ISSN 23499842(Online), Volume 1, Special Issue 2(ICITET 15), March 2015
International Journal of Innovative Trends and Emerging Technologies

ROBUST SCAN FLIP FLOP TECHNIQUE FOR SECURED ADVANCED ENCRYPTION STANDARD

D. UMESH[1], K. RAMESH[2]
[1]PG Student, VLSI Design, [2]Assistant Professor, [1,2]Dept of Electronics and Communication Engineering,
[1,2]SRM University, Kattankulathur, India

Abstract*:* The proposed is a scan-protection scheme that provides testing facilities both at production time and over the course of the circuit's life. The underlying principle is to scan-in both input vectors and expected responses and to compare expected and actual responses within the circuit. This scheme avoids the use of authentication tests. The proposed scan-protection scheme for the most secured cryptographic algorithm (AES Algorithm) to implement on any hardware with BIST architecture. This proposed method uses a Robust Scan Flip-Flops (RSFF) that delivers different outputs state for the same scan input. Thus the technique is unsusceptible to side channel attacks that hackers use to easily scan the encryption/decryption key and algorithm implementation.

I. INTRODUCTION

In today's digital world, encryption is emerging as a disintegrable part of all communication networks and information processing systems, for protecting both stored data and transmitted data. Encryption is the transformation of message input data (known as plaintext) into unintelligible data (known as cipher text) through an algorithm referred to as cipher. There are numerous encryption algorithms are commonly used in computation, but the U.S. government has adopted the Advanced Encryption Standard (AES) to be used by Federal departments and agencies for protecting sensitive information. The National Institute of Standards and Technology (NIST) have published the specifications of this encryption standard in the Federal Information Processing Standards (FIPS) Publication 1997.

Any conventional symmetric cipher, such as AES, requires a single key for both encryption and decryption, which is independent of the plaintext and the cipher itself. It is impractical to retrieve the plaintext solely based on the cipher text and the encryption algorithm, without knowing the encryption key. Thus, the secrecy of the encryption key is of high importance in symmetric ciphers such as AES. Software implementation of encryption algorithms does not provide ultimate secrecy of the key since the operating system, on which the encryption software runs; it is always vulnerable to attacks.

A) ENCRYPTION AND KEY BASED APPROACH

Different versions of AES algorithm existing today (AES128, AES196, and AES256) depending on the size of the encryption key. In this project, a hardware model for implementing the AES128 algorithm was developed using the Verilog hardware description language. A unique feature of the design proposed in this project is that the round keys, which are consumed during different iterations of encryption, are generated in parallel with the encryption process.

B) LANGUAGES

The hardware model was completely verified using a test bench, which took an advantage of the Verilog's programming feature, by constructing random test objects and providing them to the model. Then, the verified model was synthesized using the Synopsis Design-Compiler tool to get an estimated number of gates, area and timing of the hardware model. Finally, the performances of software and hardware implementations were compared.

Cryptographic systems are generally classified on the following basis:

1. *TYPE OF OPERATIONS USED TO FOR TRANSFORMING PLAINTEXT TO CIPHER **TEXT:*** Most encryption algorithms are based on two general principles,

a.Substitution, in which each element in plain text is mapped to some other element to form the cipher text

b.Transposition, in which elements in plaintext are rearranged to form cipher text.

2.*NUMBER OF KEYS USED*: If both the sender and the receiver use a same key then such a system is referred to as Symmetric, single-key, secret-key or conventional encryption. If the sender and receiver use different keys, then such a system is called Asymmetric, Two-key, or private-key encryption.

3.*PROCESSING OF PLAIN TEXT*: A Block cipher process one block input at a time, producing an output for each input block. A Stream cipher processes the input elements continuously producing output elements on the fly.

Most of the cryptographic algorithms are either symmetric or asymmetric key algorithms.

4.*SECRET KEY CRYPTOGRAPHY*: This type of cryptosystem uses the same key for both encryption and decryption. Some of the advantages of such a system are

- ➢ Very fast relative to public key cryptography
- ➢ Considered secure, as long as the key is strong

Symmetric key cryptosystems have some disadvantages too. Exchange and administration of the key becomes complicated. Non-repudiation is not possible. Some of the examples of Symmetric key cryptosystems include DES, 3-DES, RC4, RC5 etc.

5.*PUBLIC KEY CRYPTOGRAPHY*

This type of cryptosystems uses different keys for encryption and decryption. Each user has a public key, which is known to all others, and a private key, which remains a secret. The private key and public key are mathematically linked. Encryption is performed with the public key and the decryption is performed with the private key. Public key cryptosystems are considered to be very secure and supports Non-repudiation. No exchange of keys is required thus reducing key administration to a minimum. But it is much slower than Symmetric key algorithms and the cipher text tend to be much larger than plaintext. Some of the examples of public key cryptosystems include Diffie-Hellman, RSA and Elliptic Curve Cryptography.

Therefore, the implementations of these two transformations affects the implementation of the whole AES tremendously. Later in this chapter, the implementation variations of the S-box and inverse S-box including the composite field implementations are explained in detail.

SECURITY OF AES

Three possible approaches to attacking the AES algorithm are as follows:

Brute Force: This involves trying out all the possible private keys.

Mathematical attacks: There are several approaches, all equivalent in effect to factoring the product of 2 primes.

Timing attacks: These depend on the running time of the decryption algorithm.

EXISTING SYSTEM

And the composite field S-box and Inverse S-box are divided into many blocks and LUT's are used for both S-box and Inverse S-box and optimum solutions were found. Whereas this method is not opted for high speed implementation. And finally parameters are analyzed with help of EDA tools.

PROPOSED SYSTEM

In our proposed approach we introduce the new combinational logic for S-box and Inverse S-box in order to find the most optimum solutions and we also analyzed all required parameters to prove that the proposed system is not only effective in calculations and also give proper efficiency in speed ,power and area through hardware implementation.

INPUTS, OUTPUTS AND THE STATE

The plaintext input and cipher text output for the AES algorithms are blocks of 128 bits. The cipher key input is a sequence of 128, 192 or 256 bits. In other words the length of the cipher key, N_k, is 4, 6 or 8 words which represent the number of columns in the cipher key. The AES algorithm is categorized into three versions based on the cipher key length. The number of rounds of encryption for each AES version depends on the cipher key size.

In the AES algorithm, the number of rounds is represented by N_r, where $N_r = 10$ when $N_k = 4$, $N_r = 12$ when $N_k = 6$, and $N_r = 14$ when $N_k = 8$. The following table illustrated the variations of the AES algorithm. For the AES algorithm the block size (N_b), which represents the number of columns comprising the State is $N_b= 4$.

The basic processing unit for the AES algorithm is a byte. As a result, the plaintext, cipher text and the cipher key are arranged and processed as arrays of bytes.

For an input, an output or a cipher key denoted by a, the bytes in the resulting array are referenced as a_n , where n is in one of the following ranges:

Block length = 128 bits, $0 <= n < 16$
Key length = 128 bits, $0 <= n < 16$
Key length = 192 bits, $0 <= n < 24$
Key length = 256 bits, $0 <= n < 24$

All byte values in the AES algorithm are presented as the concatenation of their individual bit values between braces in the order {b7, b6, b5, b4, b3, b2, b1, b0}. All the AES algorithm operations are performed on a two dimensional 4x4 array of bytes which is called the State, and any individual byte within the State is referred to as $s_{r,c}$, where letter 'r' represent the row and letter 'c' denotes the column.

At the beginning of the encryption process, the State is populated with the plaintext. Then the cipher performs a set of substitutions and permutations on the State. After the cipher operations are conducted on the State, the final value of the state is copied to the cipher text output.

CIPHER TRANSFORMATIONS

The AES cipher either operates on individual bytes of the State or an entire row/column. At the start of the cipher, the input is copied into the State and then, an initial Round Key addition is performed on the State. Round keys are derived from the cipher key using the Key Expansion routine. The key expansion routine generates a series of round keys for each round of transformations that are performed on the State. It consists of the following four steps. Both the LUT based method and the non LUT based method consists of all these four steps.

II. IMPLEMENTATION OF AES ALGORITHM USING LUT METHOD

AES has a fixed block size of 128 bits called a *state*. Block length is limited to 128 bit

- The key size can be independently specified to 128, 192 or 256 bits
- Number of rounds, Nr, depends on key size
- Each round is a repetition of functions that perform a transformation over State array
- Consists of 4 main functions: one permutation and three substitutions

Substitute bytes, *Shift rows*, *Mix columns*, *Add round key*.

BLOCK DIAGRAM OF AES

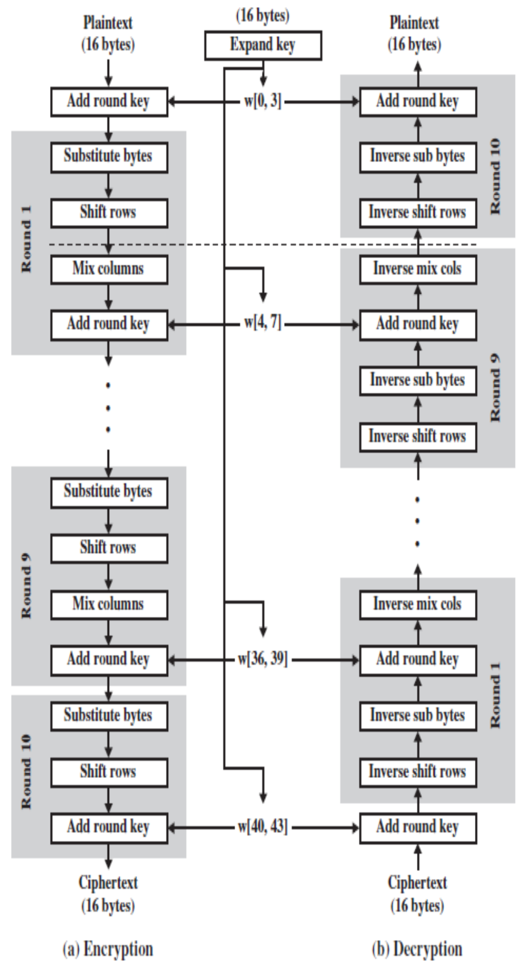


Fig.2.1 - Steps in AES algorithm

ADDROUNDKEY– round key is added to the State using XOR operation.

MIXCOLUMNS – takes all the columns of the State and mixes their data, independently of one another, making use of arithmetic over GF(2^8).

This transformation operates on the columns of the State, treating each columns as a four term polynomial the finite field GF(2^8). Each columns is multiplied modulo x^4+1 with a fixed four-term polynomial a(x) = {03}x^3 + {01}x^2 + {01}x + {02} over the GF(2^8). The MixColumns transformation can be expressed as a matrix multiplication as shown below:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} s_{0,c} \\ s_{0,c} \\ s_{0,c} \\ s_{0,c} \end{bmatrix}$$

The MixColumns transformation replaces the four bytes of the processed column with the following values:

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

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

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

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

The corresponds to the multiplication of polynomials in GF (2^8) . The MixColumns transformation is illustrated in Figure 4.5.This transformation together with ShiftRows provide substantial diffusion in the cipher meaning that the result of the cipher depends on the cipher inputs in a very complex way. In other words, in a cipher with a good diffusion, a single bit change in the plaintext will completely change the ciphertext in an unpredictable manner.

a) SHIFTROWS - Processes the State by cyclically shifting the last three rows of the State by different offsets.

b) SUBBYTES

Uses S-box to perform a byte-by-byte substitution of State

For example, if s1,1 ={53}, then the substitution value would be determinedby the intersection of the row with index ‘5’ and the column with index ‘3’ in the below table This would result in s’1,1 having a value of {ed}.

c) INVERSE CIPHER

The Cipher transformations can be inverted and the implemented in reverse order to produce a straightforward Inverse Cipher for the AES algorithm. The individual transformations used in the Inverse Cipher -InvShiftRows, InvSubBytes, InvMixColumns, and AddRoundKey process the State and are described in the following subsections.

d) *INVERSE SHIFTROWS TRANSFORMATION:*
Inverse Shift Rows is the inverse of the ShiftRows transformation.

The bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, *r* = 0, is not shifted. The bottom three rows are cyclically shifted by ***Nb**-shift*(*r*, *Nb*) bytes, where the shift value *shift(r,Nb)* depends on the row number.There is no shift for the first column, second column is left shifted once, second row is left shifted for two times and the third row left is shifted for three times

e) *INVERSE SUB BYTES TRANSFORMATION*:
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).

f*) INVERSE MIX COLUMNS TRANSFORMATION:*
Inverse Mix Columns is the inverse of the Mix Columns transformation. 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 with a fixed polynomial $a^{-1}(x)$.TheMultiplication is done as shown below.

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

g) INVERSE ADD ROUND KEY TRANSFORMATION
AddRoundKey is its own inverse, since it only involves an application of the XOR operation.

III. ROBUST SCAN TECHNIQUE

For a long time, the Data Encryption Standard (DES) was considered as a standard for the symmetric key encryption. DES has a key length of 56 bits. However, this key length is currently considered small and can easily be broken. For this reason, the National Institute of Standards and Technology (NIST) opened a formal call for algorithms in September 1997. A group of fifteen AES candidate algorithms were announced in August 1998. Next, all algorithms were subject to assessment process performed by various groups of cryptographic researchers all over the world. In August 2000, NIST selected five algorithms: Mars, RC6, Rijndael, Serpent and Twofish as the final competitors.

These algorithms were subject to further analysis prior to the selection of the best algorithm for the AES. Finally, on October 2, 2000, NIST announced that the Rijndael algorithm was the winner. Rijndael can be specified with key and block sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits. Therefore, the problem of breaking the key becomes more difficult [1]. In cryptography, the AES is also known as Rijndael [2]. AES has a fixed block size of 128 bits and a key size of 128, 192 or 256 bits.

i) SIDE CHANNEL ATTACKS

Scan test has been widely adopted as a default testing technique among most VLSI designs, including crypto cores. Unfortunately, these scan chains might be used as a “side channel” to recover the secret keys from the hardware implementations of cryptographic algorithms, for example scan-based attacks on Data Encryption Standard (DES), Advanced Encryption Standard (AES), and Elliptic Curve Cryptography (ECC) [1]–[3], respectively.

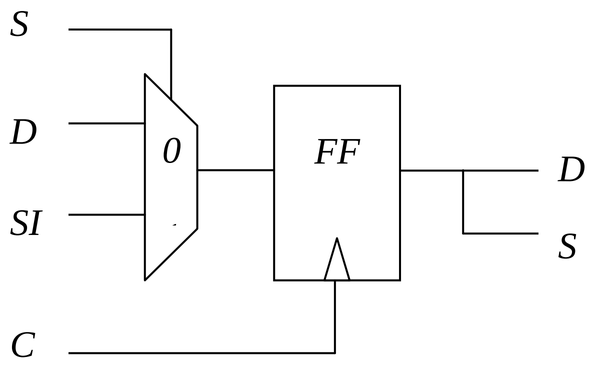


Fig 3.1. Normal Scan FF.

In general, the existing scan-based side channel attacks (SSCA) could be viewed as one kind of differential cryptanalysis by using scan chains of crypto cores. Unlike other known side channel attacks, SSCA is much easier. It is because that in SSCA, in addition to the primary outputs of the crypto cores, a hacker could use scan chain to shift out the intermediate contents during a cryptographic operation. It was illustrated in [2] that on average overall only 544 plaintexts are required to discover the AES key by using SSCA, which clearly shows the great potential threat of scan-based side channel attack

ii) PREVIOUS IMPLEMENTATIONS OF THE S-BOX

One of the most common and straight forward implementation of the S-Box for the

SubByte operation which was done in previous work was to have the pre-computed values stored in a ROM based lookup table. In this implementation, all 256 values are stored in a ROM and the input byte would be wired to the ROM's address bus. However, this method suffers from an unbreakable delay since ROMs have a fixed access time for its read and write operation. [3] Furthermore, such implementation is expensive in terms of hardware. A more refined way of implementing the S-Box is to use combinational logic.

Such examples of work that implements the S-Box using this method were [1], [3] and [5]. This S-Box has the advantage of having small area occupancy, in addition to be capable of being pipelined for increased performance in clock frequency. The S-Box architecture discussed in this paper is based on the combinational logic implementation.

iii) THE SUBBYTES AND INVSUBBYTE TRANSFORMATION

The Sub Bytes transformation is computed by taking the multiplicative inverse in GF (28) followed by an affine transformation. For its reverse, the InvSubBytetransformation,the inverse affine transformation is applied first prior to computing the multiplicative inverse.The steps involved for both transformation is shown below.

SubByte: Multiplicative Inversion in GF (28), Affine Transformation

InvSubByte: Inv Affine Transformation, Multiplicative Inversion in GF (28).

The AT and AT^{-1} are the Affine Transformation and its inverse while the vector *a* is the multiplicative inverse of the input byte from the state array. From here, it is observed that both the SubByte and the InvSubByte transformation involve a multiplicative inversion operation. Thus, both transformations may actually share the same multiplicative inversion module in a combined architecture. An example of such hardware architecture is shown below. Switching between SubByte and InvSubByte is just a matter of changing the value of INV. INV is set to 0 for SubByte while 1 is set when Inverse Sub Byte operation is desired.

iv) S-BOX CONSTRUCTION METHODOLOGY

This section illustrates the steps involved in constructing the multiplicative inverse module for the S-Box using composite field arithmetic. Since both the SubByte and InvSubByte transformation are similar other than their operations which involve the Affine Transformation and its inverse, therefore only the implementation of the SubByte operation will be discussed in this paper. The multiplicative inverse computation will first be covered and the affine transformation will then follow to complete the methodology involved for constructing the S-Box for the SubByte operation. For the InvSubByte operation, the reader can reuse multiplicative inversion module and combine it with the Inverse Affine Transformation.

Sub byte transformation:

First multiplicative inversion of the eight bit value is taken then affine transformation is done by following matrix the affine transformation matrix.

$$AT(a)=\begin{pmatrix}1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\\1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\end{pmatrix}\times\begin{pmatrix}a_7\\a_6\\a_5\\a_4\\a_3\\a_2\\a_1\\a_0\end{pmatrix}\oplus\begin{pmatrix}0\\1\\1\\0\\0\\0\\1\\1\end{pmatrix}$$

Inverse sub byte:

After taking inverse affine transform the eight bit subbytevalue is transformed into eight bit value by undergoing multiplicative inversion .

v) ROBUST SECURE SCAN

Due to the security and testability requirements as mentioned above, a novel robust secure scan-based test approach is proposed as a countermeasure against scan-based differential cryptanalysis.

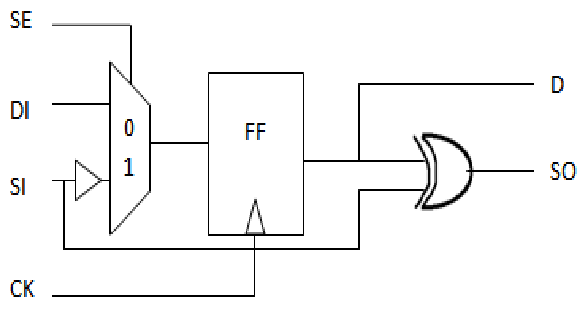


Fig –3.2 Proposed RSFF.

When in normal function mode (SE==0) SFF loads data from the logic through DI, and the output to logic is DO. Because the additional inverter and the XOR gate are inserted along the scan path, they do not affect the timing of the design. Thus in function mode, RSSF works like a traditional scan flip flop. When in scan test mode, we can observe from Fig. 1 that (3) during scan shift operation, the content of FF is XOR ed with SI to be shifted out to the next SFF and the inverted scan-in data (SI) will be loaded into FF. Thus for hackers, it becomes extremely complicated to identify the relationship between the captured response and the scan-out.

RSS design:

The basic idea of the proposed RSS design is to encrypt the contents in scan chains during scan operation, so as to reduce the controllability and observability of unintended users. By doing this, it becomes more complicated for hackers to identify the bit differences between pairs of related plaintexts when they are encrypted under the same key. One kind of the proposed RSS design is shown in Fig. 1, in which the contents of two neighboring SFFs are encoded during scan operation from a security aspect.

When compared with the traditional SFF, an extra inverter and an XOR gate are introduced in the RSS design. This simple logic could be used for encryption during scan operations. Observe that the proposed robust scan flip-flop (RSSF) has identical pin outs when compared with the traditional scan flip-flop as shown in Fig. 1, and is therefore fully compatible with industry standard design tools from a design perspective, when integrated into current design flows it only requires the RSSF added into the cell library.

vi) SECURITY AND IMPLEMENTATION ANALYSIS

In this section, security analysis and implementation overhead are discussed to show the advantages of the proposed secure test technique over existing methods.

Security analysis:

Due to the avalanche effect of cryptographic algorithms, there exist two kinds of scan-based differential cryptanalysis, called as constant based (CBA) and fixed hamming-distance-based attack (FHDA). Here let us use AES as an example cryptographic algorithm to explain these two kinds of attacks. CBA takes advantages of the fact that in encryption process, the contents of some special registers are independent on the inputted plaintext. For example, the round registers in AES, without special protection, for each normal inputs, in the first cycle they would be 0001, and then 0010,… 1010.

By using several different plaintext inputs and scanning out the contents at different times of the cryptographic operation, these registers could be easily identified. Then by setting the registers as 1010 (i.e., to indicate the round cycle is 10, the last round for 128-bit AES), which is because in AES the mix-column operation is bypassed in the last round, it became much easier to discover the secret keys. Such a kind of attack is called constant-based attack. FHDA is another kind of scan-based attack by counting the number of bit changes on relevant plaintexts so as to discover the secret key, and refer to [2] for more details on FHDA.

Reliable against attack:

When using the proposed RSS, it can be easily configured that once the intermediate data of CFFs passing the replaced RSSFs, they would be encrypted and this makes it extremely difficult to identify the positions of CFFs in the scan chain from external. In addition, because the proposed RSSFs deals with the scan-in and scan-out as well, it is also difficult for hackers to set the CFFs to desired states with no detailed knowledge of the scan structure implementation.

We simply group the registers together in the scan chain for each block, replace the last SFF in the scan chain with RSSF, and then conduct FHDA. Here we found that the two pairs of plaintexts do not belong to any of the original four pairs, which might mislead the hackers to wrong keys.

vii) DESIGN HIERARCHY

The proposed AES128 hardware model is a 3-level hierarchical design as shown in Figure 8. The root module in the hierarchy is the AES128_cipher_top. This module implements the AES128 pseudo code displayed in Figure 2. It has two 128-bit inputs for receiving the cipher key and the plaintext. There is also a single bit input signal, *'Ld'*, which is used to indicate the availability of a new set of plaintext or cipher key on the input ports. The completion of the encryption process is indicated by asserting the 'done' single bit output.

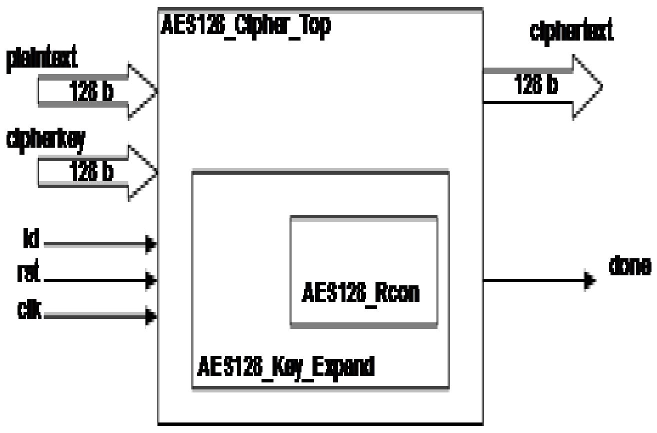
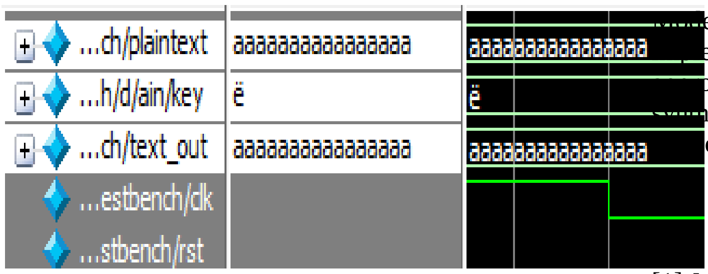


Fig -3.3 Design Hierarchy

A unique feature of the proposed design is that the AES128_Key_Expand module is pipelined with the AES128_cipher_top module. While the AES128 cipher top module is performing an iteration of the encryption. Transformations on the *State* using the previously generate round keys, the AES128 Key Expand produces the next round's set of keys to be used by the root module in the next encryption iteration.

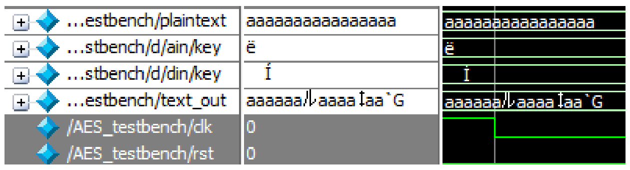
IV. SIMULATION OUTPUT WITH MASKING

A) During Scan Mode, When both the key in encryption and decryption are same:



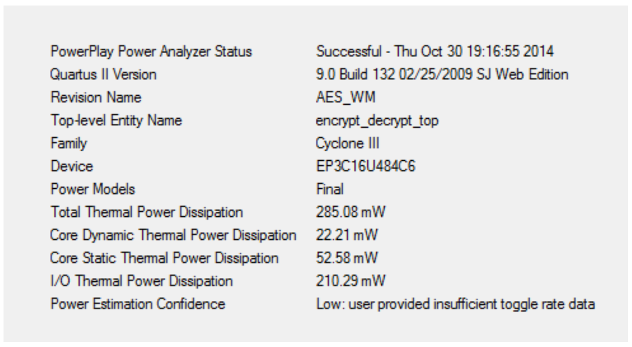
```
...ch/plaintext   aaaaaaaaaaaaaaaa   aaaaaaaaaaaaaaaa
...h/d/ain/key    ë                  ë
...ch/text_out    aaaaaaaaaaaaaaaa   aaaaaaaaaaaaaaaa
...estbench/clk
...stbench/rst
```

B) During Scan Mode, When both the key in encryption and decryption are different:



```
...estbench/plaintext   aaaaaaaaaaaaaaaa       aaaaaaaaaaaaaaaa
...stbench/d/ain/key    ë                      ë
...stbench/d/din/key    Í                      Í
...estbench/text_out    aaaaaa/\aaaa Íaa`G     aaaaaa/\aaaa Íaa`G
/AES_testbench/clk      0
/AES_testbench/rst      0
```

C) Total Power Dissipation with Masking Key:



PowerPlay Power Analyzer Status	Successful - Thu Oct 30 19:16:55 2014
Quartus II Version	9.0 Build 132 02/25/2009 SJ Web Edition
Revision Name	AES_WM
Top-level Entity Name	encrypt_decrypt_top
Family	Cyclone III
Device	EP3C16U484C6
Power Models	Final
Total Thermal Power Dissipation	285.08 mW
Core Dynamic Thermal Power Dissipation	22.21 mW
Core Static Thermal Power Dissipation	52.58 mW
I/O Thermal Power Dissipation	210.29 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

V. CONCLUSION

In this brief, we carried out implementation of AES cryptographic algorithms with scan based testing futures. It has been previously demonstrated that scan chains introduced for hardware testability open a back door to potential attacks. Here, we propose a level based masking and RSFF based flip flop masking as a scan-protection scheme that provides testing facilities both at production time and over the course of the circuit's life. Compared to regular scan tests, this technique has no impact on the quality of the test or the model-based fault diagnosis. Here we proved that RSFF based AES will give better hardware complexity & power optimization with considerable delay enhancement.

An accurate SFF-based analysis approach was introduced for AES core with single and multi FF characterizations. The proposed approach was derived from the SFF method. The method avoids the use of a large number of masking parameters to minimize the required resources for area- and power-efficient built-in testing applications. elsim based pre simulation results of an AES ementation showed the feasibility of the oach. For a QUARTUS II based hardware esis report proved the efficiency of proposed od

REFERENCES

[1] M. Akkar and C. Giraud, "An Implementation of DES and AES, Secure against Some Attacks," In Proc. of the Workshop on Cryptographic Hardware and Embedded Systems (CHES2001), Paris, France, pp. 315-325, May 2001.
[2]http://www.altera.com/products/software/products/quartus2/qts-index .html
[3] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," AES algorithm submission, June 1998.
[4] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard," IEEE Trans. on Computers, vol. 52, no. 4, pp. 492-505, April 2003.
[5] G. Bertoni, L. Breveglieri, I. Koren, and P. Maistri, "An efficient hardwarebased fault diagnosis scheme for AES: performances and cost," In Proc. of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT2004), Cannes, France, pp. 130-138, Oct. 2004.
[6] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations," Journal of Cryptology, vol. 14, no. 2, pp. 101-119, 2001.