On the Design of Different Concurrent EDC Schemes for S-Box and GF(p)

J. Mathew, H. Rahaman, A. M. Jabir[1], S. P. Mohanty[2] and Dhiraj K. Pradhan
Department of Computer Science, University of Bristol, UK
[1]Dept. of Computing & Electronics, Oxford Brookes University, UK
[2]Dept. of CSE, Univ of North Texas, Denton, TX 76203.

Abstract

Recent studies have shown that an attacker can retrieve confidential information from cryptographic hardware (e.g. the secret key) by introducing internal faults. A secure and reliable implementation of cryptographic algorithms in hardware must be able to detect or correct such malicious attacks. Error detection/correction (EDC), through fault tolerance, could be an effective way to mitigate such fault attacks in cryptographic hardware. To this end, we analyze the area, delay, and power overhead for designing the S-Box, which is one of the main complex blocks in the Advanced Encryption Standard (AES), with error detection and correction capability. We use multiple Parity Predictions (PPs), based on various error correcting codes, to detect and correct errors. Various coding techniques are presented, which include simple parity prediction, split parity codes, Hamming, Hsiao, and LDPC codes. The S-Box, GF(p), and PP circuits are synthesized from the specifications, while the decoding and correction circuits are combined to form the complete designs. The analysis shows a comparison of the different approaches characterized by their error detection capability.

1 Introduction

An important aspect of secure Cryptosystems-on-Chip (CoC) is it's ability to detect and recover from malicious attacks. There are several ways of attacking crypto hardware, that exploit the presence of various side channels in the hardware to extract the secret information. These attacks observe in a non-intrusive way computational timing, power variations or electromagnetic radiation of the device, etc. Attacks based on the Differential Power Analysis (DPA) and Simple Power Analysis (SPA) attacks sense the power consumption of the hardware to extract the secret key. These attacks either directly examine the power traces or carry out statistical operations on the power traces obtained from the hardware, while the cryptographic algorithms are being executed [1]. Recently, it has been shown that attackers may be able to reveal secret keys by introducing internal faults by intrusions [2]. Subsequently, [3] presents a technique showing how the presence of faults in the public parameters of an elliptic curve crypto system (ECC) may expose the secret keys. On chip error masking techniques such as error correction could be one of the options to mitigate the above problems. This has motivated us to consider in this paper the error detection and correction in the S-Box, which is one of the key block in the AES hardware. Although the paper considers only the S-Box design, without loss of generality, the technique can be applied to any logic circuit.

The first step in error correction is error detection. A number of schemes have been proposed in the literature [21] for error detection. Concurrent error detection is a process used to test the operation of a system while it is operating normally [18]. Various techniques are used in this regard, which include hardware duplication, parity codes, time redundancy, etc.

In crypto systems such as the ECC, the arithmetic units, in particular the multipliers over finite fields, comprise the primary components. As such, it is essential to increase the reliability of these components against fault injection and other forms of attacks. Various schemes have been proposed in the literature, which consider concurrent error detection for the finite field multiplications. In [6], a parity-based approach is used to detect errors in the bit-serial polynomial and normal basis multipliers. A similar technique is used in [13] for the bit-serial and bit-parallel polynomial basis multipliers. This scheme is extended to a multi-bit parity approach in the same paper for error detection in the bit-serial and bit-parallel polynomial basis multipliers. Based on interlacing parity codes, another approach is proposed in [19] for bit-parallel polynomial basis multipliers. In addition to the parity based approaches, time redundancy is also used for error detection in finite field multiplications. The technique of [6] considered the detection of single stuck at faults in polynomial basis Galois Field multipliers. A simple parity prediction technique was used for error detection.

Another approach is to scale the inputs of the multipliers by a factor and at the end of the multiplication, and then the correctness of the results is checked by one or two divisions [14].

Motivation and Contribution To date, analysis of the various error detection methods at different redundancy levels has not been addressed in light of cryptographic hardware defined over finite fields. Design choices for a secure implementation affect performance and overhead in terms of silicon area. There is usually a trade-off between security versus cost of implementation—the more error detection capability we want, the more we pay in terms of addition redundancy and reduced performance.

The main objective of this paper is the comparison of different techniques based on PP. Since the PP technique does not include any encoding and decoding of the input operands, and the PP circuits run in parallel with the functional parts of the circuits, the delay penalty is only due to the output parity generation, and the syndrome decoding and correction logic. We also investigate the PP hardware overhead by varying the number of parity bits with different hamming distances.

The rest of this paper is organized as follows. In Section 2, we review the basics of S-Box and other background material relevant to this paper. In Section 3, we present the different approaches for error detection and correction. Section 4 presents the experimental results. Finally, we conclude in Section 5.

2 Background

Let GF(p^m) denote a set of p^m elements, where p is a prime number and m a non zero positive integer, with two special elements 0 and 1 representing the additive and multiplicative identities respectively. In addition let the two symbols '$+$' and '$\bullet$' represent addition and multiplication operations respectively. Then GF(p^m) forms a finite field (also known as *Galois field*), if it forms a commutative ring with identity over these two operators in which every element has a multiplicative inverse.

AES and S-Box The S-Box is the most critical component for controlling the size and speed in both hardware and software implementations of the AES algorithm. The S-Boxes of the Data Encryption Standard (DES), AES, and Camellia have similar structures, consisting of multiplicative inversion on a Galois field and affine transformations. However, the inversion is the most computationally intensive operation. In this paper, we focus on error detection in the AES S-Box. The SubByte and Inv_SubByte operate on a state by substituting every byte in the state with a byte in a substitution table, which we generally refer to as the S-Box. There are two transformations in the S-Box, which first calculates the multiplicative inverse in the finite field over GF(2^8), and second applies the affine transformation over GF(2). The S-Box is invertible and is used for inverse SubByte transformation. Besides the implementation of the S-Boxes of the SubBytes transformation, the implementations of the other transformations of the AES algorithm, i.e. ShiftRows, MixColumns, AddRoundKey, are straight forward and relatively simple. There are S-Boxes with parity checking [22]. The area overhead and the fault coverage is very dependent on the concrete implementations of the S-Boxes with parity checking. The AES encrypts 128-bit plaintext data blocks by using a 128-bit user key. The decryption takes 10 rounds using the last-round key to recover the plaintext, that was originally encrypted using 10 rounds. A detailed explanation can be found in [10].

2.1 Synthesis and Optimization

We consider the technique for synthesis and optimization of the multiple-output, multivariate polynomials over GF(2^m) based on [7]. The circuits with and without the error correction schemes have been represented in terms of these polynomials, which we have synthesized with this technique. The polynomials are represented as the Shared Galois Polynomial Decision Diagrams (SGPDDs) [8]. For example, Fig. 1(a) shows the SGPDD representation of the polynomial $f(a,b,c) = a + \beta bc^3 + \beta a^2c^3$ over GF(4), where $\{\alpha,\beta\} \in$ GF(4).

If the initial specification is not over finite fields, e.g. over Boolean or MIN-MAX post algebra, then the technique of [7] is applied for computing the coefficients of the polynomials and storing them as the SGPDDs.

Once the SGPDDs are obtained, the circuits are synthesized by decomposing and factoring the SGPDDs based on finding *cuts* within the SGPDDs. A cut is a partitioning of the nodes in the SGPDD into two sets T and B, where T contains internal nodes and the root and B contains external, internal, and the last nodes, i.e. internal nodes which have the external nodes as their children. Effectively a cut can factorize an SGPDD realizing a function f over GF(2^m) as $f = D \cdot Q + R$, i.e. it can perform both *multiplicative* and *additive* decompositions, depending on how the edges are reconnected about the nodes which are the subjects of the cuts. Cut based algorithms have been used for synthesis in the Boolean domain, e.g. [9]. In this paper we quickly factorize a polynomial over GF(2^m) based on cuts on their SGPDDs to construct an expression DAG based multiple output shared netlist. The netlist constitutes two types of nodes: internal nodes which can either be GF(2^m) adders or multipliers, or external nodes which can only be constants and variables over GF(2^m). The internal nodes can have

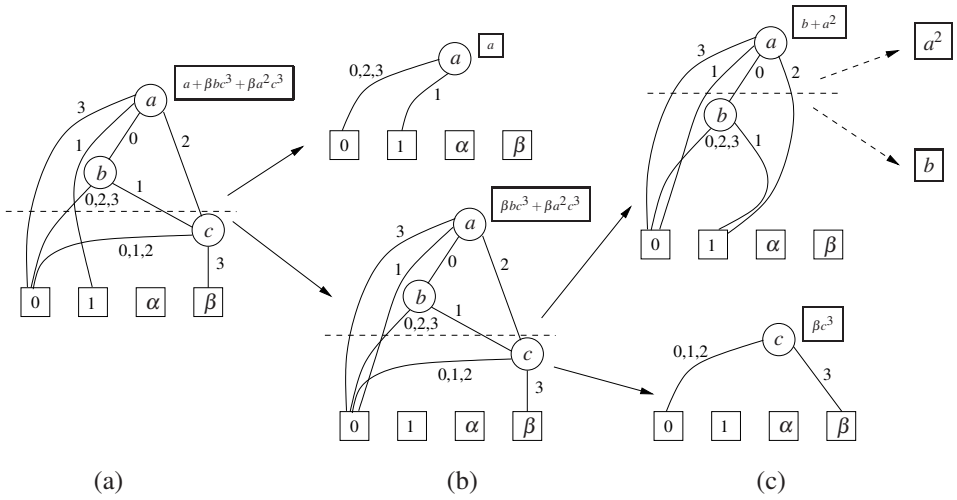

Figure 1. Decomposition—an example.

two children. The netlists are further synthesized based on additional factorization and optimization. Fig. 1 shows an example. The cuts are shown with horizontal broken lines. Fig. 1(a) performs an additive decomposition, Fig. 1(b) performs a multiplicative decomposition, and finally, Fig. 1(c) performs another additive decomposition to obtain the final result $((a)+((\beta c^3)\times((a^2)+(b))))$, which is added to the netlist. This requires one multiplier and two adders over GF(2^m).

Once the netlists are obtained, they are structurally factorized as shown in [7]. For example, given the netlist for $Z=((AX+Y)+BX)$, first node X in the netlist is determined as factorizeable, and then it is factorized by restructuring the netlist as $Z=((AX+BX)+Y)=(X(A+B)+Y)$. The netlists are also optimized by an efficient optimal algorithm based on the properties of finite fields as shown in [8].

Definition 1 *The parity check matrix* $\mathbf{H}$ *of a code consists of all the non-zero r-tuples as its columns. In the systematic form, the columns of* $\mathbf{H}$ *are arranged as* $\mathbf{H}=[\mathbf{H}_s\mathbf{H}_p]$*, where* $\mathbf{H}_s$ *is the systematic part and* $\mathbf{H}_p$ *the parity part.*

2.2 Fault Model

The type of faults that we consider are the permanent stuck-at-1 and stuck-at-0 faults. Furthermore, the proposed technique assumes faults due to transients as well as induced faults. A fault in a network is said to be masked if the occurrence of the fault does not alter the function. In this paper we shall focus on developing a design technique for detecting single/multiple error. It is important to note that when used in error correction mode it is able to detect and correct faults in the parity prediction logic as well.

3 Description of Error Detection and Correction

In this section, we present different error detection schemes. The analysis is carried out in the following three cases: First, we analyze a case where only error detection is considered. Second, single error correction at the output bits is explored. Finally, multiple error correction using multiple codes is investigated.

We start with a simple parity prediction scheme. Then we extend the scheme with multiple parity bits which increases the over all error detection capability. The basic structure of the concurrent error detection and correction scheme is shown in Fig. 2. The classical S-Box structure is synthesized from the specification which is described in Section 2. Next, we present a general algorithm for designing the proposed scheme with an example parity check matrix.

Let $\vec{c}^{\,1}=[c_0^1,c_1^1,c_2^1,\ldots,c_{m-1}^1]^T$ be the output of the functional block and $\vec{c}=[c_0,c_1,c_2,\ldots,c_{m-1}]^T$ the corrected output. Also let r be the number of parity bits, and $\vec{p}=[p_0,p_1,\ldots,p_{r-1}]^T$ and $\vec{p}^{\,1}=[p_0^1,p_1^1,\ldots,p_{r-1}^1]^T$ respectively be the predicted and the parity bits generated from the output bits. Let $\mathbf{H}$ be the parity check matrix associated with the proposed single error detection and correction scheme.

Design Procedure:

- Determine the number of parity bits (r) required.
- Construct the $\mathbf{H}$ matrix, with $(m+r)$ non-zero r-bit column vectors. The dimension of the resulting matrix is $r\times(m+r)$.
- A column vector with a single 1 is assigned to parity P_i.

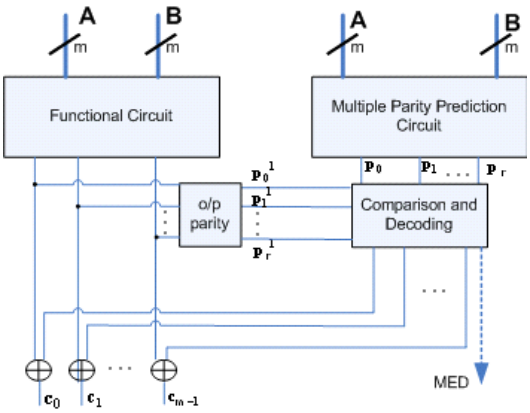

Figure 2. Function Circuit with Concurrent Error Detection and Correction

- The column vector with all 1s is assigned to output bit c_{m-1}.
- The remaining m columns are assigned the output bits c_i, without any constraints.
- Generate predicted parity expressions in terms of c_is. Next, generate the predicted output parities from the inputs.
- For Hsiao code, choose the parity check matrix such that the output bits are assigned to the columns with odd number of ones. In this case additional parity bits maybe required.
- Finally, combine the multiplier, PP, output encoder, decoder, and the correction logic as shown in Fig. 2.

The following illustrates the various error correcting codes considered.

Single Bit Parity Prediction (SBPP) Single bit Parity codes are conceptually the simplest codes for concurrent error detection. For the functional outputs $[c_0, c_1, c_2, \ldots, c_{m-1}]$, the single check bit is $P_p = [c_0 \oplus c_1 \oplus c_2 \oplus \ldots, \oplus c_{m-1}]$.

Example 1 *Consider for example the single bit parity prediction in the S-Box design. Here we have* $m = 8$. *We have the following* **H** *matrix*

$$\mathbf{H} = \begin{array}{cccccccccc} p_p & c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}.$$

Therefore, the parity check equation is: $P_p = c_0 + c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7$.

Split Parity Code A split-parity code is defined in [16]. In its simplest form it is a non-linear systematic code employing NAND/NOR gates. It has two check bits P_1 and P_2 where the EXOR-sum of these check bits equals the parity of the information bits of the code. The two check bits P_1 and P_2 are computed from the m functional information bits $[c_0, c_1, c_2, \ldots, c_{m-1}]$, as $P_1 = \overline{c_1 \vee c_2} \oplus \overline{c_3 \vee c_4} \oplus \ldots, \oplus \overline{c_{m-2} \vee c_{m-1}}$, and $P_2 = \overline{c_1 \wedge c_2} \oplus \overline{c_3 \wedge c_4} \oplus \ldots, \oplus \overline{c_{m-2} \wedge c_{m-1}}$. The linear parity function $P(m)$ is split into the two non-linear functions, P_1 and P_2. $P(m)$ is computed by $P_1 \oplus P_2$.

Hamming Code Hamming codes are the simplest of a group of codes known as the linear block codes [4]. The advantage of the Hamming codes is that the number of parity bits grows logarithmically with the number of output bits. In the proposed approach, we consider parity prediction based encoders. For memory based Hamming encoders, the bits are encoded with a tree of EXOR operations. The sizes of the parity prediction circuits depend on the number of input bits.

Example 2 *Consider the multiplier structure over* $GF(13)$. *Here we have* $m = 4$. *Therefore, we need 3 parity bits to correct single errors or detect two errors. We have,*

$$\mathbf{H} = \begin{array}{ccccccc} p_0 & p_1 & p_2 & c_0 & c_1 & c_2 & c_3 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}.$$

Therefore, the parity check equations are: $p_0 = c_0 + c_1 + c_3; p_1 = c_0 + c_2 + c_3; p_2 = c_1 + c_2 + c_3$.

Low Density Parity Check (LDPC) codes LDPC codes have received much attention because of their excellent performance and large degree of parallelism [17]. The advantage of the LDPC code is that it has reduced decoding complexity. In Hamming code based correction schemes the complexity of the encoding and decoding logic grows linearly with the number of data bits. The principal difference between the LDPC code applied to other applications and our approach is that, in our approach instead of encoders we have parity prediction circuits.

A single error correction (SEC)/double error detection (DED) code has a minimum Hamming distance of 4. It can be used in single error correction and double error detection mode or in multiple error detection mode (three errors). One such code is the Hsiao code [11]. It uses multiple parity bits to uniquely identify which bit(s) in a code word is erroneous and defines the error-free condition. Since the speed of error detection and correction in a code is dependent upon the associated check bits, the Hsiao code has high error correction overhead.

Example 3 *Consider the multiplier structure over GF(13) constructed in Example 2. The parity check matrix* **H** *that satisfies the three error detection condition is*

$$\mathbf{H} = \begin{array}{cccccccc} p_0 & p_1 & p_2 & p_3 & c_0 & c_1 & c_2 & c_3 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}.$$

Therefore, the parity check equations are $p_0 = c_0 + c_2 + c_3; p_1 = c_0 + c_1 + c_3; p_2 = c_0 + c_1 + c_2; p_3 = c_1 + c_2 + c_3$. *The final predicted parity bits for this case are calculated based on the above equation.*

Error Detection and Correction When there is mismatch between the output parity bits and predicted output parity bits, we can conclude that an error has occurred. When it is used in error correction mode using the computed syndrome, we can distinguish errors in the functional block and the PP block. Comparing the output parities and the PP bits gives a unique syndrome that shows error in the functional block or the PP block. This syndrome is decoded to identify which bit is in error and the final EXOR operation is used to correct the erroneous bit. The basic structure of the concurrent error detection and correction scheme is shown in Fig. 2. The output parity computation is merely deriving the basic parity check equation from the parity check matrix. The same equation is used for generating the predicted parity bits.

Area Overhead In the proposed PP we need to generate the output parity bits from the input operands. Apart from this, we have the decoding and correction circuitry. Hence, the total hardware overhead is greater than that of the functional block hardware. The delay overhead is contributed by the EXOR gates in the output parity generator plus one EXOR gate delay for the correction, together with the decoder delay. As we can see, in the above structural approach we did not optimize the overall hardware requirement. Instead, we employ a synthesis tool, specifically targeted for the functional blocks, to optimized the hardware. We synthesized the S-Box/GF(p) logic and PP logic separately. The synthesis and optimization technique is briefly presented in the preliminaries.

Comparing Delay Overhead The delay of the disjoint parity prediction circuit is less than that of the main functional block in the case of S-Box. Apart from the delay in the PP block, additional EXOR gates are required to implement the parity check equations, which depend on Hamming weight of the rows of **H**. The delay overhead is contributed by the output parity generator, syndrome decoding, plus one EXOR gate delay for the correction. Table 2 shows the delay comparison of the various cases. As we can see, the delay penalty is minimum in the LDPC based designs, which is mainly due to sparse matrix structure of the LDPC codes. Moreover, only a single AND gate is required for syndrome decoding.

4 Experimental Results

Various functional blocks (S-Box and GF(p) arithmetic circuits) with error detection and correction techniques such as SPBB, Split Parity, Hamming, LDPC and Hsiao codes have been designed. The analysis presented here are for the S-Box and the different finite fields over GF(p), where $11 \leq p \leq 97$. However, the technique can be easily extended to higher order fields. The designs were synthesized using [8] and technology mapped with the Synopsys tools using the 0.09 micron CMOS TSMC technology library. Synopsys's PrimePower^{TM} tool was used to estimate the power consumption.

All the codes presented in this paper have been implemented in Gnu C++ 3.2.2-5 on a computer with 2GB RAM and a 2.4GHz Athlon processor running RedHat Linux with kernel-2.4.20-43.9. The functional circuit and parity prediction circuits were stored as two-level AND-OR PLAs to enable us to determine how effective the coding technique is in optimizing area, power, and delay.

Table 1. Broad Comparison of Various Error Detection Schemes

Scheme	S-Box (area,delay,power) (μm^2, ns, mw)	PP block (area,delay,power) (μm^2, ns, mw)	% area Overhead
Simple parity	(5699.6, 2.75, 5.847)	(929.0, 1.13, 3.96)	14.02%
Split Parity	(5699.6, 2.75, 5.847)	(1806.3, 1.64,3.47)	24.06%
Hamming Code	(5699.6, 2.75, 5.847)	(3057.9, 2.21, 3.29)	34.91%
LDPC	(5699.6, 2.75, 5.847)	(3838.5, 2.26, 4.01)	40.24%
Hsiao	(5699.6, 2.75, 5.847)	(3941.7, 2.27, 4.2)	40.88%

Table 2. Delay Comparison of Various Error Detection Schemes

Scheme	parity bits	# of Errors	Delay (detection)	Delay (correction)
Simple parity	1	all odd	$3t_{xor}$	NA
Split Parity	2	all odd 50% even	$3t_{xor} + t_{and}$	NA
Hamming Code	4	2	$3t_{xor}$	$5t_{xor} + 2t_{and} + t_{inv}$
LDPC	5	2	$4t_{xor}$	$5t_{xor} + 2t_{and}$
Hsiao	5	3	$4t_{xor}$	$5t_{xor} + 2t_{and}$

We have minimized multipliers and adders over GF(p) ($11 \leq p \leq 97$) for all the prime fields. Further, the AES S-Box is also designed using above technique. In Table 3,the column with the heading "GF(p) adder" shows the result of

Table 3. GF(p) adder vs Parity Prediction .

Prim GF(p)	GF(p) Adder (area,delay,power) (μm^2, ns, mw)	Single Bit Parity Prediction (area,delay,power) (μm^2, ns, mw)
GF(11)	(1677.30, 1.51, 1.96)	(638.6, 1.06, 0.68)
GF(13)	(2809.5, 1.81, 3.32)	(793.49, 1.14, 0.92)
GF(17)	(5977.03, 3.19, 6.05)	(1206.37, 1.54, 1.92)
GF(19)	(4461.01, 2.47, 4.66)	(1586.98, 1.77, 1.62)
GF(23)	(5254.51, 2.45, 5.94)	(1683.76, 1.96, 2.01)
GF(29)	(6180.25, 2.91, 8.19)	(2680.46, 2.41, 3.04)
GF(31)	(3406.24, 2.19, 4.49)	(2683.69, 3.00, 1.99)
GF(37)	(24711.371, 5.50, 25.52)	(4690.01, 2.79, 4.87)
GF(41)	(25217.80, 4.29, 15.16)	(4980.31, 3.04, 4.82)
GF(43)	(19989.08, 5.25, 26.93)	(5412.55, 3.09, 5.22)
GF(47)	(23756.59, 3.21, 10.34)	(5460.93, 2.88, 5.10)
GF(59)	(15279.71, 3.41, 19.55)	(9141.34, 3.77, 8.94)
GF(61)	(14224.94, 4.07, 18.57)	(9221.97, 3.88, 9.64)
GF(67)	(52622.36, 10.76, 64.88)	(11118.61, 4.77, 11.28)
GF(71)	(4572.07, 6.12, 35.61)	(14050.70, 4.93, 13.63)
GF(73)	(63457.37, 11.72, 79.20)	(14647.43, 4.74,13.96)
GF(79)	(21437.40, 3.96, 21.40)	(16850.52, 5.24, 14.58)
GF(83)	(51174.26, 6.82, 56.18)	(19021.34, 5.87, 18.0)
GF(89)	(59922.14, 11.90, 61.96)	(21930.81, 5.40, 19.7)
GF(97)	(57612.64, 6.99, 61.81)	(21605.03, 5.89,20.07)

Table 4. GF(p) Multiplier vs. Parity Prediction.

Prim GF(p)	GF(p) Multiplier (area,delay,power) (μm^2, ns, mw)	Single Bit Parity Prediction (area,delay,power) (μm^2, ns, mw)
GF(11)	(1609.57, 1.36, 1.89)	(603.18, 0.99, 0.536)
GF(13)	(1919.22, 1.40, 2.04)	(616.09, 1.14, 0.665)
GF(17)	(3419.12, 2.27, 3.68)	(1254.75, 1.47, 1.41)
GF(19)	(4602.91, 2.73, 4.28)	(1545.06, 1.89, 1.70)
GF(23)	(5947.99, 2.84, 5.57)	(2177.27, 1.83, 2.13)
GF(29)	(6180.25, 2.91, 8.19)	(3219.14, 2.86, 3.19)
GF(31)	(9583.25, 3.80, 8.86)	(1490.22, 2.13, 1.90)
GF(37)	(16276.35, 4.77, 14.20)	(4602.92, 2.69, 3.51)
GF(41)	(19756.76, 4.98, 14.59)	(4951.28, 2.98, 3.60)
GF(43)	(21843.73, 5.22, 15.62)	(6167.34, 3.45, 5.35)
GF(47)	(24253.26, 6.03, 17.42)	(6225.40, 3.46, 4.40)
GF(59)	(37565.26, 6.18, 28.97)	(9302.63, 4.15, 8.30)
GF(61)	(40832.80, 6.22, 31.04)	(9941.29, 4.28, 9.12)
GF(67)	(40703.76, 6.05, 31.73)	(12476.60,4.21, 10.95)
GF(71)	(56231.86, 9.40, 33.82)	(14111.97,4.51, 11.04)
GF(73)	(63540.93, 10.54, 47.55)	(13741.02,5.24, 11.01)
GF(79)	(81549.51, 10.54, 68.44)	(16647.31,5.10, 11.85)
GF(83)	(90335.94, 11.18, 59.03)	(18553.61,4.98, 13.52)
GF(89)	(106212.2, 10.98, 68.85)	(19156.77,4.90, 13.85)
GF(97)	(117379.2, 13.02, 68.88)	(23679.10,5.88, 18.36)

synthesis of the GF adders first, and then applying the Synopsys compiler on the resulting VHDL files. Here, area, delay, and power are in $10^{-6}mm^2$, nano seconds, and mW respectively. Power was estimated at 1.8V. The second component in the Table 3 shows the area, delay, and power of the single bit parity prediction component. It can be noted that the area of single bit parity prediction logic is about 16% on an average. The tables for the other higher order parity bits are not shown here for brevity.

However, Fig. 3–5 show the comparison between the multipliers and PP logic for the various cases considered. Mostly, the area of the PP logic is less than that of the multipliers. On an average, for the designs considered here are based on the proposed technique, the total area overhead varies depending on the type of code. Also it depends on the Hamming weight of the parity check matrix used. Due to this reason the critical path delay also varies, although in most of the cases the critical path delay is less than that of the multiplier. The delay penalty also depends on the parity check matrix, because the number of EXOR stages depends on the number of ones in parity check equation. The power overhead trend is also similar to that of the area. When it is used in error correction mode, compared to the traditional techniques such as the Triple Modular Redundancy (TMR), which is associated with an overhead of more than 200%, the error correction code based technique is much better. Fig. 6 shows the overhead analysis for the adders over GF(p). In general, the trend is similar. However, the growth is not linear due to the sharing of the common subexpressions in the parity prediction logic.

4.1 Multiple Error Correction

The technique presented so far can only detect and correct single bit errors in the output bit due to single internal fault or detect multiple errors. However, an internal fault may cause multiple bit errors at the output. One of the possible solutions is to separate the output bits in such a way that they are corrected by different Hamming codes, i.e. we use multiple Hamming codes to correct multiple errors. In other words, one could use a combination of error correcting codes to correct multiple errors. Our analysis shows that the complexity of parity prediction logic becomes slightly more complex when multiple codes are used. However, the delay complexity can be reduced by splitting the whole code into smaller words.

4.2 Error Correction in Random Logic

Error correction in random logic is also important when it comes to fault tolerant designs. Therefore, the above techniques are also applied to random logic. Table 5 shows the synthesis results for the various benchmark circuits. The analysis shows that the overhead varies depending on the logic functionality. It is also found out that the critical path in the parity prediction logic is less than that of the functional circuits. Therefore, the delay overhead is due to output parity generation and error correction logic. The table shows the synthesized area of the original and the single error correction using Hamming codes in μm^2.

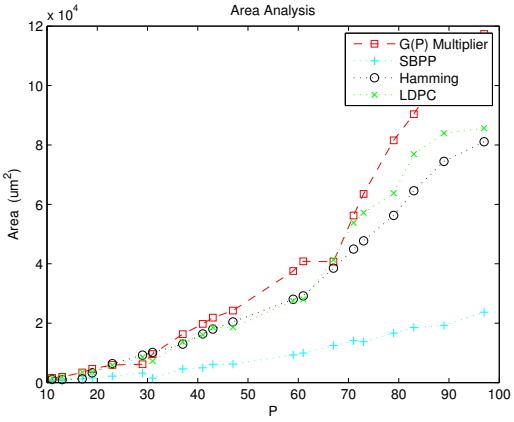

Figure 3. Area Analysis: GF(p) Multiplier,SBPP, LDPC and Hamming

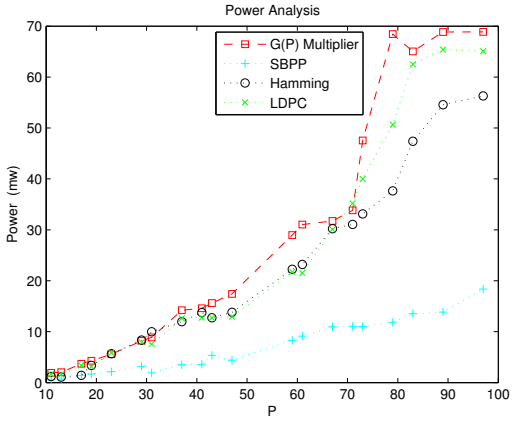

Figure 5. Power Analysis: GF(p) Multiplier,SBPP, LDPC and Hamming

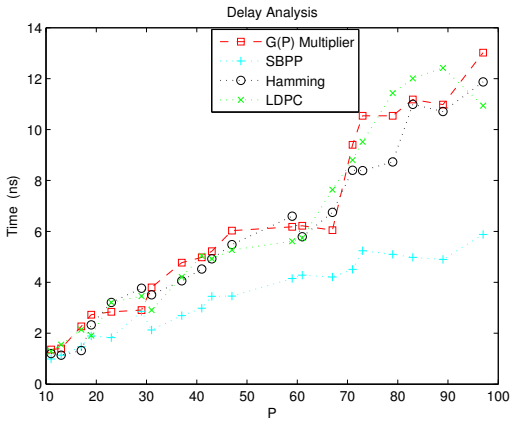

Figure 4. Delay Analysis: GF(p) Multiplier,SBPP, LDPC and Hamming

Table 5. Area Analysis for Error Correction in Random Logic.

Bench mark	# of Inputs	# of Outputs	Functional Logic	SEC Version	% Overhead
apex1	45	45	8799.39	14473.19	164.4
apex3	54	50	9138.09	15615.07	170.8
apex4	9	19	12947.53	17760.11	137.1
apex5	117	88	5202.87	10773.45	207.0
cordic	23	2	522.54	1099.91	210.4
cpa	24	109	6547.92	13092.62	199.9
e64	65	65	1474	3057.8	207.4
ex5p	128	28	2354.67	4760.95	202.1
misex1	8	7	354.80	709.6	201.0
misex2	25	18	616.08	1267.64	205.7
misex3	14	14	4899.66	10592.82	216.1
misex3c	14	14	3209.46	5225.45	162.8
seq	41	35	9502.57	12192.7	128.3
table3	14	14	6686.62	13270.02	198.4

5 Conclusions

The paper has aimed at comparing the performance of different error detection and correction techniques, which are used to mitigate malicious attacks in crypto hardware. We presented an overhead analysis for designing S-Box and GF(p) arithmetic blocks. We used a heuristic gate as well as word-level synthesis and optimization technique for the analysis. Moreover, with regards to several performance index parameters, such as area, delay, and power a large set of experimental circuits has been designed. In conclusion, what clearly comes out from the experiments is, as evident, there is a linear increase in overhead as the number of error detection features increases. The performance figures also closely match those of the structural technique.

References

[1] G. B. Ratanpal, R. D. Williams,and T. N. Blalock, "An On-Chip Signal Suppression Countermeasure to Power Analysis Attacks", *IEEE Trans. on dependable and secure computing*, vol. 1, no. 3, July 2004.

[2] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations", *Journal of Cryptology*, 14, 2001, pp. 101–119.

[3] M. Ciet and M. Joye. Dueck, "Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults", *Designs, Codes and Cryptography*, 36(1), July 2005, pp. 33–43.

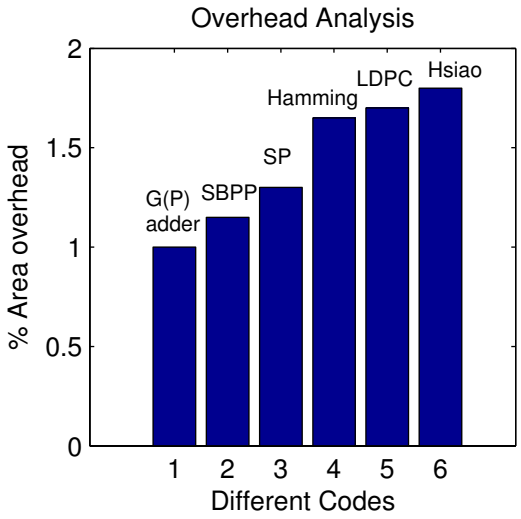

Figure 6. Area Overhead Analysis for GF(p) Adder

[4] W. Hamming, "Error Detecting and Error Correcting Codes", *Bell Systems Tech. Journal*, pp. 147–160, 1950.

[5] Mitra S., Seifert N., Zhang M., Shi Q and Kim K, "Robust System Design with Built-In Soft Error Resilience", *IEEE Computers*, Vol. 38, Number 2, pp. 43–52, Feb. 2005.

[6] A. Reyhani-Masoleh, and M. Anwar Hasan, "Fault Detection Architectures for Field Multiplication Using Polynomial Bases", *IEEE Trans. Computers*, vol. 55, No. 9, Sept. 2006.

[7] A. Jabir and D. Pradhan, "A Graph-Based Unified Technique for Computing and Representing Coefficients Over Finite Fields", *IEEE Trans. Comp.*, 56(8):1119–1132, Aug. 2007.

[8] A. Jabir, D. Pradhan, J. Mathew, "GfXpress: A Technique for Synthesis and Optimization of GF(2^m) Polynomials", *IEEE Trans. CAD*, 27(4):690–711, Apr. 2008.

[9] C. Wang, V. Singal, and M. Ciesielski, "BDD Decomposition for Efficient Logic Synthesis", *Int. Conf. Comput. Aided Design (ICCAD)*, pp. 626–631, 1999.

[10] J. Daemen and V. Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard", Springer, 2002.

[11] M.Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes", *IBM J. Res. Dev.*, 14, pp. 395–401, 1970.

[12] A. Reyhani-Masoleh and M. A. Hasan, "Towards fault-tolerant cryptographic computations over finite fields", *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 3, Nos. 3, pp. 573–613, August 2004.

[13] S. Bayat-Sarmadi and M. A. Hasan, "On concurrent detection of errors in polynomial basis multiplication", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 15, No. 4, pp. 413–426, April 2007.

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

[15] A. Reyhani-Masoleh, and M. Anwar Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over GF(2^m)", *IEEE Trans. on Computers*, Vol.53, No.8, pp.945–959, August 2004.

[16] M. Goessel and E. Sogomonyan, "Non-linear Split Error Detection Code", *Fundamenta Informaticae*, vol. 83, pp. 109–115, 2008.

[17] R. Gallager, "Low-Density Parity-Check Codes", MIT press, Cambridge,MA, 1963.

[18] S. Mitra and E. McCluskey, "Which Concurrent Error Detection Scheme to Choose?", *Proc. Test Conference*, pp. 985–994, 2000

[19] W. Chelton and M. Benaissa, "Concurrent Error Detection in GF(2^m) Multiplication and its Application in Elliptic Curve Cryptography", *IET Circuits, Devices & Systems*, vol. 2, no. 3, pp. 289–297, 2008.

[20] D. K. Pradhan, "A Theory of Galois Switching Functions", *IEEE Trans. Computers*, vol. 27, no. 3, pp.239–248, Mar. 1978.

[21] S. Fenn, M Gossel, M Benaissa, and D. Taylor, "Online Error Detection for Bit-seial Multipliers in GF(2^m)", *J. Electronic Testing: Theory and Applications*, vol. 13, pp.29–40, 1998.

[22] K. Wu, R. Karri, G. Kuznetsov and M. Goessel, "Low Cost Concurrent Error Detection for the Advanced Encryption Standard", *Proc. International Test Conference*, pp. 1242–1248, 2004.