A COMPACT ASIC IMPLEMENTATION OF THE ADVANCED ENCRYPTION STANDARD WITH CONCURRENT ERROR DETECTION

Namin Yu and Howard M. Heys
Electrical and Computer Engineering
Memorial University of Newfoundland
St. John's, Newfoundland, Canada
{namin, howard}@engr.mun.ca

ABSTRACT
In this paper, we investigate the application of concurrent error detection circuitry to a compact application-specific integrated circuit (ASIC) implementation of the Advanced Encryption Standard (AES). The specific objective of the design is to develop a method suitable for compact ASIC implementations targeted to embedded systems such that the system is resistant to fault attacks. To provide the error detection, recognizing that previously proposed schemes are not well suited to compact implementations, it is proposed to adopt a hybrid approach consisting of parity codes in combination with partial circuit redundancy. For compact ASIC implementations, taking such an approach gives a better ability to detect faults than simple parity codes, with less area cost than proposed schemes which use full hardware redundancy. The results of the implementation analysis in this paper show that it is possible to implement an error detection scheme that is robust to multiple faults in a compact AES design such that about 39% of the overall system is devoted to the error detection functionality.

KEY WORDS
application specific integrated circuit, cryptography, encryption, AES, cryptanalysis, error detection

1. Introduction

Since the National Institute of Standards and Technology (NIST) announced the selection of Rijndael as the Advanced Encryption Standard (AES) in November 2001 [1], AES has been accepted as the popular means to encrypt sensitive commercial and government data. Various hardware implementation architectures and optimizations have been proposed for different applications. Implementations that are compact in area, suitable for practical low-end embedded applications, such as smart cards, PDAs, cell phones, and other mobile devices are the focus of this paper.

Deliberately inducing malicious faults into cryptographic implementations and breaking the secret keys or cipher structures from the information resulting from faulty computations is a practical and efficient cryptanalysis technique called *fault-based cryptanalysis*, first proposed by Boneh, Demillo, and Lipton [2], and more recently applied to AES [3]. It is well understood that one approach to guarding against fault attacks on ciphers is to implement *concurrent error detection (CED)* circuitry along with the cipher functional circuit so that suitable action may be taken if an attacker attempts to acquire secret information about the circuit by inducing faults.

The objective of the research in this paper is to investigate a compact ASIC implementation of AES with concurrent error detection. It attempts to create a bridge between the area requirements of embedded applications and effective fault attack resistance, such that the system is able to effectively detect faults with modest area overhead.

1.1 Advanced Encryption Standard

AES is a symmetric-key block cipher with a data block length of 128 bits, which supports different key lengths of 128, 192 or 256 bits. In this paper, we consider the implementation of the 128-bit key system only, as this is the most commonly implemented form of AES.

AES can be used to both encrypt and decrypt data. For the 128-bit key implementation, the encryption and decryption processes consist of 10 rounds of operations. There are four main operations on the datapath in each round for the encryption process, as illustrated in Figure 1: *byte-substitution*, *shift-row, mix-column* and *add-round-key*. Another important function is the *key expansion*, which takes the 128-bit key and generates *round keys* (labeled as K_i in Figure 1) to be applied to each round.

We now briefly describe the significant operations of the cipher. For the detailed description, the reader is referred to [4]. For convenience in the description, the 128-bit data is divided into 16 bytes and arranged as a two-dimensional 4-by-4 array of bytes.

(1) *byte-substitution*: Each byte is substituted by the corresponding element in a table referred as an *s-box*. The s-box is an 8-bit input, 8-bit output component, and, hence, is represented by a table that contains 256 8-bit values. The byte-substitution operation is the only non-linear operation in the algorithm.

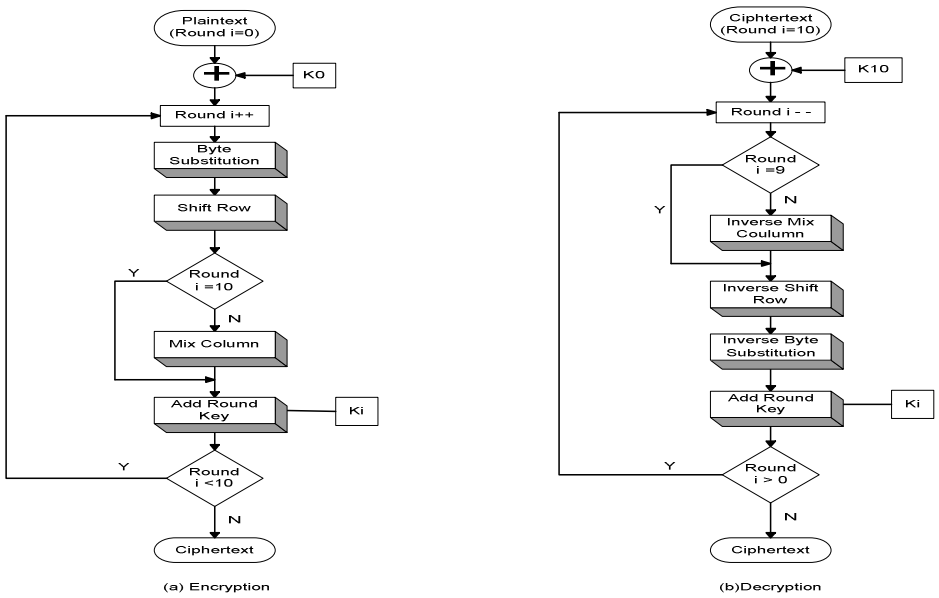

Figure 1. AES Encryption and Decryption

(2) *shift-row*: The shift-row operation is a simple transposition operation on the bytes of each row in the array. The first row has no shift, the second row has a left rotation of 1 byte, the third row has a left rotation of 2 bytes, and the last row has a left rotation of 3 bytes.

(3) *mix-column*: In this operation, a fixed array is used to perform multiplication using modulo x^4+1 with each column over $GF(2^8)$. For encryption, the mix-column operation is performed at each round except the last one.

(4) *add-round-key*: The add-round-key operation is a bit-wise exclusive-or (XOR) operation of the whole data block and round key. There is one key addition operation before the first round for pre-whitening.

(5) *key expansion*: The key expansion algorithm, or *key expander*, can take an initial key of length 128 bits, 192 bits or 256 bits. For a 128-bit key, the key expander takes the 128-bit initial key as four 32-bit words of input and generates 40 words to provide each of the 10 rounds with a 4-word round key. Each of the round keys depends on the key of the previous round.

The decryption process for AES has a slightly different structure from encryption. However, with some changes in the operation order and the key expansion function, an equivalent decryption structure can be achieved using inverse functions for the byte-substitution, shift-row and mix-column operations.

1.2 AES Hardware Implementations

The AES algorithm has a simple structure and can be implemented efficiently on a wide range of platforms. Although the software realization of the AES algorithm can lead to relatively high throughput when compared to other block ciphers, hardware implementations such as special purpose cryptographic processors are desirable in many practical applications. Hardware implementations can generally be viewed as falling into one of two categories: (1) high-speed implementations and (2) compact implementations. In this paper, we focus on the compact implementation of AES targeted to lower speed systems such as cell phones and PDAs and other embedded systems, such as smart cards. In these applications, the main concern is to minimize the area and limit power consumption of the design.

Compact AES hardware implementations are typically iterative designs based on one-round or quarter-round loop architectures, with the application of design techniques for hardware resource sharing such as the merging of the encryption and decryption datapaths and the reuse of components between the datapath and key expander.

Several papers have reported compact designs for AES ASIC implementation. For example, in [5], the design uses a methodology to minimize the s-box component area using arithmetic operations in a composite field of the form $GF(((2^2)^2)^2)$. The cipher architecture uses an iterative quarter-round structure. That means the width of the datapath is 32 bits, so that four s-boxes are processed each pass of the loop. Hence, a full round of 128-bit data needs four clock cycles to be finished. The hardware resources are efficiently shared between the encryption and decryption processes, including the sharing between s-box and inverse s-box and mix-column and inverse mix-column. The s-boxes are reused between datapath and key expander as well. The key expander generates the round keys on-the-fly, saving the memory area needed to store

the pre-computed keys. The design produced is an extremely small 128-bit key AES circuit of 5.4k gates based on a 0.11 μm CMOS standard cell library and the system has a throughput of 311 Mbps.

A more recent compact implementation of AES [6] is able to achieve an even lower gate count of about 3.4k gates for the circuit in 0.35 μm CMOS technology. Additionally, the circuit is designed for low power consumption. This is principally achieved with an iterative design based around the use of only one s-box in each iteration. As a result, the speed of the circuit is dramatically less than other designs and is only 9.9 Mbps.

In the work presented in this paper, we have chosen to focus on the iterative architecture based on the quarter-round or four s-box iteration, as we have assumed that the speed penalty paid for a reduction in circuit size as in [6], is undesirable for many applications.

It should also be noted that several papers have investigated the low complexity implementation of the AES s-box [5, 7, 8]. This work principally focuses on the implementation of the s-box based on composite field representation in $GF(((2^2)^2)^2)$ or $GF((2^4)^2)$. In [9], various s-box implementations are considered and the implementation of [7] is selected for the design presented in this paper for its low complexity and performance.

1.3 Fault-Based Cryptanalysis

Although today's hardware implementations are relatively reliable, it is still possible and practical for opponents to intentionally induce faults into hardware computations, especially for small, portable devices such as smartcards and other embedded systems. Fault-based cryptanalysis is a powerful attack technique that deliberately injects faults into the cryptographic devices and exploits the fact that the erroneous computations leak secret parameters or sensitive information about the cipher. This attack idea was first proposed in [2] and, subsequently, in [10], the attack was extended to symmetric cryptosystems such as DES. Fault analysis has now been applied to AES [3]. Different fault-based attacks are associated with different assumptions for fault models. As noted in [3], attacks must consider several aspects of a fault model such as (1) whether the fault is permanent or transient, (2) whether the fault location and/or timing of the fault can be controlled, (3) the type of fault (eg. bit flip or stuck-at-0/1), and (4) the number of faults induced. The results show that AES is sensitive to fault-based attacks and the recovering of the secret key can be achieved by using a small number of faulty ciphertexts under certain hardware fault models.

1.4 Concurrent Error Detection for AES

Concurrent error detection checks the system operation during the computation to guarantee the system output is correct. If an erroneous output is produced, CED will detect the presence of the faulty computation and the system can discard the erroneous output before transmission. Thus, the encryption system can achieve resistance to malicious fault-based attacks. Any CED technique will introduce some overhead into the system since circuitry must be added that predicts the system output or some characteristic parameter of the system output used to check the correctness of the system.

(i) Techniques Based on Hardware or Time Redundancy

Straightforward duplication of the encryption hardware for self-checking is the simplest form of the redundancy technique for concurrent error detection. The output of the duplicated circuit is compared with the result of the original hardware, and any mismatch means the detection of errors. The method can detect any type or any number of fault injections if the duplicated module is fault-free, and is highly likely to detect any errors even if faults occur in both the original and duplicated hardware as long as the faults do not occur at the same location. Since the original circuit and duplicated module are working simultaneously, this technique does not cause any notable time delay or degradation of the original hardware performance. However, it requires considerable hardware overhead of more than 100%. Therefore, this method is not suitable for area critical applications.

The time redundancy technique involves encrypting or decrypting the same data a second time using the same datapath and comparing the two results. This method has more than 100% time overhead and is only applicable to transient faults. For permanent faults in the circuit, since the same faults occur in both computations, the system can not detect errors.

A hardware and time redundancy approach for the AES system was proposed in [11] by employing the inverse relationship between the encryption and decryption process. This method performs a test decryption of the encrypted data and then checks if the decrypted data matches the original message or not. The authors exploited the inverse relationship between the encryption and decryption process at the algorithm level, round level and individual operation level. The method is able to detect any type and any number of faults, but it needs a separated datapath for encryption and decryption. Compared to an encryption/decryption integrated datapath, like the AES compact implementation to be presented in this paper, this method results in more than 100% hardware overhead.

(ii) Techniques Based on Error Detection Codes

Error detection coding techniques have been applied to CED in block ciphers in several papers and the fault detection coverage usually depends on the particular adopted coding schemes and hardware implementation details.

A simple parity check, with the advantage of low hardware overhead, has been proposed as a CED method for AES in [12] and [13, 14]. The detection latency and fault detection coverage depend on how many parity bits

the system uses and the locations of the checking points. In [12], a low-cost approach of concurrent parity checking for the AES algorithm is proposed. In this method, a simple parity bit for a 128-bit data block is used and this parity is modified at each step of the AES algorithm to generate the prediction of the output parity. The predicted parity is then compared to the actual output parity of each round to detect if there is any error in the system. The checking points are set at the end of every round, so the detection latency is the time needed to process data for one round. The parity prediction of the cipher's linear operations (i.e., shift-row, mix-column and add-round-key) is straightforward and the prediction of the non-linear byte-substitution is accomplished by using an extra output bit associated with each s-box to predict the output parity of the s-box. This approach is well suited to memory-based s-box implementations and, in this context, it is guaranteed to correct single bit faults. However, since only a single parity bit is used for the 128-bit datapath, this implementation will not detect many multiple fault scenarios. Further, a single parity bit for an s-box output does not provide a robust error detection for compact ASIC implementations of AES, such as the one proposed in this paper, which use a combinational-logic based s-box implementation, since it is possible for some types of single faults (that occur within the s-box logic), as well as many other multiple fault scenarios, to be undetectable.

The method presented in [13, 14] is similar to [12], but it associates one redundant parity bit with each byte of the 128-bit data block. Thus the parity code for this approach uses 16 bits. This 16-bit parity code uses more hardware overhead for parity code storage and prediction, but it has better fault detection coverage than the 1-bit parity code scheme. However, again the approach uses a parity bit prediction of the output of s-box assuming a memory-based s-box implementation and is therefore not well suited to compact ASIC implementations in that many single and multiple fault scenarios will be undetectable.

Other CED methods include systematic nonlinear error detection codes [15]. This scheme has better fault detection coverage than a normal linear code, and the design introduces a linear predictor to protect the encryption, decryption and key expander with about 75% hardware overhead for FPGA implementations. Unfortunately, the overhead analysis in [15] does adequately describe the design of AES and it is not clear that the results for a compact implementation would not be worse. Also, recently, in [16], the application of cyclic redundancy checks to error detection in AES is presented. Several techniques are considered varying from schemes such as a pure-parity approach to a hybrid of parity and redundancy. Unfortunately, although the authors do discuss the overhead associated with each scheme, they do not examine an actual implementation to characterize the impact of the concurrent error detection on the area of real implementations.

In this paper, we examine the application of a hybrid concurrent error detection scheme in the context of an actual implementation of a compact ASIC design of AES. Our proposed scheme has the advantage that it is effective for implementations that do not rely on memory-based s-box structures. As a result, most multiple faults and all single faults can be detected and, as shown, the area overhead is modest, especially when it is considered that the targeted design is a compact architecture.

2. A Compact AES Implementation

In this section, we outline a compact AES implementation, first referred to in [9] and detailed in [17]. In terms of hardware implementation, s-boxes are the most complex components in the AES algorithm and the s-boxes have an important influence on the area, speed and power consumption of the overall AES system. Hence, in [9], the iterative structure and three methods of compact s-box implementation were investigated through synthesis targeted at 0.18 μm CMOS technology. Based on the studies, by considering the trade-off of area and speed, it was decided to focus the design investigation on an architecture based on a quarter-round iteration (i.e., processing of 4 s-boxes per pass of the iterative loop) using s-boxes constructed based on a composite field representation of the form $GF((2^4)^2)$ [7].

2.1 Encryption/Decryption Architecture with Key Expansion

In the compact design, the encryption and decryption functionality are merged into one equivalent architecture and circuitry is provided for key scheduling on-the-fly (that is, in parallel with the datapath) for encryption and decryption. The hardware components are shared as much as possible to reduce the circuit size. The encryption/decryption architecture is shown in Figure 2, where the unlabelled boxes in the diagram represent registers. Since the architecture operates on 32 bits of datapath per iteration, a full round requires four iterations. The 32-bit shift registers not only work as data registers but also implement the shift-row operation.

In order to share the hardware resources between the encryption and decryption processes, it is necessary to modify the order of operations for the structure. Firstly, the order of byte-substitution and shift-row for the encryption process is exchanged. Since both of the operations are byte-oriented, this does not alter the result of the round. The second change of structure involves exchanging the order of mix-column and add-round-key for the decryption process. This change causes a corresponding change in the key expander such that the inverse mix-column is added at the end of key scheduling function. Also the multiplicative inverse in $GF(2^8)$ for the s-box and inverse s-box is shared, as well as the hardware between mix-column and its inverse operation.

The key expander needs the byte-substitution operation to generate the key for encryption and decryption. Since the s-box is the most costly component in the circuit, sharing between the datapath and the key expander is a

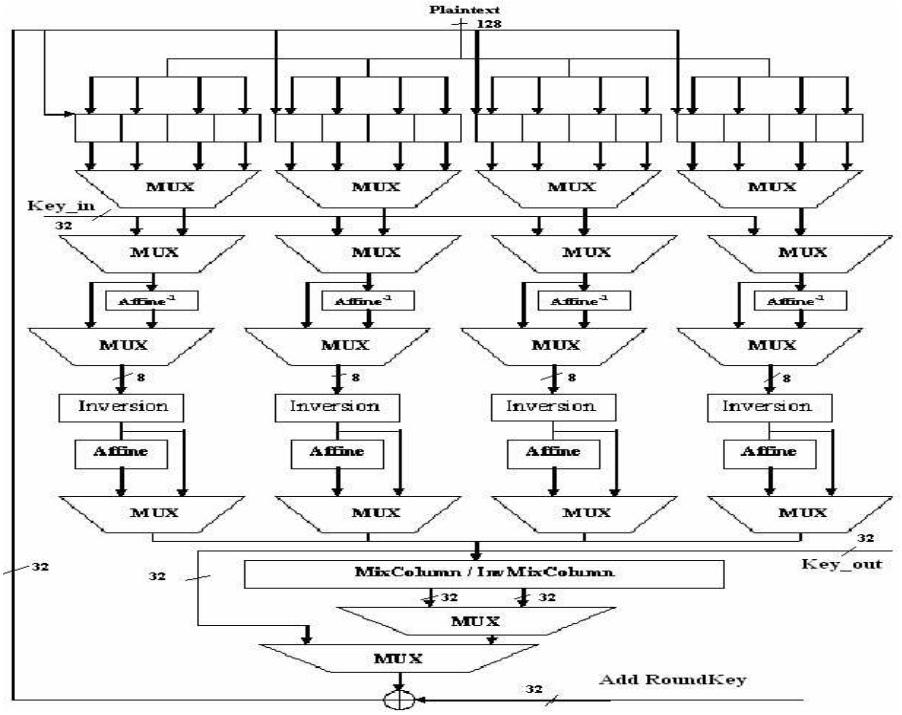

Figure 2. Encryption-Decryption Datapath

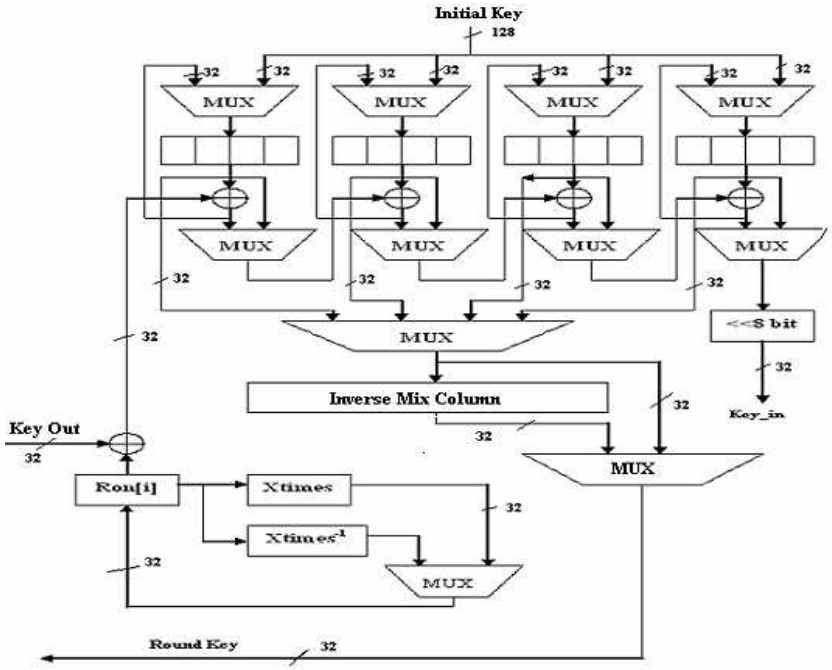

Figure 3. Encryption-Decryption Key Expander

good method to reduce the circuit size. Multiplexers are used after the shift-row operation to select to process cipher data or the round key. The key is taken after byte-substitution back to the key expander to continue the key process. This sharing of s-boxes causes an increase of one clock cycle in the datapath for each round in the encryption and decryption. The key expander circuit is illustrated in Figure 3.

2.2 Hardware Complexity Analysis

The complete AES algorithm has been designed, simulated, and synthesized using the 0.18 µm CMOS standard cell library with the Synopsys Design Analyzer as the design tool. Synthesizing based on minimizing the area of the circuit resulted in about 6.7k gates with the throughput of the circuit being 112 Mbps. The resulting

relative complexities of the various components are given in Table 1.

Component	Complexity (% area)
enc/dec s-boxes	24.7
data registers	22.7
mix-column+inverse	7.1
XORs	5.0
multiplexers	2.6
Total	**62.1**

(a) Datapath

Component	Complexity (% area)
registers	12.5
multiplexers	10.4
inverse mix-column	7.1
XORs	4.2
round constant calculation	1.3
Total	**35.5**

(b) Key Expander

Component	Complexity (% area)
enc/dec datapath	62.1
key expander	35.5
system controller	2.4

(c) Complete AES System

Table 1. Complexity of Compact AES Implementation

3. Concurrent Error Detection Applied to Compact AES

In this section, we now consider the application of an effective error detection scheme to the compact AES implementation described in the previous section. The proposed scheme is capable of detecting single bit errors caused by maliciously induced faults by attackers, as well as most multiple error scenarios. Subsequent to error detection, appropriate action is taken to suppress release of the flawed ciphertext from the system, thereby minimizing the cipher's susceptibility to fault-based attacks.

3.1 Proposed Scheme for Error Detection

Based on the review of concurrent error detection techniques and proposed schemes for CED of AES, we have investigated an error detection approach for our AES implementation that is a hybrid scheme combining both parity checking and hardware redundancy techniques. Although multiple parity bits can be an effective mechanism for detecting single and multiple bit errors, applying a parity bit to the s-box output is not useful when a fault is induced inside the combinational circuit of the s-box resulting in an even number of errors at the s-box output that can not be detected. Therefore, hardware redundancy for the s-boxes is particularly attractive when the s-boxes are implemented using a compact approach realized as combinational logic. For the mix-column, shift-row and add-round-key operations, the parity checking schemes are effective with small cost, so parity checking is adopted for these operations. The proposed scheme is implemented and analyzed based on our compact hardware implementation of the AES algorithm, and the CED scheme is applied to the whole AES system including the encryption/decryption datapath and key expander.

A multiple-bit parity code similar to [13, 14] is adopted instead of the 1-bit parity code of [12] even though the 1-bit parity code has smaller hardware overhead, because the multiple-bit parity code achieves better fault detection coverage for multiple faults. Each bit in the parity code represents a parity bit for each byte in the data. However, our scheme differs from [13, 14], in that the s-boxes are duplicated while using parity prediction for other components in the system, rather than using a pure parity-based scheme which favours a memory-based implementation of the s-boxes. For parity prediction of mix-column, the same modification algorithm as in [13, 14] is used. Check points are placed within each round to achieve good detection latency and higher fault detection coverage. The objective of the design is to yield fault detection coverage of 100% for the single faulty bit model and high coverage for multiple fault scenarios, assuming a fault model of a transient or permanent fault as a stuck-at-0 or stuck-at-1 fault in combinational logic and gate wiring or a bit-flip fault in registers.

The hybrid CED scheme applied to the quarter-round iterative structure is shown in Figure 4. The variables s_{r0}, s_{r1}, s_{r2} and s_{r3} are four bytes of data in the row r, and p_{r0}, p_{r1}, p_{r2} and p_{r3} are their corresponding four parity bits. Here we will explain the parity prediction and checking for each operation in more detail:

Data Registers and Shift-Row: A parity generator is needed to generate the parity code of the original and updated data and put a 4×4 parity code into four 4-bit shift registers according to the corresponding data byte position. These small parity shift registers are shifted and loaded with the same pace as the data registers. A parity checker is placed at the output of the registers to detect the fault in the data registers and shift-row transformation.

Byte-Substitution: Since the simple parity checking is not sufficient for the combinational logic of the s-box based on arithmetic in $GF((2^4)^2)$, the s-boxes are duplicated in hardware. An equality checker is located at the output of the s-boxes to check any fault in s-box computation. Moreover, another parity generator is needed to generate the new parity bits after the byte-substitution transformation for the use of parity checking of mix-column.

Mix-Column: The same mix-column (and inverse mix-column) parity prediction method as in [13, 14] is adopted. After the mix-column operation, a check point is applied to detect any fault that may have occurred.

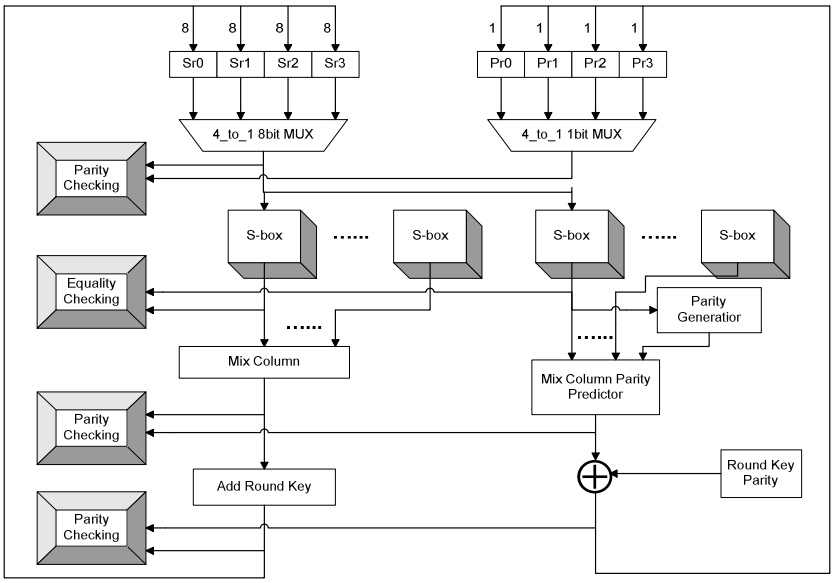

Figure 4. Hybrid CED Structure

Add-Round-Key: Since this operation is simple XOR gates, the prediction for the new parity is just the XOR between the old parity and round key parity for each byte. Also, a check point is applied after this operation.

The CED scheme is able to detect all single faults occurring at the input of each round, between the round operations or inside of each round operation. Because all single faults inside of mix-column result in an odd number of erroneous bits at the output, the resulting errors can be detected by parity checking [13]. Since the shift-row and add-round-key operation are simply implemented by wiring and XOR gates, all single faults result in a single error as well, which can be detected by parity checking. If a single fault is injected inside of the s-box circuit, the comparison of the faulty s-box output with the unfaulty s-box output will ensure that the fault is detected.

For multiple faults, the situation is more complex. For the portions of the circuit protected by the parity code, faults that result in an odd number of errors can be detected. As well, faults that result in an odd number of errors in a byte of the datapath will also be detected, even if the total number of errors is even. The parity code can not detect the faults that result in an even number of errors such that all bytes have an even number of errors. For the s-boxes, since the scheme is based on the duplication of s-box computation, all multiple fault scenarios will be detected, except for the unlikely cases where the same errors occur at the output of both s-box sets.

The system can detect the errors shortly after the faults are induced because the detection latency is only the output delay of each operation. Once an error is detected, the data currently being processed is discarded. Since the key scheduling uses similar functions as the datapath, a similar CED approach has been applied to the key expander.

3.2 Hardware Complexity Analysis

We have implemented the hybrid CED scheme for our AES compact hardware implementation, including both the encryption/decryption datapath and key expander. The system was synthesized to minimize area and, hence, can be compared to the original synthesized circuit without CED, which resulted in 6.7k gates. The resulting circuit for the AES system with concurrent error detection requires an area equivalent to 10.9k gates, with 39.1% of the circuit dedicated to CED. This is equivalent to an area overhead of 64.3% with respect to our original compact AES hardware system. A summary of the overhead of the scheme is shown in Table 2.

Component	Original Circuit (without CED) (gates)	CED Circuit (gates)	CED Overhead
datapath	4228	2555	60.4%
key expander	2428	1613	66.4%
complete system	6656	4278	64.3%

Table 2. Complexity of Hybrid CED Scheme

3. Conclusion

The primary focus of this paper has been the study of a compact hardware implementation of the AES system with concurrent error detection. The AES implementation is aimed at area-critical embedded applications, such as

smart cards, PDAs, cell phones, and other mobile devices. The proposed hybrid CED scheme achieves effective detection for single faults and most multiple faults with about 39% of the final compact AES system dedicated to the CED functionality.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and CMC Microsystems.

References

[1] National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES)", Federal Information Processing Standard Publication 197, Nov. 2001. Available at: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] D. Boneh, R.A. DeMillo, and R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults", *Advances in Cryptology – EUROCRYPT '97*, *Lecture Notes in Computer Science*, vol. 1233, Springer, pp. 37-51, 1997.

[3] J. Blomer and J. Seifert, "Fault Based Cryptanalysis of Advanced Encryption Standard (AES)", *Financial Cryptography (FC 2003), Lecture Notes in Computer Science*, vol. 2742, Springer, pp. 162-181, 2003.

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

[5] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-box Optimization", *ASIACRYPT 2001*, *Lecture Notes in Computer Science*, vol. 2248, Springer, pp. 239-254, 2001.

[6] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES Implementation on a Grain of Sand", *IEE Proceedings on Information Security*, vol. 152, no. 1, pp. 13-20, 2005.

[7] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC Implementation of the AES SBoxes", *The Cryptographer's Track at the RSA Conference (CT-RSA 2002)*, *Lecture Notes in Computer Science*, vol. 2271, Springer, pp. 67-78, 2002.

[8] D. Canright, "A Very Compact S-box for AES", *Workshop on Cryptography Hardware and Embedded System (CHES 2005), Lecture Notes in Computer Science*, vol. 3659, Springer, pp. 441-456, 2005.

[9] N. Yu and H.M. Heys, "Investigation of a Compact Hardware Implementation of the Advanced Encryption Standard", *Canadian Conference on Electrical and Computer Engineering (CCECE 2005)*, 2005.

[10] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *Advances in Cryptology - Crypto '97*, *Lecture Notes in Computer Science*, vol. 1294, Springer, pp. 513-525, 1997.

[11] R. Karri, K. Wu, P. Mishra, and Y. Kim, "Fault-based Side-channel Cryptanalysis Tolerant Rijndael Symmetric Block Cipher Architecture," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'01)*, 2001.

[12] K. Wu, R.Karri, G. Kouznetzov and M.Goessel, "Low Cost Concurrent Error Detection for the Advanced Encryption Standard," *International Test Conference 2004 (ITC 2004)*, pp. 1242-1248, 2004.

[13] 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 Transaction on Computers*, vol. 52, no.4, pp. 492-505, April 2003.

[14] G. Bertoni, L. Breveglieri, I. Koren, and P. Maistri, "An Efficient Hardware-Based Fault Diagnosis Scheme for AES: Performance and Cost", *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT '04)*, 2004.

[15] M. Karpovsky, K. Kulikowski, and A. Taubin, "Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard," *International Conference on Dependable System and Networks (DSN '04)*, 2004.

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

[17] N. Yu, "Compact Hardware Implementation of AES with Concurrent Error Detection", M.Eng. Thesis, Memorial University of Newfoundland, 2005.