On-line Error Detection in a Polynomial Basis Multiplier over $GF(2^m)$ Using Self-Checking Alternating Logic

Wen Tzeng Huang[1] Sun Yen Tan[2] Che Wun Chiou[3,*]

Chiu Ching Tuan[4] Chih Hsiang Chang[5]

[1] Department of Computer Science & Information Engineering, Minghsin University of Science & Technology
Hsinchu 30401, Taiwan
wthuang@must.edu.tw

[2] Department of Electronic Engineering, National Taipei University of Technology
Taipei 10608, Taiwan
sytan@ntut.edu.tw

[3] Department of Computer Science and Information Engineering, Chien Hsin University of Science and Technology
Taoyuan 320, Taiwan
cwchiou@uch.edu.tw

[4] Department of Electronic Engineering, National Taipei University of Technology
Taipei 10608, Taiwan
cctuan@ntut.edu.tw

[5] Institute of Computer and Communication, National Taipei University of Technology
Taipei 10608, Taiwan
garychang0706@gmail.com

Received 22 May 2013; Revised 5 July 2013 ; Accepted 25 July 2013

Abstract. Polynomial basis multipliers which are widely applied in public-key cryptosystems such as elliptic curve cryptosystem are suitable to be FPGA (Field Programmable Gate Array) and VLSI (Very Large Scale Integrated circuit) implementations due to their regularity and modularity properties. On-line error detection capability can provide a countermeasure to recently developed fault-based cryptanalysis. A novel polynomial basis multiplier using self-checking alternating logic (SCAL) method is developed. The proposed polynomial basis multiplier can provide both on-line error detection and off-line testability capabilities. Our proposed multiplier requires only one-third of the extra space complexity that existing multipliers require. Our proposed SCAL polynomial basis multiplier is the first polynomial basis multiplier over $GF(2^m)$ with both on-line error detection and off-line testability capabilities.

Keywords: Finite field multiplier, elliptic curve cryptosystem, concurrent error detection, fault-based cryptanalysis.

1 Introduction

Finite field arithmetic has been widely applied to coding theory [1], cryptography [2], and digital signal processing [3-4]. Finite field arithmetic includes addition, multiplication, division, and inversion operations. The multiplication operation is the most important arithmetic operation for application to cryptosystems such as the elliptic curve cryptosystem (ECC). ECC uses much smaller key bits than the RSA (Rivest-Shamir-Adleman) public-key cryptosystem to deliver an equivalent level of security. For example, an ECC with a 160-bit key approaches the same security level of RSA with a 1024-bit key. Moreover, less computation is needed with ECC to achieve the same level of security as RSA; therefore, ECC computes faster than RSA. These benefits make ECC more attractive than RSA for use as a cryptosystem on devices with limited resources, such as portable

* Corresponding author.

smart phones. Unfortunately, cryptosystems on smart phones are highly vulnerable to side-channel attacks, including fault-based attacks.

The applications of ECC are almost exclusively in digital systems; therefore, ECC strongly depends on finite field arithmetic operations, especially $GF(2^m)$. These $GF(2^m)$ operations include addition, multiplication, multiplicative inversion, and exponentiation operations. Addition in $GF(2^m)$ is easily performed using exclusive OR (XOR) gates. Multiplicative inversion and exponentiation are much more time-consuming than addition and multiplication but can be performed using iterative multiplications. Hence, efficient implementation of multiplication is fundamental in cryptographic applications.

The efficiency of finite field multiplication depends on the specification of the field element representation. Three major bases represent elements in $GF(2^m)$: a polynomial basis (PB) [5-10], dual basis (DB) [11-15], and normal basis (NB) [16-23]. Each basis has its own particular advantages. The PB architectures have the advantages of low circuit complexity, simplicity, regularity, and modularity. The DB multipliers have the lowest chip area among these basis multipliers. The NB multipliers can perform the squaring operation only by cyclically shifting its binary form.

Fault-based cryptanalysis is a new efficient method for attacking both private-key and public-key cryptosystems. Boneh *et al.* [24] proved that fault-based cryptanalysis is efficient against RSA cryptosystems. Kelsey *et al.* [25] used differential fault analysis to easily recover the key of symmetric data encryption standard (DES), and only required about 200 ciphers. Biham and Shamir [17] and Boneh *et al*. [24] also showed that fault-based cryptanalysis can effectively attack public-key cryptosystems. Therefore, the simplest method to avoid attacks from fault-based cryptanalysis is to check the correctness of the ciphers before output of the cipher. For this reason, the development of techniques to detect errors in cryptosystems is an important area of research.

In 1998, Fenn *et al.* [26] firstly proposed on-line error detection for bit-serial PB multipliers in $GF(2^m)$ using the parity prediction method. Reyhani-Masoleh and Hasan [27,28] provided error detection methods in bit-parallel and bit-serial polynomial basis multipliers in $GF(2^m)$ using parity checking. Bayat-Sarmadi and Hasan [29] used multiplication parity bits for concurrently detecting errors in PB multipliers; this method detected all of the odd parity errors and most of the even parity errors. However, parity checking is very time consuming. If an XOR tree is applied to compute parity bits, at least $\lceil \log_2 m \rceil$ XOR-gate delays are required. For modern cryptographic applications, the field size m can be very large, ranging from 160 to 2048 bits. For example, the time overhead in the method of Bayat-Sarmadi and Hasan [29] is approximately 20% or more, on average. Chiou [30] used the re-computing with shifted operands (RESO) method to provide a concurrent error detection method for polynomial-based multipliers using all-one polynomials. Lee *et al.* [31] extended these results to the PB multiplier generated by a general irreducible polynomial. Chiou *et al.* [32] then applied the same concept to concurrently detect errors in the PB Montgomery multiplier. Such time redundancy methods [30-32] require very little time overhead; for example, only about 1.3% time overhead is required for the method proposed by Chiou *et al.* [32].

Regular architecture is a very important property of very large scale integration (VLSI). In this study, we present a self-checking alternating logic (SCAL) bit-parallel PB multiplier using alternating inputs. Our SCAL approach can detect both permanent and transient faults and has a fault-secure property, in which any occurring fault in a fault model can be detected by at least one alternating input pair. In contrast, in existing error detection approaches, some occurring single faults are not excited and thus are not detected. Our SCAL PB multiplier retains a regular structure and requires only a small time overhead, similar to previous time redundancy methods [30-32].

The remainder of this article is organized as follows. Section 2 briefly reviews the polynomial basis and SCAL. Section 3 introduces the traditional bit-parallel polynomial basis multiplier. In Section 4, the novel bit-parallel polynomial basis multiplier with on-line error detection capability is presented. Comparison results are given in Section 5. A brief conclusion is finally made in Section 6.

2 Preliminaries

This section reviews basic concepts concerning PB multiplication over $GF(2^m)$ and SCAL. For more detail, readers can refer to [2] and [33,34,35], respectively.

2.1 Polynomial Basis

Let $P(x)$ be a degree m irreducible primitive polynomial over $GF(2)$ in Eq. (1)

$$P(x) = p_m x^m + p_{m-1} x^{m-1} + p_{m-2} x^{m-2} + \cdots + p_1 x^1 + p_0 = \sum_{i=0}^{m} p_i x^i \qquad (1)$$

where $p_i \in GF(2) = \{0,1\}$ and $p_0 = p_m = 1$. Then, let a set $\{1, x, x^2, \ldots, x^{m-2}, x^{m-1}\}$ be a polynomial basis of $GF(2^m)$ generated by $P(x)$ [2]. Thus, we can represent any element in $GF(2^m)$ defined by $p(x)$ as A, B, and C, which can be represented as follows, where $a_i, b_i, c_i \in \{0,1\}$, and $0 \le i \le m-1$.

$$A = a_0 + a_1 x + a_2 x^2 + \cdots + a_{m-2} x^{m-2} + a_{m-1} x^{m-1}$$
$$B = b_0 + b_1 x + b_2 x^2 + \cdots + b_{m-2} x^{m-2} + b_{m-1} x^{m-1}$$
$$C = c_0 + c_1 x + c_2 x^2 + \cdots + c_{m-2} x^{m-2} + c_{m-1} x^{m-1}$$

Moreover, let C be the product of A and B, which can be represented by Eq. (2).

$$\begin{aligned} & C(x) \\ & = A(x)B(x) \bmod P(x) \\ & = \left(a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots + a_{m-1} x^{m-1}\right) B(x) \bmod P(x) \\ & = \begin{pmatrix} a_0 x^0 B(x) \bmod P(x) + a_1 x^1 B(x) \bmod P(x) + a_2 x^2 B(x) \bmod P(x) + \ldots \\ + a_{m-1} x^{m-1} B(x) \bmod P(x) \end{pmatrix} \\ & = c_0 x^0 + c_1 x^1 + c_2 x^2 + \ldots + c_{m-1} x^{m-1}, \\ & \text{where } c_i \in \{0,1\},\ 0 \le i \le m-1. \end{aligned} \tag{2}$$

2.2 SCAL

A circuit whose output is encoded in an error-detecting code is called a self-checking circuit. Alternating logic design techniques achieve fault detection capability; this is one of the time redundancy techniques. Yamamoto *et al.* [33] pointed out that any combinational circuit can be made self-dual with only one extra input. The alternating logic technique can be easily extended to the multipliers and other arithmetic units, provided that the functional modules are self-dual. Reynolds and Metze [34] showed that an arbitrary function G with m variables could become a self-dual function G^* with $m+1$ variables by setting

(i) $G^*\left(Y, \overline{y_{m+1}}\right) = G(Y)$, and

(ii) $G^*\left(Y, y_{m+1}\right) = \overline{G(\overline{Y})}$.

When y_{m+1} is the logical value "0," G^* realizes the original function G. When y_{m+1} is "1," G^* performs the dual function $\overline{G}$ of G. Let Y^* represent a group of $m+1$ variables (Y, y_{m+1}). The function $F^*(Y^*)$ is an SCAL circuit if it satisfies the following condition when both alternating inputs Y^* and $\overline{Y^*}$ are applied: $\forall f, \exists Y^* \ni \overline{G_f^*(Y^*)} \ne G_f^*\left(\overline{Y^*}\right)$, where f represents a stuck-at fault in G^*, and G_f^* denotes the function F^* with an existing fault f. A design method has been reported that provides alternating logic with self-checking capability [34,35]. This proposed design can be extended to other arithmetic units, such as multipliers and dividers, using combinational circuits with hardware redundancy.

3 Bit-parallel PB Multiplier

This section will discuss the traditional bit-parallel PB multiplier. Based on the above properties, the polynomial basis multiplication of $GF(2^m)$ $C(x)=A(x)B(x)$ can be represented as follows:

$$\begin{aligned} & C(x) \\ & = A(x)B(x) \bmod P(x) \\ & = c_0 + c_1 x + c_2 x + \ldots + c_{m-1} x^{m-1}, \end{aligned}$$

where $c_i \in GF(2)$ for $0 \le i \le m-1$. Based on Horner's rule, $C(x)$ can then be computed as follows:

$$\begin{aligned} & C(x) = A(x)B(x) \\ & = \left(a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-1} x^{m-1}\right) B \\ & = a_0 B + a_1 x B + a_2 x^2 B + \ldots + a_{m-1} x^{m-1} B \end{aligned} \tag{3}$$

Because the finite field operation needs the mod $P(x)$ operation, it can be simplified using the following equation, Eq. (4).

$$x^m = p_0 + p_1 x + p_2 x^2 + \ldots + p_{m-1} x^{m-1} \quad (4)$$

Let $xB \bmod P$ be as shown in Eq. (3).

$$\begin{aligned} xB &= b_0 x + b_1 x^2 + b_2 x^3 + \cdots + b_{m-2} x^{m-1} + b_{m-1} x^m \\ &= b_0 x + b_1 x^2 + b_2 x^3 + \cdots + b_{m-2} x^{m-1} + b_{m-1}(p_0 + p_1 x + p_2 x^2 + \cdots + p_{m-2} x^{m-2} + p_{m-1} x^{m-1}) \\ &= b_{m-1} p_0 + (b_{m-1} p_1 + b_0) x + (b_{m-1} p_2 + b_1) x^2 + \cdots + (b_{m-1} p_{m-1} + b_{m-2}) x^{m-1}. \end{aligned} \quad (5)$$

Hence, let $B^i = x^i B = \beta_0^i + \beta_1^i x + \beta_2^i x^2 + \cdots + \beta_{m-1}^i x^{m-1}$ and its binary representation be expressed as follows, where $\beta_j^i \in \{0,1\}$ for $0 \le j \le m-1$. Based on Eq. (5), B^i can be directly obtained from the previously computed result of B^{i-1}, as shown in Eq. (6).

$$\begin{aligned} & B^i \\ &= xB^{i-1} \\ &= \beta_{m-1}^{i-1} p_0 + \left(\beta_{m-1}^{i-1} p_1 + \beta_0^{i-1}\right) x + \left(\beta_{m-1}^{i-1} p_2 + \beta_1^{i-1}\right) x^2 + \ldots + \left(\beta_{m-1}^{i-1} p_{m-1} + \beta_{m-2}^{i-1}\right) x^{m-1}. \end{aligned} \quad (6)$$

Thus, the relationship between the coefficients of B^i and B^{i-1} can be summed as follows:

$$\beta_j^i = \begin{cases} \beta_{m-1}^{i-1} p_0 & \text{if } j = 0 \\ \beta_{m-1}^{i-1} p_j + \beta_{j-1}^{i-1} & \text{if } 1 \le j \le m-1 \end{cases}$$

Thus, C can be represented as shown in Eq. (7).

$$\begin{aligned} & C \\ &= A \times B \bmod P \\ &= a_0 B^0 + a_1 B^1 + a_2 B^2 + \ldots + a_{m-1} B^{m-1} \\ &= c_0 x^0 + c_1 x^1 + c_2 x^2 + \ldots + c_{m-1} x^{m-1}, \\ & \text{where } c_j \in \{0,1\},\ 0 \le j \le m-1, \text{ and } c_j = \sum_{i=0}^{m-1} a_i \beta_j^i. \end{aligned} \quad (7)$$

Based on Eq. (4), the traditional bit-parallel multiplier is as shown in Figure 1.

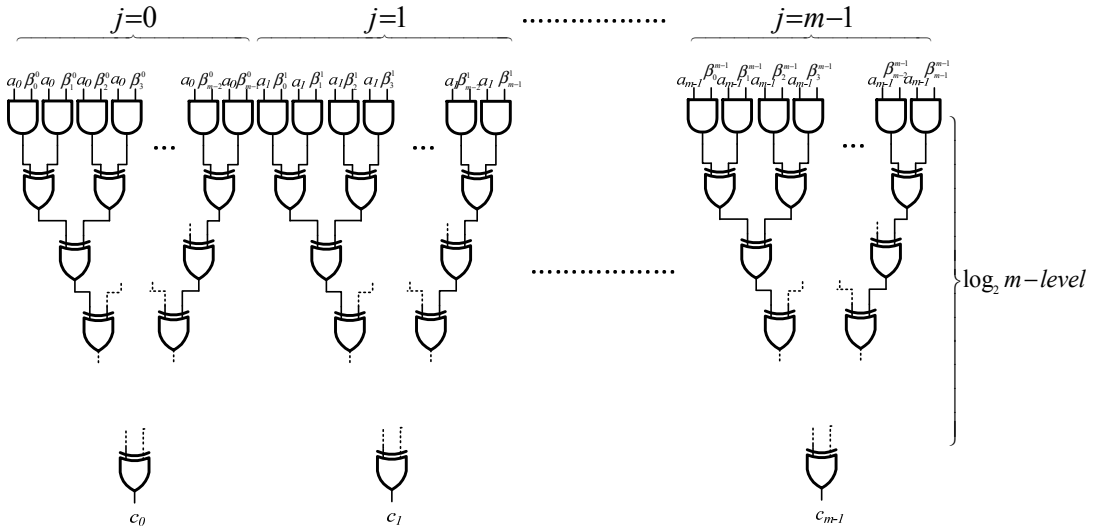


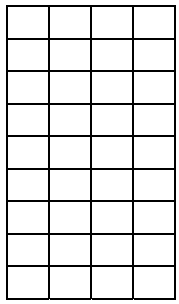
Fig. 1. The traditional bit-parallel multiplier.

4 The proposed bit-parallel self-checking PB multiplier

In this section, the proposed bit-parallel self-checking PB multiplier will be developed. The principle of the self-checking alternating logic [33] can be modified and used as the bit-parallel multiplier with the capabilities of on-line error detection and off-line testing. Therefore, if the bit-parallel self-checking PB multiplier is used to design ECC, it can avoid the attack of the fault-based cryptanalysis such that this design can guarantee the data security under the transmission process.

A bit-parallel PB multiplier with self-checking capability is easily derived from the proposed PB multiplier. A single stuck-at fault model is assumed in this study. We have dealt with the problem of concurrent error detection in the bit-parallel PB multiplier by using an SCAL design, which includes self-dual AND gate and 3-input XOR gate. Because a 3-input XOR gate is a self-dual gate, only a self-dual AND circuit needs to be designed. Table 1 shows the truth table of the self-dual AND circuit, whose equation is $z = tb + ab + ta = \overline{(\overline{tb})(\overline{ab})(\overline{ta})}$.

Table 1. Truth table of self-dual AND circuit.



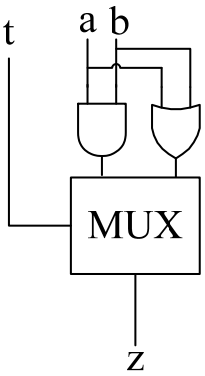


Fig. 2. Detailed circuit *K* with self-dual AND.

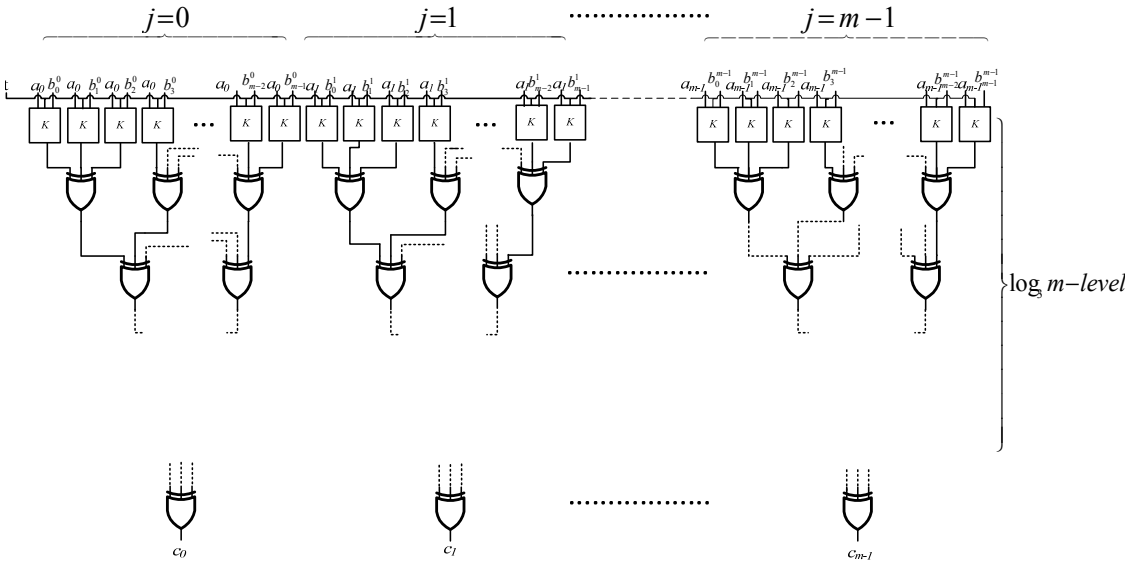


Fig. 3. The SCAL PB multiplier.

The truth table of the self-dual AND circuit is shown as Table 1. The self-dual characteristic can be find out in the AND truth table. When the output z is set 1 as the input "011" and the output z is set 0 as the input "100", these characteristics can be made the bases under the off-line testing. The results of the twice operations should be the mutual exclusion, when "011" is as the first primary input and "100" is as the second input. It represents that there is a fault of the AND gate if the output results are the same. Since the Figure 3 structure uses our proposed cell K of Figure 2, the capability of Figure 3 owns the SCAL function such that it can make the functions of the off-line testing and concurrent error detection. When the first clock inputs one normal value and then the second clock inputs its backhand value during operations, there is a fault if the both outputs are the same. On the contrary, it is under a normal operation as the both outputs to be opposite. Hence, our system can check whether there is a fault operation in the circuit.

Two steps must be executed in sequential order: the original multiplication function, $C=A\times B$, and the complemented multiplication function. Then the results of the two steps are compared. A mismatch indicates an existing error. The detailed execution status is depicted in Figure 3. The self-checking algorithm of the proposed PB multiplier is described as follows:

Algorithm-MulSCAL:

/* To execute and compare both $C = A \times B$ and $\overline{C} = \overline{A}\times\overline{B}$ */

Step 1: Perform $C = A \times B$;

Step 2: Perform $\overline{C} = \overline{A}\times\overline{B}$;

Step 3: Compare the result C in Step 1 with the complemented value calculated in Step 2. An error = 1 is signaled if a mismatch occurs.

Lemma 1: The PB multiplier in Figure 3, containing a fault *f*, produces an erroneous alternating output for the alternating input $(A; \overline{A})$ and $(B; \overline{B})$ iff both $(A; \overline{A})$ and $(B; \overline{B})$ sensitize the fault.

Proof: The circuit sensitizes the fault iff $\overline{C(X)}$ is produced instead of $C(X)$; $\overline{X}$ sensitizes the fault iff $\overline{C(\overline{X})} = C(X)$ is produced instead of $C(\overline{X}) = \overline{C(X)}$. An erroneous alternating output results iff $\overline{C(X)}$ is produced instead of $C(X)$ and $C(X)$ instead of $\overline{C(X)}$, that is, iff both $(A; \overline{A})$ and $(B; \overline{B})$ sensitize the fault.

In the terminology of totally self-checking networks, the alternating output $(C; \overline{C})$ is the correct code-space output, $(C;C)$ and $(\overline{C}; \overline{C})$ are non-code-space outputs that are detectable, and $(\overline{C}; C)$ is an incorrect code-space output that is not detectable and must be avoided.

An alternating network is called a self-checking alternating network if for every fault from a prescribed set $\mathcal{F}$ there exists at least one input $(A; \overline{A})$ and $(B; \overline{B})$ that produces a non-alternating, detectable output (the self-testing condition), and there does not exist any input $(A; \overline{A})$ and $(B; \overline{B})$ that produces an erroneous alternating, non-detectable output (the fault-secure condition). In contrast to totally self-checking networks in which the self-test condition must be met using code-space inputs only, self-checking alternating networks allow all possible inputs of the form $(A; \overline{A})$ and $(B; \overline{B})$. Therefore, meeting the self-test condition is much easier; in fact, it is equivalent to showing that the network is irredundant. In this paper, the necessary and sufficient conditions for an alternating network to be self-checking for a fault set consisting of single stuck-at fault are derived. It is assumed that the networks considered are, or can be made to be, single-line irredundant and hence self-testing; hence, a proof that the network is self-checking then reduces to a proof that it is fault-secure. The results obtained for single stuck-line-faults can sometimes be extended to certain multiple faults (e.g., unidirectional faults), but this issue is not pursued here.

Theorem 1: The PB multiplier shown in Figure 3 is fault-secure for single faults on primary input lines, primary input fan-out branch lines, and output lines of primary inverters.

Proof: Consider a single stuck-at *d* fault on a primary input line a_i or b_i. If A or B sensitizes the fault, a_i or $b_i = \overline{d}$. But then $\overline{a_i}$ or $\overline{b_i} = d$, and $\overline{A}$ or $\overline{B}$ cannot sensitize the fault. Conversely, if $\overline{A}$ or $\overline{B}$ sensitizes the fault, then a_i or $b_i = \overline{d}$, and A or B cannot sensitize the fault. So, by Lemma 1, the PB multiplier in Figure 3 is fault-secure for single faults on primary input lines. The extension to primary input fanout branch lines and to output lines of primary inverters is immediate.

Theorem 2: The PB multiplier in Figure 3 is fault-secure for all single faults if it is internal fanout-free.

Proof: Consider a single stuck-at *d* fault on an internal line *l*. (Non-internal lines are covered by Theorem 1.) By way of contradiction, assume that the circuit is not fault-secure for that fault. Then there exists an input *A* or *B* for which both *X* and $\overline{X}$ sensitize the fault and produces an erroneous alternating output. The inversion parity of a path *P* is defined as the parity of the number of inverting gates encountered in tracing the path *P* and is denoted by $\pi(P)$.

Let P_x and $P_{\bar{x}}$ denote the paths sensitized from line *l* to the output under $(A; \overline{A})$ and $(B; \overline{B})$, respectively. Then,

$$y(X) = d \oplus \pi(P_x) \text{ and } y(\overline{X}) = d \oplus \pi(P_{\bar{x}}).$$

But since the PB multiplier in Figure 3 is free of any internal fanout, then there is only one path from *l* to the network output; that is, $P_x = P_{\bar{x}}$. Hence, $\pi(P_x) = \pi(P_{\bar{x}})$; therefore, $y(X) = y(\overline{X}) = C$, say, and the non-alternating output $(C;C)$ is produced under $(A; \overline{A})$ and $(B; \overline{B})$, a contradiction.

Theorem 3: The PB multiplier in Figure 3 is fault-secure for all single faults if it is essentially inverter-free.

Proof: The proof is the same as the proof for Theorem 2, except that here $\pi(P_x)=\pi(P_{\bar{x}})$ because there are no inverters internal to the network.

Theorem 4: The PB multiplier in Figure 3 is fault-secure for all single faults if it is fault-secure for single faults on fanout gate outputs.

Proof: Consider a stuck-line fault on an internal line *l* that is not a fanout gate output. (Non-internal lines are covered by Theorem 1.) There is a unique fanout-free path segment from *l* either to the network output or to a fanout gate. For the former case, the network is fault-secure by Theorem 2. For the latter case, again by Theorem 2, a sensitized stuck-at fault on *l* can only produce a non-alternating signal, say $(C;C)$, at the fanout gate output. But this is equivalent to a stuck-at *d* fault on the fanout gate output for which the network is fault-secure by hypothesis.

Theorem 5: The PB multiplier in Figure 3 is fault-secure for all single faults if, for any fault on a fanout gate output that is sensitized to the network output, all sensitized paths to the network output have the same inversion parity.

Proof: Consider a fault on the output of a fanout gate. If an erroneous output is to be produced as a result of the fault, by Lemma 1 both *X* and $\overline{X}$ must sensitize the fault. However, by hypothesis, for any such $(A; \overline{A})$ and $(B; \overline{B})$ the inversion parities of all sensitized paths to the network output are equal, and, as before, a detectable non-alternating output is produced. The circuit must therefore be fault-secure for any fault on a fanout gate output. By Theorem 4, it is then fault-secure for all single faults.

The significance of Theorems 4 and 5 lies in the fact that to examine the fault-secureness of a functional realization that does not fit the classification of Theorems 2 and 3 (i.e., one that is neither free of internal fanouts nor essentially inverter-free), it is only necessary to simulate the network behavior under faults on fanout gate outputs and to compare it with the fault-free behavior to determine if an erroneous alternating output is produced. Because such a simulation can be done quickly, an optimal network design program, such as Davidson's branch and bound program, which produces irredundant and hence self-testing networks, could be modified to produce an optimal self-checking realization from a dualized functional specification, without significantly degrading the program performance. For some network structures not of the type considered in this section, the fault-secure property can be deduced by inspection, and, therefore, a simulation of the network behavior is not required. Corollary 1 specifies the conditions.

Corollary 1: The PB multiplier in Figure 3 is fault-secure for all single faults if, for every fanout gate, all paths to the network output have the same inversion parity.

The circuit is a multi-output network, and then the conditions for self-checking can be relaxed, because a multi-output network is fault-secure for all single faults, provided that for every input for which an erroneous alternating output is produced on one output line, a non-alternating output is produced on at least one other out-

put line. Therefore, multi-output realizations in which logic is shared between outputs should be simpler than those in which each output is produced with independent logic.

Any single stuck-at fault occurring in the proposed SCAL PB multiplier is detectable by our proposed algorithm. All unidirectional faults are also detectable by the algorithm because unidirectional faults will cause, on output, either a normal logical value 0 and faulty logical value 1, or a normal logical value 1 and faulty logical value 0, but not both, for all inputs. Thus, SCAL circuits can detect all faults that are unidirectional in nature.

5 Comparison

In this section, the proposed bit-parallel PB multiplier is compared with other existing similar multipliers. The case of general irreducible polynomial for *P(x)* is firstly compared, and then the cases of trinomial and pentanomial are compared in order. Table 2 lists the space complexities of bit-parallel PB multipliers with and without self-checking capability. The proposed SCAL PB multiplier requires about 33% of the space overhead that is required by the original PB multiplier. Compared to the PB multipliers with error detection capability proposed by Bayat-Sarmadi and Hasan [36], our proposed SCAL PB multiplier requires approximately 8.5% less space overhead. Most importantly, our proposed SCAL PB multiplier retains regular structure.

Table 2 compares the results of the proposed PB multiplier with those of a normal bit parallel PB multiplier and that of Bayat-Sarmadi and Hasan [36]. CMOS VLSI technology [37] is used to evaluate the space complexity.

Table 2. Space complexities of bit-parallel PB multipliers.

	Original bit-parallel PB multiplier (Fig. 1)	Bayat-Sarmadi and Hasan [36]	The proposed SCAL PB multiplier (Fig.3)
Error detection	No	Yes	Yes
Off-line testing	No	No	Yes
2 input AND gate	m^2	m^2	m^2
2 input OR gate	0	0	m^2
2 to 1 Multiplex	0	0	m^2
2 input XOR gate	m^2-m	m^2+4m+12	0
3 input XOR gate	0	0	$\le(\lceil m/2 \rceil)m$
Latch	0	$2m^2+m$	0
Transistor counts	$6m^2+12(m^2-m)$	$26\ m^2+56m+48$	$18\ m^2+12(\lceil m/2 \rceil)m$
Time complexity	$T_A + (2+\lceil \log_2(m-1) \rceil)T_x$	$T_A + (5+\lceil \log_2(m^2-m) \rceil)T_x$	$T_A + (3+\lceil \log_3 m \rceil)T_{3x} + T_{mux}$

Note: T_A = propagation delay of 2-input AND gate, T_x = propagation delay of 2-input XOR gate, T_{3x} = propagation delay of 3-input XOR gate, T_{mux} = propagation delay of 2-to-1 multiplexer.

(A) Case of trinomial

A design example of the proposed PB multiplier generated by trinomials is discussed here. An irreducible polynomial consisting of three non-zero terms, such as $P(x) = x^m + x^n + 1 (m > n > 0)$ is called a trinomial of degree *m*. If the proposed PB multiplier is generated by such an irreducible trinomial $P(x) = x^m + x^n + 1$, Eq. (4) can be rewritten as shown as Eq. (8).

$$\begin{aligned} & B^i \\ & = xB^{i-1} \\ & = b_{m-1}^{i-1} + b_0^{i-1}x + \ldots + b_{n-2}^{i-1}x^{n-1} + \left(b_{m-1}^{i-1} + b_{n-1}^{i-1}\right)x^n + b_n^{i-1}x^{n+1} + \ldots + b_{m-2}^{i-1}x^{m-1}. \end{aligned} \tag{8}$$

Based on equations (1), (4), and (8), the output result C of the proposed PB multiplier of $GF(2^4)$ generated by $P(x) = x^4+x^3+1$ can be obtained by Eq. (9).

$$\begin{aligned} c_0 &= a_0b_0+a_1b_3+a_2(b_2+b_3)+a_3(b_1+b_2+b_3) \\ c_1 &= a_0b_1+a_1b_0+a_2b_2+a_3(b_3+b_2) \\ c_2 &= a_0b_2+a_1b_1+a_2b_0+a_3b_3 \\ c_3 &= a_0b_3+a_1(b_2+b_3)+a_2(b_1+b_2+b_3)+a_3(b_0+b_1+b_2+b_3) \end{aligned} \tag{9}$$

The proposed SCAL bit-parallel PB multiplier with $P = x^4+x^3+1$ is shown in Figure 3. Two steps must be executed in sequential order: 1) the original multiplication function: C=A×B, and 2) the complemented multiplication function, $\overline{C} = \overline{A} \times \overline{B}$. The results of both steps are then compared, and a mismatch indicates an existing error. The detailed execution status is depicted in Figure 4. The self-checking algorithm for the proposed PB multiplier is described as follows:

Algorithm-MulSCAL:

/* To execute and compare both C=A×B and $\overline{C} = \overline{A} \times \overline{B}$ */

Step 1: Perform C=A×B.

Step 2: Perform $\overline{C} = \overline{A} \times \overline{B}$.

Step 3. Compare the result C of Step 1 with the complemented value calculated in Step 2. An error = 1 is signaled if a mismatch occurs.

Based on *Theorems* 1 through 5, Algorithm-MulSCAL can detect any single stuck-at fault occurring in any cell *K* of the proposed SCAL PB multiplier shown in Figure 4.

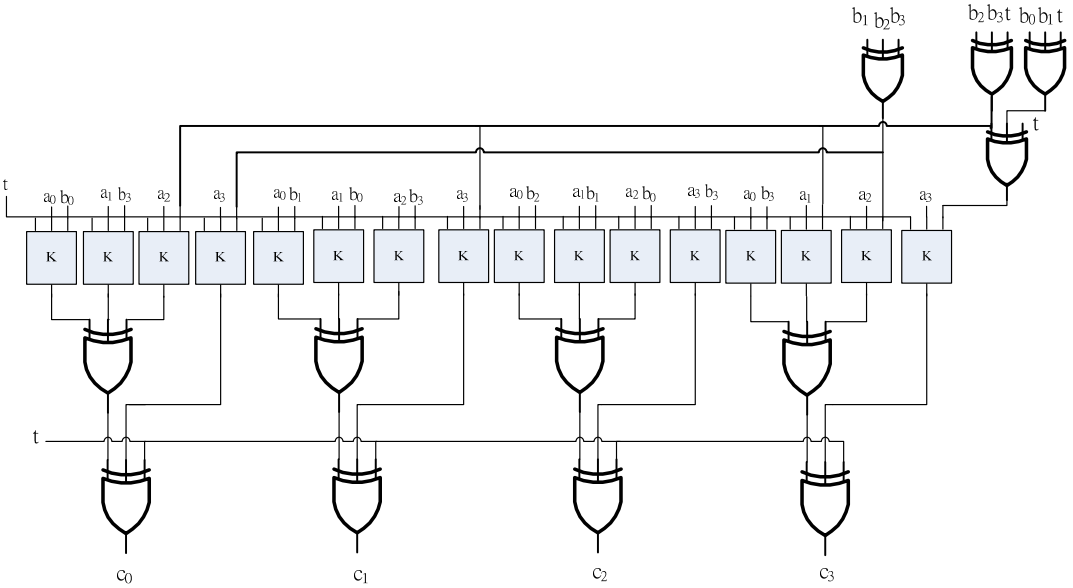


Fig. 4. An example of the proposed SCAL bit-parallel PB multiplier with $m = 4$.

Table 3 lists the space complexities of bit-parallel PB multipliers using irreducible trinomials with and without self-checking capability. As compared to Bayat-Sarmadi and Hasan's multiplier [36] for m=233 of NIST (National Institute of Standards and Technology) suggested values, the proposed SCAL PB multiplier saves about 8 % space complexity.

Table 3. Space complexities of bit-parallel PB multipliers using irreducible trinomials.

	The traditional bit-parallel PB multiplier using trinomial	Bayat-Sarmadi and Hasan [36]	The proposed SCAL PB multiplier using trinomial (Fig. 4)
On-line error detection	No	Yes	Yes
Off-line error detection	No	No	Yes
2-input AND gate	m^2	m^2	m^2
2-input OR gate	0	0	m^2
2-to-1 Multiplexer	0	0	m^2
2-input XOR gate	m^2	$m^2+4m+12$	0
3-input XOR gate	0	0	$\leq(\lceil m/2 \rceil)m$
Latch	0	$2m^2+m$	0
Transistor count	$6m^2+12(m^2)$	$26\ m^2+56m+48$	$18\ m^2+12(\lceil m/2 \rceil)m$

m=233	971610	1424610	1304334
Time complexity	$T_A +$ $(2+\lceil \log_2(m-1) \rceil)T_x$	$T_A +$ $(5+\lceil \log_2(m^2-m) \rceil)T_x$	$T_A +$ $(3+\lceil \log_3 m \rceil)T_{3x} + T_{mux}$

(B) Case of pentanomial

Pentanomials are also important because there are many values of m for which pentanomials exist. Let $P(x) = x^m + x^{n3} + x^{n2} + x^{n1} + 1$ be an irreducible pentanomial over $GF(2^m)$, where $m>n3>n2>n1>1$. The irreducible pentanomial is one kind of low-weight irreducible polynomial over $GF(2^m)$. The design example of the proposed bit-parallel PB multiplier generated by a pentanomial is discussed here. As the generated pentanomial $P(x) = x^m + x^{n3} + x^{n2} + x^{n1} + 1$ is used, Eq. (4) can be reformed and is shown as Eq. (10).

$$\begin{aligned} B^i = {} & b_{m-1}^{i-1} + b_0^{i-1}x + \ldots + b_{n1-2}^{i-1}x^{n1-1} + \left(b_{m-1}^{i-1} + b_{n-1}^{i-1}\right)x^{n1} + \\ & b_{n1}^{i-1}x^{n1+1} + \ldots + b_{n2-2}^{i-1}x^{n2-1} + \left(b_{m-1}^{i-1} + b_{n2-1}^{i-1}\right)x^{n2} + \\ & b_{n2}^{i-1}x^{n2+1} + \ldots + b_{n3-2}^{i-1}x^{n3-1} + \left(b_{m-1}^{i-1} + b_{n3-1}^{i-1}\right)x^{n3} + b_{n3}^{i-1}x^{n3+1} + \ldots + b_{m-2}^{i-1}x^{m-1}. \end{aligned} \quad (10)$$

Based on Eqs. (3) and (10), the output result C of the proposed PB multiplier of $GF(2^7)$ to be generated by $P(x) = x^7+x^5+x^3+x+1$ can be obtained and is shown as Eq. (11).

$$\begin{aligned} c_0 &= a_0b_0+a_1b_6+a_2b_5+a_3(b_4+b_6)+a_4(b_3+b_5)+a_5(b_2+b_4)+a_6(b_1+b_3), \\ c_1 &= a_0b_1+a_1(b_0+b_6)+a_2(b_5+b_6)+a_3(b_4+b_5+b_6)+a_4(b_2+b_3+b_4+b_5)+a_6(b_2+b_3+b_4+b_5), \\ c_2 &= a_0b_2+a_1b_1+a_2(b_0+b_6)+a_3(b_5+b_6)+a_4(b_4+b_5+b_6)+a_5(b_2+b_3+b_4+b_5)+ \\ &\quad a_6(b_2+b_3+b_4+b_5), \\ c_3 &= a_0b_3+a_1(b_2+b_6)+a_2(b_1+b_5)+a_3(b_0+b_4)+a_4(b_3+b_6)+a_5(b_2+b_5+b_6)+ \\ &\quad a_6(b_1+b_2+b_4+b_5), \\ c_4 &= a_0b_4+a_1b_3+a_2(b_2+b_6)+a_3(b_2+b_5)+a_4(b_4+b_6)+a_5(b_3+b_6)+a_6(b_2+b_5+b_6), \\ c_5 &= a_0b_5+a_1(b_4+b_6)+a_2(b_3+b_5)+a_3(b_2+b_4)+a_4(b_1+b_3)+a_5(b_2+b_6)+a_6(b_1+b_6), \\ c_6 &= a_0b_6+a_1b_5+a_2(b_4+b_6)+a_3(b_3+b_5)+a_4(b_2+b_4)+a_5(b_1+b_3)+a_6(b_2+b_6). \end{aligned} \quad (11)$$

The design structure of our proposed SCAL multiplier can be applied to the example of the pentanomial $P(x) = x^7+x^5+x^3+x+1$. $P(x) = x^7+x^5+x^3+x+1$ includes 72 multiplication operations. It requires calculation of the items $b_4+b_6, b_3+b_5, b_2+b_4$, $b_0+b_6, b_0+b_6, b_4+b_5+b_6, b_2+b_3+b_4+b_5$, b_0+b_6, b_5+b_6, b_2+b_6, b_1+b_5, b_0+b_4, $b_2+b_5+b_6$, $b_1+b_2+b_4+b_5$, b_2+b_5, ahead of the addition operation. Before entering the parallel addition, the sum can reduce the complexity of the calculations because each case of a pentanomial will require this operation. Moreover, to obtain the final result, our proposed structure needs $m^2 K$ cells to implement the multiplication and additions to get the summation of each column of the matrix. We use the example of $P(x) = x^7+x^5+x^3+x+1$ to illustrate this. It requires 60 addition operations.

Table 4 lists the space complexities of bit-parallel PB multipliers using irreducible pentanomial with and without self-checking capability. The proposed SCAL PB multiplier using irreducible pentanomial requires approximately 33% of the space overhead that is required by the original PB multiplier [38]. But, the proposed multiplier provides the design-for-testing capability.

Table 4. Space complexities of bit-parallel PB multipliers using irreducible pentanomials.

	The bit-parallel PB multiplier using pentanomials [38]	The proposed SCAL PB multiplier using pentanomials
On-line error detection	No	Yes
Off-line error detection	No	Yes
2-input AND gate	m^2	m^2
2-input OR gate	0	m^2
2-to-1 Multiplexer	0	m^2
2-input XOR gate	m^2+m-1	0
3-input XOR gate	0	$\leq(\lceil m/2 \rceil)m+3m$
Latch	0	0
Transistor count	$18m^2+12m-12$	$18\ m^2+12(\lceil m/2 \rceil)m+3m$
m=163	480186	643035

Time complexity	$T_A + (3 + \lceil \log_2 (m-1) \rceil) T_x$	$T_A + (3 + \lceil \log_3 m \rceil) T_{3x} + T_{mux}$

5 Conclusions

We have presented a novel bit-parallel PB multiplier over $GF(2^m)$ with on-line error detection using the alternating logic approach. The bit-parallel PB multiplier with on-line error detection and off-line testing capability is firstly proposed in this study. It modifies the logic gate structure to have self-dual and self-checking properties, and no extra gate is required. Therefore, regular structure is retained for the proposed PB multiplier and our approach is suitable for VLSI and FPGA implementation. Although the proposed SCAL PB multiplier still requires 33% of the space overhead while comparing with existing PB multipliers with on-line error detection, the proposed PB multiplier has fault-secure property. We describe that the traditional bit-parallel PB multiplier how to be modified to have the self dual characteristic. Then self-checking capability is included in the proposed PB multiplier. Therefore, our design is not only to meet the ability of concurrent error detection in operation but also make the off-line testing easy. Thus, the proposed design with alternating logic strategy can be easily extended to other arithmetic units, such as multiplicative inverter and dividers.

Acknowledgment

The authors would like to thank anonymous referees and the editor for their carefully reading the paper and for their great help in improving the paper. The authors also like to thank the National Science Council of the Republic of China, for partially financial supports to this research under Contract No. NSC 101-2221-E-231-024.

References

[1] F.J. MacWilliams, N.J.A. Sloane, *The theory of error-correcting codes*, North-Holland, Amsterdam, 1977.

[2] R. Lidl, H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, New York, 1994.

[3] R.E. Blahut, *Fast algorithms for digital signal processing*, Addison-Wesley, Reading, MA, 1985.

[4] I.S. Reed, T.K. Truong, "The use of finite fields to compute convolutions," *IEEE Trans. Inf. Theory*, Vol. IT-21, No.2, pp. 208–213, 1975.

[5] T.C. Bartee, D.J. Schneider, "Computation with Finite Fields," *Inf. Computer*, Vol. 6, pp. 79–98, 1963.

[6] E.D. Mastrovito and T. Mora, "VLSI Architectures for Multiplication over Finite Field $GF(2^m)$," in *Proceedings of 6th International Conference on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes (AAECC-6)*, pp. 297–309, July 1988.

[7] T. Itoh, S. Tsujii, "Structure of Parallel Multipliers for a Class of Fields $GF(2^m)$," *Information and Computation*, Vol. 83, pp. 21–40, 1989.

[8] C.-Y. Lee, E.H. Lu, J.Y. Lee, "Bit-parallel Systolic Multipliers for $GF(2^m)$ Fields Defined by All-one and Equally-spaced Polynomials," *IEEE Trans. Computers*, Vol.50, No.5, pp. 385–393, May 2001.

[9] W.-T. Huang, C. H. Chang, C. W. Chiou, S.-Y. Tan, "Non-XOR Approach for Low-cost Bit-parallel Polynomial Basis Multiplier over $GF(2^m)$," *IET Information Security*, Vol. 5, No. 3, pp. 152-162, Sep. 2011.

[10] C. W. Chiou, C.-Y. Lee, Y.-C. Yeh, "Multiplexer Implementation of Low-complexity Polynomial Basis Multiplier in GF(2^m) Using All One Polynomial," *Information Processing Letters*, Vol. 111, No. 3.1, pp.1044-1047, Jan. 2011.

[11] C.-Y. Lee, C.W. Chiou, "Efficient Design of Low-complexity Bit-Parallel Systolic Hankel Multipliers to Implement Multiplication in Normal and Dual Bases of $GF(2^m)$," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science*, Vol. E88-A, No. 11, pp. 3169–3179, Nov. 2005.

[12] C.W. Chiou, C.-Y. Lee, "Multiplexer-based Double-Exponentiation for Normal Basis of $GF(2^m)$," *Computers & Security*, Vol. 24, No. 1, pp. 83–86, 2005.

[13] H. Wu, M. A. Hasan, I. F. Blake, "New Low-complexity Bit-parallel Finite Field Multipliers Using Weakly Dual Bases," *IEEE Trans. Computers*, Vol. 47, No. 11, pp.1223-1234, Nov. 1998.

[14] J.-H. Wang, H.W. Chang, C.W. Chiou, W.-Y. Liang, "Low-complexity Design of Bit-parallel Dual Basis Multiplier over GF(2^m)," *IET Information Security*, Vol. 6, No. 4, pp. 324-328, Dec. 2012.

[15] Y.Y. Hua, J.-M. Lin, C.W. Chiou, C.-Y. Lee, Y. H. Liu, "A Novel Digit-serial Dual Basis Systolic Karatsuba Multiplier over GF(2^m)," *Journal of Computers*, Vol. 23, No. 2, pp. 80-94, July 2012.

[16] J.L. Massey, J.K. Komura, *Computational Method and Apparatus for Finite Field Arithmetic*, U.S. Patent 4,587,627, 1986.

[17] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," in *Proceedings of Crypto* 1997, Lecture Notes in Computer Science, Vol. 1294, pp. 513–525, 1997.

[18] C.W. Chiou, C.-Y. Lee, "Multiplexer-Based Double-Exponentiation for Normal Basis of GF (2^m)," *Computers & Security*, Vol. 24, No. 1, pp. 83-86, 2005.

[19] C.W. Chiou, T.-P. Chuang, S.-S. Lin, C.-Y. Lee, J.-M. Lin, Y.-C. Yeh, "Palindromic-like Representation for Gaussian Normal Basis Multiplier over GF(2^m) with Odd Type-t," *IET Information Security*, Vol. 6, No. 4, pp. 318-323, Dec. 2012.

[20] C.W. Chiou, H.W. Chang, W.-Y. Liang, C.-Y. Lee, J.-M. Lin, Y.-C. Yeh, "Low-complexity Gaussian Normal Basis Multiplier over $GF(2^m)$," *IET Information Security*, Vol. 6, No. 4, pp. 310-317, Dec. 2012.

[21] C.-Y. Lee, C. W. Chiou, "Scalable Gaussian Normal Basis Multipliers over GF(2^m) Using Hankel Matrix-vector Representation," *Journal of Signal Processing Systems for Signal Image and Video Technology*, Vol.69, No.2, pp.197-211, Nov. 2012.

[22] T.-P. Chuang, C. W. Chiou, S.-S. Lin, C.-Y. Lee, "Fault-tolerant Gaussian Normal Basis Multiplier over GF(2^m)," *IET Information Security*, Vol. 6, No. 3, pp. 157-170, Sep.. 2012.

[23] R. Azarderakhsh, A. Reyhani-Masoleh, "Low-complexity Multiplier Architectures for Single and Hybrid-double Multiplications in Gaussian Normal Bases," *IEEE Trans. Computers*, Vol.62, No.4, pp.744-757, April 2013.

[24] D. Boneh, R. Demillo, R. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," in *Proceedings of EUROCRYPT'97*, Lecture Notes in Computer Science, Vol. 1233, pp. 37–51, 1997.

[25] J. Kelary, B. Schneier, D. Wanger, C. Hall, "Side-channel Cryptanalysis of Product Ciphers," in *Journal of Computer Security*, Vol. 8, No. 2-3, pp. 141–158, Sep. 2000.

[26] S. Fenn, M. Gossel, M. Benaissa, D. Taylor, "On-line Error Detection for Bit-Serial Multipliers in $GF(2^m)$," *Journal of Electronic Testing: Theory and Applications*, Vol. 13, pp. 29–40, 1998.

[27] A. Reyhani-Masoleh, M.A. Hasan, "Error Detection in Polynomial Basis Multipliers over Binary Extension Fields," in *Proceedings of CHES 2002*, Lecture Notes in Computer Science, Vol. 2523, pp. 515–528, 2003.

[28] A. Reyhani-Masoleh, M.A. Hasan, "Fault Detection Architectures for Field Multiplication Using Polynomial Bases," *IEEE Trans. Computers*, Vol. 55, No. 9, pp. 1089–1103, 2006.

[29] S. Bayat-Sarmadi, M.A. Hasan, "On Concurrent Detection of Errors in Polynomial Basis Multiplication," *IEEE Trans. VLSI Systems*, Vol. 15, No. 4, pp. 413–426, 2007.

[30] 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.

[31] C.-Y. Lee, C. W. Chiou, and J.-M. Lin, "Concurrent Error Detection in a Polynomial Basis Multiplier over $GF(2^m)$," *Journal of Electronic Testing: Theory and Applications*, Vol. 22, No. 2, pp. 143–150, 2006.

[32] C.W. Chiou, C.-Y. Lee, A. W. Deng, J.-M. Lin, "Concurrent Error Detection in Montgomery Multiplication over

GF(2^m)," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Science*, Vol. E89-A, No. 2, pp. 566–574, 2006.

[33] H. Yamamoto, T. Watanabe, Y. Urano, "Alternating Logic and Its Application to Fault Detection," in *Proceedings of 1970 IEEE Int. Computing Group Conference*, pp. 220–228, 1970.

[34] D. A. Reynolds, G. Metze, "Fault Detection Capabilities of Alternating Logic," *IEEE Trans. Computers*, Vol. 27, No. 12, pp. 1093–1098, 1978.

[35] S. E. Woodard, "Design of Digital Systems Using Self-checking Alternating Logic," Ph.D. Thesis, University of Illinois at Urbana-Champaign, U.S.A., 1977.

[36] S. Bayat-Sarmadi, M.A. Hasan, "Concurrent Error Detection in Finite-field Arithmetic Operations Using Pipelined and Systolic Architectures," *IEEE Trans. Computers*, Vol. 58, No. 11, pp. 1553–1567, 2009.

[37] R. J. Baker, *CMOS-circuit, Design, Layout, and Simulation*, 2nd Edition, IEEE Press; 2004.

[38] J. L. Mana, R. Hermida, F. Tirado, "Low Complexity Bit-parallel Multipliers Based on A Class of Irreducible Pentanomials," *IEEE Trans. VLSI Systems*, Vol. 14, No. 12, pp. 1388–1393, 2006.