

Very High Speed AES Implementation For Next Generation Internet SecurityD.Sai Teja¹, A.Ashwin², S.Pavan Sai Krishna³^{1,2,3}National Institute Of Technology Puducherry, Puducherry, India, dsaiteja1@gmail.com

Abstract: The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. Encryption converts data to an unintelligible form called cipher text; decrypting the cipher text converts the data back into its original form, called plaintext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits. Due to the growth of applications in Internet and wireless communication, more and more users require the security measures and devices for protecting the data, which users transmit over the channels. Since nobody can guarantee that the information will not be stolen over open communication channels, it is a general way to encrypt the information before they are transmitted into the channels. The AES algorithm has broad applications, including smart cards and cellular phones, WWW servers and automated teller machines (ATMs), and digital video recorders. Compared to software implementations, hardware implementations of the AES algorithm provide more physical security as well as higher speed.

Keywords: AES, encipher, cryptographic keys, decipher

1. Introduction

There are many crypto-system developed in the past. According to the key type, the cryptography can be classified into two types, such as the asymmetric-key and symmetric-key cryptography. Asymmetric-key cryptography manipulates two different keys for encryption and decryption and provides a robust mechanism for the key transportation. On the other hand, symmetric-key cryptography uses an identical key for both encryption and decryption, which is more efficient for large amount of data. Moreover, resource-sharing scheme will also be employed to reduce the hardware complexity of the cipher and decipher.

This standard specifies the Rijndael algorithm ([3] and [4]), a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. Rijndael was designed to handle additional block sizes and key lengths; however they are not adopted in this standard.

Throughout the remainder of this standard, the algorithm specified herein will be referred to as “the AES algorithm.” The algorithm may be used with the three different key lengths indicated above, and therefore these different “flavors” may be referred to as “AES-128”, “AES-192”, and “AES-256”.

This specification includes the following sections:

1. Algorithm specification, covering the key expansion, encryption, and decryption routines;

2. Implementation issues, such as key length support, keying restrictions, and additional block/key/round sizes.

The standard concludes with several appendices that include step-by-step examples for Key Expansion and the Cipher, example vectors for the Cipher and Inverse Cipher, and a list of references.

Firstly, a look at the history and background of encryption should produce a clearer understanding for the need of security and how implementations such as the one being produced can help with this need.

The next part of this review will look at the more specific encryption technique known as the Advanced Encryption Standard (AES) and explain why it is used, how it works and how it can be implemented. Following this, the investigation must be expanded to search for solutions for the problem at hand. This involves looking for possible optimization techniques and how they can be incorporated into the solution as well as highlighting the differences needed between the two implementations.

In addition, this review will be considering other factors which will contribute to the overall performance of the final implementations. These factors are likely to have

a large effect of the overall efficiency of algorithms. Finally, due to the rapid advances in the technology of encryption and cryptanalysis it is necessary to examine the likelihood of the dissertation findings being obsolete shortly after they are produced. This has been known to happen in the past where weaknesses are found in other encryption systems. If this was the case then it may be necessary to use another encryption standard.

2. AES Algorithm

For the AES algorithm, the length of the input block, the output block and the State is 128 bits. This is represented by $N_b = 4$, which reflects the number of 32-bit words (number of columns) in the State.

For the AES algorithm, the length of the Cipher Key, K , is 128, 192, or 256 bits. The key length is represented by $N_k = 4, 6, \text{ or } 8$, which reflects the number of 32-bit words (number of columns) in the Cipher Key.

For the AES algorithm, the number of rounds to be performed during the execution of the algorithm is dependent on the key size. 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 only Key-Block-Round combinations that conform to this standard are given in Fig. 1. For implementation issues relating to the key length, block size and number of rounds.

Standards	Key Length	Block Size	Number of Rounds
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Figure1. Key-Block-Round Combinations

For both its Cipher and Inverse Cipher, the AES algorithm uses a round function that is composed of four different byte-oriented transformations: 1) byte substitution using a substitution table (S-box), 2) shifting rows of the State array by different offsets, 3) mixing the data within each column of the State array, and 4) adding a Round Key to the State. These transformations (and their inverses) are described in this section

2.1 Cipher

At the start of the Cipher, the input is copied to the State array using the convention. After an initial Round Key addition, the State array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length), with the final round differing slightly from the first $N_r - 1$ rounds. The final State is then copied to the output. The overall structure of AES for the case of 128-bit encryption is shown below:

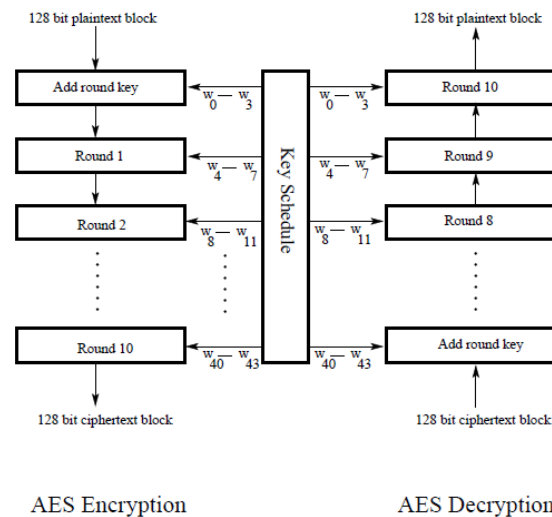


Figure 2: Overall Structure of AES for the case of 128-bit encryption key

The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words derived using the Key Expansion routine.

The Cipher is described in the pseudo code in Fig. 3. The individual transformations - SubBytes(), ShiftRows(), MixColumns(), and AddRoundKey() – process the State and are described in the following subsections. In Fig.2, the array $w[]$ contains the key schedule

As shown in Fig. 3, all N_r rounds are identical with the exception of the final round, which does not include the MixColumns() transformation.

Cipher showing values for the State array at the beginning of each round and after the application of each of the four transformations described in the following sections.

```

Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]
  state = in
  AddRoundKey(state, w[0, Nb-1])           // See Sec. 3.1.4
  for round = 1 step 1 to Nr-1
    SubBytes(state)                         // See Sec. 3.1.1
    ShiftRows(state)                       // See Sec. 3.1.2
    MixColumns(state)                      // See Sec. 3.1.3
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
  end for
  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

  out = state end

```

Fig.3. Pseudo Code for the Cipher

2.2 Key Expansion

The AES algorithm takes the Cipher Key, K, and performs a Key Expansion routine to generate a key schedule. 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 requires Nb words of key data. The resulting key schedule consists of a linear array of 4-byte words, denoted [w_i], with i in the range 0 ≤ i < Nb (Nr + 1). The expansion of the input key into the key schedule proceeds according to the pseudo code in Fig10.

SubWord() is a function that takes a four-byte input word and applies the S-box (Sec. 3.1.1, Fig. 5) to each of the four bytes to produce an output word. The function RotWord() takes a word [a₀,a₁,a₂,a₃] as input, performs a cyclic permutation, and returns the word [a₁,a₂,a₃,a₀]. The round constant word array, Rcon[i], contains the values given by [xⁱ⁻¹,{00},{00},{00}], with xⁱ⁻¹ being powers of x (x is denoted as {02}) in the field GF(28), as discussed in Sec. 5.2 (note that

i starts at 1, not 0) performs a cyclic permutation, and returns the word [a₁,a₂,a₃,a₀]. The round constant word array, Rcon[i], contains the values given by [xⁱ⁻¹,{00},{00},{00}], with xⁱ⁻¹ being powers of x (x is denoted as {02}) in the field GF(28), as discussed in Sec. 5.2 (note that i starts at 1, not 0).

From Fig10 it can be seen that the first N_k words of the expanded key are filled with the Cipher Key. Every following word, w_i, is equal to the XOR of the previous word, w_{i-1}, and the word N_k positions earlier, w_{i-N_k}. For words in positions that are a multiple of N_k, a transformation is applied to w_{i-1} prior to the XOR, followed by an XOR with a round constant, Rcon[i]. This transformation consists of a cyclic shift of the bytes in a word (RotWord()), followed by the application of a table lookup to all four bytes of the word (SubWord()).

2.3 Inverse Cipher

The Cipher transformations in Sec. 3.1 can be inverted and then 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.

The Inverse Cipher is described in the pseudo code in Fig.4. In Fig.4, the array w[] contains the key schedule, which was described previously

```

InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]

  state = in
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  for round = Nr-1 step -1 downto 1
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    InvMixColumns(state)
  end for

  InvShiftRows(state)
  InvSubBytes(state)
  AddRoundKey(state, w[0, Nb-1])

  out = state
end

```

Fig.4. Pseudo Code for the Inverse Cipher

3. Discussion of Results

The developed project is simulated and verified their functionality. Once the functional verification is done, the RTL model is taken to the synthesis process using the Xilinx ISE tool. In synthesis process, the RTL model will be converted to the gate level netlist mapped to a specific technology library. Here in this Spartan 6 family, many different devices were available in the Xilinx ISE tool. In order to synthesis this design the device named as "XC6SLX4" has been chosen and the package as "TQG144" with the device speed such as "-3".

4. Conclusion

At present the data is encrypted at 1GBPS speed, each input takes 63 clock cycles to generate cipher text. With Advanced Encryption, the data is encrypted at 100 Gbps speed, i.e each input takes one clock cycle to generate cipher text. A combinational logic based S-Box for the SubByte transformation is discussed and its internal operations are explained. As compared to the typical ROM based lookup table, the presented implementation is both capable of higher speeds since it can be pipelined and small in terms of area occupancy (43 slices for a 2 stage pipeline on a

Spartan II XCS200-5FPGA). This compact and high speed architecture allows the S-Box to be used in both area-limited and demanding throughput AES

chips for various applications, ranging from small smart cards to high speed servers.

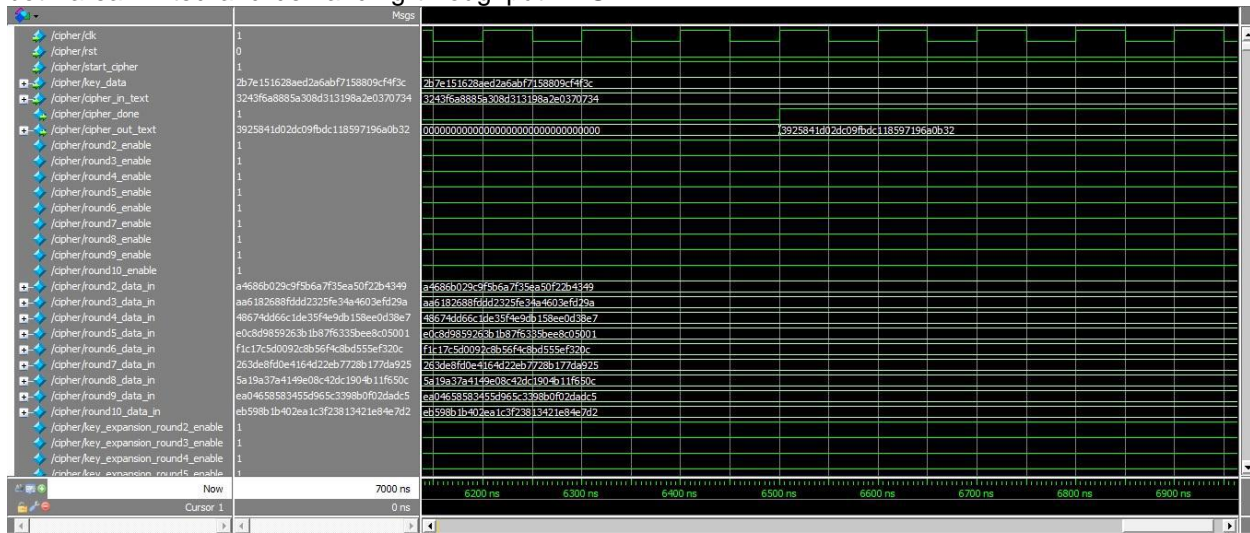


Fig.5. Simulation result for Encryption

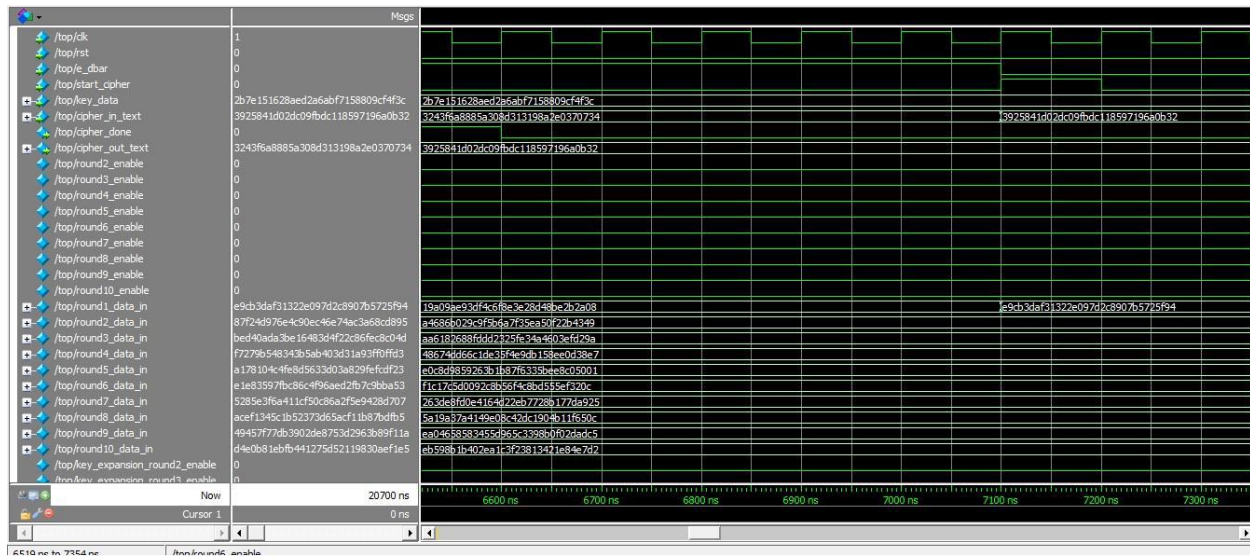


Fig.6. Simulation result for Joint Encryptor and Decryptor

References:

1. AES page available via <http://www.nist.gov/CryptoToolkit>.
2. Computer Security Objects Register (CSOR): <http://csrc.nist.gov/csor/>.
3. J. Daemen and V. Rijmen, AES Proposal: Rijndael, AES Algorithm Submission, September 3, 1999, available at [1].
4. J. Daemen and V. Rijmen, The block ciphers Rijndael, Smart Card research and Applications, LNCS 1820, Springer-Verlag, pp. 288-296.
5. B. Gladman's AES related home page http://fp.gladman.plus.com/cryptography_technology/.
6. A. Lee, NIST Special Publication 800-21, Guideline for Implementing Cryptography in the Federal Government,

National Institute of Standards and Technology, November 1999.

7. A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography, CRC Press, New York, 1997, p. 81-83.
8. J. Nechvatal, et al., Report on the Development of the Advanced Encryption Standard (AES), National Institute of Standards and Technology, October 2, 2000.