International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015
ISSN: 2395-3470
www.ijseas.com

Optimized AES Algorithm Using FeedBack Architecture

Chintan Raval[1], Maitrey Patel [2], Bhargav Tarpara[3]
[1, 2,] Pursuing M.Tech., VLSI, U.V.Patel college of Engineering and Technology, Kherva, Mehsana, India
[3]Verification Technical Assistant, eiTRA, Ahemdabad,India

***Abstract*— A new Feedback based design technology scheme of the AES-128 (Advanced Encryption Standard, with 128bit-key) algorithm is proposed in this paper. For getting the speed of encryption block, decryption blocks and as well as the key generation block, the Feedback architecture is applied and the mode of data transmission is modified in this design so that the chip size can be decreased. The 128-bit plaintext and the 128bit-initial key, as well as the 128bit- output of cipher text, are all divided into four 32bit-consecutive units respectively managed by the clock. This new program can significantly decrease quantity of chip pins and effectively optimize the area of chip[4].**

***KeyWords*— VHDl, DES, 3DES, Area Optimization, Encryption, Decryption, Key Generation, PipeLine Arch, FeedBack Arch.**

I. Introduction

The 3DES algorithm is subjected to more security than any other encryption algorithm over a long period of time and no effective cryptanalytic attack based on algorithm rather than brute force has been found. Accordingly, there is a higher level of confidence that 3-DES is very much resistant to cryptanalysis. If security is only consideration then 3-DES would be an appropriate choice for a standardized encryption and decryption algorithm for years to come[1].

The main principle drawback of 3-DES is that the algorithm is relatively sluggish in software. The original DES was designed in mid 1970s for hardware implementation and does not produce efficient software code. In 3DES it has 3 times as many rounds as DES is correspondingly slower. secondary drawback is that both DES and 3-DES are works on 64-bit block size. Just For the reasons of both efficiency and security a larger block size is desirable[1].

Just because of these drawbacks, 3-DES is not reasonable algorithm for long time use. As a replacement algorithm, NIST (National Institute Of Standard And Technology) in 1997 issued a call for proposals for a new Advanced Encryption Standard (AES) which is having security strength equal or better than 3-DES and significantly improved in working. In addition to these general requirements NIST specified that AES must be symmetric block cipher with a block length of 128 bits and support for key lengths of 128-bits,192-bits and 256-bits[1].

In first round of evaluation, 15 proposed algorithms were selected. In second round 5 were selected. NIST finished its selection process and published a final standard white paper (FIPS 197) in November 2001. NIST selected Rijndael as the proposed AES algorithm. The two scientist who developed and submitted Rijndael for the AES are both cryptographers from Belgium: Dr.Joan Daemen and Dr.Vincent Rijmen[1].

The choice of platform, software, ASIC or FPGA, is driven by several points and aspects, such as AES algorithm performance, cost, resources and flexibility. Although ASIC has high performance and lower the unit cost, it has no flexibility at all. While software has the most flexibility among all, the performance is very lower.

II. AES Block Diagram & Description

International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015
ISSN: 2395-3470
www.ijseas.com

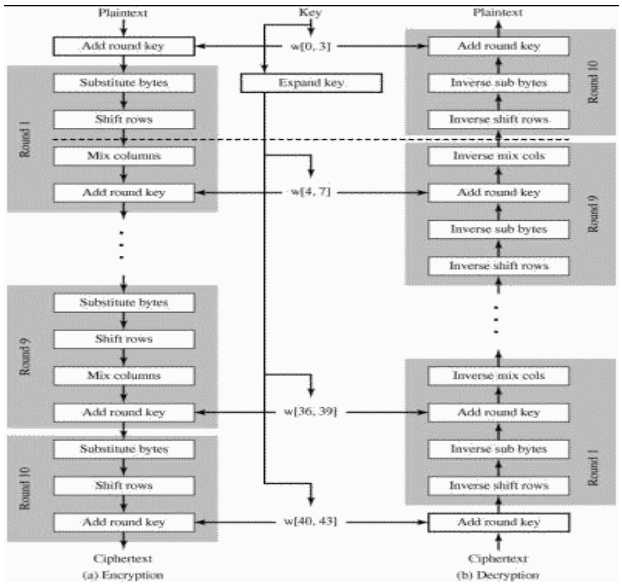

Figure 1:AES Algorithm Block Diagram

As shown in Fig.1, The algorithm consist of three parts. Encryption, Decryption and Key Generation Parts.

A. Encryption

The Encryption process consist of #10 rounds of transformation. But, the input given to the round #1 will be EX-OR of 128-bits plain text and 128-bit key. Each round from round #1 to round #9 again consist of four different transformation internally except 10th round. These transformation are namely SubBytes operation, ShiftRows operation, MixColumns operation and AddRound key transformation. The result of one transformation is given as input to the next transformation and so on as shown in the figure. Round #10 is slightly different from the other rounds in that the MixCloumns transformation is removed. The output of each round is feedback to the next round as input. Total 10 rounds of transformations produces encrypted output called cipher text[4].

B. Decryption

The Decryption process is reverse process of Encryption and it also consists of ten rounds of transformation. But the input to round #1 will be EX-OR of cipher text and key 10. Each round from round #1 to round #9 again consists of four different transformation internally except #10 round. These transformations are namely InvSubBytes operation, InvShiftRows operation, InvMixColumns operation and InvAddRound key transformation. Round #10 is slightly different from other rounds in that the InvMixColumns transformation is removed. Like in Encryption output of one round is fed as input to the next round. The usage of keys also reverse in this case i.e. round #1 uses key 9 and round #2 uses key 8 and so on. Total #10 rounds of transformations produces decrypted output i.e. Plain Text[4].

C. Key Generation

This part takes a 128-bit key. The 128-bit key is divided into four words represented in word[0,3]. These four words are used as it is for encryption initially before starting of a round and for round #10 of decryption part. These 4 words are used for producing a new key represented by word[4,7]. Which is used for round #1 of encryption and round #9 of decryption. These 4 words represented by word[4,7] which is key 1, is used for producing key2 represented by word[8,11]. Like this it will generate total 40 words i.e. 10 keys[4].

III. AES Specification

For the AES algorithm, the number of rounds to be performed during the execution of the algorithm is dependent on the key length. The number of rounds is represented by Nr, where Nr = 10 when Nk = 4, Nr = 12 when Nk = 6, and Nr = 14 when Nk = 8. **The only Key-Block-Round combinations that conform to this standard are given in Fig. 2.**

	Key Length *(Nk words)*	Block Size *(Nb words)*	Number of Rounds *(Nr)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Figure 2: Key-Block-Round Combinations.

A. Key Generation

The AES algorithm takes the cipher key K, and performs a Key Expansion routine process to generate a keys. The Key Expansion generates a total of Nb (Nr + 1) words. The algorithm requires an initial set of Nb words, and each of the Nr rounds require Nb words of key data. The resulting key schedule consists of a linear array of 4-byte words, denoted [wi], with I in the range $0 \leq I < Nb(Nr + 1)$.

The AES algorithm takes the cipher key K, and performs a Key Expansion routine process to generate keys. The Key Expansion process

generates a total of Nb (Nr + 1) words. The algorithm requires an initial set of Nb words, and each of the Nr rounds require Nb words of key data. The resulting key schedule contains a linear array of 4-byte words, denoted [wi], with I in the range 0≤ I <Nb(Nr + 1).The key expansion operation has Temp, Subword(), Rotword(), Rcon[i],w[i - Nk] stages. The following example will show its detail.

Cipher Key =

2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

for ***Nk*** = 4, which results in

*w*0 = **2b7e1516,** *w*1 = **28aed2a6,** *w*2 = **abf71588,** *w*3 = **09cf4f3c**

i (dec)	temp	After RotWord()	After SubWord()	Rcon[i/Nk]	After XOR with Rcon	w[i-Nk]	w[i]= temp XOR w[i-Nk]
4	09cf4f3c	cf4f3c09	8a84eb01	01000000	8b84eb01	2b7e1516	a0fafe17
5	a0fafe17					28aed2a6	88542cb1
6	88542cb1					abf71588	23a33939
7	23a33939					09cf4f3c	2a6c7605

Figure 3:Key Generation Example.

B. Encryption (cipher)

The Encryption consists of 4 different transformation. The individual transformations - SubBytes(), ShiftRows(), MixColumns(), and AddRoundKey() process the State and are described in the following division. All Nr rounds are identical with the exception of the final round, which does not include the MixColumns()transformation.

1. SubBytes( ).

The SubBytes() transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box). This S-box is nonvertible, is constructed by composing two transformations:

The following **Figure.4.** illustrates the effect of the **SubBytes()** function on the state.

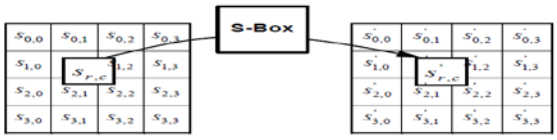

Figure 4: Subbytes applies the s-box to each of the state.

1. ShiftRows( ).

In the ShiftRows() transformation, the bytes in the last 3 rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row,r=0,is not shifted. Specifically, the **ShiftRows( )** transformation proceeds as follows:

$$shift(1,4)=1;\ shift(2,4)=2;\ shift(3,4)=3.$$

Figure 5 illustrates the **ShiftRows()**transformation.

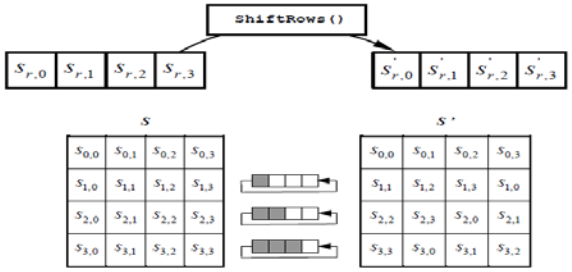

Figure 5: Shiftrows cyclically shifts last three rows.

2. MixColumns( ).

The MixColumns() transformation operates on the State column-by-column, taking each column as a four-term polynomial as described previously. The columns are treated as polynomials over GF(28) and multiplied modulo x 4 + 1 with a fixed polynomial a(x), given by

a(x)={03}x3+{01}x2+{01}x+{02}.

Let,

S'(x) = a(x) XOR s(x).

As a result of this operation, the 4-bytes in a column are placed by the following:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

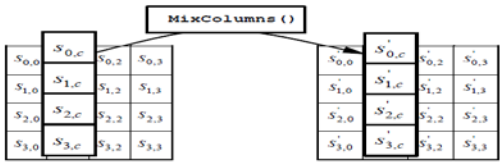

Figure 6: MixColumns operates on state column by column.

3. AddRoundKey( ).

In the AddRoundKey() transformation operation,a Round Key is added to the State by

International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015
ISSN: 2395-3470
www.ijseas.com

a simple bitwise XOR operation. Each Round Key consists of Nb words from key schedule. Those Nb words are added individually into the columns of the StateIn the AddRoundKey() transformation operation,a Round Key is added to the State by a simple bitwise XOR operation. Each Round Key consists of Nb words from key schedule. Those Nb words are added individually into the columns of the State, such that

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*Nb+c}] \quad \text{for } 0 \le c < Nb,$$

The action of this transformation is illustrated in Fig. 7, where *l = round * **Nb***.

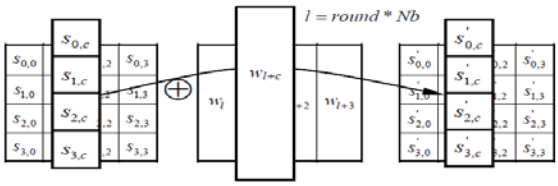

Figure 7: Addroundkey XORS each column of the state with a word from the key schedule.

2. *Decryption (Inverse Cipher)*

The Cipher transformations operation can be inverted and then implemented in reverse order to produce a straightforward Inverse Cipher for the AES algorithm decryption block. The individual transformations used in the Inverse Cipher -InvShiftRows(), InvSubBytes(), InvMixColumns(), and InvAddRoundKey() – process the State and are described in the following subsections.

1. InvShiftRows( ).

InvShiftRows() is the inverse of the ShiftRows() transformation operation. The bytes in the last 3 rows of the State are cyclically rotate over different numbers of bytes (offsets). The first row, r = 0, is unshifted. The bottom 3 rows are cyclically rotated by Nb - shift(r, Nb) bytes, where the shift value shift(r,Nb) depends on the row number, Specifically**,** the **InvShiftRows()** transformation proceeds as follows: $S'_{r,(c+Shift(r,Nb))\bmod Nb} = s_{r,c}$ for $0<r<4$ and $0 \le c < Nb$.

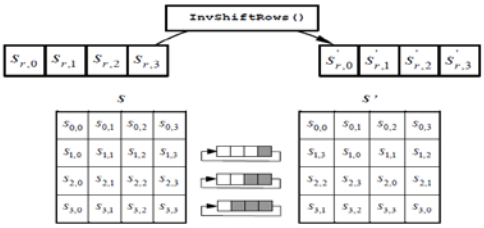

Figure 8: InvShiftRows cyclically the last three rows in a state.

2. InvSubBytes( ).

InvSubBytes() is the inverse of the byte substitution transformation, in which the inverse S-box is applied to each byte of the State. This is obtained by applying the inverse of the affine transformation followed by taking the multiplicative inverse in GF(2^8).

3. InvMixColumns( ).

InvMixColumns() is the inverse of the **MixColumns()** function. **InvMixColumns()** 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 modulo $x^4 + 1$ with a fixed polynomial $a^{-1}(x)$, given by

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}.$$

4. InvAddRoundKey( ).

It is the same **AddRoundKey** function of encryption.

IV. ALGORITHM IMPLEMENTATION

A. Implementation of KEY GENERATION Module.

This module takes in 128 bit key and a round constant as input and generates a new 128 bit key. And this makes use of S-BOX table to generate a new key. The following RTL schematic gives the detail of this entity.

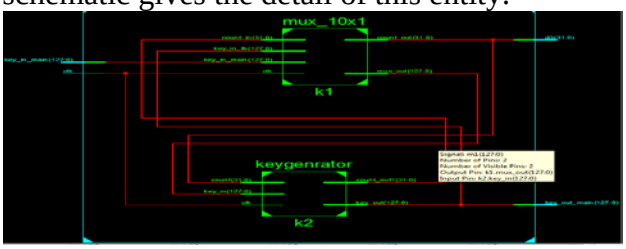

Figure 9: RTL Schematic of internally KEY Generation Round .

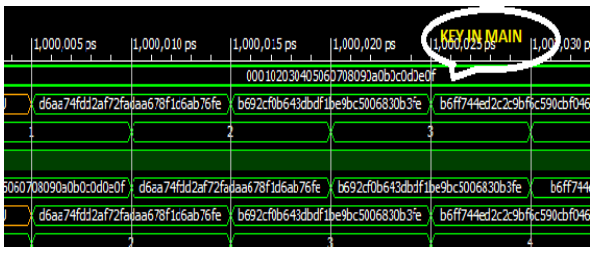

Figure 10: OUTPUT result of KEY Generation Unit .

B. Implementation of Encryption Module

The encryption module contains ten round and each round has four stages namely, SubBytes, ShiftRows, MixColumns and AddRoundKey except 10th round which has MixColumns removed. Here we are combining SubBytes and ShiftRows into a single stage to reduce device utilization. The important part at this stage is implementing **s-box**. MixColumns and AddRoundKey transformations and can be implemented directly using VHDL Behavioral codes.

1. Implementing a Round.

A single round consists of four transformations namely SubBytes, ShiftRows, MixColumns and AddRoundKey. First two transformations are combined into a single component. Hence a single round will have 3 components. The entity defined for a round takes in 128bit key for that particular round and 128bit data which is either output of a previous round if the round is other than 1st round or it would e EX-OR of key and plain text if it is Round #1. The following RTL schematic is generated for this entity

2. Implementing Complete Encryption.

The entity defined for encryption takes in 10 round keys, plain text and key as inputs and produces a cipher text as output, all of them having length of 128 bits. The RTL schematic for this entity is shown below.
The following RTL schematic is generated for this implementation[5].

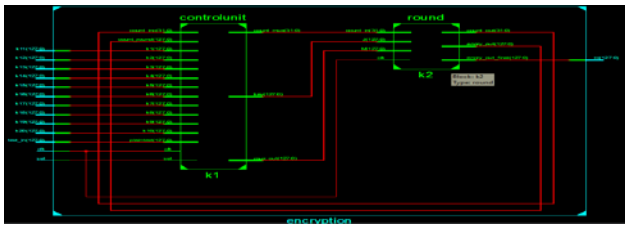

Figure 11: RTL schematic for encryption implementation(using feedback arch).

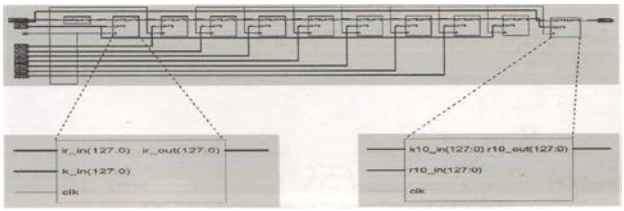

Figure 12: RTL schematic for encryption implementation(using pipeline arch).

3. Implementation of DECRYPTION module.

The decryption module also contain ten rounds and each round has four stages namely, InvSubBytes, InvShiftRows, InvMixColumns and InvAddRoundkey except 10th round which has InvMixColumn removed. The important part at this stage is implementing INVERSE S-OX. The internal architecture of decryption module is similar to the Encryption module. The differences are highlighted below.

1. Implementing Rounds

The architecture of a single round is similar to that of encryption containing four transformations. Round #10 differs in that InvMixColumns transformation is removed. The implementation of single round is done in much the same way as that of encryption round using VHDL.

2. Implementing Complete Decryption.

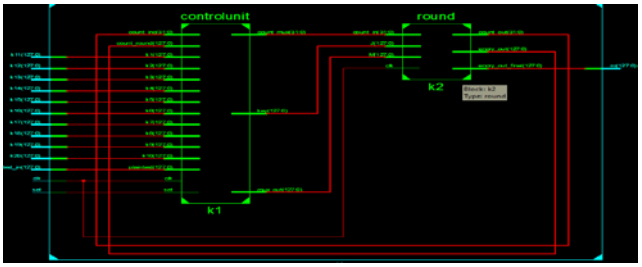

Figure 13: RTL schematic for decryption implementation (using feedback arch).

The entity defined for decryption takes in 10 round keys,output of encryption module i.e. cipher text and keys as inputs and produces a plain text as output, all of them having length of 128 bits. The RTL schematic for this entity is shown below.

V.COMPARISON BASED ON TWO DISTINCT ARCH.

The following chart compares the previous Pipeline Architecture and Latest Feedback Architecture. In which it compares the resources used by these both Architecture.

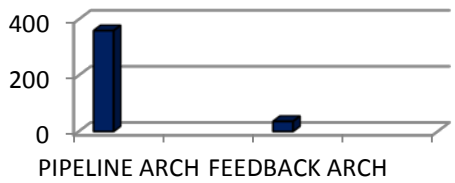

Figure 14: No. Of Resources Used By Individual Arch.

Number Of ROM's used by an individual architecture		
Architecture	**FeedBack**	**PipeLine**
Encryption	**16**	**160**
Decryption	**16**	**160**
Key Gene..	**04**	**20**

Figure 15: Comparison Of Two Arch.

VI. APPLICATION OF AES ALGORITHM.

1. For wireless communication devices like PDA's multimedia cellular phones AES can apply.
2. It can be used for security of Smart cards, wireless sensor networks, wireless mesh Networks.
3. AES have high computational efficiency, so as to be usable in high speed applications, such as broad band links.
4. AES is very well suited for restricted-space environments where either encryption or decryption is implemented. It has very low RAM and ROM requirements.
5. Web servers that need to handle many encryption sessions.
6. Any kind application where security is needed for our current cryptosystems.

CONCLUSION AND FUTURE WORK

To overcome the issue of low efficiency over the traditional CPU-based implementation of AES, we proposed a new algorithm for AES method in this paper. According to our proposal, we designed and implemented the feedback AES algorithm. Our implementation achieves up to 10x speedup over the implementation of AES on a comparable CPU. Our implementation can be applied for the computer forensics which requires high speed of data encryption. In the future, we will focus on efficient implementations of other common symmetric-key encryption algorithms, such as Blowfish, Serpent and Twofish. Besides, future work will also include GPU implementations of hashing and public key algorithms (e.g. MD5, SHA-1 and RSA) in order to create a complete cryptographic framework.

REFERENCE

[1].J. Daemen and V. Rijmen, *AES Proposal:Rijndael*, AES Algorithm Submission, September 3, 1999. http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf.

[2].J. Daemen and V. Rijmen, *The block cipher Rijndael*, Smart Card research and Applications, LNCS 1820, Springer-Verlag, pp. 288-296.

[3]. Maire McLoone, John V. McCanny:Single-Chip FPGA Implementation of the advanced encryption standard algorithm. www.springerlink.com/index/eajtwybnuw9hhejjh.

[4]. FIPS PUB197, Advanced Encryption Standard (AES),National Institute of Standards and Technology, U.S. Department of Commerce, November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197

[5]. Suresh Sharma, T S BSudarshan: Design of an Efficient Architecture for Advanced Encryption Standard Algorithm Using Systonic Structure. http://www.hipc.org/hipc2005/posters/systolic.pdf