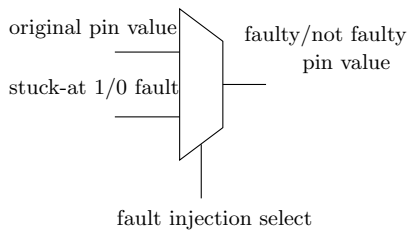


Fig. 9. Fault injection at a gate pin

In a $GF(2^m)$ PB multiplier, there are $\omega - 2$ two-input XOR gates, m two-input AND gates, and m two-input XOR gates in the α-Mul, SM and VA modules, respectively, where ω is the Hamming weight of the field defining polynomial. Single-bit stuck-at faults are injected at all input and output pins except the output pins of AND gates of SM module because they are direct inputs of the VA module's XOR gates. Therefore, the number of locations for single-bit stuck-at fault injections at a round of a bit-serial (or a row of a bit-parallel) multiplier is $3(\omega - 2) + 5m$. Additionally, for each input or output gate pin, two single-bit faults can be injected. Hence, the number of single-bit stuck-at faults that should be injected at a round of a bit-serial (or a row of a bit-parallel) multiplier is $6(\omega - 2) + 10m$.

In this experiment, we simulated the multiplier for one million random inputs and for every input, all the above mentioned single-bit stuck-at faults were injected. As shown in Table I all faults were detected.

Type of stuck-at faults	No. of stuck-at faults[1]	No. of random inputs	Error detection capability
Single-bit	1648	1000000	100%
Multiple-bit	500	1000000	99.61%

[1]in one round of a bit-serial (or one row of a bit-parallel) multiplier

TABLE I

ERROR DETECTION CAPABILITY OF THE SCHEME FOR A $GF(2^{163})$ PB MULTIPLIER AGAINST STUCK-AT FAULTS

2) Multiple-Bit Stuck-at Faults: For multiple-bit stuck-at fault injection, the location of the above mentioned single-bit faults were randomly selected and a stuck-at 0 or stuck-at 1 was

randomly injected there. Furthermore, simulations were performed for one million random inputs and for every input, 500 random multiple-bit stuck-at faults were injected. It is worth mentioning that for a $GF(2^{163})$ multiplier experiment, there are 824 single-bit stuck-at fault locations. Therefore, the number of accesses to those locations, whether or not any fault is injected, is $(1000000 * 500 * 824 =)$ 412 billions, implying that the experiment is very time consuming. As shown in Table I, the error detection capability of the scheme for multiple-bit stuck-at fault injections is 99.61%.

B. Time and Area Overheads

We have described the multiple-bit parity scheme by VHDL to obtain a realistic approximation of area and time overheads. In order to reduce the number of XOR gates in the multiplier, field defining polynomial $F(x)$ can be chosen to be a trinomial or a pentanomial such that the parity of $F(x)$ in each partition is zero, i.e., $P_{F_j} = 0$. In Section I-B, the complexity of the parity prediction circuit for NIST recommended irreducible polynomials for ECDSA is discussed.

We used Modelsim$^{(TM)}$ to simulate the design for checking its correct functionality. We implemented the multiple parity scheme on a Xilinx Spartan 3 (XC3S5000) FPGA using Xilinx ISE 7.1i.

1) Bit-Serial PB Multiplication: The circuit of a complete bit-serial multiplier with CED is shown in Fig. 10. The circuit consists of two major blocks: 1) PB multiplier with PPC and 2) checker. The parity generator of the checker is used at the initialization phase to generate the parity of input A. Note that no extra clock cycle is needed for the circuit shown in Fig. 10 when compared to a bit-serial PB multiplier without CED.

From the first experiment, we obtained the area overhead percentage of the scheme for multipliers of different field sizes. The number of parity bits for this experiment was chosen to be 8 bits since the probability of error detection was within acceptable range for our experiment ($\approx$ 0.996). Furthermore, the defining polynomial of the fields used in the experiment included the NIST recommended irreducible polynomials for ECDSA. Fig. 11 shows the result of the experiment.

As shown in the figure, the area overhead for a fixed number of parity bits tends to decrease as the size of the field increases. The area overhead does not decrease in a strictly monotonic way because the FPGA compiler used in the experiment optimizes the multiplier for different

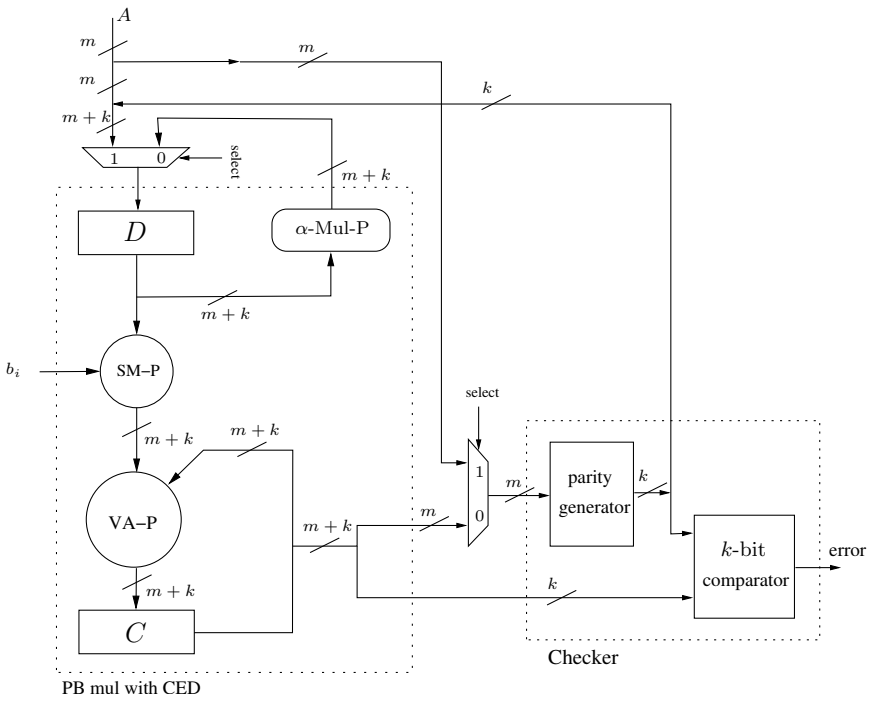


Fig. 10. A complete bit-serial multiplier with CED

field sizes differently. The worst area overhead percentage among the fields implemented is for $GF(2^{201})$ and is still reasonably low, i.e., $< 12\%$.

In the second experiment, we implemented the scheme for $m = 163$ and $m = 283$ using the NIST recommended field defining polynomials for ECDSA $F(x) = x^{163} + x^7 + x^6 + x^3 + 1$ and $F(x) = x^{283} + x^{12} + x^7 + x^5 + 1$, respectively. Both of these polynomials are quite suitable for implementation because the parity prediction circuits of the scheme would be in the simplest form since, in a k-bit parity scheme, we have:

$$\{P(F_i) = 0 \mid 0 \leq i \leq k-1 \text{ and } 2 \leq k \leq 20\}.$$

As shown in Fig. 12, area overhead cost increases as the number of parity bits increases. For all points in each graph depicted in the figure, a line is fitted as follows:

$$\begin{aligned} &\text{for } GF(2^{163}) : \text{overhead (\%)} = 0.50 \times (\text{\# of parity bits}) + 5.94, \\ &\text{for } GF(2^{283}) : \text{overhead (\%)} = 0.30 \times (\text{\# of parity bits}) + 6.44. \end{aligned} \tag{6}$$

As expected according to the first experiment, the slope of the fitted line for $GF(2^{163})$ is more than that for $GF(2^{283})$, i.e., the area overhead increase rate vs. parity-bit numbers in $GF(2^{283})$

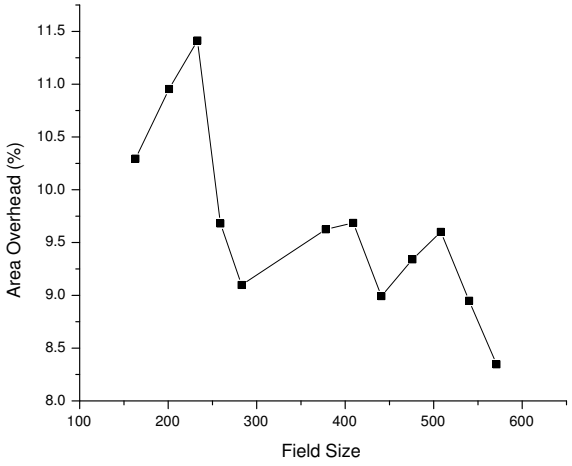


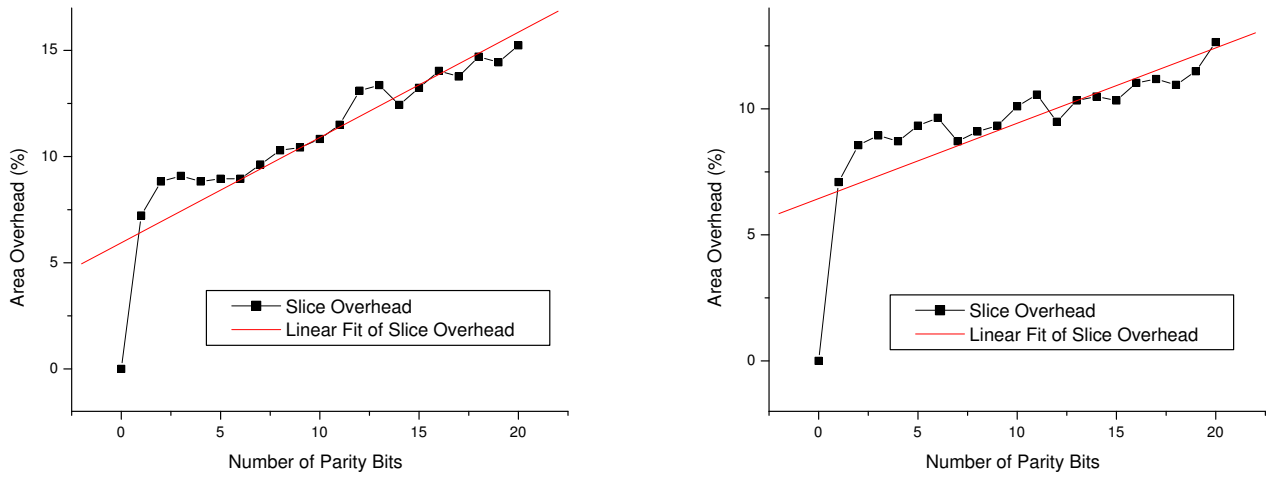
Fig. 11. Area (i.e., slice) overhead for bit-serial PB multipliers for different size of fields

is lower. Furthermore, based on the experimental results, area overhead tends to increase linearly except for very small numbers of parity bits.

Note that Equation (6) implies that even if one parity is used for each information bit, circuit overhead is not expected to be more than 100%, which is the overhead for the conventional dual modular redundant (DMR) scheme.

In the second experiment, we also investigated the time overhead of the $GF(2^{163})$ and $GF(2^{283})$ PB multipliers for different numbers of parity bits. Since there is no extra clock cycle, the time overhead is equal to the clock period overhead. We obtain the clock periods from the post place and route static timing report of Xilinx ISE. Except for four cases, there was no clock period overhead and in turn no time overhead for the bit-serial implementation of the multipliers. These four cases belong to the $GF(2^{163})$ PB multiplier shown in Table II. According to the table, the time overheads even for these cases are small.

2) Bit-Parallel PB Multiplication: A circuit diagram of a complete bit-parallel polynomial basis multiplier with CED is depicted in Fig. 13. The parity checker is very similar to that presented in Fig. 10. As shown in Fig. 13, once the inputs A and B are updated, the results of the multiplication and error detection are ready after certain amount of delay due to the propagation of various signals through the circuit where no clocking is used.



(a) $GF(2^{163})$ (b) $GF(2^{283})$

Fig. 12. Area overhead vs. parity-bit number

No. of parity bits	1	4	11	13
Time overhead (%)	12.27	4.39	15.26	4.79

TABLE II

NON ZERO TIME OVERHEADS FOR BIT-SERIAL IMPLEMENTATION WHICH BELONG TO THE $GF(2^{163})$ PB MULTIPLIER

For bit-parallel multiplier, the first experiment was to measure the area overhead percentage of the eight parity-bit scheme for multipliers of different field sizes. The results show that the area overhead decreases as the field size increases (Fig. 14).

There is a major difference between the structure of bit-serial and bit-parallel PB multipliers and this affects the area overhead considerably. A bit-serial PB multiplier contains simple and shift registers, but a bit-parallel multiplier does not. Basically, registers are relatively area consuming components in FPGAs. Therefore, assuming that one wants to implement a PB multiplier for a field of size m, the area (in terms of slices) needed for a bit-parallel PB multiplier without CED is significantly smaller than m times the area needed for a bit-serial multiplier. Accordingly, CED overhead on a bit-parallel PB multiplier is much higher than that on a bit-serial one. This fact can be observed easily in the experiments reported in this section.

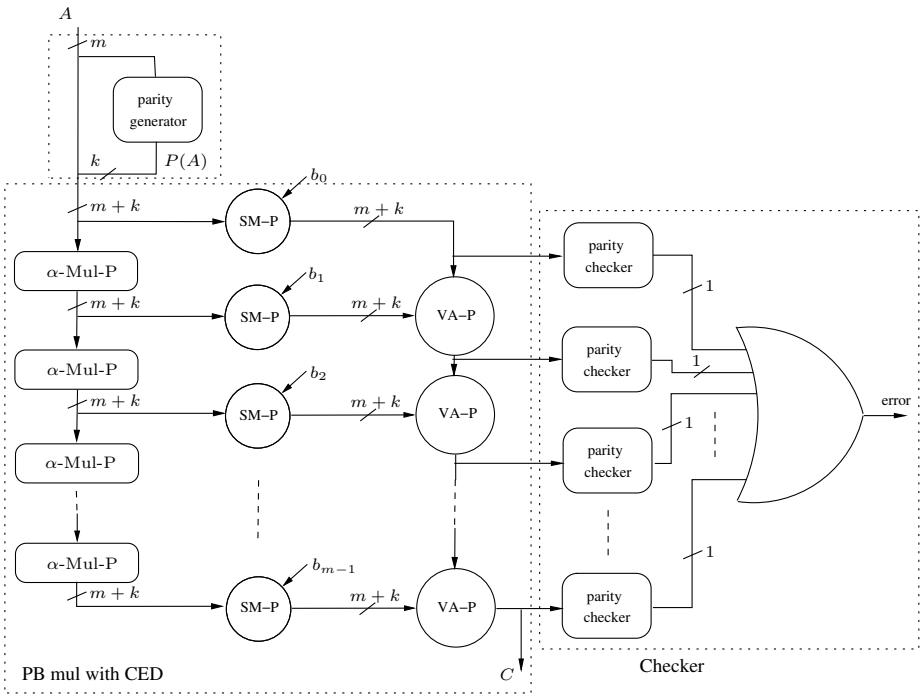


Fig. 13. A complete bit-parallel multiplier with CED

The second experiment was to investigate the area and time overheads' increase rates vs. the number of parity bits for the field $GF(2^{163})$ (see Fig. 15). The field defining polynomial is $F(x) = x^{163}+x^7+x^6+x^3+1$. According to Table III, the bit-parallel implementation is very area consuming; therefore, similar experiments for the field $GF(2^{283})$ are extremely time consuming and clearly that design does not fit into our current FPGA. However, the area overhead results for higher values of m are expected to be better than the result of this experiment as one can infer from Fig. 14, where the number of parity bits is fixed to eight.

Fig. 15 illustrates that as the number of parity bits increases, the area overhead for a bit-parallel implementation increases at a greater rate compared to the bit-serial implementation. However, the area overhead may be still acceptable for some applications. This is because for obtaining a sufficiently high probability of error detection (say ≈ 0.996), one needs only about 8 parity bits in the proposed scheme and it results in about 50% area overhead, which is better than 100% overhead of the DMR scheme.

In the bit-parallel implementation, the time overhead is the delay of the critical path, i.e., the

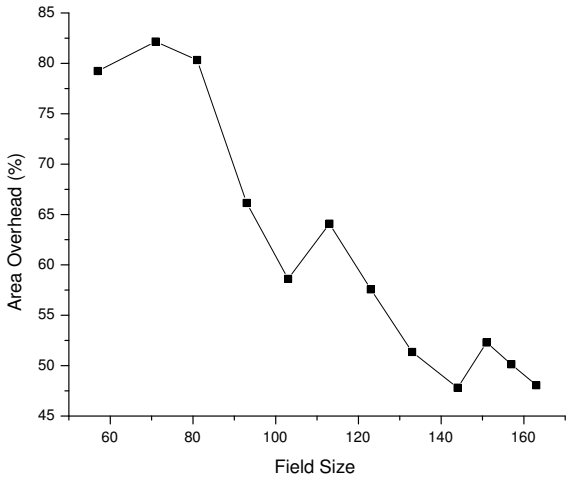


Fig. 14. Area (i.e., slice) overhead for bit-parallel PB multipliers for different size of fields

No. of parity bits	Without CED	4	8	12	16	20
Number of required Slices	13541	19121	20049	21616	22864	24390
FPGA area consumption (%)[1]	40.69	57.45	60.24	64.95	68.70	73.29

[1]The total number of slices in a Xilinx Spartan 3 (XC3S5000) FPGA is 33280.

TABLE III

FPGA AREA CONSUMPTION FOR A BIT-PARALLEL $GF(2^{163})$ PB MULTIPLIER

maximum propagation delay from one of the input pins to one of the output pins. We obtain the delay of all input pins to output pins from the post place and route static timing report of Xilinx ISE. The time overhead for the bit-parallel implementation of a $GF(2^{163})$ PB multiplier vs. number of parity bits is given in Fig 16, which shows that the time overhead is generally less than 25% when more than a couple of parity bits are used.

6. CONCLUSIONS

In this paper, a multiple parity error detection scheme is presented for concurrent detection of errors in polynomial basis multipliers. In this scheme, the probability of error detection for

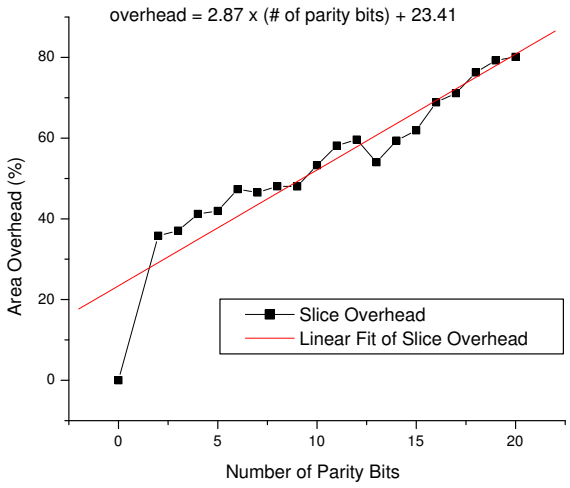


Fig. 15. Area overhead vs. parity-bit number for the field $GF(2^{163})$

random errors is more than 75% and it quickly approaches unity for approximately 8 parity bits. The overhead of our implementation tends to increase linearly as the number of parity bits increases. Results show that the area overhead cost of the bit-serial implementation is lower than that for the bit-parallel one. Both implementations have lower area overheads than the traditional dual modular redundant scheme for a sufficient number of parity bits. Additionally, the average time overhead due to the use of the scheme in bit-parallel implementations is around 25%, while for bit-serial implementations time overheads have been observed to be small to negligible. It is hoped that using the results presented in this paper, one will be able to choose an appropriate number of parity bits for specific applications.

APPENDIX I
ALTERNATIVE PARTITIONING

In this section another partitioning of A and F is presented. The new partitioning reduces the overhead of the parity prediction circuit of the α-Mul module.

As mentioned $A = \sum_{i=0}^{m-1} a_i \alpha^i$ is partitioned into k parts. As before, we assume that m is

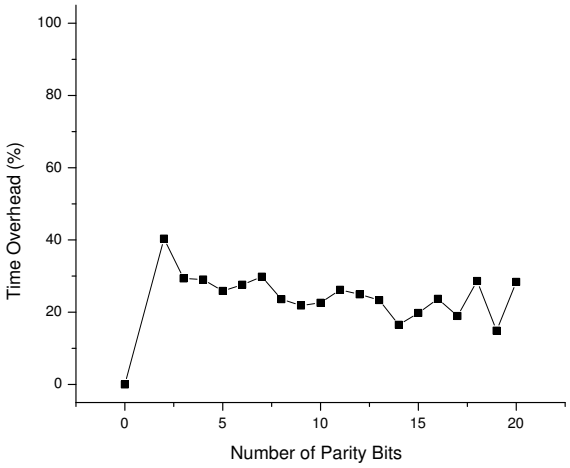


Fig. 16. Time overhead vs. parity-bit number for the field $GF(2^{163})$

divisible by k and $l = m/k$. The alternative (vertical) partitioning is illustrated below:

$$\begin{array}{ccccccccccc}
a_0 & , & a_1 & , & a_2 & , & \cdots & , & a_{k-1} & , \\
a_k & , & a_{k+1} & , & a_{k+2} & , & \cdots & , & a_{2k-1} & , \\
\vdots & , & & , & \ddots & , & & , & \vdots & , \\
a_{(l-1)k} & , & a_{(l-1)k+1} & , & a_{(l-1)k+2} & , & \cdots & , & a_{lk-1} & \\
\underbrace{\qquad} & & \underbrace{\qquad} & & \underbrace{\qquad} & & & & \underbrace{\qquad} & \\
A_0 & & A_1 & & A_2 & & \cdots & & A_{k-1} &
\end{array}$$

For $0 \leq j \leq k-1$, the j^{th} partition is:

$$A_j = \sum_{i=0}^{l-1} a_{ik+j}\alpha^{ik+j} = (a_j, a_{k+j}, a_{2k+j}, \cdots, a_{(l-1)k+j}).$$

A. Structure of α-Mul Module

$$
\begin{aligned}
A' =& \alpha A \bmod F(\alpha) \\
=& \sum_{j=0}^{k-1}\sum_{i=0}^{l-1} a_{ik+j}\alpha^{ik+j+1} \bmod F(\alpha) \\
=& \sum_{j=1}^{k}\sum_{i=0}^{l-1} a_{ik+j-1}\alpha^{ik+j} \bmod F(\alpha) \\
=& \sum_{j=1}^{k-1}\sum_{i=0}^{l-1} a_{ik+j-1}\alpha^{ik+j} + \sum_{i=0}^{l-2} a_{k(i+1)-1}\alpha^{k(i+1)} \\
& + (a_{m-1}\alpha^m \bmod F(\alpha)) \\
=& \sum_{j=1}^{k-1}\sum_{i=0}^{l-1} a_{ik+j-1}\alpha^{ik+j} + \sum_{i=1}^{l-1} a_{ki-1}\alpha^{ki} + a_{m-1}\sum_{i=0}^{m-1} f_i\alpha^i \\
=& \sum_{j=1}^{k-1}\sum_{i=0}^{l-1} \left(a_{ik+j-1} + a_{m-1}f_{ik+j}\right)\alpha^{ik+j} \\
& + \sum_{i=0}^{l-1} \left(a_{ki-1} + a_{m-1}f_{ki}\right)\alpha^{ki} \\
=& \sum_{j=0}^{k-1}\sum_{i=0}^{l-1} \left(a_{ik+j-1} + a_{m-1}f_{ik+j}\right)\alpha^{ik+j}
\end{aligned}
\tag{7}
$$

where $a_{-1} = 0$.

Fig. 17 shows the j^{th} part of the α-Mul module. The complete α-Mul module is shown in Fig. 18. The number of gates is exactly the same as for the previous α-Mul module mentioned in Section 3-A, as only the position of the coordinates is changed.

The following lemma discusses parity prediction in the j^{th} part of the α-Mul module.

Lemma 3: Let $P(A_j)$ and $P(A'_j)$ be the input and the expected output parities of the j^{th} part of the α-Mul module, respectively and $P_{F_j} = \sum_{i=0}^{l-1} f_{ik+j}$. Then,

$$
P(A'_j) = \begin{cases} P(A_{j-1}) + a_{m-1}P_{F_j} & \text{if} \quad 1 \le j \le k-1, \\ P(A_{k-1}) + a_{m-1}(P_{F_0}+1) & \text{if} \quad j = 0. \end{cases}
$$

Proof: According to (7), we have:

$$
A'_j = \sum_{i=0}^{l-1} \left(a_{ik+j-1} + a_{m-1}f_{ik+j}\right)\alpha^{ik+j}.
$$

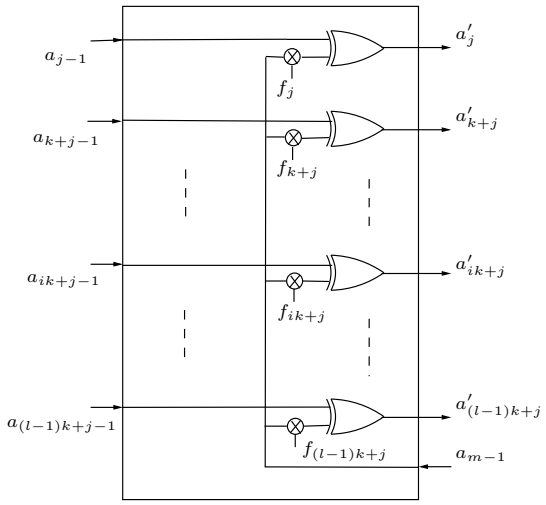


Fig. 17. The j^{th} part of the α-Mul module

Therefore, for $1 \leq j \leq k-1$, we have:

$$P(A'_j) = P\left(\sum_{i=0}^{l-1} a_{ik+j-1}\alpha^{ik+j}\right) + P\left(\sum_{i=0}^{l-1} a_{m-1}f_{ik+j}\alpha^{ik+j}\right) = P(A_{j-1}) + a_{m-1}P_{F_j}.$$

For $j = 0$, we have:

$$\begin{aligned} P(A'_0) &= P\left(\sum_{i=0}^{l-1} a_{ik-1}\alpha^{ik}\right) + P\left(\sum_{i=0}^{l-1} a_{m-1}f_{ik}\alpha^{ik}\right) \\ &= (P(A_{k-1}) + a_{m-1}) + a_{m-1}P_{F_0} \\ &= P(A_{k-1}) + a_{m-1}(P_{F_0} + 1). \end{aligned}$$

■

P_{F_j}'s can be pre-computed. Therefore, the maximum number of gates required for the parity prediction circuit of each part of the α-Mul module is one XOR gate. No XOR gate is needed for the parity prediction circuit of a part of the α-Mul module when $P_{F_0} = 1$ or $P_{F_j} = 0$ for $0 < j < k$. Furthermore, the probability of error detection can be computed by Theorem 1, since the conditions are the same.

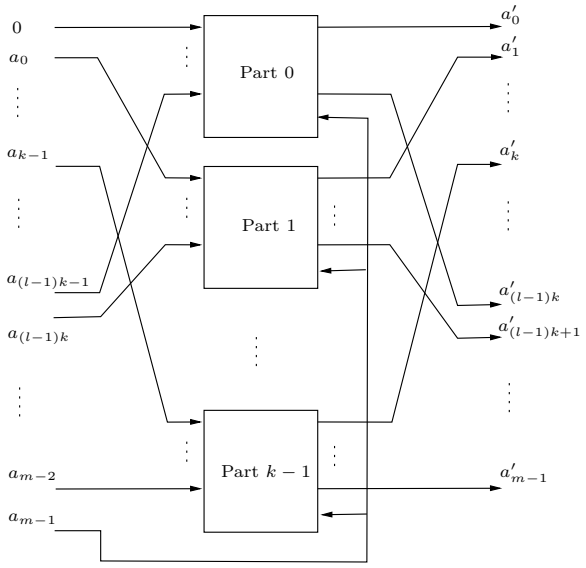


Fig. 18. α-Mul module

Irreducible polynomials	No. of nonzero-parity partitions		No. of 2-input XOR gates for PPC of α-Mul-P	
	Horizontal partitioning	Vertical partitioning	Horizontal partitioning	Vertical partitioning
$F(x) = x^{163} + x^7 + x^6 + x^3 + 1$	0	4	15	4
$F(x) = x^{233} + x^{74} + 1$	2	2	17	2
$F(x) = x^{283} + x^{12} + x^7 + x^5 + 1$	0	4	15	4
$F(x) = x^{409} + x^{87} + 1$	2	2	17	2
$F(x) = x^{571} + x^{10} + x^5 + x^2 + 1$	0	2	15	2

TABLE IV

XOR COUNTS FOR PPC OF AN α-MUL MODULE FOR NIST RECOMMENDED IRREDUCIBLE POLYNOMIALS FOR ECDSA APPLICATION

B. Comparison of α-Mul-P Modules

According to Section 5-A, the scheme with eight partitions results in a fairly high probability of error detection for values of m that are of interest for elliptic curve cryptosystems. Therefore, we have divided each of corresponding NIST recommended irreducible polynomials into eight partitions using our horizontal and vertical partitioning methods. Table IV gives the number of partitions with nonzero parity and the number of required two-input XOR gates for PPC of the

α-Mul module along with the NIST recommended irreducible polynomials.

As it can be seen in Table IV, the α-Mul-P module is relatively area efficient in the vertical paritioning than the horizontal partitioning. However, the α-Mul-P module is much less resource consuming than any of the SM-P and VA-P modules. Therefore, the overheads resulting from the vertical partitioning are expected to be very similar to those presented in Section 5 for horizontal partitioning.

ACKNOWLEDGMENTS

A preliminary version of this paper was presented at the 20th IEEE International Symposium on Defect and fault Tolerance in VLSI Systems [24]. The work was supported in part by an NSERC Strategic Project-grant awarded to Dr. Hasan. The authors also would like to thank Dr. Miguel F. Anjos for letting us run part of our simulations on his computer.

REFERENCES

[1] D. A. McGrew and J. Viega, "The Galois/Counter mode of operation (GCM)." NIST Modes of Operation for Symmetric Key Block Ciphers, 2005, http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-revised-spec.pdf.

[2] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469– 472, July 1985.

[3] I. Biehl, B. Meyer, and V. Muller, "Differential fault attacks on elliptic curve cryptosystems," in *Proceedings of the 20th Annual International Cryptology Conference (CRYPTO)*. Springer-Verlag, 2000, pp. 131–146.

[4] M. Ciet and M. Joye, "Elliptic curve cryptosystems in the presence of permanent and transient faults." *Designs, Codes and Cryptography*, vol. 36, no. 1, pp. 33–43, July 2005.

[5] J. Blomer, M. Otto, and J.-P. Seifert, "Sign change fault attacks on elliptic curve cryptosystems." Cryptology eprint archive, Report 2004/227, 2004, http://eprint.iacr.org/2004/227.

[6] S. B. Wicker and V. K. Bhargava, Eds., *Reed-Solomon Codes and Their Applications*. New York, NY, USA: John Wiley & Sons, Inc., 1999.

[7] D. Pradhan and M. Chatterjee, "Glfsr-a new test pattern generator for built-in-self-test," in *Proceedings of the International Test Conference*, 1994, pp. 481–490.

[8] T. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems.*, E. J. McCluske, Ed. Prentice Hall, Inc., 1989.

[9] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "Error analysis and detection procedures for a hardware implementation of the advanced encryption standard." *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 1–14, April 2003.

[10] C. W. Chiou, "Concurrent error detection in array multipliers for $GF(2^m)$ fields." *Electronics Letters*, vol. 38, no. 14, pp. 688–689, July 2002.

[11] S. Fenn, M. Gossel, M. Benaissa, and D. Taylor, "Online error detection for bit-serial multipliers in $GF(2^m)$." *Journal of Electronics Testing: Theory and Applications*, vol. 13, pp. 29–40, 1998.

[12] N. Joshi, K. Wu, and R. Karri, "Concurrent error detection for involutional functions with applications in fault-tolerant cryptographic hardware design." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1163– 1169, June 2006.

[13] R. Karri, K. Wu, P. Mishra, and Y. Kim, "Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1509–1517, December 2002.

[14] A. Reyhani-Masoleh and M. A. Hasan, "Error detection in polynomial basis multipliers over binary extension fields." in *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer-Verlag, 2002, pp. 515–528.

[15] ——, "Towards fault-tolerant cryptographic computations over finite fields." *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 3, pp. 593–613, August 2004.

[16] ——, "Fault detection architectures for field multiplication using polynomial bases." *IEEE Transactions on Computers-Special Issue on Fault Diagnosis and Tolerance in Cryptography*, vol. 55, no. 9, pp. 1089–1103, September 2006.

[17] K. Wu, R. Karri, G. Kuznetsov, and M. Goessel, "Parity based concurrent error detection for the advanced encryption standard." in *Proceedings of the IEEE International Test Conference (ITC)*, October 2004.

[18] G. B. Agnew, T. Beth, R. Mullin, and S. Vanstone, "Arithmetic operations in $GF(2^m)$." *Journal of Cryptography*, vol. 6, no. 1, pp. 3–13, 1993.

[19] H. Wu and M. Hasan, "Efficient exponentiation of a primitive root in $GF(2^m)$." *IEEE Transactions on Computers*, vol. 46, no. 2, pp. 162–172, February 1997.

[20] E. Fujiwara, N. Mutoh, and K. Matsuoka, "A self-testing group-parity prediction checker and its use for built-in testing." *IEEE Transactions on Computers*, vol. C-33, no. 6, pp. 578– 583, June 1984.

[21] E. S. Sogomonian, "Design of built-in self-checking monitoring circuits for combinational devices." *Automation and Remote Control*, vol. 35, no. 2, pp. 280–289, July 1974.

[22] L. Song and K. Parhi, "Low energy digit-serial/parallel finite field multipliers." *Journal of VLSI Signal Processing*, vol. 19, no. 2, pp. 149–166, 1998.

[23] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications.*, F. F. Kuo, Ed. Prentice Hall, Inc., 1983.

[24] S. Bayat-Sarmadi and M. A. Hasan, "Concurrent error detection of polynomial basis multiplication over extension fields using a multiple-bit parity scheme," in *Proceedings of the 20th IEEE International Symposium on Defect and fault Tolerance in VLSI Systems (DFT)*, Monterey, CA, 2005, pp. 102–110.