Transition Faults Detection in Bit Parallel Multipliers over $GF(2^m)$

Hafizur Rahaman
Bengal Engineering & Science University, Shibpur
Howrah-711103, India
rahaman_h@it.becs.ac.in

Jimson Mathew
Computer Science Department, University of Bristol
BS81UB, UK,
jimson@cs.bris.ac.uk

Ashutosh K. Singh
CS Dept., School of Engineering,
Curtin University of Technology, Malaysia
ashutosh.s@curtin.edu.my

Dhiraj K. Pradhan, IEEE/ACM Fellow
Computer Science Department, University of Bristol
BS81UB, UK,
pradhan@cs.bris.ac.uk

Biplab K. Sikdar
Bengal Engineering & Science University, Shibpur
Howrah-711103, India
biplab@cs.becs.ac.in

Abstract: - In this article, a C-testable design for detecting transition faults in the polynomial basis (PB) bit parallel (BP) multiplier circuits over $GF(2^m)$ is discussed. For 100 percent transition fault coverage, the proposed technique requires only 10 vectors, irrespective of multiplier size, at the cost of 6 percent extra hardware. The proposed constant test vectors which are sufficient to detect both the transition and stuck-at faults in the multiplier circuits can be derived directly without any requirement of an ATPG tool. As the $GF(2^m)$ multipliers have found critical applications in public key cryptography and need secure internal testing, a Built-in Self-Test (BIST) circuit may be used for generating test patterns internally. This will obviate the need of having three extra pins for the control inputs and also provides public-key security in cryptography. Area and delay of the testable circuit are analyzed using Synopsys® tools with 0.18µ CMOS technology library from UMC.

Key-Words: - Transition fault, Galois field, multiplier, cryptography, error control code, VLSI testing.

1 Introduction

Failures that cause logic circuits to malfunction at the desired clock rate and violate timing specifications are modeled as delay faults. Gate delay [9-10] and path delay fault model [11-12] have generally been used to model delay defects. Path delay fault model is more comprehensive in modeling delay defects, but often difficult to use in practice due to large number of paths in large circuits. Gate delay fault model is more practical for large circuits. The most commonly used gate delay fault model is transition fault model [9] which is considered as a logical model for a defect that delays a rising or falling transition at inputs and outputs of logic gates. There are two kinds of transition faults: *slow-to-rise* and *slow-to-fall. S*low-to-rise (fall) transition fault temporarily behaves like

a stuck-at-0 (1) fault. Testing of such faults requires two-pattern tests, each consisting of an initialization and a test vector. Several works have been reported on generating tests for transition faults [9-15]. Here, we present a C-testable technique for detecting transition faults in PB multipliers over $GF(2^m)$.

Arithmetic operations in *finite* or *Galois fields* of form $GF(2^m)$ have gained wide spread uses in public-key cryptography, error detecting and correcting code [2], VLSI testing [1], digital signal processing [17]. There are two basic arithmetic operations over finite fields: addition and multiplication. While addition over $GF(2^m)$ can be implemented with just m 2-input EXOR gates, multiplication is much more complex. Note that other operations like exponentiation, division, and inversion over $GF(2^m)$ can be performed by repeated multiplications [1-5]. Various techniques exist for optimal design of multipliers over $GF(2^m)$ with respect to complexities, delay, and power. Most techniques focused on VLSI implementation and synthesis of these multipliers because VLSI implementations of these circuits are very complicated due to complex routing, non-modular architecture and low testability. The C-testable systolic array design for Galois-Field inversion was proposed in [16], which requires 32 vectors to detect the faults in the circuit and extra hardware overhead to achieve C-testability. A testable implementation of BP multiplier over $GF(2^m)$ which requires a function independent test set of length $(2m+4)$ or detecting stuck-at faults in multipliers over $GF(2^m)$ was reported in [8]. To date, testability issue of *GF* multiplier has not fully been explored. In view of this fact, we present a C-testable design of PB multipliers over $GF(2^m)$. This design requires only 10 constant vectors to detect transitions faults as well as all the single stuck-at faults. This test length is lower than that generated by Synopsys ATPG tools (*Tetramax*^{TM}). Finally, we analyse the area, delay, and power using 0.18μ CMOS library from UMC.

2. Preliminaries

Let $GF(N)$ denote a set of N elements, where N is a power of a prime number, with two special elements 0 and 1 representing the additive and multiplicative identities respectively, and two operators addition '+' and multiplication '.'. $GF(N)$ defines a finite field, if it forms a *commutative ring* with identity over these two operators in which every element has a multiplicative inverse. Finite fields can be generated with primitive polynomials of the form $p(x) = x^m + \sum_{i=0}^{m-1} c_i x^i$, where $c_i \in GF(2)$ [5].

It is conventional to represent the elements of $GF(2^m)$ as a power of the *primitive element* α where α is the root of $P(x)$, i.e. $p(\alpha)=0$. The set $\{1,\alpha,\ldots,\alpha^{m-1}\}$ is referred to as the polynomial basis. Each element $A \in GF(2^m)$ can be expressed with respect to the PB as a polynomial of degree $(m-1)$ over $GF(2)$, i.e. $A(x) = \sum_{i=0}^{m-1} a_i x^i$ where $a_i \in GF(2)$. Given A, $B \in GF(2^m)$, the PB multiplication over $GF(2^m)$ can be defined as $C(x)=A(x).B(x) \bmod P(x)$. Details can be found in [2,6].

Mastrovito has proposed an algorithm along with its hardware architecture for PB multiplication [5] popularly known as the Mastrovito algorithm/multiplier. A formulation for Polynomial basis multiplication and generalized bit-parallel hardware architecture for special reduction polynomials, namely: trinomials, equally spaced polynomials (ESPs), and two classes of pentanomials has been presented in [4]. This formulation is described below. Consider a multiplier with A and B inputs where $A = [a_0, a_1, a_2, \ldots, a_{m-1}]$ and $B = [b_0, b_1, b_2, \ldots, b_{m-1}]$. The a_i, and b_i are the coordinates of A and B respectively where $0 \le i \le m-1$. The multiplication outputs are given in the equation (1).

$$c = d + Q^T e \tag{1}$$

$$d = L \times b \tag{2}$$

$$e = U \times b \tag{3}$$

where $b = B^T = [b_0, b_1, b_2, \ldots, b_{m-1}]^T$, and

$$L = \begin{bmatrix} a_0 & 0 & 0 & 0 \ldots 0 \\ a_1 & a_0 & 0 & 0 \ldots 0 \\ a_2 & a_1 & \ddots & a_0 \quad 0 \ldots 0 \\ \vdots & & \ddots & \ddots \quad \vdots \\ a_{m-2} & a_{m-3} & \ldots & a_1 \quad a_0 \quad 0 \\ a_{m-1} & a_{m-2} & \ldots & a_2 \quad a_1 \quad a_0 \end{bmatrix};$$

$$U = \begin{bmatrix} 0 & a_{m-1} & a_{m-2} & \ldots & a_2 & a_1 \\ 0 & 0 & \ddots & a_{m-1} \ldots & a_3 & a_2 \\ \vdots & & \ddots & \ddots & \vdots & \\ 0 & 0 \ldots 0 & & \ldots & a_{m-1} & a_{m-2} \\ 0 & 0 \ldots 0 & & \ldots & 0 & a_{m-1} \end{bmatrix}$$

We have derived Q matrix for the multiplier over $GF(2^4)$ from the first principles in example 1. The architecture for this implementation is shown in Fig. 1. This structure is divided into two parts: the *Inner Product (IP)-network* and *Q-network*. Q-network part is also EXOR tree. The IP-network, which has m blocks, generates d and e. For $\{0 \le i \le m-2\}$, each block constitutes two inner product cells, namely, $IP(i+1)$ and $IP(m-i-1)$. However the last block constitutes only one such cell $IP(m)$. The Q-network takes d and e as inputs and generates c. It constitutes m binary trees of EXOR gates ($BTX_0, BTX_1, \ldots, BTX_{m-1}$). The inputs d and e feed to the BTX trees. For the multiplier structure shown in Fig. 1, the IP-network has a total of m^2 AND gates and $(m-1)^2$ EXOR gates. The maximum number of EXOR gates required for the Q-network depends on the Q-matrix. The multiplier structure is the multiple outputs Positive Polarity Reed-Muller (PPRM)-like form.

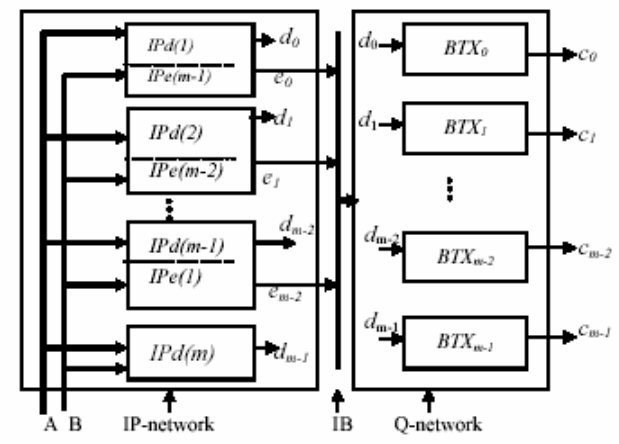

Fig. 1: Architecture of the BP multiplier over $GF(2^m)$

Definition 1: A Boolean AND-EXOR function $F(x_1, x_2, \ldots, x_n)$ is in the PPRM if only positive polarity is allowed for each input variable, i.e. each variable appears in its uncomplemented form throughout. For example $F_1 = x_1x_2 \oplus x_2x_3$ is a PPRM. Several testable techniques for AND-EXOR circuits have appeared in [7].

Example 1: A multiplier structure over *GF(16)* defined by the primitive polynomial $P(x) = x^4 + x^3 + 1$ is shown in Fig. 2.

The two inputs of the multiplier are $A = (a_0, a_1, a_2, a_3)$ and $B = (b_0, b_1, b_2, b_3)$. The polynomial representation of $GF(2^4)$ elements is as follows.
$A(x) = a_0+a_1 x+a_2 x^2 + a_3x^3$, $B(x) = b_0+b_1 x+b_2 x^2 + b_3 x^3$, where A, B $\in$ $GF(2^4)$.

The product $C(x) = A(x) \times B(x)$.

Now, $C(x) = (a_0+a_1 x+a_2 x^2 + a_3x^3) \times (b_0+b_1 x+b_2 x^2 +b_3x^3) = a_0b_0 + (a_0b_1 + a_0b_1)x+(a_0b_2 + a_1b_1+ a_2b_0)x^2 + (a_0b_3 + a_1b_2+ a_2b_1 + a_3b_0)x^3 + (a_1b_3 + a_2b_2+ a_3b_1)x^4 +(a_2b_3 + a_3b_2)x^5 + a_3b_3x^6$.

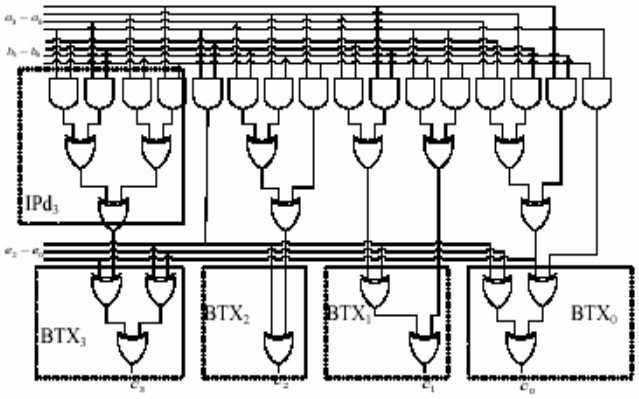

Fig.2: Architecture of the BP multiplier over $GF(2^4)$

Let us denote the lower order m coefficients as $d_0, d_1, \ldots, d_{m-1}$ and the higher order $\{m-1\}$ coefficients as $e_0, e_1, \ldots, e_{m-2}$. Then $C(x)$ can be expressed in equation (4).

$$C(x) = d_0+d_1x+d_2x^2 +d_3x^3+e_0x^4+ e_1x^5+ e_2x^6 \quad (4)$$

Here, we define product over the primitive polynomial $P(x) = x^4 + x^3 + 1$ *as* $A(x)\ B(x) \bmod P(x)$.
Hence, *we have,*
$x^4 = x^3+1$, $x^5 = x(x^3+1) = x^4 + x = x^3 + x+1$, $x^6 = x(x^5) = x(x^3 + x+1) = x^4 +x^2 + x = x^3+1+ x^2+ x = x^3+x^2+ x+1$.
Substituting the power of x^4, x^5, x^6 and simplifying we get,

$$A(x)\ B(x) \bmod P(x) = C = (e_0+e_1+e_2 +d_0)+ (e_1+e_2 +d_1)\, x+(e_2 +d_2)x^2+(e_0+e_1+e_2 +d_3)\, x^3$$

The above modulo reductions can be represented in the matrix form as given below.

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} e_0 \\ e_1 \\ e_2 \end{bmatrix} + \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

$$c = Q^T e + d \text{ , where } e = \begin{bmatrix} e_0 \\ e_1 \\ e_2 \end{bmatrix} \; d = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

We can also derive d, e from the equations (1), (2).

3 Proposed Technique

A test for a transition fault is a pair of input patterns, one known as initialisation vector to set up the initial state for the transition and another known as propagation or test vector to cause the appropriate transition and observe its effect at a primary output. The test vector is identical to a pattern that detects the corresponding stuck-at fault. The transition fault coverage is a measure of effectiveness of the delay test in detecting large delay variations. A test pair $<v_1, v_2>$ is required to detect the transition fault *f* on a signal line. The initial vector v_1 must set the target node to an initial value 0 [1] for slow-to-rise [slow-to-fall] fault. The test vector v_2 has to launch the corresponding transition at the target node and also propagate the fault effect to the primary output. Thus, v_2 is a test for s-a-0 [s-a-1] fault if the transition fault is the slow-to-rise [slow-to-fall] fault. To achieve C-testability for detection of transition faults in Bit parallel GF multipliers, the multipliers architecture as shown in Fig.1 has been augmented. The AND part of *IP-network* are modified with 3 control lines k_0, k_1 and k_2. All two inputs AND gates have been replaced by three inputs AND gates. The proposed design is shown in Fig.3.

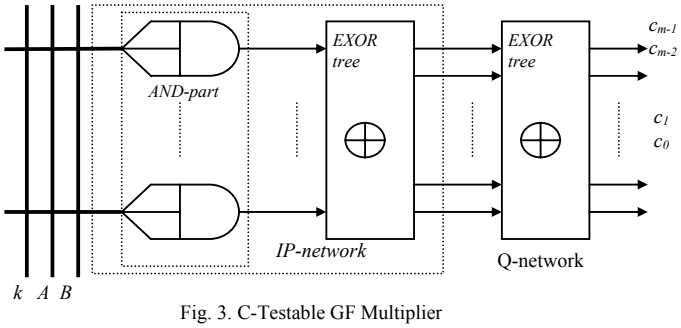

Fig. 3. C-Testable GF Multiplier

Definition 2: A circuit is C-testable if it can be tested with a constant number of vectors independent of the circuit's complexity.

For the detection of the transition faults at any input node of EXOR gate, two transitions 0→1 and 1→0 are essential. An EXOR-tree of single output can be tested for all single transition faults by *five (2m+3)*-bit function-independent tests applied to the inputs of a single-input AND-EXOR circuit. Three control inputs k_0, k_1, k_2 are used to achieve this. This scheme will allow us to apply each of the two transitions (0→1, 1→0) to the inputs of each 2-input EXOR gate in the tree. This is based on the following observation. In Fig. 5a, the two sequence q: 01100 and r: 01010 arriving at the two inputs to the last EXOR gate generate the output sequence *s*: 00110. Similarly, *q*: 01100 and *s*: 00110 arriving at the two inputs of an EXOR gate will generate the output sequence *r*: 01010. Again, input sequences *r*: 01010 and *s*: 00110 will generate q: 01100 as output. There exist the following relations among the vectors (*q, r, s*): $q \oplus r = s$, $q \oplus s = r$, $r \oplus s = q$. Applying sequence q: 01100, two transitions i.e. 0→1 and 1→0 will be achieved at any input node of EXOR gate. For the sequence r: 01010, the transitions 0→1 and 1→0 are achieved at any input node of EXOR gate. Again for sequence s: 00110, two transitions 0→1 and 1→0 at the input node are generated. Applying these sequences q, r, s, two transitions: 0→1 and 1→0 are achieved at every input node of the EXOR gate in the EXOR tree.

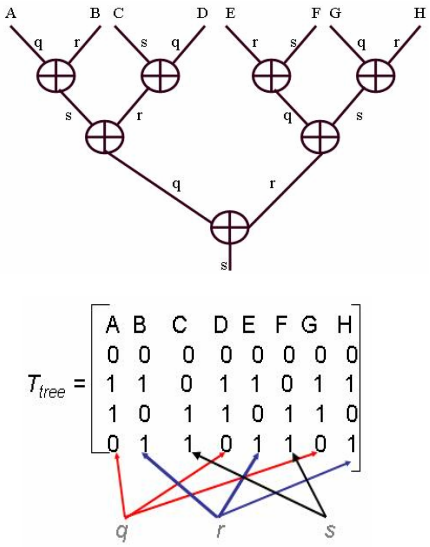

Fig. 4. Test vectors and responses in an EXOR-tree

Example 2: In the EXOR tree shown in Fig. 4, we assign sequence vectors *q, r, s, q, r, s, q, r*,... (by repeating the pattern (*q, r, s*) to the inputs of the EXOR tree from left-to right until all of them are assigned. The outputs of the first level are propagated down to the root i.e. final output of the tree. Thus, each EXOR gate in the tree receives the desired input combination from the above five combinations. The five constant test vectors that are to be applied to the inputs of the tree of Fig. 4 are shown as a matrix T_{tree}. This matrix has *five*

(constant) rows and y columns, where y is the number of leaf nodes of the tree, and is equal to the number of AND outputs (m^2) in the multiplier circuit. The columns of the matrix, if seen from *left-to-right*, will correspond to the sequence vectors: *q, r, s, q, r, s, q, r,* and so on. The number of distinct columns in the matrix is only *three* (*constant*), regardless of the size of the tree.

Since the EXOR-tree is embedded in the overall design of the single output AND-EXOR circuits, the inputs of the tree are not directly accessible. In the IP network as shown in Fig.3, each AND output feeds an EXOR input. Hence, by applying the following five vectors v_1, v_2, v_3, v_4, v_5 to the primary inputs of Fig. 3, all the three sequences q, r, s can be produced at the outputs of the AND-part.

$\{k_0\ k_1\ k_2\ a_0\ a_1 \ldots a_{m-1}\ b_0\ b_1 \ldots b_{m-1}\}$

$v_1 = \{0\ 0\ 0\ 0\ 0 \ldots 0\ 0\ 0 \ldots 0\}$

$v_2 = \{1\ 1\ 0\ 1\ 1 \ldots 1\ 1\ 1 \ldots 1\}$

$v_3 = \{1\ 0\ 1\ 1\ 1 \ldots 1\ 1\ 1 \ldots 1\}$

$v_4 = \{0\ 1\ 1\ 1\ 1 \ldots 1\ 1\ 1 \ldots 1\}$

$v_5 = \{0\ 0\ 0\ 1\ 1 \ldots 1\ 1\ 1 \ldots 1\}$

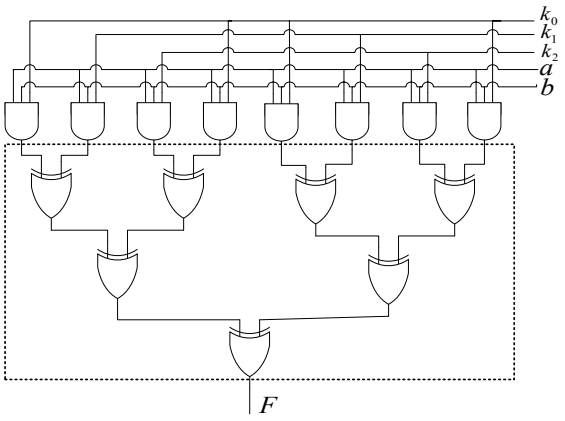

Fig. 5. EXOR-tree with a control level

A control level with three-control inputs k_0, k_1 and k_2 as shown in Fig. 5. By setting these control inputs to 1, the original function can be obtained. In this design, the AND outputs are partitioned into 3 groups based on sequence vectors *q, r, s*. The output lines of the AND gates connected with k_0, k_1 and k_2 control lines receive the sequence vector *q: 01100*, *r: 01010*, and *s: 00110* respectively. No additional hardware is essential for this testable design. Only all the two inputs AND gates in *IP-network* have been replaced by three inputs AND gates.

The technique we have discussed above is applicable to single output AND-EXOR circuits. In this section we extend this idea to multiple output AND-EXOR circuits. To achieve 100% testability in multiplier circuits, the inputs of the EXOR gates of the IP- and networks will be properly mapped. We assume that the IP-network would generate the following sequence from left-to-right: *q, r, s, q ,..., q, r, s, q...* and so on at the outputs e_j, where $0 \le j \le m$-2. To propagate these e_j outputs of the IP-network at the outputs of the Q-network, the d_i outputs, where $0 \le i \le m$, will be properly mapped with the sequences q, r, and s. The following algorithms outline this process.

Step-1: *Assignment of sequences q, r, and s to e_j, where $0 \le j \le m$-2.*

```
Algorithm_ seq_assignment _e
    for (j= 2; j<= m; j++)
        {
          e(m-j)      = q;
          e(m-(j+1)) = r;
          e(m-(j+2))  = s;
        }
```

Example 3: For the multiplier circuit over GF(2^4) of Fig.2, the e_j ($0 \le j \le 2$) are assigned as $e_2 = q$, $e_1 = r$, $e_0 = s$.

Step-2: *Assignment of the sequences q, r, and s to d_i, ($0 \le i \le m$-1).*

Condition 1: After assigning the sequences at the e_i nodes in step-1, the sequences at d_i nodes ($0 \le i \le m$-1) are to be assigned in such a way that no two input nodes of each EXOR gate receive same sequence vector in the Q-networks.

Example 4: Consider tree representation of BTX_3 block of Q-network as shown in Fig. 6a. In step-1, the nodes e_2, e_1, e_0 are already assigned with q, r, and s respectively. In Fig 6a, e_2 and e_1 will generate s sequence at y_2 node. To propagate the signal value considering condition 1, at c_3 node, q (or r) is to be assigned at y_1 node. As e_0 is already assigned with s, then r (or q) is to be assigned with d_3 to generate q (or r) at y_1 node. In this way, the sequences are to be assigned to the d_i nodes, where $0 \le i \le m$-1.

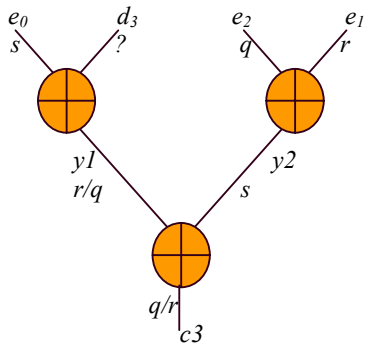

Fig.6a, Tree representation of BTX_3 block

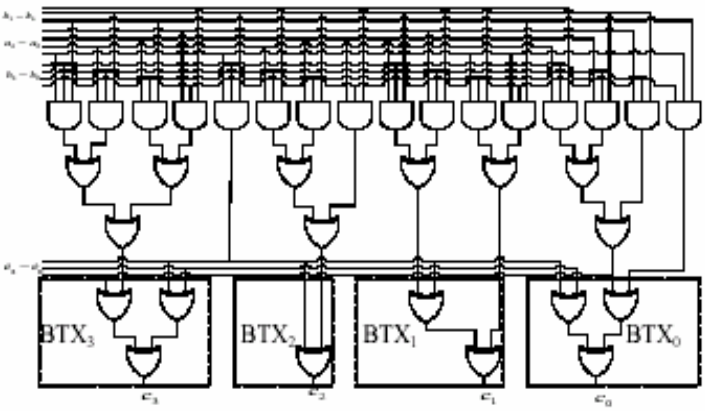

Fig. 7. Testable Design of GF(16) Multiplier

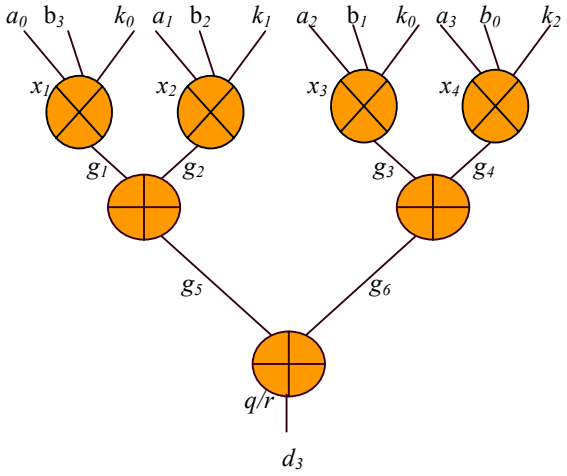

Fig.6b, Tree representation of IPd_3

Step-3: *Assignment of the sequences q, r, and s to the internal nodes of the IP-network*

Condition 2: After assigning the sequences at d_i's and e_j's, assign the input nodes of EXOR gates in IP-network with proper sequence vectors so that no two inputs of an EXOR gate receive same sequence vector.

In example 4, r (or q) is assigned to d_3 node to generate q (or r) sequence at y_1 as e_0. Similarly, considering all the other BTX blocks, *s*(or *r), r(*or *q)*, and *q*(or *r) are to be assigned to* d_2, d_1, d_0 *respectively*. After assigning e_j $(0 \le j \le 2)$ and d_i $(0 \le i \le 3)$ nodes, every EXOR gate in *IP-network* is mapped. If the test sequence v_1, v_2, v_3, v_4, v_5 is applied, then the output lines of the AND gates connected with control lines k_0, k_1 and k_2 control lines receive the sequence vector *q: 01100*, *r:01010*, and *s:00110* respectively.

Example 5: Consider IPd_3 block of IP-network of Fig. 2, as shown in the Fig.6b. The d_3 node is already assigned with either *q* or *r*. If the sequence q is assigned to d_3, then r (or s) and s (or r) will be to be assigned to g_5 and g_6 nodes respectively. If r is assigned to d_3, then q (or s) and s (or q) will be to be assigned to g_5 and g_6 nodes respectively. Suppose, we consider d_3=q, and g_5= s, g_6 = r. Again, to generate s at g_5, q and r are to be assigned to g_1, g_2 nodes respectively. Similarly, to generate r sequence at g_6 node, s and q are to be assigned to g_3, g_4 nodes respectively. Now we have g_1 = q, g_2 = r, g_3 = s, g_4 = q. To generate q sequence at g_1 node, one input of the x_1 AND gate is to be connected to k_0. Similarly, to generate r, s, q sequences at g_2, g_3, g_4 nodes respectively, one input of the x_2, x_3, x_4 AND gates is to be connected to k_1, k_2, k_0 control inputs respectively. In this way, the input nodes of the *IPd* blocks are assigned with the proper sequence vectors by selecting the proper connection of control inputs k_0, k_1, and k_2.

Example 6: The internal mapping of the interconnections in the IP- and Q-networks of Example 2 is shown in Fig.7, which is also the testable design of the multiplier designed from the primitive polynomial $P(x) = x^4 + x^3 + 1$.

Transition Fault in EXOR part: The values of q, r, s at a particular instant are shown in the table below.

Table 1: q, r, s sequence diagram

Instant	t_1	t_2	t_3	t_4	t_5
q	0	1	1	0	0
r	0	1	0	1	0
s	0	0	1	1	0

Case-1: When q and r sequence are applied at the

inputs of an EXOR gate (Table 1).

Slow-to-rise transition i.e. transition from t_1-to-t_2 at *q* input node will produce 1 instead of 0 at the output of EXOR gate because the input of r input node changes its state, but due to slow-to-rise transition at q node, the input value retains its previous state at q node and shows 1. Slow-to-rise transition i.e. transition from t_1-*to*–t_2 at r input node will produce 1 instead of 0 at the output of EXOR gate because the input of q input node changes its state, but due to slow-to-rise transition at r node, the input value retains its previous state at r node and shows 1. Slow-to-fall transition i.e. transition from t_3-to–t_4 at q input node will produce 0 instead of 1 at the output of EXOR gate. Slow-to-fall transition i.e. transition from t_2-to–t_3 at r node will produce 0 instead of 1 at the output of EXOR gate i.e., the output retains its previous value 0.

Case-2: When r and s sequence are applied at the inputs of an EXOR gate (Table 1).

Slow-to-rise transition i.e. transition from t_1-to–t_2 at *r* input node will produce 0 instead of 1 at the output of EXOR gate i.e. the output retains its previous value 0. Slow-to-rise transition i.e. transition from t_2-to–t_3 at *s* input node will produce 0 instead of 1 at the output of gate because the input value at *r* node changes its state, but due to slow–to-rise transition at *s*, the input at *s* node retains value of t_2 and output shows 0. Slow-to-fall transition i.e. transition from t_2-to–t_3 at *r* node will produce 0 instead of 1 at the output of gate because input of *s* node changes its state, but due to slow-to-fall transition at r node, the input value retains its previous state at r node and shows 0. Slow-to-fall transition i.e. transition from t_4-to–t_5 at *s* node will produce 1 instead of 0 at the output of EXOR gate because the input of r node changes its state, but due to slow–to-fall transition at s, the input of s node does not change and output shows 1.

Case-3: When q and s sequence are applied at the inputs of an EXOR gate (Table 1).

Slow-to-rise transition i.e., transition from t_1-to–t_2 at *q* input node will produce 0 instead of 1 at the output of gate i.e. the output retains previous value 0. Again, slow-to-rise transition i.e. transition from t_2-to–t_3 at *s* input node will show 1 instead of 0 at the output of the gate i.e. output retains previous output 1. Slow-to-fall transition i.e. transition from t_3-to–t_4 at *q* node will show 0 instead of 1 at the output i.e. output retains its previous value 0. Similarly, slow-to-fall transition i.e., transition from t_4-to–t_5 at s node will show 1 instead of 0 at the output i.e. the output retains it previous value 1.

Lemma 1: The vector sequence *(v_1, v_2, v_3, v_4, v_5)* will detect the transition faults in the EXOR part of the multiplier circuits.

Proof: Follows from the above discussions. □

Testability in the AND gate: The following tests will detect transition faults in a three- input AND gate with inputs a, b, k.

Table 2: input state diagram for AND gate

Instant	t_1	t_2	t_3	t_4	t_5	t_6	t_7
a	1	0	1	1	1	1	1
b	1	1	1	0	1	1	1
k	1	1	1	1	1	0	1

If the vector pair (a, b, k) = (111, 011) is applied, first vector 111 initialises AND output at logic value 1. When a slow–to-fall transition i.e. transition from t_1–to-t_2 occurs at 'a' input node, test vector 011 will show 1 at the output instead of 0, i.e. output retains previous value. Again, if the vector pair (011, 111) is applied, first vector 011 initialises the AND output at 0. When a slow–to-rise transition i.e. transition from t_2–to-t_3 occurs at 'a' input node, test vector 111 will show 0 instead of 1 at output i.e. the output retains 0. Similarly, vector sequences (111, 101,111) and (111, 110, 111) will detect transition fault at 'b, and 'k' input nodes respectively.

Lemma 2: The vector sequences v_2, v_3, v_4, v_5,v_8, v_6, v_8, v_7, v_8 will detect the transition faults in the AND part of the multiplier circuits where,

$$\begin{aligned} &\{k_0\ k_1\ k_2\ \ a_1\ \ a_2 \ldots a_{m-1}\ \ b_1\ \ b_2 \ldots b_{m-1} \\ v_6 &= \{1\ \ 1\ \ 1\ \ \ 0\ldots 0\ \ \ldots 0\ \ \ \ 1\ \ \ 1 \ldots 1\} \\ v_7 &= \{1\ \ 1\ \ 1\ \ \ 1\ldots 1 \ldots\ 1\ \ \ \ 0\ \ \ 0 \ldots 0\} \\ v_8 &= \{1\ \ 1\ \ 1\ \ \ 1\ldots 1\ldots\ \ 1\ \ \ \ 1\ \ \ 1 \ldots 1\} \end{aligned}$$

Proof: The vector pair (v_2, v_3) generates (011, 111) at the three inputs of AND gates connected with control input k_2, and (111, 011) at three inputs of AND gates connected with k_1 respectively. Again, the vector pair (v_3, v_4) generates (011, 111) at the three inputs of the AND gates connected with control input k_1, and (111, 011) at three inputs of AND gates connected with k_0 respectively. The vector pair (v_4, v_5) generates (111, 011) at the three inputs of the AND gates connected with control input k_2. The vector pair (v_5, v_8) generates (011, 111) at the three inputs of the AND gates connected with control input k_0. The sequence (v_8, v_6, v_8) will

produce (111,101,111) sequence at three inputs of all the AND gates. The sequence (v_8, v_7, v_8) will produce the sequence (111,110,111) at three inputs of all the AND gates in the multiplier circuits. Hence, proof. □

Theorem 2: Any transition fault in the proposed Multiplier network is testable by the constant test set *T* of length 10*,* where $T = (v_1, v_2, v_3, v_4, v_5, v_8, v_6, v_8, v_7, v_8)$.
Proof: Follows from *Lemma* 1, and 2. □

3.1 Gate Complexities

The gate complexities different types of polynomials are given in the Table 3. In this table, the value of s is 1 for All One Polynomial (AOP) and m/2 for trinomial defined by x^m+x^k+1 where k=m/2. For k ≠m/2, s = 1 for trinomial.

4. Experimental Results

We performed area, delay, power and test set size analysis on various $GF(2^m)$ multipliers based on different polynomial basis. The area, power and delay analysis are based on 0.18µ CMOS technology library from UMC. The table 4 shows that the area of some *GF* multiplier circuits has been increased by approximately only 6 percent to ensure 100 percent testability. Since the overall multiplier complexity depends on primitive polynomial, there is a slight variation in percentage overhead [Table 4]. The comparative analysis of area, delay and power is shown in Fig. 8. On an average there is 6 percent increase in area and power. The delay overhead is negligible, when the overall delay of the multiplier is considered. The designs were synthesized using the Synopsys tools. Synopsys's Power Compiler® was used to estimate the power consumption.

Our test set is constant of length only 10, which eliminates the need for test generation programs. Table 4 compares our test scheme with ATPG-based test generation. Synopsys® tools are used to generate ATPG based test pattern. In ATPG-based scheme, all the *GF* multiplier circuits require more test patterns than that required in the proposed easily generated test generation scheme for achieving 100% fault coverage. For example, in ATPG based scheme, $GF(2^{16})$ multiplier circuit requires 32-bit more than 97 test patterns, whereas in our scheme it requires only 32-bit 10 test patterns. As our scheme requires only 10 constant test patterns, it ensures reductions in test application time and the associated power consumptions.

BIST scheme may be incorporated to generate the required 10 vectors internally. The BIST will provide two benefits: firstly it will eliminate the need for the three control inputs necessary for fully testing the multipliers, and secondly it will provide an added level of security. It can be designed with minimum hardware to generate 10 test vectors. Note that for all fields, the logic will remain same because the pattern remains the same. Only word length varies depending upon *m*.

5. Conclusion

This paper presents a C-testable design of PB bit-parallel multipliers over $GF(2^m)$ for achieving 100% faults coverage. For an *m*-bit multiplier circuit, a constant test set of length 10 for detecting both the transition and stuck-at faults is derived. The testable design requires 6% (avg.) extra hardware and 3 control inputs. BIST circuit can be used to generate test pattern internally. This eliminates the need for the three control inputs and also provides an added level of public-key security. The test set being very short in length, reduces test application time and test power.

References

1. Y. Wu and M.I. Adham, "Scan-Based BIST Fault Diagnosis," *IEEE TCAD*, vol. 18, no. 2, pp. 203-211, 1999.
2. I.S. Reed and X. Chen, *Error-Control Coding for Data Networks*. Kluwer Academic, 1999.
3. J.H. Guo and C.L. Wang, "Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in $GF(2^m)$," *IEEE Trans. Computers*, vol. 47, no. 10, pp. 1161-1167, 1998.
4. A. Reyhani-Masoleh, and M. A. Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$", *IEEE Trans. Computers*, vol.53, no.8, pp.945-959, 2004.
5. E.D. Mastrovito, "VLSI Architectures for Computation in Galois Fields," *PhD thesis*, *Linkoping Univ., Linkoping, Sweden,* 1991.
6. D. K. Pradhan, "A Theory of Galois Switching Functions", *IEEE Trans. Computers,* vol. 27, no. 3, pp.239-248, Mar. 1978.

7. H. Rahaman, D. K. Das, and B. B. Bhattacharya, "Testable design of GRM network with EXOR-tree for detecting stuck-at and bridging faults," *ASPDAC 2004*, pp. 224-229.
8. H. Rahaman, J. Mathew, A. M. Jabir and D. K. Pradhan, "Easily Testable Implementation for Bit Parallel Multipliers in GF (2^m)", *HLDVT 2006*.
9. J. A. Waicukauski, E. Lindbloom, B. K. Rosen and V. S. Iyengar, "Transition Fault Simulation", *IEEE Design & Test of Computers*, Vol. 4, No. 2, April 1987.
10. A. K. Pramanick and S. M. Reddy, "On the Detection of Delay Faults", *ITC88*, pp. 845-856.
11. G. L. Smith, "Model for Delay Faults Based Upon Paths", *ITC85*, pp. 342-349.
12. C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits", *IEEE TCAD*, pp. 694-703, Sept. 1985.
13. I. Pomeranz and S. M. Reddy, "Static Compaction for Two-Pattern Test Sets", *Proc. ATS*, pp. 222-228, 1995.
14. X. Liu, M. S. Hsiao, S. Chakravarty and P. J. Thadikaran, "Novel ATPG Algorithms for Transition Faults", *ETW*, pp. 47-52, May 2002.
15. K. T. Cheng. Transition Fault Simulation for Sequential Circuits. In *Proc. International Test Conference,* pp.723-731, October 1992.
16. C. Haung and C. WU, "High-Speed C-Testable Systolic Array Design for Galois-Field Inversion," ED&TC 97, pp.342-346.
17. BLAHUT, R. E. 1985. *Fast Algorithms for Digital Signal Processing*. Addison- Wesley.

Table 3: Gate complexities for different Polynomial Basis

Polynomials type	Original Implementation		Testable Implementation	
	# of 2 inputs AND gate	# of 2 inputs EXOR gate	# of 3 inputs AND gate	# of 2 inputs EXOR gate
ESP	m^2	m^2-s	m^2	$m^2 - s$
Trinomial	m^2	$m^2 - 1$	m^2	$m^2 - 1$
Pentanomial	m^2	$m^2 + m$	m^2	$m^2 + m$

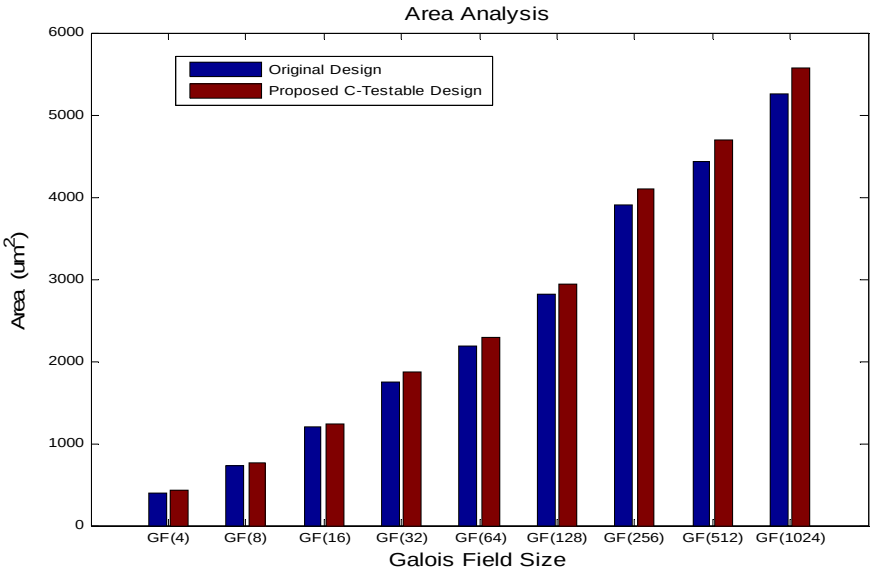

Fig. 8a: Area: Original vs C-Testable Version

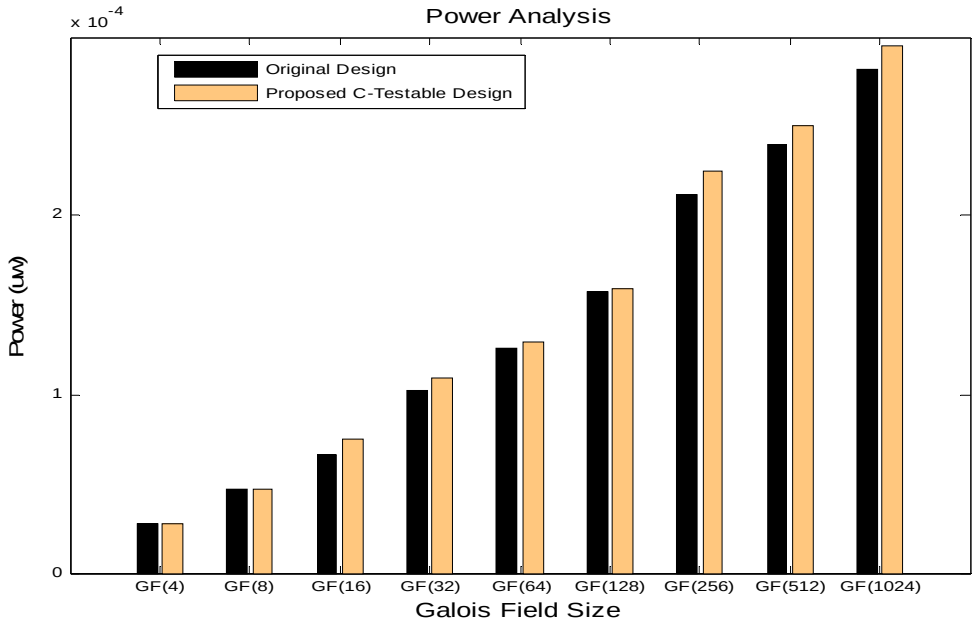

Fig. 8b: power : Original vs C-Testable Version

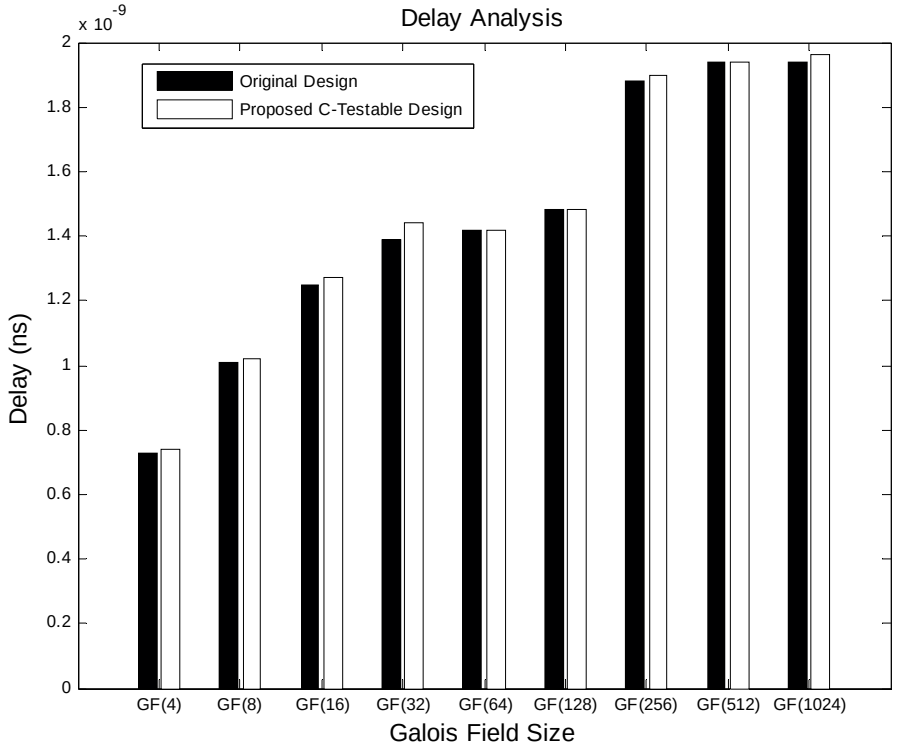

Fig. 8c: Delay Analysis: Original vs C-Testable Version

Table 4: Details of area and number of tests required to achieve 100% coverage

GF multiplier	Irreducible Polynomial	Area in μm^2		% extra area for testability	# of tests for 100% faults coverage	
		Original circuit	Testable circuit		Original ckt using ATPG	Testable circuit
$GF(2^2)$	x^2+x+1	400.02	425.8	6.4	9	10
$GF(2^3)$	x^3+x+1	728.9	758.06	5.3	15	10
$GF(2^4)$	x^4+x+1	1200.1	1238.1	5.18	22	10
$GF(2^5)$	$x^5+x^3+x^2+x+1$	1748.3	1864.4	6.64	24	10
$GF(2^6)$	x^6+x+1	2180.5	2296.6	5.32	38	10
$GF(2^7)$	x^7+x+1	2819.2	2932.1	5.31	39	10
$GF(2^8)$	$x^8+x^4+x^3+x^2+1$	3896.6	4103.0	6.7	50	10
$GF(2^9)$	x^9+x^4+1	4431.9	4693.3	5.9	72	10
$GF(2^{10})$	$x^{10}+x^3+1$	5251.3	5573.8	6.52	97	10