Concurrent Error Detection in Montgomery Multiplication over Binary Extension Fields

Arash Hariri, *Student Member*, *IEEE*, and Arash Reyhani-Masoleh, *Member*, *IEEE*

Abstract—Multiplication is one of the most important operations in finite field arithmetic. It is used in cryptographic and coding applications, such as elliptic curve cryptography and Reed-Solomon codes. In this paper, we consider the finite field multiplication used in elliptic curve cryptography and design concurrent error detection circuits. It is shown in the literature that the Montgomery multiplication can be used in cryptography to accelerate the scalar multiplication. Here, we use a parity-based concurrent error detection approach to increase the reliability of different Montgomery multipliers available in the literature. First, we consider bit-serial Montgomery multiplication and propose an error detection circuit. Then, we apply the same technique on the digit-serial Montgomery multiplication. Finally, we consider low time-complexity bit-parallel Montgomery multiplication and design the required components to implement the concurrent error detection circuits. ASIC implementations have been completed to analyze the time and area overheads of the proposed schemes. Also, the error detection capability is investigated by software simulations. We show that our approach results in efficient error detection schemes with small time and area overheads.

Index Terms—Montgomery multiplication, concurrent error detection, finite fields, elliptic curve cryptography.

1 Introduction

Finite field arithmetic has important applications in coding theory and cryptography. The main operation in the elliptic curve cryptography (ECC) [1], [2], is the scalar multiplication which is based on finite field arithmetic. Furthermore, one of the most important operations in the finite field arithmetic is multiplication. This operation has been studied in the literature extensively and different multiplication algorithms have been proposed, e.g., [3], [4], [5], [6], [7], [8], [9], [10], and [11].

The Montgomery multiplication algorithm over binary extension fields is outlined in [7] and is based on the original Montgomery multiplication algorithm of [12] which is proposed for integers. In [7], three Montgomery multiplication algorithms are proposed for bit-serial, digit-serial, and bit-parallel multiplication. In [13], bit-parallel Montgomery multiplication and squaring algorithms are proposed using irreducible trinomials which provide better time complexity in comparison to the bit-parallel polynomial basis multipliers and squarers. Another Montgomery multiplier is proposed in [14] which provides a low latency. Very recently, a number of bit-serial and bit-parallel Montgomery multipliers are proposed in [15], which show that the Montgomery multiplication can accelerate the ECC scalar multiplication. The Montgomery multiplication is used in [16] to implement an ECC-based cryptosystem.

• *The authors are with the Department of Electrical and Computer Engineering, The University of Western Ontario, 1151 Richmond Street North, Faculty of Engineering, Thompson Engineering Building, London, Ontario, Canada N6A 5B9. E-mail: arash.hariri@gmail.com, areyhani@eng.uwo.ca.*

Manuscript received 1 June 2009; revised 15 Mar. 2010; accepted 8 June 2010; published online 8 Dec. 2010.
Recommended for acceptance by C. Metra and R. Galivanche.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-2009-06-0249.
Digital Object Identifier no. 10.1109/TC.2010.258.

Concurrent error detection is a process used to test the operation of a system while it is operating normally [17]. Different techniques [18] are used in this regard, which include hardware duplication, parity codes, time redundancy, redundant residue number system [19], etc. Due to the fact that fault injection and active attacks are used against cryptosystems (cf. [20], [21]), it is very important to increase the reliability of the elliptic curve-based cryptosystems, and in particular, its main arithmetic operation, i.e., multiplication. There are different works available in the literature which consider concurrent error detection for finite field multiplication. In [22], a parity-based approach is used to detect errors in bit-serial polynomial and normal basis multipliers. A similar technique is used in [23] for bit-serial and bit-parallel polynomial basis multipliers. This scheme is extended to a multibit parity approach in [24] for error detection in bit-serial and bit-parallel polynomial basis multipliers. Based on interlacing parity codes, another approach is proposed in [25] for a bit-parallel polynomial basis multiplier. In addition to the parity-based approaches, time redundancy is also used for error detection in finite field multiplication. This technique is mainly used for semisystolic and systolic implementations of the finite field multiplication. In [26], time-redundancy-based error detection techniques are proposed for different pipelined systolic multipliers. For more time-redundancy-based approaches, one can refer to [27], [28], and [29], to name a few.

Concurrent error detection for the Montgomery multiplication over binary fields has been considered in the literature. In [30], a time-redundancy-based error detection approach is used for the semisystolic array implementation of the Montgomery multiplication [30]. Their approach uses REcomputing with Shifted Operands (RESO) and alternate data retry. Also, [31] proposes an improved time-redundancy-based approach for the semisystolic array implementation, as well as a single-bit parity-based technique for concurrent error detection in the bit-serial Montgomery multiplication.

In this paper, we extend our previous work on concurrent error detection in the Montgomery multiplication [31] to increase the error detection capability, which can be used as a countermeasure against natural faults and fault attacks in cryptography. First, we consider bit-serial Montgomery multiplication and propose a concurrent error detection circuit for the bit-serial multiplier of [7]. Then, we consider digit-serial Montgomery multiplication and propose a multibit concurrent error detection circuit for the digit-serial Montgomery multiplier of [7]. We note that to the best our knowledge, no previous work has considered error detection in digit-serial polynomial basis multiplication. Finally, we choose the bit-parallel Montgomery multiplier proposed in [15] and derive the concurrent error detection structure. We have implemented the proposed structures in ASIC and simulated their error detection capability using C++ for various cases. The results show that the area and time overheads are small and the error detection capability is significant.

The remainder of this paper is organized as follows: In Section 2, we present a brief introduction to finite fields and the Montgomery multiplication, and present our error detection approach. In Section 3, we present a concurrent error detection circuit for bit-serial Montgomery multiplication. In Sections 4 and 5, we consider concurrent error detection in digit-serial and bit-parallel Montgomery multiplication, respectively. In Section 6, we present our analysis and complexity results in terms of error simulations and ASIC implementations. Finally, we conclude this paper in Section 7.

2 Preliminaries

2.1 Binary Extension Fields

The binary extension field $GF(2^m)$ is constructed using an irreducible polynomial $F(z)$ of degree m shown as

$$F(z) = z^m + f_{m-1}z^{m-1} + \cdots + f_1 z + 1, \quad (1)$$

where $f_i \in \{0,1\}$ for $i = 1$ to $m-1$. This field contains 2^m field elements and is an extension of the basic field $GF(2) = \{0,1\}$. The field elements can be represented using different representation bases. Assuming x is a root of $F(z)$, i.e., $F(x) = 0$, any field element of $GF(2^m)$ can be represented as a polynomial of degree $m-1$. For instance, if $A \in GF(2^m)$, then it can be represented as

$$A = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0,$$

where $a_i \in \{0,1\}$ for $i = 0$ to $m-1$. This representation is called the polynomial basis. In this basis, addition of two field elements is carried out by pairwise XOR operation, e.g.,

$$A + B = (a_{m-1} + b_{m-1})x^{m-1} + \cdots + (a_0 + b_0).$$

The multiplication and squaring operations are more complicated and require more resources.

2.2 The Montgomery Multiplication over $GF(2^m)$

The Montgomery multiplication over binary extension fields is proposed in [7] and is based on the Montgomery multiplication algorithm for integers introduced in [12]. To explain this multiplication algorithm, we assume that A and B are two field elements of $GF(2^m)$, $F(z)$ is an irreducible polynomial of degree m and $F(x) = 0$. Now, let r be a fixed polynomial satisfying $\gcd(r, F(x)) = 1$, and r^{-1} and $\dot{F}(x)$ be two polynomials computed using the extended euclidean algorithm to satisfy

$$r \cdot r^{-1} + F(x) \cdot \dot{F}(x) = 1. \quad (2)$$

The general case of the Montgomery multiplication over $GF(2^m)$ is formulated as

$$C = A \cdot B \cdot r^{-1} \bmod F(x), \quad (3)$$

where r^{-1}, computed in (2), is in fact the inverse of r modulo $F(x)$. Algorithm 1 shows the Montgomery multiplication algorithm proposed in [7] to obtain C as formulated in (3).

Algorithm 1 The Montgomery Multiplication over $GF(2^m)$ [7]

Inputs: $A, B \in GF(2^m), r, F(x), \dot{F}(x)$
Output: $C = A \cdot B \cdot r^{-1} \bmod F(x)$
Step 1: $t := A \cdot B$
Step 2: $u := t \cdot \dot{F}(x) \bmod r$
Step 3: $C := (t + u \cdot F(x))/r$

Unlike the polynomial basis multiplication, where the multiplication is done modulo $F(x)$, in the Montgomery multiplication shown in Algorithm 1, Step 2 is done in modulo r and Step 3 requires a division by r. In [7], it is proposed that $r = x^m$ results in a better multiplication algorithm because modulo x^m operation is performed by using the m least significant coordinates of the operation. Also, a division by x^m is performed by shifting to the right. Thus, choosing the polynomial r, also known as the Montgomery factor, significantly affects the efficiency of this multiplication. Note that $r = x^u$, $0 < u < m$, has the same properties, however, choosing the value of u from this interval can result in faster Montgomery multiplication (see for example [13] or [15]).

2.3 Concurrent Error Detection Approach

Here, we explain our approach to implement concurrent error detection circuits for different Montgomery multipliers. In this approach, the residue of one of the operands (A as shown in Fig. 1) modulo a fixed polynomial $(x^l + 1)$ is computed and the residue of the multiplication product modulo the same polynomial is predicted (i.e., $C \bmod (x^l + 1)$). Different polynomials can be used to compute the residues. In this paper, we choose $x^l + 1$, where $1 < l < m$ is the number of the redundant bits used for concurrent error detection. Note that using $l = 1$ is equivalent to the single-bit parity-based approach, whereas $1 < l < m$ results in an l-bit interlacing parity code. Throughout this paper, we will use the hat notation to denote modulo $(x^l + 1)$ reduction, i.e.,

$$\hat{A} = A \bmod (x^l + 1) = \hat{a}_{l-1}x^{l-1} + \cdots + \hat{a}_1 x + \hat{a}_0,$$

where $A \in GF(2^m)$, $\hat{a}_i \in \{0,1\}$, and $0 \le i \le l-1$. Also, we will use the following properties of modular reduction:

$$(A + B) \bmod (x^l + 1) = \hat{A} + \hat{B}, \quad (4)$$

where A and B are two field elements of $GF(2^m)$ and

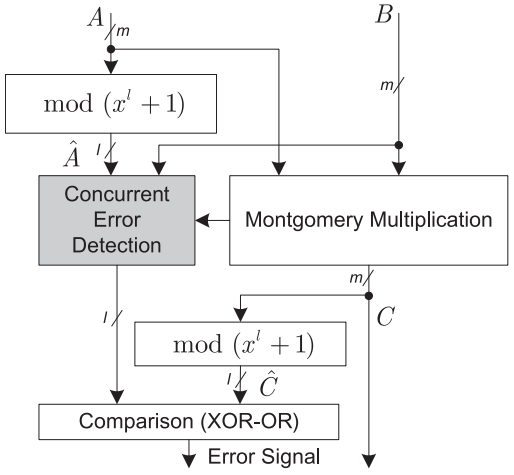

Fig. 1. Concurrent error detection scheme.

$$(b \cdot A) \bmod (x^l + 1) = b \cdot \hat{A}, \tag{5}$$

where $b \in GF(2)$.

The concurrent error detection approach has been depicted in Fig. 1. It is assumed that A and $\hat{A}$ are given at the same time, and the multiplication and concurrent error detection blocks run in parallel. The output of the concurrent error detection block is the predicted $C \bmod (x^l + 1)$. To find the possible errors, the actual $\hat{C}$ is computed using the output of the multiplier (i.e., C) and compared to the predicted one. The error signal is asserted high, if the actual and predicted residues are different. Note that there is a connection between the Concurrent Error Detection module and the multiplier in Fig. 1.

3 Concurrent Error Detection in the Bit-Serial Montgomery Multiplication over $GF(2^m)$

In this section, we consider the concurrent error detection scheme for the bit-serial Montgomery multiplier proposed in [7]. A single-bit parity code has been applied on this multiplier in [31]. However in this paper, we use another approach using multibit parities to improve the error detection capability of this multiplier.

Algorithm 2 shows the bit-serial Montgomery multiplication algorithm proposed in [7]. Combining Steps 4 and 5 of this algorithm, one can obtain

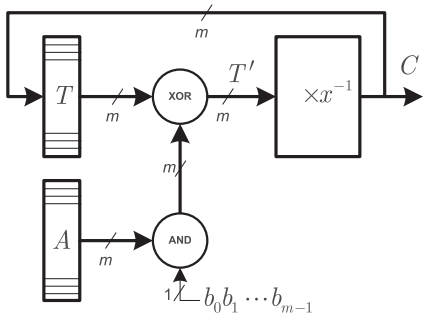

Fig. 2. Modified bit-serial Montgomery multiplication over $GF(2^m)$.

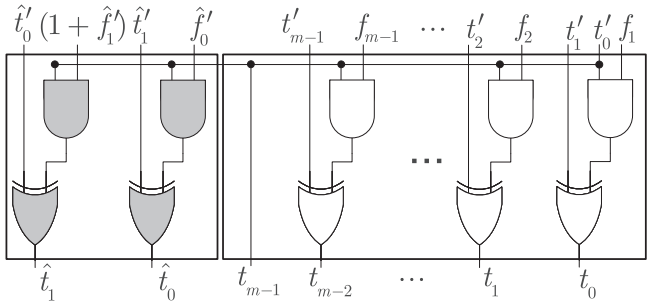

Fig. 3. The x^{-1}-module with error detection using $l = 2$. The gray gates show the overhead.

$$T = (T' + t'_0 F(x))/x. \tag{6}$$

It can be shown that the operation formulated in (6) is equivalent to $T = T' \cdot x^{-1} \bmod F(x)$. This is because one can write the following:

$$\begin{aligned} F'(x) = F(x)/x &= x^{-1} \bmod F(x) \\ &= f_m x^{m-1} + f_{m-1} x^{m-2} + \cdots + f_1. \end{aligned} \tag{7}$$

Then, 6 can be written as

$$T = \left(t'_{m-1} x^{m-2} + \cdots + t'_1\right) + t'_0 F'(x) = T' \cdot x^{-1} \bmod F(x). \tag{8}$$

By replacing Steps 4 and 5 of Algorithm 2 with $T = T' \cdot x^{-1} \bmod F(x)$, one can obtain the modified algorithm, as shown in Algorithm 3. The corresponding hardware architecture of this multiplier has been depicted in Fig. 2. In this figure, A and T are two m-bit registers initialized with the coordinates of the operand A and $0 \in GF(2^m)$, respectively. Also, the module represented by $\times x^{-1}$ performs a multiplication by x^{-1} followed by a reduction by $F(x)$. In this paper, this module is named as the x^{-1}-module and Fig. 3 shows its hardware implementation using white gates [15].

Algorithm 2 Bit-Serial Montgomery Multiplication over $GF(2^m)$ [7]

Inputs: $A, B, F(x)$
Output: $C = A \cdot B \cdot x^{-m} \bmod F(x)$
Step 1: $T := 0$
Step 2: For $i := 0$ to $m - 1$
Step 3: $T' := T + b_i A$
Step 4: $T'' := T' + t'_0 F(x)$
Step 5: $T := T''/x$
Step 6: $C := T$

Algorithm 3 The Modified Bit-Serial Montgomery Multiplication over $GF(2^m)$

Inputs: $A, B, F(x)$
Output: $C = A \cdot B \cdot x^{-m} \bmod F(x)$
Step 1: $T := 0$
Step 2: For $i := 0$ to $m - 1$
Step 3: $T' := T + b_i A$
Step 4: $T := T' \cdot x^{-1} \bmod F(x)$
Step 6: $C := T$

The other modules denoted by AND and XOR in Fig. 2 perform logical operations corresponding to Step 3 of

Algorithm 3 (i.e., $T' := T + b_i A$) using m two-input AND gates and m two-input XOR gates, respectively.

3.1 Formulation

To design a concurrent error detection mechanism for this multiplier, we consider each step in Algorithm 3 separately. We begin with Step 3 of this algorithm and present the following lemma.

Lemma 1. *Assume that $T^{(i-1)}$ is the content of T at the $(i-1)$th iteration of Algorithm 3 and $\hat{T}^{(i-1)} = \hat{t}_{l-1}^{(i-1)} x^{l-1} + \cdots + \hat{t}_1^{(i-1)} x + \hat{t}_0^{(i-1)}$. Now, the coordinates of $\hat{T}'^{(i)}$ in Step 3 of Algorithm 3 can be computed as $\hat{t}'^{(i)}_j = \hat{t}_j^{(i-1)} + b_i \hat{a}_j$, $0 \le j < l$.*

Proof. Using the computation of $T'^{(i)} = T^{(i-1)} + b_i A$ in Step 3 of Algorithm 3, and the properties 4 and 5, one can obtain $\hat{T}'^{(i)} = \hat{T}^{(i-1)} + b_i \hat{A}$. Thus, the coordinates of $\hat{T}'^{(i)}$ can be obtained from

$$\hat{t}'^{(i)}_j = \hat{t}_j^{(i-1)} + b_i \hat{a}_j, \quad 0 \le j < l \tag{9}$$

and the proof is complete. □

Step 4 of Algorithm 3 performs $T^{(i)} = T'^{(i)} \cdot x^{-1} \bmod F(x)$. Thus, by computing the both sides of this equation modulo $(x^l + 1)$, one can obtain

$$\hat{T}^{(i)} = \left(T'^{(i)} \cdot x^{-1} \bmod F(x)\right) \bmod (x^l + 1).$$

To consider the concurrent error detection for this operation, we first present the following lemma for Step 4 of Algorithm 3.

Lemma 2. *Let T' and $T = T' \cdot x^{-1} \bmod F(x)$ be two field elements in $GF(2^m)$ constructed by the irreducible polynomial $F(z)$ and $F(x) = 0$. Also, assume $\hat{T}' = \hat{t}'_{l-1} x^{l-1} + \cdots + \hat{t}'_1 x + \hat{t}'_0$ and $\hat{F}'(x) = \hat{f}'_{l-1} x^{l-1} + \cdots + \hat{f}'_1 x + \hat{f}'_0$, where $F'(x)$ is defined in (7). Then, the coordinates of $\hat{T} = T \bmod (x^l + 1)$ can be found as follows:*

$$\hat{t}_j = \begin{cases} \hat{t}'_{j+1} + t'_0 \hat{f}'_j & 0 \le j < l-1 \\ \hat{t}'_0 + t'_0 \left(1 + \hat{f}'_{l-1}\right) & j = l-1. \end{cases}$$

Proof. Computing (8) modulo $(x^l + 1)$, one can write the following:

$$\hat{T} = \left(t'_{m-1} x^{m-2} + \cdots + t'_1 + t'_0 F'(x)\right) \bmod (x^l + 1). \tag{10}$$

First, we consider the first m terms in (10) and write

$$\begin{aligned} &(t'_{m-1} x^{m-2} + \cdots + t'_1) \bmod (x^l + 1) \\ &\quad = \left(\left(t'_{m-1} x^{m-1} + \cdots + t'_1 x + t'_0\right) \cdot x^{-1} + t'_0 x^{-1}\right) \\ &\qquad \bmod (x^l + 1), \end{aligned}$$

which can be rewritten as

$$\left(\left(\hat{t}'_{l-1} x^{l-1} + \cdots + \hat{t}'_1 x + \hat{t}'_0\right) \cdot x^{-1} + t'_0 x^{-1}\right) \bmod (x^l + 1),$$

or

$$\left(\hat{t}'_{l-1} x^{l-2} + \cdots + \hat{t}'_1 + \hat{t}'_0 x^{-1} + t'_0 x^{-1}\right) \bmod (x^l + 1). \tag{11}$$

Taking into account that $x^{-1} \bmod (x^l + 1) = x^{l-1}$ and by applying the properties mentioned in (4) and (5) on (10) and (11), one can write

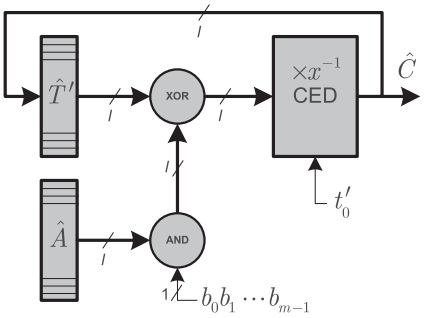

Fig. 4. Concurrent error detection for the bit-serial Montgomery multiplier.

$$\begin{aligned} \hat{T} = &(\hat{t}'_0 + t'_0(1 + \hat{f}'_{l-1})) x^{l-1} + \cdots + (\hat{t}'_2 + t'_0 \hat{f}'_1) x \\ &+ (\hat{t}'_1 + t'_0 \hat{f}'_0), \end{aligned}$$

which completes the proof. □

3.2 Architecture

The hardware structure of the x^{-1}-module with its error detection mechanism is shown in Fig. 3 for the general irreducible polynomial $F(x)$ and $l = 2$. In this figure, the white gates perform the normal function of the module and the gray gates are the overhead to perform the error detection. It is noted that $F(x)$ is usually fixed in cryptographic applications, and as a result, most of the gates shown in Fig. 3 can be removed.

Remark 1. The concurrent error detection circuit in the x^{-1}-module requires at most l two-input XOR gates for a fixed irreducible polynomial $F(z)$. However, assuming $F(z)$ is an ω-nomial (i.e., $F(z)$ has ω nonzero coordinates), this circuit requires at most $\min(\omega, l)$ two-input XOR gates. Note that in practical cryptographic applications ω equals three or five [32].

The overhead circuit to perform concurrent error detection in the bit-serial Montgomery multiplier is shown in Fig. 4. In this figure, $\hat{T}'$ and $\hat{A}$ are two l-bit registers which store $T' \bmod (x^l + 1)$ and $A \bmod (x^l + 1)$, respectively. The gray blocks labeled AND and XOR perform logical operations based on Lemma 1, and the block labeled $\times x^{-1}$ CED, is constructed using Lemma 2 and is similar to the left-hand side of Fig. 3. Finally, $t'_0 = t_0^{(i-1)} + b_i a_0$ is the LSB of T' computed by the multiplier shown in Fig. 2.

The bit-serial Montgomery multiplier shown in Fig. 2 requires $2m - 1$ two-input XOR gates and $2m - 1$ two-input AND gates for general irreducible polynomials. Also, it requires two m-bit registers to store T and A. The critical path delay of this multiplier is $2T_A + 2T_X$, where T_A and T_X represent the delays of a two-input AND gate and a two-input XOR gate, respectively.

Since both Figs. 2 and 4 have the critical path delay of $2T_A + 2T_X$, implementing the proposed concurrent error detection mechanism imposes time and area overheads as a result of the comparison and modulo $(x^l + 1)$ blocks in Fig. 1. Besides these two components, the area overhead of the concurrent error detection circuit (without fixing the irreducible polynomial) is $2l$ two-input AND gates, $2l$ two-input XOR gates, and $2l$ flip-flops.

Remark 2. Using Lemma 1, one can design similar concurrent error detection circuits for the bit-serial multipliers of [15].

4 Concurrent Error Detection in Digit-Serial Montgomery Multiplication over $GF(2^m)$

In this section, we consider concurrent error detection for the digit-serial Montgomery multiplier proposed in [7]. In this multiplier, the operand B is split into D-bit digits ($D \geq 2$) as

$$B = B_{n-1}x^{(n-1)D} + \cdots + B_1 x^D + B_0,$$

where $n = \lceil \frac{m}{D} \rceil$, $B_i = \sum_{j=0}^{D-1} b_{iD+j}x^j$ for $0 \leq i \leq n-2$ and $B_{n-1} = \sum_{j=0}^{m-1-D(n-1)} b_{(n-1)D+j}x^j$. This Montgomery multiplication algorithm is shown in Algorithm 4.

Algorithm 4 Digit-serial Montgomery Multiplication over $GF(2^m)$ [7]

Inputs: A, B, , $F(x)$, $\dot{F}_0(x)$
Output: $C = A \cdot B \cdot x^{-nD} \bmod F(x)$
Step 1: $C := 0$
Step 2: For $i := 0$ to $n-1$
Step 3: $C' := C + B_i A$
Step 4: $M := C'_0 \cdot \dot{F}_0(x) \bmod x^D$
Step 5: $C'' := C' + M \cdot F(x)$
Step 6: $C := C''/x^D$

From (2), it follows that $F(x) \cdot \dot{F}(x) = 1 \bmod x^D$ for the Montgomery multiplication over binary extension fields and $r = x^m$. It is noted that the following property also holds [7]:

$$F_0(x) \cdot \dot{F}_0(x) = 1 \bmod x^D, \tag{12}$$

where $F_0(x)$ and $\dot{F}_0(x)$ in (12) are polynomials with the degree of at most $(D-1)$ and represent the least significant digits of $F(x)$ and $\dot{F}(x)$, respectively. One can notice that $\dot{F}_0(x) = 1$ simplifies the digit-serial Montgomery multiplication. Now, we assume that the irreducible polynomial is of the form $F(x) = x^m + \sum_{i=k+1}^{m-1} f_i x^i + x^k + 1$, where k is the degree of the second smallest nonzero term in $F(x)$. In this case, for $D \leq k$, we always have $F_0(x) = 1$. Replacing $F_0(x)$ with 1 in (12) results in $\dot{F}_0(x) = 1$, which simplifies Step 4 of Algorithm 4 to $M := C'_0 \bmod x^D = C'_0$.

Now, Steps 5 and 6 of Algorithm 4 can be combined as

$$C = \big(C' + C'_0 F(x)\big)/x^D. \tag{13}$$

Similar to (6), (13) can be written as follows:

$$C = C' \cdot x^{-D} \bmod F(x). \tag{14}$$

It is shown in [33] that this operation is optimized for the shifted polynomial basis if $D \leq k$. It is also true for the Montgomery multiplication and is compatible with our previous assumption to have $\dot{F}_0(x) = 1$. Thus, we will use the condition $D \leq k$ in the rest of this paper. Replacing Steps 5 and 6 of Algorithm 4 with 14, the modified digit-serial Montgomery multiplication algorithm is shown in Algorithm 5.

Algorithm 5 The Modified Digit-Serial Montgomery Multiplication over $GF(2^m)$

Inputs: A, B, , $F(x)$
Output: $C = A \cdot B \cdot x^{-nD} \bmod F(x)$
Step 1: $C := 0$
Step 2: For $i := 0$ to $n-1$
Step 3: $C' := C + B_i A$
Step 4: $C := C' \cdot x^{-D} \bmod F(x)$

4.1 Formulation

Now, we design the concurrent error detection scheme for this multiplier, where $D \leq k$. Let B_i, $0 \leq i < n$, denote the ith digit of B in Step 3 of Algorithm 5 and $B_{i,j}$, $0 \leq j < D$ represent its jth coordinate. It can be represented in the polynomial basis as

$$B_i = B_{i,D-1}x^{D-1} + \cdots + B_{i,1}x + B_{i,0}. \tag{15}$$

Using (15), the multiplication of A by B_i in Step 3 of Algorithm 5 modulo $(x^l + 1)$ can be written as

$$A \cdot B_i \bmod (x^l+1) = \sum_{j=0}^{D-1} B_{i,j} \cdot A \cdot x^j \bmod (x^l+1). \tag{16}$$

The following lemma is used to evaluate (16).

Lemma 3. *Let A be a field element in $GF(2^m)$ and $\hat{A} = A \bmod (x^l+1) = \sum_{t=0}^{l-1} \hat{a}_t x^t$. Then $A \cdot x^j \bmod (x^l+1)$ can be obtained by j-bit circular left shift of $\hat{A}$, where $j \geq 0$.*

Proof. If $j = 0$, the lemma is clear. For $j > 0$, first we compute $A \cdot x \bmod (x^l+1)$ as

$$A \cdot x \bmod (x^l+1) = \left(x \cdot \sum_{t=0}^{l-1} \hat{a}_t x^t \right) \bmod (x^l+1). \tag{17}$$

Taking into account that $x^l \bmod (x^l+1) = 1$, (17) can be written as

$$A \cdot x \bmod (x^l+1) = \sum_{t=0}^{l-2} \hat{a}_t x^{t+1} + \hat{a}_{l-1}. \tag{18}$$

One can notice that $A \cdot x \bmod (x^l+1)$ in (18) is obtained by one-bit circular left shift of $\hat{A}$. Similarly, it can be shown that $A \cdot x^j \bmod (x^l+1)$ is obtained by j-bit circular left shift of $\hat{A}$. □

Assuming $\hat{A}_{(j)} = A \cdot x^j \bmod (x^l+1)$, (16) can be rewritten as

$$A \cdot B_i \bmod (x^l+1) = \sum_{j=0}^{D-1} B_{i,j} \cdot \hat{A}_{(j)}.$$

Consequently, from Step 3 of Algorithm 5 and using the property mentioned in (4), one can obtain the following:

$$\hat{C}' = \hat{C} + \sum_{j=0}^{D-1} B_{i,j} \cdot \hat{A}_{(j)}, \tag{19}$$

where $\hat{C} = \sum_{t=0}^{l-1} \hat{c}_t x^t$.

Step 4 of Algorithm 5 performs a multiplication by x^{-D} followed by a reduction by $F(x)$ and we present the following lemma to design the concurrent error detection circuit for this operation.

Lemma 4. *Let S and $S' = S \cdot x^{-D} \bmod F(x)$ be two field elements in $GF(2^m)$ constructed by the irreducible polynomial $F(z) = z^m + \sum_{j=k+1}^{m-1} f_j z^j + z^k + 1$ and $F(x) = 0$. Assuming $\hat{S} = \hat{s}_{l-1}x^{l-1} + \cdots + \hat{s}_1 x + \hat{s}_0$, then $\hat{S}'$ is obtained as*

$$\hat{S}' = \sum_{i=0}^{l-1} \hat{s}_{|i+D|}x^i + \sum_{i=0}^{D-1} s_i x^{|-D+i|} + \sum_{j=0}^{D-1} s_{D-1-j} \sum_{i=0}^{l-1} \hat{f}'_{|i+j|}x^i,$$

where $D \le k$ and $\hat{F}'(x) = F'(x) \bmod (x^l + 1) = \sum_{i=0}^{l-1} \hat{f}'_i x^i$.

Proof. We begin deriving the error detection formulation with

$$\begin{aligned} S' =& (s_{m-1}x^{m-D-1} + \cdots + s_{D+1}x \\ &+ s_D + s_{D-1}x^{-1} + \cdots + s_0 x^{-D}) \bmod F(x). \end{aligned} \tag{20}$$

For $D \le k$ and $1 \le i < D$, one can write the following recursive formulation:

$$x^{-(i+1)} \bmod F(x) = x^{-(i)} \cdot x^{-1} \bmod F(x),$$

where

$$x^{-1} \bmod F(x) = F'(x) = f_m x^{m-1} + \cdots + x^{k-1}.$$

So, $x^{-(i+1)} \bmod F(x)$ can be obtained by shifting $F'(x)i$ bits to the right. Consequently, $(x^{-(i+1)} \bmod F(x)) \bmod (x^l + 1)$, $1 \le i < D$, can be computed by i-bit circular right shift of $F'(x) \bmod (x^l + 1)$. As a result, for $1 \le i < D$ one can write

$$\begin{aligned} \left(x^{-(i+1)} \bmod F(x)\right) \bmod (x^l + 1) =& \hat{f}'_{i-1}x^{l-1} + \cdots + \\ & \hat{f}'_{i+1}x + \hat{f}'_i. \end{aligned} \tag{21}$$

Let us represent S' shown in (20) as

$$S' = S \cdot x^{-D} \bmod F(x) = S'_2 + S'_1,$$

where

$$S'_1 = (s_{D-1}x^{-1} + \cdots + s_0 x^{-D}) \bmod F(x),$$

and

$$S'_2 = (s_{m-1}x^{m-D-1} + \cdots + s_{D+1}x + s_D).$$

To obtain $\hat{S}'_1 = S'_1 \bmod (x^l + 1)$, we use (21) and write

$$\hat{S}'_1 = \sum_{j=0}^{D-1} s_{D-1-j} \sum_{i=0}^{l-1} \hat{f}'_{|i+j|}x^i, \tag{22}$$

where $|i + j| = (i + j) \bmod l$. Also, $\hat{S}'_2 = S'_2 \bmod (x^l + 1)$ can be written as

$$\hat{S}'_2 = (S \cdot x^{-D} + s_{D-1}x^{-1} + \cdots + s_0 x^{-D}) \bmod (x^l + 1).$$

Note that $S \cdot x^{-D} \bmod (x^l + 1)$ is obtained by D-bit circular right shift of $\hat{S}$. Thus, $\hat{S}'_2$ can be written as

$$\hat{S}'_2 = \sum_{i=0}^{l-1} \hat{s}_{|i+D|}x^i + \sum_{i=0}^{D-1} s_i x^{|-D+i|}. \tag{23}$$

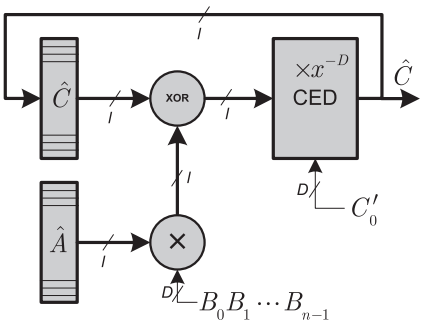

Fig. 5. Concurrent error detection in digit-serial Montgomery multiplication.

Now using (22) and (23), one can conclude that

$$\hat{S}' = \sum_{i=0}^{l-1} \hat{s}_{|i+D|}x^i + \sum_{i=0}^{D-1} s_i x^{|-D+i|} + \sum_{j=0}^{D-1} s_{D-1-j} \sum_{i=0}^{l-1} \hat{f}'_{|i+j|}x^i,$$

and the proof is complete. □

Corollary 1. *For the special case, where $D \le k$ and $D \le l$, the coordinates of $\hat{S}'$ are obtained as $\hat{s}'_i = \hat{s}_{i+D} + \sum_{j=0}^{D-1} s_{D-1-j}\hat{f}'_{i+j}$, when $0 \le i < l - D$, and $\hat{s}'_i = \hat{s}_{i-(l-D)} + s_{i-(l-D)} + \sum_{j=0}^{D-1} s_{D-1-j}\hat{f}'_{i+j}$, when $l - D \le i \le l - 1$.*

4.2 Architecture

The concurrent error detection scheme for the digit-serial Montgomery multiplication algorithm is shown in Fig. 5. In this figure, the module labeled $\times x^{-D}$ CED, implements Lemma 4 and the modules labeled $\times$ and XOR, realize (19). Also, $\hat{A}$ and $\hat{C}$ are two l-bit registers which are initialized with the coordinates of $A \bmod (x^l + 1)$ and 0, respectively. We have presented Example 1 in Appendix A to explain the structure of these modules in details.

The digit-serial Montgomery multiplier of Algorithm 4 requires two m-bit registers to store A and C. The $\times$-module requires $D \times m$ two-input AND gates and $(D-1) \times (m-1)$ two-input XOR gates. The XOR module requires m two-input XOR gates. Finally, the x^{-D} module requires $D \times (m-k)$ two-input AND gates and $D \times (m-k+1) - 1$ two-input XOR gates. Note that all the AND gates in the x^{-D} module can be removed if $F(z)$ is fixed. The critical path delay of the $\times$-module together with the XOR block is $T_A + \lceil log_2(D+1) \rceil \cdot T_X$. Also, the critical path delay of the x^{-D}-module is $\lceil log_2(D+1) \rceil \cdot T_X$ for a fixed $F(z)$. As a result, the maximum critical path delay of this multiplier is $T_A + 2\lceil log_2(D+1) \rceil \cdot T_X$.

The concurrent error detection for the $\times$-module is implemented using $D \times l$ two-input AND gates and $(D-1) \times l$ two-input XOR gates. Two l-bit registers are required to store $\hat{A}$ and $\hat{C}$. The number of the gates in the $\times x^{-D}$-CED module depends on the values of D and l. For example, the special case mentioned in Corollary 1 requires $D \times l$ two-input AND gates and $(D+1) \times l$ two-input XOR gates. Note that the AND gates in this block can be removed if $F(z)$ is fixed. Finally, the XOR block requires l two-input XOR gates. For the special case mentioned in Corollary 1, the critical path delay of the error detection scheme is $T_A + (\lceil log_2(D+1) \rceil + \lceil log_2(D+2) \rceil) \cdot T_X$.

5 Concurrent Error Detection in Bit-Parallel Montgomery Multiplication over $GF(2^m)$

In this section, we study concurrent error detection for the bit-parallel Montgomery multiplication. We apply the concurrent error detection approach used in this paper on the general bit-parallel Montgomery multiplier of [15]. This multiplier is based on a more general case of the Montgomery multiplication which generates

$$C = A \cdot B \cdot x^{-u} \bmod F(x), \tag{24}$$

where $0 < u \leq m$. In this case, (24) can be written as

$$C = \left(b_{m-1}x^{m-u-1} + \cdots + b_u + \cdots + b_0 x^{-u}\right) \cdot A \bmod F(x),$$

or

$$C = b_{m-1}Ax^{m-u-1} + \cdots + b_u A + \cdots + b_0 Ax^{-u} \bmod F(x). \tag{25}$$

In this section, we use $A^{(i)}$ to represent

$$A^{(i)} = Ax^i \bmod F(x). \tag{26}$$

The multiplication shown in (25) can be done by using the matrix $\mathbf{M}$, whose columns show the representation of $A^{(i)}$ with respect to the polynomial basis for $i \in [-u, m-u-1]$ [15]. So, the matrix $\mathbf{M}$ has m rows and m columns, and this matrix should be computed in the first step. Then, the Montgomery multiplication over $GF(2^m)$ can be formulated as

$$\mathbf{C} = \mathbf{M} \cdot \mathbf{B}^T, \tag{27}$$

where $\mathbf{C} = [c_0, c_1, \ldots, c_{m-1}]^T$ and $\mathbf{B} = [b_0, b_1, \ldots, b_{m-1}]$. Therefore, the second step of the bit-parallel Montgomery multiplication obtains (27), which can be written as

$$C = b_{m-1}A^{(m-u-1)} + \cdots + b_u A + \cdots + b_0 A^{(-u)}. \tag{28}$$

5.1 Formulation

We have explained the procedure to detect errors in the x^{-1}-module in Lemma 2. For the x-modules, first we define the following:

$$F''(x) = x^m \bmod F(x) = f_{m-1}x^{m-1} + \cdots + f_1 x + 1, \tag{29}$$

and represent $F''(x) \bmod (x^l + 1)$ as

$$\hat{F}'' = F''(x) \bmod (x^l + 1) = \sum_{t=0}^{l-1} \hat{f}''_t x^t. \tag{30}$$

Now, we present the following lemma which is similar to the results obtained in [24] and [25].

Lemma 5. *Let S and $S'' = S \cdot x \bmod F(x)$ be two field elements in $GF(2^m)$ constructed by the irreducible polynomial $F(z)$ and $F(x) = 0$. Assuming $\hat{S} = S \bmod (x^l + 1) = \hat{s}_{l-1}x^{l-1} + \cdots + \hat{s}_1 x + \hat{s}_0$, the coordinates of $\hat{S}'' = S'' \bmod (x^l + 1)$ equal*

$$\hat{s''}_j = \begin{cases} \hat{s}_{|j-1|} + s_{m-1}\left(1 + \hat{f''}_j\right) & if\ j = m \bmod l \\ \hat{s}_{|j-1|} + s_{m-1}\hat{f''}_j & otherwise, \end{cases}$$

where $|j-1|$ represents $(j-1) \bmod l$.

Proof. The field element S can be represented as $S = \sum_{t=0}^{m-1} s_t x^t$. Then, $S \cdot x = s_{m-1}x^m + \cdots + s_1 x^2 + s_0 x$. Using (29), one can write the following:

$$S \cdot x \bmod F(x) = s_{m-1}F''(x) + s_{m-2}x^{m-1} + \cdots + s_1 x^2 + s_0 x. \tag{31}$$

Using (30) and the property mentioned in (5), one can obtain the following:

$$s_{m-1}F''(x) \bmod (x^l + 1) = s_{m-1}\sum_{t=0}^{l-1} \hat{f}''_t x^t. \tag{32}$$

The rest of the terms in (31) can be written as

$$s_{m-2}x^{m-1} + \cdots + s_1 x^2 + s_0 x = S \cdot x + s_{m-1}x^m. \tag{33}$$

For the right-hand side of (33), one can write

$$(S \cdot x + s_{m-1}x^m) \bmod (x^l + 1) = (\hat{s}_{l-1}x^{l-1} + \cdots + \hat{s}_1 x + \hat{s}_0) \cdot x + s_{m-1}x^{|m|},$$

where $|m| = m \bmod l$. Taking into account that $x^l \bmod (x^l + 1) = 1$, one can obtain

$$(S \cdot x + s_{m-1}x^m) \bmod (x^l + 1) = \sum_{t=0}^{l-2} \hat{s}_t x^{t+1} + \hat{s}_{l-1} + s_{m-1}x^{|m|}. \tag{34}$$

Now using (31), (32), and (34), $\hat{S}''$ can be written as

$$\hat{S}'' = S'' \bmod (x^l + 1) = s_{m-1}\sum_{t=0}^{l-1} \hat{f}''_t x^t + \sum_{t=0}^{l-2} \hat{s}_t x^{t+1} + \hat{s}_{l-1} + s_{m-1}x^{|m|},$$

and the proof is complete. □

So far, we have obtained the concurrent error detection for $A^{(i)}$, $i \in [-u, m-u-1]$. We denote $A^{(i)} \bmod (x^l + 1)$ as $\hat{A}^{(i)}$. To design the concurrent error detection scheme for obtaining (27), we present the following lemma:

Lemma 6. *Assuming $A^{(i)}$, $i \in [-u, m-u-1]$, is defined based on 26 and $\hat{A}^{(i)} = A^{(i)} \bmod (x^l + 1) = \sum_{t=0}^{l-1} \hat{a}_t^{(i)} x^t$, the coordinates of $\hat{C} = C \bmod (x^l + 1) = \sum_{t=0}^{l-1} \hat{c}_t x^t$ can be obtained by $\hat{c}_t = \sum_{j=0}^{m-1} b_j \cdot \hat{a}_t^{(j-u)}$.*

Proof. To obtain $\hat{C}$, one can compute both sides of (28) modulo $(x^l + 1)$ and use the properties mentioned in (4) and (5) to obtain

$$C \bmod (x^l + 1) = b_{m-1} \cdot \hat{A}^{(m-u-1)} + \cdots + b_u \hat{A} + \cdots + b_0 \hat{A}^{(-u)}.$$

This results in the following:

$$\hat{c}_t = \sum_{j=0}^{m-1} b_j \cdot \hat{a}_t^{(j-u)}, \tag{35}$$

where $0 \leq t < l$, and the proof is complete. □

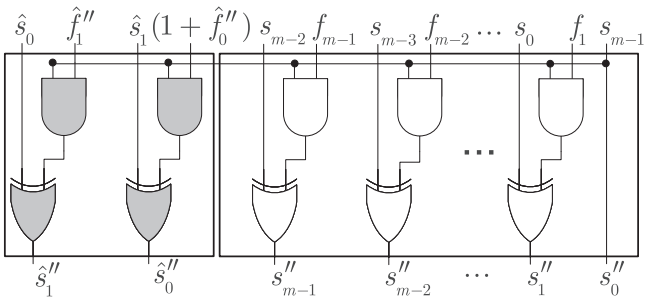

Fig. 6. Error detection in the x-module using $l = 2$.

5.2 Architecture

Using Lemmas 2 and 5, one can design the concurrent error detection circuit for the first step of the bit-parallel Montgomery multiplication (i.e., computing the matrix $\mathbf{M}$). The x^{-1}-module with concurrent error detection was shown before in Fig. 3. The concurrent error detection circuit for the x-module is shown in Fig. 6 for $l = 2$. Note that most of the gates can be removed if one fixes the irreducible polynomial. However, in the general case, the x-module requires $(m-1)$ two-input AND gates and $(m-1)$ two-input XOR gates. Also, its concurrent error detection circuit requires l two-input AND gates and l two-input XOR gates. The critical path delay of this module is still $T_A + T_X$. The following remark provides more details about the area complexity.

Remark 3. The concurrent error detection in the x-module requires at most l two-input XOR gates for a fixed irreducible polynomial. Assuming $F(z)$ is an ω-nomial, this circuit requires at most $\min(\omega, l)$ two-input XOR gates, where in practical cryptographic applications ω is three or five [32].

First, we investigate the time and area complexities of concurrent error detection in computing the matrix $\mathbf{M}$. We assume that there are λ and σ gray XOR gates in x^{-1} and x modules, respectively. For irreducible trinomials (respectively pentanomials), the maximum value of λ and σ is three (respectively five). As a result, the concurrent detection scheme for the matrix $\mathbf{M}$ requires at most $u \cdot \lambda + (m-u-1) \cdot \sigma$ two-input XOR gates. In this case, obtaining $\hat{A}^{(i)}$ for $i \in [-u, -1]$ has the delay of $\lceil \frac{u \cdot \lambda}{l} \rceil \cdot T_X$ and obtaining $\hat{A}^{(i)}$ for $i \in [1, m-u-1]$ has the delay of $\lceil \frac{(m-u-1)\sigma}{l} \rceil \cdot T_X$. Consequently, the theoretical time complexity of the concurrent error detection to obtain the matrix $\mathbf{M}$ is

$$Y = \max\left(\left\lceil \frac{u \cdot \lambda}{l} \right\rceil \cdot T_X, \left\lceil \frac{(m-u-1) \cdot \sigma}{l} \right\rceil \cdot T_X \right). \quad (36)$$

It is interesting to note that u instances of the x^{-1}-module and $(m-u-1)$ instances of the x-module are cascaded in the bit-parallel Montgomery multiplication, whereas in bit-parallel polynomial basis multiplication $(m-1)$ instances of the x-module are cascaded [25]. This results in reducing the time overhead of the concurrent error detection process in the Montgomery multiplication.

Now, we consider the time and area complexities of the concurrent error detection scheme for the second step (i.e., obtaining (27)). Obtaining (35) requires $l \times m$ two-input AND gates and l XOR-trees with m inputs. Thus, the total number of XOR gates required to obtain (35) is $l \times (m-1)$. The time complexity of computing (35) is $T_A + T_{XORm}$, where T_{XORm} represents the delay of an XOR tree with m inputs. One can notice that this time complexity is equal to the time complexity of obtaining (27). As a result, obtaining $\hat{A}^{(i)}$, $i \in [-u, m-u-1]$, accounts for the time overhead of concurrent error detection scheme in this multiplier. Example 2 in Appendix A explains the concurrent error detection in $GF(2^7)$.

6 Analysis and Simulation Results

In this section, we first consider the error detection capability, and then, the time and area overheads of the proposed concurrent error detection circuits.

6.1 Error Detection Capability

Theoretically, using modulo $(x^l + 1)$ operations to implement error detection, which is equivalent to the interlacing parity codes in $GF(2^m)$, has the error detection probability of [25]

$$\frac{2^{m+l} - 2^m}{2^{m+l}} = 1 - \frac{1}{2^l}.$$

However, to evaluate the error detection capability of the proposed scheme, we have modeled all three Montgomery multipliers using C++. We have selected the binary extension field $GF(2^{163})$ constructed by the type-II irreducible pentanomial $F(z) = z^{163} + z^{72} + z^{71} + z^{70} + 1$. It has been assumed that the faults are permanent and are injected at the inputs and outputs of the gates or flip-flops. As a result in all the simulations, for each two-input AND and XOR gate, we have considered six possible stuck-at fault situations. Also, we have considered two possible stuck-at faults for the flip-flops. Eight values of l, i.e., $l = 2$, 4, 6, 8, 10, 12, 14, and 16, have been chosen. For each of the three Montgomery multipliers considered in this paper, we have conducted four experiments and in each experiment, we have used 1,000 random values for A and B, and run the program 1,000 times. In total, each experiment has been run 1,000,000 times. At each iteration in the first experiment, we have injected one stuck-at fault at a random location. In the second and third experiments, two and three stuck-at faults have been injected at random locations, respectively. The last experiment is carried out by injecting a random number of stuck-at faults in each iteration at random locations.

The results of our simulations have been summarized in Table 1. The column titled "Error Occurred" shows the number of cases, where the fault injection has resulted in an error. Also, the column titled "Error Detected" shows how many of erroneous multiplication products have been detected. The table shows that after injecting random number of stuck-at faults using $l = 8$, in the bit-serial, digit-serial ($D = 2$), and the bit-parallel Montgomery multipliers 99.6 percent, 99.61 percent, and 99.61 percent of the errors have been detected.

6.2 Time and Area Overheads

We have summarized the theoretical complexity of the presented multipliers with error detection in Table 2. The results for the digit-serial multiplier are for the special case $D \leq l$ using a fixed $F(z)$. Also, for the bit-parallel multiplier, $F(z)$ is assumed to be fixed and Y is shown in (36).

TABLE 1
Error Detection Capability

No. of stuck-at faults	l	Error Occurred	Error Detected	Percentage	l	Error Occurred	Error Detected	Percentage
Bit-Serial Montgomery Multiplication								
One	2	833,532	624,446	74.91	4	834,018	781,552	93.70
	6	833,510	820,316	98.41	8	833,408	830,208	99.61
	10	833,406	832,547	99.89	12	834,156	834,043	99.99
	14	833,825	833,788	99.99	16	833,093	833,087	99.99
Two	2	971,805	730,049	75.12	4	971,940	910,960	93.72
	6	972,231	952,102	97.92	8	971,998	968,250	99.61
	10	971,998	968,322	99.62	12	972,0240	969,507	99.74
	14	972,210	970,228	99.80	16	971,856	970,174	99.83
Three	2	995,287	745,735	74.92	4	995,361	933,014	93.73
	6	995,287	977,263	98.18	8	995,190	991,363	99.61
	10	995,287	992,911	99.76	12	995,183	993,757	99.85
	14	995,244	994,257	99.90	16	995,249	994,369	99.91
Random	2	1,000,000	750,597	75.05	4	1,000,000	937,485	93.74
	6	1,000,000	984,405	98.44	8	1,000,000	996,041	99.60
	10	1,000,000	998,980	99.89	12	1,000,000	999,758	99.97
	14	1,000,000	999,945	99.99	16	1,000,000	999,983	99.99
Digit-Serial Montgomery Multiplication								
One	2	801,106	430,709	46.23	4	801,071	617,556	77.09
	6	801,106	709,635	88.58	8	801,329	756,513	94.40
	10	801,106	779,818	97.34	12	800,857	789,178	98.54
	14	801,535	796,377	99.35	16	801,466	798,437	99.62
Two	2	960,158	521,711	54.33	4	960,345	783,816	81.61
	6	960,097	873,680	90.99	8	960,100	917,279	95.53
	10	960,097	938,094	97.70	12	960,033	947,848	98.73
	14	960,509	953,903	99.31	16	960,445	956,598	99.59
Three	2	991,833	601,442	60.63	4	991,904	850,098	85.70
	6	992,134	928,674	93.60	8	991,936	961,735	96.95
	10	991,964	976,927	98.48	12	992,036	984,199	99.21
	14	992,073	987,655	99.55	16	991,989	989,562	99.75
Random	2	1,000,000	749,313	74.93	4	1,000,000	937,583	93.75
	6	1,000,000	984,163	98.41	8	1,000,000	996,172	99.61
	10	1,000,000	998,961	99.89	12	1,000,000	999,745	99.97
	14	1,000,000	999,938	99.99	16	1,000,000	999,990	99.99
Bit-Parallel Montgomery Multiplication								
One	2	365,256	363,533	99.52	4	364,340	362,919	99.60
	6	364,340	362,902	99.60	8	364,349	362,984	99.62
	10	364,910	364,903	99.99	12	363,936	363,935	99.99
	14	365,033	365,032	99.99	16	364,914	364,914	100
Two	2	595,480	529,053	88.84	4	595,826	564,350	94.71
	6	595,480	574,907	96.54	8	596,372	593,863	99.57
	10	595,423	595,182	99.95	12	595,893	595,794	99.98
	14	595,312	595,263	99.99	16	596,145	596,099	99.99
Three	2	742,037	616,635	83.10	4	741,999	681,326	91.82
	6	741,703	702,680	94.73	8	742,037	739,860	99.70
	10	742,801	741.389	99.80	12	741,874	740,983	99.87
	14	742,526	742,303	99.96	16	741,795	741,743	99.99
Random	2	1,000,000	750,472	75.04	4	1,000,000	937565	93.75
	6	1,000,000	984,395	99.43	8	1,000,000	996,102	99.61
	10	1,000,000	999,048	99.90	12	1,000,000	999,765	99.97
	14	1,000,000	999,924	99.99	16	1,000,000	999,986	99.99

To find the practical overheads, we have described the bit-serial, digit-serial, and bit-parallel Montgomery multipliers using VHDL and implemented it on $0.18\ \mu$m CMOS ASIC technology using the Synopsys Design Analyzer. The Map Effort was set to medium and the type-II irreducible pentanomial $F(z) = z^{163} + z^{72} + z^{71} + z^{70} + 1$ is used for all the implementations.

For the bit-serial multiplier, the implementations have been done with a target clock period of 3.0 ns and the results are obtained for the original multiplier and the multiplier with error detection capability. We have selected 15 values for l, i.e., $2 \leq l \leq 16$, and obtained the area overheads as depicted in Fig. 7. Note that this figure also includes the area overheads of the final modulo $(x^l + 1)$ operation and comparison shown in Fig. 1.

Similarly, we have implemented the digit-serial Montgomery multiplier on ASIC with a target clock period of 10.0 ns. The multiplier has been implemented with two digit sizes, i.e., $D = 2$ and $D = 8$, and the concurrent error detection circuit has been implemented with $2 \leq l \leq 16$. The time and area overheads are shown in Figs. 8 and 9, respectively. Note that Fig. 9 includes the area overheads of the final modulo $(x^l + 1)$ operation and comparison shown in Fig. 1. However, the time overheads of these modules are negligible in comparison to the time complexity of the digit-serial Montgomery multiplier and therefore, Fig. 8 shows the time overhead in critical path delay of the multiplier.

Theoretically, the hardware implementation of the bit-parallel Montgomery multiplier requires 162×3 two-input

TABLE 2
Theoretical Area and Time Complexities

		Bit-serial	Digit-serial	Bit-parallel
Multiplier	XOR	$2m-1$	$D(2m-k)+m-1$	$(m-1)(m+\omega-2)$
	AND	$2m-1$	$D \times m$	m^2
	FFs	$2m$	$2m$	–
	Delay	$2T_A+2T_X$	$T_A+2\lceil log_2(D+1)\rceil T_X$	$\cong T_A+(\omega+\lceil log_2 m\rceil)T_X$
CED	XOR	$2l$	$l(2D+1)$	$u \cdot \lambda+(m-u-1)\cdot\sigma+l\cdot(m-1)$
	AND	$2l$	$D \times l$	$m \cdot l$
	FFs	$2l$	$2l$	–
	Delay	$2T_A+2T_X$	$T_A+(\lceil log_2(D+1)\rceil+\lceil log_2(D+2)\rceil)T_X$	$T_A+(Y+\lceil log_2 m\rceil)T_X$

XOR gates to obtain the matrix $\mathbf{M}$. Also, obtaining (27) requires 163^2 two-input AND gates and 163×162 two-input XOR gates. In total, this multiplier requires $163^2 = 26{,}569$ AND gates and $166 \times 162 = 26{,}892$ XOR gates. To investigate the theoretical overhead of the concurrent error detection process, we choose $1 < l \leq 16$, and provide the number of the required gates in Table 3. The number of the required AND gates is computed from (35) and the number of the XOR gates is based on (35) and Lemmas 2 and 5.

To have a better evaluation of the proposed concurrent error detection approach for the bit-parallel Montgomery multiplier, it has been implemented on ASIC with a target clock period of 30.0 ns using $F(z) = z^{163} + z^{72} + z^{71} + z^{70} + 1$ as the irreducible polynomial. Note that the ASIC implementations also include the final modulo $(x^l + 1)$ operation and comparison modules shown in Fig. 1. The ASIC implementation results for the area overhead are shown in Fig. 10. One can observe that the greatest area overhead occurs when $l = 16$ and is equal to 11.32 percent. The time overhead of the concurrent error detection scheme has been depicted in Fig. 11 for $1 < l \leq 16$. It is interesting to note that some values of l (e.g., $l = 4$ and 7) result in a very low time overhead. The time overhead for $l = 8$ is 27.80 percent.

The concurrent error detection schemes for the systolic Montgomery multiplication have been considered in [30] and [31]. These schemes are based on time-redundancy which detect all the single cells faults. However, the technique is only applicable on pipelined multipliers. The parity-based approach used in [31] is for the bit-serial Montgomery multiplier and can detect approximately 50 percent of the errors. The schemes proposed in this paper are for unpipelined Montgomery multipliers and offer a very high error detection capability as compared to [31].

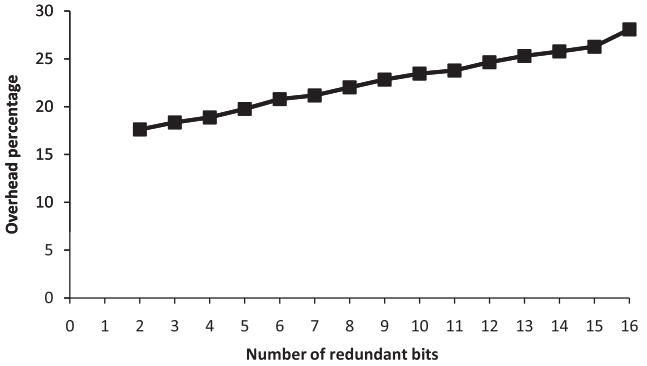

Fig. 7. ASIC implementation results for the area overhead in bit-serial Montgomery multiplication.

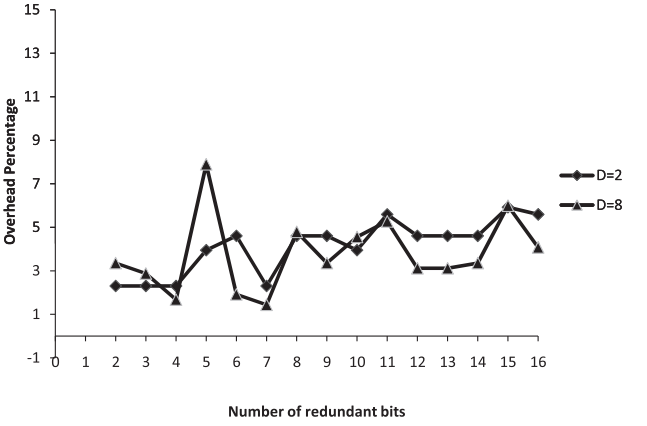

Fig. 8. ASIC implementation results for the time overhead in digit-serial Montgomery multiplication.

7 Conclusions

In this paper, we have considered concurrent error detection for the Montgomery multiplication over binary extension fields. Three different multipliers, namely the bit-serial, digit-serial, and bit-parallel multipliers, have been considered and the concurrent error detection scheme has been derived and implemented for each of them.

The time and area overheads of the proposed schemes have been reported and ASIC implementation have been done to confirm the theoretical overheads. The results show that the proposed schemes result in small time and area overheads. Furthermore, our software simulations have shown that the proposed concurrent error detection has a significant error detection capability.

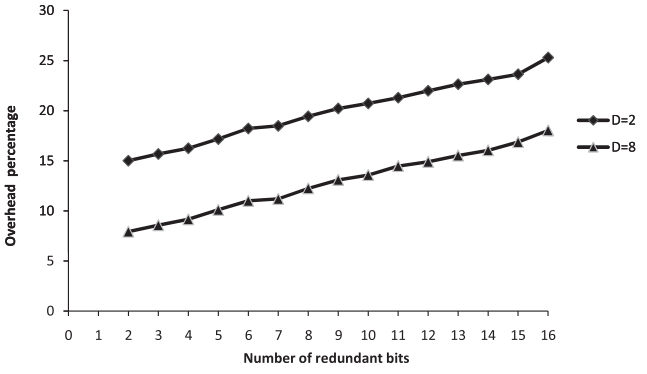

Fig. 9. ASIC implementation results for the area overhead in digit-serial Montgomery multiplication.

TABLE 3
Theoretical Time and Area Overheads in the Bit-Parallel Montgomery Multiplication

	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$	$l=7$	$l=8$	$l=9$	$l=10$	$l=11$	$l=12$	$l=13$	$l=14$	$l=15$	$l=16$
#AND	326	489	652	815	978	1141	1304	1467	1630	1793	1956	2119	2282	2445	2608
#XOR	325	648	810	1296	1458	1296	1782	1944	2106	2592	2522	2592	2754	3240	3402

APPENDIX

Example 1. Let $m=11$, $D=k=l=4$, $C'=\sum_{i=0}^{13} c'_i x^i$, and $\hat{C}'=C' \bmod (x^4+1)=\hat{c}'_3x^3+\hat{c}'_2x^2+\hat{c}'_1x+\hat{c}'_0$. In this case, $\hat{c}'_0=c'_0+c'_4+c'_8+c'_{12}$, $\hat{c}'_1=c'_1+c'_5+c'_9+c'_{13}$, $\hat{c}'_2=c'_2+c'_6+c'_{10}$, and $\hat{c}'_3=c'_3+c'_7+c'_{11}$. We assume that $\hat{A}=\hat{a}_3x^3+\hat{a}_2x^2+\hat{a}_1x^1+\hat{a}_0$. For this example, the hardware realization of (19) has been depicted on the left-hand side of Fig. 12a using gray cells and gates. The right-hand side of this figure implements Step 3 of Algorithm 5. The dashed line labeled with XOR shows that all the cells in the same column should be added together using XOR trees in order to obtain C' and $\hat{C}'$. Now,

$$C' \cdot x^{-4}=c'_{13}x^9+\cdots+c'_4+c'_3x^{-1}+\cdots+c'_0x^{-4},$$

and $C' \cdot x^{-4} \bmod F(x)$ can be written as

$$c'_{13}x^9+\cdots+c'_4+c'_3F(x)\cdot x^{-1}+\cdots+c'_0F(x)\cdot x^{-4}. \quad (37)$$

Since $k=4$, $F(x)\cdot x^{-1}$ to $F(x)\cdot x^{-4}$ are obtained by right-shifting the coordinates of $F(x)$. Note that $F(x)\cdot x^{-1}$ is named $F'(x)$. The implementation of this operation is shown in Fig. 12b using the cells and gates drawn in white.

Now, one can use (37) to obtain $(C=C'\cdot x^{-D} \bmod F(x)) \bmod (x^4+1)$. The results are as follows:

$$\begin{aligned}
\hat{c}_0&=c'_4+c'_8+c'_{12}+c'_3\hat{f}'_0+c'_2\hat{f}'_1+c'_1\hat{f}'_2+c'_0\hat{f}'_3,\\
\hat{c}_1&=c'_5+c'_9+c'_{13}+c'_3\hat{f}'_1+c'_2\hat{f}'_2+c'_1\hat{f}'_3+c'_0\hat{f}'_0,\\
\hat{c}_2&=c'_6+c'_{10}+c'_3\hat{f}'_2+c'_2\hat{f}'_3+c'_1\hat{f}'_0+c'_0\hat{f}'_1, \quad \text{and}\\
\hat{c}_3&=c'_7+c'_{11}+c'_3\hat{f}'_3+c'_2\hat{f}'_0+c'_1\hat{f}'_1+c'_0\hat{f}'_2.
\end{aligned}$$

Using the coordinates of $\hat{C}'$ defined in this example, the coordinates of $\hat{C}$ can be rewritten as

$$\begin{aligned}
\hat{c}_0&=\hat{c}'_0+c'_0+c'_3\hat{f}'_0+c'_2\hat{f}'_1+c'_1\hat{f}'_2+c'_0\hat{f}'_3,\\
\hat{c}_1&=\hat{c}'_1+c'_1+c'_3\hat{f}'_1+c'_2\hat{f}'_2+c'_1\hat{f}'_3+c'_0\hat{f}'_0,\\
\hat{c}_2&=\hat{c}'_2+c'_2+c'_3\hat{f}'_2+c'_2\hat{f}'_3+c'_1\hat{f}'_0+c'_0\hat{f}'_1, \quad \text{and}\\
\hat{c}_3&=\hat{c}'_3+c'_3+c'_3\hat{f}'_3+c'_2\hat{f}'_0+c'_1\hat{f}'_1+c'_0\hat{f}'_2.
\end{aligned}$$

Fig. 12b shows the x^{-D} module with the concurrent error detection scheme. As mentioned previously, the white cells and gates implement the normal function of this block (i.e., a multiplication by x^{-D} followed by a reduction by $F(x)$). The gray cells and gates in this figure implement the concurrent error detection mechanism. Note that the last row in this figure represents C and $\hat{C}=C \bmod (x^l+1)$. Each cell in this row is obtained by summing up all the cells in its corresponding column (shown by a dashed line, labeled XOR).

Example 2. We consider bit-parallel Montgomery multiplication over $GF(2^7)$ using $F(z)=z^7+z^4+z^3+z^2+1$, where $F(x)=0$. Choosing x^2 as the Montgomery factor, we have $C=A\cdot B\cdot x^{-2} \bmod F(x)$. Using the notation presented before, this can be written as

$$C=b_7\cdot A^{(4)}+\cdots+b_2\cdot A^{(0)}+\cdots+b_0\cdot A^{(-2)}.$$

The matrix $\mathbf{M}$ corresponding to this multiplication is shown in Fig. 13, using the white gates. As it can be seen from the figure, there are two x^{-1}-modules and four x-modules. Each of these modules includes three two-input XOR gates. The critical path delay of this part is $3\cdot T_X$.

To obtain the concurrent error detection circuit, two polynomials are defined as $F'(x)=x^6+x^3+x^2+x$, and $F''(x)=x^4+x^3+x^2+1$. It is assumed that $l=3$ and consequently, $F'(x) \bmod (x^3+1)=1\cdot x^2+1\cdot x+0$, which

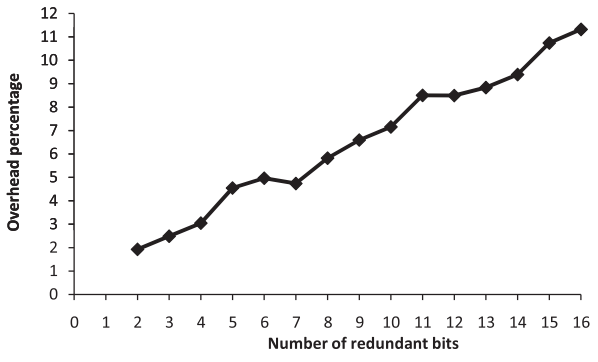

Fig. 10. ASIC implementation area overhead percentage of the concurrent error detection in bit-parallel Montgomery multiplication using $F(z)=z^{163}+z^{72}+z^{71}+z^{70}+1$.

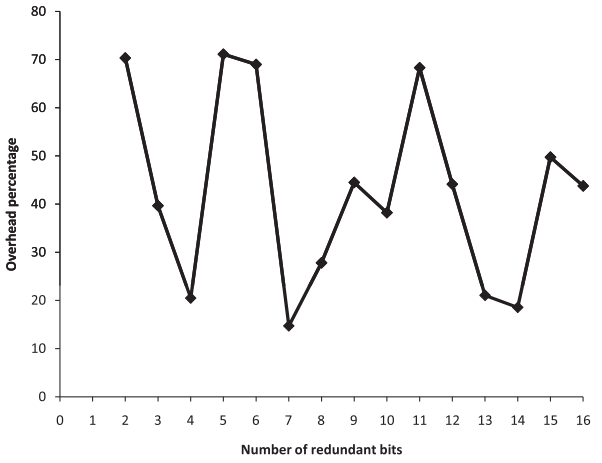

Fig. 11. ASIC implementation time overhead percentage of the concurrent error detection in bit-parallel Montgomery multiplication using $F(z)=z^{163}+z^{72}+z^{71}+z^{70}+1$.

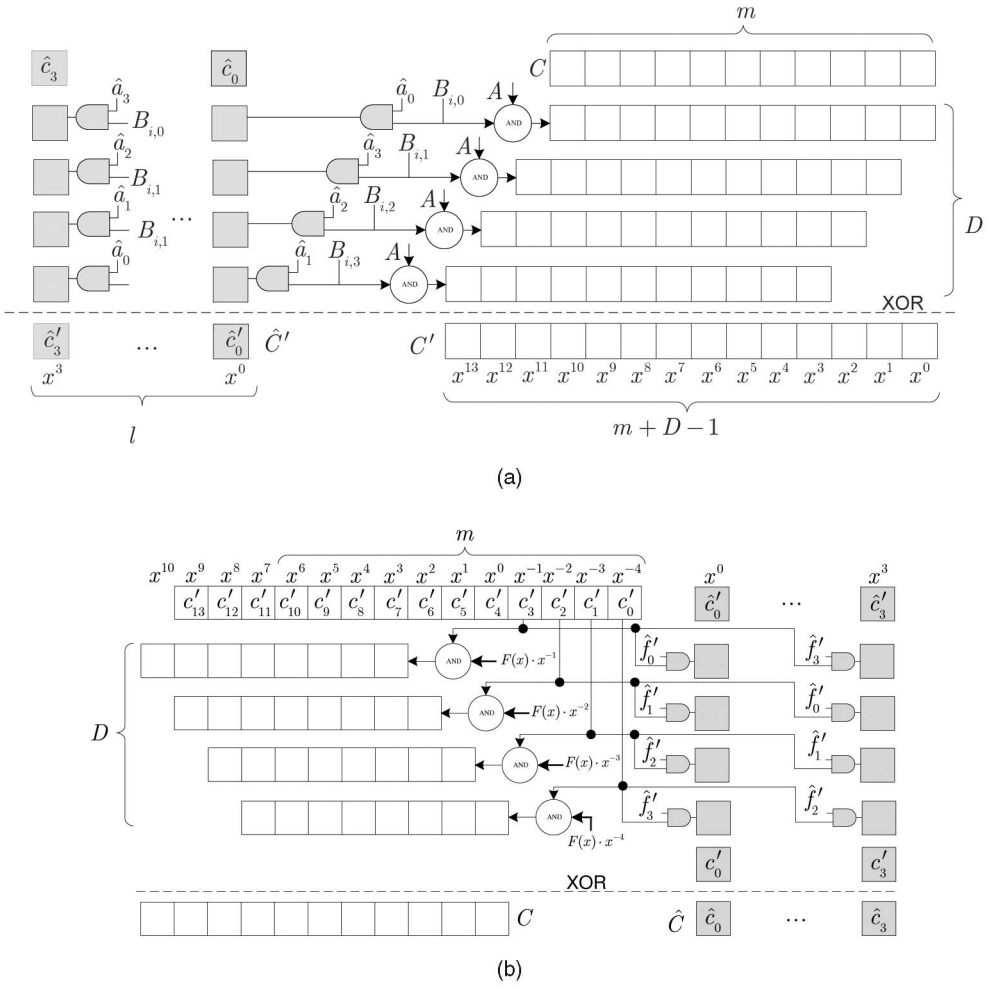

Fig. 12. Concurrent error detection for $m = 11$, $D = k = l = 4$: (a) the $\times$ module, (b) the x^{-D} module.

means $\hat{f}'_2 = 1$, $\hat{f}'_1 = 1$, and $\hat{f}'_0 = 0$. Also, $F''(x) \bmod (x^3+1) = 1 \cdot x^2 + 1 \cdot x + 0$, meaning $\hat{f}''_2 = 1$, $\hat{f}''_1 = 1$, and $\hat{f}''_0 = 0$. Next, it is assumed that $A^{(0)} \bmod (x^3+1) = A \bmod (x^3+1) = \hat{a}_2 x^2 + \hat{a}_1 x + \hat{a}_0$. Now, $A^{(i)} \bmod (x^3+1)$, $i \in [-2,-1]$, is computed based on Lemma 2 using the following recursive equation:

$$\begin{cases} \hat{a}_0^{(i)} = & \hat{a}_1^{(i-1)} \\ \hat{a}_1^{(i)} = & \hat{a}_2^{(i-1)} + a_0^{(i-1)} \\ \hat{a}_2^{(i)} = & \hat{a}_0^{(i-1)}, \end{cases} \tag{38}$$

and $A^{(i)} \bmod (x^3+1)$, $i \in [1,4]$, is obtained based on Lemma 5, using

$$\begin{cases} \hat{a}_0^{(i)} = & \hat{a}_2^{(i-1)} \\ \hat{a}_1^{(i)} = & \hat{a}_0^{(i-1)} \\ \hat{a}_2^{(i)} = & \hat{a}_1^{(i-1)} + a_6^{(i-1)}. \end{cases} \tag{39}$$

Using (38) and (39), the concurrent error detection for step one has been depicted in Fig. 13 using gray cells. It can be seen from the figure that obtaining $A^{(i)} \bmod (x^3+1)$ for $i \in [-2,4]$ requires six two-input XOR gates in total. The longest path in the concurrent error detection includes two XOR gates, and as a result, its delay is $2 \cdot T_X$.

The error detection for the second part is straightforward and is obtained using (35) as follows:

$$\hat{c}_i = \sum_{j=0}^{6} b_j \cdot \hat{a}_i^{(j-2)}.$$

This requires 21 two-input AND gates and 18 two-input XOR gates and has the critical path delay of $T_A + 3 \cdot T_X$. Obtaining C in (35) itself requires 49 two-input AND gates and 42 two-input XOR gates and has the delay of $T_A + 3 \cdot T_X$. Now, it can be concluded that concurrent error detection in

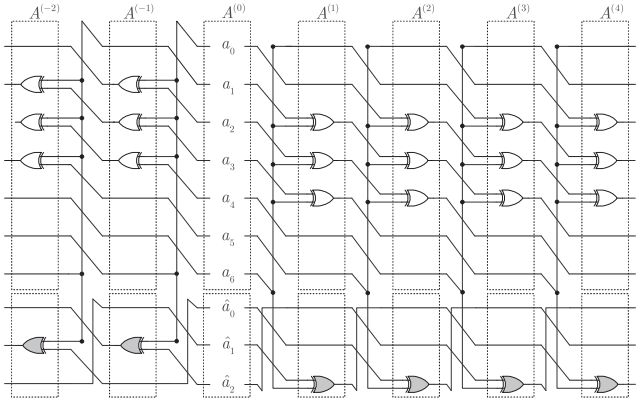

Fig. 13. Obtaining the matrix $\mathbf{M}$ with concurrent error detection for $F(z) = z^7 + z^4 + z^3 + z^2 + 1$ using $u = 2$.

this example (excluding the final modulo $(x^l + 1)$ operation and comparison) does not impose any time overhead and requires 42 percent more AND gates and 40 percent more XOR gate. It is noted that in the general case, the time and area overheads depend on the parameters m, l, u, and $F(z)$.

ACKNOWLEDGMENTS

The authors thank the reviewers for their constructive comments. This work was supported in part by an NSERC Discovery grant awarded to Arash Reyhani-Masoleh.

REFERENCES

[1] V.S. Miller, "Use of Elliptic Curves in Cryptography," *Proc. Advances in Cryptology-CRYPTO 85,* pp. 417-426, 1986.
[2] N. Koblitz, "Elliptic Curve Cryptosystems," *Math. of Computation,* vol. 48, no. 177, pp. 203-209, 1987.
[3] E.D. Mastrovito, *VLSI Architectures for Computation in Galois Fields*, PhD thesis, Linkoping Univ., 1991.
[4] L. Song and K. Parhi, "Low-Energy Digit-Serial/Parallel Finite Field Multipliers," *The J. VLSI Signal Processing,* vol. 19, no. 2, pp. 149-166, 1998.
[5] A. Reyhani-Masoleh and M. Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$," *IEEE Trans. Computers,* vol. 53, no. 8, pp. 945-959, Aug. 2004.
[6] F. Rodriguez-Henriguez and C. Koc, "Parallel Multipliers Based on Special Irreducible Pentanomials," *IEEE Trans. Computers,* vol. 52, no. 12, pp. 1535-1542, Dec. 2003.
[7] C. Koc and T. Acar, "Montgomery Multiplication in $GF(2^k)$," *Designs, Codes and Cryptography,* vol. 14, no. 1, pp. 57-69, 1998.
[8] H. Fan and Y. Dai, "Fast Bit-Parallel $GF(2^n)$ Multiplier for All Trinomials," *IEEE Trans. Computers,* vol. 54, no. 4, pp. 485-490, Apr. 2005.
[9] J. Massey and J. Omura, "Computational Method and Apparatus for Finite Field Arithmetic," US Patent 4,587,627. 1986.
[10] H. Wu, M. Hasan, and I. 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.
[11] T. Beth and D. Gollman, "Algorithm Engineering for Public Key Algorithms," *IEEE J. Selected Areas in Communications,* vol. 7, no. 4, pp. 458-466, May 1989.
[12] P. Montgomery, "Modular Multiplication without Trial Division," *Math. of Computation,* vol. 44, no. 170, pp. 519-521, 1985.
[13] H. Wu, "Montgomery Multiplier and Squarer for a Class of Finite Fields," *IEEE Trans. Computers,* vol. 51, no. 5, pp. 521-529, May 2002.
[14] L. Batina, N. Mentens, B. Preneel, and I. Verbauwhede, "Balanced Point Operations for Side-Channel Protection of Elliptic Curve Cryptography," *IEEE Proc. Information Security,* vol. 152, no. 1, pp. 57-65, Oct. 2005.
[15] A. Hariri and A. Reyhani-Masoleh, "Bit-Serial and Bit-Parallel Montgomery Multiplication and Squaring over $GF(2^m)$," *IEEE Trans. Computers,* vol. 58, no. 10, pp. 1332-1345, Oct. 2009.
[16] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "High-Performance Public-Key Cryptoprocessor for Wireless Mobile Applications," *Mobile Networks and Applications,* vol. 12, no. 4, pp. 245-258, 2007.
[17] S. Mitra and E. McCluskey, "Which Concurrent Error Detection Scheme to Choose?" *Proc. Int'l Test Conf.,* pp. 985-994, 2000.
[18] I. Koren and C.M. Krishna, *Fault-Tolerant Systems.* Morgan Kaufman, 2007.
[19] *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing,* M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, eds., IEEE Press, 1986.
[20] C. Giraud and H. Thiebeauld, "A Survey on Fault Attacks," *Proc. Smart Card Research and Advanced Applications VI,* pp. 159-176, 2004.
[21] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," *Proc. IEEE,* vol. 94, no. 2, pp. 370-382, Feb. 2006.
[22] S. Fenn, M. Gossel, M. Benaissa, and D. Taylor, "On-Line Error Detection for Bit-Serial Multipliers in $GF(2^m)$," *J. Electronic Testing: Theory and Applications,* vol. 13, no. 1, pp. 29-40, 1998.
[23] A. Reyhani-Masoleh and M. Hasan, "Fault Detection Architectures for Field Multiplication Using Polynomial Bases," *IEEE Trans. Computers,* vol. 55, no. 9, pp. 1089-1103, Sept. 2006.
[24] S. Bayat-Sarmadi and M. Hasan, "On Concurrent Detection of Errors in Polynomial Basis Multiplication," *IEEE Trans. Very Large Scale Integration Systems,* vol. 15, no. 4, pp. 413-426, Apr. 2007.
[25] W. Chelton and M. Benaissa, "Concurrent Error Detection in $GF(2^m)$ Multiplication and Its Application in Elliptic Curve Cryptography," *IET Circuits, Devices and Systems,* vol. 2, no. 3, pp. 289-297, 2008.
[26] S. Bayat-Sarmadi and M. Hasan, "Concurrent Error Detection in Finite Field Arithmetic Operations Using Pipelined and Systolic Architectures," *IEEE Trans. Computers,* vol. 58, no. 11, pp. 1553-1567, Nov. 2009.
[27] C.W. Chiou, C.-C. Chang, C.-Y. Lee, T.-W. Hou, and J.-M. Lin, "Concurrent Error Detection and Correction in Gaussian Normal Basis Multiplier Over $GF(2^m)$," *IEEE Trans. Computers,* vol. 58, no. 6, pp. 851-857, June 2009.
[28] C.-Y. Lee, C.W. Chiou, and J.-M. Lin, "Concurrent Error Detection in a Bit-Parallel Systolic Multiplier for Dual Basis of $GF(2^m)$," *J. Electronic Testing: Theory and Applications,* vol. 21, no. 5, pp. 539-549, 2005.
[29] C.-Y. Lee, C.W. Chiou, and J.-M. Lin, "Concurrent Error Detection in a Polynomial Basis Multiplier over $GF(2^m)$," *J. Electronic Testing: Theory and Applications,* vol. 22, no. 2, pp. 143-150, 2006.
[30] C.W. Chiou, C.Y. Lee, A.W. Deng, and J.M. Lin, "Concurrent Error Detection in Montgomery Multiplication over $GF(2^m)$," *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences,* vol. E89-A, no. 2, pp. 566-574, 2006.
[31] A. Hariri and A. Reyhani-Masoleh, "Fault Detection Structures for the Montgomery Multiplication over Binary Extension Fields," *Proc. Workshop Fault Diagnosis and Tolerance in Cryptography (FDTC),* pp. 37-46, 2007.
[32] Recommended Elliptic Curves for Fed. Gov. Use http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf, 2009.
[33] A. Hariri and A. Reyhani-Masoleh, "Digit-Serial Structures for the Shifted Polynomial Basis Multiplication over Binary Extension Fields," *Proc. Second Int'l Workshop Arithmetic of Finite Fields (WAIFI),* pp. 103-116, 2008.

Arash Hariri received the BSc degree from Amirkabir University of Technology, Iran, in 2003, and the MSc degree from Shahid Beheshti University, Iran, in 2006, both in computer engineering. He is currently working toward the PhD degree in electrical and computer engineering from The University of Western Ontario, Canada. His current research interests include computer arithmetic, cryptography, and fault tolerance. He is a student member of the IEEE.

Arash Reyhani-Masoleh received the BSc degree in electrical and electronic engineering from Iran University of Science and Technology in 1989, and the MSc degree in electrical and electronic engineering from the University of Tehran in 1991, both with the first rank, and the PhD degree in electrical and computer engineering from the University of Waterloo in 2001. From 1991 to 1997, he was with the Department of Electrical Engineering, Iran University of Science and Technology. From June 2001 to September 2004, he was with the Centre for Applied Cryptographic Research, University of Waterloo, where he was awarded a Natural Sciences and Engineering Research Council of Canada (NSERC) postdoctoral fellowship in 2002. In October 2004, he joined the Department of Electrical and Computer Engineering, University of Western Ontario, London, Canada, where he is currently a tenured associate professor. He has been awarded an NSERC Discovery Accelerator Supplement (DAS) in 2010. Currently, he serves as an associate editor for Integration, the *VLSI Journal* (Elsevier). His current research interests include algorithms and VLSI architectures for computations in finite fields, fault-tolerant computing, and error-control coding. He is a member of the IEEE and the IEEE Computer Society.