A Self-Testing Fully Pipelined Implementation for the Advanced Encryption Standard

Mahdi Nazm-Bojnordi, Naser Sedaghati-Mokhtari, *and* Seid Mehdi Fakhraie
{m.bojnordi, n.sedaghati} @ece.ut.ac.ir, fakhraie@ut.ac.ir
Silicon Intelligence and VLSI Signal Processing Laboratory
ECE Department, University of Tehran, Tehran, IRAN.

Abstract— **In contrast to software implementation, hardware implementation of encryption protocols provides a higher level of security and cryptography speed at some flexibility cost. In this paper, different existing implementations of Advanced Encryption Standard (AES) are considered and a fully pipelined implementation for the AES is presented. Implementation considers both encryption and decryption. The design is optimized for achieving higher speed and lower area cost. The Selected algorithm for our design is Rijndael. The major part of an AES design is designing substitute boxes (S-box). S-boxes in our design are implemented at a lower cost rather than the existing implementations. Throughput of up to 6 Gbps is gained by our proposed architecture. This implementation is equipped with BIST architecture for self testing.**

Index Terms—**AES, self-testing, BIST, Rijndael, fully pipeline implementation.**

I. Introduction

THE use of encryption/decryption is as old as the art of communication. The Advanced Encryption Standard (AES) is an encryption algorithm for securing sensitive but unclassified material by U.S. Government agencies. In March 1999, the National Institute of Standards and Technology (NIST) organized the Second Advanced Encryption Standard Candidate Conference (SAESCC) in Rome where a series of analyses of various algorithms were presented. This analysis includes evaluating not only their security capabilities, but also their performance, flexibility of implementation and other issues [1] Finally in October 2000, NIST announced the Rijndael as the winner algorithm for AES. In many implementations, Rijndael shows that it is an efficient algorithm for the AES; for example, in IPsec applications that handle more than thousands of security associations, it works very well [2].

In November 2001, the AES was accepted as a standard of the Federal Information Processing Standards (FIPS) [3]. Since then, because of high volume uses of encryption applications, many software and hardware implementations have been published for the AES, where each of them consider features of optimization differently. For example, it is used in many places in Internet applications, such as routers.

Generally, when there is no concern on performance objectives and higher encryption or decryption speed, the software implementation may be the appropriate choice. In addition to lower speed of the software implementations, since the AES is a symmetric key encryption, the cipher key is easily vulnerable. In hardware implementation, however it is very hard to detect the cipher key by the attacker.

As a consequence, there is a growing interest in efficient implementations of the AES. For many applications, these implementations need to be resistant against side channel attacks, i.e. it should not be too easy to extract secret information from physical measurements on the device. Also there are many applications using AES as an efficient encryption method to secure their data exchange, such as the applications working on the networks. Also, the AES is an efficient and reliable method to be used in real-time applications such as multimedia broadcasting. Using the AES for real-time applications needs to consider low delay and fast architectures. In the literature, there are some AES hardware implementations for both ASIC and FPGAs. Also speed up to 609 Mbps is available for ASIC technology implementation [4]. In this paper a fast fully pipelined architecture for the AES encryption method is implemented that is suitable for securing data exchange in real-time applications such as video encryption.

The rest of this paper is organized as follows: The AES algorithm is explained in Section II, its hardware implementation and the BIST architecture are discussed in Section III and the paper conclusion appeares in Section IV.

II. Algorithm

This section introduces a briefing on the selected algorithm by the NIST for the AES. The Rijndael algorithm is a symmetric block cipher (series of transformations that converts plaintext to ciphertext) methodology that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. However, in excess of AES design criteria, the block sizes can mirror those of the keys [3]. Rijndael uses a variable number of rounds, depending on key/block sizes, as follows: a) 9 rounds if the key/block size is 128 bits. b) 11 rounds if the key/block size is 192 bits. c) 13 rounds if the key/block size is 256 bits. The AES block can be divided into four functional blocks; each of them operates on 128-bit input data (called state) and prepares the 128-bit output state for the next block. The State can be thought of as an array, structured with 4 rows and the column number being the block length divided by bit length (for example, divided by 32 and equal to 4 that perform an 4x4 array) . AES functional blocks are as they are described in the following.

A. *SubBytes Transformation*

This is a non-linear transformation that operates independently on each byte of the state using a substitution table (called S-box) [5]. The S-box is a one-to-one mapping table and consequently it is invertible. In the SubBytes step, each byte in the array is updated using an 8-bit S-box. This operation provides the non-linearity in the cipher. This transformation is constructed based on the two following phases:

1. Take the multiplicative inverse in the finite field GF (2^8), (Galois fields) [6],
2. Apply an affine (over GF (2)) transformation defined by

$$b'_i = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i .$$

To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points (and so is a rearrangement), and also any opposite fixed points.

B. *ShiftRows Transformation*

The ShiftRows block operates on the rows of the state; it cyclically shifts the bytes in each row by a certain offset. For AES, the first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. In this way, each column of the output state of the ShiftRows block is composed of bytes from each column of the input state. (as shown at Fig. 1.) (Rijndael variants with a larger block size have slightly different offsets). In fact this block is used for shuffling the state.

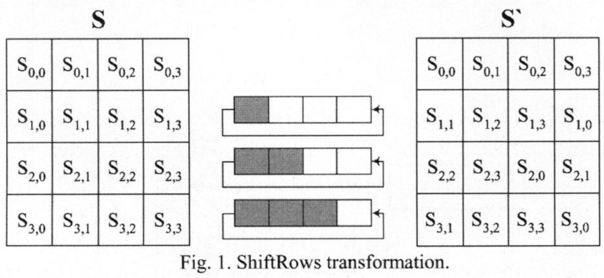
S
S`
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,1}$ $S_{1,2}$ $S_{1,3}$ $S_{1,0}$
$S_{2,2}$ $S_{2,3}$ $S_{2,0}$ $S_{2,1}$
$S_{3,1}$ $S_{3,2}$ $S_{3,3}$ $S_{3,0}$

Fig. 1. ShiftRows transformation.

C. *MixColumns Transformation*

In the MixColumns block, the four bytes of each column of the state are combined using an invertible linear transformation. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with ShiftRows, MixColumns provides diffusion in the cipher. Each column is treated as a polynomial over GF (2^8) and is then multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x) = 3x^3 + x^2 + x + 2$. The MixColumns block can also be viewed as a matrix multiply in Rijndael's finite field. This transformation operates on the state column-by-column.

D. *AddRoundKey Transformation*

In this transformation a Round Key is added to the state by a simple bitwise XOR operation. Using this transformation encryption process depends on the secret key. As will be discussed later, in each round of encryption process the state is XORed bit-by-bit with a key pattern that prepared for the same round. Also, this transformation is used in the decryption process.

Each sequence of these four operations constitutes one round of the encryption process and in the reverse order is used for the decryption process. Number of rounds in an AES implementation depends on key length (see TABLE I).

Initializing the process is done by XORing the cipher key and input data. After initializing, the state goes through the rounds transformations. In the last round, however, the MixColumn block does not exist. On the other hand, appropriate keys for each round of the process are fed to AddRoundKey block by KeyGenerator block. This process which is called KeyScheduling, consists of some XOR operations and byte substitution on the cipher key.

All of these operations are used to decrypt the cipher text and extract the plaintext in the inverse order.

TABLE I
NUMBER OF ROUNDS ACCORDING TO KEY LENGTH

	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

TABLE II
COMPARISONS OF 8-BIT S-BOXES

	SRAM 256x8	ROM 2x256x8	GF(2^4)	Our design
Gate count	2138	1866	762	754
Delay (ns)	-	-	5	4.88

III. IMPLEMENTATION AND DISCUSSION

Implementation of our proposed AES architecture is described in three major parts as follows.

A. *S-box Implementation*

A huge amount of area in AES implementation is occupied by the S-box lookup tables. Therefore, most of the AES implementations are concentrated on optimizing the S-boxes used in the design. In some of AES implementations S-box is implemented using LUT which is efficient for FPGA not ASIC. An efficient method of S-box implementation represents the polynomial elements in GF (2^8) as the elements in GF (2^4), and then uses a mapping followed by an affine transformation [5].

An 8-bit S-box which is implemented in 0.35μm ASIC technology using this method consists of 762 gates and has a delay of 5ns [7]. We have also implemented an 8-bit S-box to compare our results to the implemented architecture in [7]. The results obtained for our design appear in TABLE II.

As mentioned before, S-box in AES implementation is the most frequently used component. In our design, 200 instances of S-box are used. 81% of the design is occupied by the S-boxes.

B. AES Core Implementation

For AES, there are several methods of implementation with respect to area or delay optimizations, such as high throughput loop iterative implementation presented in [7] or loop unrolling and pipelining implementation [8].

The concern of this paper for optimization is achieving higher speed rather than minimizing area. In Fig. 2, a block diagram of our design is illustrated.

As depicted in Fig. 2, the design has an unrolled pipelined structure. To prepare the keys for pipeline stages, there is a key generator block which gets the cipher key as its input and has eleven output ports for feeding the appropriate keys to the stages of pipeline at proper times. This component has a pipelined structure that lets the design work properly without any stalls. All the blocks in Fig. 2 have a 1-bit controlling input that specifies the operational mode of each of them. Each block is able to operate for encryption or decryption process.

Using this structure, the AES chip can switch between encryption and decryption modes freely. Moreover, input cipher key may change without any pipeline stalls if needed.

Each pipeline stage has a general structure as illustrated in Fig. 3. The paths of encryption and decryption are separated by two multiplexers.

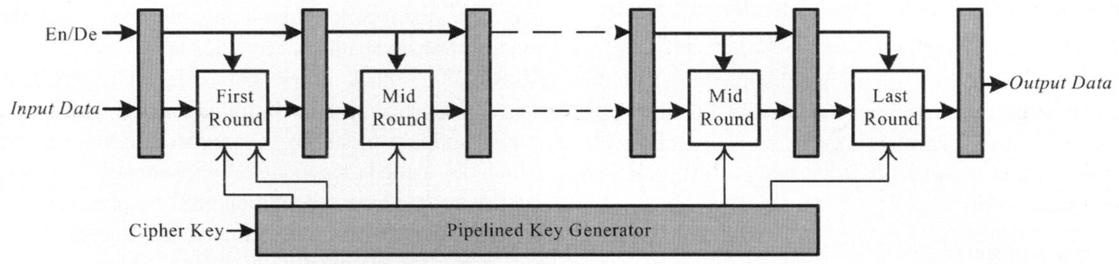

Fig. 2 Unrolled & pipelined AES architecture.

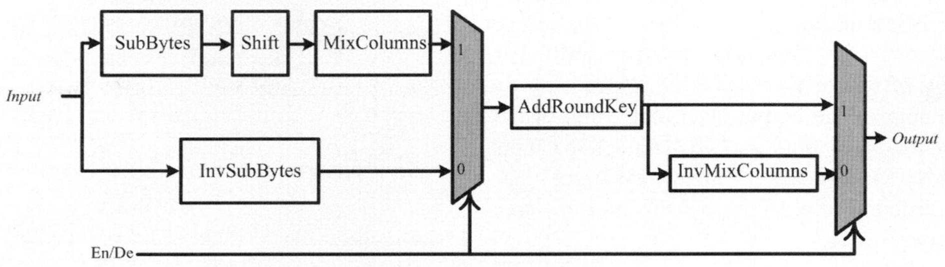

Fig. 3 Each middle pipeline stage.

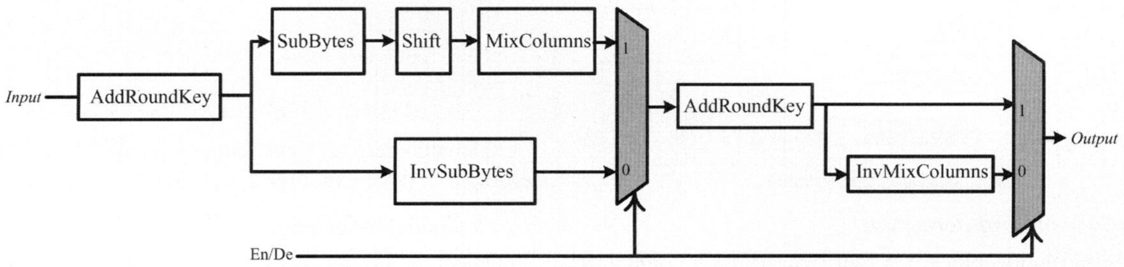

Fig. 4 The first pipeline stage.

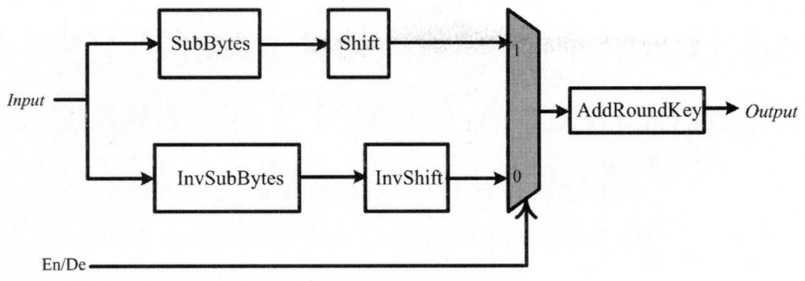

Fig. 5 Last pipeline stage.

For the middle stages of the pipeline, this structure is used as shown in Fig. 3. For the first stage component, however, an AddRoundKey block is added to the head of this structure (Fig. 4.). The last stage, MixColumns and InvMixColumns blocks are removed from this structure (Fig. 5).

A. BIST Implementation

To make our design testable, Built-In Self-Test (BIST) architecture is added to the design. Since all the primary inputs and outputs are buffered, there is no need to parallel pattern generator/collector. All the flip-flops in the design are extracted and modified to be connected as a scan chain. Using a 32-bit LFSR (Linear Feedback Shift Register, that generates pseudo-random numbers) test vectors are generated and using a 32-bit SISR (Single Input Signature Register), signatures are collected. Fig. 6 shows the implemented BIST for our design.

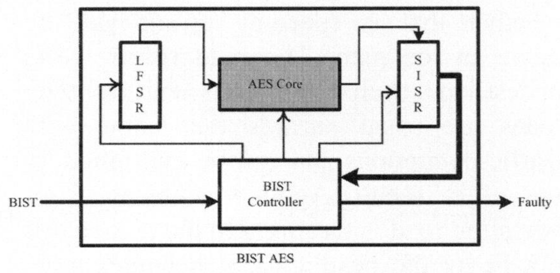

Fig. 6 BIST architecture for AES.

Without the ROM unit which is containing the signatures for comparison, overhead of the implemented BIST for our design is just 636 gates. This number of gates is much less than core gate count that can be ignored (636 vs. 180176). The major part of the BIST architecture is a ROM component containing signatures which are obtained by applying the generated test vectors to the golden model (of the design) at the simulation time.

IV. Conclusions

This paper made a brief analysis on AES implementations and presented a testable unrolled pipelined implementation for AES. The design is synthesized using a 0.35μm ASIC library, for which a delay less than 20ns is extracted for each pipeline stage of the design. Therefore it can achieve a maximum throughput of 6 Gbps. With the added BIST architecture we obtained test coverage about 98%. A top level diagram for our design is shown in Fig. 7.

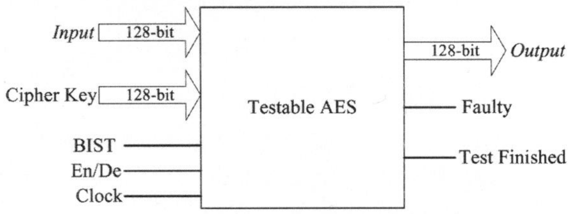

Fig. 7 Implemented AES block.

References

[1] D. Forte, "The future of the advanced encryption standard," *In Journal of Network Security, Elsevier Science Ltd*, pp. 10-13, 1999.

[2] D. Piper, "The Internet IP Security Domain of Interpretation for ISAKMP," *RFC 2407*, Nov 1998.

[3] "Advanced Encryption Standard (AES)," Federal Information Processing Standards Publication 197, Nov. 26, 2001.

[4] C. Lu, S, Tseng, "Integrated design of AES (Advanced Encryption Standard) encrypter and decrypter," i*n Proc. International Conference on Application-Specific Systems*, pp. 277-285, Jul. 2002.

[5] Rijmen Vincent, "Efficient Implementation of the Rijndael S-box," Available on: http://www.iaik.tu-graz.ac.at/research/krypto/AES/old/~ri jmen/rijndael/sbox.pdf

[6] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[7] T. F. Lin, C. P. Su, C. T. Huang and C. W. Wu, "A High-throughput low-cost AES cipher chip," i*n Proc. IEEE Asia-Pacific Conference on ASIC*, pp. 85 – 88, 2002.

[8] A. Hodjat, W. Ingrid, "A 21.54 Gbits/s fully pipelined processor on FPGA," *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 308 - 309, April 2004.