An Efficient Hardware FPGA Implementation of AES-128 Cryptosystem Using Vedic Multiplier and Non LFSR

Anjali.L
Dept.Of VLSI and Embedded Systems
Musaliar College of Engineering &Technology, Kerala

ABSTRACT
We propose an efficient hardware architecture design & implementation of Advanced Encryption Standard (AES)-Rijndael cryptosystem. The AES algorithm defined by the National Institute of Standard and Technology (NIST) of United States has been widely accepted. The cryptographic algorithms can be implemented with software or built with pure hardware. However Field Programmable Gate Arrays (FPGA) implementation offers quicker solution and can be easily upgraded to incorporate any protocol changes. This contribution investigates the AES encryption and decryption cryptosystem with regard to FPGA and Very High Speed Integrated Circuit Hardware Description language (VHDL). Optimized and Synthesizable VHDL code is developed for the implementation of both 128-bit data encryption and decryption process. Here Direct method of computing the discrete linear convolution of finite length sequences is used. The approach is easy to learn because of the similarities to computing the multiplication of two numbers by a pencil and paper calculation. Multipliers are basic building blocks of convolver. Since it dominates most of the execution time, for optimizing the speed, 4×4 bit Vedic multipliers based on Urdhva Tiryagbhyam sutra are used. Convolver has delay of 17.996 ns when implemented on 90 nm process technology FPGA. It also provides necessary modularity, expandability, and regularity to form different convolutions for any number of bits. AES key expansion can be done by LFSR it is more secure. The coding is done in VHDL (Very High Speed Integrated Circuits Hardware Description Language) for the FPGA, as it is being increasingly used for variety of computationally intensive applications. Simulation and synthesis is done using Xilinx 14.2i.

Keywords: Advanced Encryption Standard (AES), Cryptography, Decryption, Encryption, Rijniael, LFSR, Urdhva Triyakbhyam Sutra ,Vedic Mathematics.

I. INTRODUCTION
In cryptography, encryption is the process of translation of data into a secret code. It is unreadable by anyone except those possessing the key. Encryption is important to secure data such that it is received with full integrity at the receiver end. There are two types of encryption called asymmetric encryption (public-key encryption) and symmetric encryption. Decryption is the process of converting the data back into its original form. Symmetric cryptosystems such as Data Encryption Standard (DES), 3 DES, and Advanced Encryption Standard (AES), uses an identical key for the sender and receiver; both to encrypt the message text and decrypt the cipher text. Asymmetric cryptosystems such as Rivest-Shamir-Adleman (RSA) & Elliptic Curve Cryptosystem (ECC) uses different keys for encryption and decryption. Symmetric cryptosystem is more suitable to encrypt large amount of data with high speed.

To replace the old Data Encryption Standard, in Sept 12 of 19997, the National Institute of Standard Technology (NIST) required proposals to what was called Advanced Encryption Standard (AES). Many algorithms were presented originally with researches from 12 different nations. Fifteen algorithms were selected to the Round one. Next five were chosen to the Round two. Five algorithms finalized by NIST are MARS, RC6, RIJNDAEL, SERPENT and TWOFISH. On October 2nd 2000, NIST has announced the Rijndael algorithm is the best in security, performance, efficiency, implement ability, & flexibility. The Rijndael algorithm was developed by Joan Daemen of Proton World International and Vincent Rijmen of Katholieke University at Leuven.

AES encryption is an efficient scheme for both hardware and software implementation. As compare to software implementation, hardware implementation provides greater physical security and higher speed. Hardware implementation is useful in wireless security like military communication and mobile telephony where there is a greater emphasis on the speed of communication. Most of the work has been presented on hardware implementation of AES using FPGA .in our proposed paper introduce a vedic multiplier and a linear feedback shift register. The worst case propagation delay in the Optimized Vedic multiplier case is 31.526ns. automatically generating the linear by the LFSR.

www.ijsret.org
A number of AES parameters depend on the key length. For example, if the key size used is 128 then the number of rounds is 10 whereas it is 12 and 14 for 192 and 256 bits respectively. At present the most common key size likely to be used is the 128 bit key. This description of the AES algorithm therefore describes this particular implementation.

Rijndael was designed to have the following characteristics:

- Resistance against all known attacks.
- Speed and code compactness on a wide range of platforms.
- Design Simplicity

II. AES ALGORITHM OVERVIEW

The AES accepts a 128-bit plain text, and produces a 128-bit cipher text under the control of a 128, 192, or 256-bit secret key. It is a Substitution-Permutation Network design with a single collection of steps called a round. For the AES algorithm, the number of rounds to be performed during the execution of the algorithm is dependent on the key length.

Table 1: Block- Round Combinations

<table>
<thead>
<tr>
<th>Bit pattern</th>
<th>Key Length (NK Words)</th>
<th>Block Size (NB Words)</th>
<th>No of Rounds (NR Words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>4</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>AES-192</td>
<td>6</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>AES-256</td>
<td>8</td>
<td>4</td>
<td>14</td>
</tr>
</tbody>
</table>

that are repeated 10, 12, or 14 times (depending on the key length) to map the plain text to cipher text each round uses its own 128-bit round key, which is derived from the supplied secret key through a process known as a key schedule. It distributes the entropy of the key across each of the round keys. If that entropy is not spread properly, it causes all kinds of trouble such as equivalent keys, related keys, and other similar distinguishing attacks.

A single round of AES consists of four Transformations namely Sub Bytes, Shift Rows, Mix Columns and Add Round Key shown in Fig 1. The four different transformations are described in detail below.

1) Sub Bytes Transformation: It is a non-linear substitution of bytes that operates independently on each byte of the State using a substitution table (S box). This S-box which is invertible is constructed by first taking the multiplicative inverse in the finite field GF (2^8) with irreducible polynomial m(x) = x^8 + x4 + x^3 + x + 1. The element {00} is mapped to itself. Then affine transformation is applied (over GF (2)).

2) Shift Rows Transformation: Cyclically shifts the rows of the State over different offsets. The operation is almost the same in the decryption process except for the fact that the shifting offsets have different values.

3) Mix Columns Transformation: This transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over GF (2^8) and multiplied by modulo x^4 + 1 with a fixed polynomial a(x) = {03} x^3 + {01} x^2 + {02} x.

4) Add Round Key Transformation: In this transformation, a Round Key is added to the State by a simple bitwise XOR operation. Each Round Key consists of Nb words from the key expansion. Those Nb words are each added into the columns of the State. Key Addition is the same for the decryption process.

5) Key Expansion: Each round key is a 4-word (128-bit) array generated as a product of the previous round key, a constant that changes each round, and a series of S-Box lookups for each 32-bit word of the key. The Key schedule Expansion generates a total of Nb (Nr + 1) words.

The decryption process is direct inverse of the encryption process. All the transformations applied in encryption process are inversely applied to this process. Hence the last round values of both the data and key are first round inputs for the decryption process and follow in decreasing order.
III. DISADVANTAGES

The LUT is used in encryption and decryption using AES algorithm. It is used in the mix columns block where Galois field multiplication is performed. Galois field multiplication is a finite field multiplication. Here the finite field is limited to $2^8$. There are two "look up" tables. "Look up" tables are pre-computed tables that are used to compute multiplication in Galois field($2^8$) for any input value.

Disadvantages of LUT method

- More memory is required to store the two look up tables.
- Computation time is more since we refer two tables for calculating one product.

IV PROPOSED APPROACH

Vedic mathematics is part of four Vedas (books of wisdom). It is part of Sthapatya-Veda (book on civil engineering and architecture), which is an upa-veda (supplement) of Atharva Veda. It gives explanation of several mathematical terms including arithmetic, geometry (plane, co-ordinate), trigonometry, quadratic equations, factorization and even calculus. His Holiness Jagadguru Shankaracharya Bharati Krishna Teerthaji Maharaja (1884-1960) comprised all this work together and gave its mathematical explanation while discussing it for various applications.

The work presented here, makes use of Vedic Ma “Vertically and Crosswise mathematics for Algorithm” multiplication is used to of develop Vedi digital multiplier architecture. This looks quite similar to the popular array multiplier architecture. This Sutra shows how to handle multiplication of a larger number (N x N, of N bits each) by breaking it into smaller numbers of size (N/2 = n, say) and these smaller numbers can again be broken into smaller numbers (n/2 each) till we reach multiplicand size of (2 x 2). Thus, simplifying the whole multiplication process.

Let number1 = (10)$_2$ and number2 = (10)$_2$. Their multiplication using Urdhva Tiryagbhyam is shown fig. 2.
In a regular Urdhwa sutra multiplier, partial products are obtained by ANDing the inputs and adding the bi-products. In our approach, we replace addition with XORing. The product obtained is limited to 8 bits using conventional methods of Galois field restriction. The addition of XOR gates reduces the complexity of the existing Vedic Mathematics Architecture.

V. EXPERIMENTAL RESULTS

All the results are based on simulations from the Xilinx ISE tools, using Test Bench Waveform Generator. All the individual transformation of both encryption and decryption are simulated using FPGA SPARATAN 3E family.

A. Simulation Results:

1) Encryption Process (Cipher):

AES block length/Plane Text = 128bits (Nb=4)  
Key length = 128 bits (Nk =4); No. of Rounds = 10(Nr =10)

```
: 00112233445566778899aabbccdd
eeff
```

Plane Text: 00102030405060708090a0b0c0d

Key: 000102030405060708090a0b0c0d0e0f

Output/Cipher Text: 69c4e0d86a7b0430d8c7bd8070b4c55a

Figure 4 represents the waveforms generated by the 128-bit complete encryption Process. The inputs are clock1 & clock2, Active High reset, 4-bit round, and 128-bit state & key as a standard logic vectors, whose output is the 128-bit cipher (encrypted) data.

2) Decryption Process (Inverse Cipher):

AES block length/Cipher Text = 128bits (Nb=4)

Key length = 128 bits (Nk =4); No of Rounds = 10(Nr =10)

```
Input /Cipher Text: 69c4e0d86a7b0430d8c7bd8070b4c55a
Key: 000102030405060708090a0b0c0d0e0f
Output/Plain Text: 00112233445566778899aabbccdd
eeff
```

Figure 5 represents the waveforms generated by the 128-bit complete decryption Process. The inputs are clock1 & clock2, Active High reset, 4-bit round, and 128-bit state & key as standard logic vectors, whose output is the 128-bit plain text (decrypted data).

VI. LFSR

LFSRs (linear-feedback shift registers) find extensive use in cryptography. For example, the cryptographic algorithms in the GSM (Global System for Mobile communications) mobile-phone system rely on the use of LFSRs. An LFSR comprises a register containing a sequence of bits and a feedback function. In general, this function is an XOR (exclusive-OR) operation on certain bits in the register. The list of these bits is a "tap sequence." You use an LFSR to generate a pseudorandom sequence of bits that undergo an XOR operation. The XOR result then connects to the input of the LFSR. Repeating the process at the decoder side returns the original sequence of bits. This presents the encryption and decryption process. To generate a pseudorandom sequence, you load the register with a nonzero content, and the software then computes the XOR of the taps and shifts all bits in the register one bit to the left. Finally, the routine inserts the results of the XOR operation in the rightmost position.

The unusual sequence of values generated by an LFSR can be gainfully employed in the encryption (scrambling) and decryption (unscrambling) of data. A stream of data bits can be encrypted by XOR-ing them with the output from an LFSR (Fig 6).
The stream of encrypted data bits seen by a receiver can be decrypted by XOR-ing them with the output of an identical LFSR. This is obviously a very trivial form of encryption that’s not very secure, but it's "cheap-and-cheerful" and may be useful in certain applications.

VII. CONCLUSION

This paper presents a low area, cost effective AES cipher for encryption /decryption using a 128 bit iterative architecture. In this work, the amount of hardware resources has been optimized with respect to various proposed designs on alternative platforms (Spartan 3, and VirtixE). The cipher has been synthesized using Xilinx 14.2.

The architecture needs fewer logic cells than other ciphers and uses a few memory blocks as possible. Future work should concentrate on speed performance. Now the LFSR gives more secure. The look up table approach & our proposed architecture was implemented on Spartan 3e series of FPGA and the synthesis results for the same is as shown below. It is clearly observed the LUT approach is 3 ns faster than our proposed architecture. Yet, it is not preferable since more than 100% of the FPGA resources are utilized. On the other hand, the Vedic Mathematics approach occupies only 6% of area when implemented on a Spartan 3e series of FPGA.

REFERENCES


