Parallel AES Encryption Engine for Many core processor Arrays using Masked S-Box

Dhanya Pushkaran
M.Tech, VLSI and Embedded System
ECE, SNGCE
Kadayirippu, Kolenchery

Neethu Bhaskar
Assistant Professor
ECE, SNGCE
Kadayirippu, Kollenchery

Abstract— With the ever increasing growth of data communication, hardware encryption technology will become an irreplaceable safety technology. In this paper, I present a method of AES encryption and decryption algorithm with 128 bit key on an FPGA. In order to protect “data-at-rest” in memory from differential power analysis attacks with high-throughput advanced encryption standard (AES) engine with masked S-Box is proposed. By exploring different granularities of data-level and task-level parallelism, we map 2 implementations of an Advanced Encryption Standard (AES) cipher with online key expansion on a fine-grained many-core system.

Index Terms— Advanced encryption standard (AES), differential power analysis (DPA), field programmable gate array (FPGA), masking, fine-grained, many-core, parallel processor

I. INTRODUCTION

With the development of information technology, protection of information through encryption is very important in day to day life. In 2001, national institute of standard and technology replaces the data encryption standard and select the Rijndael algorithm as the advanced encryption standard(AES)[1]. AES has been used in many applications, such as secure communication system, digital video/audio recorder, RFID tags and smart cards etc. One of the main advantage of Rijndael algorithm is that it can be used for both hardware and software implementation.

To satisfy many application numerous hardware implementation of AES has been reported to achieve high throughput even though time consuming and costly. One of the main block of AES is the SubByte transformation [1] which uses S-box look-up table that is stored in memory. This data stored in storage are under the risk of information leakage in embedded applications. The differential power analysis (DPA) attack [2] was further developed as one of the most promising power analysis attacks which is related to the power consumption. So the protection of data from DPA is very important. For that instead of using S-Box lookup table masked S-Box is being implemented. We perform the masked S-Box mainly over $GF(2^4)$. Therefore, we only need to transform the input values from $GF(2^4)$ to $GF(2^n)$ and transform the output values back from $GF(2^n)$ to $GF(2^4)$ which reduces the hardware resources.

This paper present the online expansion of two type AES implementation on a fine grained many core system to achieve high performance and throughput per unit of chip.

II. AES ALGORITHM

AES is a key iterated block cipher that contains several round of transformation on the state. It is a symmetric encryption algorithm uses 128 bit key to generate output cipher text. It takes 128 bits of data block and each 128-bit data block is considered as a 4-by-4 array of bytes, called the state. The number of iteration in the AES, $N_r$, is defined by the length of the round key, which are 10 for key lengths of 128 bits.

The figure 1 shows the basic steps of AES algorithm with online key expansion. The steps include:
1. SubBytes: Nonlinear bite transformation which replace each input byte with the byte value from the substitution box. Substitution box is explained in section
2. ShiftRow: Each row of the state is left shifted according to the row number. First row no shifting is done, for 2\textsuperscript{nd} row 1 byte shifting is done and so on.

3. MixColumn: Each column of the array is considered as a polynomial over GF(2\textsuperscript{8}) and modular multiplication is done with irreducible polynomial x\textsuperscript{4}+1. The resulting polynomial is then multiplied with a fixed polynomial given in equation (1).

\[ A(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (i) \]

4. AddRoundKey: Simple bitwise XOR operation of the state with the key expanded value is done. The key expansion is done by the following steps:
   1. KeySubWord: Each byte of the key value is replaced with the values from the substitution box.
   2. KeyRotWord: Each row is done a 1 byte shifting to the left.
   3. KeyXor: Each row w[i] is XORed with the previous row w[i-1] to form a new row w'[i].

III. MASKED S-BOX

In SubByte transformation, each byte is replaced with a value from S-Box. Since there are only 256 representation of 1 byte, a lookup table of S-Box can be implemented. So the power and time consumption is reduced. But this result in differential power analysis (DPA) attach\cite{3}\cite{4}.

So here S-Box using galois field can be implemented to avoid DPA attach. It can be implemented by taking the multiplicative inverse and apply the affine transformation. But calculating the multiplicative inverse in GF(2\textsuperscript{8}) is very expensive. So masked S-Box is implemented that calculates multiplicative inverse of GF(2\textsuperscript{8}) using GF(2\textsuperscript{4}). The input byte is mapped to two elements of GF(2\textsuperscript{4}) and then find out the multiplicative inverse using GF(2\textsuperscript{4}). After that the two elements inverse mapping to GF(2\textsuperscript{8}) is done. Figure 2 shows the steps to find out the masked s-box.

A. Multiplicative inverse

For hardware implementation far better suited representation is to see field GF(2\textsuperscript{8}) as a quadratic extension of the field GF(2\textsuperscript{4}). In this case, an element a ε GF(2\textsuperscript{8}) is represented as the linear polynomial with coefficient in GF(2\textsuperscript{4})

\[ \text{Map}(a) = ah x + al, \quad a \in \text{GF}(2^8) \]

For hardware implementation, the equation for map (a) is shown in equation 2.

\[ ah x + al = \text{map}(a), \quad ah, al \in \text{GF}(2^4) \quad (2) \]

\[ aA = a1 \oplus a7, \quad aB = a5 \oplus a7, \]
\[ aC = a4 \oplus a6 \quad a0 = ac \oplus d0 \oplus a5, \]
\[ a1 = a1 \oplus a2, \quad a3 = a2 \oplus a4 \]
\[ ah = aA \oplus aC, \quad ah3 = aB \]
\[ a2 = aA, \quad ah0 = ac \oplus a5, \quad ah1 = aA \oplus a4 \]
\[ a3 = aB \oplus a2 \oplus a3, \quad ah2 = aB \oplus a2 \oplus a3, \]

After finding out the multiplicative inverse in GF(2\textsuperscript{4}), two term polynomial ah x + al converted back to element in GF(2\textsuperscript{8}). The equation for map\textsuperscript{-1} is shown in equation 3.

\[ \text{map}\textsuperscript{-1}(ah x + al) = a, \quad ah, al \in \text{GF}(2^4), \quad a \in \text{GF}(2^8) \quad (3) \]

\begin{align*}
    aA &= a1 \quad ah3, \quad aB &= ah0 \quad ah1 \\
    a0 &= a0 \oplus ah0, \quad a1 &= aB \oplus ah3, \\
    a2 &= aA \oplus aB, \quad a3 &= aB \oplus ah1 \oplus ah2, \\
    a4 &= aA \oplus aB \oplus al3, \quad a5 &= aB \oplus al2, \\
    a6 &= aA \oplus al2 \oplus al3 \oplus ah0, \quad a7 &= aB \oplus al2 \oplus ah3
\end{align*}

Multiplication in GF(2\textsuperscript{4}) corresponds to multiplication of polynomial modulo an irreducible polynomial of degree 4. The irreducible polynomial is given by

\[ M(x) = x^4 + x + 1 \]

For hardware implementation, byte multiplication is given in equation 4.
\[ q(x) = a(x) \cdot b(x) \mod m(x), \quad a(x), b(x), q(x) \in GF(2^4) \quad (4) \]

\[ aA = a_0 \oplus a_3 \quad \text{and} \quad aB = a_2 \oplus a_3 \]

\[ q_0 = a_0b_0 \oplus a_3b_1 \oplus a_2b_2 \oplus a_1b_3 \]

\[ q_2 = a_2b_2 \oplus a_1b_1 \oplus a_0b_0 \oplus a_3b_3 \]

\[ q_3 = a_3b_3 \oplus a_2b_2 \oplus a_1b_1 \oplus a_0b_0 \oplus a_3b_3 \]

The multiplicative inverse can be found out using extended Euclidean algorithm. It can be derived by solving the equation \( a(x) \cdot a^{-1}(x) \mod m_4(x) = 1 \). Solution is shown in equation 5.

\[ q(x) = a(x)^{-1} \mod m_4(x), \quad q(x), a(x) \in GF(2^4) \quad (5) \]

\[ aA = a_1 \oplus a_2 \oplus a_3 \oplus a_1a_2 \]

\[ q_0 = aA \oplus a_0 \oplus a_2 \oplus a_1a_2 \oplus a_0a_1a_2 \]

\[ q_1 = a_0a_1 \oplus a_0a_2 \oplus a_1a_2 \oplus a_3 \oplus a_1a_3 \oplus a_0a_1a_3 \]

\[ q_2 = a_0a_1 \oplus a_2 \oplus a_0a_2 \oplus a_3 \oplus a_0a_3 \oplus a_0a_2a_3 \]

\[ q_3 = aA \oplus a_0a_3 \oplus a_1a_3 \oplus a_2a_3 \]

\[ q = \text{aff\_tran}(a) \quad \text{and} \quad q = \text{aff\_trans}^{-1}(a) \quad (6) \]

\[ aA = a_1 \oplus a_3 \]

\[ aB = a_2 \oplus a_3 \]

\[ aC = a_4 \oplus a_5 \]

\[ aD = a_6 \oplus a_7 \]

\[ q_0 = a_0 \oplus a_3 \oplus a_1a_2 \]

\[ q_1 = a_3 \oplus a_1a_2 \]

\[ q_2 = a_1a_2 \oplus a_1a_3 \oplus a_0a_1a_3 \]

\[ q_3 = aA \oplus a_0a_3 \oplus a_1a_3 \oplus a_2a_3 \]

B. Affine Transformation

Affine transformation is given by, \( A' = M(a) \cdot X \oplus [v] \)

Where \([v] = x^7 + x^6 + x^2 + x \) and \( m(a) = x^7 + x^4 + x^3 + x + 1 \).

The equation for hardware implementation is given in equation 6.

IV. Fine Grained Many Core Architecture

The performance of architecture is roughly proportional to the square root of its complexity. So as the complexity is decreased the performance will increase but it may increase the logical area. So a many core architecture can perform better with complexity. That is instead of using single complicated core many cores are used, which increases the performance.

V. AES Implementation

In this paper I present two different AES implementations with online key expansion and the throughput of the design is measured.

A. One task one processor (OTOP)

Each step in the AES algorithm is considered as a task as shown in the dataflow diagram in figure 3. Each task is mapped on to one processor in many core processors. So we call this implementation One Task One processor. For single iteration about 10 cores are required and after completing first iteration the same cores are used for the following iteration.

B. Loop unrolled nine times

To enhance the throughput, new design is implemented as shown in figure 4. Here each loop is done by another set of core. So loop unrolled nine times break the data dependency and work on multiple data block. About 60 cores are required to implement this design.

VI. Result

I have implemented the proposed design with hardware description language which is synthesized using Xilinx ISE 14.1 and ported the design to Spartan-6 LX45 FPGA. The table 1 shows the throughput obtained from the two designs. From this table it is clear that the loop unrolled nine times design is very much faster than one task one processor design.
TABLE I.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>One Task One Processor</td>
<td>1.98 Gbps</td>
</tr>
<tr>
<td>Loop Unrolled Nine Times</td>
<td>85.15 Gbps</td>
</tr>
</tbody>
</table>

VII. CONCLUSION

Secure “data-at-rest” and enhance the throughput are the important factor for large data transformation system. So, modern systems shift the data encryption from a software platform to a hardware platform. But the hardware based encryption still facing the possibility of DPA attacks. In this case, an AES with masked S-box has been proposed to resist the DPA attack with acceptable area on FPGA. The proposed masked -Box needs to map the input values from GF(2^8) to GF(2^4) at the beginning of the operation and map the result back from GF(2^4) to GF(2^8) once at the end of the operation. Which reduce about 20% area resources. By implementing the design using Loop unrolled nine times not only protect from DPA attack but also increases the throughput.

ACKNOWLEDGMENT

I would like to express my heartfelt gratitude and thanks to my beloved guide Ms. Neethu Bhaskaran, Assistant Professor, Dept. of Electronics and Communication Engineering, SNGCE Kadayiruppu, whose guidance I could complete the thesis work to the level I had planned, for the regular reviews and suggestions. It gives me great pleasure to thank her for the conviction she brought in into selecting the topic of work, and the technical and literary guidance she imparted through the different stages of its execution.

REFERENCES