Concurrent Error Detection in Polynomial Basis Multiplication Using Linear Codes

Siavash Bayat-Saramdi and M.A. Hasan

University of Waterloo, Ontario, Canada

September 15, 2006

Abstract

A cryptosystem implemented in hardware may produce erroneous results due to faults occurred by natural causes or injected deliberately by an attacker. In this work we consider strategies to detect such errors in polynomial basis multiplication units, which are dominant building blocks in some cryptosystems. Error detection is performed in a concurrent manner using a class of linear codes. Results of our single-input encoding strategy show that area overhead and the probability of error detection using eight redundant bits are 23.2% and 0.996, respectively, for bit-serial implementation of the multiplication unit.

Index Terms

polynomial basis multiplication, concurrent error detection, finite field, linear code.

I. Introduction

Hardware implementation of a high performance cryptosystem requires significant amount of circuits. For example, a bit-parallel $GF(2^{571})$ multiplier alone is likely to require more than a million transistors. During the use of the cryptosystem, faults in those circuits may occur either due to natural causes or deliberate fault injection by an attacker (see for example [2], [3], [4], [5]). Faulty circuits are likely to generate erroneous results that may make the whole cryptosystem unusable or may help the attacker to break the system. As a result, error correction and detection are important for cryptographic hardware.

In this work, we consider detection of errors in polynomial basis finite field multipliers. Our proposed scheme detects certain errors while the multiplier is working (i.e., concurrent error

detection). In order to detect errors in finite field multipliers, a number of schemes have been proposed in the recent past. One approach to detect errors in a finite field multiplier is to use parity bits, see for example [1], [6], [10], [11], [12]. Another 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, see for example [7].

This article presents a scheme for the detection of errors in both bit-serial and bit-parallel polynomial basis multipliers over binary extension fields based on the second approach. In this scheme, we use linear codes. Such codes have also been used in [7]. Important differences between this work and [7] are as follows. First, the error model of this work is more generic and the error can occur in any location of the circuit. Secondly, this work gives much more flexibility to choose the field defining and the code generator polynomials. This leads to a reduction in the number of redundant bits and in turn a reduction in the area overhead. In our work, the error detection capability of the proposed scheme is given. The time and area overhead of the scheme is also investigated. The proposed scheme can be applied to any finite field $GF(2^m)$. Our experimental results for bit-serial implementation using the proposed single-input encoding scheme show that with 8 redundant bits the error detection probability is 0.996 and the area overhead is 23.2%, which is lower than the overhead of the conventional *dual modular redundant* systems. Furthermore, the proposed error detection scheme does not increase the number of clock cycles required for the multiplication.

The organization of this article is as follows. In Section II, some preliminaries about polynomial basis multiplication and coding theory are discussed. A concurrent error detection strategy is presented in Section III. The error detection capability of the scheme is investigated in Section IV. In Section V, the time and area overheads for the scheme are presented. A second strategy for error detection is explained in Section VI. Finally, Section VII gives a few concluding remarks.

II. Preliminaries

In this section, first polynomial basis multiplication is briefly reviewed. Then a class of linear codes is explained.

A. Polynomial Basis Multipliers

Let $f(x) = x^m + \sum_{i=1}^{m-1} f_i x^i + 1$ be an irreducible polynomial over $GF(2)$ of degree m. Polynomial (or canonical) basis is defined as the following set:

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

Each element A of $GF(2^m)$ can be represented using the polynomial basis (PB) as $A = \sum_{i=0}^{m-1} a_i x^i$ where $a_i \in GF(2)$. Let C be the product of two elements A and B of $GF(2^m)$. Then PB representation of C is as follows:

$$\begin{aligned} C = AB \bmod f(x) &= A \sum_{i=0}^{m-1} b_i x^i \bmod f(x) \\ &= \sum_{i=0}^{m-1} b_i.A_i = (b_{m-1}.A_{m-1} + b_{m-2}.A_{m-2} + \cdots + b_1.A_1 + b_0.A_0), \end{aligned} \tag{1}$$

where $A_0 = A$ and $A_i = xA_{i-1} \bmod f(x)$. The multiplication of x and an arbitrary element A of $GF(2^m)$ is performed as follows:

$$\begin{aligned} xA &= x \sum_{i=0}^{m-1} a_i x^i \bmod f(x) \\ &= a_{m-1} + \sum_{i=1}^{m-1} \left(a_{m-1} f_i + a_{i-1}\right) x^i. \end{aligned} \tag{2}$$

Hereafter, the hardware that receives $A \in GF(2^m)$ as input and generates $xA \bmod f(x)$ as output will be referred to as Shift-and-Reduce (SR) module. In (1), '.' denotes a scalar multiplication of $b_i \in GF(2)$ and $A_i \in GF(2^m)$, and '+' is a vector addition of two elements of $GF(2^m)$. Hardware for scalar multiplication and that for vector addition are hereafter referred to as SM and VA modules, respectively. Using SR, SM, and VA modules, one can construct PB multipliers in accordance with (1). For bit-serial implementation, in addition to these modules, registers are used for storing intermediate results.

B. Linear Codes

In an (n, m) block code, the input information sequence is divided into m-bit blocks and each block is encoded to an n-bit codeword $(n > m)$. One important class of block codes is linear codes. These are extensively used in communication applications for correcting/detecting errors

in transmission channels. Here, the binary linear codes are considered for detecting errors in the polynomial basis multipliers. In the simplest form, an (n, m) block code is linear if and only if the modulo-2 addition of two codewords is also a codeword.

Let $V = (v_0, v_1, \cdots, v_{n-1})$ be a codeword. A polynomial whose coefficients are the components of V, is said to be a code polynomial. A code polynomial of degree up to $n-1$ is generated with a polynomial of degree $n-m$ of the following form:

$$g(x) = 1 + g_1 x + g_2 x^2 + \cdots + g_{n-m-1} x^{n-m-1} + x^{n-m}.$$

Polynomial $g(x)$ is called a generator polynomial. Every code polynomial in the code is a multiple of $g(x)$. In fact, our (n, m) linear code, which hereafter is referred to as $\mathcal{L}$ code, maps an element of a finite field $GF(2^m)$ to an element of a communicative ring with modulus $f(x)g(x)$, where $f(x)$ is the irreducible polynomial used for representing the elements of $GF(2^m)$.

Note that the well-known cyclic code has the corresponding modulus as $x^n - 1$. For given $f(x)$ and n, the use of cyclic codes, however, limits the number of choices of $g(x)$.

III. Concurrent Error Detection Strategy

In this section, $\mathcal{L}$ codes are used for concurrent error detection of polynomial basis multipliers. 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. We assume that locations of these faults, occurred naturally or injected by an attacker, are random. In this article, we investigate two strategies for detecting errors. In the first strategy, which lays foundation of discussions for the second strategy, only one of the inputs of the PB multiplier is encoded, i.e., it is multiplied by generator $g(x)$. The second input is not encoded. In the second strategy, both inputs are encoded. In general, they can be encoded with two different generators, $g_1(x)$ and $g_2(x)$. The first and the second strategies are referred to as single-input encoding (SIE) and double-input encoding (DIE), respectively. As expected, DIE has better error detection capability but its area overhead is higher. Nevertheless, the probability of error detection of SIE can be within an acceptable range because for some applications, for example in an elliptic curve cryptographic processor, the second input either comes from other operations such as adders and multipliers or comes as the direct input to the multiplier. In the first case, if the previous operation has an error detection circuitry, its output, which is the second input of the current multiplier, is

expected to be error free. In the second case, one can use a concurrent error detection technique for the input of the multiplier once to avoid faulty inputs. Depending on the further use of the multiplier's output, the PB multiplier with one of these strategies can produce either an encoded output, i.e., multiplied by only one generator, or an unencoded output.

As mentioned in Section II, a PB multiplier can be constructed with three types of modules: 1) SR, 2) SM, and 3) VA. In the following, (n, m) $\mathcal{L}$ codes are applied to the inputs of these modules to obtain error detectable multipliers. For bit-serial implementation, clearly, the size of registers should increase from m bits to n bits.

A. SM and VA Modules

Suppose that an (n, m) $\mathcal{L}$ code is used and $g(x)$ is the generator polynomial. Let A, B, S and $P \in GF(2^m)$ and $b \in GF(2)$, where scalar multiplication $b.A = P$ and vector addition $A + B = S$. Suppose A', B', S' and $P' \in GF(2^n)$ are the results of encoding A, B, S and P, respectively. Thus, for scalar multiplication we have:

$$b.A' = b.Ag = Pg = P',$$

and for vector addition we have:

$$A' + B' = Ag + Bg = (A + B)g = Sg = S'.$$

Accordingly, for using $\mathcal{L}$ codes, the sizes of SM and VA modules should increase from m bits to n bits each.

B. SR Module

As shown in Fig. 1.a, the unencoded input and the output of the SR module are $U(x) = \sum_{i=0}^{m-1} u_i x^i$ and $U_s(x) = \sum_{i=0}^{m-1} u_{s_i} x^i$, respectively. The code generator polynomial, $g(x)$, over $GF(2)$ of degree $n - m$ is used for encoding. The encoded input and the output of the SR module (see Fig. 1.b) are $V(x) = \sum_{i=0}^{n-1} v_i x^i$ and $V_s(x) = \sum_{i=0}^{n-1} v_{s_i} x^i$, respectively.

In an SR module with unencoded input, we have:

$$U_s(x) = xU(x) \bmod f(x).$$

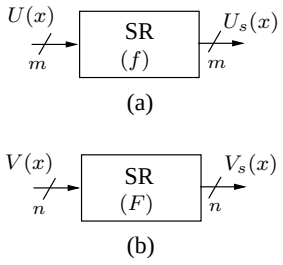

Fig. 1. SR module: (a) with unencoded input, SR depends on $f(x)$, (b) with encoded input, SR depends on $F(x)$

According to (2):

$$\begin{aligned} U_s(x) &= \sum_{i=0}^{m-2} u_i x^{i+1} + u_{m-1} \sum_{i=0}^{m-1} f_i x^i = x \sum_{i=0}^{m-1} u_i x^i + u_{m-1} \left(x^m + \sum_{i=0}^{m-1} f_i x^i \right) \\ &= xU(x) + u_{m-1} f(x). \end{aligned} \tag{3}$$

On the other hand, for encoded inputs to SR module we have:

$$V(x) = U(x)g(x). \tag{4}$$

Thus, using (3) and (4), for input $V(x)$ the output of the SR module is:

$$\begin{aligned} V_s(x) = U_s(x)g(x) &= xU(x)g(x) + u_{m-1} f(x)g(x) \\ &= xV(x) + u_{m-1} f(x)g(x). \end{aligned} \tag{5}$$

Let $F(x) = f(x)g(x)$. Since $F(x)$ can be considered to be fixed, it can be pre-computed. On the other hand, $v_{n-1} = u_{m-1}.g_{n-m}$ and $g_{n-m} = 1$, thus:

$$v_{n-1} = u_{m-1}. \tag{6}$$

Therefore, using (5) and (6) we have:

$$V_s(x) = xV(x) + v_{n-1}F(x) \tag{7}$$

Remark 1: Let $\omega(F)$ be the Hamming weight of $F(x)$. The number of XOR gates required for constructing the SR module with encoded input, shown in Fig. 2, is $\omega(F) - 2$.

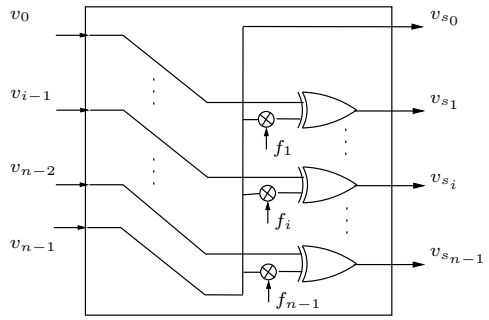

Fig. 2. SR module with encoded input

C. Bit-serial and Bit-parallel Polynomial Basis Multipliers

To construct a bit-serial and a bit-parallel multiplier with concurrent error detection capability, we will use updated versions of SR, SM, and VA modules with encoded input. Fig. 3.a shows a bit-serial multiplier with CED capability. For multiplying A and B with CED capability, register D is initialized with encoded A, i.e., A'. An error checker can be placed at each of the three locations: L1, L2 and L3. In the next section, the frequency of check points will be discussed.

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

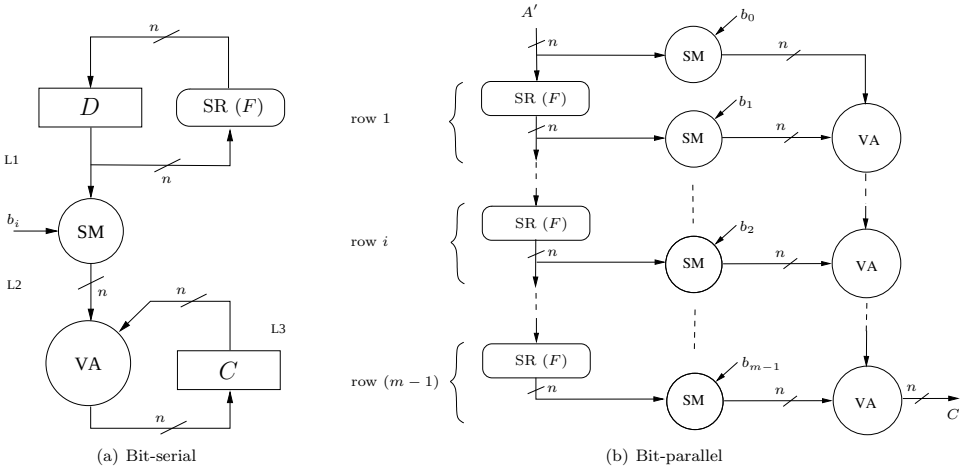

Fig. 3. Polynomial-basis multiplication

D. $\mathcal{L}$ Code Encoders and Checkers

Encoders, decoders and/or checkers of linear codes are well studied in the literature, e.g., see [8] for shift register based architecture. Below, this approach is used for adopting encoders and checkers of the $\mathcal{L}$ code.

For encoding, data (i.e., an element of $GF(2^m)$) is multiplied by generator polynomial, $g(x)$. The encoder can be implemented in serial fashion using shift registers and combinational gates or in parallel fashion using only combinational circuits.

A serial implementation of an encoder is shown in Fig. 4. The initial content of the flip flops are zero. In every clock cycle, each data bit is shifted into the encoder serially and is multiplied by the coefficients of $g(x)$ and the results are added to the previous content of the flip flops. After m clock cycles, the flip flops contain the corresponding codeword. For parallel implementation of an encoder, a parallel multiplier that multiplies the data by a generator $g(x)$ should be used.

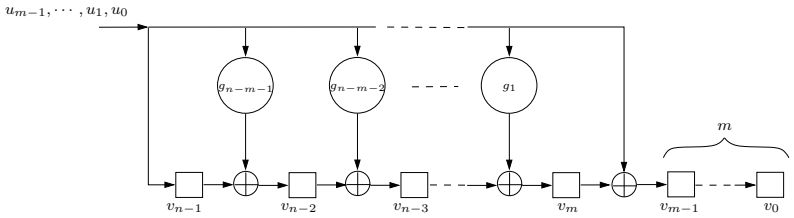

Fig. 4. Serial encoder of an $\mathcal{L}$ code

To check whether an n-tuple at a certain location in the circuit is a codeword, a checker is placed at that point. A checker, basically, divides the polynomial corresponding to the n-tuple by the generator polynomial, $g(x)$, of the the $\mathcal{L}$ code and if the division has a nonzero remainder, an error signal is given. Again, checkers can be implemented in serial fashion using linear feedback shift registers or in parallel fashion using only combinational logic.

In serial implementation shown in Fig. 5 flip flops are initialized by zero and after n clock cycles, the flip flops contain the remainder of the division operation. The number of clock cycles can be reduced to m, if the flip flops are initialized by the first $n-m$ higher degree bits of the codeword. Furthermore, the decoded data is ready after division. For parallel implementation, a parallel divider can be used.

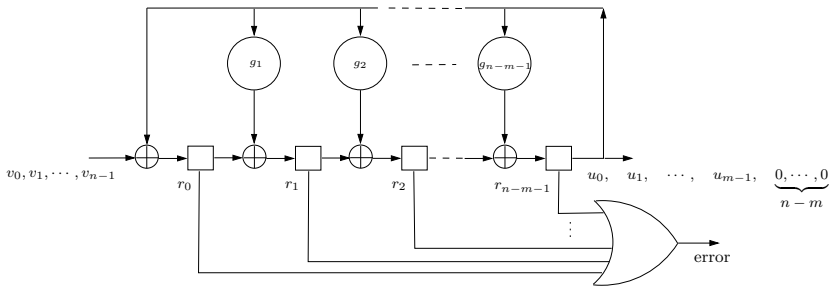

Fig. 5. Serial checker of an $\mathcal{L}$ code

IV. Error Detection Capability

In this section, our error model and the probability of an undetected error of the proposed scheme are given. The frequency of the check points is also discussed.

A. Error Modelling

The error model in this work is a bit-flip model. To illustrate the model, suppose that the error free value of a location, say L, of a polynomial basis multiplier is an n-tuple, say $v = (v_0, v_1, \cdots, v_{n-1})$. An error vector is also an n-tuple, say $e = (e_0, e_1, \cdots, e_{n-1})$. The number of possible errors is $2^n - 1$. Suppose the erroneous value of the location L is v_e, then we have:

$$v_e = v + e,$$

where '+' is bitwise XOR. In other words, an error is a module-2 additive term at a certain location of a PB multiplier and the i^{th} bit of the error vector e being one implies that the i^{th} bit of the value of the location L has changed from 0 to 1 or vise versa. If the location is one of the modules (SR, SM or VA), without loss of generality we can assume that the error vector should be XORed with the output of the component.

Note that the encoders and checkers should be fault free or at least self checking [9]. Since in practice the number of redundant bits, $n - m$, is expected to be much less than the size of the input operands of the multiplier, m, the self checking technique is feasible. In this work, for simplicity we assume that these encoders and decoders are error free. It will be shown in Section V that with a moderate number of redundant bits the probability of error detection becomes quite close to unity. As an example, for $m = 163$, with 8 redundant bits, the error detection probability

is approximately 0.996. In the following, we investigate what kind of errors cannot be detected by this scheme.

B. Probability of an Undetected Error

For the purpose of error detection, a received n-tuple should be checked if it is still a codeword or not. Therefore, based on our error model, any nonzero error that is a multiple of the generator polynomial $g(x)$ cannot be detected. In other words, any nonzero error vector from the set of all codewords is an undetectable error.

Let the probability of error detection and the probability of an undetected error be referred to as Pr_D and Pr_U, respectively. Clearly, $Pr_D = 1 - Pr_U$. Suppose W_i is the number of codewords of weight i in an (n, m) $\mathcal{L}$ code, i.e., W_i is the number of codewords that contain i ones. The probability of an undetected error can be computed using such weight distribution of the code. As mentioned, an undetected error occurs when the error vector is among one of the nonzero codewords. Thus,

$$Pr_U = \sum_{i=1}^{n} W_i p^i (1-p)^{n-i},$$

where p is the probability of a bit of error vector being one.

The weight distribution is known for some special codes such as Hamming codes and Reed-Solomon codes; however; the distribution is not known for the one we use in this work. Hence, a closed form for Pr_U cannot be obtained and the probability of an undetected error is investigated by simulation. Fig. 6 shows the result of our simulation for a $(171, 163)$ $\mathcal{L}$ code with generator polynomial $g(x) = x^8 + x^4 + x^3 + x^2 + 1$.

In the figure, plus symbols ('+') are for the probability of an undetected error for different values of p. A well-known upper bound for the probability of an undetected error for some (n, m) codes such as Hamming code is $2^{-(n-m)}$. Here, the number of redundant bits is 8. The solid line in the figure shows the value 2^{-8}. According to the figure, the values of Pr_U are either smaller than or approximately same as the bound value.

C. Frequency of 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 discussed in previous section, each of the above

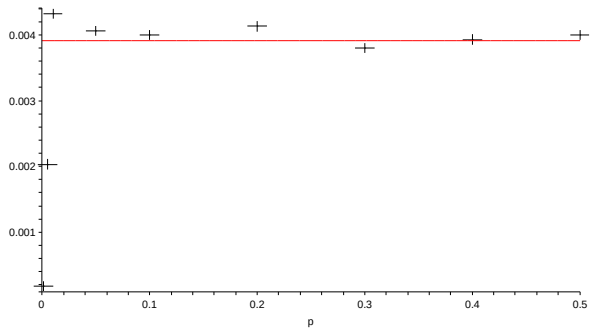

Fig. 6. Probability of an undetected error vs. p

mentioned locations in Section III-C should have a parity checker. This requires a very high area overhead especially for bit-parallel multipliers. The following lemma helps us reduce the number of checkers considerably.

Lemma 1: [1] 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. 3). Then any such error can be detected with the probability Pr_D, discussed in Section IV-B, using a parity checker at L3 of the bit-serial multiplier or a parity checker before the vertical input of every VA and one parity checker after the final VA in the bit-parallel multiplier.

V. Analysis of Time and Area Overheads

In this section, area and time overheads of the error detection scheme are investigated. The circuit of a complete bit-serial multiplier with concurrent error detection capability is shown in Fig. 7. Register D is initialized with the encoded input A'.

We implemented the scheme for $m = 163$ using the NIST recommended field defining polynomials for ECDSA $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$. Furthermore, a $(171, 163)$ $\mathcal{L}$ code was used. We described the scheme by VHDL to obtain a realistic approximation of the area overhead. We used Modelsim to simulate the design for checking its correct functionality and we implemented the scheme on a Xilinx Spartan 3 (XC3S5000) FPGA using Xilinx ISE 7.1i.

As mentioned in Section III-D, the encoder and checker can be implemented in serial or in parallel fashions. Therefore, three different implementations for bit-serial polynomial basis

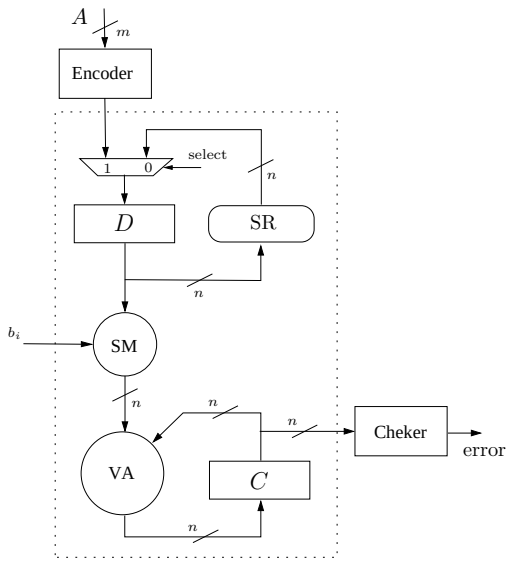

Fig. 7. A complete bit-serial multiplier with CED

multipliers are considered based on the implementations of the encoder and checker: 1) serial-encoder serial-checker (SE-SC), 2) serial-encoder parallel-checker (SE-PC), and 3) parallel-encoder parallel-checker (PE-PC). The time and the area overheads of each implementation are presented in Table V.

	Bit-serial implementation		
Overhead	SE-SC	SE-PC	PE-PC
Area (%)	32.7	26.7	23.2
Time (clock cycle)	mn	m	0

TABLE I

THE TIME AND THE AREA OVERHEADS FOR THE DIFFERENT IMPLEMENTATIONS OF A BIT-SERIAL PB MULTIPLIER

As shown in the table, the lowest area overhead is for the PE-PC implementation. The reason is that serial encoders or checkers contain registers and shift registers but parallel ones do not. Such registers and shift registers are relatively area consuming components in FPGAs. On the other hand, as discussed in Section III-D, a serial encoder needs m clock cycles to encode the data and a serial checker needs at least m clock cycles to check whether the input n-tuple is a

codeword or not and give the corresponding (not encoded) data. Clearly, parallel encoders and decoders need no extra clock cycles, neither does the PE-PC implementation. Therefore, our best implementation, on the XC3S5000 FPGA, in terms of area and time is the PE-PC.

VI. Double-Input Encoding (DIE)

Having only one input of the PB multiplier encoded has one major disadvantage. If the second input of the multiplier becomes erroneous, it cannot be detected. One way to improve this situation is to encode both input operands. In general, the generators for encoding inputs can be different. However, there are some issues with regard to choosing the generators and they are discussed in Section VI-B.

A. Polynomial Basis Multipliers with CED Capability

In the double-input encoding, input A is encoded by the generator $g_1(x)$ and B by $g_2(x)$. Let $C = A \cdot B \bmod f(x)$, where $f(x)$ is the field defining polynomial. Multiplying each side by $g_1(x)g_2(x)$, we obtain:

$$Cg_1g_2 = ABg_1g_2 \bmod fg_1g_2.$$

Hence,

$$E_{g_1g_2}(C) = E_{g_1}(A)E_{g_2}(B) \bmod \mathcal{F}(x),$$

where $\mathcal{F}(x) = f(x)g_1(x)g_2(x)$ and $E_g(Z)$ implies that Z is encoded by generator g. Let the degrees of $g_1(x)$ and $g_2(x)$ be r_1 and r_2, respectively. Clearly, the degree of $\mathcal{F}(x)$ is $\mathcal{N} = m + r_1 + r_2$.

An SR module can be constructed using (7) and by replacements of $F(x)$ and n with $\mathcal{F}(x)$ and $\mathcal{N}$, respectively. To construct a bit-serial multiplier and/or a bit-parallel multiplier with CED capability, we use updated versions of SR, SM, and VA modules in a very similar manner as shown in Fig. 3. Here, the number of rounds of the bit-serial multiplier and the number of rows of the bit-parallel multiplier are $m + r_2$ each.

B. Error Detection Capability

Like Section IV-A, here, the bit-flip error model is used. For the purpose of error detection, checkers that use the generator g_1 are placed in the same locations as discussed in Section III-C.

If there is no error in the circuit, then the output value of the last checker that uses the generator g_1 is $Cg_2 = ABg_2$. Therefore, one more checker that uses the generator g_2 should be placed at the output of the last checker. Then, the final result of the multiplication is the output of the checker that used the generator g_2. Assuming that only a maximum of one multiple-bit error occurs per round of a bit-serial multiplier or per row of a bit-parallel multiplier, we have:

- if an error occurs on input B and the error is a multiple of g_2, it cannot be detected.
- if errors occur on input A and/or inside the PB multiplier and they are not multiples of g_1, they are detected. If they are multiples of g_1 but the output of the last checker that used generator g_1 is not a multiple of g_2, the errors are detected as well. Otherwise, they are not detected.

Note that g_2 can be preferably chosen such that its degree is smaller than that of g_1. Polynomial g_2 is mainly used for detecting errors in input B although it affects the error detection of the entire multiplier circuit. Furthermore, this choice decreases the area overhead of the scheme.

VII. Conclusions

This article introduces a scheme for detection of multiple-bit errors in polynomial basis multipliers using linear codes. Based on our simulation, the probability of an undetected error is close to 2^{-r} where r is the number of redundant bits in the codewords. Furthermore, the overhead of our error detection scheme for bit-serial implementation is much lower than the overhead of the dual modular redundant scheme for a sufficient number of redundant bits. Additionally, for the parallel-encoder parallel-checker implementation of the scheme, there is no extra clock cycle in the actual multiplication operation.

References

[1] S. Bayat-Sarmadi and M. A. Hasan. On concurrent detection of errors in polynomial basis multiplication. CACR Technical Report archive, Report 2006-12, 2006. http://www.cacr.math.uwaterloo.ca/techreports/2006/cacr2006-12.pdf.

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

[3] 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.

[4] D. Boneh, R. Demillo, and R. Lipton. On the improtance of checking cryptographic protocols for faults. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, pages 37–51. Springer-Verlag, 1997.

[5] M. Ciet and M. Joye. Elliptic curve cryptosystems in the presence of permanent and transient faults. *Designs, Codes and Cryptography*, 36(1):33–43, July 2005.

[6] 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*, 13:29–40, 1998.

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

[8] W. W. Peterson and E. J. Weldon. *Error Correcting Codes.* MIT Press, Cambridge, MA, 2nd edition, 1972.

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

[10] A. Reyhani-Masoleh and M. Hasan. Towards fault-tolerant cryptographic computations over finite fields. *ACM Transactions on Embedded Computing Systems*, 3(3):593–613, August 2004.

[11] 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)*, pages 515–528. Springer-Verlag, 2002.

[12] A. Reyhani-Masoleh and M. A. Hasan. Fault detection architectures for field multiplication using polynomial bases. *IEEE Transactions on Computers-Special Issue on Fault Diagnosis and Tolerance in Cryptography*, 55(9):1089–1103, September 2006.