A HIGH THROUGHPUT AND ERROR TOLERANT AES DESIGN

C.Thamilarasi,K.Shanmugapriya

Abstract— The Advanced Encryption Standard (AES) has been lately accepted as the symmetric cryptography standard for confidential data transmission. The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plain-text into the final output of cipher-text. All rounds consists of several processing steps including one that depends on the encryption key. A set of reverse rounds applied to transform cipher-text back into the original plain-text using the same encryption key. The proposed schemes are independent of the way the S-box and the inverse S-box are constructed. Therefore, they can be used for both S-boxes and the inverse S-boxes using lookup tables and those utilizing logic gates based on composite fields. Furthermore, for each composite field constructions, there exists eight possible isomorphic mapping. Therefore, after the exploitation of a new common subexpression elimination algorithm, the isomorphic mapping that result in the minimal implementation area cost is chosen. A high throughput hardware implementations of our proposed CFA AES S-boxes are reported. In order to avoid data corruption due to SEU’s a novel fault tolerant model of AES is presented which is based on the Hamming error correction code. This reduces the data corruption and increase the performance. Thus the data corruption due to Single Event Upset can be avoided and the performance was increased.

Key words — Advanced Encryption Standard (AES), algebraic normal form (ANF), composite field arithmetic (CFA), S-box.

I. INTRODUCTION

The Advanced Encryption Standard, in the following referenced as AES. Fifteen candidates were accepted in 1998 and based on public comments the pool was reduced to five finalists in 1999. In October 2000 one of these five algorithm was selected as the forthcoming standard: a slightly modified version of the Rijndael.

The Rijndael name is based on the name of its two Belgian inventors, Joan Daemen and Vincent Rijmen, is a Block cipher, which means that it works on fixed-length group of bits, which are called blocks.

The AES standard specifies the Rijndael algorithm, a symmetric block cipher that can process data blocks of 128 bits using cipher keys with lengths of 128, 192 and 256 bit. Rijndael are designed to handle additional block sizes and key length, however they are not adopted in this standard. Throughout the remainder of this standard, the algorithm specified will be referred to as “the AES algorithm”. The algorithms may be used with three different key lengths indicated above and therefore these different “flavors” may be referred to as “AES-128”, “AES-192”, and “AES-256”. The input and output for the AES algorithm each consist of sequences of 128 bits. These sequences will sometimes be referred to as blocks and the number of bits they contain will be referred to as their length. The Cipher Key for the ES algorithm is a sequence of 128, 192 or 256 bits. Other input, output and Cipher Key lengths are not permitted by this standard.

The bits within such sequences will be numbered starting at zero and ending at one less than the sequence length. The number i attached to a bit is known as its index and will be in one of the ranges 0 < i < 128, 0 < i < 192 or 0 < i < 256 depending on the block length and key length (specified above)

II. DERIVATION OF MULTIPLICATIVE INVERSE IN S-BOX ALGORITHM USING CFA

In AES, the encryption of the data is performed on blocks of byte, through the representation in $\mathrm{GF}(2^8)$ with the specified field polynomial $q(x)$ = $x^8$ + $x^4$ + $x^3$ + $x^2$ + 1. Every round in AES consists of four identical transformations, i.e., SubBytes, ShiftRows, MixColumns, and AddRoundKey. In a nutshell, the S-box function, which is claimed to be most resource consuming operation, involves finding a multiplicative inverse over $\mathrm{GF}(2^8)$ and followed by an affine transformation.

Finding the multiplicative inverse of elements from the higher order field is a very tedious effort. Rijmen [2] was the first to propose an alternative solutions by employing CFA [10], where one maps the element in $\mathrm{GF}(2^8)$ to its subfield that yields less complexity in the multiplicative inverse. A composite field can be built iteratively from its lower order fields; therefore the actual mathematical manipulation can be done in the lower fields rather in the original higher order field. The following summarizes the step in performing multiplicative inversion using CFA:
1) map all elements of field A to composite field B using isomorphism function; b = f(a) = δ×a;

2) compute the multiplicative inverse over B; x = b⁻¹(except if b=0, then x=0)

3) remap the computation result to A, using the inverse isomorphism function; a = f⁻¹(x) = δ⁻¹×x.

Complexity of a field is heavily dependent on several factors: field of mapping, representations of the field elements, (i.e., field polynomials and basis representations used) and isomorphic mapping chosen for the representation. Thus, one can conveniently take the advantage of the isomorphism to map a computation from one field to another to search for the most efficient implementation. Mapping a Galois Field from GF(2^8) to GF(((2^8)^2)) requires three stages of isomorphism and field polynomials which are stated (in a general form) as follows:

\[ r(y) = y^2 + G y + v \] (isomorphism for GF(2^8) / GF(2^4)) \hspace{1cm} (1)

\[ s(z) = z^2 + T z + N \] (isomorphism for GF(2^4) / GF(2^2)) \hspace{1cm} (2)

\[ t(w) = w^2 + w + 1 \] (isomorphism for GF(2^2) / GF(2)) \hspace{1cm} (3)

In this work, we present CFA for multiplicative inverse (in S-box algorithm) over the composite field GF(((2^8)^2)) with respect to both polynomial basis and normal basis. These constructions are further elaborated based on the coefficients of their respective field polynomials: (1)-(3). It is essential that we pick the coefficient that will result in the minimal arithmetic complexity.

As \( w^2 + w + 1 = 0 \) is the only irreducible polynomial of degree 2 over GF(2), there is no other candidate coefficient for (3). For (1) and (2), we need to determine all the possible coefficients of \( v, G, N \) and \( T \) in both normal and polynomial bases. In order to promote simplicity in CFA, we can either have the trace or the norm of \( r(y) \) and \( s(z) \) equal to unity but not both. To our best knowledge, previous studies attempted optimization only with traces of field polynomials equal to unity. For instance, the work of Zhang and Parhi [8] used a polynomial basis representation, while the work of Canright [7] used a normal basis representation.

Therefore, we extend these studies by choosing the norms (and) of the field polynomials to be unity for both polynomial and normal bases. Following this approach, there are two possible \( T \) values for \( s(z) \) to be irreducible over GF(2^2). Meanwhile, for \( r(y) \) to be irreducible over GF((2^8)^2), there will be eight choices for \( G \) with respect to each of the \( T \) value.

These two new constructions (hereafter, referred to as Case I and Case II) are then compared with the ones in [7] and [8]. Eventually, based on the thorough reviews of these architectures, we derived yet another new normal basis composite field AES S-box that uses a combination

<table>
<thead>
<tr>
<th>( \Delta_1 )</th>
<th>( \Delta_0 )</th>
<th>( T )</th>
<th>( \gamma )</th>
<th>( \gamma' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T</td>
<td>{01}_2</td>
<td>{0001}_2</td>
<td>( \nu \gamma^2 = [T(\Gamma_1 + \Gamma_0)^2]Z^1 + [\Gamma_0^2]Z )</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>{10}_2</td>
<td>{0010}_2</td>
<td>( \nu \gamma^2 = [T(\Gamma_1 + \Gamma_0)^2]Z^1 + [\Gamma_0]Z )</td>
</tr>
</tbody>
</table>

of norm and trace unities in different field polynomials. In total, we propose three new constructions listed as follows.

Case I Using polynomial basis representation with field polynomials’ norms equal to unity (both \( \Delta_1 \) in (1) and (2) equal to unity).
Case II Using normal basis representation with field polynomials’ norms equal to unity (both \( \Delta_0 \) in (1) and (2) equal to unity).
Case III Using normal basis representation (in (1) and \( \Delta_0 \) in (2) equal to unity).

III. HARDWARE IMPLEMENTATION OF CFA AES S-BOXES

In this section, we discuss on the actual hardware implementation of the proposed CFA AES S-box constructions. First, we manually coded the circuit using a hardware description language for all of the three proposed CFA AES S-boxes. Next, we employ ANF representation along with a strategic fine-grained pipeline registers insertion, in an attempt to validate the feasibility of the proposed compact CFA AES S-boxes in achieving high throughput hardware implementations.

In the previous studies, high speed CFA AES S-box implementations often employed pipelining scheme of which row of registers are placed in between the sub-operations such as the one reported in [9]. Consequently, the fastest speed achievable is bounded by the time required for the most complex...
sub-operations, in this case, the multiplication in GF(2^4). In order to overcome this constraint, we propose to divide the multiplication sub-operation into two parts, i.e., fine-grained pipelining.

To ensure the efficiency of the pipelined constructions, we need to first convert the complicated circuit into several logical expressions without violating the functionality of the circuit’s properties. In other words, the sub-operations over different isomorphism stages are now replaced with direct computation modules which are expressed in ANF, consisting of only AND gates and XOR gates.

Converting a CFA AES S-box into logical expressions has to be done in a way that it does not induce excessive area increase. To ensure this, we first translate all the sub-operations in GF(2^8) inversion into logical expressions individually. These sub-operations include the isomorphism function, the 4-bit adder, the square-scaler/squarer/scaler, the GF(2^4) multiplier, the GF(2^8) inverter and the inverse isomorphism function with affine transformation. Next, we group and merge some of the sub-operations into several ANF modules before inserting pipeline stages.

In addition to that, the choice of implementation platform is another important consideration factor in our pipelining scheme. For this work, we have implemented the proposed AES S-boxes on the Altera FPGA platform. On this platform, any abstract 4-input combinations will be fitted into a single logic element (LE). Consequently, the number of 4-input LE in the critical paths together with the routing delays will determine the maximum cycle period attainable. Therefore, the optimal placement of pipeline registers can be justified through the synthesis estimates obtained for the design. From the estimate obtained, we have decided to place a pipeline cut for every two LEs (best case) which results to a total of seven stages pipelining for all the three cases. The physical placement of the fine-grained pipelining stages for each our ANF-CFA AES S-boxes are as depicted in Fig. 1.

All of the proposed CFA AES S-boxes (Case I, Case II, Case III) had been implemented in Cyclone II EP2C5T144C6 and were synthesized using Quartus II. Having the architectures clocked at 100 MHz the timing analysis of the architectures were deduced using TimeQuest Timing Analyzer. In this work, two groups of hardware implementations were performed; our original CFA AES S-boxes and our seven stages pipelined ANF-CFA AES S-boxes.

The summary of the hardware requirements (total resource and power consumption) and their respective performances (speed and throughput) of all the three designs (for both groups of hardware implementations) are as tabulated in Table II.

Based on Table II, our original best case (Case III) architecture requires a total of 83 LEs, with the highest achievable frequency of 122.28 MHz, consumes total power of 33.45 mW and having a throughput of 0.98 Gbps. After the deployment of our proposed speed optimization methodologies, the architecture now requires 96 LEs, with the highest achievable frequency of 436.3 MHz, consumes total power of 34.80 mW and having a throughput of 3.49 Gbps.
V. FAULT-TOLERANT MODEL OF THE AES ALGORITHM

This section presents a novel fault-tolerant model for the AES algorithm, which is immune to radiation-induced SEUs occurring during encryption and can be used in hardware implementations on board small OE satellites. The model is based on a self-repairing EDAC scheme, which is built in the AES algorithmic flow and utilizes the Hamming error correcting code.

The proposed Hamming code-based fault-tolerant model of AES can be adapted to all the five modes of AES to correct SEUs on board. Even though the calculation of the Hamming code is carried out within the AES it does not alter any of the transformations of the algorithm and does not affect in any way the operation of AES. Also as the Hamming parity data are not sent to ground, they are not available to leak any information about the AES algorithm. Therefore the fault-tolerant AES model does not require a new cryptanalysis.

A. Model Description

The proposed fault-tolerant model is based on the single error correcting Hamming code (12,8), the simplest of the available error correcting codes. The Hamming code (12,8) detects and corrects a single bit fault in a byte and it is a good choice for satellite applications, as most frequently occurring faults in on-board electronics are bit flips induced by radiation[14]. However, the AES correction model can be extended to correct multiple bit faults by using other error correcting codes such as the modified Hamming code.

1) Calculation of Hamming Code:

The parity check bits of each byte of the S-Box LUTs are precalculated. These Hamming code bits can be formally expressed as below:

\[
\begin{align*}
h&(\text{SRD}[a]) \rightarrow h\text{RD}[a] \\
h&(\text{SRD}[a] f\{2g\}) \rightarrow h2\text{RD}[a] \\
h&(\text{SRD}[a] f\{03g\}) \rightarrow h3\text{RD}[a] (1)
\end{align*}
\]

where “a” is the state byte and “h” represents the calculation of the Hamming Code. As can be seen from (1), hRD is given by the parity check bits of the S-Box LUT SRD, h2RD is given by the parity check bits of (SRD − f02g), and h3RD is given by the parity check bits of (SRD − f03g). The procedure to derive the hRD parity bits is described below by taking one state byte a, represented by bits (b7,b6,b5,b4,b3,b2,b1,b0) for an example.

The Hamming code of the state byte a is a four-bit parity code, represented by bits (p3,p2,p1,p0), which are derived as follows:

- p3 → is parity bit group of b7,b6,b5,b4, b3, b2, b1
- p2 → is parity bit group of b7,b5,b4,b2, b1
- p0 → is parity bit group of b3, b2, b1, b0

2) Detection and Correction of Fault Using Hamming Code Bits:

The Hamming code matrix of the SubBytes transformation is predicted by referring to the hRD table. The Hamming code matrix prediction for ShiftRows involves a simple cyclic rotation of the SubBytes Hamming code bits[14]. The Hamming code state matrix for MixColumns is predicted with the help of the hRD, h2RD and h3RD parity bits and it is expressed by the equations below:

\[
\begin{align*}
h0,j &= h2\text{RD}[a0,j] h3\text{RD}[a1,j] h2\text{RD}[a2,j] h3\text{RD}[a3,j] \\
h1,j &= h\text{RD}[a0,j] h2\text{RD}[a1,j] h3\text{RD}[a2,j] h2\text{RD}[a3,j] \\
h2,j &= h\text{RD}[a0,j] h\text{RD}[a1,j] h2\text{RD}[a2,j] h3\text{RD}[a3,j] \\
h3,j &= h3\text{RD}[a0,j] h\text{RD}[a1,j] h\text{RD}[a2,j] h2\text{RD}[a3,j] (3)
\end{align*}
\]

Hamming code is predicted using the input data state to the transformation by referring to the parity check bit tables and also the parity check bits are calculated from the output of the transformation. The below Fig. 5 shows the flow chart of fault detection and correction.

![Fault detection and correction flow chart](image)

The predicted and calculated check bits are compared with detect and correct the fault as discussed below. Let the predicted check bits of the transformation input be represented by \((x3,x2,x1,x0)\) and the calculated check bits of the transformation output be represented by \((y3,y2,y1,y0)\). Once the faulty bit position is identified, the fault correction is performed by simply flipping that bit. The encryption is then continued without any interruption to the encryption process. Here we assume that the Hamming code tables will be protected from SEU by...
TABLE II
AREA REQUIREMENT, TIMING ANALYSIS, AND POWER CONSUMPTION OF FPGA IMPLEMENTATION ON CYCLONE II EP2C5T144C6 DEVICE FOR OUR ORIGINAL CFA AES S-BOXES AND OUR SEVEN STAGES PIPELINED ANF-CFA AES S-BOXES

<table>
<thead>
<tr>
<th>Hardware Performance/Requirement</th>
<th>CFA AES S-box</th>
<th>Pipelined ANF-CFA AES S-box</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Case I</td>
<td>Case II</td>
</tr>
<tr>
<td>Total L. E (4608)</td>
<td>81</td>
<td>80</td>
</tr>
<tr>
<td>Combinational Function (4608)</td>
<td>81</td>
<td>80</td>
</tr>
<tr>
<td>Logic Register (4698)</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Fmax (MHz)</td>
<td>120.83</td>
<td>121.09</td>
</tr>
<tr>
<td>Total Thermal Power Dissipation (mW)</td>
<td>34.88</td>
<td>33.31</td>
</tr>
<tr>
<td>Core Dynamic Thermal Power Dissipation (mW)</td>
<td>2.64</td>
<td>1.42</td>
</tr>
<tr>
<td>Core Static Power Dissipation (mW)</td>
<td>18.02</td>
<td>18.01</td>
</tr>
<tr>
<td>Throughput (Gbps)</td>
<td>0.96</td>
<td>0.97</td>
</tr>
</tbody>
</table>

IV. CONCLUSION

The major contribution of our work was the derivation of a new composite field AES S-box that achieves an optimally balanced construction in terms of area of implementation and critical path, compared to the previous design. The proposed fault detection and correction AES model targets the satellite application domain, however it can also be used in other applications aimed at hostile environments such as nuclear reactors, interplanetary exploration, unmanned aerial vehicles, etc. Terrestrial applications which require a high level of reliability, such as bank servers, telecommunication servers, etc. can benefit from the use of AES fault-tolerant techniques. Through the exploitation of both algebraic normal form and pipelining, our best case achieves a high throughput.

REFERENCES

First Author - C. Thamilarasi, II-Master of Engineering in VLSI DESIGN, Student at sasurie college of engineering.
Second author – K. Shanmugapriya, Assistant Professor at sasurie college of engineering.