6-27-2011 12:00 AM

Reliable and High-Performance Hardware Architectures for the Advanced Encryption Standard/Galois Counter Mode

Mehran Mozaffari-Kermani, *The University of Western Ontario*

Supervisor: Dr. Arash Reyhani-Masoleh, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Electrical and Computer Engineering

RELIABLE AND HIGH-PERFORMANCE HARDWARE ARCHITECTURES FOR THE ADVANCED ENCRYPTION STANDARD/GALOIS COUNTER MODE

(Spine Title: Reliable and High-Performance Architectures for the AES/GCM)
(Thesis Format: Monograph)

by

Mehran Mozaffari Kermani

Graduate Program in Electrical and Computer Engineering

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada
June 2011

THE UNIVERSITY OF WESTERN ONTARIO
School of Graduate and Postdoctoral Studies

CERTIFICATE OF EXAMINATION

Examination Board:

.....
Dr. Amr M. Youssef

Supervisor:

.....
Dr. Anestis Dounavis

.....
Dr. Arash Reyhani-Masoleh

.....
Dr. Hanan Lutfiyya

.....
Dr. Xianbin Wang

The thesis by

Mehran Mozaffari Kermani

entitled:

Reliable and High-Performance Hardware Architectures for the Advanced Encryption Standard/Galois Counter Mode

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

.....
Date

.....
Chair of the Thesis Examination Board

Abstract

The high level of security and the fast hardware and software implementations of the Advanced Encryption Standard (AES) have made it the first choice for many critical applications. Since its acceptance as the adopted symmetric-key algorithm, the AES has been utilized in various security-constrained applications, many of which are power and resource constrained and require reliable and efficient hardware implementations.

In this thesis, first, we investigate the AES algorithm from the concurrent fault detection point of view. We note that in addition to the efficiency requirements of the AES, it must be reliable against transient and permanent internal faults or malicious faults aiming at revealing the secret key. This reliability analysis and proposing efficient and effective fault detection schemes are essential because fault attacks have become a serious concern in cryptographic applications. Therefore, we propose, design, and implement various novel concurrent fault detection schemes for different AES hardware architectures. These include different structure-dependent and independent approaches for detecting single and multiple stuck-at faults using single and multi-bit signatures.

The recently standardized authentication mode of the AES, i.e., Galois/Counter Mode (GCM), is also considered in this thesis. We propose efficient architectures for the AES-GCM algorithm. In this regard, we investigate the AES algorithm and we propose low-complexity and low-power hardware implementations for it, emphasizing on its nonlinear transformation, i.e., SubBytes (S-boxes). We present new formulations for this transformation and through exhaustive hardware implementations, we show that the proposed architectures outperform their counterparts in terms of efficiency. Moreover, we present parallel, high-performance new schemes for the hardware implementations of the GCM to improve its throughput and reduce its latency.

The performance of the proposed efficient architectures for the AES-GCM and their fault detection approaches are benchmarked using application-specific integrated circuit (ASIC) and field-programmable gate array (FPGA) hardware platforms. Our comparison results show that the proposed hardware architectures outperform their existing counterparts in terms of efficiency and fault detection capability.

Keywords: Advanced Encryption Standard, finite field, Galois/Counter Mode, high performance, concurrent fault detection.

Dedication

To my parents for their love, inspiration, and guidance.

Acknowledgements

I would like to express my sincere appreciation and gratitude to Dr. Arash Reyhani-Masoleh for supervising my research during my Ph.D. studies. I would like to also thank my lab-mates, Dr. Arash Hariri, Christopher Kennedy, and Reza Azarderakhsh for sharing their experience and knowledge with me.

Contents

List of Figures

List of Tables

List of Abbreviations

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
ASIC	Application-Specific Integrated Circuit
DES	Data Encryption Standard
FPGA	Field-Programmable Gate Array
GCM	Galois/Counter Mode
GF	Galois Field
LUT	Look-up Table
NIST	National Institute of Standards and Technology
VHDL	Very-high-speed integrated circuit Hardware Description Language
VLSI	Very Large Scale Integration

Chapter 1

Introduction and Preliminaries

IN this chapter, we present an introduction for this thesis. We also provide the preliminaries, motivations, and thesis outline.

Symmetric key cryptography uses a shared key in both sender and receiver ends during encryption and decryption for secure communications. For the drawbacks of the previous symmetric-key cryptographic standards such as the DES and the 3DES, they have been replaced by the Advanced Encryption Standard (AES) [1]. In particular, the AES has overcome the drawbacks of the previous standards in terms of vulnerability to brute force attacks and slow software implementations. The AES was accepted by the National Institute of Standards and Technology (NIST) in 2001 and since its acceptance, it has been utilized in a variety of security-constrained applications. For instance, it has been included in wireless standards of Wi-Fi [2] and WiMAX [3] and many other applications, ranging from the security of smart cards to the bitstream security mechanisms in FPGAs [4].

The Advanced Encryption Standard-Galois/Counter Mode (AES-GCM) provides authentication and confidentiality for sensitive data simultaneously. In the AES-GCM, data confidentiality is provided by the AES [1]. The authentication of the AES-GCM is provided by the Galois/Counter Mode (GCM) [5] using a universal hash function. The AES-GCM has been used for a number of applications such as the new LAN security standard WLAN 802.1ae (MACSec) [6] and Fibre Channel Security Protocols (FC-SP) [7]. Moreover, it has been utilized in a number of cores from industry, see, for example, [8], [9], and [10]. In addition, two AES-GCM software-based implementations have been presented in [11] and [12].

In what follows, we present the details on the AES and the GCM algorithms.

1.1 Advanced Encryption Standard

In the AES encryption, the input and the output blocks are limited to 128 bits. However, based on the security requirements, the key size could be determined as 128 (AES-128), 192 (AES-192) or 256 (AES-256). For each of these three types of the AES, different number of rounds corresponding to different levels of security objectives is utilized, i.e., for the AES-128, 10 rounds, for the AES-192, 12 rounds and for the AES-256, 14 rounds are processed [1]. In the AES encryption, four transformations in all the rounds, except for the last round which has three transformations, are utilized.

The AES encryption transformations for the typical round j are depicted in Fig. 1.1. The 128 bits of the input and output of each transformation are considered as four by four matrices, called *states* (shown in dotted rectangles in Fig. 1.1), whose entries are eight bits. The first transformation in each round is *SubBytes* (S-boxes), which is implemented by 16 S-boxes. In the S-box, each byte of the input state (X_i, $0 \leq i \leq 15$, in Fig. 1.1) is substituted by a new byte (Y_i, $0 \leq i \leq 15$, in Fig. 1.1). *ShiftRows* is the second transformation in which the first row of the state remains intact and the four bytes of the last three rows of the input state are cyclically shifted. The third transformation is *MixColumns* in which each column is modified individually. As shown in Fig. 1.1, the columns are considered as polynomials over $GF(2^8)$ and are multiplied by a fixed polynomial. The final transformation is *AddRoundKey* which performs the modulo-2 addition of the input state and the key of the corresponding round, i.e., k_j, $1 \leq j \leq 10$, 12 or 14 for the AES-128, 192 or 256, respectively [1]. In the AES decryption, the reverse procedure of the AES encryption is performed [1].

For realizing the S-box, the irreducible polynomial of $P(x) = x^8 + x^4 + x^3 + x + 1$ is used to construct the binary field $GF(2^8)$. Let $X_i \in GF(2^8)$ and $Y_i \in GF(2^8)$ be the input and the output of the S-box. Then, the S-box consists of finding the multiplicative inversion, i.e., ${X_i}^{-1} \in GF(2^8)$ with the exception of mapping the zero input to the zero output, followed by the affine transformation in $GF(2^8)$ [1]. For the inverse S-boxes of the AES decryption, inverse affine transformation precedes the multiplicative inversion [1].

Among the four different transformations in the AES, only the S-box and the inverse S-box are non-linear. Additionally, all the S-boxes (resp. the inverse S-boxes) occupy

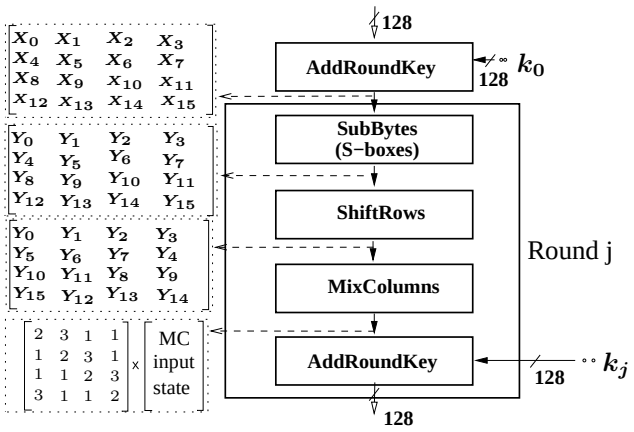


Figure 1.1: The AES encryption round transformations [1].

much of the total AES encryption (resp. decryption) area and their power consumption is around three fourths of that of the entire AES [13]. In what follows, we present the preliminaries regarding the hardware implementations of the S-boxes and the inverse S-boxes within the AES using look-up tables (LUTs) and composite fields.

1.1.1 LUT-Based Architectures

The AES S-boxes and inverse S-boxes can be implemented using LUTs. For this purpose, 256×8 memory cells are used to store the 256 possible 8-bit outputs of each S-box/inverse S-box. The LUT-based implementation is suitable for the field-programmable gate array (FPGA) platforms in which block memories are available, see, for example, [14], [15], and [16]. However, although this implementation reaches high-speed architectures, it is not suitable for applications requiring low-complexity AES application-specific integrated circuit (ASIC) implementations [17].

The S-box and the inverse S-box are nonlinear operations which take 8-bit inputs and generate 8-bit outputs. In the S-box, the irreducible polynomial of $P(x) = x^8 + x^4 + x^3 + x + 1$ is used to construct the binary field $GF(2^8)$. The usage of arithmetic in composite fields reduces the space complexity of the S-box. Moreover, it allows us to use pipelining and therefore the effective speed of the AES is increased while processing independent messages. Consequently, the S-boxes and inverse S-boxes implemented using composite fields can lead to area-efficient and high-performance structures [17]. In the following, the preliminaries on composite field realizations are presented.

1.1.2 Composite Field Architectures

In this section, we describe the composite field arithmetics to calculate the multiplicative inversion over $GF(2^8)$. This approach has received much attention in the literature, see, for example, [13], [17], [18], [19], [20], [21], [22], [23], [24], and [25]. Moreover, there have been low-power implementations for the S-boxes (resp. the inverse S-boxes) such as the ones in [13] and [26]. It is noted that the low-power S-box (resp. inverse S-box) presented in [13] uses composite fields.

The composite fields can be represented using normal basis [23] or polynomial basis [18], [20], [21], [22]. The composite field realizations of the S-box using polynomial and normal bases are presented in Fig. 1.2. As seen in this figure, a transformation matrix transforms a field element $X \in GF(2^8)$ to the corresponding representation in the composite field $GF(16^2)$, i.e., η. We consider the irreducible polynomial of $u^2 + u + \nu$, where ν is chosen over $GF(16)$ depending on the composite fields. Then, the multiplicative inversion generates the inverse as $\psi = \eta^{-1}$. Finally, as seen in Fig. 1.2, the inverse transformation matrix transforms the composite field element to the one in the binary field, i.e., $Y \in GF(2^8)$.

Using polynomial basis constructed by the irreducible polynomial of $u^2 + u + \nu$, one can obtain the coordinates of ψ as $\psi_h = \eta_h(\eta_h^2\nu + \eta_h\eta_l + \eta_l^2)^{-1}$ and $\psi_l = (\eta_l + \eta_h)(\eta_h^2\nu + \eta_h\eta_l + \eta_l^2)^{-1}$ [20]. This multiplicative inversion in composite fields using polynomial basis is shown in the top part of Fig. 1.2 by a dotted rectangle. Similarly, for normal basis, the coordinates of ψ are obtained as $\psi_h = (\eta_h\eta_l + ({\eta_h}^2 + {\eta_l}^2)\nu)^{-1}\eta_l$ and $\psi_l = (\eta_h\eta_l + ({\eta_h}^2 + {\eta_l}^2)\nu)^{-1}\eta_h$ [23], shown in the dotted rectangle in the bottom of Fig. 1.2. One can refer to [20] and [23] for more details on the composite field S-box architectures. As seen in Fig. 1.2, the above multiplicative inversions consist of composite field multiplications, additions and inversion in the sub-field $GF(16)$. In this figure, the sub-field multiplications are shown by crossed circles. Moreover, the circle with plus inside represents $GF(2^4)$ addition using 4 XOR gates.

1.2 The Galois/Counter Mode

Authenticated encryption and decryption are the two functions within the GCM. The authenticated encryption performs two tasks; encrypting the confidential data and com-

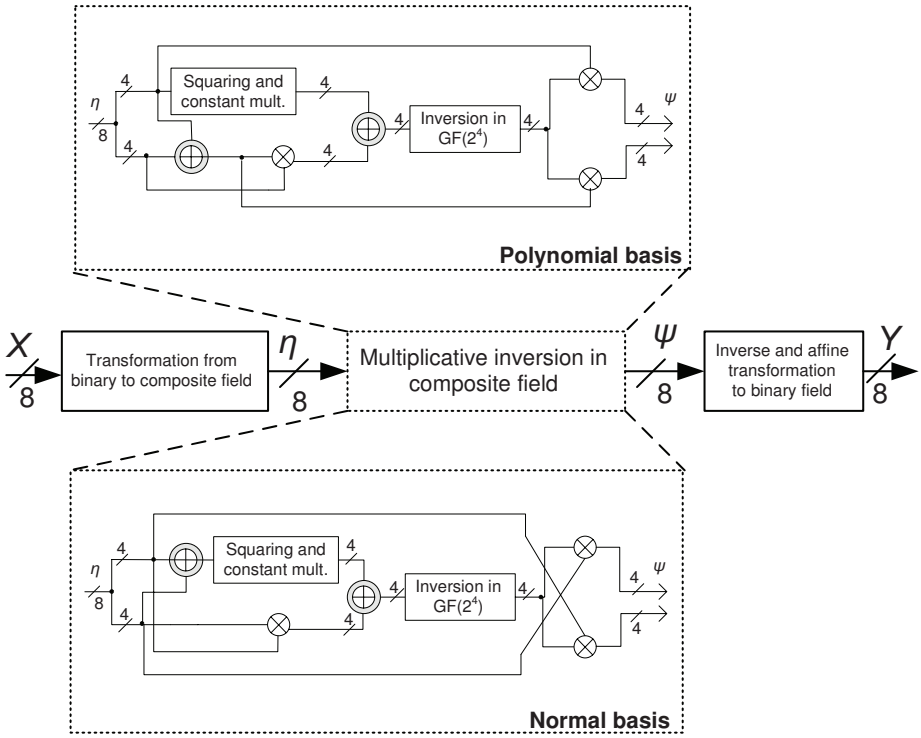


Figure 1.2: The composite field S-box architecture using polynomial basis [20] and normal basis [23].

puting an authentication tag. The authenticated decryption function decrypts the confidential data and verifies the tag [5]. The data flow of the authenticated encryption is shown in Fig. 1.3. As seen in this figure, the mechanism for the confidentiality of data is a variation of the block cipher counter mode of operation, denoted by $GCTR_K$ (Galois Counter with the key K) [5]. For the AES-GCM, the block cipher encryption with the specific key K is shown by AES_K in Fig. 1.3. Then, the function $GCTR_K$ performs the block cipher counter mode with the *Initial Counter Block (ICB)* and its increments $(CB_2 - CB_i)$ and the plaintext blocks $(P_1 - P_i)$ as the inputs.

As shown in Fig. 1.3, the Galois Hash ($GHASH_H$) function within the GCM provides the authentication for the confidential data. This function is constructed by $GF(2^{128})$ multiplications with a fixed parameter, called the hash subkey (H). The $GHASH_H$ function calculates

$$\sum_{j=1}^{n} X_j H^{n-j+1} = X_1 \cdot H^n \oplus X_2 \cdot H^{n-1} \oplus \ldots \oplus X_n \cdot H, \tag{1.1}$$

where X_1 to X_n are the n, 128-bit blocks of the input [5]. It is noted that the hash subkey

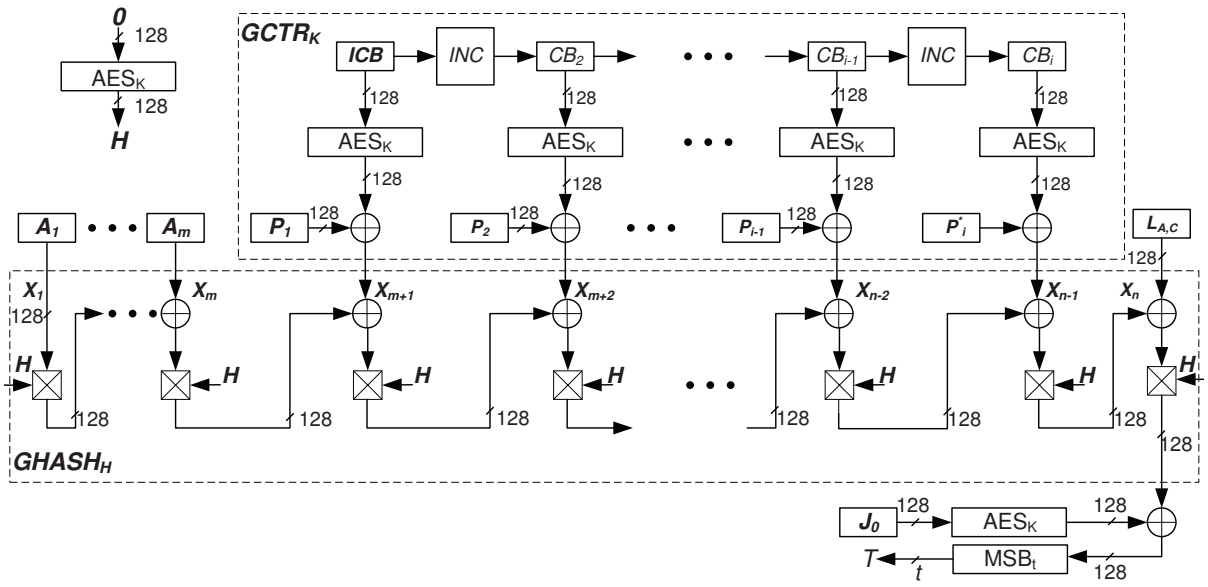


Figure 1.3: The GCM authenticated encryption data flow [5].

is generated by applying the AES to the *zero* block, i.e., $0 = (0, 0, ..., 0) \in GF(2^{128})$. Then, the $GHASH_H$ function calculates (1.1) [5]. All the arithmetic operations in (1.1), i.e., additions, GF multiplications, and exponentiations are performed over $GF(2^{128})$ constructed by the irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$. As seen in Fig. 1.3, the total number of input blocks to $GHASH_H$ is $n = m + i + 1$, where m and i are the number of blocks for the additional authenticated data ($A_1 - A_m$) and the output of $GCTR_K$, respectively. Eventually, the authentication tag T with length of t bits is derived. In the authenticated decryption, the same $GHASH_H$ procedure is performed on the authenticated data and ciphertext blocks to verify the tag. For the entire description of the GCM, one can refer to [5] and Algorithms 1 and 2.

Algorithm 1 shows the GCM authenticated encryption [5]. In this algorithm, IV is the Initialization Vector, P is the Plaintext, A is the Additional Authenticated Data, and K is the Key. It is noted that the authentication in the GCM is performed based on the hash function $GHASH_H$. After deriving J_0, $GHASH_H$ is applied to $(A\|0^v\|C\|0^u\|[len(A)]_{64}\|$ $[len(C)]_{64})$ to obtain block S, from which the authentication tag T with length of t is derived.

Algorithm 2 depicts the GCM authenticated decryption [5]. This algorithm uses the same functions as Algorithm 1, in which another authentication tag (T') is derived. Al-

Algorithm 1 The Authenticated Encryption GCM-AE_K(IV, P, A)

1: Let $H = CIPH_K(0^{128})$.
2: Define a block, J_0, as follows:
If $len(IV) = 96$, then let $J_0 = IV \| 0^{31} \| 1$.
If $len(IV) \neq 96$, then let $s = 128\lceil len(IV)/128 \rceil - len(IV)$, and let $J_0 = GHASH_H(IV \| 0^{s+64} \| [len(IV)]_{64})$.
3: Let $C = GCTR_K(inc_{32}(J_0), P)$.
4: Let $u = 128\lceil len(C)/128 \rceil - len(C)$ and $v = 128\lceil len(A)/128 \rceil - len(A)$.
5: Define a block, S, as follows:
$S = GHASH_H(A \| 0^v \| C \| 0^u \| [len(A)]_{64} \| [len(C)]_{64})$.
6: Let $T = MSB_t(GCTR_K(J_0, S))$.
7: Return (C, T).

Algorithm 2 The Authenticated Decryption GCM-AD_K(IV, C, A, T)

1: If the bit lengths of IV, A or C are not supported, or if $len(T) \neq t$, then return $FAIL$.
2: Let $H = CIPH_K(0^{128})$.
3: Define a block, J_0, as follows:
If $len(IV) = 96$, then let $J_0 = IV \| 0^{31} \| 1$.
If $len(IV) \neq 96$, then let $s = 128\lceil len(IV)/128 \rceil - len(IV)$, and let $J_0 = GHASH_H(IV \| 0^{s+64} \| [len(IV)]_{64})$.
4: Let $P = GCTR_K(inc_{32}(J_0), C)$.
5: Let $u = 128\lceil len(C)/128 \rceil - len(C)$ and $v = 128\lceil len(A)/128 \rceil - len(A)$.
6: Define a block, S, as follows:
$S = GHASH_H(A \| 0^v \| C \| 0^u \| [len(A)]_{64} \| [len(C)]_{64})$.
7: Let $T' = MSB_t(GCTR_K(J_0, S))$.
8: If $T = T'$, return P otherwise return $FAIL$.

gorithm 2 performs the verification of authenticity by checking if the sent authentication tag T is the same as T'.

1.3 Motivation

Using the AES, the sender and the receiver of the sensitive data share a secret key to ensure the confidentiality of the information. Nonetheless, a malicious attacker can take over the secret key and compromise the standard. One of the methods for extracting the side-channel information is the fault attacks for which several approaches have been introduced, see, for instance, [27], [28], [29], [30], [31], [32], and [33]. It is noted that the internal hardware failures may also result in malfunctioning of the AES encryption/decryption. This has been the motivation for the first contribution of this thesis to

develop high-performance and low-overhead fault detection schemes for the AES.

Different GCM architectures have been presented in the literature, the details of which are provided throughout this thesis. These methods of realization mostly need many clock cycles to execute, reducing the performance of the GCM architectures and resulting in low throughput. This has been a motivation for the second contribution of this thesis to propose high-performance parallel methods for obtaining the GCM and developing efficient structures for the AES-GCM. The proposed methods are suitable for high-performance applications.

1.4 Thesis Outline

The rest of this thesis is organized as follows. In Chapter 2, we review some of the existing works in the literature. In Chapter 3, different AES S-boxes are evaluated and benchmarked in terms of area, delay, and simulation-based power consumption. In Chapter 4, a high-performance fault detection scheme for the composite field S-boxes and inverse S-boxes of the AES is presented and benchmarked. Chapter 5 presents a concurrent low-overhead fault detection method for the AES with emphasis on burst fault detection. In Chapter 6, a structure-independent fault detection approach for the entire AES encryption and decryption is presented. Chapter 7 covers the proposed high-performance parallel hardware architecture for the AES-GCM. Finally, in Chapter 8, we summarize our contributions.

Chapter 2

Literature Review

THIS chapter presents some previous works on both fault detection and hardware implementations of the AES-GCM.

2.1 Fault Detection Schemes

Several fault detection schemes have been proposed to date to counteract the fault attacks and detect the natural faults in cryptographic algorithms and the AES, see, for example, [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], and [51].

For fault detection of the encryption or decryption in AES one may use redundant units [34], [42], where algorithm-level, round-level and operation-level concurrent error detection for the AES are used. In the algorithm-level, comparing the plain text with the output of a decryption after an encryption is proposed. The round-level error detection uses similar ideas in the rounds, where, the output of a round in encryption is applied to a round in decryption and is compared with the input. The operation-level (or transformation-level) error detection uses the inversion of a transformation in each round and compares the output with the input. Fig. 2.1 shows the operation-level concurrent error detection for S-box and inverse S-box presented in [34]. In this figure, the 8-bit input X of the S-box (8-bit input Y of the inverse S-box) is compared with the output of two consecutive transformations, S-box and inverse S-box (inverse S-box and S-box) using an 8-bit comparator to generate the error indication flag.

There exist a number of fault detection schemes based on the error detecting codes, see, for example, [35], [36], [37], [38], [39], [40], and [41]. Using one parity bit for each

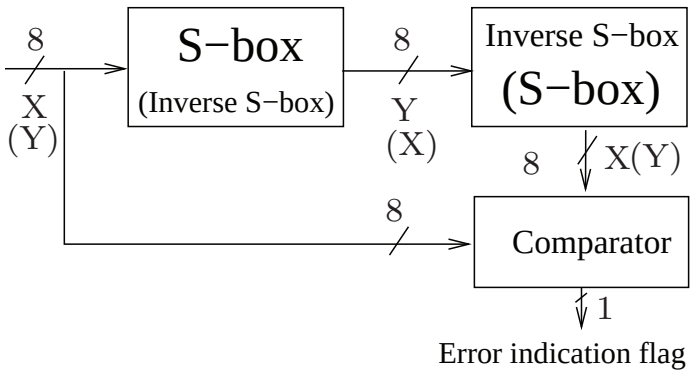


Figure 2.1: Redundant unit fault detection structure for S-box (inverse S-box) [34], [42].

byte of a transformation, one can obtain the structure shown in Fig. 2.2 for the round i, $1 \leq i \leq 9$, of the encryption of the AES-128 (128-bit key) to achieve a parity-based fault detection scheme. Similar structure can be obtained for the AES-128 decryption. The AES-128 encryption/decryption has 10 consecutive rounds which are similar except for the last one in which one of the transformations is not used. As seen in Fig. 2.2, the output parity bits of each transformation in every round of the AES encryption are predicted from the inputs using the prediction boxes denoted by $\hat{P}$ notations. Then, the comparisons between the predicted parities (shown by a matrix with 16-bit entries) and the actual parities (obtained using the actual parity block) in Fig. 2.2 can be scheduled so that the desired fault detection capability is obtained.

Parity predictions of ShiftRows, InvShiftRows, and AddRoundKey are straightforward and those of MixColumns and InvMixColumns can be done using the equations given in [35], [36], [40], and [41]. It is noted that the parity predictions of the S-box and the inverse S-box proposed in [36] are based on LUTs implementations in which 512×9 memory cells are used to generate the predicted parity bit as well as the 8-bit output. In Fig. 2.2, let k_0 be the 128-bit input key to the key expander. Then, all the modified keys, i.e., k'_i, $0 \leq i \leq 10$, consist of the 128-bit expanded key k_i and 16-bit parities, if one bit parity is used for each byte.

The parity-based scheme proposed in [35] is one of the first fault detection schemes and has received attention in the literature. Although the approach in [35] is a good scheme in terms of the fault detection capability, it has two drawbacks. First, this approach

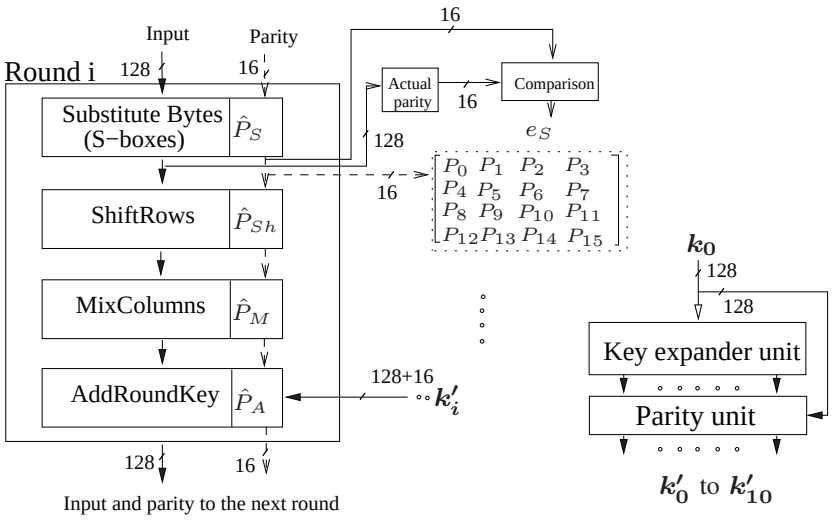


Figure 2.2: Parity-based fault detection structure of the i_{th} round in the AES-128 encryption.

is based on using the expanded S-boxes and inverse S-boxes for parity predictions, i.e., two blocks of 256×9 memory cells. Not only does this restrict the AES encryption and decryption implementations to LUT-based S-boxes and inverse S-boxes, but it has the area overhead of greater than 100% for either the S-box or the inverse S-box. The second drawback of the approach in [35] is the relatively high area complexity of the parity predictions of MixColumns in the AES encryption. For the AES decryption, the area complexity of the predicted parities of InvMixColumns is even more [36].

In [37] and [39], instead of using one parity bit or two signatures in case of using the scheme presented in [38] for each byte, one bit parity is used for 128-bit data using the LUT S-boxes. The multiplication-based fault detection scheme [38] for the multiplicative inversion of the S-box is shown in Fig. 2.3. In this scheme, the 8-bit input of the multiplicative inversion is multiplied by the 8-bit output and the n-bit result, $1 \leq n \leq 8$, of the multiplication is compared with the n-bit actual result, i.e., $1 \in GF(2^8)$ if $X \neq 0$ and $0 \in GF(2^8)$ if $X = 0$. Because the multiplicative inversion is also used in the inverse S-box, the same scheme can be used for the inverse S-box.

The schemes presented in [34] and [42] use the redundant unit fault detection approach. It is noted that this results in the area, power, and delay overheads of approximately 100%. In addition, the scheme in [43] proposes using the transformations in an AES round twice for the same data to detect the transient errors. In [44], a concurrent

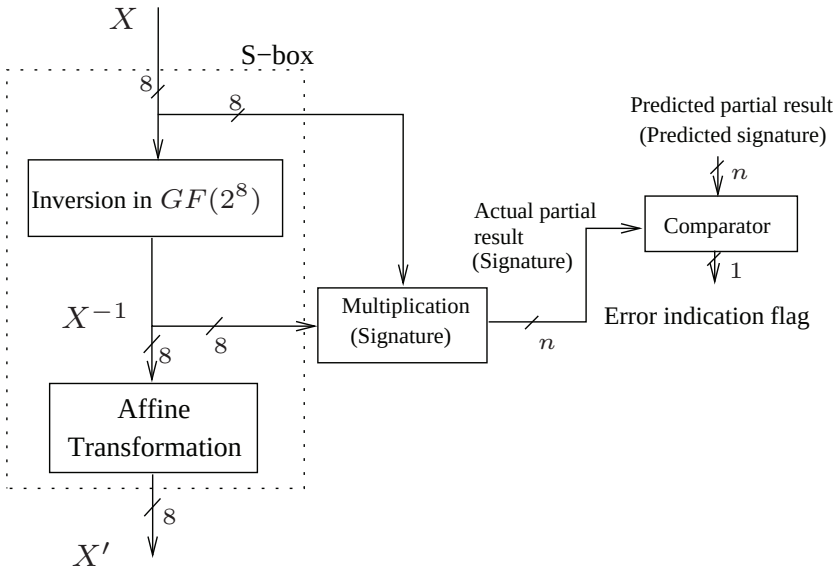


Figure 2.3: The multiplication-based scheme for the fault detection of the multiplicative inversion [38].

fault detection scheme based on the merged S-box and inverse S-box is proposed. It is also noted that the scheme presented in [49] uses double-data-rate computation for counteracting the fault attacks. Additionally, a fault detection scheme based on the Hamming and Reed-Solomon codes for protecting the storage elements within the AES is proposed in [50]. Furthermore, for the logic elements, the scheme in [36] and the use of the partial duplication of the most vulnerable elements are proposed in [50]. Moreover, the approach in [51] is based on implementing functional redundancy in the AES.

There exist a number of fault detection approaches which are specific to composite field S-boxes and inverse S-boxes, see, for example, [52], [53], [54], [55], and [56]. In the scheme of [52], the fault detection of the multiplicative inversion of the S-box is considered for two specific composite fields. The transformation and affine matrices are excluded in this approach. Moreover, in [53], predicted parities have been used for the multiplicative inversion of a specific S-box using composite field and polynomial basis. Furthermore, the transformation matrices are also considered. In [54], [55], and [56], the composite field S-boxes and inverse S-boxes (using polynomial basis) have been divided into sub-blocks and parity predictions are used for their fault detection. Moreover, FPGA implementations have been performed in [56] to benchmark the presented method. It is noted that the approaches in [55] and [56] (for single fault detection) have been extended

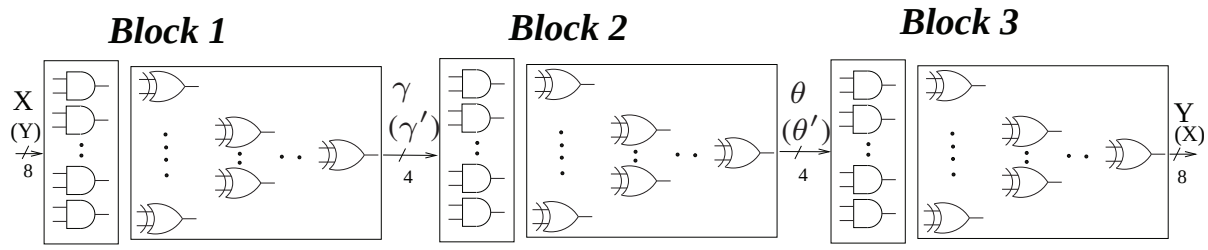


Figure 2.4: Low-power S-box (resp. inverse S-box) architecture using composite fields and polynomial basis [13].

in the work in [57]. This work presents new structures for the S-box and the inverse S-box with higher complexities compared to the original structures for detecting 100% of single faults. We note that unlike the schemes presented in [55] and [56], this work focuses on the stuck-at faults injected not only at the outputs but at any net in the circuit. The results in [57] have been benchmarked using ASIC platform.

2.2 AES-GCM Architectures

As mentioned in the previous chapter, the S-boxes and the inverse S-boxes are the only nonlinear transformations in the AES, whose hardware implementations affect that of the AES significantly. A low-power implementation of the S-box (resp. inverse S-box) has been presented in [13] which uses the composite field in [20]. For reaching a low-power architecture with acceptable hardware complexity, it is suggested in [13] that the structures are partitioned into three blocks (see Fig. 2.4). Then, the logic gates within each of these blocks are implemented using two-level logics consisting of the arrays of ANDs and XORs. Although this method increases the area of the composite fields implementation, it reduces the power consumption significantly [13].

The AND-XOR structure of each block shown in Fig. 2.4 results in the low number of transitions and thus low power consumption. This is because the AND array has 50% propagation probability of signal transitions. In [13], similar to many other publications such as [17], [18], [20], and [22], the irreducible polynomials $u^2 + u + \nu$ and $v^2 + v + \Phi$, where $\nu = \{1100\}_2$ and $\Phi = \{10\}_2$, are used for the composite fields. As seen in Fig. 2.4, for block 1, a field element X for the S-box (Y for the inverse S-box) in the binary field $GF(2^8)$ is converted to the corresponding representation in the composite

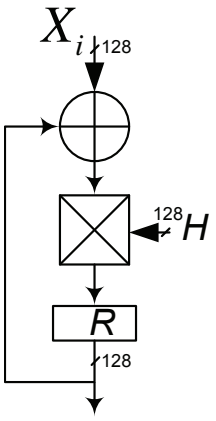


Figure 2.5: The sequential method used in [64], [65], and [66] for the hardware implementation of the $GHASH_H$.

field $GF(2^8)/GF(((2^2)^2)^2)$. The output of block 1 is then obtained as $\gamma \in GF(2^4)$ ($\gamma' \in GF(2^4)$ for the inverse S-box). As seen in Fig. 2.4, $\theta \in GF(2^4)$ ($\theta' \in GF(2^4)$ for the inverse S-box) is then derived as the output of block 2. Eventually, using the irreducible polynomials $u^2 + u + \nu$ and $v^2 + v + \Phi$, the output of the S-box, i.e., Y (X for the inverse S-box), is obtained after conversion from the composite field $GF(2^8)/GF(((2^2)^2)^2)$ to the binary field $GF(2^8)$.

In some previous works such as [13], [20], [22], [27], and [58], one specific S-box and in [59], three reported S-boxes have been synthesized on ASIC. However, exhaustive search has not been performed for all suitable composite fields to evaluate their performance metrics using the same technology. It is also noted that in some other works, see, for instance, [23], [24], [30], [60], [61], and [62], the hardware and timing complexities of different composite field S-boxes have been evaluated in terms of logic gates (in [63], software implementations have been performed). However, benchmarking the performance (including power consumptions through simulation-based approaches) of the S-boxes implementations on hardware platforms has not been performed in these works.

Different GCM architectures have been presented in the literature. In [64], [65], and [66], the sequential method for the hardware implementation of the GCM function is adopted. The sequential method is shown in Fig. 2.5, where one GF multiplier, a set of 128 XOR gates, and a 128-bit register (R) are utilized to perform the operation. Let the register R in Fig. 2.5 be cleared initially. Let n be the number of input blocks to

the $GHASH_H$ function, i.e., X_i, $1 \leq i \leq n$. Then, after n clock cycles, register R in Fig. 2.5 contains the result. Although this method of realization is area-efficient, it needs many clock cycles (equal to the number of input blocks), reducing the performance of the architecture.

Because of the low throughput of the sequential method, a parallel method is proposed in [67] which uses two $GF(2^{128})$ multipliers to perform this operation in parallel. This parallel implementation has been generalized in [68] and [69] so that q, $q \geq 2$, parallel $GF(2^{128})$ multiplications are performed concurrently. In the most efficient method in [68] and [69], for the case of $q = 4$ and $n = 8$, the operation in the GCM is realized according to the following calculation steps:

$$\begin{aligned}
&\overbrace{\underbrace{(\overbrace{X_1H^4}^{j=1} \oplus X_5)H^4}_{j=2\ldots}}^{j=4} \times 1 \times 1 \oplus \overbrace{\underbrace{(\overbrace{X_2H^4}^{j=1} \oplus X_6)H}_{j=2\ldots}}^{j=4} \times H \times H \oplus \\
&\overbrace{\underbrace{(\overbrace{X_3H^4}^{j=1} \oplus X_7)H}_{j=2\ldots}}^{j=4} \times H \times 1 \oplus \overbrace{\underbrace{(\overbrace{X_4H^4}^{j=1} \oplus X_8)H}_{j=2\ldots}}^{j=4} \times 1 \times 1,
\end{aligned} \tag{2.1}$$

where all operations are performed over $GF(2^{128})$ constructed by the irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$ and $\bigoplus$ comprises 128 XOR gates. Consecutive GF multiplications with H are performed for deriving the powers of the hash subkey used.

Recently, a high-performance approach for computing the $GHASH_H$ function for long messages has been proposed in [70]. However, in this scheme the hardware complexity is increased. Therefore, a high-performance parallel method for obtaining the GCM by relying on the low-complexity powers of the hash subkey is needed so that without pre-computing the hash subkey exponents, compact realizations of these exponents are obtained and implemented. This results in high-throughput and low-latency GCM hardware architectures, suitable for high-performance applications.

Chapter 3

Performance Evaluations and Comparisons of the AES S-boxes

IN this chapter, different ASIC architectures of building blocks of the AES S-boxes, the only nonlinear AES transformation, are evaluated and optimized to identify high-performance and low-power architectures. We evaluate the performance of more than 40 S-boxes utilizing a fixed benchmark platform in 65-nm CMOS technology. To obtain the least-complexity S-box, the formulations for the Galois Field (GF) sub-field inversions in $GF(2^4)$ are optimized. By conducting exhaustive simulations for the input transitions, we analyze the average and peak power consumptions of the AES S-boxes considering the switching activities, gate-level netlists, and parasitic information.

In this chapter, we logic-gate optimize and perform comprehensive ASIC syntheses of more than 40 different S-boxes for deriving their performance metrics. This benchmarking, which is done on the same platform, results in having a clear picture of the performance metrics of different designs. We synthesize the structures of different AES S-boxes using the Synopsys® Design Vision® (which is the graphical user interface to Synopsys® Design Compiler®) [73] in STM 65-nm CMOS standard technology [74]. Then, the areas and delays of these hardware architectures are derived and compared. To achieve the least dynamic power-consuming AES S-box, we obtain the average and peak power consumptions of the S-boxes through exhaustive searches considering the possible input transitions. These derivations are based on a timing simulation-based analysis using the switching activities of internal nodes with Synopsys® PrimeTime® PX [73] and ModelSim® [75].

The implementation complexities of the S-boxes using composite fields are dependent

on the choice of the coefficients $\nu \in GF(2^4)$ and $\Phi \in GF(2^2)$ in the irreducible polynomials $u^2+u+\nu$ and $v^2+v+\Phi$ used for the composite fields, respectively. The composite fields $GF(((2^2)^2)^2)$ in polynomial basis use iterations to construct the S-box. For these composite fields, the constants $\nu \in GF(2^4)$ and $\Phi \in GF(2^2)$ are over $GF((2^2)^2)/v^2+v+\Phi$ and $GF(2^2)/x^2+x+1$, respectively. According to [24], after exhaustive search for finding the possible choices for $\nu \in GF(2^4)$ and $\Phi \in GF(2^2)$, the following 16 combinations are obtained: $\Phi \in \{\{10\}_2, \{11\}_2\}$ and $\nu \in \{\{1000\}_2, \{1001\}_2, \{1010\}_2, \{1011\}_2, \{1100\}_2, \{1101\}_2$ $, \{1110\}_2, \{1111\}_2\}$. Similarly, for normal basis, it can be derived that the only two acceptable values for Φ are $\Phi = \{10\}_2$ and $\Phi = \{01\}_2$. The following 8 values of ν are acceptable: $\nu \in \{\{0100\}_2, \{0001\}_2, \{1000\}_2, \{0010\}_2, \{0111\}_2, \{1101\}_2, \{1011\}_2, \{1110\}_2\}$.

Based on the possible values of ν and Φ in polynomial basis representation, the (inverse) transformation matrices can be constructed using the algorithm presented in [22]. In this algorithm, using an exhaustive search, the transformation matrix is constructed using eight base elements in $GF(((2^2)^2)^2)$, i.e., $1, \xi, \xi^2, \ldots, \xi^7$, to which eight base elements of $GF(2^8)$ are mapped. We note that for each combination of ν and Φ, there exist eight possible (inverse) transformation matrices. These are constructed according to the base element ξ and the conjugates of this base element, i.e., ξ^{2^i}, $i = 1, 2, \ldots, 7$. In what follows, for each combination of ν and Φ, one of these possible matrices is considered. As suggested in [22], we have also used subexpression sharing for obtaining the low-complexity implementations for these matrices. We note that different (inverse) transformation matrices in normal basis are derived simply by reordering the columns.

The organization of this chapter is as follows. In Section 3.1, logic-gate optimizations for the inversions in $GF(2^4)$ within the S-boxes are presented. In Section 3.2, we present the results of our syntheses for different S-boxes. Power consumption derivations and comparisons of the S-boxes through a simulation-based method are presented in Section 3.3. The results presented in this chapter can also be found in [71] and [72].

3.1 Logic-gate Optimizations

In this section, first we present the architecture of the low-complexity S-box using normal basis. The previously presented low-complexity S-box using normal basis [23] is improved and the hardware complexity of the inversion in $GF(2^4)$ is reduced.

Let $\gamma = (\gamma_3, \gamma_2, \gamma_1, \gamma_0)$ be the input and $\theta = (\theta_3, \theta_2, \theta_1, \theta_0)$ be the output of an inverter in $GF(2^4)$ using normal basis. Then, the formulations for the inversion in $GF(2^4)$ using the low-complexity normal basis ($\Phi = \{10\}_2$) presented in [23] are obtained as follows

$$\begin{aligned}
\theta_3 &= \gamma_2\gamma_1\gamma_0 + \gamma_3\gamma_1 + \gamma_2\gamma_1 + \gamma_1 + \gamma_0,\\
\theta_2 &= \gamma_3\gamma_1\gamma_0 + \gamma_3\gamma_1 + \gamma_2\gamma_1 + \gamma_2\gamma_0 + \gamma_0,\\
\theta_1 &= \gamma_3\gamma_2\gamma_0 + \gamma_3\gamma_1 + \gamma_3\gamma_0 + \gamma_3 + \gamma_2,\\
\theta_0 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_1 + \gamma_3\gamma_0 + \gamma_2\gamma_0 + \gamma_2,
\end{aligned} \tag{3.1}$$

where, "+" represents the modulo-2 addition which uses an XOR gate in hardware.

Considering the formulations above, we present the following lemma for reaching a low-complexity architecture of an inverter in $GF(2^4)$.

Lemma 3.1 *The low-complexity formulations for the inversion in $GF(2^4)$ using normal basis can be written as follows.*

$$\begin{aligned}
\theta_3 &= (\gamma_2\gamma_1 \vee \gamma_0) + \overline{\gamma_3}\gamma_1,\\
\theta_2 &= \gamma_0(\gamma_1 \vee \overline{\gamma_2}) \vee \gamma_1(\gamma_2 + \gamma_3),\\
\theta_1 &= (\gamma_3\gamma_0 \vee \gamma_2) + \gamma_3\overline{\gamma_1},\\
\theta_0 &= \gamma_2(\gamma_3 \vee \overline{\gamma_0}) \vee \gamma_3(\gamma_1 + \gamma_0),
\end{aligned} \tag{3.2}$$

where, "+" and "$\vee$" represent the XOR and OR operations, respectively.

Proof For having low-complexity structures for θ_3 and θ_1, we use the fact that for two Boolean variables x and y, we have

$$x + y + xy = x \vee y. \tag{3.3}$$

Then, using θ_3 in (3.1) and considering $x = \gamma_2\gamma_1$ and $y = \gamma_0$ in (3.3), one can find

$$\begin{aligned}
\theta_3 &= \gamma_2\gamma_1\gamma_0 + \gamma_3\gamma_1 + \gamma_2\gamma_1 + \gamma_1 + \gamma_0\theta_3\\
&= (\gamma_2\gamma_1 \vee \gamma_0) + \gamma_1(\gamma_3 + 1)\\
&= (\gamma_2\gamma_1 \vee \gamma_0) + \overline{\gamma_3}\gamma_1.
\end{aligned} \tag{3.4}$$

Similarly, one can consider $x = \gamma_3\gamma_0$ and $y = \gamma_2$ in (3.3) for θ_1 in (3.1) to obtain

$$\begin{aligned}
\theta_1 &= \gamma_3\gamma_2\gamma_0 + \gamma_3\gamma_1 + \gamma_3\gamma_0 + \gamma_3 + \gamma_2\\
&= (\gamma_3\gamma_0 \vee \gamma_2) + \gamma_3(\gamma_1 + 1)\\
&= (\gamma_3\gamma_0 \vee \gamma_2) + \gamma_3\overline{\gamma_1}.
\end{aligned} \tag{3.5}$$

We now prove the formulations for θ_2 and θ_0. According to (3.1) and noting that $\gamma_i+1=\overline{\gamma_i}$, we obtain

$$\begin{aligned}\theta_2 &= \gamma_3\gamma_1\gamma_0+\gamma_3\gamma_1+\gamma_2\gamma_1+\gamma_2\gamma_0+\gamma_0\\ &= \gamma_1(\gamma_2+\gamma_3(\gamma_0+1))+\gamma_0(\gamma_2+1)\\ &= \gamma_1(\gamma_2+\gamma_3\overline{\gamma_0})+\gamma_0\overline{\gamma_2}. \end{aligned} \tag{3.6}$$

By the definition of the XOR we have $\gamma_2+\gamma_3\overline{\gamma_0}=\overline{\gamma_2}\gamma_3\overline{\gamma_0}\vee\gamma_2(\overline{\gamma_3}\vee\gamma_0)$. Then, (3.6) can be written as

$$\begin{aligned}\theta_2 &= \gamma_1(\overline{\gamma_2}\gamma_3\overline{\gamma_0}\vee\gamma_2(\overline{\gamma_3}\vee\gamma_0))+\gamma_0\overline{\gamma_2}\\ &= (\gamma_1\gamma_2\overline{\gamma_3}\vee\gamma_2\gamma_1\gamma_0\vee\gamma_3\overline{\gamma_2}\gamma_1\overline{\gamma_0})+\gamma_0\overline{\gamma_2}. \end{aligned} \tag{3.7}$$

It is noted that for having a low-complexity structure for θ_2, we use the fact that for two Boolean variables x and y, one can prove that

$$x+\overline{x}y=x\vee y. \tag{3.8}$$

Then, by distributing the XOR in (3.7) and using (3.8), the following terms are obtained

$$\gamma_3\overline{\gamma_2}\gamma_1\overline{\gamma_0}+\gamma_0\overline{\gamma_2}=\overline{\gamma_2}(\gamma_3\gamma_1\overline{\gamma_0}+\gamma_0)=\overline{\gamma_2}(\gamma_3\gamma_1\vee\gamma_0), \tag{3.9}$$

$$\gamma_2\gamma_1\gamma_0+\gamma_0\overline{\gamma_2}=\gamma_0(\gamma_2\gamma_1+\overline{\gamma_2})=\gamma_0(\overline{\gamma_2}\vee\gamma_1), \tag{3.10}$$

$$\overline{\gamma_3}\gamma_2\gamma_1+\gamma_0\overline{\gamma_2}=\overline{\gamma_3}\gamma_2\gamma_1\vee\gamma_0\overline{\gamma_2}. \tag{3.11}$$

Then, according to (3.7), by ORing (3.9)-(3.11) and noting that $\overline{\gamma_3}\gamma_2\gamma_1\vee\gamma_3\overline{\gamma_2}\gamma_1=\gamma_1(\gamma_2+\gamma_3)$, it is straightforward to obtain θ_2 in (3.2).

We obtain the following for θ_0

$$\begin{aligned}\theta_0 &= \gamma_3\gamma_2\gamma_1+\gamma_3\gamma_1+\gamma_3\gamma_0+\gamma_2\gamma_0+\gamma_2\\ &= \gamma_3(\gamma_0+\gamma_1(\gamma_2+1))+\gamma_2(\gamma_0+1)\\ &= \gamma_3(\gamma_0+\gamma_1\overline{\gamma_2})+\gamma_2\overline{\gamma_0}. \end{aligned} \tag{3.12}$$

By the definition of the XOR for $\gamma_0+\gamma_1\overline{\gamma_2}$, (3.12) can be written as

$$\begin{aligned}\theta_0 &= \gamma_3(\overline{\gamma_2}\gamma_1\overline{\gamma_0}\vee\gamma_0(\overline{\gamma_1}\vee\gamma_2))+\gamma_2\overline{\gamma_0}\\ &= (\gamma_3\gamma_0\overline{\gamma_1}\vee\gamma_3\gamma_2\gamma_0\vee\gamma_3\overline{\gamma_2}\gamma_1\overline{\gamma_0})+\gamma_2\overline{\gamma_0}. \end{aligned} \tag{3.13}$$

Then, by distributing the XOR in (3.13) and using (3.8), the following terms are obtained

$$\gamma_3\overline{\gamma_2}\gamma_1\overline{\gamma_0} + \gamma_2\overline{\gamma_0} = \overline{\gamma_0}(\gamma_3\gamma_1\overline{\gamma_2} + \gamma_2) = \overline{\gamma_0}(\gamma_3\gamma_1 \vee \gamma_2), \tag{3.14}$$

$$\gamma_3\gamma_2\gamma_0 + \gamma_2\overline{\gamma_0} = \gamma_2(\gamma_3\gamma_0 + \overline{\gamma_0}) = \gamma_2(\overline{\gamma_0} \vee \gamma_3), \tag{3.15}$$

$$\overline{\gamma_1}\gamma_3\gamma_0 + \gamma_2\overline{\gamma_0} = \overline{\gamma_1}\gamma_3\gamma_0 \vee \gamma_2\overline{\gamma_0}. \tag{3.16}$$

Then, according to (3.13), by ORing (3.14)-(3.16) and noting that $\overline{\gamma_0}\gamma_3\gamma_1 \vee \gamma_3\overline{\gamma_1}\gamma_0 = \gamma_3(\gamma_1 + \gamma_0)$, one can obtain θ_0 in (3.2). ∎

It is noted that for reaching a low-complexity architecture, the formulations in (3.2) can be implemented using only NOR, NAND, and XOR gates as follows

$$\begin{aligned}
\theta_3 &= XOR(NOR(NOR(\overline{\gamma_2}, \overline{\gamma_1}), \gamma_0), NAND(\overline{\gamma_3}, \gamma_1)),\\
\theta_2 &= NAND(NAND(\gamma_0, NAND(\gamma_2, \overline{\gamma_1})), NAND(\gamma_1, XOR(\gamma_2, \gamma_3))),\\
\theta_1 &= XOR(NOR(NOR(\overline{\gamma_3}, \overline{\gamma_0}), \gamma_2), NAND(\overline{\gamma_1}, \gamma_3)),\\
\theta_0 &= NAND(NAND(\gamma_2, NAND(\gamma_0, \overline{\gamma_3})), NAND(\gamma_3, XOR(\gamma_1, \gamma_0))).
\end{aligned} \tag{3.17}$$

In what presented above, the field inversion in $GF(2^4)$ of the most compact composite field in [23] has been modified to decrease its hardware complexity. This field uses normal basis with $\Phi = \{10\}_2$ and $\nu = \{0001\}_2$. Now, we consider polynomial basis to further optimize the S-boxes using polynomial basis. We present the following lemma through which the hardware complexity of the composite field inversion in $GF(2^4)$ is decreased. This is performed by presenting low-complexity formulations for the inversion in $GF(2^4)$ through logic-gate minimization. Moreover, these formulations are implemented using NAND, NOR, and XOR gates for reducing the complexity.

Lemma 3.2 *Let $\gamma = (\gamma_3, \gamma_2, \gamma_1, \gamma_0)$ be the input and $\theta = (\theta_3, \theta_2, \theta_1, \theta_0)$ be the output of an inverter in $GF(2^4)$. Then, the formulations for the low-complexity inversion in $GF(2^4)$ using polynomial basis with $\Phi = \{11\}_2$ are as follows:*

$$\begin{aligned}
\theta_3 &= \gamma_2\overline{\gamma_3\gamma_1} + \gamma_3\gamma_0,\\
\theta_2 &= \gamma_3\overline{\gamma_0} \vee \gamma_2(\gamma_3 \vee \gamma_1),\\
\theta_1 &= \gamma_2\overline{\gamma_0} \vee \gamma_3\overline{\gamma_1}\gamma_0 \vee \overline{\gamma_3}\gamma_1\overline{\gamma_2},\\
\theta_0 &= \gamma_3 \vee \gamma_1\overline{\gamma_0} \vee \overline{\gamma_2}\gamma_0\overline{\gamma_1} + \gamma_1(\gamma_2 \vee \gamma_3\overline{\gamma_0}).
\end{aligned} \tag{3.18}$$

Moreover, for $\Phi = \{10\}_2$*, one reaches the following:*

$$\begin{aligned}
\theta_3 &= \gamma_2\overline{\gamma_3\gamma_1} + \gamma_3\overline{\gamma_0},\\
\theta_2 &= \gamma_2\overline{\gamma_1} \vee \gamma_3(\gamma_2 \vee \gamma_0),\\
\theta_1 &= \gamma_3\gamma_1(\gamma_2 \vee \gamma_0) \vee \gamma_2\gamma_0 + \gamma_3 + \gamma_2 + \gamma_1,\\
\theta_0 &= \gamma_0 \vee \gamma_2\overline{\gamma_3} \vee \overline{\gamma_1}\gamma_3\overline{\gamma_2} + \gamma_2(\gamma_1 \vee \gamma_0\overline{\gamma_3}).
\end{aligned} \tag{3.19}$$

where "+" and "∨" represent the XOR and OR operations, respectively.

Proof For $\gamma = (\gamma_3, \gamma_2, \gamma_1, \gamma_0)$ as the input and $\theta = (\theta_3, \theta_2, \theta_1, \theta_0)$ as the output of an inverter in $GF(2^4)$, the formulations for the inversion in $GF(2^4)$ using the polynomial basis with $\Phi = \{11\}_2$ and $\Phi = \{10\}_2$ are obtained as follows, respectively, [24], [22]:

$$\begin{aligned}
\theta_3 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_0 + \gamma_2,\\
\theta_2 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_2\gamma_0 + \gamma_3\gamma_0 + \gamma_2\gamma_1 + \gamma_3,\\
\theta_1 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_1\gamma_0 + \gamma_3\gamma_0 + \gamma_3\gamma_1 + \gamma_2\gamma_0 +\\
&\gamma_2\gamma_1 + \gamma_2 + \gamma_1,\\
\theta_0 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_2\gamma_0 + \gamma_3\gamma_1\gamma_0 + \gamma_2\gamma_1\gamma_0 +\\
&\gamma_2\gamma_0 + \gamma_3\gamma_0 + \gamma_2\gamma_1 + \gamma_3 + \gamma_1 + \gamma_0,
\end{aligned} \tag{3.20}$$

$$\begin{aligned}
\theta_3 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_0 + \gamma_3 + \gamma_2,\\
\theta_2 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_2\gamma_0 + \gamma_3\gamma_0 + \gamma_2\gamma_1 + \gamma_2,\\
\theta_1 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_1\gamma_0 + \gamma_2\gamma_0 + \gamma_3 + \gamma_2 + \gamma_1,\\
\theta_0 &= \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_2\gamma_0 + \gamma_3\gamma_1\gamma_0 + \gamma_2\gamma_1\gamma_0 + \gamma_3\gamma_1 +\\
&\gamma_3\gamma_0 + \gamma_2\gamma_1 + \gamma_2 + \gamma_1 + \gamma_0.
\end{aligned} \tag{3.21}$$

One can obtain θ_3-θ_0 in (3.18) and (3.19) from those of (3.20) and (3.21), respectively. For performing this, we note that $\gamma_i + 1 = \overline{\gamma_i}$ and $\gamma_i + \gamma_j + \gamma_i\gamma_j = \gamma_i \vee \gamma_j$. For instance, now obtain θ_3 in (3.18) from that of (3.20) as $\theta_3 = \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_0 + \gamma_2 = \gamma_2(\gamma_3\gamma_1 + 1) + \gamma_3\gamma_0 = \gamma_2\overline{\gamma_3\gamma_1} + \gamma_3\gamma_0$. Using similar methods, one can obtain (3.18). As another example, one can obtain θ_3 in (3.19) from that of (3.21) as $\theta_3 = \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_0 + \gamma_3 + \gamma_2 = \gamma_2(\gamma_3\gamma_1 + 1) + \gamma_3(\gamma_0 + 1) = \gamma_2\overline{\gamma_3\gamma_1} + \gamma_3\overline{\gamma_0}$. By verifying the 16 combinations of the input γ, same results are obtained for (3.18) and (3.20) ((3.19) and (3.21)). ∎

Table 3.1: Evaluation of the performance metrics of the S-boxes on ASIC using the STM 65-nm CMOS standard technology.

Structure	Specifications		Area		Delay [Freq.]	Thro'put	Effic.
	Φ	ν	(μm^2)	GE[a]	(ns) [MHz]	(Gbps)	($\frac{Mbps}{\mu m^2}$)
Polynomial basis	10	1000 [30], [24][b]	525.2	252.5	1.31 [763]	6.1	11.6
		1000 (proposed, (3.19))	518.9	249.4	1.15 [869]	7.0	**13.5**[3]
		1001	537.2	258.2	1.43 [699]	5.6	10.4
		1010	535.6	257.5	1.36 [735]	5.9	11.0
		1011	540.8	260.0	1.43 [699]	5.6	10.3
		1100 [13], [20], [22][c]	540.3	259.7	1.37 [730]	5.8	10.8
		1101	548.6	263.7	1.34 [746]	6.0	10.9
		1110	524.7	252.3	1.40 [714]	5.7	10.9
		1110 (proposed, (3.19))	**510.2**[3]	245.2	1.25 [800]	6.4	12.5
		1111	535.6	257.5	1.40 [714]	5.7	10.6
	11	1000	528.3	253.9	1.39 [719]	5.8	10.9
		1000 (proposed, (3.18))	516.3	248.2	**1.11 [900]**[3]	**7.2**[3]	**13.9**[2]
		1001	534.6	257.0	1.56 [641]	5.1	9.6
		1010 [24][b]	519.0	249.5	1.42 [704]	5.6	10.9
		1010 (proposed, (3.18))	**498.4**[2]	239.6	**1.11 [900]**[3]	**7.2**[3]	**14.4**[1]
		1011	531.0	255.2	1.45 [690]	5.5	10.4
		1100	548.1	263.5	1.49 [671]	5.4	9.8
		1101	546.5	262.7	1.42 [704]	5.6	9.8
		1110	542.8	260.9	1.52 [657]	5.3	9.7
		1111	542.9	261.0	1.52 [657]	5.3	9.7
Normal basis	10	0001 [23][b]	569.4	273.7	1.59 [629]	5.0	8.8
		0001 [23] (3.17)[d]	511.7	246.0	1.45 [690]	5.5	10.7
		0010	564.2	271.2	1.42 [704]	5.6	10.0
		0100	576.7	277.2	1.57 [637]	5.1	8.8
		1000	579.3	278.5	1.46 [685]	5.5	9.4
		0111	575.1	276.5	1.56 [641]	5.1	8.9
		1011	589.2	283.2	1.55 [645]	5.2	8.7
		1101	572.0	275.0	1.60 [625]	5.0	8.7
		1110	588.6	282.0	1.57 [637]	5.1	8.6
	01	0001	564.2	272.2	1.58 [633]	5.1	9.0
		0010	577.2	277.5	1.51 [662]	5.3	9.2
		0100	564.2	271.2	1.59 [629]	5.0	8.9
		1000	570.9	274.4	1.58 [633]	5.1	8.9
		0111	583.9	280.7	1.49 [671]	5.4	9.2
		1011	572.5	275.2	1.58 [633]	5.1	8.9
		1101	583.9	280.7	1.48 [676]	5.4	9.3
		1110	571.9	274.9	1.59 [629]	5.0	8.8
Normal basis	-	0001 [60][e]	**403.2**[1]	193.8	1.64 [610]	4.9	12.1
Polynomial basis	-	1110 [27][f]	554.3	266.5	1.26 [794]	6.4	11.5
Mixed basis [62][g]	-	-	571.1	274.5	1.30 [769]	6.2	10.9
LUT/ROM [77], [59][h]	-	-	1,407.1	676.4	**0.60 [1666]**[1]	**13.3**[1]	9.5
LUT/ROM-MI [59][i]	-	-	1,434.6	689.4	**0.68 [1470]**[2]	**11.8**[2]	8.2

1, 2, and 3 are the best cases for each performance metric.

[a] Gate equivalent in terms of two-input NAND.

[b] Among all fields considered, the presented composite field has the least hardware complexities in terms of logic-gate counts.

[c] These are some works in which this composite field is used.

[d] The hardware complexity of this composite field has been improved by (3.17).

[e] This implementation is based on a minimization method at the expense of more timing complexity.

[f] This architecture is based on the composite field $GF((2^4)^2)$.

[g] Has been presented very recently based on mixed polynomial and normal bases and only focuses on decreasing the critical path delay.

[h] Using synthesized ROM-based LUTs.

[i] LUTs for the multiplicative inversion (MI) and logic gates for the affine transformation.

In what follows, we evaluate and compare the performance metrics of different S-boxes. The presented results confirm efficiency increase using (3.18) and (3.19).

3.2 Area and Delay Evaluations

In the following, we evaluate and compare the areas, delays, throughputs, and efficiencies of different S-boxes, including the ones presented in [13], [20], [22], [23], [24], [27], [30], [60], and [62]. It is noted that the implementation in [61] is for the inversion in $GF(2^4)$ and does not provide the entire S-box architecture.

Using MATLAB® [76], we have derived the low-complexity transformation and mixed inverse and affine transformation matrices for the syntheses. We have used the STM 65-nm CMOS standard technology and CORE65LPSVT standard cell library [74]. This library is optimized for using in low-power applications. The nominal junction temperature is 25 °C and VHDL has been used as the design entry to the Synopsys® Design Vision® [73]. We note that the presented results are post synthesis and do not consider the post layout routing.

The results of our syntheses are presented in Table 3.1. As seen in this table, for different S-boxes, the areas (in terms of μm^2), critical path delays (in terms of ns), maximum working frequencies (in terms of MHz), throughputs (in terms of Gbps), and efficiencies (in terms of $\frac{Mbps}{\mu m^2}$) have been obtained. According to the STM 65-nm standard cell library information, the lowest and nominal drive strength for the cells is two. It is noted that the area of a NAND gate in the utilized STM 65-nm CMOS library for the drive strength of two is $2.08\mu m^2$. Then, using this area, we have also provided the gate equivalent (GE) measure for different S-boxes in the table. Note that if we increase the area effort, lower areas are usually achieved mostly at the expense of more delay overhead.

Memory macros tend to be expensive in hardware for implementing the S-boxes, resulting in high hardware complexity and power consumption. Therefore, this implementation is not considered in this chapter. We have considered two different methods of realization of the LUT S-boxes. In these methods, read-only LUTs are used for implementing the S-box, see, for instance, [77] and the hw-lut/hybrid-lut architectures in [59]. This allows us to logic-optimize the S-box architecture by synthesis of hardware

description languages, leading to low-area implementations. In the first method (denoted by LUT/ROM), the entire S-box is implemented using LUTs. Moreover, we consider the S-boxes in which only the multiplicative inversion (MI) in $GF(2^8)$ is implemented using LUTs and the affine transformation is implemented separately (denoted by LUT/ROM-MI). This enables the designers to share the multiplicative inversion in $GF(2^8)$ for the S-box and the inverse S-box in the merged structures.

In some of the previous works such as [20], [23], [27], and [30], the area of the S-box has been presented in terms of GE. For instance, in [20] and [30], the areas of the implemented S-boxes have been provided as 294 GE and 272 GE using $0.11\mu m$ and $0.18\mu m$ technologies, respectively. Based on the information of the cell library in a $0.18\mu m$ technology, the gate count of the S-box has been converted to gate equivalent as 180 GE in [23]. We note that the result in [23] (unlike those in [20], [27], [30], and this chapter) is the direct conversion of the gate count (without synthesis) to GE. In addition, the conversion factor of 1.75 has been used in [23] for obtaining the GE for XOR/XNOR and MUX21. However, in the cell library used in this chapter, these conversion factors are 2.25 and 2, respectively. Another factor causing the reported areas in terms of GE in different works to vary is the type of the synthesis tools used and the map effort specified.

Using (3.18) and (3.19) of Lemma 3.1, we have also presented the results of the logic-gate optimized S-boxes in Table 3.1. Specifically, we have used (3.18) and (3.19) for two most compact S-boxes using polynomial basis for $\Phi = \{11\}_2$ and $\Phi = \{10\}_2$ in Table 3.1. It is also noted that for each of the evaluated performance metrics, the three best cases among different results for the S-boxes have been marked with superscripts 1, 2, and 3. As shown in Table 3.1, the areas for the composite field S-boxes range from 403.2-589.2 μm^2 (difference of 46.1%), the working frequencies from 625-900 MHz (difference of 44.0%), the throughputs from 5.0-7.2 Gbps (difference of 44.0%), and the efficiencies from 8.6-14.4 $\frac{Mbps}{\mu m^2}$ (difference of 67.4%).

As seen in Table 3.1, the S-boxes using LUTs (last two rows) are the fastest S-boxes. However, their efficiencies are not the highest among other S-boxes in Table 3.1. Among the composite field S-boxes, the one using normal basis presented in [60] is the most compact one (see the area column in Table 3.1). However, it has the worst working frequency and throughput. The S-boxes using polynomial basis (optimized using (3.18)) have the highest frequency and throughput among the composite field S-boxes. Finally,

the highest efficiency (see the last column in Table 3.1) is obtained for the one using polynomial basis with $\Phi = \{11\}_2$ and $\nu = \{1010\}_2$ (optimized using (3.18)).

3.3 Power Consumptions and Comparisons

In the following, the power consumption results for different S-boxes are presented. We have derived the power consumptions of the S-boxes within the AES through a simulation-based analysis method. In what follows, we present the power derivation method as well as the results of our analysis and comparison.

3.3.1 Power Derivation Method

We use VHDL as the design entry to the Synopsys® Design Vision®. After obtaining the gate-level netlists of the S-boxes, timing simulations are performed using ModelSim® SE 6.2*d* [75]. The testbench used for timing simulations covers all the $256 \times 255 = 65280$ possible transitions for the 8-bit input of the S-box. This exhaustive input pattern assertion includes all the possible transitions between each two different pairs of the possible 256 inputs. Then, for each and every S-box, the results of the switching activities of all internal nodes have been logged in the VCD (Value Change Dump) files. We have set the resolution of the timing simulations to high so that the VCD files contain the switching activities of glitches (dynamic hazards) occurring in the logic gates. Then, as the final step, the power consumption of the circuit is computed from the VCD logs, gate-level netlists, cell information, and parasitics of the target ASIC library. We have utilized the Synopsys® PrimeTime® PX [73] to obtain the average power (including net switching power, cell internal power, and cell leakage power), peak and instantaneous power consumption details. It is noteworthy that the power consumption results are for the working frequency of 50 MHz and for the high resolutions for both timing and power consumption.

3.3.2 Analysis and Comparison

The results of our simulation-based power computations are presented in Table 3.2. As depicted in this table, for different S-boxes, we have derived the average power (in terms of μW), peak power (in terms of mW), and the input pattern transition for which the

peak power happens. As shown in Table 3.2, the average powers for the composite field S-boxes range from 44.39-58.96 μW (difference of 32.8%) and the peak powers from 1.013-1.324 mW (difference of 30.7%). We have also marked (with superscripts 1, 2, and 3) the three cases for which the lowest power consumptions are achieved.

Comparing the results in Tables 3.1 and 3.2 shows that generally and with few exceptions, the S-boxes with more hardware complexities consume more power. As seen in Table 3.2, the highest and lowest average power consumptions are achieved for the LUT-based (using memories) S-box and the normal basis S-box presented in [60], respectively. Based on our results in Table 3.1, these two S-boxes have the highest and lowest hardware complexities, respectively. On the other hand, according to the results of Table 3.1, the normal basis S-box presented in [60] has the highest timing complexity among the composite field S-boxes.

The transitions of the inputs of the S-boxes when the peak powers occur have been also shown in Table 3.2. As shown in this table, most of the peak powers occur when the S-box input changes to the all-zero input.

Table 3.2: Evaluation of the power consumptions of the S-boxes on ASIC using the STM 65-nm CMOS standard technology and the Synopsys® PrimeTime® PX [73].

Structure	Specification		Average[a]	Peak[b]	
	Φ	ν	(μW)	(mW)	Transition
Polynomial basis	10	1000 [30], [24][c]	54.99	1.283	$78 \rightarrow 00$
		1001	55.77	1.184	$1C \rightarrow 00$
		1010	54.91	1.165	$F4 \rightarrow 58$
		1011	56.45	1.262	$79 \rightarrow 00$
		1100 [13], [20], [22][d]	55.98	1.283	$C0 \rightarrow 00$
		1101	56.63	1.324	$D5 \rightarrow 01$
		1110	55.28	1.313	$B5 \rightarrow 00$
		1111	55.87	1.161	$C3 \rightarrow 07$
	11	1000	54.69	**1.134**[2]	$27 \rightarrow 00$
		1000 (proposed, using (3.18))	54.15	1.188	$B8 \rightarrow 00$
		1001	55.44	1.214	$54 \rightarrow 01$
		1010 [30][c]	54.12	**1.145**[3]	$71 \rightarrow 00$
		1010 (proposed, using (3.18))	**53.78**[2]	1.229	$C9 \rightarrow 00$
		1011	54.80	1.218	$55 \rightarrow 00$
		1100	55.13	1.178	$D7 \rightarrow 00$
		1101	56.51	1.244	$BA \rightarrow 00$
		1110	55.91	1.239	$3B \rightarrow 00$
		1111	55.40	1.185	$9C \rightarrow 00$
Normal basis	10	0001 [23][c]	58.02	1.268	$46 \rightarrow 00$
		0001 [23] (3.17)[e]	**54.03**[3]	1.189	$46 \rightarrow 0A$
		0010	58.51	1.291	$41 \rightarrow 00$
		0100	58.03	1.283	$46 \rightarrow 00$
		1000	58.15	1.290	$41 \rightarrow 00$
		0111	58.79	1.299	$F2 \rightarrow 00$
		1011	58.35	1.247	$91 \rightarrow 00$
		1101	58.81	1.300	$73 \rightarrow 00$
		1110	58.96	1.309	$91 \rightarrow 00$
	01	0001	58.19	1.323	$68 \rightarrow 00$
		0010	58.07	1.284	$92 \rightarrow 00$
		0100	57.90	1.292	$68 \rightarrow 00$
		1000	58.17	1.292	$92 \rightarrow 00$
		0111	58.54	1.291	$43 \rightarrow 00$
		1011	57.88	1.231	$E7 \rightarrow 00$
		1101	58.70	1.282	$42 \rightarrow 00$
		1110	58.23	1.246	$5B \rightarrow 00$
Normal basis	-	0001 [60][f]	**44.39**[1]	**1.013**[1]	$27 \rightarrow 00$
Polynomial basis	-	1110 [27][g]	55.48	1.208	$22 \rightarrow 00$
Mixed basis [62]	-	-	58.06	1.242	$46 \rightarrow 00$
LUT/ROM [77], [59]	-	-	63.18	1.337	$91 \rightarrow 00$
LUT/ROM-MI [59]	-	-	66.20	1.344	$91 \rightarrow 00$

1, 2, and 3 are the best cases for each performance metric.

[a] Includes net switching, cell internal, and cell leakage power.

[b] Obtained from the instantaneous power values for each case.

[c] Among all fields considered, the presented composite field has the least hardware complexities in terms of logic-gate counts.

[d] These are some works in which this composite field is used.

[e] The power consumption of this composite field has been improved using (3.17).

[f] The lowest-power yet the slowest composite field S-box.

[g] This architecture is based on the composite field $GF((2^4)^2)$.

Chapter 4

A Lightweight Fault Detection Scheme for the (Inverse) S-box Using Composite Fields

IN this chapter, we present a lightweight concurrent fault detection scheme for the AES. In the proposed approach, the composite field S-box and inverse S-box are divided into blocks and the predicted parities of these blocks are obtained. Through exhaustive searches among all available composite fields, we find the optimum solutions for the least overhead parity-based fault detection structures. Moreover, through our error injection simulations for one S-box (resp. inverse S-box), we show that the total error coverage of 99.998% for 16 S-boxes (resp. inverse S-boxes) can be achieved. Finally, it is shown that both the ASIC and FPGA implementations of the fault detection structures using the obtained optimum composite fields, have better hardware and time complexities compared to their counterparts.

We present a low-cost parity-based fault detection scheme for the S-box and the inverse S-box using composite fields. In the presented approach, for increasing the error coverage, the predicted parities of the five blocks of the S-box and the inverse S-box are obtained (three predicted parities for the multiplicative inversion and two for the transformation and affine matrices). It is interesting to note that the cost of our multi-bit parity prediction approach is lower than its counterparts which use single-bit parity. It also has higher error coverage than the approaches using single-bit parities. We implement both the proposed fault detection S-box and inverse S-box and other counterparts. Our both ASIC and FPGA implementation results show that compared to the approaches presented in [52] and [53], the complexities of the proposed fault detection scheme are

lower. Through exhaustive searches, we obtain the least area and delay overhead fault detection structures for the optimum composite fields using both polynomial basis and normal basis. The proposed fault detection scheme is simulated and we show that the error coverages of 99.998% for 16 S-boxes (resp. inverse S-boxes) can be obtained. Finally, we have implemented the fault detection hardware structures of the AES using both 0.18μ CMOS technology and on Xilinx® Virtex™-II Pro FPGA [80]. It is shown that the fault detection scheme using the optimum polynomial and normal bases have lower complexities than those using other composite fields for both with and without fault detection capability.

The organization of this chapter is as follows. In Section 4.1, some preliminaries related to the composite fields are presented. The proposed fault detection approach for the S-box and the inverse S-box is presented in Section 4.2. Furthermore, the time and hardware complexities analysis is preformed in this section. In Section 4.3, the results of the simulations of the proposed approach are presented; through which, the fault detection capabilities are derived. In Section 4.4, through FPGA and ASIC implementations, the performance metrics of the proposed fault detection scheme and the previously reported ones are compared. The results presented in this chapter can also be found in [78] and [79].

4.1 Some Notes on Polynomial and Normal Bases

The composite fields can be represented using normal basis [23] or polynomial basis [18], [20], [21], [22]. The S-box and inverse S-box for the polynomial and normal bases are shown in Figs. 4.1 and 4.2, respectively. As shown in these figures, for the S-box using polynomial basis (resp. normal basis), the transformation matrix $\mathbf{\Psi}$ (resp. $\mathbf{\Psi'}$[1]) transforms a field element X in the binary field $GF(2^8)$ to the corresponding representation in the composite fields $GF(2^8)/GF(2^4)$. It is noted that the result of $X = \eta_h u + \eta_l$ in Fig. 4.1 (resp. $X = \eta'_h u^{16} + \eta'_l u$ in Fig. 4.2) is obtained using the irreducible polynomial of $u^2 + \tau u + \nu$ (resp. $u^2 + \tau' u + \nu'$).

The multiplicative inversion in Fig. 4.1 consists of composite-field multiplications, additions and an inversion in the sub-field $GF(2^4)$ over $GF(2)/x^4 + x + 1$ [21]. The

[1]We use prime notations for the composite fields using normal basis.

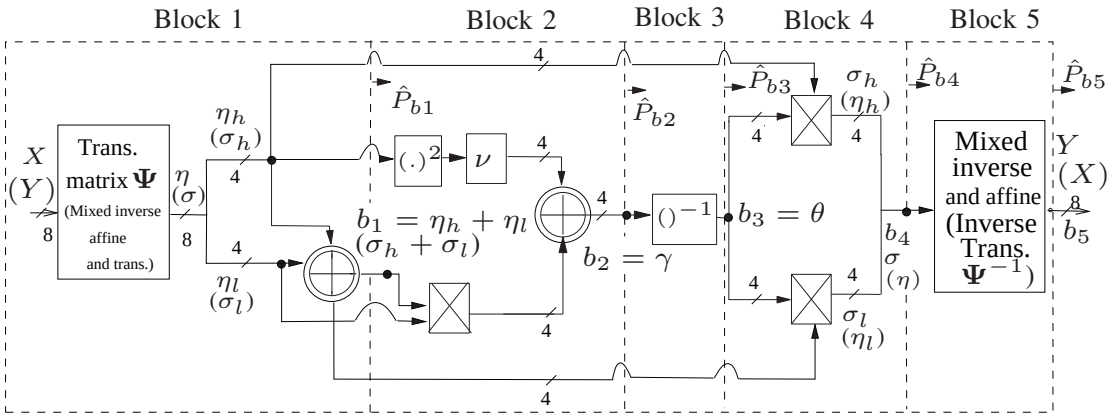


Figure 4.1: The S-box (the inverse S-box) using composite fields and polynomial basis [20] and their fault detection blocks.

decomposition can be further applied to represent $GF(2^4)$ as a linear polynomial over $GF(2^2)$ and then $GF(2)$ using the irreducible polynomials of $v^2+\Omega v+\Phi$ and w^2+w+1, respectively. As a result, it is understood that the implementation of the multiplicative inversion can be performed using the field represented by $GF((2^4)^2)$, see for example, [18] and [21], or the field represented by $GF(((2^2)^2)^2)$ and has been used in the literature, see for example [20] and [22]. Finally, as seen in Fig. 4.2 for normal basis, the decomposition is performed using the irreducible polynomials of $v^2 + \Omega' v + \Phi'$ and $w^2 + w + 1$.

For calculating the multiplicative inversion, the most efficient choice is to let $\Omega = \tau = 1$ ($\Omega' = \tau' = 1$) in the above irreducible polynomials [23]. Then, we have the following for the multiplicative inversion of the S-box using polynomial basis (Fig. 4.1) and normal basis (Fig. 4.2), respectively, [20], [23]

$$(\eta_h u + \eta_l)^{-1} = \tag{4.1}$$
$$[((\eta_h + \eta_l)\eta_l + {\eta_h}^2\nu)^{-1}\eta_h]u + ((\eta_h + \eta_l)\eta_l + {\eta_h}^2\nu)^{-1}(\eta_h + \eta_l),$$

$$(\eta'_h u^{16} + \eta'_l u)^{-1} = \tag{4.2}$$
$$[(\eta'_h\eta'_l + ({\eta'_h}^2 + {\eta'_l}^2)\nu')^{-1}\eta'_h]u^{16} + [(\eta'_h\eta'_l + ({\eta'_h}^2 + {\eta'_l}^2)\nu')^{-1}\eta'_l]u.$$

It is noted that one can replace η (η') with σ (σ') to obtain (4.1) and (4.2) for the inverse S-box. In the next section, we propose the low-cost fault detection scheme for the S-box and the inverse S-box.

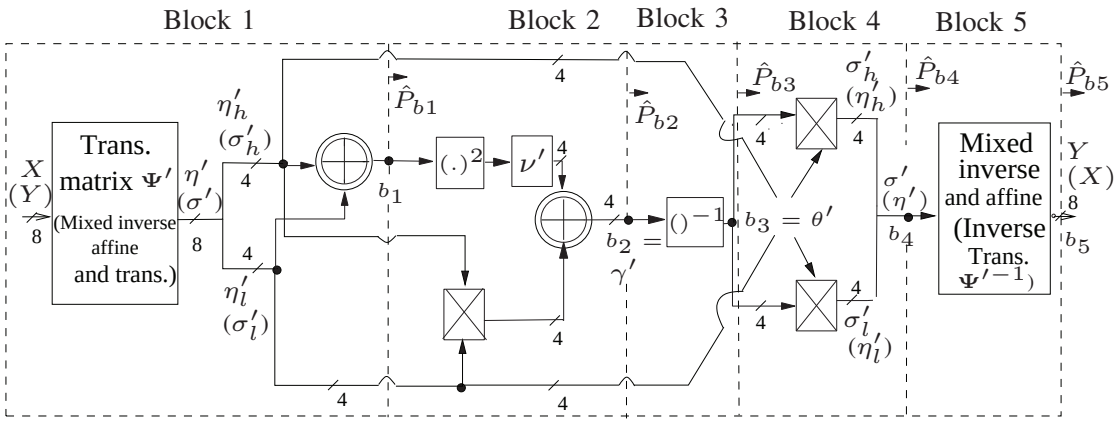


Figure 4.2: The S-box (the inverse S-box) using composite fields and normal basis [23] and their fault detection blocks.

4.2 Fault Detection Scheme

To obtain low-overhead parity prediction, we have divided the S-box and the inverse S-box into 5 blocks as shown in Figs. 4.1 and 4.2. In these figures, the modulo-2 additions, consisting of 4 XOR gates, are shown by two concentric circles with a plus inside. Furthermore, the multiplications in $GF(2^4)$ are shown by rectangles with crosses inside. Let b_i be the output of the block i denoted by dots in Figs. 4.1 and 4.2 for the S-box. As seen in Fig. 4.1, $b_1 = \eta_h + \eta_l, b_2 = \gamma, b_3 = \theta, b_4 = \sigma$, and $b_5 = Y$. Similarly, from Fig. 4.2, $b_1 = \eta'_h + \eta'_l, b_2 = \gamma', b_3 = \theta', b_4 = \sigma'$, and $b_5 = Y$. One can replace η (η') with σ (σ') and X with Y for the inverse S-box. In the following, we have exhaustively searched for the least overhead parity predictions of these blocks denoted by $\hat{P}_{b1}$-$\hat{P}_{b5}$ in Figs. 4.1 and 4.2.

4.2.1 The S-box and the Inverse S-box Using Polynomial Basis

The implementation complexities of different blocks of the S-box and the inverse S-box and those for their predicted parities are dependent on the choice of the coefficients $\nu \in GF(2^4)$ and $\Phi \in GF(2^2)$ in the irreducible polynomials $u^2 + u + \nu$ and $v^2 + v + \Phi$ used for the composite fields. Our goal in the following is to find $\nu \in GF(2^4)$ and $\Phi \in GF(2^2)$ for the composite fields $GF(((2^2)^2)^2)$ and $\nu \in GF(2^4)$ for the composite fields $GF((2^4)^2)$ so that the area complexity of the entire fault detection implementations becomes optimum. According to [24], 16 the possible combinations for $\nu \in GF(2^4)$ and

$\Phi \in GF(2^2)$ exist. Moreover, for the composite fields $GF((2^4)^2)$, we have exhaustively searched and have found the possible choices for ν making the polynomial $x^2 + x + \nu$ irreducible. These parameters determine the complexities of some blocks as explained below.

Blocks 1 and 5: Based on the possible values of ν and Φ in $GF(((2^2)^2)^2)$ (ν in $GF((2^4)^2)$), the transformation matrices in Fig. 4.1 in blocks 1 and 5 of the S-box and the inverse S-box can be constructed using the algorithm presented in [24]. Using an exhaustive search, eight base elements in $GF(((2^2)^2)^2)$ (or $GF((2^4)^2)$) to which eight base elements of $GF(2^8)$ are mapped, are found to construct the transformation matrix.

In [81], the Hamming weights, i.e., the number of non-zero elements, of the transformation matrices for the case $\Phi = \{10\}_2$ and different values of ν in $GF(((2^2)^2)^2)$ are obtained. It is noted that in [24], instead of considering the Hamming weights, subexpression sharing is suggested for obtaining the low-complexity implementations for the S-box. Here, we have also considered these transformation matrices for $\Phi = \{11\}_2$ as well as the transformation matrices for the inverse S-box for different values of ν and Φ and have derived their area and delay complexities. Moreover, the gate count and the critical path delay for blocks 1 and 5 and their predicted parities, i.e., $\hat{P}_{b1}$ and $\hat{P}_{b5}$, of the S-box and the inverse S-box in $GF((2^4)^2)$ have been obtained.

Blocks 2 and 4: As shown in Fig. 4.1, block 2 of the S-box and the inverse S-box consists of a multiplication, an addition, a squaring and a multiplication by constant ν in $GF((2^2)^2)$. We present the following lemma for deriving the predicted parity of the multiplication in $GF((2^2)^2)$, using which the predicted parities of blocks 2 and 4 in Fig. 4.1 are obtained.

Lemma 4.1 *Let $\lambda = (\lambda_3, \lambda_2, \lambda_1, \lambda_0)$ and $\delta = (\delta_3, \delta_2, \delta_1, \delta_0)$ be the inputs of a multiplier in $GF((2^2)^2)$. The predicted parities of the result of the multiplication of λ and δ in $GF((2^2)^2)$ for $\Phi = \{10\}_2$ and $\Phi = \{11\}_2$ are as follows, respectively,*

$$\begin{aligned}\hat{P}_\pi &= \lambda_3(\delta_3 + \delta_2 + \delta_0) + \lambda_2(\delta_3 + \delta_1 + \delta_0) + \lambda_1(\delta_2 + \delta_0) \\ &+ \lambda_0(\delta_3 + \delta_2 + \delta_1 + \delta_0) \qquad if \quad \Phi = \{10\}_2. \end{aligned} \tag{4.3}$$

$$\begin{aligned}\hat{P}_\pi &= \lambda_3(\delta_3 + \delta_0) + \lambda_2(\delta_2 + \delta_1 + \delta_0) + \lambda_1(\delta_2 + \delta_0) \\ &+ \lambda_0(\delta_3 + \delta_2 + \delta_1 + \delta_0) \qquad if \quad \Phi = \{11\}_2. \end{aligned} \tag{4.4}$$

Proof One can perform modulo-2 addition of the coordinates of the result of the multiplication over $GF((2^2)^2)$ [20]. Then, by reordering and factoring of the result for $\Phi = \{10\}_2$ and $\Phi = \{11\}_2$, the predicted parities in (4.3) and (4.4) are obtained. ∎

The predicted parity of block 2 of the S-box and the inverse S-box, i.e., $\hat{P}_{b2} = \hat{P}_{\eta_h{}^2\nu} + \hat{P}_{(\eta_h+\eta_l)\eta_l}$ in Fig. 4.1, depends on the choice of the coefficients $\nu \in GF((2^2)^2)$ and $\Phi \in GF(2^2)$. Using Lemma 4.1, we have derived the complexity of the predicted parity of block 2 for these coefficients. Furthermore, for block 4 in Fig. 4.1, which consists of two multiplications in $GF((2^2)^2)$, one can also use Lemma 4.1 to derive the predicted parity. For block 2 of the S-box (resp. the inverse S-box) over $GF((2^4)^2)$ in Fig. 4.1, only the multiplication by constant ν is affected for different values of νs. For this block, we have exhaustively searched for and obtained the optimum implementation for different values of νs. Moreover, block 4 in Fig. 4.1 is independent of the value of ν. Therefore, the complexity of the predicted parity for this block is the same for all possible νs.

Block 3: We present the following theorem for block 3 of the S-box and the inverse S-box over $GF((2^2)^2)$ in Fig. 4.1.

Theorem 4.1 *Let $\gamma = (\gamma_3, \gamma_2, \gamma_1, \gamma_0)$ be the input and $\theta = (\theta_3, \theta_2, \theta_1, \theta_0)$ be the output of an inverter in $GF((2^2)^2)$. The predicted parities of the result of the inversion in $GF((2^2)^2)$, i.e., $\hat{P}_{b3}$, for $\Phi = \{10\}_2$ and $\Phi = \{11\}_2$ are as follows, respectively,*

$$\hat{P}_\theta = (\overline{\gamma}_2 \vee \gamma_1)\gamma_0 + (\gamma_1 + \gamma_0)\gamma_3 \qquad if \quad \Phi = \{10\}_2, \tag{4.5}$$

$$\hat{P}_\theta = (\gamma_2\gamma_1 \vee \gamma_0) + \gamma_3\gamma_1 \qquad if \quad \Phi = \{11\}_2, \tag{4.6}$$

where, $\vee$ represents an OR operation.

Proof By Modulo-2 addition of the coordinates of the result of the inversion in $GF((2^2)^2)$ for $\Phi = \{10\}_2$ in [20], one can obtain the predicted parity of θ as $\hat{P}_\theta = \gamma_2\gamma_0 + \gamma_2\gamma_1\gamma_0 + \gamma_3\gamma_1 + \gamma_0 + \gamma_3\gamma_0 = \gamma_0(\gamma_2(\gamma_1 + 1) + 1) + \gamma_3(\gamma_1 + \gamma_0)$. By noting that $\gamma_1 + 1 = \overline{\gamma}_1$ and $\overline{\gamma_2\overline{\gamma}_1} = \overline{\gamma}_2 \vee \gamma_1$, one can reach (4.5). Moreover, by XORing the result for $\Phi = \{11\}_2$, $\hat{P}_\theta$ is obtained as $\hat{P}_\theta = \gamma_3\gamma_1 + \gamma_2\gamma_1\gamma_0 + \gamma_2\gamma_1 + \gamma_0$. Noting that $\gamma_2\gamma_1\gamma_0 + \gamma_2\gamma_1 + \gamma_0 = \gamma_2\gamma_1 \vee \gamma_0$, one can simplify $\hat{P}_\theta$ to reach (4.6) and the proof is complete. ∎

It is noted that the inversion in $GF(2^4)$ in Fig. 4.1 is independent of the value of ν. Therefore, the complexity of the predicted parity for this block is the same for any possible νs.

Considering the discussions presented in this section for different combinations of ν and Φ for polynomial basis, we present the following for the optimum parity predictions.

Proposition 4.1 *The fault detection S-box using composite fields $GF(((2^2)^2)^2)$ has the least area complexity for $\Phi = \{11\}_2$ and $\nu = \{1010\}_2$. For this optimum S-box (PB_1), we have the following predicted parities for the 5 blocks in Fig. 4.1: $\hat{P}_{b1} = x_0, \hat{P}_{b2} = \eta_3(\overline{\eta_7} + \eta_4) + \eta_2(\overline{\eta_7} + P_{\eta_h}) + \eta_1(\eta_6 + \eta_4) + \eta_0\overline{P_{\eta_h}} + \eta_6 + \eta_7, \hat{P}_{b3} = (\gamma_2\gamma_1 \vee \gamma_0) + \gamma_1\gamma_3, \hat{P}_{b4} = \eta_3(\theta_3 + \theta_0) + \eta_2(P_\Theta + \theta_3) + \eta_1(\theta_2 + \theta_0) + \eta_0 P_\Theta, \hat{P}_{b5} = \sigma_7 + \sigma_5 + \sigma_3 + \sigma_2 + \sigma_0$, where, $P_{\eta_h} = \eta_7 + \eta_6 + \eta_5 + \eta_4$ and $P_\Theta = \theta_3 + \theta_2 + \theta_1 + \theta_0$. Additionally, among all the possible values for ν using composite fields $GF((2^4)^2)$, $\nu = \{1010\}_2$ yields to the least-complexity architecture for the optimum S-box (PB_2), respectively. Then, for the S-box we have: $\hat{P}_{b1} = x_7 + x_0, \hat{P}_{b2} = \eta_3\eta_4 + \eta_2(\eta_5 + \eta_4) + \eta_1(\overline{P_{\eta_h}} + \eta_7) + \eta_0\overline{P_{\eta_h}} + P_{\eta_h} + \eta_4, \hat{P}_{b3} = \overline{\gamma_3}\gamma_2\overline{\gamma_0} + \gamma_0(\overline{\gamma_1} \vee \overline{(\gamma_2 + \gamma_3)}), \hat{P}_{b4} = \eta_3\theta_0 + \eta_2(\theta_1 + \theta_0) + \eta_1(P_\Theta + \theta_3) + \eta_0 P_\Theta, \hat{P}_{b5} = \sigma_4 + \sigma_3 + \sigma_2 + \sigma_1 + \sigma_0$.*

Furthermore, we have the following for the inverse S-box.

Proposition 4.2 *For the inverse S-box using composite field $GF(((2^2)^2)^2)$, choosing $\Phi = \{11\}_2$ and $\nu = \{1011\}_2$ and for the one using composite field $GF((2^4)^2)$ having $\nu = \{1001\}_2$ yields to the lowest area complexity architecture. It is noted that blocks 3 and 4 have the same predicted parities as the S-box by swapping η and σ. For other blocks of the optimum inverse S-box (PB_1) we have: $\hat{P}_{b1} = y_7 + y_6 + y_5 + y_2, \hat{P}_{b2} = \sigma_3(\overline{\sigma_7} + \sigma_4) + \sigma_2(\overline{\sigma_7} + P_{\sigma_h}) + \sigma_1(\sigma_6 + \sigma_4) + \sigma_0\overline{P_{\sigma_h}} + \sigma_4, \hat{P}_{b5} = \eta_7 + \eta_6 + \eta_3 + \eta_2 + \eta_0$. Additionally, for the optimum inverse S-box (PB_2) we have: $\hat{P}_{b1} = \overline{y_7 + y_6 + y_3}, \hat{P}_{b2} = \sigma_3\sigma_4 + \sigma_2(\sigma_5 + \sigma_4) + \sigma_1(\overline{P_{\sigma_h}} + \sigma_7) + \sigma_0\overline{P_{\sigma_h}} + \sigma_7, \hat{P}_{b5} = \eta_0$.*

4.2.2 The S-box and the Inverse S-box Using Normal Basis

Based on the possible values of ν' and Φ', the transformation matrices in blocks 1 and 5 of the S-box, denoted as $\mathbf{\Psi}'$ and $\mathbf{\Psi}'^{-1}$/affine, can be constructed using the algorithm presented in [24] with a slight modification for normal basis. One possible way to find the least complex transformation matrices is to calculate the Hamming weights, i.e., the number of non-zero elements, of the matrices $\mathbf{\Psi}'$ and $\mathbf{\Psi}'^{-1}$/affine. It is noted in [23] that instead of considering the Hamming weights, subexpression sharing is used for obtaining the low complexity implementations. We have exhaustively searched for the least overhead transformation matrices and their parity predictions combined, the results

Table 4.1: Area/delay complexities of blocks 1 and 5 of the S-box and their predicted parities for possible values of ν's and Φ's.

Φ'	ν'	H(Ψ')+H (Ψ'^{-1}/affine)	Total area of blocks 1 and 5	Total delay of blocks 1 and 5	Total area of $\hat{P}_{b1}$ and $\hat{P}_{b5}$	Total delay of $\hat{P}_{b1}$ and $\hat{P}_{b5}$
10	0001	57	28X	$7T_X$	5X	$4T_X$
	0010	57	32X		5X	
	0100	57	34X		5X	
	1000	57	30X		5X	
	0111	67	34X		3X	
	1011	65	30X		5X	
	1101	67	34X		3X	
	1110	65	31X		5X	
01	0001	57	32X		5X	
	0010	57	32X		5X	
	0100	57	29X		5X	
	1000	57	34X		5X	
	0111	65	34X		5X	
	1011	67	37X		3X	
	1101	65	34X		5X	
	1110	67	32X		3X	

$X = XOR$, T_X= Delay of an XOR

of which are presented in Table 4.1. In this table, for every possible combination of ν' and Φ', the Hamming weights of $\mathbf{\Psi'}$ and $\mathbf{\Psi'}^{-1}$/affine for the least complex cases are tabulated in column 3. Also, the number of gates needed for the low complexity implementation of blocks 1 and 5 are presented in column 4 of the table. Furthermore, the total number of XOR gates needed for the predicted parities of blocks 1 and 5 of the S-box, i.e., $\hat{P}_{b1}$ and $\hat{P}_{b5}$, and the delays associated with them are also shown in the table (see Fig. 4.2).

Block 2: As shown in Fig. 4.2, block 2 of the S-box consists of a multiplication, an addition, a squaring and a multiplication by constant ν' in $GF(2^4)$. The multiplication in $GF(2^4)$ consists of three multiplications, additions and a multiplication by constant Φ' in $GF(2^2)$. The following lemmas are used for deriving the predicted parity of the multiplication in $GF(2^4)$ and block 2, respectively.

Lemma 4.2 *Let $\lambda' = (\lambda'_3, \lambda'_2, \lambda'_1, \lambda'_0)$ and $\delta' = (\delta'_3, \delta'_2, \delta'_1, \delta'_0)$ be the inputs of a multiplier in $GF(2^4)$. The predicted parity of the result of the multiplication of λ' and δ' in $GF(2^4)$ is independent of Φ' and can be derived as*

$$\hat{P}'_\pi = \lambda'_3\delta'_3 + \lambda'_2\delta'_2 + \lambda'_1\delta'_1 + \lambda'_0\delta'_0. \tag{4.7}$$

Proof For the inputs $\Lambda' = (\Lambda'_1, \Lambda'_0)$ and $\Delta' = (\Delta'_1, \Delta'_0)$, the two-bit result of the multiplication in $GF(2^2)$, $\Pi' = (\Pi'_1, \Pi'_0)$, can be derived as $\Pi'_1 = \Delta'_1\Lambda'_0 + \Delta'_0\Lambda'_1 + \Delta'_0\Lambda'_0$ and

$\Pi'_0 = \Delta'_1\Lambda'_0 + \Delta'_0\Lambda'_1 + \Delta'_1\Lambda'_1$. Furthermore, multiplication by two possible values of Φ', i.e., $\Phi' = w^2 = \{10\}_2$ and $\Phi' = w = \{01\}_2$, can be obtained by putting $\Delta' = \Phi'$. Then, we have $\Pi'_1 = \Lambda'_0$ and $\Pi'_0 = \Lambda'_1 + \Lambda'_0$ for $\Phi' = w^2 = \{10\}_2$ and $\Pi'_1 = \Lambda'_1 + \Lambda'_0$ and $\Pi'_0 = \Lambda'_1$ for $\Phi' = w = \{01\}_2$. Consequently, one can derive the coordinates of π'. Therefore, for $\Phi' = w^2 = \{10\}_2$ we have

$$\begin{aligned}
\pi'_3 &= \lambda'_3(\delta'_3 + \delta'_1 + \delta'_0) + \lambda'_2(\delta'_1 + \delta'_2) + \lambda'_1(\delta'_3 + \delta'_2 + \delta'_1 + \delta'_0) + \lambda'_0(\delta'_3 + \delta'_1),\\
\pi'_2 &= \lambda'_3(\delta'_2 + \delta'_1) + \lambda'_2(\delta'_3 + \delta'_2 + \delta'_0) + \lambda'_1(\delta'_3 + \delta'_1) + \lambda'_0(\delta'_2 + \delta'_0),\\
\pi'_1 &= \lambda'_3(\delta'_3 + \delta'_2 + \delta'_1 + \delta'_0) + \lambda'_2(\delta'_3 + \delta'_1) + \lambda'_1(\delta'_3 + \delta'_2 + \delta'_1) + \lambda'_0(\delta'_3 + \delta'_0),\\
\pi'_0 &= \lambda'_3(\delta'_3 + \delta'_1) + \lambda'_2(\delta'_2 + \delta'_0) + \lambda'_1(\delta'_3 + \delta'_0) + \lambda'_0(\delta'_2 + \delta'_1 + \delta'_0).
\end{aligned} \tag{4.8}$$

Also, for $\Phi' = w = \{01\}_2$ we have the result as

$$\begin{aligned}
\pi'_3 &= \lambda'_3(\delta'_3 + \delta'_2 + \delta'_1) + \lambda'_2(\delta'_3 + \delta'_0) + \lambda'_1(\delta'_3 + \delta'_1) + \lambda'_0(\delta'_2 + \delta'_0),\\
\pi'_2 &= \lambda'_3(\delta'_3 + \delta'_0) + \lambda'_2(\delta'_2 + \delta'_1 + \delta'_0) + \lambda'_1(\delta'_2 + \delta'_0) + \lambda'_0(\delta'_3 + \delta'_2 + \delta'_1 + \delta'_0),\\
\pi'_1 &= \lambda'_3(\delta'_3 + \delta'_1) + \lambda'_2(\delta'_2 + \delta'_0) + \lambda'_1(\delta'_3 + \delta'_1 + \delta'_0) + \lambda'_0(\delta'_2 + \delta'_1),\\
\pi'_0 &= \lambda'_3(\delta'_2 + \delta'_0) + \lambda'_2(\delta'_3 + \delta'_2 + \delta'_1 + \delta'_0) + \lambda'_1(\delta'_2 + \delta'_1) + \lambda'_0(\delta'_3 + \delta'_2 + \delta'_0).
\end{aligned} \tag{4.9}$$

Modulo-2 adding the coordinates of (4.8) or (4.9) gives (4.7) and the proof is complete. ∎

Lemma 4.3 *The predicted parity of block 2, i.e., $\hat{P}_{b2}$, depends on the choice of the coefficients $\nu' \in GF(2^4)$ and $\Phi' \in GF(2^2)$ in the irreducible polynomials $u^2 + u + \nu'$ and $v^2 + v + \Phi'$ used for the composite field.*

Proof Considering the fact that $\hat{P}_{b2} = \hat{P}_{(\eta'_h+\eta'_l)^2\nu'} + \hat{P}_{\eta'_h\eta'_l}$, one can use Lemma 4.2 to obtain $\hat{P}_{\eta'_h\eta'_l}$ independent of the values of ν' and Φ'. However, $\hat{P}_{(\eta'_h+\eta'_l)^2\nu'}$ depends on the elements ν' and Φ'. This is because of having squaring in $GF(2^4)$, i.e., $(\eta'_h + \eta'_l)^2$, and also a multiplication by ν' to obtain $\hat{P}_{(\eta'_h+\eta'_l)^2\nu'}$. Therefore, the predicted parity of block 2 is also dependent on these values and the proof is complete. ∎

Using these lemmas, we can state the following to predict the parity of block 2.

Lemma 4.4 *The predicted parity of block 2, i.e., $\hat{P}_{b2}$, can be derived as shown in Table 4.2.*

Table 4.2: Parity predictions and complexities of block 2 of the normal basis S-box for possible values of ν' and Φ'.

Φ'	ν'	Area of block 2	Delay of block 2	Predicted parity ($\hat{P}_{b2}$)	Area of $\hat{P}_{b2}$	Delay of $\hat{P}_{b2}$
10	0001	28X+9A	$6T_X$ $+1T_A$	$(\eta'_7 \vee \eta'_3) + (\eta'_6 \vee \eta'_2) + (\eta'_4 \vee \eta'_0) + \eta'_5\eta'_1$	3X+3O+1A	$2T_X$ $+1T_A$
	0010	29X+9A		$(\eta'_7 \vee \eta'_3) + (\eta'_5 \vee \eta'_1) + (\eta'_4 \vee \eta'_0) + \eta'_6\eta'_2$	3X+3O+1A	
	0100	28X+9A		$(\eta'_6 \vee \eta'_2) + (\eta'_5 \vee \eta'_1) + (\eta'_4 \vee \eta'_0) + \eta'_7\eta'_3$	3X+3O+1A	
	1000	29X+9A		$(\eta'_7 \vee \eta'_3) + (\eta'_6 \vee \eta'_2) + (\eta'_5 \vee \eta'_1) + \eta'_4\eta'_0$	3X+3O+1A	
	0111	28X+9A		$(\eta'_4 \vee \eta'_0) + \eta'_7\eta'_3 + \eta'_6\eta'_2 + \eta'_5\eta'_1$	3X+3A+1O	
	1011	29X+9A		$(\eta'_7 \vee \eta'_3) + \eta'_6\eta'_2 + \eta'_5\eta'_1 + \eta'_4\eta'_0$	3X+3A+1O	
	1101	28X+9A		$(\eta'_6 \vee \eta'_2) + \eta'_7\eta'_3 + \eta'_5\eta'_1 + \eta'_4\eta'_0$	3X+3A+1O	
	1110	29X+9A		$(\eta'_5 \vee \eta'_1) + \eta'_7\eta'_3 + \eta'_6\eta'_2 + \eta'_4\eta'_0$	3X+3A+1O	
01	0001	29X+9A		$(\eta'_6 \vee \eta'_2) + (\eta'_5 \vee \eta'_1) + (\eta'_4 \vee \eta'_0) + \eta'_7\eta'_3$	3X+3O+1A	
	0010	28X+9A		$(\eta'_7 \vee \eta'_3) + (\eta'_6 \vee \eta'_2) + (\eta'_5 \vee \eta'_1) + \eta'_4\eta'_0$	3X+3O+1A	
	0100	29X+9A		$(\eta'_7 \vee \eta'_3) + (\eta'_6 \vee \eta'_2) + (\eta'_4 \vee \eta'_0) + \eta'_5\eta'_1$	3X+3O+1A	
	1000	28X+9A		$(\eta'_7 \vee \eta'_3) + (\eta'_5 \vee \eta'_1) + (\eta'_4 \vee \eta'_0) + \eta'_6\eta'_2$	3X+3O+1A	
	0111	29X+9A		$(\eta'_6 \vee \eta'_2) + \eta'_7\eta'_3 + \eta'_5\eta'_1 + \eta'_4\eta'_0$	3X+3A+1O	
	1011	28X+9A		$(\eta'_5 \vee \eta'_1) + \eta'_7\eta'_3 + \eta'_6\eta'_2 + \eta'_4\eta'_0$	3X+3A+1O	
	1101	29X+9A		$(\eta'_4 \vee \eta'_0) + \eta'_7\eta'_3 + \eta'_6\eta'_2 + \eta'_5\eta'_1$	3X+3A+1O	
	1110	28X+9A		$(\eta'_7 \vee \eta'_3) + \eta'_6\eta'_2 + \eta'_5\eta'_1 + \eta'_4\eta'_0$	3X+3A+1O	

$A = AND, \{+, X\} = XOR, \{\vee, O\} = OR$

T_X= Delay of an XOR, T_A= Delay of an AND= Delay of an OR

Proof One can use Lemma 4.2 to obtain $\hat{P}_{(\eta'_h+\eta'_l)^2\nu'}$ and $\hat{P}_{\eta'_h\eta'_l}$ in $\hat{P}_{b2} = \hat{P}_{(\eta'_h+\eta'_l)^2\nu'} + \hat{P}_{\eta'_h\eta'_l}$. $\hat{P}_{\eta'_h\eta'_l}$ can be easily found using Lemma 4.2. Furthermore, using Lemma 4.2 with the inputs being $\lambda' = (\eta'_h + \eta'_l)^2$ and $\delta' = \nu'$ one can obtain $\hat{P}_{(\eta'_h+\eta'_l)^2\nu'}$. Noting that the possible values for Φ' are $\Phi' = w^2 = \{10\}_2$ and $\Phi' = w = \{01\}_2$, one can find the corresponding possible $(\eta'_h + \eta'_l)^2$ using (4.8) and (4.9). This is achieved by putting both inputs in (4.8) or (4.9) as $\eta'_h + \eta'_l$. Then, for $\Phi' = w^2 = \{10\}_2$ we have

$$\begin{aligned}(\eta'_h + \eta'_l)^2 =&(\eta'_7 + \eta'_6 + \eta'_5 + \eta'_3 + \eta'_2 + \eta'_1, \eta'_6 + \eta'_5 + \eta'_4 + \eta'_2 + \eta'_1 + \eta'_0,\\ &\eta'_7 + \eta'_5 + \eta'_4 + \eta'_3 + \eta'_1 + \eta'_0, \eta'_7 + \eta'_6 + \eta'_4 + \eta'_3 + \eta'_2 + \eta'_0),\end{aligned} \tag{4.10}$$

and for $\Phi' = w = \{01\}_2$ we have

$$\begin{aligned}(\eta'_h + \eta'_l)^2 =&(\eta'_7 + \eta'_5 + \eta'_4 + \eta'_3 + \eta'_1 + \eta'_0, \eta'_7 + \eta'_6 + \eta'_4 + \eta'_3 + \eta'_2 + \eta'_0,\\ &\eta'_7 + \eta'_6 + \eta'_5 + \eta'_3 + \eta'_2 + \eta'_1, \eta'_6 + \eta'_5 + \eta'_4 + \eta'_2 + \eta'_1 + \eta'_0).\end{aligned} \tag{4.11}$$

One can obtain the predicted parities of block 2, i.e., $\hat{P}_{b2} = \hat{P}_{(\eta'_h+\eta'_l)^2\nu'} + \hat{P}_{\eta'_h\eta'_l}$, for all the possible combinations of ν' and Φ'. The results are presented in Table 4.2. ∎

Table 4.2 shows the predicted parities for different combinations of ν' and Φ' and their area/delay complexities. Moreover, the complexities for block 2 are shown in this

table. As seen in Table 4.2, the delay overhead for both the original block and its parity prediction is the same for all the cases. Whereas, the area in terms of the number of gates are different for different values of ν' and Φ'.

Block 3: Block 3 in Fig. 4.2 consists of an inversion in $GF(2^4)$. The inversion in $GF(2^4)$ is dependent on the two possible choices of Φ' and is the same for different values of ν'. Therefore, depending on the choice of Φ', there are two possible choices for this block and its parity prediction. It is noted that for both of these implementations, the area and the critical path delay are the same. The following theorem is used for obtaining the predicted parity of block 3, i.e., $\hat{P}_{b3}$.

Theorem 4.2 *Let $\gamma' = (\gamma'_3, \gamma'_2, \gamma'_1, \gamma'_0)$ be the input and $\theta' = (\theta'_3, \theta'_2, \theta'_1, \theta'_0)$ be the output of an inverter in $GF(2^4)$. Then, for $\Phi' = w^2 = \{10\}_2$, the predicted parity of block 3, i.e., $\hat{P}_{b3}$, can be found as*

$$\hat{P}_{b3} = \hat{P}_{\theta'} = \overline{\gamma'_2\gamma'_0}(\gamma'_3 + \gamma'_1) + \gamma'_3\gamma'_1(\gamma'_2 + \gamma'_0). \tag{4.12}$$

Also, for $\Phi' = w = \{01\}_2$ we have

$$\hat{P}_{b3} = \hat{P}_{\theta'} = \overline{\gamma'_3\gamma'_1}(\gamma'_2 + \gamma'_0) + \gamma'_2\gamma'_0(\gamma'_3 + \gamma'_1). \tag{4.13}$$

Proof According to [23], we have $\hat{P}_{\theta'} = \hat{P}_{\Upsilon^{-1}\gamma'_h} + \hat{P}_{\Upsilon^{-1}\gamma'_l} = \hat{P}_{\Upsilon^{-1}(\gamma'_h+\gamma'_l)}$. Then, according to the predicted parity of the multiplication in $GF(2^2)$ in the proof of Lemma 4.2, we have $\hat{P}_{\Upsilon^{-1}(\gamma'_h+\gamma'_l)} = \Upsilon_1^{-1}(\gamma'_3 + \gamma'_1) + \Upsilon_0^{-1}(\gamma'_2 + \gamma'_0)$. Moreover, considering the fact that the inversion in $GF(2^2)$ is free, i.e., $\Upsilon^{-1} = (\Upsilon_0, \Upsilon_1)$, we reach $\hat{P}_{\theta'} = \Upsilon_0(\gamma'_3+\gamma'_1)+\Upsilon_1(\gamma'_2+\gamma'_0)$. Then, according to the formulations for the multiplication in $GF(2^2)$ and knowing that the squaring in $GF(2^2)$ is free, finding the coordinates of Υ for two values of Φ' is straightforward and the proof is complete. ∎

Block 4: Block 4 of the S-box consists of two multiplications in $GF(2^4)$. According to Lemma 4.2, the area/delay overhead of the multiplications in $GF(2^4)$ and that of their predicted parity are the same for both $\Phi' = w = \{01\}_2$ and $\Phi' = w^2 = \{10\}_2$. Moreover, we have $\hat{P}_{b4} = \hat{P}_{\eta'_h\theta'} + \hat{P}_{\eta'_l\theta'} = \hat{P}_{(\eta'_h+\eta'_l)\theta'}$. Then, according to (4.7) in Lemma 4.2 with the inputs of $\eta'_h + \eta'_l$ and θ', one can find $\hat{P}_{b4}$ as

$$\hat{P}_{b4} = (\eta'_7 + \eta'_3)\theta'_3 + (\eta'_6 + \eta'_2)\theta'_2 + (\eta'_5 + \eta'_1)\theta'_1 + (\eta'_4 + \eta'_0)\theta'_0. \tag{4.14}$$

It is noted that for the implementation of $\hat{P}_{b4}$, the modulo-2 additions of $\eta'_7 + \eta'_3$, $\eta'_6 + \eta_2$, $\eta'_5 + \eta'_1$, and $\eta'_4 + \eta'_0$ are already available at the input of block 2. Therefore, this implementation only needs 3 XORs and 4ANDs.

Above, the optimum fault detection S-box using normal basis in Fig. 4.2 has been derived. In the following, we have also performed an exhaustive search for finding the optimum predicted parities based on the choice of the coefficients $\nu' \in GF(2^4)$ and $\Phi' \in GF(2^2)$ for the five blocks of the inverse S-box using normal basis. We have exhaustively searched for the least overhead transformation matrices and their parity predictions combined for the inverse S-box and have derived the total complexity for the predicted parities of blocks 1 and 5, i.e., $\hat{P}_{b1}$ and $\hat{P}_{b5}$, and the delays associated with them. These are used to obtain the optimum S-box inverse S-box and its parity predictions in this section. It is also noted that as shown in Fig. 4.2, blocks 2, 3, and 4 of the S-box and the inverse S-box are the same. Therefore, the predicted parities of these blocks can be obtained for the inverse S-box. Using the discussions presented in this section, we present the following for the optimum parity predictions.

Proposition 4.3 *For different combinations of ν' and Φ' for normal basis, for the S-box and the inverse S-box, $\Phi' = \{10\}_2$ and $\nu' = \{0001\}_2$ have the least area for the operations and their fault detection circuits combined. The following is the predicted parities for the S-box: $\hat{P}_{b1} = x_7 + x_5, \hat{P}_{b2} = (\eta''_7 \vee \eta'_3) + (\eta'_6 \vee \eta'_2) + (\eta'_4 \vee \eta'_0) + \eta'_5\eta'_1, \hat{P}_{b3} = \overline{\gamma'_2\gamma'_0}(\gamma'_3+\gamma'_1)+\gamma'_3\gamma'_1(\gamma'_2+\gamma'_0), \hat{P}_{b4} = (\eta'_7+\eta'_3)\theta'_3+(\eta'_6+\eta'_2)\theta'_2+(\eta'_5+\eta'_1)\theta'_1+(\eta'_4+\eta'_0)\theta'_0, \hat{P}_{b5} = \sigma'_7 + \sigma'_5 + \sigma'_4 + \sigma'_3 + \sigma'_2$. Moreover, for the inverse S-box, $\hat{P}_{b2} - \hat{P}_{b4}$ are the same as those for the S-box by swapping η' and σ'. For the other blocks, we have: $\hat{P}_{b1} = y_7+y_6+y_2+y_1$ and $\hat{P}_{b5} = \eta'_7 + \eta'_5 + \eta'_4 + \eta'_3 + \eta'_2$.*

It is noted that the area overhead of the proposed scheme for the optimum structures consists of those of the optimum parity predictions. In addition, 23 XORs for the actual parities (3 XORs for adding the coordinates of each of $\eta'_h + \eta'_l$, γ', and θ' and 7 XORs each for those of σ' and Y) are utilized. Moreover, the delay overhead of the predicted parities of 5 blocks can overlap the delays for the implementations of 5 blocks in Figs. 4.1 and 4.2. The only delay overhead for this scheme is the delay of the actual parity of block 5, which is $3T_X$, where, T_X is the delay of an XOR gate.

Table 4.3: Error simulation results of the optimum S-box and inverse S-box after injecting $500,000$ errors.

Operations	Field	Errors covered	Error Coverage
S-box (Inverse S-box)	PB_1	485,008 (485,106)	97.002% (97.021%)
	PB_2	485,039 (485,015)	97.008% (97.002%)
	NB	485,015 (485,174)	97.003% (97.035%)

4.3 Error Simulations

If exactly one bit error appears at the output of the S-box (resp. inverse S-box), the presented fault detection scheme is able to detect it and the error coverage is about 99.998%. This is because in this case, the error indication flag of the corresponding block alarms the error. However, due to the technological constraints, single stuck-at error may not be applicable for an attacker to gain more information [82]. Thus, multiple bits will actually be flipped and hence multiple stuck-at errors are also considered in this chapter covering both natural faults and fault attacks [82].

For the calculation of the error coverage for the multiple errors, we define p_i as the probability of error detection in block i, $1 \leq i \leq 5$, in Figs. 4.1 and 4.2. Then, the probability of not detecting the errors in block i is $(1 - p_i)$. For randomly distributed errors in the S-box (resp. inverse S-box), this probability for each block is independent of those of other blocks. Therefore, one can derive the equation for the error coverage of the randomly distributed errors as

$$EC\% = 100 \times (1 - \prod_{i \in \mathbb{S}} (1 - p_i))\%, \tag{4.15}$$

where $\mathbb{S}$ is the set of the block numbers where the faults are injected. For randomly distributed errors, the error coverage for each block is $p_i \approx \frac{1}{2}$. Then, the representation of (4.15) can be simplified as $EC\% = 100 \times (1 - (\frac{1}{2})^n)\%$, where, n is the number of blocks. Therefore, if multiple errors are randomly distributed in all blocks, the error coverage reaches 97% using $n = 5$ error indication flags.

We have performed error simulations for the S-boxes and the inverse S-boxes using the optimum composite field obtained in the previous section to confirm our above theoretical computation. In our simulations, we use stuck-at error model at the outputs of the five blocks forcing one or multiple nodes to be stuck at logic one (for stuck-at one) or zero (for

stuck-at zero) independent of the error-free values. We use Fibonacci implementation of the LFSRs for injecting random multiple errors, where, the numbers, the locations and the types of the errors are randomly chosen. In this regard, the maximum sequence length polynomial for the feedback is selected. The injected errors are transient, i.e., they last for one clock cycle. However, the results would be the same if permanent errors are considered.

The results of the error simulations using Xilinx® ISE™ version 9.1i Simulator (ISim) [80] are presented in Table 4.3. As seen in this table, up to $500,000$ random errors are injected for both the S-box and the inverse S-box. It is noted that in these tables, the optimum polynomial basis $GF(((2^2)^2)^2)$ denoted by PB_1, $GF((2^4)^2)$ denoted by PB_2 and normal basis (NB) are presented. As shown in the table, using 5 parity bits of the 5 blocks, the error coverage for random faults reaches 97% which is the same as our theoretical computation in this section. This error coverage will be increased if the outputs of more than one S-box (resp. inverse S-box) of the AES implementation are erroneous. In this case, the errors detected in any of 16 S-boxes (resp. inverse S-boxes) contribute to the total error coverage. Thus, error coverage of very close to 100% (99.998%) is achieved.

4.4 ASIC and FPGA Implementations and Comparisons

In this section, we compare the areas and the delays of the presented scheme with those of the previously reported ones in both ASIC and FPGA implementations. We have implemented the S-boxes using memories and the ones presented in [21], [22] (the hardware optimization of [20]), and [81] which use polynomial basis representation in composite fields. We have also implemented the fault detection schemes proposed in [34], [36], and [42] (both united and parity-based) which are based on the ROM-based implementation of the S-box. The results of the implementations for both original and fault detection scheme (FDS) in terms of delay and area have been tabulated in Tables 4.4 and 4.5. As seen in these tables, the original structures are not divided into blocks and full optimization of the original entire architecture as a single block is performed in both ASIC and FPGA. This allows us to find the actual overhead of the presented fault detection scheme as compared to the original structures which are not divided into five blocks. We have

Table 4.4: ASIC implementations of the fault detection schemes for the S-box (SB) and the inverse S-box using 0.18μ CMOS technology.

Operation	Architecture		Area (μm^2) , Delay (ns)	
	Structure	FDS	Original	FDS
S-box	ROM SB	United S-box [34], [42]	169×10^3 , 5.4	344×10^3 , 7.7
	ROM SB	Two 256×9 ROMs [36]	169×10^3 , 5.4	378×10^3 , 5.8
	ROM (mult. inv.)	Parity-based SB [42]	185×10^3 , 5.8	191×10^3 , 5.9
	PB [22]	[52] (mult. inv.)	5315 , 12.0	6869 , 12.8
	PB [22]	[53]	5315 , 12.0	7047 , 14.1
	PB [22]	[57] for the original SB	5315 , 12.0	6763 , 14.1
	PB [21]	Proposed scheme applied	5642 , 11.3	7113 , 13.0
	PB [81]	Proposed scheme applied	5547 , 13.2	7034 , 13.8
	NB	**[78]**	**5179 , 12.9**	**6712 , 14.7**
	PB_1	**[79]**	**5217 , 10.6**	**6723 , 12.5**
	PB_2	**[79]**	**5290 , 9.2**	**6739 , 11.5**
Inverse S-box	**NB**	**[79]**	**5187 , 13.2**	**6480 , 14.5**
	PB_1	**[79]**	**5225 , 10.9**	**6537 , 13.0**
	PB_2	**[79]**	**5274 , 9.4**	**6619 , 11.3**

used 0.18μ CMOS technology for the ASIC implementations. These architectures have been coded in VHDL as the design entry to the Synopsys Design Analyzer. The results are tabulated in Table 4.4. Moreover, for the FPGA implementations in Table 4.5, the Xilinx® Virtex™-II Pro FPGA (xc2vp2-7) [80] is utilized in the Xilinx® ISE™ version 9.1i. Furthermore, the synthesis is performed using the XST™.

As seen in Tables 4.4 and 4.5, we have implemented the fault detection scheme presented in [34] and [42] based on using redundant units for the S-box (united S-box). Furthermore, the fault detection scheme proposed in [36] is implemented. This scheme uses 512×9 memory cells to generate the predicted parity bit and the 8-bit output of the S-box [36]. One can obtain from Tables 4.4 and 4.5 that for both of these schemes, the area overhead is more than 100%. As mentioned in the introduction, the approach in [50] utilizes the scheme in [36] for protecting the combinational logic elements, whose implementation results are also shown in Tables 4.4 and 4.5. Additionally, for certain AES implementations containing storage elements, one can use the error correcting code-based approach presented in [50] in addition to the proposed scheme in this chapter to make a more reliable AES implementation. Moreover, the parity-based scheme in [42] which only realizes the multiplicative inversion (mult. inv.) using memories is imple-

Table 4.5: Xilinx® Virtex™-II Pro FPGA implementations (xc2vp2-7) of the fault detection schemes for the S-box (SB) and the inverse S-box.

Operation	Architecture		Slice , Delay (ns)	
	Structure	FDS	Original	FDS
S-box	ROM (SB)	United SB [34], [42]	69 , 3.826	150 , 5.398
	ROM (SB)	Two 256 × 9 ROMs [36]	69 , 3.826	159 , 4.287
	ROM (mult. inv.)	Parity-based SB [42]	88 , 5.734	100 , 6.370
	PB [22]	[52] (mult. inv.)	33 , 9.375	44 , 9.869
	PB [22]	[53]	33 , 9.375	47 , 9.996
	PB [22]	[57] for the original SB	33 , 9.375	42 , 10.317
	PB [21]	Proposed scheme applied	38 , 8.285	50 , 9.582
	PB [81]	Proposed scheme applied	37 , 9.986	47 , 10.832
	NB	**[78]**	**31 , 9.339**	**39 , 10.026**
	PB_1	**[79]**	**31 , 7.284**	**40 , 7.465**
	PB_2	**[79]**	**32 , 7.356**	**41 , 8.150**
Inverse S-box	**NB**	**[79]**	**31 , 7.736**	**38 , 7.964**
	PB_1	**[79]**	**32 , 6.992**	**42 , 7.423**
	PB_2	**[79]**	**32 , 7.550**	**44 , 8.181**

mented. As seen in these tables, we have also implemented the schemes in [52] and [53]. It is noted that the scheme in [52] is for the multiplicative inversion and does not present the parity predictions for the transformation matrices. Moreover, we have applied the presented fault detection scheme to the S-boxes in [21] and [81]. As seen in bold faces in Tables 4.4 and 4.5, with the error coverage of 99.998%, the presented low-complexity fault detection S-boxes are the most compact ones among the other S-boxes. The optimum S-box and inverse S-box using normal basis have the least hardware complexity with the fault detection scheme. Moreover, as seen in the tables, the optimum structures using composite fields and polynomial basis (PB_1 and PB_2) have the least post place and route timing overhead among other schemes. It is noted that using sub-pipelining for the presented fault detection scheme in this chapter, one can reach much more faster hardware implementations of the composite field fault detection structures.

We have also implemented the AES encryption using the presented optimum S-boxes excluding the key expansion. Then, we have added the proposed scheme for SubBytes and ShiftRows considering that ShiftRows is the rewiring from the output of SubBytes. The results are presented in Tables 4.6 and 4.7. As one can notice, the S-boxes occupy more than three fourths of the AES encryption. As shown in these tables, the most com-

Table 4.6: ASIC implementations of the fault detection schemes of the AES encryption using 0.18μ CMOS technology.

AES encryption	Optimum S-box	Area (μm^2)		Freq. (MHz)
		S-boxes	All	
Original without fault detection	PB_1	692781 (80%)	859471	79.4
	PB_2	704490 (80%)	871180	91.8
	NB	680590 (80%)	845426	73.5
Presented scheme for SubBytes (ShiftRows)	PB_1	956233	-	78.8
	PB_2	972217	-	89.2
	NB	946476	-	69.5
Presented scheme for SubBytes (ShiftRows) scheme in [36] for others	PB_1	-	1268520	68.2
	PB_2	-	1280412	70.1
	NB	-	1256812	60.3

Table 4.7: Xilinx® Virtex™-II Pro FPGA implementations of the fault detection schemes of the AES encryption.

AES encryption	Optimum S-box	Slice		Freq. (MHz)
		S-boxes	All	
Original without fault detection	PB_1	5248 (77%)	6760	81.1
	PB_2	5417 (78%)	6913	89.8
	NB	5112 (78%)	6579	75.8
Presented scheme for SubBytes (ShiftRows)	PB_1	6896	-	79.3
	PB_2	6958	-	84.0
	NB	6342	-	73.2
Presented scheme for SubBytes (ShiftRows) scheme in [36] for others	PB_1	-	9881	65.8
	PB_2	-	9921	64.8
	NB	-	9405	60.8

pact AES encryption with and without the fault detection scheme is for normal basis. Furthermore, the frequency degradation is negligible. Moreover, the original AES encryption for PB_2 and the ones with fault detection for PB_1 and PB_2 have the highest working frequencies. In addition, as seen in the tables, we have applied the presented scheme to SubBytes and ShiftRows and used the scheme in [36] for the other transformations.

In this chapter, we have presented a high performance parity-based concurrent fault detection scheme for the AES using the S-box and the inverse S-box in composite fields. Using exhaustive searches, we have found the least complexity S-boxes and inverse S-boxes as well as their fault detection circuits. Our error simulation results show that very high error coverages for the presented scheme are obtained. Moreover, a number of fault detection schemes from the literature have been implemented on ASIC and FPGA and compared with the ones presented here. Our implementations show that the optimum

S-boxes and the inverse S-boxes using normal basis are more compact than the ones using polynomial basis. However, the ones using polynomial basis result in the fastest implementations. We have also implemented the AES encryption using the proposed fault detection scheme. The results of the ASIC and FPGA mapping show that the costs of the presented scheme are reasonable with acceptable post place and route delays.

Chapter 5

A High-Performance Concurrent Fault Detection Approach for the Composite Field (Inverse) S-box

IN the previous chapter, an exhaustive search-based fault detection scheme for the AES S-boxes and inverse S-boxes was presented. In this chapter, we also present a concurrent fault detection scheme for the S-box and the inverse S-box based on the low-cost composite field implementations of the S-box and the inverse S-box. However, we divide the structures of these operations into three blocks and find the predicted parities of these blocks. Our simulations show that except for the redundant units approach which has the hardware and time overheads of close to 100%, the fault detection capabilities of the proposed scheme for the burst and random multiple faults are higher than the previously reported ones. Finally, through ASIC implementations, it is shown that for the maximum target frequency, the proposed fault detection S-box and inverse S-box in this chapter have the least areas, critical path delays, and power consumptions compared to their counterparts with similar fault detection capabilities.

In this chapter, we present a low-power and high-performance parity-based fault detection approach for the S-box, the inverse S-box, and the merged S-box/inverse S-box within the AES using composite fields. We obtain new formulations for the five predicted parities for three blocks of the S-box and the inverse S-box. To reach high multiple and burst fault detection capabilities, multiple-bit signatures are obtained within the blocks constituting more area in the structures of the S-box and the inverse S-box. Our simulation results show higher burst fault detection capability for the proposed scheme compared to the previously presented schemes with similar comparable overheads. This

can be used as an effective countermeasure against the fault attacks noting that in realistic fault attacks, multiple adjacent bits are actually flipped [82]. Moreover, using the proposed scheme, for multiple random faults, the entire SubBytes and inverse SubBytes are capable of detecting 99.998% of the injected faults. Through ASIC implementations, it is shown that for the maximum target frequency, the timing, power and area of the proposed scheme are the least compared to the schemes with similar fault detection capabilities. It is noted that the fault detection scheme proposed in this chapter can also be applied to both the low-area S-box and inverse S-box presented in [17], [18], [20], [22], and the low-power one proposed in [13].

The organization of this chapter is as follows. In Section 5.1, preliminaries related to the S-box and the inverse S-box arithmetic used in this chapter are presented. The proposed fault detection approach for the S-box, the inverse S-box, and the merged structures is presented in Section 5.2. Furthermore, the time and hardware complexities analysis is preformed in this section. In Section 5.3, the results of the simulations of the proposed approach are presented; through which, the fault detection capabilities are derived. In Section 5.4, through ASIC implementations, the areas, power consumptions, and critical path delays of the proposed fault detection scheme and the previously reported ones are compared. We also present the formulations for the mixes S-box in Section 5.5. The results presented in this chapter can also be found in [83] and [84].

5.1 S-box and Inverse S-box Arithmetic Used in This Chapter

The structures of the S-box and the inverse S-box using composite field and polynomial basis are shown in Fig. 5.1. As seen in Fig. 5.1, for the S-box, the transformation matrix $\mathbf{\Psi}$ transforms a field element $X = \sum_{i=0}^{7} x_i \alpha^i$ in the binary field $GF(2^8)$ to the corresponding representation in the composite field $GF(2^8)/GF(((2^2)^2)^2)$ for performing the multiplicative inversion. Then, using the inverse transformation matrix $\mathbf{\Psi^{-1}}$, the result of the multiplicative inversion, i.e., X^{-1}, is obtained. This is performed using the irreducible polynomial of $u^2 + u + \nu$. It is noted that the decomposition can be further applied to represent $GF((2^2)^2)$ as a linear polynomial over $GF(2^2)$ and then $GF(2)$ using the irreducible polynomials of $v^2 + v + \Phi$ and $w^2 + w + 1$, respectively. Eventually, as

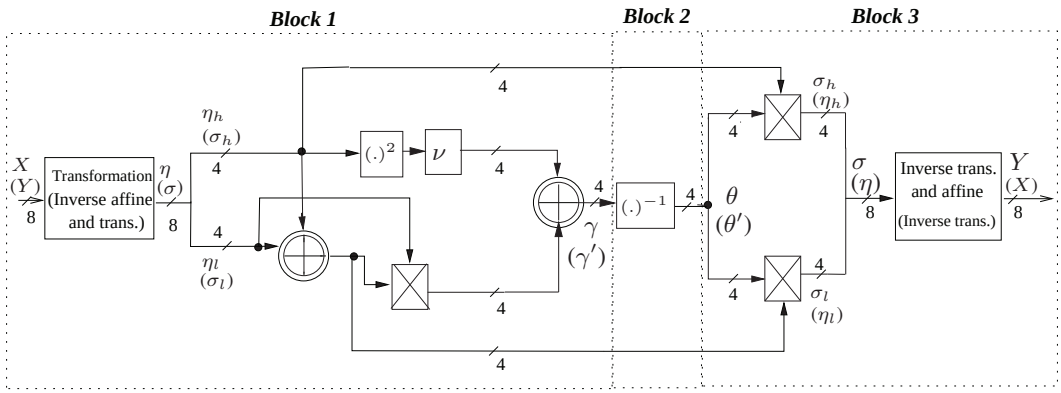


Figure 5.1: The architecture of the S-box (resp. the inverse S-box) using composite field and polynomial basis [20].

seen in Fig. 5.1, using the affine transformation, the 8-bit output of the S-box, i.e., Y, is derived. Furthermore, as seen in Fig. 5.1, for the inverse S-box, the reverse procedure is performed to obtain the output X from the input Y. It is noted that in Fig. 5.1, the notations for the inverse S-box are presented in parentheses.

All arithmetic operations including the multiplications, the inversion and the squaring in Fig. 5.1 are over $GF((2^2)^2)$. In Fig. 5.1, the two concentric circles with a plus inside represent 4 XOR gates which perform the modulo-2 addition. Moreover, the three finite field multiplications and the inversion in $GF((2^2)^2)$ are shown by crossed rectangles and $(.)^{-1}$, respectively. Furthermore, the multiplication by constant ν and squaring $(.)^2$ in $GF((2^2)^2)$ are shown in this figure. As seen in Fig. 5.1 for the S-box, for the output of the multiplicative inversion $\sigma_h x + \sigma_l = (\eta_h x + \eta_l)^{-1}$ we have the following [20]

$$\begin{aligned} \sigma_h &= ((\eta_h + \eta_l)\eta_l + {\eta_h}^2\nu)^{-1}\eta_h, \\ \sigma_l &= ((\eta_h + \eta_l)\eta_l + {\eta_h}^2\nu)^{-1}(\eta_h + \eta_l). \end{aligned} \tag{5.1}$$

Moreover, for the inverse S-box in Fig. 5.1, one can swap η and σ to derive the relation for the multiplicative inversion.

5.2 Proposed Fault Detection Approach

The parity-based fault detection scheme has received much attention in the literature, see, for example, [85], [86], [87], [88], [89], and [90]. In such schemes, the parity of a block is predicted and compared with the actual parity of the block. The result is the error

indication flag of the corresponding block which alarms the detected faults. Let τ and ρ be the input and the output of the block under test, respectively. Then, the predicted parity of ρ is obtained from the input τ, i.e., $\hat{P}_\rho(\tau)$, and the actual parity is implemented from the output ρ, i.e., $P_\rho(\rho)$. The comparison between the actual and predicted parities is implemented by an XOR gate to generate the error indication flag $e_\rho = \hat{P}_\rho(\tau) + P_\rho(\rho)$.

In the presented parity-based fault detection scheme, we divide the structures of the S-box and the inverse S-box using polynomial basis into 3 blocks as shown in Fig. 5.1 so that it can also be used for the low-power structures presented in [13] (see Fig. 2.4). One can obtain that for the S-box and inverse S-box presented in Fig. 5.1 [20], blocks 1 and 3 occupy around 86% of the area of the entire operations. Therefore, these two blocks are more susceptible to the internal faults and more prone to fault attacks. Consequently, we propose using two bits predicted parities for each of these two blocks. Furthermore, one predicted parity is used for block 2. The details of the proposed schemes are presented below.

5.2.1 S-box

In the proposed scheme, five predicted parities are derived for 3 blocks of the S-box. Then, by comparing these with the five actual parities, five error indication flags are obtained. All five flags should be zero for the error free computations. The proposed fault detection scheme for the S-box is shown in Fig. 5.2. As seen in this figure, for block 1, two predicted parities, i.e., $\hat{P}^1_{b1}$ and $\hat{P}^2_{b1}$, are obtained using the parity prediction unit (PP_1). As seen from Fig. 5.2, the predicted parity of the second block $\hat{P}_{b2}$ is obtained by the parity prediction unit (PP_2). Furthermore, for block 3, two predicted parities, i.e., $\hat{P}^1_{b3}$ and $\hat{P}^2_{b3}$, are derived using the parity prediction unit (PP_3).

The derivations of the actual parities are also shown in Fig. 5.2. As seen from Fig. 5.2, two actual parities for the two most and least significant bits of γ, i.e., $P^1_{b1} = \sum_{i=2}^{3} \gamma_i$ and $P^2_{b1} = \sum_{i=0}^{1} \gamma_i$, have been derived from the output of block 1 using two trees of XOR gates. Similarly, as shown in Fig. 5.2, the two actual parities for block 3 are obtained from the output of block 3 for the four most and least significant bits of Y, i.e., $P^1_{b3} = \sum_{i=4}^{7} y_i$ and $P^2_{b3} = \sum_{i=0}^{3} y_i$. In addition, one actual parity is obtained for block 2 as $P_{b2} = \sum_{i=0}^{3} \theta_i$. Then, as shown in Fig. 5.2, by comparing the predicted and actual parities, the error indication flags of three blocks, i.e., e_1-e_5, are obtained.

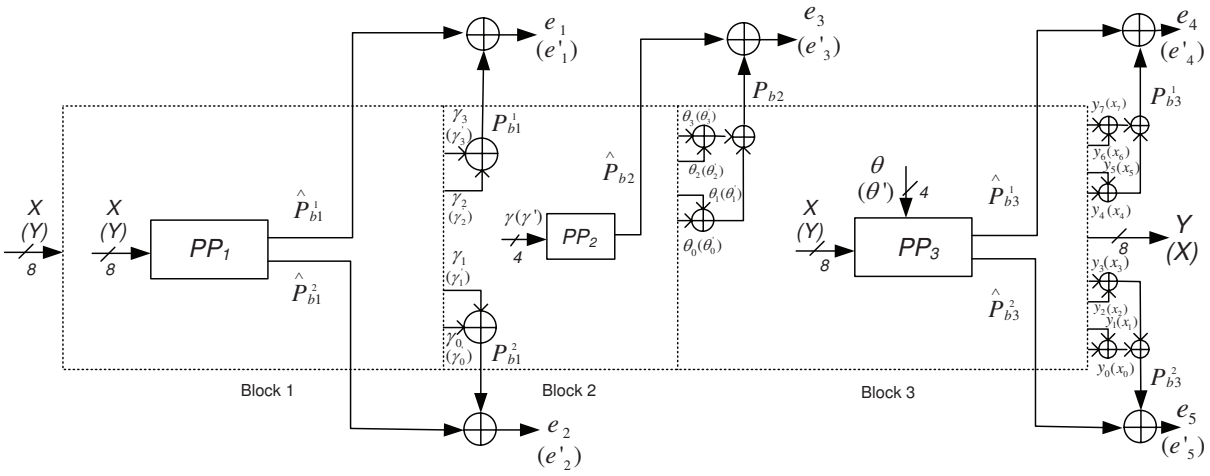


Figure 5.2: The proposed parity-based fault detection scheme for the S-box (resp. inverse S-box).

The following lemma is used from [20] for the multiplication in $GF((2^2)^2)$ used in blocks 1 and 3. Then, using this lemma, the predicted parities for the S-box in Fig. 5.2 are derived.

Lemma 5.1 *[20] Let $U = (u_3, u_2, u_1, u_0)$ and $V = (v_3, v_2, v_1, v_0)$ be the inputs of a multiplier in $GF((2^2)^2)$. Then, the result of multiplication, i.e., $Z = UV$, is*

$$\begin{aligned} z_3 &= u_3(v_3 + v_2 + v_1 + v_0) + u_2(v_3 + v_1) + u_1(v_3 \\ &\quad + v_2) + u_0 v_3, \\ z_2 &= u_3(v_3 + v_1) + u_2(v_2 + v_0) + u_1 v_3 + u_0 v_2, \\ z_1 &= u_3 v_2 + u_2(v_3 + v_2) + u_1(v_1 + v_0) + u_0 v_1, \\ z_0 &= u_3(v_3 + v_2) + u_2 v_3 + u_1 v_1 + u_0 v_0. \end{aligned} \tag{5.2}$$

Using Lemma 5.1, we present the formulations for these five predicted parities in the following theorem.

Theorem 5.1 *Let $X \in GF(2^8)$ be the input of the S-box. Then, the five predicted parities of the three blocks of the S-box in Fig. 5.2, i.e., $\hat{P}^1_{b1}$, $\hat{P}^2_{b1}$, $\hat{P}_{b2}$, $\hat{P}^1_{b3}$, and $\hat{P}^2_{b3}$, are obtained as follows*

$$\hat{P}^1_{b1} = x_7(D + x_5) + x_4 B + x_3(B + x_4) + x_0 D + x_1 x_2, \tag{5.3}$$

$$\hat{P}^2_{b1} = x_7(G + x_6) + x_4 I + x_1(C + E) + x_2 \vee x_5 + P_X, \tag{5.4}$$

$$\hat{P}_{b2} = (\overline{\gamma}_2 \vee \gamma_1)\gamma_0 + P_{\gamma_l}\gamma_3, \tag{5.5}$$

$$\hat{P}^1_{b3} = \theta_3 H + \theta_2(G + x_7) + \theta_1(J + C) + \theta_0 J, \tag{5.6}$$

$$\hat{P}^2_{b3} = \theta_3(C + x_0) + \theta_2(H + x_3) + \theta_1(I + x_7) + \theta_0(A + x_2), \tag{5.7}$$

where $x_1 + x_6 = A$, $x_5 + A = B$, $x_3 + x_2 = C$, $P_X + H = D$, $x_0 + x_6 = E$, $x_2 + x_5 = F$, $F + x_4 = G$, $x_0 + x_7 = H$, $B + C = I$, *and* $E + F = J$. *Furthermore, "+" and* $\vee$ *represent the modulo-2 addition using an XOR gate and the OR operation, respectively. Moreover,* $P_X = \sum_{i=0}^{7} x_i$ *and* $P_{\gamma_l} = \gamma_1 + \gamma_0$.

Proof First, we obtain the two predicted parities of block 1, i.e., $\hat{P}^1_{b1} = \hat{P}_{\gamma_h}$ and $\hat{P}^2_{b1} = \hat{P}_{\gamma_l}$ in (5.3) and (5.4). As seen from Fig. 5.1, block 1 consists of the transformation matrix $\mathbf{\Psi}$, a field multiplication, modulo-2 additions, and squaring followed by the multiplication by the constant ν. From [20], one can obtain that for the input of $\eta_h = (\eta_7, \eta_6, \eta_5, \eta_4)$, the result of the squarer-ν is

$${\eta_h}^2\nu = (\eta_7 + \eta_4, \eta_7 + \eta_6 + \eta_5, \eta_4, \eta_5). \tag{5.8}$$

Moreover, using (5.2) with the inputs $u = \eta_l$ and $v = \eta_h + \eta_l$, one can obtain the result of the field multiplication in this block. By modulo-2 adding the coordinates of $\gamma_h = (\gamma_3, \gamma_2)$ and $\gamma_l = (\gamma_1, \gamma_0)$, i.e., two most and least significant bits of (5.8) and that of the result of the multiplication, respectively, one can obtain

$$\begin{aligned}\hat{P}^1_{b1} &= \eta_3(\eta_6 + \eta_4) + \eta_2(\eta_7 + \eta_6 + \eta_5 + \eta_4) + \eta_1\eta_6 \\ &+ \eta_0(\eta_7 + \eta_6) + \eta_7 + \eta_6 + \eta_5 + \eta_2,\end{aligned} \tag{5.9}$$

$$\hat{P}^2_{b1} = \eta_3\eta_7 + \eta_2\eta_6 + \eta_1\eta_4 + \eta_0(\eta_5 + \eta_4) + \eta_6 + \eta_2 + \eta_0. \tag{5.10}$$

By substituting the coordinates of η with those of X and reordering the results in (5.9) and (5.10), one reaches the following

$$\begin{aligned}\hat{P}^1_{b1} &= x_7(x_6 + x_4 + x_3 + x_2 + x_1) + x_4(x_6 + x_5 + x_1) \\ &+ x_3(x_6 + x_5 + x_4 + x_1) + x_0(x_6 + x_5 + x_4 + x_3 \\ &+ x_2 + x_1) + x_1x_2,\end{aligned} \tag{5.11}$$

$$\begin{aligned}\hat{P}^2_{b1} &= x_7(x_6 + x_5 + x_4 + x_2) + x_4(x_6 + x_5 + x_3 \\ &+ x_2 + x_1) + x_1(x_6 + x_3 + x_2 + x_0) + x_2 \vee x_5 + P_X.\end{aligned} \tag{5.12}$$

Using subexpression sharing, it is straightforward to obtain (5.3) and (5.4) from (5.11) and (5.12), respectively. It is also noted that the predicted parity of block 2 in (5.5) is derived from that of block 3 in the scheme in [57] noting that $P_{\gamma_l} = \gamma_1 + \gamma_0$.

Now, we derive the two predicted parities of block 3, i.e., $\hat{P}^1_{b3} = \hat{P}_{Y_h}$ and $\hat{P}^2_{b3} = \hat{P}_{Y_l}$. As seen from Fig. 5.1, block 3 consists of the mixed inverse and affine transformation matrices and two field multiplications. It is straightforward that we obtain the formulations for these mixed transformation matrices as follows

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{A}\boldsymbol{\Psi}^{-1}\boldsymbol{\sigma} + \boldsymbol{b} \\ &= \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \boldsymbol{\sigma} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}. \end{aligned} \tag{5.13}$$

Eventually, $\hat{P}_{Y_h}$ and $\hat{P}_{Y_l}$, i.e., two predicted parities of block 3 in Fig. 5.2, are obtained as follows $\hat{P}_{Y_h} = \sigma_6 + \sigma_5 + \sigma_3 + \sigma_1 + \sigma_0$ and $\hat{P}_{Y_l} = \sigma_5 + \sigma_4 + \sigma_3 + \sigma_2$. Then, by multiplying $u = \theta$ and $v = \eta_h + \eta_l$ and also $u = \theta$ and $v = \eta_h$ using (5.2), one can obtain the coordinates of σ. Substituting these in above, the following is obtained for the two predicted parities of block 3 of the S-box in Fig. 5.2:

$$\begin{aligned} \hat{P}^1_{b3} &= \theta_3(x_7 + x_0) + \theta_2(x_7 + x_5 + x_4 + x_2) \\ &+ \theta_1(x_6 + x_5 + x_3 + x_0) + \theta_0(x_6 + x_5 + x_2 + x_0), \end{aligned} \tag{5.14}$$

$$\begin{aligned} \hat{P}^2_{b3} &= \theta_3(x_3 + x_2 + x_0) + \theta_2(x_7 + x_3 + x_0) \\ &+ \theta_1(x_7 + x_6 + x_5 + x_3 + x_2 + x_1) + \theta_0(x_6 + x_2 + x_1). \end{aligned} \tag{5.15}$$

Then, using subexpression sharing for (5.14) and (5.15), one can obtain (5.6) and (5.7) and the proof is complete. ∎

5.2.2 Inverse S-box

As seen in Fig. 5.2, similar to the S-box, for blocks 1-3 of the inverse S-box, five predicted parities are derived using the parity prediction units. This is also depicted in Fig. 5.2. It is noted that the notations for the inverse S-box are denoted by parentheses to be contrasted from those for the S-box. Additionally, similar to the S-box, the actual parities

of the three blocks for the inverse S-box are derived using XOR trees. It is noted that for blocks 1 and 3, the actual parities are obtained as $P^1_{b1} = \sum_{i=2}^{3} \gamma'_i$ and $P^2_{b1} = \sum_{i=0}^{1} \gamma'_i$ for block 1 and $P^1_{b3} = \sum_{i=4}^{7} x_i$ and $P^2_{b3} = \sum_{i=0}^{3} x_i$ for block 3. Then, as seen in Fig. 5.2, by comparing the predicted and actual parities, five error indication flags of three blocks, i.e., e'_1-e'_5, are obtained.

Using Lemma 5.1 and considering Theorem 5.1, we present the formulations for the five predicted parities of the inverse S-box for the 3 blocks shown in Fig. 5.2 in the following theorem.

Theorem 5.2 *Let $Y \in GF(2^8)$ be the output of the inverse S-box. The five predicted parities of the three blocks of the inverse S-box in Fig. 5.2 are obtained as follows.*

$$\hat{P}^1_{b1} = y_0 e + y_5(y_4 + y_3 + a) + y_2 b + y_7 y_4 + \overline{b}, \tag{5.16}$$

$$\hat{P}^2_{b1} = y_1(y_7 + y_5 + h) + y_2 a + y_3(y_5 + y_4) + y_5 h + y_0 + e, \tag{5.17}$$

$$\hat{P}_{b2} = (\overline{\gamma'}_2 \vee \gamma'_1)\gamma'_0 + P_{\gamma'_l}\gamma'_3, \tag{5.18}$$

$$\hat{P}^1_{b3} = \theta'_3 \overline{f} + \theta'_2(\overline{P_Y} + d + y_7) + \theta'_1(\overline{c} + y_7 + y_4) + \theta'_0(\overline{a} + y_4 + y_2), \tag{5.19}$$

$$\hat{P}^2_{b3} = \theta'_3(y_1 + d) + \theta'_2(y_0 + g) + \theta'_1(y_6 + g) + \theta'_0(y_1 + f), \tag{5.20}$$

where $y_6 + y_7 = a$, $y_1 + a = b$, $y_1 + y_2 = c$, $y_3 + y_6 = d$, $c + d = e$, $P_Y + y_4 + y_6 = f$, $\overline{P_Y} + y_2 = g$, and $y_4 + y_0 = h$. Furthermore, "+" and $\vee$ represent the modulo-2 addition using an XOR gate and the OR operation, respectively. Moreover, $P_Y = \sum_{i=0}^{7} y_i$ and $P_{\gamma'_l} = \gamma'_1 + \gamma'_0$.

Proof As seen in Fig. 5.2, the S-box and the inverse S-box share block 2. Therefore, the predicted parity of this block is the same for them.

Now, we obtain the two predicted parities of block 1, i.e., $\hat{P}^1_{b1}$ and $\hat{P}^2_{b1}$ in (5.16) and (5.17). As seen from Fig. 5.1, block 1 consists of the transformation matrix $\mathbf{\Psi}$ preceded by the inverse affine transformation. Moreover, as seen in Fig. 5.1, similar to the S-box, a field multiplication, modulo-2 additions, and squaring followed by the multiplication by the constant ν are utilized in this block. Similar to the S-box, using (5.2) with the inputs $u = \sigma_l$ and $v = \sigma_h + \sigma_l$, one can obtain the result of the field multiplication in this block. Moreover, one can obtain that the result of the squarer-ν in Fig. 5.1 is

$${\sigma_h}^2\nu = (\sigma_7 + \sigma_4, \sigma_7 + \sigma_6 + \sigma_5, \sigma_4, \sigma_5). \tag{5.21}$$

By modulo-2 adding the two most and least significant bits of the result of the squarer-ν in (5.21) and that of the result of the multiplication, respectively, one can obtain

$$\hat{P}_{b1}^{1} = \sigma_3(\sigma_6 + \sigma_4) + \sigma_2(\sigma_7 + \sigma_6 + \sigma_5 + \sigma_4) + \sigma_1\sigma_6 + \sigma_0(\sigma_7 + \sigma_6) + \sigma_7 + \sigma_6 + \sigma_5 + \sigma_2, \tag{5.22}$$

$$\hat{P}_{b1}^{2} = \sigma_3\sigma_7 + \sigma_2\sigma_6 + \sigma_1\sigma_4 + \sigma_0(\sigma_5 + \sigma_4) + \sigma_6 + \sigma_2 + \sigma_0. \tag{5.23}$$

One can substitute the coordinates of σ with those of Y. This is performed by utilizing the following as the result of mixing the inverse affine and transformation matrices.

$$\begin{aligned}\boldsymbol{\sigma} &= \boldsymbol{\Psi}\boldsymbol{A}^{-1}\boldsymbol{y} + \boldsymbol{\Psi}\boldsymbol{A}^{-1}\boldsymbol{b} \\ &= \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \boldsymbol{y} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}.\end{aligned} \tag{5.24}$$

Then, by reordering the result in (5.22) and (5.23), the following is derived.

$$\hat{P}_{b1}^{1} = y_0(y_6 + y_3 + y_2 + y_1) + y_5(y_7 + y_6 + y_4 + y_3) + y_2(y_7 + y_6 + y_1) + y_7y_4 + \overline{y_7 + y_6 + y_1}, \tag{5.25}$$

$$\hat{P}_{b1}^{2} = y_1(y_7 + y_5 + y_4 + y_0) + y_2(y_7 + y_6) + y_3(y_5 + y_4) + y_5(y_4 + y_0) + y_6 + y_3 + y_2 + y_1 + y_0. \tag{5.26}$$

Using subexpression sharing, it is straightforward to obtain (5.16) and (5.17) from (5.25) and (5.26), respectively.

Now, we derive the two predicted parities of block 3 of the inverse S-box in Fig. 5.2, i.e., $\hat{P}_{b3}^{1} = \hat{P}_{X_h}$ and $\hat{P}_{b3}^{2} = \hat{P}_{X_l}$. As seen from Fig. 5.1, block 3 consists of the inverse transformation and two field multiplications. It is straightforward that considering the inverse transformation matrix we obtain $\hat{P}_{X_h}$ and $\hat{P}_{X_l}$ as follows $\hat{P}_{X_h} = \eta_7 + \eta_5 + \eta_4 + \eta_1$ and $\hat{P}_{X_l} = \eta_7 + \eta_6 + \eta_5 + \eta_2 + \eta_0$. Then, by multiplying $u = \theta'$ and $v = \sigma_h + \sigma_l$ and also $u = \theta'$ and $v = \sigma_h$ using (5.2), the coordinates of η are obtained. Substituting these in

above, the following is derived

$$\hat{P}^1_{b3} = \theta'_3(P_Y + y_6 + y_4 + 1) + \theta'_2(P_Y + y_7 + y_6 + y_3 + 1)$$
$$+ \theta'_1(y_7 + y_4 + y_2 + y_1 + 1) + \theta'_0(y_7 + y_6 + y_4 + y_2 + 1), \quad (5.27)$$
$$\hat{P}^2_{b3} = \theta'_3(y_6 + y_3 + y_1) + \theta'_2(P_Y + y_2 + y_0 + 1)$$
$$+ \theta'_1(P_Y + y_6 + y_2 + 1) + \theta'_0(P_Y + y_6 + y_4 + y_1). \quad (5.28)$$

Then, using subexpression sharing for (5.27) and (5.28), one can obtain (5.19) and (5.20) and the proof is complete. ∎

5.2.3 Merged S-box and Inverse S-box

In some low-complexity implementations that use encryption or decryption at a time, multiplicative inversions of the S-box and the inverse S-box are shared (see, for example, the joint encrypter/decrypter in [20] and [22] and the merged encryption and decryption S-boxes/inverse S-boxes in [21]). The multiplicative inversion in the finite field $GF(2^8)$ is needed for both the S-box and the inverse S-box. Therefore, one can merge them in order to reuse the multiplicative inversion and its parity predictions. It is noted that when there is no need to utilize both the S-box and the inverse S-box at the same time, this merged structure leads to a low-area design. Fig. 5.3 shows the merged S-box (SB) and inverse S-box (ISB) and their corresponding predicted parities for the three blocks. As seen in this figure, the multiplicative inversion in Fig. 5.1 is used for both the S-box and the inverse S-box. On the other hand, as seen in Fig. 5.3, two multiplexers are used for choosing the transformation matrix and the inverse and affine transformations (for the S-box with the select input $SB = 1$) and the inverse affine and transformation matrices and the inverse transformation (for the inverse S-box with the select input $ISB = 1$). The parity prediction unit is also shown in Fig. 5.3. As seen in this figure, these multiplexers also choose between the predicted parities of blocks 1 and 3 for the S-box and the inverse S-box. As a result, a parity-based fault detection merged structure is obtained.

5.2.4 Complexity Analysis

In what follows, we obtain the hardware and time complexities of the proposed schemes for the S-box and the inverse S-box. We use two-input gates in the implementation of the predicted parities of the proposed schemes. We have obtained the number of gates

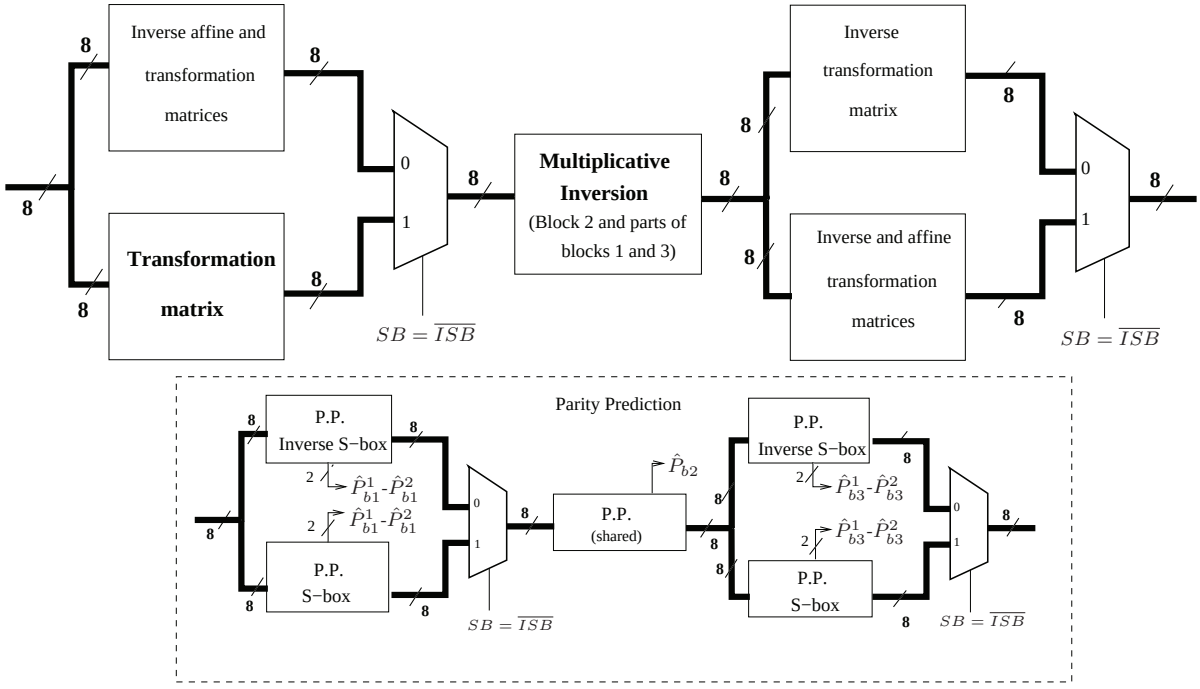


Figure 5.3: Merged S-box (SB) and inverse S-box (ISB) and the corresponding predicted parities for different blocks.

needed for implementing the predicted parities of the S-box in (5.3)-(5.7) as 33 XORs, 19 NANDs, 2 XNORs, and one NOR gate. Moreover, for the inverse S-box, one needs 40 XORs and 19 NANDs to implement (5.16)-(5.22). Furthermore, for obtaining the actual parities of blocks 1-3, 2 XORs (one XOR for each of P^1_{b1} and P^2_{b1}), 3 XORs (for P_{b2}), and 6 XORs (three XORs for each of P^1_{b3} and P^2_{b3}) are needed, respectively. Moreover, 5 XOR gates are used for comparing the five predicted and actual parities to obtain the indication flags. Later in this chapter, through ASIC implementations, we derive the chip area of the proposed schemes for the S-box and the inverse S-box. Furthermore, the area, critical path delay, and power consumption overheads are derived.

The timing overhead of the proposed scheme can be overlapped by the time needed for performing the operations in blocks 1-3. In other words, as seen in Fig. 5.2, the predicted parities are obtained concurrently with the time needed for the blocks. Table 5.1 presents the details of the timings of the three blocks for the S-box and the inverse S-box (presented in Fig. 5.1) as well as those for obtaining the predicted parities of these blocks. As seen in this table, for all the blocks, the times needed for deriving the

Table 5.1: The timing details of the proposed concurrent scheme for the S-box and the inverse S-box.

Operation	Block 1		Block 2		Block 3	
	original	predicted parity	original	predicted parity	original	predicted parity
S-box	$10T_X + 1T_A$	$5T_X + 1T_A$	$3T_X + 2T_A$	$2T_X + 1T_A$	$8T_X + 1T_A$	$6T_X + 1T_A$
Inverse S-box	$10T_X + 1T_A$	$5T_X + 1T_A$	$3T_X + 2T_A$	$2T_X + 1T_A$	$7T_X + 1T_A$	$4T_X + 1T_A$

predicted parities are less than those of the operations. Therefore, no overhead exists for obtaining these predicted parities. It is also noted that the actual parities are obtained in the time allotted to the next block. Therefore, the only timing overhead is for obtaining the actual parity of block 3 and comparing it with the corresponding predicted parity (see Fig. 5.2). These are equal to $2T_X$ and $1T_X$, respectively. Therefore, the total timing overhead is $3T_X$ for both operations.

The implementations of the S-box and the inverse S-box using composite fields are area efficient in comparison with those using LUTs. Moreover, the critical path delay can be reduced using sub-pipelining. In [17], sub-pipelining of the S-box and the inverse S-box is done by placing one, two, and three-stage registers between the blocks. Although the sub-pipelining techniques used in [17] are based on the implementations of the S-box and the inverse S-box over $GF((2^4)^2)$, similar pipelining techniques can be used for the composite field $GF(((2^2)^2)^2)$ (see, for example, [22]). The proposed fault detection scheme can take advantage of sub-pipelining without adding delay to the original pipelined structure. In the pipelined fault detection scheme, we use the parity prediction units of each pipelined block and obtain the error indication flag. According to Table 5.1, one can observe that the critical path delays of the predicted parity bits of each block of the S-box and the inverse S-box is less than the critical path delay of that block. Therefore, we can use the parity prediction schemes in the pipelined structures of the blocks without affecting the frequency of the clock signal; the predicted parity bits of the blocks are obtained in the same clock cycle as the outputs of the blocks are calculated. Calculating the actual parity and comparing it with predicted parity to obtain the error indication flag can be done in the next clock cycle. Using the above-mentioned pipelined structure, one can see that the time overhead will be only one extra clock cycle which may be overlapped with other computations in the pipelined fault detection implementation of AES.

5.3 Simulation Results

In the following, we evaluate the proposed fault detection scheme for single stuck-at errors, burst faults, and multiple random faults to model both natural faults and fault attacks.

The single stuck-at errors are at the output of the S-box (the inverse S-box). Such errors are covered 100% in the proposed scheme which is the same as those of the schemes in [57] and [78]. However, due to the technological constraints, injecting single stuck-at errors may not be applicable in practice [82]. Therefore, we rely on simulations to consider both the burst and the multiple permanent and transient faults; the details of which are presented in the following.

Burst Faults

Although the fault attacker gains more information through injecting single faults, due to the technological constraints, injecting single stuck-at faults may not be applicable in the practical fault attacks [82]. Therefore, in realistic fault attacks, multiple adjacent bits are actually flipped. Moreover, natural failures can be of the correlated type causing neighboring faults [82]. Consequently, in what follows, we consider the fault detection capability of the proposed scheme for neighboring faults referred to as burst faults.

Because of the nonlinear structure of the S-box (resp. the inverse S-box), the burst faults in a block of the S-box (resp. the inverse S-box) appear as random multiple errors at the output of that block. Moreover, the burst faults that occur in two adjacent blocks appear as multiple random errors at the outputs of the adjacent blocks. For deriving the burst fault detection capability of the proposed scheme, we have performed error simulations for blocks 1-3 of the S-box and the inverse S-box in Fig. 5.2; the details of which are presented in the following.

Linear Feedback Shift Registers (LFSRs) are used for injecting the errors at the output of one block or two adjacent blocks for modeling the burst faults. The stuck-at error model used forces multiple output bits to be stuck at logic one (for stuck-at one) or zero (for stuck-at zero) independent of the error-free values. We use Fibonacci implementation of the LFSR with 4 (for the outputs of blocks 1 and 2) or 8 (for the random input and output of block 3) output taps for injecting the errors, where the numbers, locations

and types of the errors are randomly chosen. In this regard, according to the maximum sequence length taps presented in [91], the maximum sequence length polynomial for the feedback are selected as $L_1(X) = X^4 + X$ and $L_2(X) = X^8 + X^4 + X^3 + X^2$ for the 4 and 8 output taps, respectively. Moreover, for our simulations, we use the ModelSim® SE 6.2*d* [75]. We have injected 100,000 burst faults at the outputs of the blocks for 100,000 random 8-bit inputs of the S-box and the inverse S-box. Then, we have used the five error indication flags at the outputs of three blocks of the S-box and the inverse S-box to detect the burst faults. The results of our simulations show that for the S-box and the inverse S-box 71,257 and 72,321 of the faults are detected, respectively. This yields to 71.3% and 72.3% burst fault detection capabilities for these two structures, respectively. It is noted that these are higher compared to the scheme in [57] for the original S-box and the one in [78], which have the burst fault detection capability of close to 50%. The complete comparison of the fault detection capabilities of the proposed schemes and the previous ones are presented in the next section.

Multiple Faults

The fault detection capability of the presented scheme depends on the number of the S-box and the inverse S-box blocks and the number of the predicted parities used for them. Two predicted parities have been used for blocks 1 and 3 of the S-box and the inverse S-box which constitute much of the area. Because at least one predicted parity is used for each block of the S-box and the inverse S-box, all odd number of errors in each of three blocks can be detected using the error indication flags. The error indiction flags of blocks 1 and 3 can also detect certain even number of errors comprising two odd number of errors in two partitions of these blocks. In the remaining of this section, it is shown that for the entire SubBytes, the error coverage is very close to 100% (99.998%).

For the randomly distributed multiple faults in the entire S-box and inverse S-box, the fault detection capabilities can be obtained. It is noted that in our simulations, we use a transient stuck-at error model. Nonetheless, the simulation results are also the same for the permanent errors, including the permanent internal failures and the malicious fault attacks aiming at destroying the chip. Similar to the burst faults, we use LFSRs for injecting the errors. This is performed using a 16-output tap LFSR for injecting the random multiple errors at the outputs of three blocks utilizing $L_3(X) =$

Table 5.2: Fault detection capabilities of the proposed schemes after injecting 1,000,000 random multiple faults.

Operation	Initial values	Detected	Fault Coverage (%)
S-box	$L_2 = \{9D\}_h$ $L_3 = \{AFA2\}_h$	966,324	$\approx 97\%$
	$L_2 = \{B0\}_h$ $L_3 = \{3DA9\}_h$	972,198	$\approx 97\%$
	$L_2 = \{73\}_h$ $L_3 = \{2BBF\}_h$	968,775	$\approx 97\%$
Inverse S-box	$L_2 = \{9D\}_h$ $L_3 = \{AFA2\}_h$	977,760	$\approx 98\%$
	$L_2 = \{B0\}_h$ $L_3 = \{3DA9\}_h$	969,139	$\approx 97\%$
	$L_2 = \{73\}_h$ $L_3 = \{2BBF\}_h$	971,815	$\approx 97\%$

$X^{16} + X^{12} + X^3 + X$ and an 8-bit LFSR for applying the random input of the S-box or the inverse S-box using $L_2(X) = X^8 + X^4 + X^3 + X^2$ [91].

The results of our simulations for three different initial values of the LFSRs L_2 and L_3 polynomials are depicted in Table 5.2. As seen in this table, after injecting 1,000,000 random multiple faults, the fault detection capabilities for one S-box or inverse S-box are close to 97%. It is interesting to note that for the entire SubBytes or inverse SubBytes, i.e., 16 S-boxes or inverse S-boxes, respectively, injecting this number of multiple faults resulted in the fault detection of very close to 100% (99.998%). As a matter of fact, in this case, the faults are detected by the $5 \times 16 = 80$ flags for the entire SubBytes or inverse SubBytes transformations, yielding to approximately complete fault detection capabilities, i.e., approximately $100 \times (1 - 2^{-80})\%$.

5.4 ASIC Implementations and Comparisons

In this section, we present the results of the syntheses we have performed for the proposed and previously presented fault detection schemes of the S-box and the inverse S-box. We have used the STM 65-nm CMOS standard technology [74] for the syntheses. Moreover, VHDL has been used as the design entry to the Synopsys Design Vision [73]. We have set the target frequency as 500 MHz, 1 GHz, and 1.1 GHz corresponding to the delays of 2 ns, 1 ns, and 0.91 ns, respectively. Using Synopsys Design Vision, we have obtained the

maximum target frequency in which our fault detection structure can operate without violating the timing constraints. This maximum target frequency has been obtained as 1.1 GHz in the 65-nm technology. The proposed fault detection schemes and the ones presented in [34], [36], [38], [39], [42], [57], [78], [79], and [92] have been synthesized and their areas, delays and power consumptions are derived. The results for different target frequencies are shown in Table 5.3 (for the S-box) and Table 5.4 (for the inverse S-box). As seen in these tables, areas (μm^2), critical path delays (ns), total power consumptions (μW), and fault coverages (%) are shown. In the following, the syntheses details of the structures are explained.

As seen in Table 5.3 for the S-box, the first three schemes, i.e., the schemes presented in [34], [42], [39], and [36], use the LUT S-box in their structures. The schemes in [34] and [42] use the S-box followed by the inverse S-box. These can be implemented using two 256×8 LUTs. Then, the result is compared with the input to detect the faults in the structure of the S-box or the inverse S-box. It is noted that although its fault detection capability reaches 100%, this method has the critical path delay and the area overheads of close to 100%. Furthermore, as seen in Table 5.3, because of the use of LUT S-box, areas and power consumptions are higher than the schemes using composite fields.

Table 5.3: Comparing the areas, critical path delays, power consumptions, and fault detection capabilities of the proposed and previously presented fault detection schemes for the S-box using the 65-nm CMOS standard technology.

Fault detection scheme	Target frequency: 500 MHz			Target frequency: 1 GHz			Target frequency: 1.1 GHz			Fault coverage (%)	
	Area (μm^2)	Delay (ns)	Total power (μW)	Area (μm^2)	Delay (ns)	Total power (μW)	Area (μm^2)	Delay (ns)	Total power (μW)	Burst faults	Multiple faults
Redun. units [34], United S-box [42] (LUTs)	52.3 $\times 10^3$	1.23	7.2 $\times 10^3$	54.2 $\times 10^3$	0.95	15.4 $\times 10^3$	54.7 $\times 10^3$	0.87	16.9 $\times 10^3$	100%	100%
Parity-based scheme in [39] (256×9 LUT)	29.5 $\times 10^3$	0.59	4.3 $\times 10^3$	29.5 $\times 10^3$	0.59	8.4 $\times 10^3$	29.5 $\times 10^3$	0.59	9.5 $\times 10^3$	$\approx 50\%$ (SubB.)	$\approx 50\%$ (SubB.)
Parity-based scheme in [36] (512×9 LUT)	57.1 $\times 10^3$	0.68	7.8 $\times 10^3$	57.1 $\times 10^3$	0.68	15.6 $\times 10^3$	57.1 $\times 10^3$	0.68	17.1 $\times 10^3$	$\approx 50\%$	$\approx 50\%$
Multiplication approach in [38] (polynomial basis)	876	1.88	630.3	1829	0.96	3000.7	2121	0.88	3600.1	$\approx 75\%$ (mult. inv.)	$\approx 75\%$ (mult. inv.)
Struc.-independent scheme in [92] (polynomial basis)	754	1.90	574.9	1459	0.97	2263,8	1763	0.87	2902.5	$\approx 50\%$	$\approx 50\%$
Scheme in [57] for original S-box (polynomial basis)	881	1.92	607.7	1748	0.96	2709.4	Target is not achieved	Target is not achieved	Target is not achieved	$\approx 50\%$	$\approx 97\%$
Parity-based scheme in [79] (polynomial basis)	865	1.82	616.2	1645	0.96	2507.8	1742	0.88	2921.8	$\approx 50\%$	$\approx 97\%$
Scheme in [78] (normal basis)	858	1.90	620.0	1755	1.0	2672.9	Target is not achieved	Target is not achieved	Target is not achieved	$\approx 50\%$	$\approx 97\%$
This Chapter (polynomial basis)	953	1.80	712.3	1683	0.95	2600.2	1730	0.87	2912.2	71.3%	$\approx 97\%$

Additionally, the schemes in [39] and [36] use the error detecting codes (parity) for the LUT S-box, where the S-box is expanded. Similar to the scheme in [34] and [42], using the LUT S-box increases the areas and power consumptions of these schemes considerably. In the low-cost scheme presented in [39], the modulo-2 addition of the predicted parities of the input and output of the S-box along with the S-box itself are stored in a 256×9 LUT. Then, a comparison with the actual parities is performed for deriving the error indication flags. As seen in Table 5.3, the burst and multiple fault detection capabilities of this scheme for the entire SubBytes (not each S-box) is around 50%. The parity-based scheme presented in [36] utilizes a 512×9 LUT to store the predicted parities as well as the output of the S-box. This results in reaching the burst and multiple fault detection capability of approximately 50% for each S-box at the cost of more area and power consumption and slightly more delay compared to the scheme in [39].

As presented in Table 5.3, the last six fault detection schemes use the S-box using composite fields; represented either in polynomial basis or normal basis. It is noteworthy that sub-pipelining of these fault detection S-boxes has not been performed and these syntheses are only intended to compare different presented schemes. The scheme in [38] uses two flags for the fault detection of the non-linear part of the S-box, i.e., the multiplicative inversion. This is performed by comparing the result of multiplying the input and the output of the multiplicative inversion with the actual result, i.e., $\{01\}_2$. As seen in Table 5.3, this yields to the fault detection capability of approximately 75%. The structure-independent scheme in [92] uses one-bit parity in the multiplication scheme for obtaining the fault detection capability of around 50% for the S-box. Although the fault detection capability is less than that of [38], as seen in Table 5.3, better area and power consumption results are obtained.

The results for the proposed scheme in this chapter are shown in bold face in Table 5.3. As depicted in the table, for the target frequency of 1.1 GHz, the proposed scheme in this chapter for the S-box has the least area, power consumption, and critical path delay among the schemes that have similar or slightly more fault detection capabilities, i.e., the schemes presented in [34], [42], [57], [79] and [78]. Specifically, compared to the schemes presented in [57], [78], and [79], for the low frequency of 500 MHz, the presented scheme in this chapter is faster at the expense of more area. Nonetheless, as seen from the table, the maximum target frequency of 1.1 GHz cannot be achieved for the schemes of [57]

and [78]. Nevertheless, in higher frequencies, e.g., 1.1 GHz in Table 5.3, the presented scheme outperforms the one proposed in [79] in terms of area, power consumption and delay. It is also noted that the schemes proposed in [57], [78], and [79] yield to the fault detection capability of around 50% for the burst faults which is less compared to the presented scheme in this chapter.

It is also noted that compared to the schemes with lower fault detection capability in Table 5.3, for this maximum target frequency, the proposed scheme is more compact. Moreover, it has less power consumption except for the scheme presented in [92]. Nonetheless, the fault detection capabilities of the structure-independent scheme in [92] for burst and multiple faults are around 50%, i.e., approximately half of that of the proposed scheme for the multiple faults and less for burst faults. Finally, using sub-pipelining, the critical path delay of the proposed scheme can be considerably reduced. This can result in even better critical path delays compared to the schemes using LUTs at the expense of more hardware utilizations for the pipelining registers. It is noted that the sub-pipelined composite field structures are still much more compact than the schemes taking advantage of LUTs.

We have also implemented the proposed scheme for the inverse S-box for the three target frequencies; the results of which are presented in Table 5.4 in bold face. As seen in this table, in addition, the schemes for the inverse S-box presented in [34], [42], [36], [39], [92], [57], [79] and [78] have been synthesized and their areas, delays and power consumptions are derived. As seen from Table 5.4, similar to the S-box, for the low frequency of 500 MHz, the presented scheme for the inverse S-box is the fastest compared to [57], [78], and [79]. Additionally, for the maximum target frequency of 1.1 GHz, it has the lowest area, delay and power consumption compared to those of [57], [78], and [79]. It is also noted that as presented in Table 5.4, the target frequency of 1.1 GHz cannot be achieved by the scheme in [79].

Table 5.4: Comparing the areas, critical path delays, power consumptions, and fault detection capabilities of the proposed and previously presented fault detection schemes for the inverse S-box using the 65-nm CMOS standard technology.

Fault detection scheme	Target frequency: 500 MHz			Target frequency: 1 GHz			Target frequency: 1.1 GHz			Fault coverage (%)	
	Area (μm^2)	Delay (ns)	Total power (μW)	Area (μm^2)	Delay (ns)	Total power (μW)	Area (μm^2)	Delay (ns)	Total power (μW)	Burst faults	Multiple faults
Redun. units [34], United S-box [42] (LUTs)	52.3 $\times 10^3$	1.23	7.2 $\times 10^3$	54.2 $\times 10^3$	0.95	15.4 $\times 10^3$	54.7 $\times 10^3$	0.87	16.9 $\times 10^3$	100%	100%
Parity-based scheme in [39] (256×9 LUT)	29.5 $\times 10^3$	0.59	4.3 $\times 10^3$	29.5 $\times 10^3$	0.59	8.4 $\times 10^3$	29.5 $\times 10^3$	0.59	9.5 $\times 10^3$	$\approx 50\%$ (Inv. SubB.)	$\approx 50\%$ (Inv. SubB.)
Parity-based scheme in [36] (512×9 LUT)	57.1 $\times 10^3$	0.68	7.8 $\times 10^3$	57.1 $\times 10^3$	0.68	15.6 $\times 10^3$	57.1 $\times 10^3$	0.68	17.1 $\times 10^3$	$\approx 50\%$	$\approx 50\%$
Struc.-independent scheme in [92] (polynomial basis)	783	1.72	581.3	1450	0.97	2262.6	1683	0.89	2893.4	$\approx 50\%$	$\approx 50\%$
Scheme in [57] for original S-box (polynomial basis)	886	1.85	629.4	1689	0.97	2711.1	1993	0.88	3612.6	$\approx 50\%$	$\approx 97\%$
Parity-based scheme in [79] (polynomial basis)	865	1.85	623.6	1667	0.96	2692.3	1964	0.88	3528.5	$\approx 50\%$	$\approx 97\%$
Parity-based scheme in [79] (normal basis)	855	1.85	574.0	1578	1.0	2374.4	Target is not achieved	Target is not achieved	Target is not achieved	$\approx 50\%$	$\approx 97\%$
This Chapter (polynomial basis)	916	1.68	636.4	1481	0.96	2200.5	1709	0.88	2812.8	72.3%	$\approx 97\%$

As depicted in Table 5.4, for the highest frequency to achieve, i.e., 1.1 GHz, the proposed scheme in this chapter is the most compact scheme with the lowest power consumption compared to the schemes presented in [34], [42], [36], [39], [57], [79] and [78]. It is also noted that similar to the S-box, the fault detection structure of the inverse S-box can be sub-pipelined so that with a reasonable hardware overhead, the critical path delay is highly reduced. The proposed scheme in this chapter has more area and less power consumption compared to the one in [92]. As mentioned previously, however, the fault detection capability of the scheme in [92] for the burst and multiple faults is around 50%. This is less than the fault detection capabilities of 97% and 72.3% for the proposed scheme for the multiple and burst faults, respectively.

Furthermore, we have compared the areas, critical path delays, and power consumptions of the proposed schemes for the S-box and the inverse S-box with those for the original ones presented in [22]. For this purpose, we have implemented both the original and the fault detection S-box and inverse S-box for several target frequencies ranging from 500 MHz to 1.1 GHz. The results are shown in Fig. 5.4. As seen in Fig. 5.4a and Fig. 5.4d for the S-box and the inverse S-box, respectively, the areas of both the original structures (solid lines with $\circ$ marks) and the fault detection ones (dotted lines with + marks) for different target frequencies are depicted. As seen in these figures, as the target frequency increases, it is reached by increasing the occupied area. This yields to having the areas ranging from 698 μm^2 - 1338 μm^2 and 662 μm^2 - 1334 μm^2 for the original S-box and inverse S-box, respectively. Moreover, for the fault detection S-box and inverse S-box presented in this chapter, the areas of 953 μm^2 - 1730 μm^2 and 916 μm^2 - 1709 μm^2 are achieved, respectively.

Moreover, the results of our implementations for the power consumptions of the original and the fault detection S-box and inverse S-box are depicted in Fig. 5.4b and Fig. 5.4e, respectively. As seen from these figures, for the low target frequencies, the power consumptions of the original structures and the fault detection ones are close to each other. Nonetheless, as seen in Fig. 5.4b and Fig. 5.4e, these differences increase after applying tighter critical path delay constraints. As an example, for the target frequency of 1.1 GHz, the power consumption for the original S-box (resp. inverse S-box) becomes 2.2 mW (2.3 mW). Moreover, for the fault detection S-box (resp. inverse S-box) it reaches 2.9 mW (2.8 mW). Finally, the critical path delays of the original structures

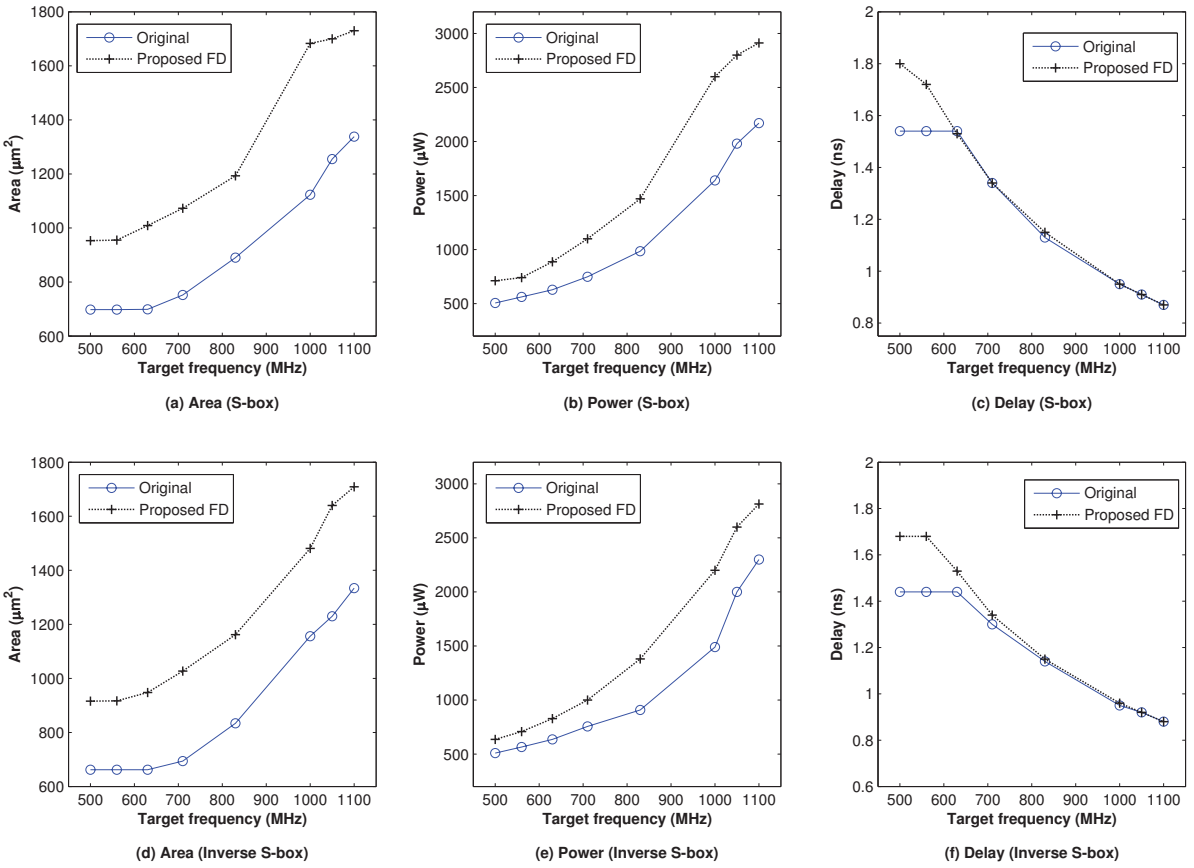


Figure 5.4: The areas, critical path delays, and power consumptions of the original [22] and the proposed fault detection S-box and inverse S-box.

and those for the proposed scheme in this chapter for the S-box and the inverse S-box are presented in Fig. 5.4c and Fig. 5.4f. As seen in these figures, for the target frequency of 500 MHz, the critical path delays of the original and the fault detection S-box are 1.54 ns (working frequency of 649 MHz) and 1.80 ns (working frequency of 555 MHz), respectively. Furthermore, for the inverse S-box, the critical path delays of 1.44 ns (working frequency of 694 MHz) and 1.68 ns (working frequency of 595 MHz) are obtained for the original and fault detection structures, respectively. It is also noted that, for the maximum target frequency to achieve, the original and fault detection S-box (inverse S-box) reaches the critical path delay of 0.87 ns (0.88 ns), i.e., the working frequency of 1.15 GHz (1.14 GHz). As seen in Fig. 5.4, this is for the cost of the increased areas and power consumptions for the structures.

We conclude this section by deriving the area, delay, and power consumption over-

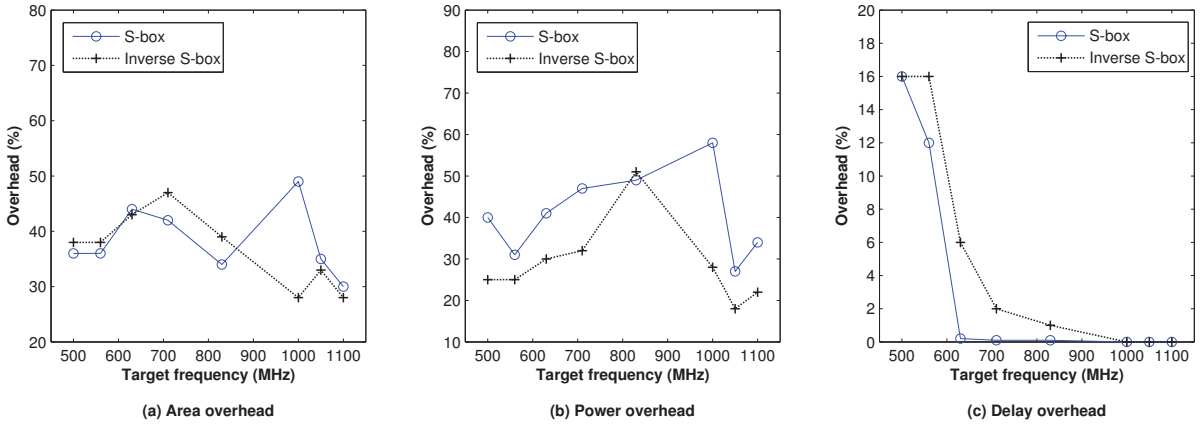


Figure 5.5: The Area, delay, and power consumption overheads of the proposed schemes for the S-box and the inverse S-box.

heads of the proposed scheme for the S-box and the inverse S-box. To this end, we have considered the areas, delays, and power consumptions of the original operations presented in [22] and the fault detection structures shown in Fig. 5.4. Then, we have obtained the overheads; the results of which are presented in Fig. 5.5. The results in this table show that for the low frequency of 500 MHz for the S-box (see the dotted lines with $\circ$ marks) and the inverse S-box (see the solid lines with + marks), the area overheads are approximately 36% and 38%, respectively (see Fig. 5.5a). Moreover, in this frequency, the overheads for the critical path delays and the power consumptions for the S-box are 16% and 40%, respectively. Additionally, for the inverse S-box, for the target frequency of 500 MHz, the critical path delay and the power consumption overheads of 16% and 25% are obtained, respectively. However, as we increase the target frequency, the critical path delay overhead decreases (see Fig. 5.5c). It is noted that as seen in Fig. 5.5c, no timing overhead is observed for the target frequencies higher than 1 GHz. Finally, as presented in earlier in this chapter, with the mentioned overheads, the fault detection scheme proposed in this chapter achieves high fault coverages. This makes the presented fault detection S-box and inverse S-box suitable choices in counteracting the fault attacks and detecting the internal failures.

5.5 Formulations for Mixed Bases

The hardware implementation of the S-box using mixed bases has been presented recently in [62]. In this S-box, in contrast to the conventional works, polynomial and normal bases are used in mixture (mixed bases). The theoretical analysis in [62] shows lower timing complexity for this S-box compared to the ones using polynomial and normal bases presented in [20] and [23]. In what follows, we present a multi-bit parity-based fault diagnosis approach for the S-box using mixed bases. We derive formulations for these multi-bit parities and optimize them to reach low complexity. Based on the presented formulations, we present two different flavors for our fault detection scheme considering compromise between fault detection capability and resources needed. Moreover, we evaluate the fault detection capabilities of the proposed reliable architectures.

We present the formulations for the five predicted parities of the mixed bases S-box of Fig. 5.6 in the following theorem.

Theorem 5.3 *For $X \in GF(2^8)$ as the input of the S-box, the predicted parities of block 1 ($\hat{P}_1$-$\hat{P}_2$), block 2 ($\hat{P}_3$), and block 3 ($\hat{P}_4$-$\hat{P}_5$) of the S-box in Fig. 5.6 are obtained as follows*

$$\begin{aligned}
\hat{P}_1 &= x_6(x_7 + x_5 + x_0) + x_5 Z_3 + x_4(Z_7 + Z_1 + x_2) \\
&\quad + x_2(\overline{x_5 + x_3}) + x_1 Z_1 + (x_7 \vee x_4) + (x_5 \vee x_1), && (5.29)\\
\hat{P}_2 &= x_6(Z_4 + x_1 + x_0) + x_4\overline{Z_6} + x_3 x_7 + x_0 Z_2 + \\
&\quad (x_7 \vee x_5) + (x_2 \vee x_1), && (5.30)\\
\hat{P}_3 &= (\gamma_3\gamma_1 \vee \gamma_2) + \gamma_3\gamma_2\gamma_0, && (5.31)\\
\hat{P}_4 &= \theta_3(Z_4 + Z_3 + x_6) + \theta_2(Z_8 + x_4) + \theta_1(Z_6 \\
&\quad + x_6) + \theta_0(Z_5 + Z_2 + x_2), && (5.32)\\
\hat{P}_5 &= \theta_3(Z_8 + Z_4) + \theta_2(Z_7 + x_7) + \theta_1(Z_9 + Z_5) + \\
&\quad \theta_0(Z_5 + x_7 + x_1), && (5.33)
\end{aligned}$$

where $Z_1 = x_3 + x_0$, $Z_2 = x_5 + x_1$, $Z_3 = Z_2 + Z_1$, $Z_4 = x_7 + x_2$, $Z_5 = x_6 + x_3$, $Z_6 = Z_1 + x_5$, $Z_7 = x_6 + x_1$, $Z_8 = Z_7 + x_0$, and $Z_9 = Z_5 + Z_2$. Moreover, "+" and $\vee$ represent modulo-2 addition using an XOR gate and the OR operation, respectively.

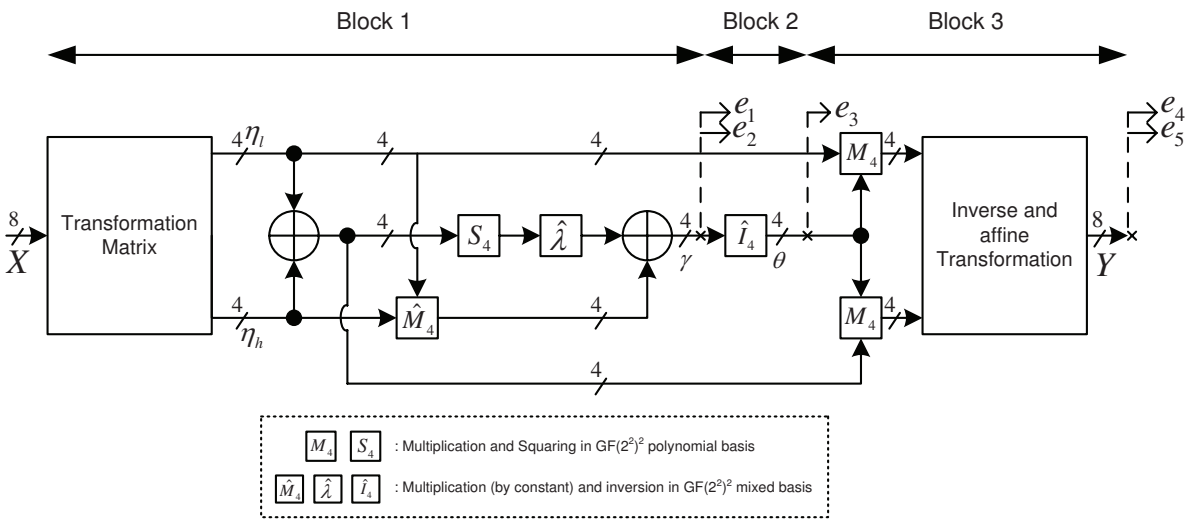


Figure 5.6: The presented fault detection structure for the mixed bases S-box [62].

Proof The two predicted parities of block 1, i.e., $\hat{P}_1$ and $\hat{P}_2$ in (5.29) and (5.30), are obtained according to Fig. 5.6. As seen in Fig. 5.6, block 1 consists of a transformation matrix ($\boldsymbol{T}$) which transforms the coordinates of X in binary field to η in composite field and is defined as [62]:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}. \tag{5.34}$$

Moreover, considering [62] and Fig. 5.6, for the input $(\eta_7+\eta_3, \eta_6+\eta_2, \eta_5+\eta_1, \eta_4+\eta_0)$, we obtain the merged S_4-$\hat{\lambda}$ in Fig. 5.6 as $(\eta_6+\eta_2, \eta_7+\eta_3, \eta_7+\eta_5+\eta_3+\eta_1, \eta_7+\eta_6+\eta_5+\eta_4+\eta_3+\eta_2+\eta_1+\eta_0)$. In addition, according to [62] and for inputs η_l and η_h, one can obtain the result of the field multiplication ($\hat{M}_4$) in this block as

$$\begin{aligned} c'_0 &= \eta_7(\eta_3+\eta_1+\eta_0)+\eta_6(\eta_2+\eta_1)+\eta_5(\eta_3+\eta_2+\eta_1+\eta_0)+\eta_4(\eta_3+\eta_1), \\ c'_1 &= \eta_7(\eta_3+\eta_2+\eta_0)+\eta_6(\eta_3+\eta_1+\eta_0)+\eta_5(\eta_2+\eta_0)+\eta_4(\eta_3+\eta_2+\eta_1+\eta_0), \\ c'_2 &= \eta_7\eta_2+\eta_6(\eta_3+\eta_2)+\eta_5(\eta_1+\eta_0)+\eta_4\eta_1, \\ c'_3 &= \eta_7\eta_3+\eta_6\eta_2+\eta_5\eta_0+\eta_4(\eta_1+\eta_0). \end{aligned} \tag{5.35}$$

Therefore, by modulo-2 adding the coordinates of (γ_3, γ_2) and (γ_1, γ_0), i.e., two most and least significant bits of squarer-$\hat{\lambda}$ in Fig. 5.6 and result of the multiplication in (5.35), one can obtain

$$\begin{aligned}\hat{P}_1 &= \eta_7(\eta_3 + \eta_2) + \eta_6\eta_3 + \eta_5\eta_1 + \eta_4\eta_0 + \eta_7 + \\ &\quad \eta_6 + \eta_3 + \eta_2, \qquad (5.36)\\ \hat{P}_2 &= \eta_7(\eta_2 + \eta_1) + \eta_6(\eta_3 + \eta_2 + \eta_0) + \eta_5(\eta_3 + \\ &\quad \eta_1) + \eta_4(\eta_2 + \eta_0) + \eta_6 + \eta_4 + \eta_2 + \eta_0. \qquad (5.37)\end{aligned}$$

By substituting the coordinates of η with those of X using (5.34), reordering, and using subexpression sharing, it is straightforward to obtain (5.29) and (5.30) from (5.36) and (5.37), respectively.

Block 2 of the S-box in Fig. 5.6 consists of an inversion in $GF((2^2)^2)$. According to [62], we have obtained the formulations for the inversion of the input $\gamma \in GF((2^2)^2)$ as $\theta \in GF((2^2)^2) = (\gamma)^{-1}$ according to the following

$$\begin{aligned}\theta_0 &= (\gamma_2 + \gamma_0)I_1 + (\gamma_3 + \gamma_1)(I_1 + I_0),\\ \theta_1 &= (\gamma_2 + \gamma_0)(I_1 + I_0) + (\gamma_3 + \gamma_1)I_0,\\ \theta_2 &= \gamma_0 I_1 + \gamma_1(I_1 + I_0), \qquad (5.38)\\ \theta_3 &= \gamma_0(I_1 + I_0) + \gamma_1 I_0.\end{aligned}$$

where $I_1 = \gamma_3(\gamma_1 + \gamma_0) + \gamma_2\overline{\gamma_1} + \gamma_0$ and $I_0 = \gamma_3\overline{\gamma_0} + \gamma_2 \vee \gamma_1 + \gamma_0\overline{\gamma_2}$. Then, one can obtain from (5.38) that $\hat{P}_3 = \gamma_2 I_0 + \gamma_3 I_1$ which leads to $\hat{P}_3 = \gamma_2 + \gamma_3\gamma_1 + \gamma_3\gamma_2\gamma_1 + \gamma_3\gamma_2\gamma_0$. Noting that $\gamma_2 + \gamma_3\gamma_1 + \gamma_3\gamma_2\gamma_1 = \gamma_2 \vee \gamma_3\gamma_1$, one can obtain (5.31).

Now, we derive the two predicted parities of block 3, i.e., $\hat{P}_4$ and $\hat{P}_5$. Let $U = (u_3, u_2, u_1, u_0)$ and $V = (v_3, v_2, v_1, v_0)$ be the inputs of a multiplier in $GF((2^2)^2)$ in block 3 (M_4). Then, the result of multiplication is

$$\begin{aligned}c_0 &= u_3v_2 + u_2(v_3 + v_2) + u_1(v_1 + v_0) + u_0v_1,\\ c_1 &= u_3v_3 + u_2v_2 + u_1v_0 + u_0(v_1 + v_0),\\ c_2 &= u_3(v_3 + v_2 + v_1 + v_0) + u_2(v_3 + v_1) + \\ &\quad u_1(v_3 + v_2) + u_0v_3, \qquad (5.39)\\ c_3 &= u_3(v_2 + v_0) + u_2(v_3 + v_2 + v_1 + v_0) + u_1v_2 + \\ &\quad u_0(v_3 + v_2).\end{aligned}$$

Moreover, as seen from Fig. 5.6, block 3 consists of the mixed inverse ($\boldsymbol{T}^{-1}$) and affine ($\boldsymbol{A}$) transformation matrices. The formulation for this mixed transformation (which is added later with constant $\{63\}_h$) is as follows [62].

$$\boldsymbol{A} \times \boldsymbol{T}^{-1} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{5.40}$$

Finally, according to (5.40), the two predicted parities of block 3 in Fig. 5.6 are obtained by adding the multiplications output coordinates 3 and 4 (for $\hat{P}_4$) and 1, 3, 5, 6, and 7 (for $\hat{P}_5$). Then, according to (5.39), by multiplying $U = \theta$ and $V = \eta_h + \eta_l$ and also $U = \theta$ and $V = \eta_l$, one can obtain these coordinates. The following is obtained for the two predicted parities of block 3 of the S-box in Fig. 5.6:

$$\begin{aligned} &\hat{P}_4 = \theta_3(\eta_6 + \eta_0) + \theta_2(\eta_7 + \eta_6 + \eta_1 + \eta_0) + \\ &\theta_1(\eta_5 + \eta_4 + \eta_2 + \eta_1 + \eta_0) + \theta_0(\eta_5 + \eta_3 + \eta_2 + \eta_1), \end{aligned} \tag{5.41}$$

$$\begin{aligned} &\hat{P}_5 = \theta_3(\eta_5 + \eta_3 + \eta_2 + \eta_1 + \eta_0) + \theta_2(\eta_4 + \eta_3 + \\ &\eta_1) + \theta_1(\eta_7 + \eta_4 + \eta_3 + \eta_2) + \theta_0(\eta_6 + \eta_5 + \\ &\eta_4 + \eta_3). \end{aligned} \tag{5.42}$$

Then, using subexpression sharing and by substituting the coordinates of η with those of X in (5.41) and (5.42) using (5.34), one can obtain (5.32) and (5.33) and the proof is complete.

The critical path delay of the structure presented in Fig. 5.6 is determined by that of the S-box and the fault detection scheme. Because of the concurrency of the scheme in Fig. 5.6, the predicted parities of all the three blocks and the actual parities of blocks 1 and 2 are obtained during the computations of the S-box itself. Thus, the only delay that the scheme in Fig. 5.6 adds to the architecture is the delay of computing the actual parities of block 3 and their corresponding comparisons with the predicted parities for obtaining the error indication flags e_4 and e_5. It is interesting to note that

having two predicted parities for the last block (instead of one) reduces the critical path delay overhead. Based on these observations, the critical path delay of the presented fault detection structure in Fig. 5.6 is just $3T_X$ ($2T_X$ for computing the actual parities of block 3 and $1T_X$ for obtaining their error indication flags).

5.5.1 Other Variants

An advantage for the scheme proposed in Theorem 5.3 is that based on the reliability requirements and the available resources, one may use different number of predicted parities for different blocks. For instance, for applications which have tight resource constraints, one may use one predicted parity for each block, i.e., three predicted parities in total for the entire S-box, to reduce the performance metrics overheads at the expense of reducing the error coverage. This can be performed by modulo-2 adding (5.29) and (5.30) to obtain one predicted parity for block 1 and (5.32) and (5.33) for the one for block 3 (see also Fig. 5.6). In other words we can use $\hat{P}_{1+2} = \hat{P}_1 + \hat{P}_2$ (using (5.36) and (5.37)), $\hat{P}_{4+5} = \hat{P}_4 + \hat{P}_5$ (using (5.41) and (5.42)), and $\hat{P}_3$ as three predicted parities for the S-box in Fig. 5.6. For $\hat{P}_{1+2}$ and $\hat{P}_{4+5}$ we have

$$\begin{aligned}\hat{P}_{1+2} &= \eta_7(\eta_3+\eta_1) + \eta_6(\eta_2+\eta_0) + \eta_5\eta_3 + \eta_4\eta_2 + \\ &\quad \eta_7 + \eta_4 + \eta_3 + \eta_0, \qquad (5.43)\\ \hat{P}_{4+5} &= \theta_3(\eta_6+\eta_5+\eta_3+\eta_2+\eta_1) + \theta_2(\eta_7+\eta_6+ \\ &\quad \eta_4+\eta_3+\eta_0) + \theta_1(\eta_7+\eta_5+\eta_3+\eta_1+\eta_0) + \\ &\quad \theta_0(\eta_6+\eta_4+\eta_2+\eta_1). \qquad (5.44)\end{aligned}$$

Our simulations (if the entire SubBytes is considered) show multiple random error coverage of very close to 100% (99.998%) for mixed bases S-box. In addition, we have performed ASIC syntheses using a 65-nm CMOS standard technology for the proposed concurrent fault detection architectures and some of the previous ones. Compared to the approaches with similar error coverage, the proposed approach in this section is the most efficient one, reaching the efficiency of 5.02 $\frac{Mbps}{\mu m^2}$ while maintaining the throughput of 5 Gbps. Based on the error coverage needed and the performance requirements, one may use the proposed high-speed concurrent fault detection approach to reach the desired coverage/performance goals.

Chapter 6

Concurrent Structure-Independent Fault Detection Schemes for the AES

IN the previous two chapters, we proposed two methods for fault detection of the S-boxes and inverse S-boxes using composite fields. In this chapter, we propose a structure-independent fault detection scheme for the entire AES encryption and decryption. Specifically, we obtain new formulations for the fault detection of SubBytes and inverse SubBytes using the relation between the input and the output of the S-box and the inverse S-box. The proposed schemes are independent of the way the S-box and the inverse S-box are constructed. Therefore, they can be used for both the S-boxes and the inverse S-boxes using look-up tables and those utilizing logic gates based on composite fields. Our simulation results show the error coverage of greater than 99% for the proposed schemes. Moreover, the proposed and the previously reported fault detection schemes have been implemented on the most recent Xilinx® Virtex™ FPGAs. Their area and delay overheads have been compared and it is shown that the proposed schemes outperform the previously reported ones.

As presented before (see Fig. 2.3), a multiplication-based scheme is presented in [38]. In this scheme, the result of the multiplication of the input and the output of the multiplicative inversion is compared with the predicted result of unity. However, this scheme is not suitable for the S-boxes and inverse S-boxes implemented using look-up tables (LUTs). This is because the output (the input) of the multiplicative inversion in the S-box (the inverse S-box) may not be accessible in the LUT-based implementations. Therefore, the fault detection scheme presented in [38] is not applicable for these

implementations.

In this chapter, we present structure-independent fault detection schemes for obtaining a reliable AES implementation. We present a systematic method for obtaining the fault detection signatures for the multiplicative inversion of the S-boxes (inverse S-boxes). We propose new formulations resulting in novel fault detection schemes for checking SubBytes, inverse SubBytes, and the other transformations in the encryption and the decryption of the AES. The proposed schemes are independent of the method the S-box (resp. the inverse S-box) is implemented. Thus, they can be applied to both the LUT and composite fields implementations. Moreover, we simulate the proposed fault detection structures for the AES encryption and decryption. Through our simulations after injecting up to 700,000 random stuck-at errors, we show that the proposed low cost schemes reach the error coverage of greater than 99%. Finally, our proposed fault detection schemes and almost all of the previously reported ones are implemented on the recent Xilinx® Virtex™ FPGAs and their area and delay overheads have been derived and compared. The FPGA implementation results show the low area and delay overheads for the proposed fault detection schemes.

The organization of this chapter is as follows: In Section 6.1, we present some brief preliminaries regarding the AES algorithm. The proposed structure-independent schemes for the fault detection of the S-boxes and the inverse S-boxes are presented in Section 6.2. Then, the fault detection schemes for the entire AES encryption and decryption are considered in Section 6.3. In Section 6.4, the results of the simulations of the proposed schemes are presented and their error coverages are obtained. In Section 6.5, the presented fault detection schemes and the previously reported ones are implemented on FPGAs and they are compared in terms of time and space complexities. The results presented in this chapter can also be found in [92] and [93].

6.1 Notations Used in This Chapter

In this section, we briefly present the notations and preliminaries used throughout this chapter for the four transformations of each round of the encryption and the decryption in the AES.

Each transformation in every round acts on its 128-bit input denoted as the *state*. The

states are considered as four by four matrices whose entries are eight bits. For example, the input state $\boldsymbol{S}$ with its 8-bit entries, i.e., $s_{r,c}$, $0 \leq r, c \leq 3$, is represented as follows:

$$\boldsymbol{S} = [s_{r,c}]_{r,c=0}^{3}. \tag{6.1}$$

6.1.1 AES Encryption

Considering (6.1) as the input state of an encryption round, the transformations in each round of encryption (except for the last round) are as follows [1]:

- **SubBytes**: The first transformation in each round is the bytes substitution (*SubBytes*) implemented by 16 S-boxes. Let $s_{r,c} \in GF(2^8)$ and $s'_{r,c} \in GF(2^8)$ be the 8-bit input and output of each S-box, respectively. Then, the S-box consists of a multiplicative inversion, i.e., $s_{r,c}^{-1} \in GF(2^8)$, followed by an affine transformation consisting of the matrix $\boldsymbol{\Gamma}$ and the vector $\boldsymbol{\gamma}$ to generate the output as

$$\boldsymbol{s}'_{r,c} = \boldsymbol{\Gamma}\boldsymbol{s}_{r,c}^{-1} + \boldsymbol{\gamma} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \boldsymbol{s}_{r,c}^{-1} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}. \tag{6.2}$$

  The 8-bit outputs of 16 S-boxes are used to obtain the output state of the SubBytes transformation as

$$\boldsymbol{S}' = [s'_{r,c}]_{r,c=0}^{3}. \tag{6.3}$$

- **ShiftRows**: In the second transformation, *ShiftRows*, four bytes of the rows of the input state are cyclically shifted to the left and the first row is left unchanged to obtain the output state, i.e., $SR(\boldsymbol{S}')$, as

$$SR(\boldsymbol{S}') = \begin{pmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,1} & s'_{1,2} & s'_{1,3} & s'_{1,0} \\ s'_{2,2} & s'_{2,3} & s'_{2,0} & s'_{2,1} \\ s'_{3,3} & s'_{3,0} & s'_{3,1} & s'_{3,2} \end{pmatrix} = [s'_{r,(r+c)\,mod\,4}]_{r,c=0}^{3}. \tag{6.4}$$

- **MixColumns**: In the third transformation, *MixColumns*, the output state is obtained by multiplying a constant matrix with the output state of ShiftRows, $SR(\boldsymbol{S}')$

in (6.4), to obtain the output state of MixColumns, i.e., the matrix $\boldsymbol{S''}$, as

$$\boldsymbol{S''} = [s''_{r,c}]_{r,c=0}^{3} = \begin{pmatrix} \{2\}_h & \{3\}_h & \{1\}_h & \{1\}_h \\ \{1\}_h & \{2\}_h & \{3\}_h & \{1\}_h \\ \{1\}_h & \{1\}_h & \{2\}_h & \{3\}_h \\ \{3\}_h & \{1\}_h & \{1\}_h & \{2\}_h \end{pmatrix} SR(\boldsymbol{S'}). \tag{6.5}$$

- **AddRoundKey**: The final transformation is *AddRoundKey* in which the input state is added (modulo-2) with the key of the round. Considering the roundkey input state as the matrix $\boldsymbol{K} = [k_{r,c}]_{r,c=0}^{3}$, with entries $k_{r,c}$, $0 \leq r, c \leq 3$, the output state of the AddRoundKey transformation, i.e., $\boldsymbol{O}$, is obtained as

$$\boldsymbol{O} = [o_{r,c}]_{r,c=0}^{3} = \boldsymbol{S''} + \boldsymbol{K}. \tag{6.6}$$

6.1.2 AES Decryption

In the AES decryption rounds, four transformations, i.e., *InvShiftRows*, *InvSubBytes*, *AddRoundKey* and *InvMixColumns* are utilized. Considering $\boldsymbol{S'}$ as the input state of a decryption round, in the first transformation, *InvShiftRows*, similar to *ShiftRows* in encryption, the first row of the input state remains unchanged. However, the other rows entries are cyclically shifted to the right as follows

$$ISR(\boldsymbol{S'}) = \begin{pmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,3} & s'_{1,0} & s'_{1,1} & s'_{1,2} \\ s'_{2,2} & s'_{2,3} & s'_{2,0} & s'_{2,1} \\ s'_{3,1} & s'_{3,2} & s'_{3,3} & s'_{3,0} \end{pmatrix}. \tag{6.7}$$

The next transformation in each round is *InvSubBytes* implemented by 16 inverse S-boxes. In the inverse S-box, the inverse affine transformation precedes the multiplicative inversion in $GF(2^8)$ to generate $\boldsymbol{s}_{r,c}^{-1} = \boldsymbol{\Gamma}^{-1}\boldsymbol{s}'_{r,c} + \boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma}$, where, $\boldsymbol{\Gamma}$ and $\boldsymbol{\gamma}$ are presented in (6.2). The 8-bit outputs of 16 inverse S-boxes are used to obtain the output state of the InvSubBytes transformation as $\boldsymbol{S} = [s_{r,c}]_{r,c=0}^{3}$.

The next transformation is *AddRoundKey* in which the input state is added with the key of the round. Then, the output state of AddRoundKey is obtained as $\boldsymbol{S''} = [s''_{r,c}]_{r,c=0}^{3} = \boldsymbol{S} + \boldsymbol{K}$. Finally, the last transformation, *InvMixColumns*, is equivalent to multiplying the input state with a constant matrix with hexadecimal entries to obtain the output state of the round as

$$\boldsymbol{O} = [o_{r,c}]_{r,c=0}^{3} = \begin{pmatrix} \{0e\}_h & \{0b\}_h & \{0d\}_h & \{09\}_h \\ \{09\}_h & \{0e\}_h & \{0b\}_h & \{0d\}_h \\ \{0d\}_h & \{09\}_h & \{0e\}_h & \{0b\}_h \\ \{0b\}_h & \{0d\}_h & \{09\}_h & \{0e\}_h \end{pmatrix} \boldsymbol{S''}. \tag{6.8}$$

6.2 A New Fault Detection Scheme for the S-box and the Inverse S-box

In this section, first we present a systematic method for the fault detection of the multiplicative inversion of the S-box and the inverse S-box. Then, the new scheme for the entire S-box and the inverse S-box is presented.

6.2.1 The Systematic Scheme for the Multiplicative Inversion

In what follows, we present a systematic method for the fault detection scheme for the multiplicative inversion by deriving the matrix-based formulations for the multiplicative inversion in the S-box/inverse S-box.

We use the following theorem from [94] to obtain the multiplication of field elements $A = \sum_{i=0}^{m-1} a_i\alpha^i$ and $B = \sum_{i=0}^{m-1} b_i\alpha^i$ in the finite field $GF(2^m)$ constructed by the irreducible polynomial of $P(x)$ with the primitive root of α.

Theorem 6.1 *[94] Let $C = \sum_{i=0}^{m-1} c_i\alpha^i$ be the multiplication of A and $B \in GF(2^m)$. Then, the coordinates of C can be obtained from*

$$[c_0, c_1, \cdots, c_{m-1}]^T = (\boldsymbol{L} + \boldsymbol{Q^T U})\boldsymbol{b}, \tag{6.9}$$

where, $\boldsymbol{b} = [b_0, b_1, \cdots, b_{m-1}]^T$,

$$\boldsymbol{L} = \begin{pmatrix} a_0 & 0 & 0 & 0 & \dots & 0 \\ a_1 & a_0 & 0 & 0 & \dots & 0 \\ a_2 & a_1 & a_0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ a_{m-2} & a_{m-3} & \dots & a_1 & a_0 & 0 \\ a_{m-1} & a_{m-2} & \dots & a_2 & a_1 & a_0 \end{pmatrix}, \tag{6.10}$$

$$\boldsymbol{U} = \begin{pmatrix} 0 & a_{m-1} & a_{m-2} & \dots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & \dots & a_3 & a_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & a_{m-1} & a_{m-2} \\ 0 & 0 & \dots & 0 & 0 & a_{m-1} \end{pmatrix}, \tag{6.11}$$

and the $m-1$ by m binary matrix $\boldsymbol{Q}$ is obtained as follows

$$[\alpha^m, \alpha^{m+1}, \dots, \alpha^{2m-2}]^T = \boldsymbol{Q}[1, \alpha, \alpha^2, \dots, \alpha^{m-1}]^T (\mathrm{mod} P(\alpha)). \tag{6.12}$$

Let $s = s_7\alpha^7 + s_6\alpha^6 + s_5\alpha^5 + s_4\alpha^4 + s_3\alpha^3 + s_2\alpha^2 + s_1\alpha + s_0$ and $s^{-1} = s_7^{-1}\alpha^7 + s_6^{-1}\alpha^6 + s_5^{-1}\alpha^5 + s_4^{-1}\alpha^4 + s_3^{-1}\alpha^3 + s_2^{-1}\alpha^2 + s_1^{-1}\alpha + s_0^{-1}$ be the 8-bit input and output of the multiplicative inversion in the binary field $GF(2^8)$, respectively. Considering the fact that the result of the multiplication of the 8-bit input s, $s \neq 0$, and the output s^{-1} of the multiplicative inversion is the unity polynomial $1 \in GF(2^8)$, the following is derived from Theorem 6.1 for the relation between s and s^{-1}.

Corollary 6.1 *Let* $\boldsymbol{s} = [s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7]^T$ *and* $\boldsymbol{s}^{-1} = [s_0^{-1}, s_1^{-1}, s_2^{-1}, s_3^{-1}, s_4^{-1}, s_5^{-1}, s_6^{-1}, s_7^{-1}]^T$ *be the vectors corresponding to the input and output of the multiplicative inversion. Then, the matrix formulation of the multiplicative inversion of the S-box (resp. the inverse S-box) is as follows*

$$\boldsymbol{Z}\boldsymbol{s}^{-1} = \boldsymbol{u}, \tag{6.13}$$

where,

$$\boldsymbol{Z} = \begin{pmatrix} s_0 & s_7 & s_6 & s_5 & s_4 & s_{7,3} & s_{7,6,2} & s_{6,5,1} \\ s_1 & s_{7,0} & s_{7,6} & s_{6,5} & s_{5,4} & s_{7,4,3} & s_{6,3,2} & s_{7,5,2,1} \\ s_2 & s_1 & s_{7,0} & s_{7,6} & s_{6,5} & s_{5,4} & s_{7,4,3} & s_{6,3,2} \\ s_3 & s_{7,2} & s_{6,1} & s_{7,5,0} & s_{7,6,4} & s_{7,6,5,3} & s_{7,6,5,4,2} & s_{7,6,5,4,3,1} \\ s_4 & s_{7,3} & s_{7,6,2} & s_{6,5,1} & s_{7,5,4,0} & s_{6,4,3} & s_{5,3,2} & s_{7,4,2,1} \\ s_5 & s_4 & s_{7,3} & s_{7,6,2} & s_{6,5,1} & s_{7,5,4,0} & s_{6,4,3} & s_{5,3,2} \\ s_6 & s_5 & s_4 & s_{7,3} & s_{7,6,2} & s_{6,5,1} & s_{7,5,4,0} & s_{6,4,3} \\ s_7 & s_6 & s_5 & s_4 & s_{7,3} & s_{7,6,2} & s_{6,5,1} & s_{7,5,4,0} \end{pmatrix} \tag{6.14}$$

$\boldsymbol{u} = [u, 0, 0, 0, 0, 0, 0, 0]^T$*, and* u *is obtained by logical OR operations of all inputs and outputs, i.e.,* $u = (s_0 \vee s_1 \vee \ldots s_7) \vee (s_0^{-1} \vee s_1^{-1} \vee \ldots s_7^{-1})$*. Moreover, in (6.14), the modulo-2 additions (XOR operations) of the coordinates of* s *are shown with commas in indices, e.g.,* $s_{7,0} = s_7 + s_0$*.*

Proof We prove (6.13) for two cases of $s \neq 0$ and $s = 0$, separately. Let the input s be a non-zero field element in $GF(2^8)$ generated by $P(x) = x^8 + x^4 + x^3 + x + 1$. Then, the multiplicative inversion should generate s^{-1}. Using (6.12) in Theorem 6.1 and considering the irreducible polynomial of $P(x)$, the 7×8 matrix $\boldsymbol{Q}$ can be obtained as

$$\boldsymbol{Q} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{6.15}$$

This matrix is obtained by using the representations of $\alpha^8, \alpha^9, \ldots, \alpha^{14}$ with respect to the polynomial basis for different rows of $\boldsymbol{Q}$. Considering $A = s \neq 0$ and $B = s^{-1}$ in

Theorem 6.1, the matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ in (6.10) and (6.11) are functions of the 8-bit input vector $\boldsymbol{s}$ as

$$\boldsymbol{L} = \begin{pmatrix} s_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ s_1 & s_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ s_2 & s_1 & s_0 & 0 & 0 & 0 & 0 & 0 \\ s_3 & s_2 & s_1 & s_0 & 0 & 0 & 0 & 0 \\ s_4 & s_3 & s_2 & s_1 & s_0 & 0 & 0 & 0 \\ s_5 & s_4 & s_3 & s_2 & s_1 & s_0 & 0 & 0 \\ s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0 & 0 \\ s_7 & s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0 \end{pmatrix}, \tag{6.16}$$

$$\boldsymbol{U} = \begin{pmatrix} 0 & s_7 & s_6 & s_5 & s_4 & s_3 & s_2 & s_1 \\ 0 & 0 & s_7 & s_6 & s_5 & s_4 & s_3 & s_2 \\ 0 & 0 & 0 & s_7 & s_6 & s_5 & s_4 & s_3 \\ 0 & 0 & 0 & 0 & s_7 & s_6 & s_5 & s_4 \\ 0 & 0 & 0 & 0 & 0 & s_7 & s_6 & s_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & s_7 & s_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & s_7 \end{pmatrix}. \tag{6.17}$$

Substituting $\boldsymbol{Q}$, $\boldsymbol{L}$, and $\boldsymbol{U}$ from (6.15)-(6.17) into (6.9) and denoting $\boldsymbol{Z} = \boldsymbol{L} + \boldsymbol{Q}^T\boldsymbol{U}$, one can obtain the matrix $\boldsymbol{Z}$ presented in (6.14). Since $s \neq 0 = (0, 0, ..., 0) \in GF(2^8)$, $u = 1$ and the result of multiplication is $C = A.B \ mod \ P(x) = 1 \in GF(2^8)$, i.e., $[c_0, c_1, ..., c_7]^T = [1, 0, ..., 0]^T$. Therefore, using (6.9), one can prove that (6.13) is valid for $s \neq 0$. Moreover, for $s = 0$, the output of the multiplicative inversion generates $0 = (0, 0, ..., 0)$. Thus, all entries of the matrix $\boldsymbol{Z}$ and hence all 8 entries of the left hand side vector of (6.13) are equal to zero. In such a case, the vector $\boldsymbol{u} = [0, 0, ..., 0]^T$ since the result of the OR operation among all s_is and s_i^{-1}s are zero, i.e., $u = 0$. Therefore, the proof is complete. ∎

The validity of (6.13) can be used to detect specific faults in the inversion block. Let us consider (6.13) for 3 special cases. If both the input and the output are zero, i.e., $s = s^{-1} = 0 \in GF(2^8)$, the output is error-free. Then, both sides of (6.13) are zero and thus it holds which means no fault is detected. On the other hand, the left hand side of (6.13) is zero while in the right hand side, $u = 1$ in the following two cases: (i) the input is zero ($s = 0$) and the erroneous output is not zero, i.e., $s^{-1} \neq 0$, (ii) the input is not zero, i.e., $s \neq 0$, but the erroneous output is zero ($s^{-1} = 0$). Thus, in both cases (6.13) does not hold which indicates that the errors in the output of the multiplicative inversion have been occurred.

One can figure out that implementing (6.13) needs 64 ANDs, 15 ORs, and 143 XOR gates. It is noted that using subexpression sharing, one can reduce the number of XOR gates to 84. If one implements the S-box using the composite field presented in [22], it requires 36 AND gates and 123 XOR gates for the original S-box implementation. Then, adding this fault detection scheme would require approximately 91% area overhead. This

is derived assuming that an XOR gate is implemented by 10 transistors [95] and the silicon area of an AND is 0.6 of that of an XOR gate. Furthermore, the upper bound delay of the multiplication can be derived as $T_M \leq T_A + 5T_X$, where T_A and T_X are the delays for an AND and an XOR gate, respectively [94]. This is the delay overhead after the derivation of the output of the SubBytes transformation. As a result of this high overhead, this scheme may not be applied for the area/delay-constrained applications.

As mentioned above, comparing the actual result of the multiplication of the input and the output of the multiplicative inversion with the predicted one is not area efficient. Therefore, considering our derivations of matrix $\boldsymbol{Z}$ in (6.14), the complexity of the fault detection scheme of the multiplicative inversion can be reduced by deriving the partial result of the multiplication of the input and the output based on the rows that have the lowest overhead. Therefore, one can use this low-complexity signature for the fault detection of the multiplicative inversion.

6.2.2 The Proposed Scheme for the S-box and the Inverse S-box

The scheme in [38] does not take the affine transformation into account and checks it separately with an additional overhead. Furthermore, if one implements SubBytes in the AES using LUTs, there is no access to the output of the multiplicative inversion. Therefore, the above mentioned scheme cannot be used. In what follows, we propose a new scheme which is independent of the way the S-box and the inverse S-box are implemented. First, we obtain the matrix-based S-box formulations as follows:

Theorem 6.2 *Let* $s = s_7\alpha^7 + s_6\alpha^6 + s_5\alpha^5 + s_4\alpha^4 + s_3\alpha^3 + s_2\alpha^2 + s_1\alpha + s_0$ *and* $s' = s'_7\alpha^7 + s'_6\alpha^6 + s'_5\alpha^5 + s'_4\alpha^4 + s'_3\alpha^3 + s'_2\alpha^2 + s'_1\alpha + s'_0$ *be the 8-bit input and output of the S-box. Then, one can obtain the relation between the input and output of the S-box as:*

$$\boldsymbol{M}\boldsymbol{s}' + \boldsymbol{m} = \boldsymbol{u}', \tag{6.18}$$

where, $\boldsymbol{u}' = [u', 0, 0, 0, 0, 0, 0, 0]^T$, $u' = (s_0 \vee s_1 \vee \ldots s_7) \vee (\overline{s'_0} \vee \overline{s'_1} \vee s'_2 \vee s'_3 \vee s'_4 \vee \overline{s'_5} \vee \overline{s'_6} \vee s'_7)$, $\boldsymbol{s}' = [s'_0, s'_1, s'_2, s'_3, s'_4, s'_5, s'_6, s'_7]^T$, *and* $\boldsymbol{m} = [s_{6,0},\ s_{7,6,1},\ s_{7,2,0},\ s_{6,3,1},\ s_{7,6,4,2},\ s_{7,5,3},\ s_{6,4},\ s_{7,5}]^T$.

Furthermore, the 8×8 *matrix* $\boldsymbol{M}$ *is denoted as follows*

$$\boldsymbol{M} = \begin{pmatrix} s_{6,5,2} & s_{5,4,1} & s_{7,5,3,0} & s_{6,4,2} & s_{7,5,3,1} & s_{7,6,5,2,0} & s_{7,6,5,4,1} & s_{7,6,3,0} \\ s_{7,5,3,2,0} & s_{6,4,2,1} & s_{7,6,5,4,3,1} & s_{7,6,5,4,3,2,0} & s_{7,6,5,4,3,2,1} & s_{5,3,2,1} & s_{4,2,1,0} & s_{6,4,3,1} \\ s_{6,4,3,1} & s_{7,5,3,2,0} & s_{7,6,5,4,2} & s_{7,6,5,4,3,1} & s_{7,6,5,4,3,2,0} & s_{6,4,3,2} & s_{5,3,2,1} & s_{7,5,4,2,0} \\ s_{7,6,4,0} & s_{6,5,3} & s_{6,0} & s_{7,5} & s_{6,4} & s_{6,4,3,2,0} & s_{7,5,3,2,1} & s_{7,5,1} \\ s_{7,6,2,1} & s_{7,6,5,1,0} & s_{5,3,1} & s_{4,2,0} & s_{3,1} & s_{6,4,3,2,1} & s_{7,5,3,2,1,0} & s_{7,3,2} \\ s_{7,3,2} & s_{7,6,2,1} & s_{6,4,2,0} & s_{5,3,1} & s_{4,2,0} & s_{7,5,4,3,2} & s_{6,4,3,2,1} & s_{4,3,0} \\ s_{4,3,0} & s_{7,3,2} & s_{7,5,3,1} & s_{6,4,2,0} & s_{5,3,1} & s_{6,5,4,3,0} & s_{7,5,4,3,2} & s_{5,4,1} \\ s_{5,4,1} & s_{4,3,0} & s_{6,4,2} & s_{7,5,3,1} & s_{6,4,2,0} & s_{7,6,5,4,1} & s_{6,5,4,3,0} & s_{6,5,2} \end{pmatrix}. \quad (6.19)$$

Proof We prove (6.18) for two cases of $s \neq 0$ and $s = 0$, separately. Let the 8-bit input s be a non-zero field element in $GF(2^8)$. Considering (6.2), one can obtain

$$\boldsymbol{s}^{-1} = \boldsymbol{\Gamma}^{-1}\boldsymbol{s}' + \boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \boldsymbol{s}' + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \quad (6.20)$$

By substituting $\boldsymbol{s}^{-1}$ from (6.20) in (6.13), one reaches $\boldsymbol{Z}\boldsymbol{\Gamma}^{-1}\boldsymbol{s}' + \boldsymbol{Z}\boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma}$ which is the same as the left hand side of (6.18). Now, let us denote $\boldsymbol{Z}\boldsymbol{\Gamma}^{-1} = \boldsymbol{M}$ and $\boldsymbol{Z}\boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma} = \boldsymbol{m}$. Then, the left hand side of (6.18) is obtained. Since $s \neq 0 = (0, 0, ..., 0) \in GF(2^8)$, $u' = 1$. Moreover, according to the proof of Corollary 6.1, for $s \neq 0$, the left hand side of (6.13) is $[1, 0, ..., 0]^T$, i.e., the result of multiplication $C = A.B \ mod \ P(x) = 1 \in GF(2^8)$. This implies that the left hand side of (6.13) be $\boldsymbol{Z}\boldsymbol{s}^{-1} = [1, 0, ..., 0]^T = \boldsymbol{u'}$. Furthermore, because we have $\boldsymbol{Z}\boldsymbol{s}^{-1} = \boldsymbol{M}\boldsymbol{s}' + \boldsymbol{m}$, one can prove that (6.18) is valid for $s \neq 0$. Moreover, according to (6.2), for the input $s = 0 = (0, 0, ..., 0) \in GF(2^8)$, we have the output as $\boldsymbol{s'} = [s'_0, s'_1, ..., s'_7]^T = [1, 1, 0, 0, 0, 1, 1, 0]^T$ which corresponds to the field element $s' = \{63\}_h = (0, 1, 1, 0, 0, 0, 1, 1) \in GF(2^8)$. Therefore, as seen in Theorem 6.2, $\boldsymbol{u'} = [0, 0, ..., 0]^T$ since we have $u' = (s_0 \vee s_1 \vee \ldots s_7) \vee (\overline{s'_0} \vee \overline{s'_1} \vee s'_2 \vee s'_3 \vee s'_4 \vee \overline{s'_5} \vee \overline{s'_6} \vee s'_7) = 0$. In addition, for $s = 0$, all the entries of the matrix $\boldsymbol{M}$ and the vector $\boldsymbol{m}$ in the left hand side of (6.18) are equal to zero. This results in the vector $[0, 0, ..., 0]^T = \boldsymbol{u'}$ for the left hand side of (6.18). Therefore, the proof is complete. ∎

Let us consider (6.18) for the input $s = 0 = (0, 0, ..., 0) \in GF(2^8)$. For this input, the correct output is $s' = \{63\}_h = (0, 1, 1, 0, 0, 0, 1, 1) \in GF(2^8)$ (see (6.2)). If the erroneous output is not $s' = \{63\}_h = (0, 1, 1, 0, 0, 0, 1, 1) \in GF(2^8)$, in the right hand side of (6.18) we have $u' = 1$, whereas, the left hand side is zero. Therefore, the erroneous output is detected.

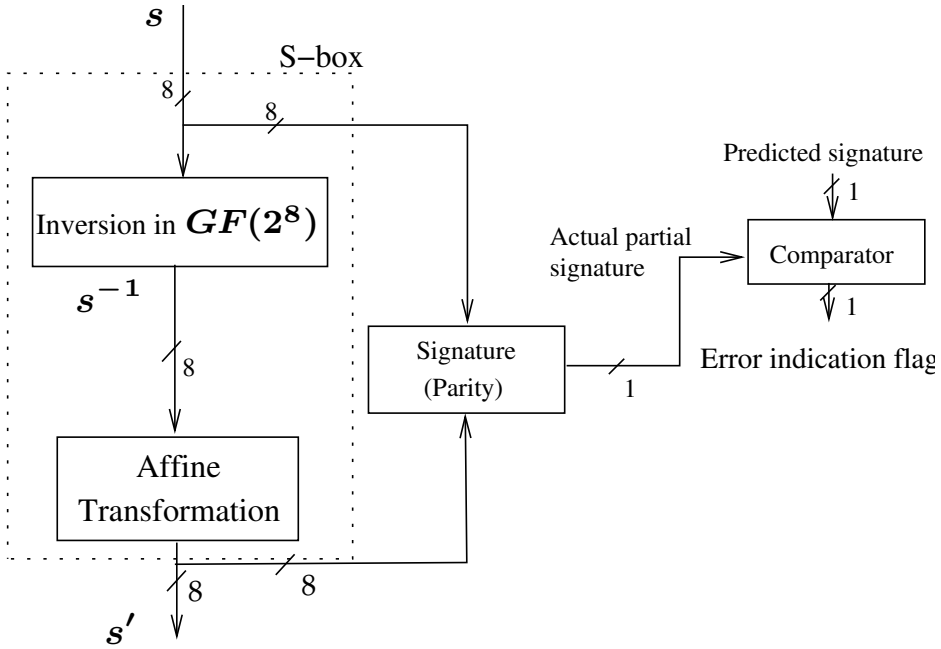


Figure 6.1: The proposed structure-independent fault detection scheme of the S-box.

Proposition 6.1 *Using subexpression sharing, the implementation of the left hand side of (6.18) needs* 64 *AND gates and* 111 *XOR gates. Furthermore, the upper bound delay of the relation in the left hand side of (6.18) is* $T_A + 6T_X$*, where,* T_A *and* T_X *are the delays for an AND and an XOR gate, respectively.*

Although checking the formulation of (6.18) detects all errors in the output of the S-box, its implementation is very costly (see Proposition 6.1). To reduce the overhead of the fault detection scheme, as seen in Fig. 6.1, we have obtained the single-bit parity for the formulation of (6.18). As shown in this figure, this is obtained in order to compare only one bit for an 8-bit data to detect any combination of odd number of erroneous bits at the result of the left hand side of (6.18). Thus, one can check the parity of two sides of (6.18) to obtain one bit equation for checking the S-box as follows:

Theorem 6.3 *Let* $s = s_7\alpha^7 + s_6\alpha^6 + s_5\alpha^5 + s_4\alpha^4 + s_3\alpha^3 + s_2\alpha^2 + s_1\alpha + s_0 \in GF(2^8)$*, and* $s' = s'_7\alpha^7 + s'_6\alpha^6 + s'_5\alpha^5 + s'_4\alpha^4 + s'_3\alpha^3 + s'_2\alpha^2 + s'_1\alpha + s'_0 \in GF(2^8)$ *be the 8-bit input and output of the S-box. Then, the following equation holds for all the possible patterns*

of s and s'.

$$P_{(\boldsymbol{M}\boldsymbol{s}'+\boldsymbol{m})} = s_0(s'_b + s'_c) + s_1 s'_b + s_2 s'_d + s_3 s'_4 + s_4(s'_c + s'_3) + s_5 s'_a + s_6(s'_d + \overline{s'_6}) + s_7(\overline{s'_5 + s'_4}) = u', \tag{6.21}$$

where, $s'_a = s'_0 + s'_2 + s'_3 + s'_5$, $s'_b = s'_a + s'_7$, $s'_c = s'_1 + s'_4 + s'_6$, and $s'_d = s'_2 + s'_7$.

Proof After obtaining the parity of two sides of (6.18) we have

$$P_{(\boldsymbol{M}\boldsymbol{s}'+\boldsymbol{m})} = P_{\boldsymbol{u}'} = u', \tag{6.22}$$

where, $\boldsymbol{M}$, $\boldsymbol{m}$ and $\boldsymbol{u}'$ are presented in Theorem 6.2. Considering the fact that parity is a linear operation, one can obtain the left hand side of (6.22) as $P_{(\boldsymbol{M}\boldsymbol{s}'+\boldsymbol{m})} = P_{\boldsymbol{M}\boldsymbol{s}'} + P_{\boldsymbol{m}}$. Then, using $\boldsymbol{M}$ and $\boldsymbol{m}$ defined in Theorem 6.2, one can obtain $P_{\boldsymbol{M}\boldsymbol{s}'} = s'_0 s_a + s'_1 s_b + s'_2 s_c + s'_3(s_a + s_4) + s'_4(s_b + s_3 + s_7) + s'_5(s_a + s_7) + s'_6(s_b + s_6) + s'_7(s_5 + s_c)$ and $P_{\boldsymbol{m}} = s_6 + s_7$, where, $s_a = s_0 + s_1 + s_5$, $s_b = s_0 + s_4$, $s_c = s_a + s_2 + s_6$. After rearranging, one reaches (6.21) and the proof is complete. ∎

To implement (6.21), 18 XOR gates and 8 AND gates and two NOT gates are needed. Also, the delay overhead associated with this implementation is the delay of 6 XORs and one AND after the completion of the S-box. It is noted that this delay can be overlapped by other AES round transformations and hence it will not reduce the speed of the entire fault detection AES implementation. More details on this will be presented later in this chapter. The parity obtained by the parity circuit is then compared with u' (see Theorems 6.2 and 6.3) to obtain the error indication flag of each S-box, i.e., $e_{r,c}$, $0 \leq r, c \leq 3$. It is noted that using an OR tree for the error indication flags of 16 S-boxes, the final error indication flag of the entire SubBytes transformation is obtained. The final error indication flag of the SubBytes transformation signals the errors if at least one of the error indication flags of 16 S-boxes detect errors.

Now, we want to present the fault detection scheme for the inverse S-box in the AES decryption. The inverse S-box of the decryption consists of the inverse affine transformation (the inverse of the affine transformation in (6.2)) followed by the multiplicative inversion. In other words, one can obtain the inverse S-box by removing the affine transformation and adding the inverse affine one. This uses the input of s' and the output of s^{-1} with the following multiplicative inversion having the input of s^{-1} and the output of

s. Therefore, Theorems 6.2 and 6.3 are also valid for the inverse S-box and hence we can conclude the following for the inverse S-box.

Corollary 6.2 *For the fault detection of the inverse S-box, one can use (6.21) by changing the place of the input and output, i.e., swapping the coordinates of s with s'.*

6.3 Proposed Fault Detection Schemes for the AES

As mentioned before, the parity-based scheme proposed in [35] is one of the first fault detection schemes and has received attention in the literature. Although the approach in [35] is a good scheme in terms of the fault detection capability, it has two drawbacks. First, this approach is based on using the expanded S-boxes and inverse S-boxes for parity predictions, i.e., two blocks of 256×9 memory cells. Not only does this restrict the AES encryption and decryption implementations to LUT-based S-boxes and inverse S-boxes, but it has high area overhead. To counteract this drawback, one may use the proposed fault detection scheme for the S-box or the inverse S-box. As an example, for the AES encryption one may use (6.21) for the S-boxes. This results in obtaining the output parity of each S-box concurrently without having an extra circuit for deriving it, i.e., $P_{s'} = \sum_{i=0}^{7} s'_i = s'_b + s'_c$ in (6.21). This simplifies the fault detection circuit of the AES when the output parities of the S-boxes are utilized for the fault detection of other transformations in the AES rounds in [35]. More specifically, if one uses the scheme presented in [35] for the fault detection of the MixColumns transformation, the predicted parities of this transformation become functions of the output parities of the ShiftRows (SubBytes) transformation. Using the proposed scheme for the S-box in this chapter, one can easily utilize the output parities of the S-boxes to predict the parities of the MixColumns transformation.

The second drawback of the approach in [35] is the relatively high area complexity of the parity predictions of MixColumns in the AES encryption. For the AES decryption, the area complexity of the predicted parities of InvMixColumns is even more [36]. The implementation results presented later in this chapter show the high area overhead of this scheme. Considering the fact that a low-cost fault detection scheme for the AES encryption and decryption is preferred, in this section, we propose signature-based low complexity fault detection schemes for the transformations in the AES encryption and

decryption. We consider AES-128 (which is denoted as AES in the remaining of this chapter) for the sake of brevity. It is noted that the proposed schemes can be also applied to AES-192 and AES-256. The proposed schemes for the AES transformations are based on deriving the low-cost output signatures of the transformations in the AES rounds and comparing them with their actual signatures for reaching the error indication flags.

6.3.1 AES Encryption

We present the new fault detection structure for the AES encryption in the following. A typical AES encryption round (except for the last round) consists of four transformations, the fault detection schemes are shown in Fig. 6.2 and presented in details below.

SubBytes and ShiftRows

In the AES encryption, the SubBytes transformation consists of 16 S-boxes (see (6.3)). Let $e_{r,c}$, $0 \leq r, c \leq 3$, be the error indication flag for the S-box with the input and the output of $s_{r,c}$ and $s'_{r,c}$, respectively. The output state of such flags can be re-written as 16 formulations as follows

$$e_{r,c} = P_{(\boldsymbol{M}_{r,c}\boldsymbol{s'}_{r,c}+\boldsymbol{m}_{r,c})} + u'_{r,c}, \quad 0 \leq r, c \leq 3, \tag{6.23}$$

where, $u'_{r,c}$ is defined in Theorem 6.2 and for a typical S-box, $P_{(\boldsymbol{M}_{r,c}\boldsymbol{s'}_{r,c}+\boldsymbol{m}_{r,c})}$ is presented in (6.21).

The 128-bit output of the SubBytes transformation acts as the input to ShiftRows. As seen in (6.4), the output state of ShiftRows is obtained by shifting the state entries in (6.3). Therefore, by considering the corresponding output of ShiftRows in (6.4), one can check two transformations of SubBytes and ShiftRows together using 16 error indication flags. According to (6.4) and considering (6.23), for row r and column c, the output state of the flags can be re-written as 16 formulations as follows

$$e_{r,c} = P_{(\boldsymbol{M}_{r,c^*}\boldsymbol{s'}_{r,c^*}+\boldsymbol{m}_{r,c^*})} + u'_{r,c^*}, \quad 0 \leq r, c \leq 3, \tag{6.24}$$

where, $c^* = (r + c)\ mod\ 4$.

According to (6.24), 16 error indication flags for the SubBytes and ShiftRows transformations, i.e., one error indication flag for each byte, are obtained. This is shown in

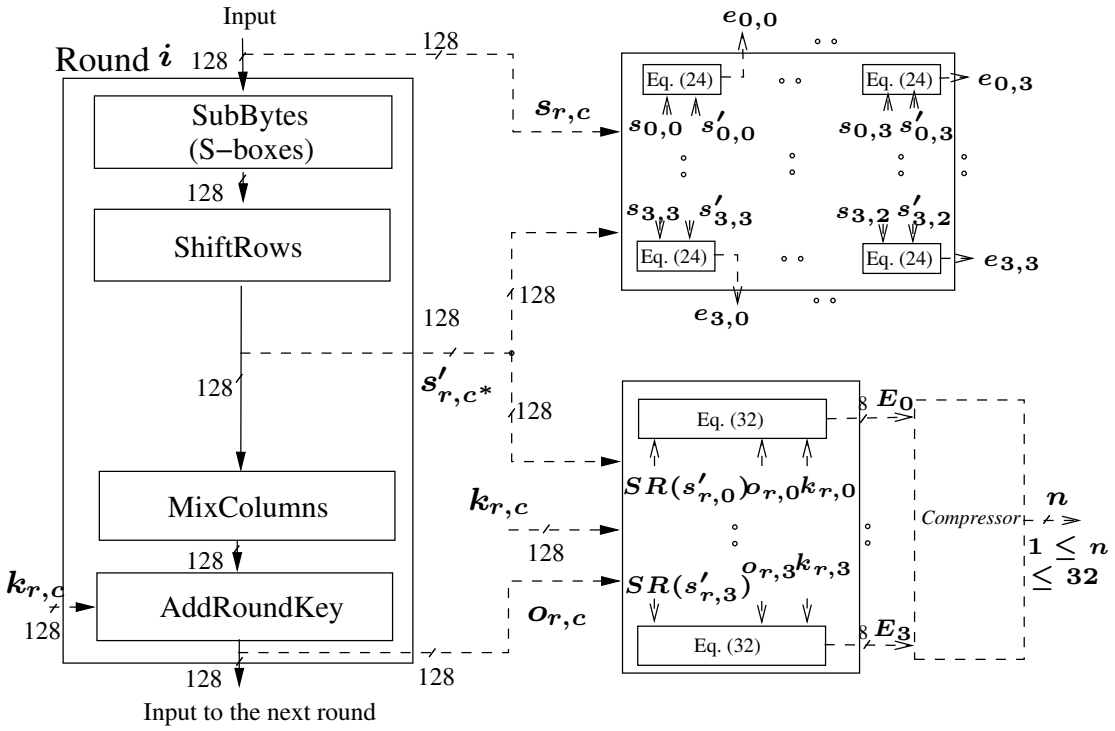


Figure 6.2: The proposed fault detection scheme for the *ith* round of the AES encryption.

Fig. 6.2. As seen in this figure, (6.24), i.e., instances of the hardware implementation of (6.21), is utilized for obtaining 16 error indication flags.

MixColumns and AddRoundKey

The third and the fourth transformations in a typical AES encryption round are MixColumns and AddRoundKey. It is noted that MixColumns is constructed using (6.5). Furthermore, according to (6.6), AddRoundKey is the modulo-2 addition of the input state with the roundkey. In what follows, we present a key formulation that is used for deriving a low-complexity fault detection scheme for MixColumns and AddRoundKey combined.

Theorem 6.4 *Let* $SR(\boldsymbol{S}') = [s'_{r,c^*}]_{r,c=0}^{3}$ *and* $\boldsymbol{K} = [k_{r,c}]_{r,c=0}^{3}$ *be the input and the round-key input of MixColumns and AddRoundKey in round i, respectively. Let the output of AddRoundKey be* $\boldsymbol{O} = [o_{r,c}]_{r,c=0}^{3}$ *(see (6.6)). Then, the following holds:*

$$\sum_{r=0}^{3}(s'_{r,c^*} + k_{r,c} + o_{r,c}) = 0 \in GF(2^8), \quad 0 \leq c \leq 3, \tag{6.25}$$

where, $c^* = (r + c) \, mod \, 4$, and each summation is over $GF(2^8)$ which consists of eight modulo-2 additions.

Proof After adding the columns of $\boldsymbol{S}''$ in (6.5), one reaches the following:

$$s''_{0,0} + s''_{1,0} + s''_{2,0} + s''_{3,0} = \tag{6.26}$$
$$(\{2\}_{16} + \{1\}_{16} + \{1\}_{16} + \{3\}_{16})(s'_{0,0} + s'_{1,1} + s'_{2,2} + s'_{3,3}),$$

$$s''_{0,1} + s''_{1,1} + s''_{2,1} + s''_{3,1} = \tag{6.27}$$
$$(\{2\}_{16} + \{1\}_{16} + \{1\}_{16} + \{3\}_{16})(s'_{0,1} + s'_{1,2} + s'_{2,3} + s'_{3,0}),$$

$$s''_{0,2} + s''_{1,2} + s''_{2,2} + s''_{3,2} = \tag{6.28}$$
$$(\{2\}_{16} + \{1\}_{16} + \{1\}_{16} + \{3\}_{16})(s'_{0,2} + s'_{1,3} + s'_{2,0} + s'_{3,1}),$$

$$s''_{0,3} + s''_{1,3} + s''_{2,3} + s''_{3,3} = \tag{6.29}$$
$$(\{2\}_{16} + \{1\}_{16} + \{1\}_{16} + \{3\}_{16})(s'_{0,3} + s'_{1,0} + s'_{2,1} + s'_{3,2}).$$

Considering the fact that $\{3\}_{16} = \{1\}_{16} + \{2\}_{16}$, we have $(\{2\}_{16} + \{1\}_{16} + \{1\}_{16} + \{3\}_{16}) = \{1\}_{16}$. Moreover, the right hand sides of (6.26)-(6.29) are the additions of the columns of matrix $SR(\boldsymbol{S}')$ in (6.4). Therefore, the addition of the column elements of $\boldsymbol{S}''$ is equal to that of the corresponding column of $SR(\boldsymbol{S}')$, i.e., $\sum_{r=0}^{3} s''_{r,c} = \sum_{r=0}^{3} s'_{r,c^*}$, $0 \leq c \leq 3$. Furthermore, according to (6.6), we have

$$\sum_{r=0}^{3} o_{r,c} = \sum_{r=0}^{3} s''_{r,c} + \sum_{r=0}^{3} k_{r,c}, \quad 0 \leq c \leq 3. \tag{6.30}$$

Therefore, considering (6.30) we reach

$$\sum_{r=0}^{3} o_{r,c} = \sum_{r=0}^{3} s'_{r,c^*} + \sum_{r=0}^{3} k_{r,c}, \quad 0 \leq c \leq 3. \tag{6.31}$$

Considering (6.31), we have $\sum_{r=0}^{3}(s'_{r,c^*} + k_{r,c} + o_{r,c}) = (0, 0, ..., 0) \in GF(2^8)$ and the proof is complete. ∎

Now, let us introduce the four 8-bit error indication flags for four columns of the state as

$$E_c = \sum_{r=0}^{3}(s'_{r,c^*} + k_{r,c} + o_{r,c}), \quad 0 \leq c \leq 3. \tag{6.32}$$

One can use Theorem 6.4 to verify that for the error-free situation, all 32 bits of such flags in (6.32) are zero, i.e., $E_c = 0 = (0, 0, ..., 0) \in GF(2^8)$, $0 \leq c \leq 3$. These 32 error indication flags can be used for the MixColumns and AddRoundKey transformations combined, i.e., 8 error indication flags for each column of the state matrix. This is shown in Fig. 6.2. It is noted that in Fig. 6.2, $[k_{r,c}]_{r,c=0}^{3}$ is the round i key. As seen in this figure, using (6.32), 32 error indication flags are obtained. It is noted that these error indication flags can be compressed so that n, $1 \leq n \leq 32$, error indication flags for these two transformations are achieved. This can be performed by ORing different combinations of the 32 error indication flags obtained in (6.32) as denoted by the compressor block in Fig. 6.2. This gives us the freedom in the number of the error indication flags used in the fault detection scheme of the MixColumns and AddRoundKey transformations. It is interesting to note that although up to 32 flags can be used, our simulations show that using 16 error indication flags (the same number as the flags derived for SubBytes and ShiftRows), greater than 99% of the errors are covered.

The last round of the AES encryption (round 10 in AES-128 encryption) consists of three transformations, i.e., SubBytes, ShiftRows and AddRoundKey. In other words, compared to the other encryption rounds, the MixColumns transformation has been removed. We present the following for the fault detection of this round.

Remark Similar to the fault detection scheme for the other rounds of the AES encryption, one can use (6.24) for the last encryption round to derive 16 error indication flags for SubBytes and ShiftRows combined. Furthermore, one can use (6.31) for the relation of the inputs and the output of AddRoundKey (see also Fig. 6.2 by removing MixColumns). Therefore, (6.32) can also be used for the last round. Consequently, by removing the MixColumns transformation, one can also utilize the fault detection scheme in Fig. 6.2 for the last encryption round of the AES.

Further Improvements

The proposed fault detection scheme for a typical round of the AES encryption can be modified so that the complexity of the scheme is reduced. This improvement is based on the fact that using subexpression sharing, one can reduce the number of logic gates utilized in obtaining two sets of the error indication flags shown in Fig. 6.2. Specifically, in this chapter, we propose a fault detection scheme for the MixColumns transformation

which has 25% less area overhead than the scheme presented in [35] and [36].

As seen in Fig. 6.2, the error indication flags of SubBytes and ShiftRows are obtained utilizing the output state of ShiftRows, i.e., $SR(\boldsymbol{S}')$ in (6.4). Furthermore, as shown in this figure, this state is also used in obtaining the error indication flags of MixColumns and AddRoundKey. This leads us to perform subexpression sharing in deriving these two sets of error indication flags to have low-complexity fault detection scheme of the AES encryption. We use (6.32) to derive 16 low-complexity signatures for the MixColumns and AddRoundKey transformations, i.e., 4 signatures for each column of the state matrix. This is performed by modulo-2 addition of two sets of four coordinates of (6.32) for each column, i.e., $E_c = (e_{c,7}, e_{c,6}, ..., e_{c,0}) \in GF(2^8)$, $0 \leq c \leq 3$. Let $\hat{E}_c = (e_{c,4}, e_{c,2}, e_{c,1}, e_{c,0})$ and $\check{E}_c = (e_{c,5}, e_{c,7}, e_{c,6}, e_{c,3})$. Then, the four error indication flags for column c of the state are

$$\overline{E}_c = \hat{E}_c + \check{E}_c, \quad 0 \leq c \leq 3. \tag{6.33}$$

One can utilize four sets of modulo-2 additions of the output bits of each S-box pre-computed in (6.21), i.e., $s'_4 + s'_5$, $s'_2 + s'_7$, $s'_1 + s'_6$, and $s'_0 + s'_3$, to obtain the low-complexity error indication flags in (6.33). This is shown in Fig. 6.3. As seen in this figure, the *Common Subexpressions* (CS) unit has been utilized to obtain 64 common subexpressions, i.e., 4 for each of the 16 S-boxes in the SubBytes transformation. As depicted in Fig. 6.3, these outputs are then used in obtaining the two sets of 16 error indication flags for SubBytes and ShiftRows combined, i.e., $e_{r,c}$, $0 \leq r, c \leq 3$, and for MixColumns and AddRoundKey combined, i.e., $\overline{E}_c$, $0 \leq c \leq 3$, respectively. In Fig. 6.3, realizing (6.24) is less complex than the one in Fig. 6.2. This is because (6.24) utilizes the hardware implementation of (6.21) which is less complex when the common subexpressions are used. It is noted that if any of the derived two sets of error indication flags are one, the error is detected. Whereas, if all of them are zero, no error has been detected although the output can be erroneous or correct.

One can compare the complexity of the proposed fault detection scheme for MixColumns with that of [35] and [36]. For comparison, we consider the error indication flags of this transformation separately, i.e., without considering AddRoundKey. In the fault detection scheme of MixColumns, we only need 3 XOR gates for each signature, i.e., modulo-2 adding of the 4 common subexpressions presented above, e.g., $s'_4 + s'_5$, in four rows. Therefore, we have the following remark:

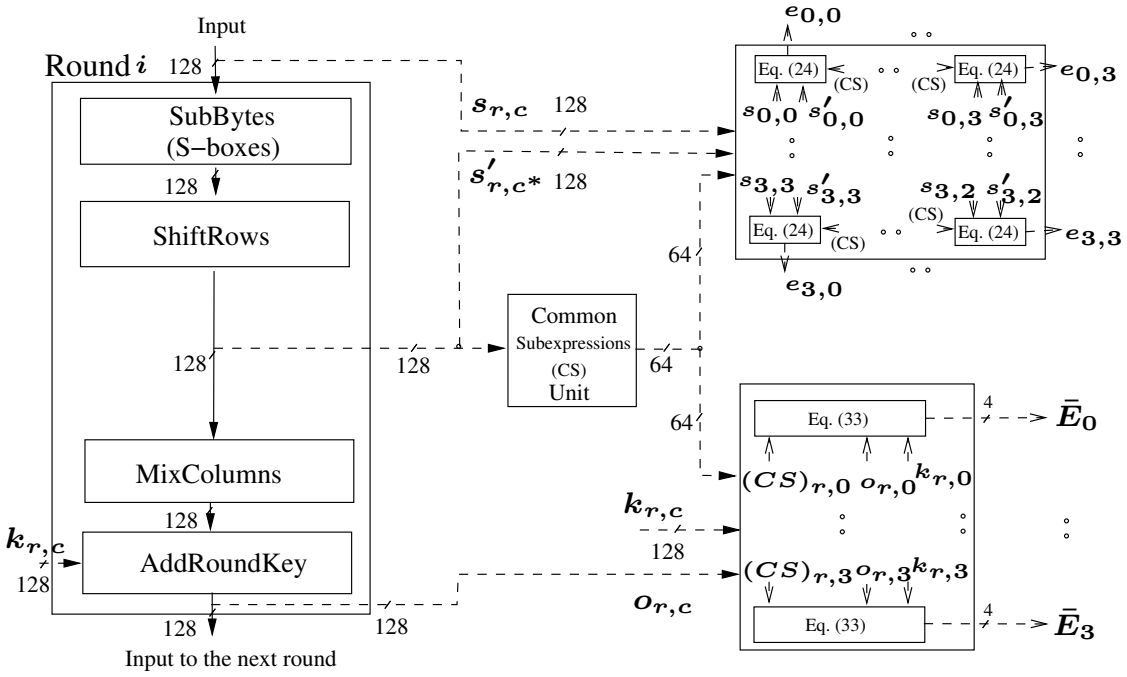


Figure 6.3: The proposed low-complexity fault detection scheme for the *ith* round of the AES encryption utilizing subexpression sharing.

Remark For having 16 signatures for the MixColumns transformation, 48 XOR gates are needed. Comparing this with the parity-based scheme presented in [35] and [36] which needs 64 XOR gates for the predicted parities, this is a 25% area overhead reduction. Moreover, there are two XORs in the critical path delay of the proposed scheme for MixColumns compared to 3 XORs for the scheme in [35] and [36] which is a 33% reduction in the critical path delay.

6.3.2 AES Decryption

We present the fault detection scheme for the AES decryption in what follows. It is noted that the AES decryption rounds (except for the last round) consist of four transformations, i.e., InvShiftRows, InvSubBytes, AddRoundKey and InvMixColumns. The fault detection schemes of these transformations are presented in details in the following.

InvShiftRows and InvSubBytes

As seen in (6.7), in the AES decryption, the 128-bit input to InvShiftRows, i.e., the state matrix $\boldsymbol{S}'$ entries, are cyclically shifted to the right with the first row remained unchanged. Therefore, this transformation is just a re-wiring in hardware.

The output state of the InvShiftRows transformation, i.e., $ISR(\boldsymbol{S}')$ in (6.7), acts as the input to InvSubBytes. The InvSubBytes transformation in the AES decryption consists of 16 inverse S-boxes. One can use Corollary 6.2 for the fault detection scheme of the inverse S-boxes. Then, the fault detection scheme for InvShiftRows and InvSubBytes combined can be derived so that we are able to check these two transformations together. Let $e_{r,c}$, $0 \leq r, c \leq 3$, be the error indication flag of each byte of these two transformations combined with the input and the output of $s'_{r,c}$ and $s_{r,c}$, respectively. Then, according to (6.18), the output state of such flags can be re-written as 16 formulations as follows

$$e_{r,c} = P_{(\boldsymbol{M}_{r,c}\boldsymbol{s'}_{r,c^{**}}+\boldsymbol{m}_{r,c})} + u'_{r,c}, \quad 0 \leq r, c \leq 3, \tag{6.34}$$

where, $c^{**} = |r - c|$.

According to (6.34), 16 error indication flags for the InvShiftRows and InvSubBytes transformations, i.e., one error indication flag for each byte, are obtained. This is shown in Fig. 6.4. As seen in this figure, (6.34), i.e., instances of the hardware implementation of (6.21), is utilized for obtaining these 16 error indication flags.

AddRoundKey and InvMixColumns

As seen in Fig. 6.4, the third and the forth transformations in a typical AES decryption round are AddRoundKey and InvMixColumns. In the AddRoundKey transformation, the input state, i.e., $\boldsymbol{S}$, is added with the roundkey input state, i.e., $\boldsymbol{K}$. Furthermore, the InvMixColumns transformation is equivalent to multiplying the input state with the constant matrix in (6.8). In what follows, we present a key formulation used for deriving a low-complexity fault detection scheme for these two transformations combined.

Theorem 6.5 *Let $\boldsymbol{K} = [k_{r,c}]_{r,c=0}^{3}$ and $\boldsymbol{S} = [s_{r,c}]_{r,c=0}^{3}$ be the roundkey input and the input of AddRoundKey in round i, respectively. Let the output of InvMixColumns be $\boldsymbol{O} = [o_{r,c}]_{r,c=0}^{3}$ (see (6.8)). Then, the following holds:*

$$\sum_{r=0}^{3}(s_{r,c} + k_{r,c} + o_{r,c}) = 0 \in GF(2^8), \quad 0 \leq c \leq 3, \tag{6.35}$$

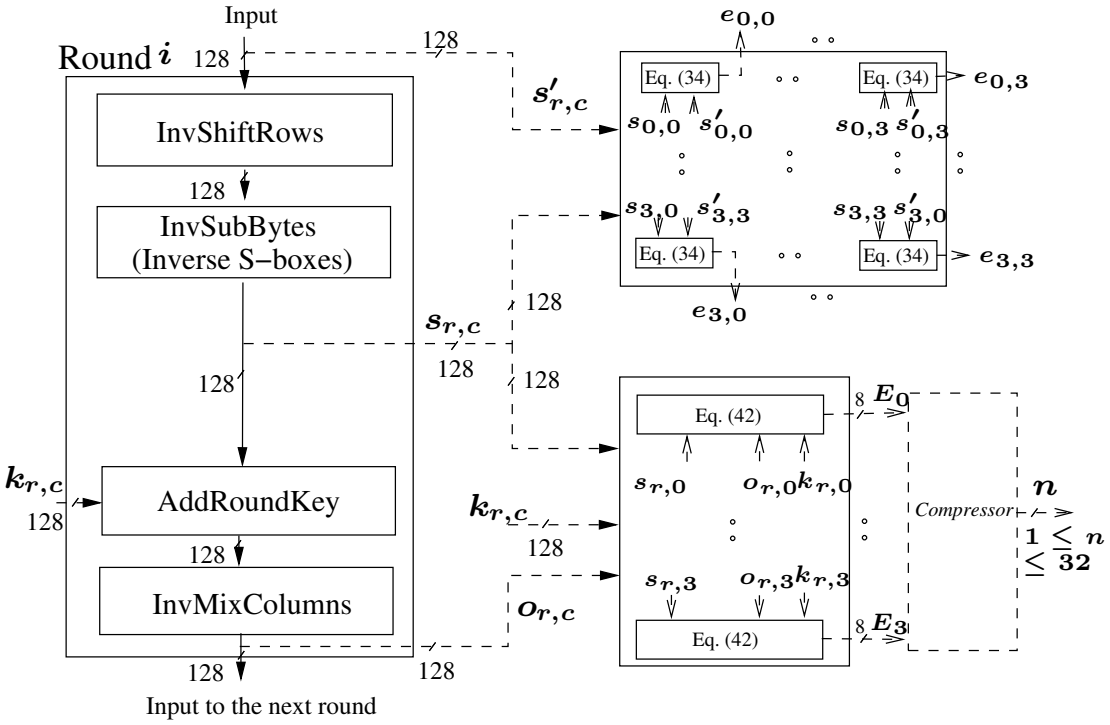


Figure 6.4: The proposed fault detection scheme for the *ith* round of the AES decryption.

where, each summation is over $GF(2^8)$ which consists of eight modulo-2 additions.

Proof After adding the columns of $\boldsymbol{O}$, according to (6.8) one reaches

$$o_{0,0} + o_{1,0} + o_{2,0} + o_{3,0} = \tag{6.36}$$
$$(\{e\}_{16} + \{9\}_{16} + \{d\}_{16} + \{b\}_{16})(s''_{0,0} + s''_{1,0} + s''_{2,0} + s''_{3,0}),$$
$$o_{0,1} + o_{1,1} + o_{2,1} + o_{3,1} = \tag{6.37}$$
$$(\{e\}_{16} + \{9\}_{16} + \{d\}_{16} + \{b\}_{16})(s''_{0,1} + s''_{1,1} + s''_{2,1} + s''_{3,1}),$$

$$o_{0,2} + o_{1,2} + o_{2,2} + o_{3,2} = \tag{6.38}$$
$$(\{e\}_{16} + \{9\}_{16} + \{d\}_{16} + \{b\}_{16})(s''_{0,2} + s''_{1,2} + s''_{2,2} + s''_{3,2}),$$
$$o_{0,3} + o_{1,3} + o_{2,3} + o_{3,3} = \tag{6.39}$$
$$(\{e\}_{16} + \{9\}_{16} + \{d\}_{16} + \{b\}_{16})(s''_{0,3} + s''_{1,3} + s''_{2,3} + s''_{3,3}).$$

We have $\{e\}_{16} + \{9\}_{16} + \{d\}_{16} + \{b\}_{16} = \{1\}_{16}$. Noting that the right hand sides of (6.36)-(6.39) are the additions of the columns of the output state of InvMixColumns,

the addition of the column elements of $\boldsymbol{S''}$ is equal to that of the corresponding column of $\boldsymbol{O}$, i.e., $\sum_{r=0}^{3} {s''}_{r,c} = \sum_{r=0}^{3} o_{r,c}$, $0 \leq c \leq 3$. Furthermore, for the AddRoundKey transformation we have

$$\sum_{r=0}^{3} {s''}_{r,c} = \sum_{r=0}^{3} s_{r,c} + \sum_{r=0}^{3} k_{r,c}, \quad 0 \leq c \leq 3. \tag{6.40}$$

Therefore, according to (6.40) we reach

$$\sum_{r=0}^{3} o_{r,c} = \sum_{r=0}^{3} s_{r,c} + \sum_{r=0}^{3} k_{r,c}, \quad 0 \leq c \leq 3. \tag{6.41}$$

Considering (6.41), one can obtain $\sum_{r=0}^{3}(s_{r,c} + k_{r,c} + o_{r,c}) = (0, 0, ..., 0) \in GF(2^8)$ and the proof is complete. ∎

Similar to the AES encryption, for the AES decryption, we introduce the four 8-bit error indication flags for four columns of the state as

$$E_c = \sum_{r=0}^{3}(s_{r,c} + k_{r,c} + o_{r,c}), \quad 0 \leq c \leq 3. \tag{6.42}$$

These 32 error indication flags for four columns of the state can be utilized for the fault detection of the AddRoundKey and InvMixColumns transformations combined. This is shown in Fig. 6.4. It is noted that like the AES encryption, these error indication flags can be compressed so that n, $1 \leq n \leq 32$, error indication flags for these two transformations are achieved. This gives us the freedom in the number of the error indication flags used in the fault detection scheme of the AddRoundKey and InvMixColumns transformations. It is interesting to note that our simulations for the AES decryption show that using 16 error indication flags more than 99% of the errors are covered.

Similar to the AES encryption, in the last round of the AES decryption, three transformations are used, i.e., InvMixColumns is removed. We present the following for the fault detection of this round.

Remark Similar to the fault detection scheme for the other rounds of the AES decryption, one can use (6.34) for the last decryption round to derive 16 error indication flags for InvShiftRows and InvSubBytes combined. Furthermore, one can use (6.41) for the relation of the inputs and the output of AddRoundKey (see also Fig. 6.4 by removing InvMixColumns). Therefore, (6.42) can also be used for the last round. Consequently, by removing the InvMixColumns transformation, one can also utilize the fault detection scheme in Fig. 6.4 for the last decryption round of the AES.

Further Improvements

Using subexpression sharing, the proposed fault detection scheme for a typical AES decryption round can be modified so that its hardware complexity is reduced. As seen in Fig. 6.4, the error indication flags of InvShiftRows and InvSubBytes are obtained utilizing the output state of InvSubBytes, i.e., $\boldsymbol{S}$. As shown in Fig. 6.4, this output state is also used in obtaining the error indication flags of AddRoundKey and InvMixColumns. Therefore, similar to the fault detection scheme for the AES encryption, we can perform subexpression sharing to obtain these two sets of error indication flags to have low-complexity fault detection scheme of the AES decryption. First, we present the following for the inverse S-boxes by rearranging Corollary 6.2 so that we are able to present a low-complexity fault detection scheme for the AES decryption.

Corollary 6.3 *Let* $s' = s'_7\alpha^7 + s'_6\alpha^6 + s'_5\alpha^5 + s'_4\alpha^4 + s'_3\alpha^3 + s'_2\alpha^2 + s'_1\alpha + s'_0 \in GF(2^8)$, *and* $s = s_7\alpha^7 + s_6\alpha^6 + s_5\alpha^5 + s_4\alpha^4 + s_3\alpha^3 + s_2\alpha^2 + s_1\alpha + s_0 \in GF(2^8)$ *be the 8-bit input and output of the inverse S-box. Then, the following equation holds for all the possible patterns of* s *and* s'.

$$\begin{aligned} P_{(\boldsymbol{M}\boldsymbol{s}'+\boldsymbol{m})} &= s'_0 s_a + s'_1 s_b + s'_2 s_c + s'_3(s_a + s_4) + s'_4(s_b + s_3 \\ &+ s_7) + s'_5(s_a + s_7) + s'_6(s_b + s_6) + s'_7(s_5 + s_c) + s_6 \\ &+ s_7 = u', \end{aligned} \tag{6.43}$$

where, $s_a = s_0 + s_1 + s_5$, $s_b = s_0 + s_4$, $s_c = s_a + s_2 + s_6$, *and* $u' = (s_0 \vee s_1 \vee \ldots s_7) \vee (\overline{s'_0} \vee \overline{s'_1} \vee s'_2 \vee s'_3 \vee s'_4 \vee \overline{s'_5} \vee \overline{s'_6} \vee s'_7)$.

Proof According to Theorem 6.3 and Corollary 6.2, one can re-write (6.21) and swap the input and the output to derive (6.43). Therefore, the proof is complete. ∎

To implement the signature presented in the left hand side of (6.43), 20 XOR gates and 8 AND gates are needed. Then, it is compared with u' to obtain the error indication flag of each inverse S-box.

Using Corollary 6.3 and Theorem 6.5, we derive 16 low-complexity signatures for the AddRoundKey and InvMixColumns transformations, i.e., 4 signatures for each column of the state matrix. This is performed by modulo-2 addition of two sets of four coordinates of (6.42) for each column, i.e., $E_c = (e_{c,7}, e_{c,6}, ..., e_{c,0}) \in GF(2^8)$, $0 \leq c \leq 3$. For the

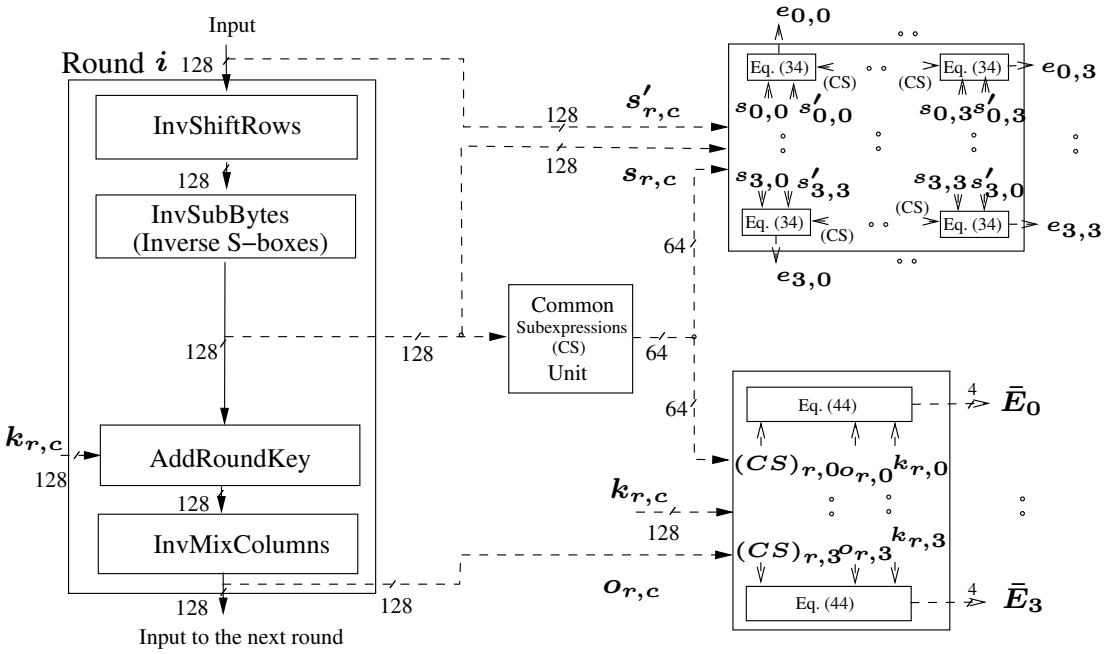


Figure 6.5: The proposed low-complexity fault detection scheme for the *ith* round of the AES decryption utilizing subexpression sharing.

AES decryption, let $\acute{E}_c = (e_{c,3}, e_{c,2}, e_{c,1}, e_{c,0})$ and $\grave{E}_c = (e_{c,7}, e_{c,6}, e_{c,5}, e_{c,4})$. Then, the four error indication flags for column c of the state are

$$\overline{E}_c = \acute{E}_c + \grave{E}_c, \quad 0 \leq c \leq 3. \tag{6.44}$$

One can utilize four sets of modulo-2 additions of the output bits of each inverse S-box pre-computed in Corollary 6.3, i.e., $s_0 + s_4$, $s_1 + s_5$, $s_2 + s_6$ and $s_3 + s_7$, to obtain the low-complexity error indication flags in (6.44). This is shown in Fig. 6.5. As seen in this figure, similar to the AES encryption, the *Common Subexpressions* (CS) unit has been utilized to obtain 64 common subexpressions. Then, these outputs are used in obtaining the two sets of 16 error indication flags for the AES decryption, respectively. It is noted that in Fig. 6.5, the hardware implementation of (6.43) is used in (6.34) which is less complex when the common subexpressions are used.

The proposed fault detection scheme for InvMixColumns requires 48 XOR gates with two XOR gates in the critical path. Compared to the scheme presented in [36] for the InvMixColumns transformation, the proposed scheme has less area and critical path delay. It is noted that the authors in [36] have not presented the equations for the

parity-based fault detection scheme of InvMixColumns, mentioning that they have the same structure as those of MixColumns but they are more complicated. Therefore, at least a 25% area overhead reduction and a 33% reduction in the critical path delay are expected for the proposed scheme.

6.4 Error Simulations

We have considered both single and multiple stuck-at errors for the proposed scheme. These models cover both natural faults and fault attacks [82]. If exactly one bit error appears at the output of the AES encryption or decryption rounds, the presented parity-based fault detection scheme is able to detect it and the error coverage of the proposed scheme is about 100%. This is because in this case, one of the 8-bit four error indication flags in (6.33) or (6.44) alarms the error. However, due to the technological constraints, single stuck-at error may not be applicable for an attacker to flip exactly one bit to gain more information [82]. Thus, multiple bits will actually be flipped and hence multiple stuck-at errors are also considered in this chapter.

For the multiple stuck-at error models, we rely on simulations for both burst and random errors. In the case of fault attacks, it is more likely that a transient burst error appears instead of one-bit flips due to the present constraints [82]. Moreover, most internal faults are modeled by transient random errors [82]. It is noteworthy that the results of our simulations are valid for the transient errors. Furthermore, in case of occurring permanent internal faults, the same simulation results are achieved.

We use stuck-at error model at the outputs of the AES transformations. This type of error forces multiple nodes to be stuck at logic one (for stuck-at one) or zero (for stuck-at zero) independent of the error-free values. It is noted that we use Fibonacci implementation of the Linear Feedback Shift Registers (LFSR) with 128 output taps for injecting random multiple errors, where, the numbers, locations and types of the errors are randomly chosen. In this regard, maximum sequence length polynomial for the feedback is selected as $L(X) = X^{128} + X^{29} + X^{27} + X^2 + 1$ according to the maximum sequence length taps presented in [91].

We use the fault detection schemes presented in the previous section and shown in Fig. 6.3 and Fig. 6.5 for the AES encryption and decryption, respectively. In our simulations

using Xilinx® ISE™ version 9.1 Simulator [80], we use the error indication flags at the outputs of ShiftRows (cover the errors for SubBytes and ShiftRows) and AddRoundKey (cover the errors for MixColumns and AddRoundKey) for the AES encryption in Fig. 6.3. Moreover, for the AES decryption in Fig. 6.5, we obtain the error indication flags at the outputs of InvSubBytes (cover the errors for InvShiftRows and InvSubBytes) and InvMixColumns (cover the errors for AddRoundKey and InvMixColumns). The results of our simulations show that by having these two sets of error indication flags, an acceptable error coverage is achieved.

In our simulations, we inject errors in two manners, i.e., burst and random errors, and obtain the error coverage for these two cases, the details of which are as follows.

Burst Errors

The first type of errors that we consider is the burst errors. For this type of errors, we assume that stuck-at errors occur at the output of only one transformation at a time, i.e., the errors are injected at the 128-bit output of only one transformation in the AES encryption/decryption in Fig. 6.3 and Fig. 6.5. This includes both stuck-at zero and stuck-one errors. Then, using two series of 16-bit signatures shown in these figures, the error coverage is obtained. The results of our simulations for the burst errors in the AES encryption and decryption are shown in Fig. 6.6. In this figure, the solid and dashed lines represent the error coverage for the AES encryption and decryption, respectively. As seen in this figure, we have injected up to $700,000$ burst errors at the transformation outputs, one at a time, and have monitored the errors that are covered by the error indication flags. It is noted that because the errors are injected only at the output of one transformation, only one of the two series of the error indication flags can detect them. As seen in this figure, after injecting up to $700,000$ burst errors, for both the AES encryption and decryption, the error coverage for the two sets of error indication flags is greater than 99.996%.

Random Errors

The second type of errors is random errors, where errors are injected at random locations, i.e., four 128-bit outputs of the transformations. Our simulations show that after injecting up to $700,000$ random errors, the higher error coverages of very close to 100% are

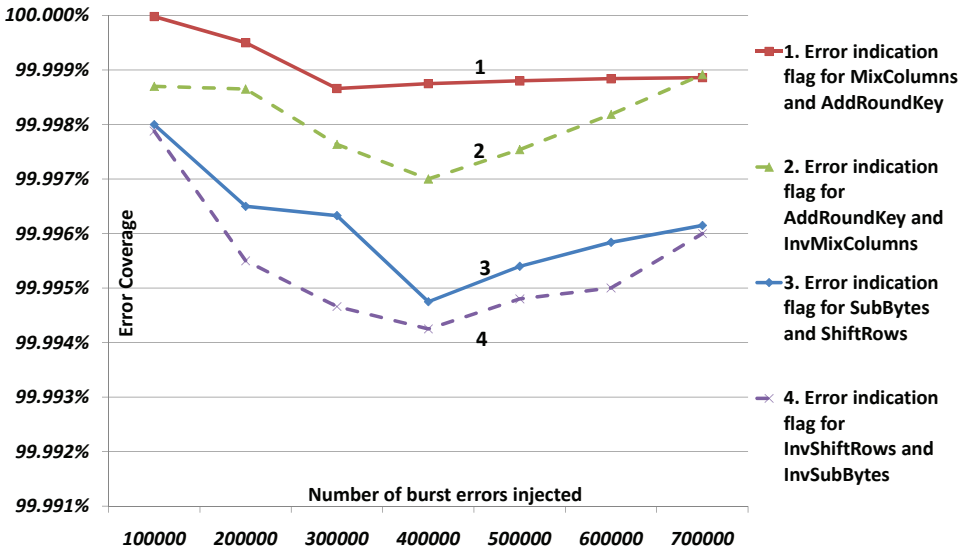


Figure 6.6: Simulation results for the error coverages of the proposed fault detection schemes.

obtained, i.e., all the errors are covered by at least one of the two series of the error indication flags. We also expect the error coverage of close to 100% if we increase the number of errors injected. The high error coverages of the proposed scheme for the AES rounds is suitable for the security-constrained applications on FPGAs. These include any AES algorithms implemented on the FPGAs as well as the bitstream security mechanisms.

6.5 AES FPGA Implementations and Comparisons

The proposed schemes in this chapter are structure-independent and can be applied to the AES using both the LUT-based and the composite field S-boxes and inverse S-boxes. In this section, we have implemented both of these structures so that we are able to compare the results for the presented schemes with those using LUTs and composite fields. In what follows, we consider the implementation of both the AES encryption and decryption.

For the FPGA implementations, we have used VHDL as the design-entry for ISE™ version 9.1. Furthermore, the synthesis is performed using Xilinx® Synthesis Tool (XST™) on Virtex™-4 and Virtex™-5 families [80]. It is noted that the results of the implementations in this section, i.e., the number of occupied slices and the minimum periods (maximum working frequencies), are all post place and route results.

We have implemented the original AES using LUT-based S-boxes and inverse S-boxes on Virtex™-4 (xc4vlx160-12) and Virtex™-5 (xc5vlx110-3) devices. These larger devices are chosen to have enough number of slices needed for the fault detection scheme in [35] and [36]. We have used pipelined distributed memories for the LUT-based S-boxes and inverse S-boxes in the AES to increase the design speed and the overall frequency. The XST™ uses the LUT resources in the FPGAs in order to implement the distributed memories. Furthermore, pipelining is achieved by describing the necessary registers in the design-entry language. The schemes in [34], [35], [36], [39], Hardware Redundancy and the proposed ones in this chapter have been implemented and the results are depicted in Table 6.1. As seen in this table, the Error Coverage (EC %), the number of occupied slices, the maximum working frequency (MHz), the throughput (Gbps) and the efficiency (Mbps/slice) for the original schemes and the Fault Detection (FD) ones are derived. Moreover, the slice overheads (overheads for the number of occupied slices) are presented. It is noted that there is a difference in the implementations of the LUT-based S-boxes and inverse S-boxes using distributed memories for the selected FPGAs. Specifically, for Virtex™-5 and Virtex™-4, 256 and 64 bits per CLB are specified for the distributed memories, respectively. This causes the LUT implementations for Virtex™-5 to be more compact as compared to those on Virtex™-4 [80]. This can be observed in Table 6.1. In this regard, the number of slices for the original AES encryption and decryption using LUTs and the slice overhead for the scheme in [35] and [36] whose area overhead is dominated by the expansion of the S-box to 512×9 memories is less on Virtex™-5. This makes Virtex™-5 a suitable family for the AES using memory-based S-boxes and inverse S-boxes and their fault detection schemes. Because of the higher number of slices for the original AES encryption and decryption on Virtex™-4, the slice overheads of the proposed schemes and the scheme in [39] are less as compared to those for Virtex™-5.

Table 6.1: Comparisons of the implementations of the fault detection schemes of the AES using LUT S-boxes and inverse S-boxes on Xilinx® FPGAs.

FPGA family (Device)	FDS		Encryption				Decryption			
	Scheme	EC(%)	Slice *(overhead)*	Freq. (MHz)	Thro. (Gbps)	Eff.(Mbps /slice)	Slice *(overhead)*	Freq. (MHz)	Thro. (Gbps)	Eff.(Mbps /slice)
Virtex™-4 (xc4vlx160 -12)	Original	-	18335 *(-)*	240.5	30.8	1.7	19322 *(-)*	203.5	26.0	1.3
	Algorithm-level [34]	100%	38273 *(108.7%)*	194.9	24.9[a]	0.6	38273 *(98.1%)*	194.9	24.9[a]	0.6
	Hardware Redundancy	100%	28905 *(57.6%)*[b]	240.5	30.8	1.1	35421 *(83.3%)*	203.5	26.0	0.7
	FD in [35][c] for encryption FD in [36][c] for decryption	99.997%	39104 *(113.3%)*	163.5	20.9	0.5	40244 *(108.3%)*	145.4	18.6	0.5
	FD in [39][d] from the general scheme in [37]	98.7% sin. 48-53% mult.	21211 *(15.7%)*	240.5	30.8	1.4	22280 *(15.3%)*	203.5	26.0	1.1
	Proposed (this chapter)	**99.996%**	**20127** ***(9.8%)***	**240.5**	**30.8**	**1.5**	**20909** ***(8.2%)***	**203.5**	**26.0**	**1.2**
Virtex™-5 (xc5vlx110 -3)	Original	-	2960 *(-)*	371.7	47.6	16.1	3906 *(-)*	296.3	37.9	9.7
	Algorithm-level [34]	100%	5849 *(97.6%)*	284.4	36.4[a]	6.2	5849 *(49.7%)*	284.4	36.4[a]	6.2
	Hardware Redundancy	100%	4637 *(56.7%)*[b]	371.7	47.6	10.2	7200 *(84.3%)*	296.3	37.9	5.5
	FD in [35][c] for encryption FD in [36][c] for decryption	99.997%	5590 *(88.9%)*	282.8	36.2	6.5	6688 *(71.2%)*	260.2	33.3	4.9
	FD in [39][d] from the general scheme in [37]	98.7% sin. 48-53% mult.	3619 *(22.3%)*	304.0	38.9	10.7	4426 *(13.3%)*	277.0	35.5	8.0
	Proposed (this chapter)	**99.996%**	**3757** ***(26.9%)***	**371.7**	**47.6**	**12.7**	**4286** ***(9.7%)***	**296.3**	**37.9**	**8.8**

[a]The latency is twice as much as the original AES encryption or decryption.

[b]Although the overhead of greater than 100% is expected, the overhead for the number of occupied slices is less.

[c]Using two (256×9) memories for the fault detection of each S-box or inverse S-box.

[d]Using (256×9) memories for the fault detection of each S-box or inverse S-box.

As seen in Table 6.1, the number of slices for the original decryption is more than that of the encryption. This is mainly because of the InvMixColumns transformation which is more complex than MixColumns in the AES encryption. Furthermore, the slice overhead for the scheme in [36] in which the LUTs sizes are expanded to 512×9 is less on Virtex^{TM}-5 family compared to Virtex^{TM}-4. As seen in Table 6.1 in bold faces, the proposed structure-independent scheme for the AES decryption is the most efficient and the most compact one among the other schemes. Moreover, for the Virtex^{TM}-5, the proposed scheme for the AES encryption has the least slice overhead. However, the slice overhead of the proposed scheme implemented on Virtex^{TM}-4 is slightly more than that of the scheme in [39]. It is noted that the low overhead of the scheme in [39] is because it uses one-bit signatures for the 128-bit block of data. While, the proposed schemes and the one in [35] and [36] use 16 bits for each 128-bit block. As seen in Table 6.1, this leads to much higher error coverage.

The scheme in [38] is based on using the output of the multiplicative inversion (not that of the S-box) to obtain a signature for fault detection. This scheme cannot be applied to the S-boxes using LUTs where the output of the multiplicative inversion is not accessible. Therefore, we have implemented the original AES encryption which uses the S-boxes using polynomial basis and composite fields in order to have access to the output of the multiplicative inversion. For this reason, we utilize the AES presented in [22]. This implementation of the AES is a hardware optimization for the scheme in [20], which is extensively used in the literature, see for example [13], [15]. Then, we have implemented the scheme of [38] and compared it with the proposed scheme presented in this chapter. Moreover, the scheme in [34] and Hardware Redundancy have been implemented.

The results of the implementations are shown in Table 6.2. It is worth noting that in [38], the fault detection scheme for the AES decryption is not presented. Therefore, no comparison for the AES decryption with this scheme is presented in this table. It is noted that we have not used sub-pipelining for the implementations and registers are only used at the output of each round. Using sub-pipelining for the S-boxes using composite fields, one can reach higher working frequencies compared to those for LUT-based S-boxes. As seen in this table, the number of slices for the original AES encryption using S-boxes in composite fields is less than those of the LUTs for Virtex^{TM}-4 (compare Tables 6.1

and 6.2 for Virtex^{TM}-4). However, the original AES using LUT-based S-boxes is more compact when Virtex^{TM}-5 is used. As mentioned before, this is due to the low number of slices needed for the implementation of the memories in this device family. As seen in Table 6.2, the proposed scheme is the most compact and the most efficient scheme compared to the scheme in [38], i.e., the efficiency degradations (percent degradation from the efficiency of the original operations) and the slice overheads are the least for two devices. It is noted that the proposed scheme in this chapter uses 16 error indication flags for the 128-bit output states of the transformations. However, the scheme in [38] utilizes 32 error indication flags for each output state. Therefore, more slice overhead and greater error coverage are expected for that scheme. However, as discussed earlier, this scheme cannot be applied to the AES using LUTs.

Furthermore, we have compared the proposed schemes in this chapter with the light weight concurrent fault detection scheme for the AES S-boxes presented in [78]. This scheme is based on using normal basis for logic gate implementations of the S-boxes in the AES encryption. In this fault detection scheme, the structure of the S-box using normal basis has been divided into 5 blocks. Then, the predicted parities of these blocks are obtained. Moreover, through an exhaustive search among all available composite fields, the optimum solution for the least overhead S-box and its parity predictions is achieved. We have implemented the AES encryption with the original S-boxes using normal basis in composite fields proposed in [23] and verified with the FPGA implementations in [78]. Then, the fault detection scheme for the S-boxes in [78] has been utilized for the SubBytes transformation while the proposed scheme in this chapter is used for the other transformations. In other words, we derive 5 error indication flags for each S-box in SubBytes ($5 \times 16 = 80$ flags for the entire SubBytes transformation), while the scheme in Fig. 6.3 is used for other AES encryption transformations using 16-bit flags. Moreover, the proposed signature-based structure-independent scheme in this chapter, i.e., the scheme in Fig. 6.3, has been implemented for the AES encryption with the S-boxes using normal basis. The results of these implementations are also presented and compared in Table 6.2. As seen in this table, the FPGA implementations of the original AES encryption with the S-boxes using normal basis representation in composite fields have less area compared to the traditional ones using polynomial basis, i.e., 6752 and 3692 compared to 7498 and 3718 for two devices, respectively.

Table 6.2: Implementation comparisons of the fault detection schemes of the AES encryption using composite field S-boxes on Xilinx® FPGAs.

FPGA	S-boxes structures[a]	FDS	Slice			Thro. (Gbps)		Eff. (Mbps/slice)		
			Org.	FDS	Over.	Org.	FDS	Org.	FDS	deg.
Virtex™-4	PB[b]	Algorithm-level [34]	7498	17075	127.7%	14.6	14.6[c]	1.9	0.9	52.6%
	PB[b]	Hardware Redundancy	7498	14968	99.6%	14.6	14.6	1.9	1.0	47.4%
	PB[b]	[38]	7498	10340	37.9%	14.6	12.3	1.9	1.2	36.8%
	PB[b]	**Proposed (Fig. 6.3)**	**7498**	**9252**	**23.4%**	**14.6**	**12.6**	**1.9**	**1.4**	**26.3%**
	NB[d]	**Proposed**[e]	**6752**	**9325**	**38%**	**17.1**	**12.9**	**2.5**	**1.4**	**44%**
	NB[d]	**Proposed (Fig. 6.3)**	**6752**	**8216**	**21.7%**	**17.1**	**12.3**	**2.5**	**1.8**	**28.0%**
Virtex™-5	PB[b]	Algorithm-level [34]	3718	7492	101.5%	18.2	15.7	4.9	2.1	57.1%
	PB[b]	Hardware Redundancy	3718	7162	92.6%	18.2	16.7	4.9	2.3	53.1%
	PB[b]	[38]	3718	4750	27.8%	18.2	14.2	4.9	3.0	38.8%
	PB[b]	**Proposed (Fig. 6.3)**	**3718**	**4354**	**17.1%**	**18.2**	**14.6**	**4.9**	**3.4**	**30.6%**
	NB[d]	**Proposed**[e]	**3692**	**4683**	**26.8%**	**17.9**	**14.8**	**4.8**	**3.1**	**35.4%**
	NB[d]	**Proposed (Fig. 6.3)**	**3692**	**4286**	**16.1%**	**17.9**	**14.5**	**4.8**	**3.4**	**29.2%**

[a]The original AES encryption implementations differ only in the S-boxes structures.

[b]The S-boxes using polynomial basis presented in [22].

[c]Although the throughput is the same as the original AES, the latency is twice as much as the original one.

[d]The original S-boxes using normal basis in composite fields proposed in [23] and verified with the FPGA implementations in [78].

[e]Fault detection scheme for the S-boxes from [78] and the proposed scheme in this chapter for the other transformations.

In addition, the proposed structure-independent scheme in this chapter has the least area overhead complexities and the most efficiencies for both FPGA families. At this point, we would like to mention that for the scheme in [78], higher error coverage and slightly higher throughput are achieved compared to the proposed scheme in this chapter. However, this is at the cost of the higher area overhead complexity. It is also noted that the fault detection scheme in [78] not only can be only applied for the composite field S-boxes but it is also dependent on the composite fields and normal basis chosen, i.e., the parity predictions would be different if other composite fields are used. Whereas, the proposed scheme in this chapter is independent of the structures of the S-boxes used in the AES encryption.

Recently, a fault tolerant approach which is resistant to fault attacks is proposed in [50]. This approach is based on protecting the logic blocks and memories of the AES. To protect the combinational logic blocks used in the four rounds of the AES, either the parity-based scheme proposed in [36] or the duplication one presented in [96] is implemented. Furthermore, to protect the memories used for storing the expanded key and the state matrix either the Hamming or Reed-Solomon error correcting code is implemented. The results of the comparison of the proposed scheme in this chapter with the parity-based scheme of [35] and [36] for protecting the combinational logic elements of the AES are depicted in Table 6.1. Moreover, for certain AES implementations containing storage elements, one can use the error correcting code-based approach presented in [50] in addition to the proposed scheme in this chapter to make a more reliable AES implementation.

To conclude, in this chapter, we have studied a number of fault detection schemes for the encryption and the decryption of the AES. New fault detection schemes which are independent of the structures of the S-boxes and the inverse S-boxes have been proposed. Our simulations show that for the AES encryption and decryption, these structure-independent schemes reach high error coverage.

Furthermore, our proposed fault detection schemes and almost all of the previously reported ones have been implemented on the recent Xilinx® Virtex™ FPGAs. Their area and delay overheads for the AES encryption and decryption have been derived and compared. In our implementations, we have considered using both the look-up table-based and the composite field AES structures. Our FPGA implementations show that

for the AES encryption, the slice overhead of the proposed scheme is around 9.8% to 26.9%, depending on the FPGA family and the AES implementation. In addition, for the AES decryption, lower slice overhead is achieved. These slice overheads are less than those for the other schemes which have the same error coverages.

According to our simulation and implementation results, with acceptable error coverages, the structure-independent schemes proposed in this chapter have the highest efficiencies, showing reasonable area and time complexity overheads. Based on the AES structure chosen, the performance goals to achieve, and the resources available, one can use combinations of the presented schemes in order to have much more reliable AES encryption and decryption structures.

Chapter 7

Efficient and High-Performance Parallel Hardware Architectures for the AES-GCM

IN the previous chapters, we have proposed different high-performance fault diagnosis approaches for the AES. These approaches help making the AES hardware architectures reliable. In this chapter, we present high-speed, parallel hardware architectures for reaching low-latency and high-throughput structures of the GCM. By investigating the high-performance $GF(2^{128})$ multiplier architectures, we benchmark the proposed AES-GCM architectures using quadratic and sub-quadratic hardware complexity $GF(2^{128})$ multipliers. It is shown that the performance of the presented AES-GCM architectures outperforms the previously reported ones in the utilized 65-nm CMOS technology.

In this chapter, using a complexity reduction technique, the hardware complexities of different architectures for the subkey exponentiations in the GCM are reduced. Then, by utilizing these low-complexity exponentiations, we propose efficient architectures for the GCM, yielding high throughput and low latency. The proposed hardware architectures for the AES-GCM are synthesized considering two types of $GF(2^{128})$ multipliers. We investigate the performance of quadratic and six different sub-quadratic complexity $GF(2^{128})$ multipliers. It is shown that the proposed architectures for the AES-GCM have higher throughput and efficiency and reach lower latency compared to the previously reported ones.

The organization of this chapter is as follows. In Section 7.1, the proposed high-performance architectures for implementing the GCM are presented. Section 7.2 presents the ASIC syntheses and comparisons of the proposed architectures and the previously

reported ones. The results presented in this chapter can also be found in [72].

7.1 High-Performance GCM Parallel Architecture

In this section, we propose high-performance parallel architectures for the GCM. These architectures improve the throughput and the latency of the structures presented in [68] and [69] for $GHASH_H$. They also remove the need for consecutive $GF(2^{128})$ multiplications with H for deriving (1.1). We also derive the hardware implementations of the exponentiations of the hash subkey to the powers of 2, i.e., in the form of H^{2^j}, needing only XOR gates. Because of the low complexity of the implementations of these exponents, we take advantage of these low-cost hash subkey powers in the proposed high-performance architectures. We utilize the powers in the form of H^{2^j} to obtain the other powers of the hash subkey with the least number of GF multiplications over $GF(2^{128})$ for proposed architectures. For instance, we derive $H^3 = H^2 \times H$ or $H^6 = H^4 \times H^2$.

7.1.1 High-Performance $GHASH_H$ Function

Algorithm 3 is used for obtaining the key formulation for the proposed $GHASH_H$ function. Although there is no restriction in choosing q, i.e., the number of parallel adder-multipliers, we use $q = 2^j$, $1 \leq j \leq \lfloor \log_2(n) \rfloor$. This leads to lower number of clock cycles and higher throughput needed for the implementations. In Algorithm 3, the output $GHASH(X, H)$ is obtained as follows:

$$\begin{aligned}
&X_1 \cdot \underbrace{H^q \times \ldots \times H^q}_{\frac{n}{q}\, times} \oplus X_2 \cdot \underbrace{H^q \times \ldots \times H^q}_{\frac{n}{q}-1\, times} \times H^{q-1} \oplus \ldots \\
&\oplus X_j \cdot \underbrace{H^q \times \ldots \times H^q}_{\frac{n}{q}-1\, times} \times H^{q-j+1} \oplus \ldots \\
&\oplus X_q \cdot \underbrace{H^q \times \ldots \times H^q}_{\frac{n}{q}-1\, times} \times H \oplus X_{q+1} \cdot \underbrace{H^q \times \ldots \times H^q}_{\frac{n}{q}-1\, times} \\
&\oplus X_{q+2} \cdot \underbrace{H^q \times \ldots \times H^q}_{\frac{n}{q}-2\, times} \times H^{q-1} \oplus \ldots \oplus X_n H, \qquad (7.1)
\end{aligned}$$

where all operations are performed over $GF(2^{128})$ constructed by the irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$ and $\bigoplus$ comprises 128 XOR gates.

One can re-write (7.1) so that only the exponentiations of the hash subkey to the powers of 2 in the form of H^{2^j} are utilized. This method of exponentiation is based on

Algorithm 3 The proposed high-performance approach for implementing the GCM.

Inputs: $X_p \in GF(2^{128})$, $1 \leq p \leq n$, and $H^{2^j} \in GF(2^{128})$, $0 \leq j \leq \log_2(q)$.
Output: $GHASH(X,H)=\sum_{j=1}^{n} X_j H^{n-j+1}$.

1: **for** $i = 1$ to q **do**
2: $temp_i \leftarrow X_i$
3: **for** $j = 1$ to $\frac{n}{q} - 1$ **do**
4: $temp_i = (temp_i \times H^q \oplus X_{i+jq})$
5: **end for**
6: Let $q - i + 1 = ({a_0}^{(i)}, \ldots, {a_{\log_2(q)}}^{(i)})_2$
7: $temp_i = temp_i \times (H^{{a_0}^{(i)} q} \times H^{\frac{{a_1}^{(i)} q}{2}} \times \ldots \times H^{{a_{\log_2(q)}}^{(i)}})$
8: **end for**
9: $GHASH(X,H) = \sum_{i=1}^{q} temp_i$
10: **return** $GHASH(X,H)$.

the binary exponentiation, see, for example, [97]. As seen from this algorithm, for the exponentiations H^{q-i+1}, $1 \leq i \leq q$, one can use the binary representation of $q - i + 1$ as $({a_0}^{(i)}, \ldots, {a_{\log_2(q)}}^{(i)})_2$.

The hardware implementation of Algorithm 3 has been presented in Fig. 7.1. For implementing Algorithm 3 in hardware, in total, $\frac{n}{q} + \log_2(q)$ clock cycles are needed. For the first $\frac{n}{q} - 1$ clock cycles, the $GF(2^{128})$ multiplications by H^q are performed. This is achieved by a simple control unit selecting H^q. Then, for the next $\log_2(q)$ clock cycles, the other exponentiations are used. These include the powers of the hash subkey in the form of H^{2^j} and a number of field elements $1 = (0, 0, ..., 1) \in GF(2^{128})$ for bypassing the $GF(2^{128})$ multiplication operations. We note that if n is not a multiple of q, one needs to add $q - \text{mod}\ (n, q)$ blocks containing $0 = (0, 0, ..., 0) \in GF(2^{128})$ to the beginning of the n blocks to make the total blocks processed multiple of q. Performing this, the hash computation can be done normally based on the presented procedure. As seen in Fig. 7.1, q adder-multipliers are required and multiplexers are also utilized to select different exponentiations.

To illustrate the proposed scheme, we use the case with $n = 16$ and $q = 8$. In the first clock cycle ($j = 1$), the outputs of all the multiplexers in Fig. 7.1 are H^8 for this case. Then, according to the following, the outputs of the multiplexers in the other cycles can

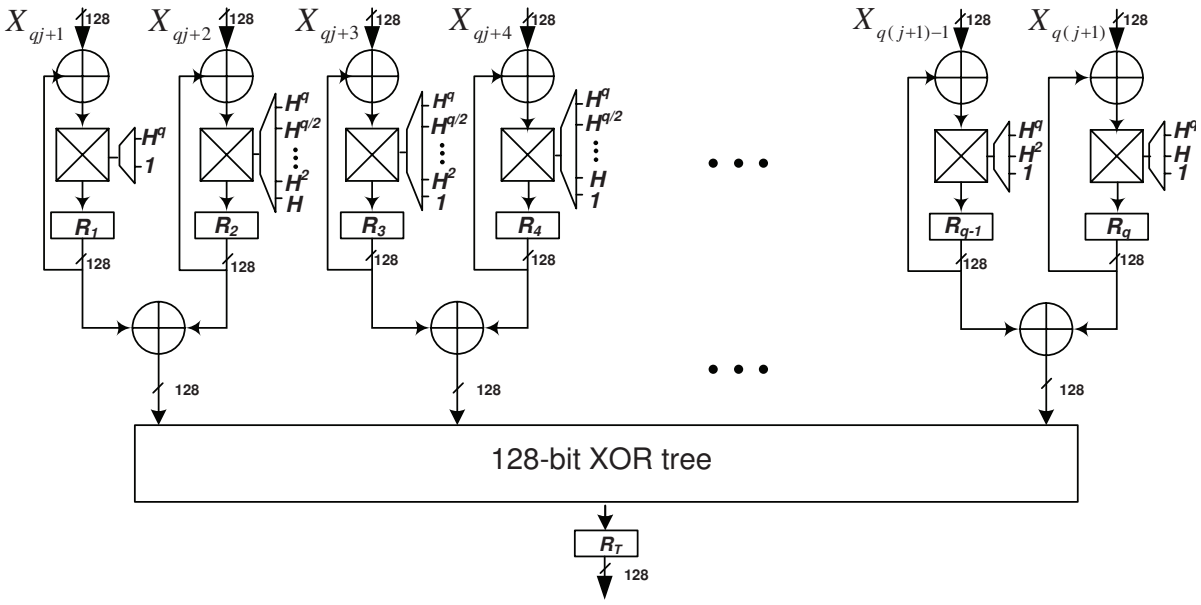


Figure 7.1: The hardware architecture of the proposed high-performance GCM $GHASH_H$ function.

be found.

$$
\begin{aligned}
&\underbrace{\underbrace{\overbrace{(X_1H^8}^{j=1}\oplus X_9)H^8}_{j=2}\times 1\times 1}_{j=4}\oplus\underbrace{\underbrace{\overbrace{(X_2H^8}^{j=1}\oplus X_{10})H^4}_{j=2}\times H^2\times H}_{j=4}\oplus\\
&\ldots\oplus\underbrace{\underbrace{\overbrace{(X_iH^8}^{j=1}\oplus X_{i+8})H^{4a_1^{(i)}}}_{j=2}\times H^{2a_2^{(i)}}\times H^{a_3^{(i)}}}_{j=4}\oplus\\
&\ldots\oplus\underbrace{\underbrace{\overbrace{(X_8H^8}^{j=1}\oplus X_{16})H}_{j=2}\times 1\times 1}_{j=4},
\end{aligned}
\tag{7.2}
$$

where $(a_1, a_2, a_3)_2$ is the binary representation of $q-i+1=9-i$, $1\le i\le 8$. Five cycles are required to implement (7.2); 4 cycles are shown in (7.2) with $j=1$ to $j=4$, and the last one is used for the addition of the results of the registers R_1-R_8 to have the final result in R_T.

Table 7.1: Performance analysis and comparison of $GHASH_H$ within the GCM for n blocks and q parallel structures.

Approach	Latency	Throughput
Sequential [64], [65], [66]	n	$\frac{128}{(T_{mul}+T_X)n}$
[68], [69]	$\frac{n}{q}+q-1$	$\frac{128}{(T_{mul}+T_X)(\frac{n}{q}+q-1)}$
Proposed	$\frac{n}{q}+\log_2(q)$	$\frac{128}{(T_{mul}+T_X)(\frac{n}{q}+\log_2(q))}$

According to Fig. 7.1, the working frequency of the proposed scheme is obtained as $T_{mul}+T_X$ (we note that this delay is larger than that of the XOR tree). It is noted that T_{mul} is the time delay of the used multiplier and T_X is the time delay of one set of modulo-2 additions in the critical path. Furthermore, according to Algorithm 3, the number of clock cycles needed for the $GHASH_H$ function is $\frac{n}{q}+\log_2(q)$. Latency and throughput of the proposed scheme are compared with the ones presented in [64], [65], [66], [68], and [69] in Table 7.1. As seen in this table, the sequential approach has the least throughput which leads to low-performance hardware implementations. The throughput of the proposed scheme, i.e., $\frac{128}{(T_{mul}+T_X)(\frac{n}{q}+\log_2(q))}$, is higher than that of the scheme in [68] and [69], i.e., $\frac{128}{(T_{mul}+T_X)(\frac{n}{q}+q-1)}$, especially for high values of parallel structures, i.e., high values of q. For example, for the case presented in (7.2), the proposed architectures of this chapter need $\frac{n}{q}+\log_2(q)=2+3=5$ clock cycles to obtain the result. This can be compared with the linear relation of the scheme in [68] and [69] with q, leading to $\frac{n}{q}+q-1=2+8-1=9$ clock cycles needed. The complete comparison in terms of hardware and timing complexities of the proposed architectures with the previous ones is presented later in this chapter using ASIC syntheses.

As discussed earlier, with the change in the block cipher key, the re-calculations of the hash subkey and then raising it to different powers are inevitable. In the following, we present different methods in obtaining the hash subkey powers required in the proposed architectures of this section, i.e., H^{2^j}. Moreover, through complexity reduction techniques, low-complexity structures for the exponentiations are obtained in which the timing complexities are remained unchanged.

7.1.2 High-Speed Structures for Hash Subkey Powers

In the following, using squaring operations, we present three methods for implementing the hash subkey exponentiations. Using a complexity reduction algorithm, we also derive their hardware-optimized architectures.

According to [5], it is less likely that the GCM is invoked with the same key on distinct sets of input data. Thus, a new hash subkey and its powers need to be obtained in each invocation. It is known that the squaring operation in binary extension fields leads to a linear structure, see, for example, [98]. In other words, implementing squaring in hardware is less costly than $GF(2^{128})$ multiplications. The squaring of a field element over $GF(2^{128})$ in the GCM uses the irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$. Utilizing $P(x)$, we have obtained the formulations for the squaring after performing modular reduction. It is noted that MATLAB® [76] has been utilized to verify the formulations used for squaring. For the GCM, the critical path delay of squaring is obtained as $3T_X$, where T_X is the XOR gate delay. Moreover, it requires 202 XOR gates.

To implement H^{2^j}, $2 \leq j \leq \lfloor \log_2(q) \rfloor$, one can cascade j squaring architectures or use a feedback for deriving them. We refrain using the feedback structure because of its low throughput and high latency. According to the hardware and timing complexities of squaring derived in this section, for H^{2^j}, the cascade structure yields to the hardware and timing complexities of $202j$ XOR gates and $3j$ T_X, respectively. This leads to low-speed implementations which are not desirable in applications requiring high performance. It is possible to reduce the delay of the implementations of these exponentiations for the high-performance hardware implementations. To achieve this, we do not cascade the squaring implementations. Instead, we find the squaring exponentiations separately so that their derivations become in parallel. This reduces the critical path delay of the realizations. We present the following lemma for obtaining the exponentiations of the hash subkey within the GCM.

Lemma 7.1 *The squaring exponentiations of the hash subkey, i.e., H^{2^j}, $2 \leq j \leq \lfloor \log_2(q) \rfloor$, are obtained using the following.*

$$H^{2^j} \mod P(x) = d + \sum_{i=1}^{2^j - 1} \tilde{e}_i, \tag{7.3}$$

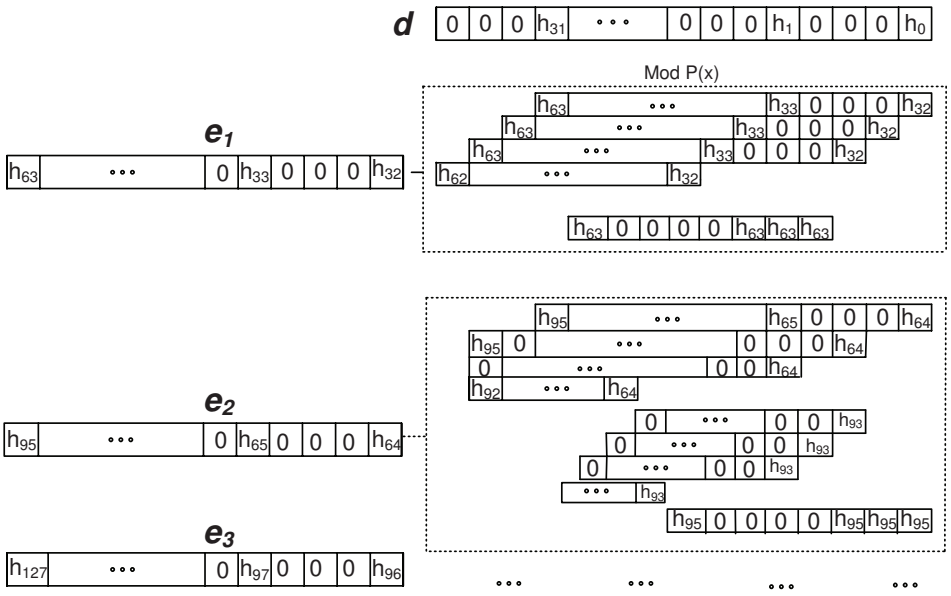


Figure 7.2: The derivation of H^4 of the GCM hash subkey.

where, d and $\tilde{e}_i$, $1 \leq i \leq 2^j - 1$, are field elements in $GF(2^{128})$ defined as follows

$$d = \sum_{s=0}^{\frac{128}{2^j}-1} h_s x^{2^j \times s}, \quad \tilde{e}_i = \Big(\sum_{s=\frac{128i}{2^j}}^{\frac{128(i+1)}{2^j}-1} h_s x^{2^j \times s} \Big) \mod P(x)$$

Proof Let $H = \sum_{s=0}^{127} h_s x^s \in GF(2^{128})$ be the hash subkey of the *GHASH* function. Then, we have $H^{2^j} = (\sum_{s=0}^{127} h_s x^{2^j \times s}) \mod P(x) = \sum_{s=0}^{\frac{128}{2^j}-1} h_s x^{2^j \times s} + (\sum_{s=\frac{128}{2^j}}^{127} h_s x^{2^j \times s} \mod P(x)) = d + (\sum_{i=1}^{2^j-1} \sum_{s=\frac{128i}{2^j}}^{\frac{128(i+1)}{2^j}-1} h_s x^{2^j \times s}) \mod P(x) = d + \sum_{i=1}^{2^j-1} \tilde{e}_i$ and the proof is complete. ∎

For clarifying the method, we present the structure for deriving H^4 in Fig. 7.2. We obtain the polynomials d and e_1-e_3 in (7.3) as: $d = h_{31}x^{124} + h_{30}x^{120} + \ldots + h_0$, $e_1 = h_{63}x^{252} + h_{62}x^{248} + \ldots + h_{32}x^{128}$, $e_2 = h_{95}x^{380} + h_{94}x^{376} + \ldots + h_{64}x^{256}$, and $e_3 = h_{127}x^{508} + h_{126}x^{504} + \ldots + h_{96}x^{384}$. As seen in this figure, the coefficients of d are added with the reduced coefficients of e_1-e_3 using $P(x)$.

The complexity reduction techniques use different methods for decreasing the number of gates needed in the implementations, see, for example, the ones in [99] and [100]. Because it is not guaranteed that the delay of the method in [99] is maintained, we have implemented the complexity reduction algorithm presented in [100] using a C code. In

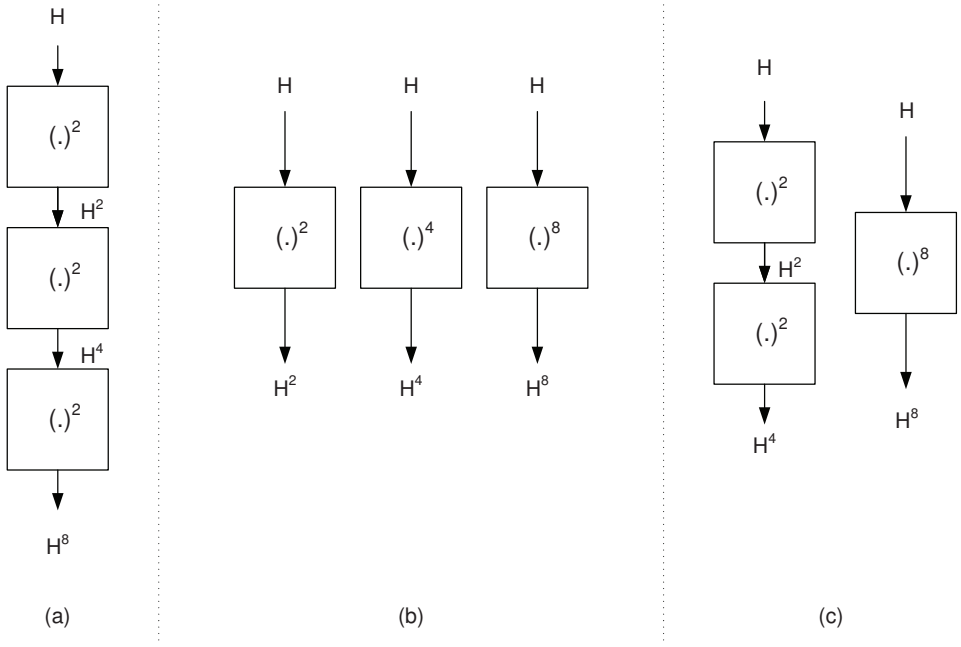


Figure 7.3: (a) Cascade, (b) parallel, and (c) hybrid realization methods for the hash subkey exponentiations.

our program, the procedure suggested in [100] (to find the shared XOR terms) has been utilized for the case study of $q = 8$, which requires implementing H^2, H^4 and H^8. It is noted that through the employed technique, we reach low hardware complexities without changing the critical path delays.

We have performed three experiments for implementing H^2, H^4 and H^8. These are shown in Fig. 7.3. As seen in Fig. 7.3a, in the cascade method, three identical squaring architectures are used consecutively. This method has the lowest hardware complexity and the highest timing complexity. In Fig. 7.3b, the parallel method of implementation of the hash subkey exponentiations is utilized. Compared to the other methods, this method has the lowest critical path delay while its hardware complexity is the highest. On the other hand, in the hybrid method which is shown in Fig. 7.3c, a compromise between hardware and timing complexities is achieved.

The timing and hardware complexities of these methods and the results of the complexity reduction technique utilized for them are depicted in Table 7.2. In this table, for three methods presented in Fig. 7.3, the hardware complexities before and after complexity reduction are derived. The timing complexity is remained unchanged after

Table 7.2: Complexities of the realizations of the hash subkey exponentiations for $q = 8$ parallel architectures for $GHASH_H$.

Method	Hardware Complexity	Hardware Complexity after complexity reduction	Complexity reduction(%)	Timing Complexity
Cascade (Fig. 7.3a)	606 XORs	**594 XORs**	$\approx 2\%$	$9T_X$
Parallel (Fig. 7.3b)	1986 XORs	1099 XORs	$\approx$ **45**%	$\mathbf{5T_X}$
Hybrid (Fig. 7.3c)	1627 XORs	1062 XORs	$\approx 35\%$	$6T_X$

applying the complexity reductions. As seen in Table 7.2 in bold face, the least hardware complexity is achieved for the cascade method after the complexity reduction, i.e., 594 XOR gates. However, the timing complexity of this method is the highest among the three methods as depicted in this table. On the other hand, the timing complexity of the parallel method is the lowest, i.e., $5T_X$. As shown in Table 7.2, this is at the expense of higher hardware complexity which is 1099 XOR gates after about 45% complexity reduction.

7.1.3 $GF(2^{128})$ Multipliers for the GCM

Different types of $GF(2^{128})$ multipliers are utilized in the literature for implementing the $GF(2^{128})$ multiplications in the GCM. In [64], [68], and [69], the multiplications have been performed using bit-parallel, digit-serial, and hybrid multipliers in composite fields. Furthermore, in [65] and [101], the efficiency of different multipliers, including the sub-quadratic ones, are compared. Moreover, in [102] a high-speed AES-GCM core has been presented. It is noted that the considered $GF(2^{128})$ multipliers in these works include the Mastrovito multiplier [103] with quadratic space complexity, the Karatsuba-Ofman multiplier [104] and the $GF(2^{128})$ multiplier in [105].

We have considered the bit-parallel $GF(2^{128})$ multiplier presented in [94] which has quadratic hardware complexity. It is noted that this $GF(2^{128})$ multiplier has lower timing complexity compared to the sub-quadratic hardware complexity $GF(2^{128})$ multipliers. However, we note that according to the latency of the proposed architectures, i.e., $\frac{n}{q} + \log_2(q)$, increasing the number of parallel structures (q) results in having higher throughputs. On the other hand, having higher values for q increases the hardware complexities of $GHASH_H$. Therefore, for reducing the hardware complexity, using sub-quadratic hardware complexity $GF(2^{128})$ multipliers is beneficial when high values of q are utilized.

Table 7.3: Hardware and timing complexities analysis of the utilized bit-parallel multipliers for the GCM.

Multiplier		GE[a]	Delay	Efficiency[b]
Complexity	Type			$(10^3 \times \frac{Throughput}{GE})$
Quad. [94]	-	56,957	$T_A + 10T_X$	$0.21/T_X$
Sub-quad. [106]	KO_1	44,338	$T_A + 12T_X$	$0.23/T_X$
	KO_2	34,660	$T_A + 14T_X$	$0.25/T_X$
	KO_3	28,195	$T_A + 16T_X$	$0.27/T_X$
	KO_4	24,517	$T_A + 18T_X$	$0.28/T_X$
	KO_5	23,443	$T_A + 20T_X$	$0.27/T_X$
	KO_6	24,961	$T_A + 21T_X$	$0.24/T_X$

[a]Gate equivalent in terms of two-input NAND.
[b]Considering $T_X = 1.99T_A$ according to the utilized technology.

For reducing the hardware complexity of the AES-GCM, we have also used the efficient realization of the Karatsuba-Ofman multiplier presented in [106] as the sub-quadratic hardware complexity $GF(2^{128})$ multiplier. It is noted that the gate count of different steps for one Karatsuba-Ofman multiplier has been presented in [106]. Based on our technology hardware and timing specifications, we have presented the performance of the $GF(2^{128})$ multipliers in Table 7.3. As shown in this table, six different steps for the Karatsuba-Ofman multipliers are considered. We denote these realizations by KO_1 (for the case that only one step is performed) to KO_6 (for which the 128-bit $GF(2^{128})$ multiplier is broken all the way to 2-bit multiplications using Karatsuba-Ofman method). Applying the Karatsuba-Ofman method recursively to obtain KO_i, $2 \leq i \leq 6$ for the GCM would result in low-area implementations with higher timing complexities. As seen from this table, although the sub-quadratic multiplier KO_5 is the most compact implementation, the sub-quadratic multiplier KO_4 reaches the best efficiency. In the next section, we present the synthesis results of these sub-quadratic multipliers for our proposed architectures. We also compare the power consumptions and the efficiencies of different methods for realizing these multipliers.

7.2 AES-GCM Performance Comparisons

In this section, first different AES architectures are presented and then we present and compare the ASIC synthesis results of the proposed and the previously presented archi-

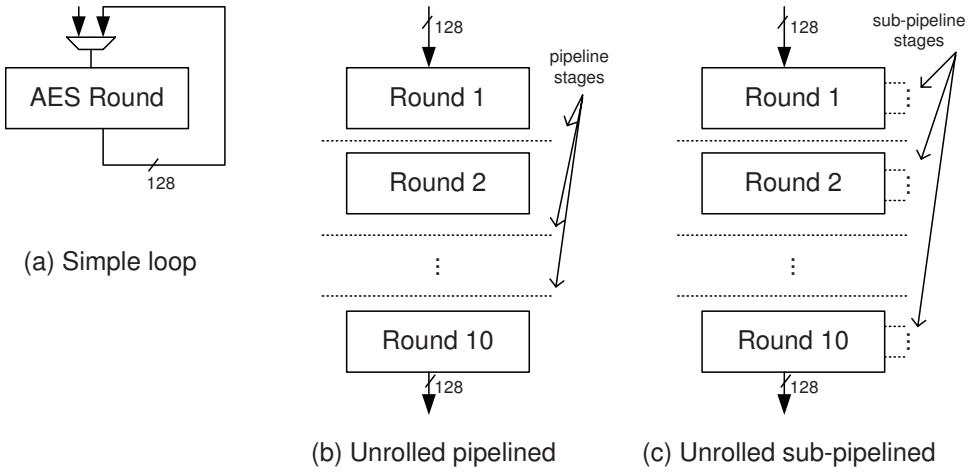


Figure 7.4: The AES-128 structure for (a) simple loop, (b) unrolled pipelined, and (c) unrolled sub-pipelined architectures (MixColumns is bypassed in the last round).

Table 7.4: The proposed architecture for the AES-GCM.

AES (Unrolled pipelined)	GCM (Proposed using Algorithm 3)	
	Exponents	Multiplier
PB S-box ($\Phi = \{11\}_2$, $\nu = \{1010\}_2$) optimized using (3.18)	Complexity-reduced parallel method (Table 7.2 and Fig. 7.3b)	Quad. and six sub-quad. (KO_1 -KO_6) in Table 7.3

tectures for the AES-GCM function.

We have presented different AES-128 architectures in Fig. 7.4. As seen in the AES simple loop structure (Fig. 7.4a), the AES rounds are executed serially (in the last round MixColumns is bypassed). This architecture is the most compact AES architecture and has been used in the literature, see, for instance, [20]. However, it suffers from low throughput. In Fig. 7.4b, the AES unrolled pipelined structure is shown in which the pipeline stages are shown by dotted lines (see, for instance, [17]). As seen in this figure, 10 AES rounds are duplicated, with the last round without the MixColumns transformation. Although this architecture needs 10 AES rounds to be implemented, it allows the designers to use pipelining and hence process multiple inputs sequentially for achieving high throughput. For further increasing the throughput, sub-pipelining of the AES transformations can be used as depicted in Fig. 7.4c.

Sub-pipelining is useful in increasing the working frequency of the AES at the expense of more area used for the pipeline registers. However, it increases the latency of structures.

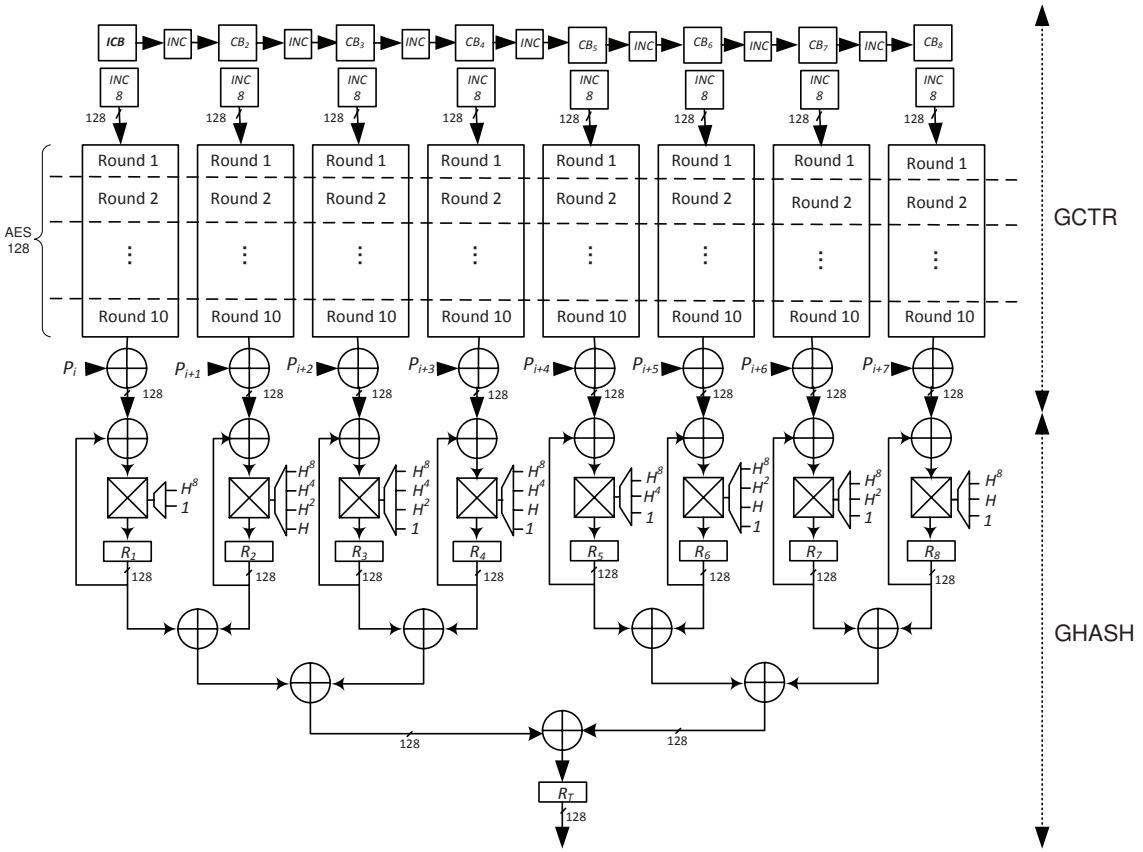


Figure 7.5: The proposed AES-GCM high-performance architecture for $q = 8$.

For instance, the latency of a 3-stage sub-pipelined AES is 3 times more than that of the unrolled pipelined. We also note that if the critical path delay is determined by the multipliers in the GCM architecture, sub-pipelining of the AES transformations cannot increase the working frequency. Although both pipelined and sub-pipelined AES architectures can be utilized, in this chapter, for the syntheses and comparisons, we use pipelined AES architecture presented in Fig. 7.4b. Moreover, for analyzing the effect of sub-pipelining, we have used sub-pipelined AES for two AES-GCM architectures. The details of our implementations are presented later in this section.

According to Table 7.4, we use the most efficient S-box, i.e., the one using polynomial basis (PB) based on (3.18), to reach the AES-GCM with the highest performance. The AES-128 encryption is considered as the block cipher for the GCM and as indicated in Table 7.4, the 10 rounds of the AES-128 are unrolled and pipelined. Moreover, as seen in Table 7.4, we use the proposed Algorithm 1 for the GCM and utilize the parallel method

in Fig. 7.3b for hash subkey exponentiations (hardware optimized through complexity reduction methods in the previous section). Finally, As seen in this table, we use both quadratic and sub-quadratic multipliers presented in Table 7.3.

Fig. 7.5 presents the proposed architecture for the AES-GCM for $q = 8$ parallel structures. The AES-128 pipeline registers are shown by dashed lines in Fig. 7.5. As seen in this figure, 10 clock cycles are needed for obtaining the ciphertext. After these first 10 clock cycles, the results are obtained after each clock cycle. According to Fig. 7.5, 8 parallel AES-128 structures are implemented as part of $GCTR_K$ to provide inputs to $GHASH_H$. As seen in this figure, the function $GCTR_K$ performs the AES counter mode with the *Initial Counter Block (ICB)* and its one-increments (CB_i). Moreover, $q = 8$ increments (using *INC 8* module) and the plaintext blocks (P_i) are used as the inputs. It is assumed that the data is encrypted and the IV in the GCM is 96 bits which is recommended for high throughput implementations [5].

The results of our syntheses for the AES-GCM using the STM 65-nm CMOS technology [74] are presented in Table 7.5. The architectures have been coded in VHDL as the design entry to the Synopsys® Design Vision® [73]. The proposed architectures in this chapter and the ones in [64], [65], [66], [68] and [69] have been synthesized. The syntheses are based on the case for $q = 8$ parallel addition-multiplications using the bit-parallel $GF(2^{128})$ multiplier presented in [94] which has quadratic hardware complexity. For achieving low hardware complexity for the AES-GCM, we have also synthesized six different steps for the Karatsuba-Ofman multipliers. As seen in Table 7.5, areas, power consumptions, and maximum working frequencies are tabulated. From the discussions in this chapter, for n input blocks and q parallel structures, the latency for the architecture in [64], [65] and [66] is n, for the one in [68] and [69] is $\frac{n}{q} + q - 1$, and for our proposed architectures is $\frac{n}{q} + \log_2(q)$. According to these, for different architectures presented in Table 7.5, throughputs and efficiencies are also presented.

As presented in Table 7.5, the sequential approach in [64], [65], and [66] has the lowest hardware complexity compared to other approaches. However, it has the least throughput leading to low-performance hardware implementations. As depicted in Table 7.5, lower areas and power consumptions are achieved for the sub-quadratic hardware complexity $GF(2^{128})$ multipliers used in our proposed architectures compared to the one in [94]. As seen in this table, the maximum working frequency is decreased as we increase

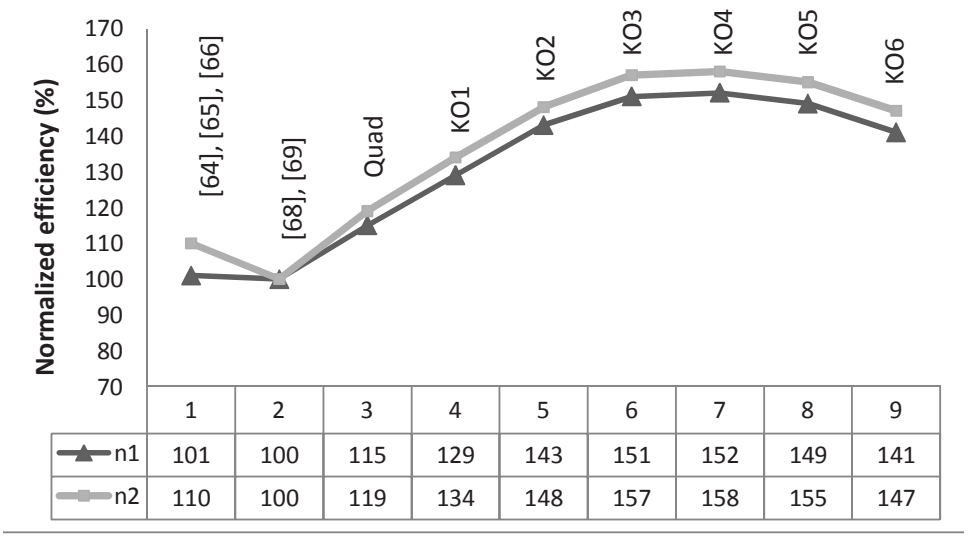
Table 7.5: ASIC synthesis comparisons of the AES-GCM using the STM 65-nm CMOS technology.

Scheme[a]	Total Area [AES][b]		Power	Freq.	Thro.	Eff.
	(mm^2)	K-GE[c]	(mW)	(MHz)	(Gbps)	($\frac{Gbps}{mm^2}$)
[64], [65], [66]	0.23 [0.12]	110 [57]	19.6	568	$\frac{72.7}{n}$	$\frac{316.0}{n}$
[68], [69][d]	1.86 [1.02]	894 [490]	144.3	568	$\frac{72.7}{\frac{n}{8}+7}$	$\frac{39.1}{\frac{n}{8}+7}$
Proposed (quad.)[e]	1.82 [0.92]	875 [442]	142.5	641	$\frac{82.0}{\frac{n}{8}+3}$	$\frac{45.1}{\frac{n}{8}+3}$
Proposed (KO_1)[e]	1.62 [0.92]	779 [442]	124.6	641	$\frac{82.0}{\frac{n}{8}+3}$	$\frac{50.6}{\frac{n}{8}+3}$
Proposed (KO_2)[e]	1.46 [0.92]	702 [442]	113.0	641	$\frac{82.0}{\frac{n}{8}+3}$	$\frac{56.2}{\frac{n}{8}+3}$
Proposed (KO_3)[e]	1.34 [0.92]	644 [442]	104.8	621	$\frac{79.4}{\frac{n}{8}+3}$	$\frac{59.3}{\frac{n}{8}+3}$
Proposed (KO_4)[e]	1.31 [0.92]	630 [442]	101.2	613	$\frac{78.4}{\frac{n}{8}+3}$	$\frac{59.8}{\frac{n}{8}+3}$
Proposed (KO_5)[e]	1.30 [0.92]	625 [442]	102.0	595	$\frac{76.1}{\frac{n}{8}+3}$	$\frac{58.5}{\frac{n}{8}+3}$
Proposed (KO_6)[e]	1.33 [0.92]	639 [442]	105.2	578	$\frac{73.9}{\frac{n}{8}+3}$	$\frac{55.6}{\frac{n}{8}+3}$

[a]For the case of $q = 8$ parallel structures.
[b]The area of the AES is shown inside brackets.
[c]10^3 gate equivalent in terms of two-input NAND.
[d]The better scheme in [68] and [69], in terms of timing and hardware complexities has been synthesized.
[e]The quadratic and six sub-quadratic multipliers used in the proposed AES-GCM.

the number of multiplication steps. However, this trend is not observed for the hardware complexity, i.e., it is decreased up to KO_5 as the optimum value and then rises for KO_6.

The highest throughput is achieved for the proposed architectures in this chapter, i.e., $\frac{82.0}{\frac{n}{8}+3}$ Gbps using quadratic and KO_1/KO_2 sub-quadratic multipliers. As seen in Table 7.5, the highest efficiency is derived for KO_4, i.e., $\frac{59.8}{\frac{n}{8}+3}$ $\frac{Gbps}{mm^2}$. As seen in this table, the working frequencies and throughputs for KO_1 and KO_2 are similar. We have observed that this is because for these two multipliers, the critical path delay is dominated by the AES rounds and not the sub-quadratic multiplier. Inner-round pipelining can be performed to increase the working frequencies of the implementations. Nevertheless, this sub-pipelining increases the area and latency of the AES-GCM architectures. We have performed experiments by sub-pipelining the AES rounds for the architectures using KO_1



	1	2	3	4	5	6	7	8	9
n1	101	100	115	129	143	151	152	149	141
n2	110	100	119	134	148	157	158	155	147

Figure 7.6: Comparison of the efficiencies of nine different AES-GCM architectures for $n_1 = 2^{32} - 2$ and $n_2 = 2^{10}$.

and KO_2 multipliers. This is achieved by adding one pipeline stage after ShiftRows and right before MixColumns. The results of our experiments show no major difference in the maximum working frequency of the design utilizing KO_2 multiplier and increase in its hardware complexity. However, for the architectures using KO_1 multipliers, the working frequency of 689 MHz with the increased area of 1.70 mm^2 is achieved. Therefore, for this architecture which uses KO_1 multipliers, the best speed is obtained compared to the results in Table 7.5. However, its efficiency is obtained as $\frac{51.9}{\frac{n}{8}+3}\ \frac{Gbps}{mm^2}$ which is less than that of the architectures with KO_4 multipliers (see Table 7.5).

For comparing the efficiencies of the schemes presented in Table 7.5, we have presented Fig. 7.6. Based on the derived values for efficiencies in the last column of Table 7.5, two different graphs for two values of n are presented in Fig. 7.6. We consider two different values of n, i.e., $n_1 = 2^{32} - 2$ (the largest encrypted message size allowed) and $n_2 = 2^{10}$. It is noted that considering the normalized efficiency (%) of the scheme in [68], [69] as 100, the relative efficiencies for different architectures are presented in this figure. As seen in Fig. 7.6, for $n_1 = 2^{32} - 2$ and $n_2 = 2^{10}$, KO_4 has the highest efficiencies (51% and 44% more than the sequential method, respectively).

We conclude this chapter by a summary of the work presented. In this chapter, we have obtained optimized building blocks for the AES-GCM to propose efficient and

high-performance architectures. For the AES, through logic-gate minimizations for the inversion in $GF(2^4)$, the areas of the S-boxes have been reduced. We have also evaluated and compared the performance of different S-box architectures using an ASIC 65-nm CMOS technology. Furthermore, through exhaustive searches for the input patterns, we have performed simulation-based average and peak power derivations for different S-boxes to reach more accurate results compared to the statistical power derivation methods.

We have also proposed high-performance and efficient architectures for the GCM. For the case study of $q = 8$ parallel structures in $GHASH_H$, we have performed a hardware complexity reduction technique for the hash subkey exponentiations, having their timing complexities intact. For comparison, the proposed architectures and the previous ones have been synthesized on ASIC. The results show that better efficiencies are achieved for the proposed architectures. Moreover, according to our results, the structures using the four-step Karatsuba-Ofman $GF(2^{128})$ multiplier are the most efficient ones for our proposed architectures. Based on the available resources and performance goals to achieve, one can choose the proposed AES-GCM architectures to fulfill the constraints needed for the required applications.

Chapter 8

Summary and Future Work

8.1 Thesis Summary

IN this thesis, we have proposed reliable and high-performance hardware implementations for the AES-GCM. This includes novel lightweight and concurrent fault detection schemes for the AES for making it reliable and high-performance hardware architectures for the AES-GCM for reaching efficient VLSI implementations. The following summarizes the contributions of this work.

In Chapter 3, which has been presented in [71] and [72], we have evaluated the performance of more than 40 S-boxes utilizing a fixed benchmark platform in 65-nm CMOS technology. To obtain the least-complexity S-box, the formulations for the Galois Field (GF) sub-field inversions in $GF(2^4)$ have been optimized. By conducting exhaustive simulations for the input transitions, we have analyzed the average and peak power consumptions of the AES S-boxes considering the switching activities, gate-level netlists, and parasitic information.

In Chapter 4, which has been presented in [78] and [79], we have proposed a lightweight concurrent fault detection scheme for the AES. In the presented approach, for increasing the error coverage, the predicted parities of the five blocks of the S-box and the inverse S-box have been obtained (three predicted parities for the multiplicative inversion and two for the transformation and affine matrices). Through exhaustive searches among all available composite fields, we have found the optimum solutions for the least overhead parity-based fault detection structures. Moreover, through our error injection simulations for one S-box (resp. inverse S-box), we have shown that the total error coverage of almost 100% (99.998%) for 16 S-boxes (resp. inverse S-boxes) can be achieved. Finally, it is

shown that both the ASIC and FPGA implementations of the fault detection structures using the obtained optimum composite fields, have better hardware and time complexities compared to their counterparts.

In Chapter 5, which has been presented in [83] and [84], we have proposed a concurrent fault detection scheme for the S-box and the inverse S-box based on the low-cost composite field implementations of the S-box and the inverse S-box. We have divided the structures of these operations into three blocks and found the predicted parities of these blocks. We have obtained new formulations for the five predicted parities for three blocks of the S-box and the inverse S-box. To reach high multiple and burst fault detection capabilities, multiple-bit signatures have been obtained within the blocks constituting more area in the structures of the S-box and the inverse S-box. Our simulations have shown that except for the redundant units approach which has the hardware and time overheads of close to 100%, the fault detection capabilities of the proposed scheme for the burst and random multiple faults are higher than the previously reported ones. Finally, through ASIC implementations, it has been shown that for the maximum target frequency, the proposed fault detection S-box and inverse S-box in this chapter have the least areas, critical path delays, and power consumptions compared to their counterparts with similar fault detection capabilities.

In Chapter 6, which has been presented in [92] and [93], we have proposed a structure-independent fault detection scheme for the entire AES encryption and decryption. Specifically, we have obtained new formulations for the fault detection of SubBytes and inverse SubBytes using the relation between the input and the output of the S-box and the inverse S-box. The proposed schemes are independent of the way the S-box and the inverse S-box are constructed. Therefore, they can be used for both the S-boxes and the inverse S-boxes using look-up tables and those utilizing logic gates based on composite fields. Our simulation results have shown very high error coverage for the proposed schemes. Finally, our proposed fault detection schemes and almost all of the previously reported ones have been implemented on FPGAs and their area and delay overheads have been derived and compared. The FPGA implementation results have shown the low area and delay overheads for the proposed fault detection schemes.

Finally, in Chapter 7, which has been presented in [72], we have presented high-speed, parallel hardware architectures for reaching low-latency and high-throughput structures

of the GCM. Having investigated the high-performance $GF(2^{128})$ multiplier architectures, we have benchmarked the proposed AES-GCM architectures using quadratic and sub-quadratic hardware complexity $GF(2^{128})$ multipliers. It has been shown that the performance of the presented AES-GCM architectures outperforms the previously reported ones in the utilized 65-nm CMOS technology.

Based on the above summary, the contributions of this thesis are

- Optimization and benchmarking the AES S-boxes on a fixed hardware platform
- Devising lightweight concurrent exhaustive search-based fault detection schemes for the AES S-boxes using polynomial and normal bases
- Proposing multi-bit signature-based fault diagnosis approaches for the AES S-boxes, inverse S-boxes, and mixed operations
- Presenting structure-independent schemes for fault detection of the entire AES encryption and decryption
- Proposing high-speed, parallel hardware architectures for the GCM

8.2 Future Work

As future works for this thesis, the followings can be pursued.

- The fault detection schemes proposed in this thesis have been evaluated using extensive simulations and benchmarked on ASIC hardware platforms. As a future work for this thesis, our proposed fault diagnosis approaches and corresponding original architectures can be fabricated on chip and actual error injections can be performed. This error injection to the fabricated chip verifies the effectiveness of the proposed fault detection approaches one level beyond the simulation level.
- Another future work for the FPGA platform can be explored noting that the AES is utilized for bitstream security mechanisms. Specifically, the AES decryption is hardware-implemented in many recent FPGAs. Incorporating the proposed hardware countermeasures and evaluating their effectiveness in counteracting internal/malicious faults on FPGAs would be an interesting future research topic.

- As an extension for this thesis, one can integrate reliability into the design of recent cryptographic data authentication algorithms. One can carry out research on developing reliable architectures for the third-round SHA-3 (Secure Hash Algorithm-3) candidates, one of which containing the AES algorithm as its building blocks. This development will be a promising advancement in cryptography research and will result in selecting the final candidates of the ongoing competition to choose the winning function for SHA-3 in 2012.

- Finally, one can work on devising reliable architectures for the recently standardized GCM, which provides data authentication to block ciphers such as the AES. To the best of our knowledge, the aforementioned research on reliability of these architectures will be carried out for the first time.

Bibliography

[1] National Institute of Standards and Technologies, "Announcing the Advanced Encryption Standard (AES)," Federal Information Processing Standards Publication, no. 197, Nov. 2001.

[2] Wi-Fi, http://standards.ieee.org/getieee802/download/802.11-2007.pdf.

[3] WiMAX, http://standards.ieee.org/getieee802/download/802.16e-2005.pdf.

[4] S. Trimberger, "Security in SRAM FPGAs," *IEEE Design and Test of Computers*, vol. 24, no. 6, pp. 581, Nov. 2007.

[5] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," *NIST SP 800-38D*, 2007.

[6] IEEE Standard for Local and Metropolitan Area Networks, Media Access Control (MAC) Security, 2006.

[7] Fibre Channel Security Protocols (FC-SP), 2006. Available: http://www.t10.org/ftp/t11/document.06/06-157v0.pdf.

[8] Algotronics Ltd.: GCM Extension for AES G3 Core, 2007.

[9] Helion Technology: AES-GCM Cores, 2007.

[10] Elliptic Semiconductor Inc.: CLP-15: Ultra-High Throughput AES-GCM Core-40 Gbps, 2008.

[11] E. Käsper and P. Schwabe, "Faster and Timing-Attack Resistant AES-GCM," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '09)*, LNCS 5747, pp. 1-17, 2009.

[12] K. Jankowski and P. Laurent, "Packed AES-GCM Algorithm Suitable for AES/PCLMULQDQ Instructions," *IEEE Trans. Computers*, vol. 60, no. 1, pp. 135-138, Jan. 2011.

[13] S. Morioka and A. Satoh, "An Optimized S-Box Circuit Architecture for Low Power AES Design," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '02)*, pp. 172-186, Aug. 2002.

[14] M. McLoone and J.V. McCanny, "High Performance Single-chip FPGA Rijndael Algorithm Implementations," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '01)*, LNCS 2162, pp. 65-76, 2001.

[15] F. X. Standaert, G. Rouvroy, J. J. Quisquater, and J. D. Legat, "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '03)*, LNCS 2779, Springer, pp. 334-350, Sep. 2003.

[16] P. Bulens, F.-X. Standaert, J.-J. Quisquater, P. Pellegrin, and G. Rouvroy, "Implementation of the AES-128 on Virtex-5 FPGAs," *Proc. Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '08)*, LNCS 5023, pp. 16-26, 2008.

[17] A. Hodjat and I. Verbauwhede, "Area-Throughput Trade-Offs for Fully Pipelined 30 to 70 Gbits/s AES Processors," *IEEE Trans. Computers*, vol. 55, no. 4, pp. 366-372, April 2006.

[18] V. Rijmen, "Efficient Implementation of the Rijndael S-box," Katholieke Universiteit Leuven, Dept. ESAT, Belgium, 2000, available at: http://www.esat.kuleuven.ac.be/ rijmen/rijndael/sbox.pdf.

[19] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, "Efficient Rijndael Encryption Implementation with Composite Field Arithmetic," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '01)*, pp. 171-184, May 2001.

[20] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," *Proc. Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '01)*, pp. 239-254, Dec. 2001.

[21] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC Implementation of the AES SBoxes," *Proc. Cryptographers' Track-RSA '02*, pp. 67-78, Jan. 2002.

[22] X. Zhang and K. K. Parhi, "High-Speed VLSI Architectures for the AES Algorithm," *IEEE Trans. Very Large Scale Integration Systems*, vol. 12, no. 9, pp. 957-967, Sep. 2004.

[23] D. Canright, "A Very Compact S-Box for AES," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '05)*, pp. 441-455, Aug. 2005.

[24] X. Zhang and K. K. Parhi, "On the Optimum Constructions of Composite Field for the AES Algorithm," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 53, no. 10, pp. 1153-1157, Oct. 2006.

[25] S. Nikova, V. Rijmen, and M. Schlaffer, "Using Normal Bases for Compact Hardware Implementations of the AES S-Box," *Proc. Security in Communication Networks '08*, pp. 236-245, 2008.

[26] G. Bertoni, M. Macchetti, and L. Negri, "Power-efficient ASIC Synthesis of Cryptographic Sboxes," *Proc. Great Lakes Symposium on VLSI '04*, pp. 277-281, April 2004.

[27] J. Blömer and J. P. Seifert, "Fault Based Cryptanalysis of the Advanced Encryption Standard (AES)," *Proc. Financial Cryptography '03*, pp. 162-181, Jan. 2003.

[28] G. Piret and J. J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '03)*, pp. 77-88, Sep. 2003.

[29] P. Dusart, G. Letourneux, and O. Vivolo, "Differential Fault Analysis on AES," *Proc. Int'l Conf. Applied Cryptography and Network Security (ACNS '03)*, pp. 293-306, Oct. 2003.

[30] C. Giraud, "DFA on AES," *In Proc. of AES 2004*, pp. 27-41, May 2004.

[31] J. Blömer and V. Krummel, "Fault Based Collision Attacks on AES," *Proc. Int'l Workshop Fault Diagnosis and Tolerance in Cryptography (FDTC '06)*, pp. 106-120, Oct. 2006.

[32] J. Takahashi, T. Fukunaga, and K. Yamakoshi, "DFA Mechanism on the AES Key Schedule," *Proc. Int'l Workshop Fault Diagnosis and Tolerance in Cryptography (FDTC '07)*, pp. 62-72, Sep. 2007.

[33] M. Rivain, "On the Physical Security of Cryptographic Implementations," *Ph.D. thesis*, University of Luxamburg, Sep. 2009.

[34] R. Karri, K. Wu, P. Mishra, and K. Yongkook, "Fault-based Side-Channel Cryptanalysis Tolerant Rijndael Symmetric Block Cipher Architecture," *Proc. IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '01)*, pp. 418-426, Oct. 2001.

[35] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "A Parity Code Based Fault Detection for an Implementation of the Advanced Encryption Standard," *Proc. IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '02)*, pp. 51-59, Nov. 2002.

[36] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard," *IEEE Trans. Computers*, vol. 52, no. 4, pp. 492-505, April 2003.

[37] R. Karri, G. Kuznetsov, and M. Goessel, "Parity-based Concurrent Error Detection of Substitution-Permutation Network Block Ciphers," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '03)*, pp. 113-124, Sep. 2003.

[38] M. Karpovsky, K. J. Kulikowski, and A. Taubin, "Differential Fault Analysis Attack Resistant Architectures for the Advanced Encryption Standard," *Proc. Conf. Smart Card Research and Advanced Applications (CARDIS '04)*, vol. 153, pp. 177-192, Aug. 2004.

[39] K. Wu, R. Karri, G. Kuznetsov, and M. Goessel, "Low Cost Concurrent Error Detection for the Advanced Encryption Standard," *Proc. Int'l Test Conf. '04*, pp. 1242-1248, Oct. 2004.

[40] G. Bertoni, L. Breveglieri, I. Koren, and P. Maistri, "An Efficient Hardware-based Fault Diagnosis Scheme for AES: Performances and Cost," *Proc. IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '04)*, pp. 130-138, Oct. 2004.

[41] L. Breveglieri, I. Koren, and P. Maistri, "Incorporating Error Detection and Online Reconfiguration into a Regular Architecture for the AES," *Proc. IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '05)*, pp. 72-80, Oct. 2005.

[42] C. H. Yen and B. F. Wu, "Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard," *IEEE Trans. Computers*, vol. 55, no. 6, pp. 720-731, June 2006.

[43] T. G. Malkin, F. X. Standaert, and M. Yung, "A Comparative Cost/Security Analysis of Fault Attack Countermeasures," *Proc. Int'l Workshop Fault Diagnosis and Tolerance in Cryptography (FDTC '06)*, pp. 159-172, Oct. 2006.

[44] A. Satoh, T. Sugawara, N. Homma, and T. Aoki, "High-Performance Concurrent Error Detection Scheme for AES Hardware," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '08)*, pp. 100-112, Aug. 2008.

[45] G. Canivet, P. Maistri, R. Leveugle, F. Valette, J. Clediere, M. Renaudin, "Dependability Analysis of a Countermeasure against Fault Attacks by Means of Laser Shots onto a SRAM-based FPGA," *21st IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP)*, pp. 115-122, July 2010.

[46] U. Legat, A. Biasizzoa, and F. Novaka, "A Compact AES Core with On-line Error-detection for FPGA Applications with Modest Hardware Resources," *Journal of Microprocessors and Microsystems*, vol. 35, no. 4, pp. 405-416, June 2011.

[47] M. Medwed and J.-M. Schmidt, "A Continuous Fault Countermeasure for AES Providing a Constant Error Detection Rate," *In Proc. of FDTC 2010*, IEEE Press, pp. 66-71, August 2010.

[48] S. Ghaznavi, "Soft Error Resistant Design of the AES Cipher Using SRAM-based FPGA," *Ph.D. thesis*, University of Waterloo, Ontario, Canada, 2011.

[49] P. Maistri and R. Leveugle, "Double-Data-Rate Computation as a Countermeasure against Fault Analysis," *IEEE Trans. Computers*, vol. 57, no. 11, pp. 1528-1539, Nov. 2008.

[50] C. Moratelli, F. Ghellar, E. Cota, and M. Lubaszewski, "A Fault-Tolerant DFA-Resistant AES Core," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS '08)*, pp. 244-247, May 2008.

[51] G. Di Natale, M. Doulcier, M. L. Flottes, and B. Rouzeyre, "A Reliable Architecture for Parallel Implementations of the Advanced Encryption Standard," *Journal of Elec. Testing*, vol. 25, no. 4, pp. 269-278, Aug. 2009.

[52] S.-Y. Wu and H.-T. Yen, "On the S-Box Architectures with Concurrent Error Detection for the Advanced Encryption Standard," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E89-A, no. 10, pp. 2583-2588, Oct. 2006.

[53] A. E. Cohen, "Architectures for Cryptography Accelerators," PhD dissertation, University of Minnesota, Sep. 2007.

[54] M. Mozaffari Kermani and A. Reyhani-Masoleh, "Parity Prediction of S-box for AES," *In Proc. of the IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2006)*, pp. 2357-2360, May 2006.

[55] M. Mozaffari Kermani and A. Reyhani-Masoleh, "Parity-based Fault Detection Architecture of S-box for Advanced Encryption Standard," *In Proc. of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2006)*, pp. 572-580, Oct. 2006.

[56] M. Mozaffari Kermani, "Fault Detection Schemes for High Performance VLSI Implementations of the Advanced Encryption Standard," *M.E.Sc. Thesis*, Department of Electrical and Computer Engineering, The University of Western Ontario, London, Ontario, Canada, April 2007.

[57] M. Mozaffari Kermani and A. Reyhani-Masoleh, "Fault Detection Structures of the S-boxes and the Inverse S-boxes for the Advanced Encryption Standard," *Journal of Elec. Testing*, vol. 25, no. 4, pp. 225-245, Aug. 2009.

[58] T. Good and M. Benaissa, "692-nW Advanced Encryption Standard (AES) on a 0.13-μm CMOS," *IEEE Trans. VLSI Systems*, vol. 18, no. 12, pp. 1753-1757, Dec. 2010.

[59] S. Tillich, M. Feldhofer, T. Popp, and J. Großschädl, "Area, Delay, and Power Characteristics of Standard-Cell Implementations of the AES S-Box," *J. Sign. Process Syst,* vol. 50, pp. 251-261, 2008.

[60] J. Boyar and R. Peralta, "A New Combinational Logic Minimization Technique with Applications to Cryptology," *In Proc. of SEA 2010,* pp. 178-189, 2010.

[61] S. Nikova, V. Rijmen, and M. Schläffer, "Using Normal Bases for Compact Hardware Implementations of the AES S-Box," *In Proc. of the SCN 2008*, LNCS 5229, Springer, pp. 236-245, 2008.

[62] Y. Nogami, K. Nekado, T. Toyota, N. Hongo, and Y. Morikawa, "Mixed Bases for Efficienct Inversion in $F_{((2^2)^2)^2}$ and Conversion Matrices of SubBytes of AES," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '10)*, LNCS 6225, Springer, pp. 234-247, August 2010.

[63] D. Canright and D.A. Osvik, "A More Compact AES," *In Proc. of SAC 2009*, LNCS 5867, pp. 157-169, 2009.

[64] S. Lemsitzer, J. Wolkerstorfer, N. Felbert, and M. Braendli, "Multi-gigabit GCM-AES Architecture Optimized for FPGAs," *In Proc. of CHES 2007*, LNCS, vol. 4727, pp. 227-238, 2007.

[65] P. Patel, "Parallel Multiplier Designs for the Galois/Counter Mode of Operation," Master of Applied Science Thesis, The University of Waterloo, 2008.

[66] B. Yang, S. Mishra, and R. Karri, "High Speed Architecture for Galois/Counter Mode of Operation (GCM)," *Cryptology ePrint Archive: Report 2005/146*, June 2005.

[67] D. A. McGrew and J. Viega, "The Galois/counter Mode of Operation (GCM)," 2005.

[68] A. Satoh, "High-speed Parallel Hardware Architecture for Galois Counter Mode," *In Proc. of ISCAS*, pp. 1863-1866, 2007.

[69] A. Satoh, T. Sugawara, and T. Aoki, "High-Performance Hardware Architectures for Galois Counter Mode," *IEEE Trans. Computers*, vol. 58, no. 7, pp. 917-930, 2009.

[70] N. Meloni, C. Nègre, and M. A. Hasan, "High Performance GHASH Function for Long Messages," *In Proc. of ACNS 2010*, pp. 154-167, 2010.

[71] M. Mozaffari Kermani and A. Reyhani-Masoleh, "A Low-Cost S-box for the Advanced Encryption Standard Using Normal Basis," *In Proc. of the IEEE International Conference on Electro/Information Technology (EIT 2009)*, pp. 52-55, June 2009.

[72] M. Mozaffari Kermani and A. Reyhani-Masoleh, "Efficient and High-Performance Parallel Hardware Architectures for the AES-GCM," *To appear in IEEE Trans. on Computers*.

[73] Synopsys, http://www.synopsys.com/.

[74] STMicroelectronics, http://www.st.com/.

[75] ModelSim, http://www.model.com/.

[76] Mathworks, http://www.mathworks.com/.

[77] S.-Y. Lin and C.-T. Huang, "A High-Throughput Low-Power AES Cipher for Network Applications," *In Proc. of ASP-DAC 2007*, pp. 595-600, 2007.

[78] M. Mozaffari Kermani and A. Reyhani-Masoleh, "A Lightweight Concurrent Fault Detection Scheme for the AES S-boxes Using Normal Basis," *In Proc. of CHES 2008*, pp. 113-129, Aug. 2008.

[79] M. Mozaffari Kermani and A. Reyhani-Masoleh, "A Lightweight High-Performance Fault Detection Scheme for the Advanced Encryption Standard Using Composite Fields," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 1, pp. 85-91, Jan. 2011.

[80] XILINX, http://xilinx.com.

[81] N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "A Systematic Evaluation of Compact Hardware Implementations for the Rijndael S-box," *In Proc. of CT-RSA*, pp. 323-333, Feb. 2005.

[82] L. Breveglieri, I. Koren, and P. Maistri, "An Operation-Centered Approach to Fault Detection in Symmetric Cryptography Ciphers," *IEEE Trans. Computers*, vol. 56, no. 5, pp. 534-540, May 2007.

[83] M. Mozaffari Kermani and A. Reyhani-Masoleh, "A Low-Power High-Performance Concurrent Fault Detection Approach for the Composite Field S-box and Inverse S-box," *To appear in IEEE Trans. Computers (special issue on Concurrent On-Line Testing and Error/Fault Resilience of Digital Systems).*

[84] M. Mozaffari Kermani and A. Reyhani-Masoleh, "A High-Performance Fault Diagnosis Approach for the AES SubBytes Utilizing Mixed Bases," *To appear in Proc. of FDTC 2011.*

[85] M. Nicolaidis, R. O. Duarte, S. Manich, and J. Figueras, "Fault-Secure Parity Prediction Arithmetic Operators," *IEEE Des. Test*, vol. 14, no. 2, pp. 60-71, 1997.

[86] N. A. Touba and E. J. McCluskey, "Logic Synthesis of Multilevel Circuits with Concurrent Error Detection," *IEEE Trans. CAD*, vol. 16, no. 7, pp. 783-789, 1997.

[87] S. Fenn, M. Goessel, M. Benaissa, and D. Taylor, "On-Line Error Detection for Bit-Serial Multipliers in GF(2^m)," *Journal of Elec. Testing*, vol. 13, pp. 29-40, 1998.

[88] C. Metra, M. Favalli, and B. Ricco, "Novel Implementation for Highly Testable Parity Code Checkers," *Proc. Int'l Workshop On-Line Testing*, pp. 167-171, 1998.

[89] A. Reyhani-Masoleh and M. A. Hasan, "Fault Detection Architectures for Field Multiplication Using Polynomial Bases," *IEEE Trans. Computers, Special Issue on Fault Diagnosis and Tolerance in Cryptography*, vol. 55, no. 9, pp. 1089-1103, Sep. 2006.

[90] G. C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano, "Fault Localization, Error Correction, and Graceful Degradation in Radix 2 Signed Digit-based Adders," *IEEE Trans. Computers*, vol. 55, no. 5, pp. 534-540, 2006.

[91] M. George P. and Alfke, "Linear Feedback Shift Registers in Virtex Devices," *Xilinx Application Note 210*, available at: http://www.xilinx.com/support/documentation/application_notes/ xapp210.pdf.

[92] M. Mozaffari Kermani and A. Reyhani-Masoleh, "Concurrent Structure-Independent Fault Detection Schemes for the Advanced Encryption Standard," *IEEE Trans. Computers, Special Issue on System Level Design of Reliable Architectures*, vol. 59, no. 5, pp. 608-622, May 2010.

[93] M. Mozaffari Kermani and A. Reyhani-Masoleh, "A Structure-independent Approach for Fault Detection Hardware Implementations of the Advanced Encryption Standard," *In Proc. of FDTC 2007*, IEEE Press, pp. 47-53, Sep. 2007.

[94] 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, 2004.

[95] R. Zimmermann and W. Fichtner, "Low-power Logic Styles: CMOS versus Pass-transistor Logic," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 7, pp. 1079-1090, 1997.

[96] C. Moratelli, E. Cota, and M. Lubaszewski, "A Cryptography Core Tolerant to DFA Fault Attacks," *In Proc. of SBCCI 2006*, pp. 190-195, Sep. 2006.

[97] D. E. Knuth, The Art of Computer Programming, vol. 2, "Semi-numerical Algorithms," pp. 441-466, Addison-Wesley, 1981.

[98] R. Lidl and H. Niederreiter, "Introduction to Finite Fields and Their Applications," Cambridge Univ. Press, 1994.

[99] O. Gustafsson and M. Olofsson, "Complexity Reduction of Constant Matrix Computations over the Binary Field," *In Proc. of WAIFI 2007*, pp. 103-115, 2007.

[100] H. Yi, J. Song, S. Park and C. Park, "Parallel CRC Logic Optimization Algorithm for High Speed Communication Systems," *In Proc. of ICCS 2006*, pp. 1-5, 2006.

[101] G. Zhou, H. Michalik, and L. Hinsenkamp, "Improving Throughput of AES-GCM with Pipelined Karatsuba Multipliers on FPGAs," *In Proc. of ARC 2009*, LNCS 5453, pp. 193-203, 2009.

[102] J. Lázaro, A. Astarloa, U. Bidarte, J. Jiménez, and A. Zuloaga, "AES-Galois Counter Mode Encryption/Decryption FPGA Core for Industrial and Residential Gigabit Ethernet Communications," *In Proc. of ARC 2009*, LNCS 5453, pp. 312-317, 2009.

[103] E. D. Mastrovito, "VLSI Architectures for Computation in Galois Fields," *Ph.D. Thesis*, Linköping University, 1991.

[104] A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata," *Soviet Physics Doklady*, vol. 7, pp. 595, 1963.

[105] H. Fan and M. A. Hasan, "A New Approach to Subquadratic Space Complexity Parallel Multipliers for Extended Binary Fields," *IEEE Trans. Computers*, vol. 56, no. 2, pp. 224-233, 2007.

[106] G. Zhou, H. Michalik, and L. Hinsenkamp, "Complexity Analysis and Efficient Implementations of Bit Parallel Finite Field Multipliers Based on Karatsuba-Ofman Algorithm on FPGAs," *IEEE Trans. VLSI Systems*, vol. 18, no. 7, pp. 1057-1066, 2010.

Curriculum Vitae

Name: Mehran Mozaffari Kermani

Post-Secondary Education and Degrees:

The University of Western Ontario
2007 - 2011, Ph.D.

The University of Western Ontario
2005 - 2007, M.E.Sc.

Tehran University
2000 - 2005, B.Sc.

Honours and Awards:

- NSERC Postdoctoral Fellowship (PDF), 2011 - 2013
- Ontario Graduate Scholarship (OGS), 2010 - 2011
- Ontario Graduate Scholarship in Science and Technology (OGSST), 2009 - 2010
- Western Graduate Research Scholarship, 2005 - present
- Western Conference Travel Grants, 2006 - 2007, 2009
- Dean of School of Engineering's highest honor award at Tehran University, 2000

Related Work Experience:

- Senior ASIC/Layout Design Engineer (May 2011 - present)
  AMD, Markham, Ontario

- Graduate Research Assistant (September 2005 - present)
  The University of Western Ontario, London, Ontario

- Graduate Teaching Assistant (September 2005 - present)
  The University of Western Ontario, London, Ontario

- Research Assistant (May 2003 - October 2003)
  Iran Telecommunication Research Center, Tehran, Iran

Publications:

Journal Articles

• M. Mozaffari Kermani and A. Reyhani-Masoleh, "Efficient and High-Performance Parallel Hardware Architectures for the AES-GCM," *To appear in IEEE Transactions on Computers.*

• M. Mozaffari Kermani and A. Reyhani-Masoleh, "A Low-Power High-Performance Concurrent Fault Detection Approach for the Composite Field S-box and Inverse S-box," *To appear in IEEE Transactions on Computers (special issue on Concurrent On-Line Testing and Error/Fault Resilience of Digital Systems).*

• M. Mozaffari Kermani and A. Reyhani-Masoleh, "A Lightweight High-Performance Fault Detection Scheme for the Advanced Encryption Standard Using Composite Fields," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 1, pp. 85-91, Jan. 2011.

• M. Mozaffari Kermani and A. Reyhani-Masoleh, "Concurrent Structure-Independent Fault Detection Schemes for the Advanced Encryption Standard," *IEEE Transactions on Computers, Special Issue on System Level Design of Reliable Architectures*, vol. 59, no. 5, pp. 608-622, May 2010.

• M. Mozaffari Kermani and A. Reyhani-Masoleh, "Fault Detection Structures of the S-boxes and the Inverse S-boxes for the Advanced Encryption Standard," *Journal of Electronic Testing*, vol. 25, no. 4, pp. 225-245, Aug. 2009.

Conference Papers

• M. Mozaffari Kermani and A. Reyhani-Masoleh, "A High-Performance Fault Diagnosis Approach for the AES SubBytes Utilizing Mixed Bases," *To appear in Proc. of FDTC 2011.*

• M. Mozaffari Kermani and A. Reyhani-Masoleh, "A Low-Cost S-box for the Advanced Encryption Standard Using Normal Basis," *In Proc. of EIT 2009*, pp. 52-55, June 2009.

• M. Mozaffari Kermani and A. Reyhani-Masoleh, "A Lightweight Concurrent Fault Detection Scheme for the AES S-boxes Using Normal Basis," *In Proc. of CHES 2008*, pp. 113-129, Aug. 2008. (Blind reviewed-Acceptance ratio: 25%).

• M. Mozaffari Kermani and A. Reyhani-Masoleh, "A Structure-independent Approach for Fault Detection Hardware Implementations of the Advanced Encryption Standard," *In Proc. of FDTC 2007*, IEEE Press, pp. 47-53, Sep. 2007.

• M. Mozaffari Kermani and A. Reyhani-Masoleh, "Parity-based Fault Detection Architecture of S-box for Advanced Encryption Standard," *In Proc. of DFT 2006*, pp. 572-580, Oct. 2006.

• M. Mozaffari Kermani and A. Reyhani-Masoleh, "Parity Prediction of S-box for AES," *In Proc. of CCECE 2006*, pp. 2357-2360, May 2006.