Concurrent Error Detection Schemes for Involution Ciphers

Nikhil Joshi, Kaijie Wu, and Ramesh Karri

Department of Electrical and Computer Engineering, Polytechnic University
Brooklyn, NY 11201 USA
{njoshi01, kwu03}@utopia.poly.edu
ramesh@india.poly.edu

Abstract. Because of the rapidly shrinking dimensions in VLSI, transient and permanent faults arise and will continue to occur in the near future in increasing numbers. Since cryptographic chips are a consumer product produced in large quantities, cheap solutions for concurrent checking are needed. Concurrent Error Detection (CED) for cryptographic chips also has a great potential for detecting (deliberate) fault injection attacks where faults are injected into a cryptographic chip to break the key. In this paper we propose a low cost, low latency, time redundancy based CED technique for a class of symmetric block ciphers whose round functions are involutions. This CED technique can detect both permanent and transient faults with almost no time overhead. A function F is an involution if F(F(x))=x. The proposed CED architecture (i) exploits the involution property of the ciphers and checks if x=F(F(x)) for each of the involutional round functions to detect transient and permanent faults and (ii) uses the idle cycles in the design to achieve close to a 0% time overhead. Our preliminary ASIC synthesis experiment with the involutional cipher KHAZAD resulted in an area overhead of 23.8% and a throughput degradation of 8%. A fault injection based simulation shows that the proposed architecture detects all single-bit faults.

Keywords: Concurrent Error Detection (CED), Fault Tolerance, Involutional ciphers, KHAZAD

1 Introduction

Because of the rapidly shrinking dimensions in VLSI, faults arise and will continue to occur in the near future in increasing numbers. Faults can broadly be classified in to two categories: Transient faults that die away after sometime and permanent faults that do not die away with time but remain until they are repaired or the faulty component is replaced. The origin of these faults could be due to the internal phenomena in the system such as threshold change, shorts, opens etc. or due to external influences like electromagnetic radiation. The faults could also be deliberately injected by attackers in order to extract sensitive information stored in the system. These faults affect the memory as well as the combinational parts of a circuit and can only be detected using Concurrent Error Detection (CED). This is especially true for sensitive devices such

M. Joye and J.-J. Quisquater (Eds.): CHES 2004, LNCS 3156, pp. 400–412, 2004.

as cryptographic chips. Hence, CED for cryptographic chips is growing in importance. Since cryptographic chips are a consumer product produced in large quantities, cheap solutions for concurrent checking are needed. CED for cryptographic chips also has a great potential for detecting (deliberate) fault injection attacks where faults are injected into a cryptographic chip to break the key [1 2 3 4]. A Differential Fault Attack technique against AES and KHAZAD was discussed in [5]. It exploited the faults in bytes. It was shown that if the fault location could be chosen by the attacker, it required only 2 ciphertexts for a successful attack. CED techniques could help prevent such kind of attacks.

Until now some of the following CED methods for cryptographic algorithms are known. In [6] a CED approach for AES and other symmetric block ciphers that exploits the inverse relationship between the encryption and decryption at the algorithm level, round level and individual operation level was developed. This technique had an area overhead of 21% and a time overhead up to 61.15%. In [7] this inverse-relationship based technique was extended to AES round key generation. A drawback of this approach is that it assumes that the cipher device operates in a half-duplex mode (i.e. either encryption or decryption but not both are simultaneously active).
In [8] a parity-based method of CED for encryption algorithms was presented. The technique adds one additional parity bit per byte resulting in 16 additional bits for the 128-bit data stream. Each of the sixteen 8-bit×8-bit AES S-Boxes is modified into 8-bit×9-bit S-Boxes. In addition, this technique adds an extra parity bit per byte to the outputs of the Mix-Column operation because Mix-Column does not preserve parity of its inputs at the byte-level.

In this paper, we propose a low cost, low latency CED technique for involution-based symmetric block ciphers. Any function F is an involution if F(F(x))=x. An involutional symmetric block cipher is one in which each round operation of the cipher is an involution. Usually, in symmetric block ciphers, the decryption operation differs significantly from encryption. Although it is possible to implement decryption in such a way that it has the same sequence of operations as encryption, round level operations such as S-Box etc are different. With involutional ciphers, all the operations are involutions. So, it becomes possible to implement the decryption in such a way that only the round keys used are different from that for encryption. Besides the implementation benefit, an involutional structure also implies equivalent security for both encryption and decryption [9].

The CED technique we propose exploits the involution property of the round operations of this class of symmetric block ciphers and checks if x=F(F(x)) for each of the involutional round functions to detect the faults in the system. It offers optimum trade-off between area and time overhead, performance and fault detection latency. Further, it requires minimal modification to the encryption device and is easily applicable to all involution-based symmetric block ciphers. Traditionally, time redundancy based CED schemes cannot detect permanent faults but usually entail >100% time overhead. Although the CED scheme we propose is time redundancy based, it entails almost zero time overhead.

The paper is organized as follows. In section 2 we will recapitulate Involution-based symmetric block ciphers. In section 3 we will describe the involution based CED architecture and the error detection capability of the proposed method. To obtain the area overhead and the additional time delay, we modeled the method using VHDL and synthesized using cadence ASIC synthesis tool PKS. The results of this implementation are presented in section 4. The error detection capabilities of the proposed methods are discussed in section 5 and finally, Conclusions are reported in section 6.

2 Involutional Block Ciphers

A substitution permutation network (SPN) symmetric block cipher is composed of several rounds and each round consists of a non-linear substitution layer (S-Box), a linear diffusion layer and a layer to perform exclusive-or operation with the round key. The linear diffusion layer ensures that after a few rounds all the output bits depend on all the input bits. The nonlinear layer ensures that this dependency is of a complex and nonlinear nature. exclusive-or with the round key introduces the key material. Recently, several SPN symmetric block ciphers that are composed of involution functions have been proposed and analyzed [10,11,12,13,14]. In an involutional SPN cipher, the non-linear S-Box layer and the linear diffusion layer are involutions. An advantage of using involutional components is that the encryption data path is identical to the decryption data path. In an involutional SPN cipher, the round key generation algorithm for decryption is the inverse of the round key generation algorithm for encryption.

In this paper we will consider the 64-bit involutional SPN cipher KHAZAD[10] shown in Figure 1 as a running example. The involutional SPN cipher KHAZAD uses seven identical encryption/decryption rounds (ρ) with each encryption/decryption round composed of an involutional non-linear byte substitution γ (i.e. γ(γ(x)) = x), an involutional linear diffusion layer θ (i.e. θ(θ(x)) = x) and exclusive-or with key σ which is an involution as well (i.e. σ(σ(x)) = x).

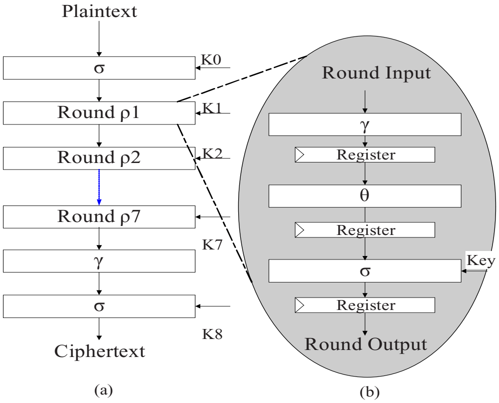

Fig. 1. (a) Khazad Cipher (b) Round Function $\rho[Key](x) = \sigma_{Key}(\theta(\gamma(x)))$

2.1 The Non-linear Substitution Function γ

The involutional non-linear substitution layer γ of KHAZAD uses eight 8x8 involutional S-Boxes, each of which is constructed from three layers of 4x4 involutional P-Boxes and Q-Boxes as shown in Figure 2. The truth tables for the P-Box and the Q-Box are given in Table 1. Observe that the P-Box and the Q- box are also involutions. We implemented the P-Box and Q-box as ROMs and the γ layer uses a total of 24 P-Boxes and 24 Q-Boxes.

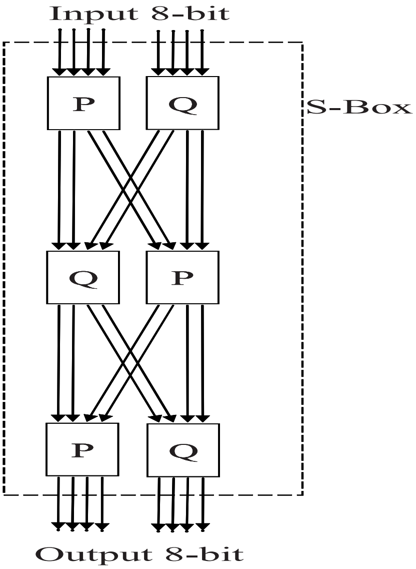

Fig. 2. S-Box in the non-linear layer γ of KHAZAD

Table 1. 4x4 involutional P-Box and Q-Box

x	P(x)	Q(x)
0	3	9
1	F	E
2	E	5
3	0	6
4	5	A
5	4	2
6	B	3
7	C	C
8	D	F
9	A	0
A	9	4
B	6	D
C	7	7
D	8	B
E	2	1
F	1	8

2.2 The Linear Diffusion Layer θ

The diffusion layer θ is a linear mapping based on the [16, 8, 9] MDS code with generator matrix GH = [I H], and H = had(01x, 03x, 04x, 05x, 06x, 08x, 0bx, 07x) i.e. $\theta(a) = b \Leftrightarrow b = a \times H$, where

$$H = \begin{bmatrix} 01x & 03x & 04x & 05x & 06x & 08x & 0Bx & 07x \\ 03x & 01x & 05x & 04x & 08x & 06x & 07x & 0Bx \\ 04x & 05x & 01x & 03x & 0Bx & 07x & 06x & 08x \\ 05x & 04x & 03x & 01x & 07x & 0Bx & 08x & 06x \\ 06x & 08x & 0Bx & 07x & 01x & 03x & 04x & 05x \\ 08x & 06x & 07x & 0Bx & 03x & 01x & 05x & 04x \\ 0Bx & 07x & 06x & 08x & 04x & 05x & 01x & 03x \\ 07x & 0Bx & 08x & 06x & 05x & 04x & 03x & 01x \end{bmatrix}$$

The H matrix is symmetric and unitary and therefore θ is an involution. For efficient hardware implementation the θ layer was described in [15] as follows:

$$
\begin{aligned}
b_0 &= a_0 \oplus a_1 \oplus a_3 \oplus a_6 \oplus a_7 \oplus X(a_1 \oplus a_4 \oplus a_6 \oplus a_7) \oplus X^2(a_2 \oplus a_3 \oplus a_4 \oplus a_7) \oplus X^3(a_5 \oplus a_6)\\
b_1 &= a_0 \oplus a_1 \oplus a_2 \oplus a_6 \oplus a_7 \oplus X(a_0 \oplus a_5 \oplus a_6 \oplus a_7) \oplus X^2(a_2 \oplus a_3 \oplus a_5 \oplus a_6) \oplus X^3(a_4 \oplus a_7)\\
b_2 &= a_1 \oplus a_2 \oplus a_3 \oplus a_4 \oplus a_5 \oplus X(a_3 \oplus a_4 \oplus a_5 \oplus a_6) \oplus X^2(a_0 \oplus a_1 \oplus a_5 \oplus a_6) \oplus X^3(a_4 \oplus a_7)\\
b_3 &= a_0 \oplus a_2 \oplus a_3 \oplus a_4 \oplus a_5 \oplus X(a_2 \oplus a_4 \oplus a_5 \oplus a_7) \oplus X^2(a_0 \oplus a_1 \oplus a_4 \oplus a_7) \oplus X^3(a_5 \oplus a_6)\\
b_4 &= a_2 \oplus a_3 \oplus a_4 \oplus a_5 \oplus a_7 \oplus X(a_0 \oplus a_2 \oplus a_3 \oplus a_5) \oplus X^2(a_0 \oplus a_3 \oplus a_6 \oplus a_7) \oplus X^3(a_1 \oplus a_2)\\
b_5 &= a_2 \oplus a_3 \oplus a_4 \oplus a_5 \oplus a_6 \oplus X(a_1 \oplus a_2 \oplus a_3 \oplus a_4) \oplus X^2(a_1 \oplus a_2 \oplus a_3 \oplus a_7) \oplus X^3(a_0 \oplus a_3)\\
b_6 &= a_0 \oplus a_1 \oplus a_5 \oplus a_6 \oplus a_7 \oplus X(a_0 \oplus a_1 \oplus a_2 \oplus a_7) \oplus X^2(a_1 \oplus a_2 \oplus a_4 \oplus a_5) \oplus X^3(a_0 \oplus a_3)\\
b_7 &= a_0 \oplus a_1 \oplus a_4 \oplus a_6 \oplus a_7 \oplus X(a_0 \oplus a_1 \oplus a_3 \oplus a_6) \oplus X^2(a_0 \oplus a_3 \oplus a_4 \oplus a_5) \oplus X^3(a_1 \oplus a_2)
\end{aligned}
$$

where a_7-a_0 and b_7-b_0 are the eight input bytes and the eight output bytes of the diffusion layer θ.

Assuming that a(7),a(6),a(5),a(4),a(3),a(2),a(1) and a(0) are the eight bits in an input byte a, the three byte level functions X(a), X^2(a)=X(X(a)) and X^3(a)=X(X(X(a))) can be defined as follows:

$$
\begin{aligned}
&X(a(7),a(6),a(5),a(4),a(3),a(2),a(1),a(0))\\
&= a(6),a(5),a(4),a(3)\oplus a(7),a(2)\oplus a(7),a(1)\oplus a(7),a(0),a(7)\\
&X^2(a(7),a(6),a(5),a(4),a(3),a(2),a(1),a(0))\\
&= a(5),a(4),a(3)\oplus a(7),a(2)\oplus a(7)\oplus a(6),a(1)\oplus a(7)\oplus a(6),a(0)\oplus a(6),a(7),a(6)\\
&X^3(a(7),a(6),a(5),a(4),a(3),a(2),a(1),a(0))\\
&= a(4),a(3)\oplus a(7),a(2)\oplus a(7)\oplus a(6),a(1)\oplus a(7)\oplus a(6)\oplus a(5),a(0)\oplus a(6)\oplus a(5),a(7)\oplus a(5),a(6),a(5)
\end{aligned}
$$

It can be seen that if bit a(7) is “0” then function X(a) reduces to a single bit left rotate. Similarly, the function X^2(a) reduces to a 2-bit left rotate when a(7) and a(6) are “0” and finally, the function X^3(a) reduces to a 3-bit left rotate when a(7), a(6) and a(5) are “0”.

2.3 The Exclusive-or Function σ

The key addition σ layer consists of bitwise exclusive-or of the 64-bit round-key K^r with the input to the module. Hence, σ is also an involution.

3 Concurrent Error Detection of Involution Functions

We will first describe a simple CED scheme that exploits the involution property of any involution function and then extend it to the involutional SPN cipher KHAZAD.

If a hardware module implements a function F that satisfies the involution property, faults in this module can be detected by checking if x=F(F(x)). At the beginning of operation, the input x is buffered. The output of the hardware module implementing F is fed back to the input of the module F. The result is compared to the original input x and a mismatch indicates an error. Figure 3 shows this basic idea behind CED scheme for an involutional function.

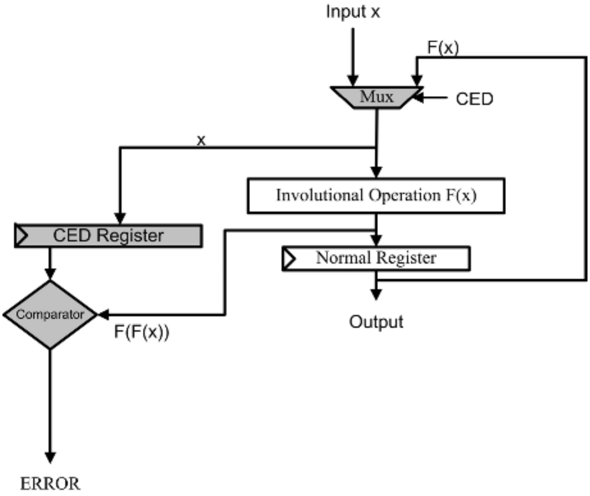

Fig. 3. CED scheme for an involution function

We will show how this CED scheme can be incorporate into a non-pipelined architecture for KHAZAD. Since symmetric block ciphers are frequently used in one of three feedback modes: Cipher Block Chaining (CBC), Output FeedBack (OFB) or Cipher Feedback (CFB), such a non-pipelined architecture is indeed reasonable. The proposed scheme works even in a non-feedback Electronic Code Book (ECB) mode to facilitate pipelining, with an appropriately modified architecture. If a pipelined architecture is implemented, the scheme can result in increased throughput at the cost of a little extra hardware compared to the non-pipelined version owing to a more complex controller. Also, ECB mode is not popularly implemented. Hence, we will consider a non-pipelined version. In the non-pipelined KHAZAD architecture, each KHAZAD round ρ takes three clock cycles to finish with round operation γ, round operation θ and round operation σ completing in clock cycles 1, 2 and 3 respectively.

3.1 CED Scheme 1

Consider a straightforward time redundancy based CED scheme (Scheme 1) wherein round operation γ is performed in clock cycles 1 and 2 on the same input x both the times. If $\gamma(x)$ obtained at the end of clock cycle 1 = $\gamma(x)$ obtained at the end of clock cycle 2 (i.e. no transient fault is detected in module γ), round operation θ is performed in clock cycles 3 and 4 on the same input $\gamma(x)$. If $\theta(\gamma(x))$ at the end of clock cycle 3 = $\theta(\gamma(x))$ at the end of clock cycle 4 (i.e. no transient fault is detected in module θ), round operation σ is performed in clock cycles 5 and 6 on the same input $\theta(\gamma(x))$. If $\sigma(\theta(\gamma(x)))$ at the end of clock cycle 5= $\sigma(\theta(\gamma(x)))$ (i.e. no transient fault is detected in module σ) one KHAZAD round ρ is successfully completed. This time redundancy based scheme can only detect transient faults and entails >100% time overhead.

3.2 CED Scheme 2

As a first modification to the above CED scheme, we propose to integrate involution based CED shown in Figure 4 into the non-pipelined KHAZAD (Scheme 2) as follows. Round operation γ is performed in clock cycle 1 on input x followed by the corresponding CED operation γ(γ(x)) in clock cycle 2. If x= γ(γ(x)) (i.e. no fault is detected in module γ), round operation θ is performed in clock cycle 3 on γ(x) followed by the corresponding CED operation θ(θ(γ(x))) in clock cycle 4. If γ(x)= θ(θ(γ(x))) (i.e. no fault is detected in module θ) then round operation σ is performed in clock cycle 5 on θ(γ(x)) followed by the corresponding CED operation σ(σ(θ(γ(x)))) in clock cycle 6. If θ(γ(x))= σ(σ(θ(γ(x)))) (i.e. no fault is detected in module σ) one KHAZAD round ρ is successful. This modification can detect permanent faults in addition to the transient faults. This is because, although the same module is used twice, the data that it is operating on is different in each case. This was possible due to the involution property of the modules. The time overhead of this modified time redundancy based CED is still >100%.

3.3 CED Scheme 3

During a complete KHAZAD encryption/decryption, round operation γ is busy in clock cycles 1, 4, 7 … and idles in clock cycles 2, 3, 5, 6, 8, 9… Similarly, round operation θ is busy in clock cycles 2, 5, 8 ... and idles in clock cycles 1, 3, 4, 6, 7… Finally, round operation σ is busy in clock cycles 3, 6, 9 … and idles in clock cycles 1, 2, 4, 5, 7, 8 … The involution based CED scheme in Figure 4 can be adapted to the non-pipelined KHAZAD architecture to exploit these idle clock cycles as follows (Scheme 3): Round operation γ(x) is performed in clock cycle 1. The corresponding CED operation for γ(x) i.e., γ(γ(x)) is performed in clock cycle 2 concurrent with the round operation θ(γ(x)). If x= γ(γ(x)) then no fault was detected in module γ and hence no errors are reported. The corresponding CED operation for θ(γ(x)) i.e., θ(θ(γ(x))) is performed in clock cycle 3 concurrent with round operation σ(θ(γ(x))). If γ(x)= θ(θ(γ(x))) then no fault was detected in module θ. At this point, one KHAZAD round ρ is completed only in 3 clock cycles in contrast to the 6 cycles consumed by the two other schemes described above. Now, in clock cycle 4, the corresponding CED operation for σ(θ(γ(x))) i.e., σ(σ(θ(γ(x)))) is performed concurrent with the round operation γ(y) where y is the input to the second round of the KHAZAD encryption/decryption given by y= σ(θ(γ(x))). If σ(σ(θ(γ(x))))= θ(γ(x)) then no fault was detected in module σ. The comparisons between the 3 schemes are presented in Table 2.

As explained above, Scheme 3 uses idle clock cycles to re-compute the round operations on the corresponding round outputs by feeding back the output as the input. The result obtained is compared to the original input value stored in the buffer. These two values should be equal since every round operation is an involution. If they are not equal, an error is reported. As seen from Table 2, this time redundancy based CED method (Scheme 3) entails almost no time overhead because one round operation is completed per clock cycle. Another inherent advantage of the proposed CED method is that we can detect permanent faults in the system even though the faults might not affect the output, i.e. are not activated by current inputs. Consider a situation where a faulty bit is stuck- at-1 and the output at that bit was supposed to be logic ‘1’. Now, although the output is correct, the fault in the system will be detected because the involution will not yield the correct result. This enhances the security of the implementation since any attempts to clandestinely attack the algorithm by an external agent can be detected. This also improves the overall fault coverage as well as the error detection latency. The CED architecture for Scheme 3 is shown in Figure 4 with the hardware overhead shown by the shaded blocks in Figure 4.

Table 2. Comparison between the three CED schemes during the first six clock cycles

Clock Cycle	Time redundancy based CED		
	Scheme 1 (Basic approach) Transient faults only 100% time overhead	Scheme 2 (+involution) Transient and permanent faults 100% time overhead	Scheme 3 (+involution +idle cycles) Transient and permanent faults almost 0% time overhead
1	$\gamma(x)$ of round 1	$\gamma(x)$ of round 1	γ (x) of round 1
2	$\gamma(x)$ of round 1+check	$\gamma(\gamma(x))$ of round 1+check	$\theta(\gamma(x)))$ of round 1, $\gamma(\gamma(x))$ of round 1+check
3	$\theta(\gamma(x))$ of round 1	$\theta(\gamma(x)))$ of round 1	$\sigma(\theta(\gamma(x)))$ of round 1, $\theta(\theta(\gamma(x)))$ of round 1+check
4	$\theta(\gamma(x))$ of round 1+check	$\theta(\theta(\gamma(x)))$ of round 1 +check	$\gamma(y)$* of round 2, $\sigma(\sigma(\theta(\gamma(x))))$ of round 1 +check
5	$\sigma(\theta(\gamma(x)))$ of round 1	$\sigma(\theta(\gamma(x)))$ of round 1	$\theta(\gamma(y)))$ of round 2, $\gamma(\gamma(y))$ of round 2 +check
6	$\sigma(\theta(\gamma(x)))$ of round 1 +check	$\sigma(\sigma(\theta(\gamma(x))))$ of round 1 +check	$\sigma(\theta(\gamma(y)))$ of round 2, $\theta(\theta(\gamma(y)))$ of round 2 +check

* ‘y’ is the input to round 2

An interesting observation in the figure 4 is the requirement of a second multiplexer for the CED of the σ operation. This is due to the fact that a direct involutional operation on the σ layer yields in a fault-coverage of only 50%. The σ function is a 64-bit exclusive-or operation, and applying an involution operation on an XOR module will not enable us to detect all faults. i.e, if an exclusive-or function has a stuck-at fault at one of its output bits and a faulty output is obtained because of this fault, the result of involution would in fact, be the correct input that was applied. For example, if the input to the σ layer is 0x12345678 and the round key value is 0xABCDEF01, the normal output would be 0xB9F9B979. If there is a fault in the system such that the 2nd

LSB (Least Significant Bit) of the exclusive-or output is stuck-at-1, instead of obtaining the correct result, we obtain 0xB9F9B97B which is a faulty output. But when we perform the exclusive-or operation again on this faulty output, we get back 0x12345678. In such cases, ordinary involution based CED fails. In such cases, ordinary involution based CED fails. To solve this problem, we propose the following. The operands for all the exclusive-or operators are exchanged. So, the 64-bit exclusive-or operation is now divided into two parts, the left and the right with each part consisting of 32-bit exclusive-or operations. Similarly the 64-bit exclusive-or operator is also divided into two parts, the left and the right with each part consisting of 32-bit exclusive-or operators. During the normal computation, the left part of exclusive-or operation is allocated to the left part of exclusive-or operator while the right part of exclusive-or operation is allocated to the right part of exclusive-or operator. But for the involution based CED, we interchange the operators. i.e., the right part is allocated to the left and vice-versa. Fault simulation shows the single-bit fault coverage of this scheme is 100%.

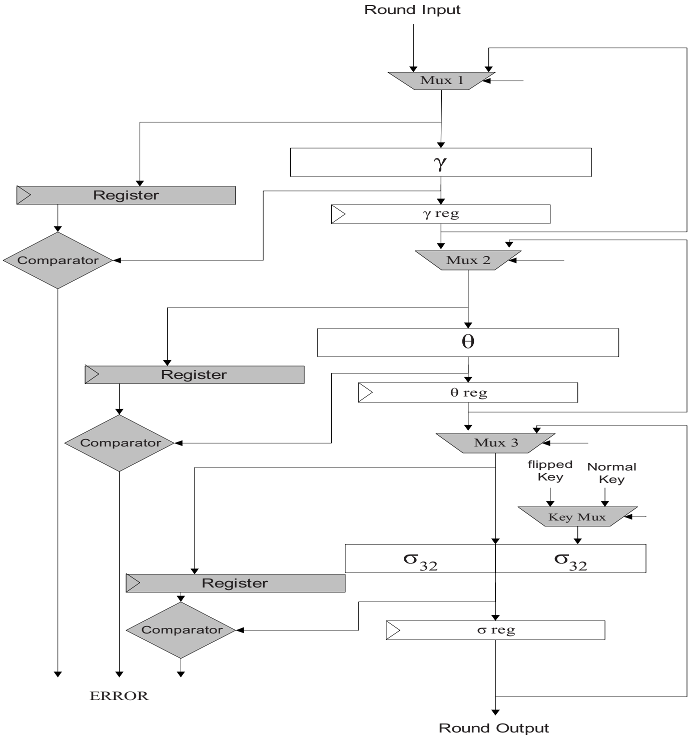

Fig. 4. KHAZAD round function ρ with CED

4 Implementation Based Validation

We implemented KHAZAD with involution based CED using IBM 0.13 micron library. The modeling of the architecture was done using VHDL, and Cadence Buildgates PKS was used to do the synthesis and place route. KHAZAD without CED datapath was also implemented using the same library and design flow. Table 3 shows the details of the overhead. The second row shows the area used by both designs. An inverter of this library takes 32 units area. The area overhead of the CED design is 23.8%. The third row shows the minimum clock period of synthesized designs. Due to the multiplexers inserted in the datapath, the clock period of CED design is 3.3%

more than the normal design. The fourth row shows that the CED design takes one more clock cycle than the normal design. This is because in the CED design, the re-computation of a round operation lags one clock cycle to the normal computation. This means that if we ignore the CED only for the σ layer in the last round, the normal architecture and the involution based CED architecture in fact take the same number of clock cycles to complete i.e., no time overhead. Finally the throughput comparison is shown in the fifth row. The throughput is calculated as the number of bits encrypted per second, i.e. the # of text / (the # of clock cycles × clock period).

Table 3. Overhead for the CED computation

	Normal	Involution based CED (scheme 3 above)	Overhead
Area	27453	34024	23.8%
Clock period	4712.69 ps	4870.99 ps	3.4%
#clock cycles	22	23	4.5%
Throughput	0.62 Gbps	0.57 Gbps	8%

5 Fault Injection Simulation

5.1 Single-Bit Faults

To check the error detection capability, we modeled our implementation using C. A stuck-at fault at a function output was injected by adding a multiplexer at the output of the function as shown in Figure 5.

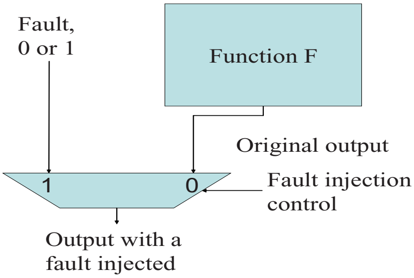

Fig. 5. Fault injection on the output of the function

By setting the fault injection control to 1, a stuck-at fault (either 0 or 1) is injected at the output of the function. Similarly, a stuck-at fault can be injected at the input of a function. Therefore, the number of connections between gates/functions gives the number of possible single-bit faults. Note that in this simulation we treat Function F as a black box and only consider the faults at inputs and outputs. If we break down the Function F into smaller components and consider the inputs and outputs of these smaller components, the number of single-bit faults is increased. In our simulation we consider the P-Box and Q-box of an S-Box, and the exclusive-or for the functions θ and σ as the black box operations. The number of single-bit faults is shown in Table 4. For example, as shown in figure 2, since an S-Box consists of three 4x4 P-Box and three 4x4 Q-Boxes, the total number of connections of an S-Box is 4 × 3 (for P-Box) + 4 × 3 (for Q-Box) + 8 (the number of inputs to S-Box) = 32. Since the γ function of KHAZAD consists of 8 S-Boxes, the total number of connections and hence the total number of single-bit faults is 256. We ran simulations for random 1.5 million inputs, and for every input we simulated all the possible single-bit faults, i.e. only one bit is stuck at 1 or 0. Table 4 shows the fault coverage. The lowest level of fault injection in the design was performed at the bit-wise exclusive-or level. As seen from the table, all the single-bit faults are detected.

Table 4. Fault Coverage of the Implementation

Function Module	# of possible single bit faults	# of inputs applied	Fault coverage
γ	256	1500000	100%
θ	1072	1500000	100%
σ	192	1500000	100%

5.2 Multiple-Bit Faults

The injection of random multiple bit faults into the system yielded an overall fault coverage of approximately 99% over a random 1.5 million input test simulation run. The reason for not achieving 100% fault coverage with multiple-bit faults is because in some exceptionally rare cases, the fault in the system gets nullified in the case of the γ and σ layers. Consider a single S-Box component of the non-linear γ layer. If we have an input of 0xD5 to the S-box, the output obtained is 0x11. After involution, we get back 0xD5, which was the original input applied. Hence, the system will not report an error. Now, if a fault occurs in the system such that the two LSBs of the P-box are stuck at logic 1, then after 0xD5 is passed through the S-Box, 0x71 is obtained, which is a faulty output. Interestingly, the involution output obtained in this case is also 0xD5. Since this value is equal to the original input, the system fails to report an error. This implies that in extremely rare cases as the one explained above, the CED method that we propose does not yield accurate results. Problems like the one described above do not affect the θ layer because θ is a diffusion layer and every bit in the output is dependent on every bit in the input. Hence, on performing involution, all multiple-bit errors are also detected, giving 100% fault coverage.

6 Conclusion

We proposed a low cost CED technique for involutional ciphers which exploits the involution property of the cipher. For the involutional cipher the proposed technique entails an additional 23.8% silicon area and degrades the throughput by less than 10%. This technique entails a 4.5% time overhead (which can be reduced to 0% if the CED is ignored only for the σ layer in the last round of encryption/decryption). The fault injection based simulation shows the proposed CED technique detects all single-bit faults and around 99% of all multiple-bit faults.

KHAZAD round key generation algorithm expands the 128-bit user key K in to nine 64-bit round keys K^0,K^1…, K^8. The round key K^r for the r^{th} round is derived as $K^r = \rho[c^r](K^{r-1}) \oplus K^{r-2}, 0 \le r \le 8$ where, K^{-2} and K^{-1} are the most and least significant parts of the key user key K and c^r is a 64-bit constant for the r^{th} round derived as $c^r{}_i = S[8r+i], 0 \le r \le 8, 0 \le i \le 7$. Since round key generation uses the KHAZAD round function ρ, the CED method proposed in this paper can be applied to detect all single-bit faults.

.

References

1. D. Boneh, R. DeMillo and R. Lipton, "On the importance of checking cryptographic protocols for faults", *Proceedings of Eurocrypt*, Lecture Notes in Computer Science vol 1233, Springer-Verlag, pp. 37-51, 1997
2. E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems", *Proceedings of Crypto*, Aug 1997
3. J. Bloemer and J.P. Seifert, "Fault based cryptanalysis of the Advanced Encryption Standard," www.iacr.org/eprint/2002/075.pdf
4. C. Giraud, "Differential Fault Analysis on AES", http://eprint.iacr.org/2003/008.ps
5. Jean-Jacques Quisquater, Gilles Piret, "A Differential Fault Attack Technique Against SPN Structures, with Application to the AES and KHAZAD,"*Fifth International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003)*, Volume 2779 of Lecture Notes in Computer Science, pages 77-88, Springer-Verlag, September 2003
6. R. Karri, K. Wu, P. Mishra and Y. Kim, "Concurrent Error Detection of Fault Based Side Channel Cryptanalysis of 128-Bit Symmetric Block Ciphers," *IEEE Transactions on CAD*, Dec 2002
7. G. Bertoni, L. Breveglieri, I. Koren and V. Piuri, "On the propagation of faults and their detection in a hardware implementation of the advanced encryption standard," *Proceedings of ASAP'02*, pp. 303-312, 2002
8. G. . Bertoni, L. Breveglieri, I. Koren, and V. Piuri, "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard," *IEEE Transactions on Computers*, vol. 52, No. 4, pp. 492-505, Apr 2003
9. Joan Daemen, Vincent Rijmen, Paulo S.L.M. Barreto, "Rijndael: Beyond the AES," *Mikulášská kryptobesídka 2002 -- 3rd Czech and Slovak cryptography workshop*, Dec. 2002, Prague, Czech Republic

10. P.S.L.M. Barreto and V.Rijmen, “The KHAZAD legacy-level Block Cipher,” *First open NESSIE Workshop*, Leuven, 13-14 November 2000
11. A. Biryukov, “Analysis of Involutional Ciphers: KHAZAD and ANUBIS,” Proceedings of the 3rd NESSIE Workshop, Springer-Verlag pp. 45 – 53
12. J. Daemen, M.Peeters, G.Assche and V.Rijmen, “The Noekeon Block Cipher,” First Open NESSIE workshop, November 2000
13. P.S.L.M. Barreto and V.Rijmen, “The ANUBIS Block Cipher,” Primitive submitted to NESSIE, September 2000,available at www.cosic.esat.kuleuven.ac.be/nessie
14. F. Standaert, G. Piret, G. Rouvroy, “ICEBERG: an involutional cipher Efficient for block encryption in Reconfigurable hardware,” *FSE 2004*, Springer-Verlag, February 2004
15. F. Standaert, G. Rouvroy, J. Quisquater, J.Legat, “Efficient FPGA Implementations of Block Ciphers KHAZAD and MISTY1,” proceedings of the 3rd NESSIE Workshop, Munich, November, 2002