IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Design and Verification of VLSI Based AES Crypto Core Processor Using Verilog HDL

Dr.K.Padama Priya[*1], N. Deepthi Priya[2]
[*1,2] Dept of ECE,JNTU Anantapur, India
kesaripadmapriya@yahoo.com

Abstract

Advanced Encryption Standard (AES), has received significant interest over the past decade due to its performance and security level. In most of the previous works subbytes and inverse subbytes are implemented in Separate Modules using lookup table method. In this paper we used combinational logic which helps for making inner round pipelining in an efficient manner. Furthermore, composite field arithmetic helped in obtaining lesser area. Using proposed architecture, a fully sub pipelined encryptor/ decryptor with 3 substage pipelining in each round can achieve a throughput of 25.89Gbps on Xilinx xc5vlx110t-1 device which is faster.

This AES design was implemented using Verilog HDL and synthesized with Xilinx ISE using Spartran3 Xilinx Family , Simulation and Verification was done using Mentor-Graphics ModelSim-6.5e and achieved the maximum through put..

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 and in transit data. Encryption is the transformation of plain data (known as plaintext) into unintelligible data (known as ciphertext) through an algorithm referred to as cipher.

There are numerous encryption algorithms that are now 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) has published the specifications of this encryption standard in the Federal Information Processing Standards (FIPS) Publication 197. [1]

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 should be impractical to retrieve the plaintext solely based on the ciphertext 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, is always vulnerable to attacks.

There are other important drawbacks in software implementation of any encryption algorithm, including lack of CPU instructions operating on very large operands, word size mismatch on different operating systems and less parallelism in software. In addition, software implementation does not fulfill the required speed for time critical encryption applications. Thus, hardware implementation of encryption algorithms is an important alternative, since it provides ultimate secrecy of the encryption key, faster speed and more efficiency through higher levels of parallelism.

Different versions of AES algorithm exist today (AES128, AES196) depending on the size of the encryption key. In this project, a hardware model for implementing the AES128 algorithm was developed using the SystemVerilog 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.

AES algorithm provide more physical security as well as higher speed. Three architectural optimization approaches can be employed to speed up the hardware implementations: pipelining, subpipelining, and loop-unrolling. Among these approaches, the subpipelined architecture can achieve maximum speedup and optimum speed–area ratio in non-feedback modes. In order to explore the

advantage of subpipelining further, each round unit needs to be divided into more substages with equal delay. However, the SubBytes and the InvSubBytes in the AES algorithm are traditionally implemented by look-up tables (LUT)The Large and growing number of internet and wireless communication users has led to an increasing demand of security measures and devices for protecting the user data transmitted over the open channels. Two types of cryptographic systems are mainly used for security purpose, one is symmetric-key crypto system and other is asymmetric-key crypto system. Symmetric-key cryptography (DES, 3DES and AES) uses same key for both encryption and decryption. The asymmetric-key cryptography (RSA and Elliptic curve cryptography) uses different keys for encryption and decryption. The major disadvantage of DES is its key length is small. In November 2001, the National Institute of Standards and Technology (NIST) of the United States chose the Rijndael algorithm as the suitable Advanced Encryption Standard (AES) to replace previous algorithms like DES algorithm.

The rest of the paper is organized as follows. Section II describes basic AES algorithm. Section III describes novel on-the-fly key expansion module. Section IV describes pipeline design. Section V describes comparison work. Finally we concluded the paper in section VI

AES Alogoritham

The AES algorithm is a symmetric block cipher that processes data blocks of 128 bits using a cipher key of length 128, 192 bits. In addition, the AES algorithm is an iterative algorithm. Each iteration can be called a round, and the total number of rounds, Nr, is 10, 12, or 14, when the key length is 128, 192, or 256 bits, respectively. Table 1 shows the number of rounds as a function of key length.

AES Version	Key Length	Block Size	No of Rounds
AES 128	4	4	10
AES 192	6	4	12

Table 1 – AES Variations

The basic processing unit for the AES algorithm is a byte. As a result, the plaintext, ciphertext 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 *an* , 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, $0 <= n < 24$

The 128-bit data block is divided into 16 bytes. These bytes are mapped to a 4x4 array called the State and the state undergoes all the internal operations of AES algorithm. The transformations performed on the state are similar among all AES versions but the number of transformation rounds depends on the cipher key length

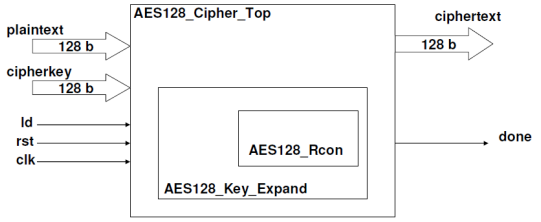

Figure 1 – AES128 Block Diagram

The final round in all AES versions differs slightly from the first Nr −1 rounds as it has one less transformation performed on the *State*. Each round of AES cipher (except the last one) consists of all the following transformation:

- SubBytes( )
- ShiftRows( )
- MixColumns( )
- AddRoundKey ( )

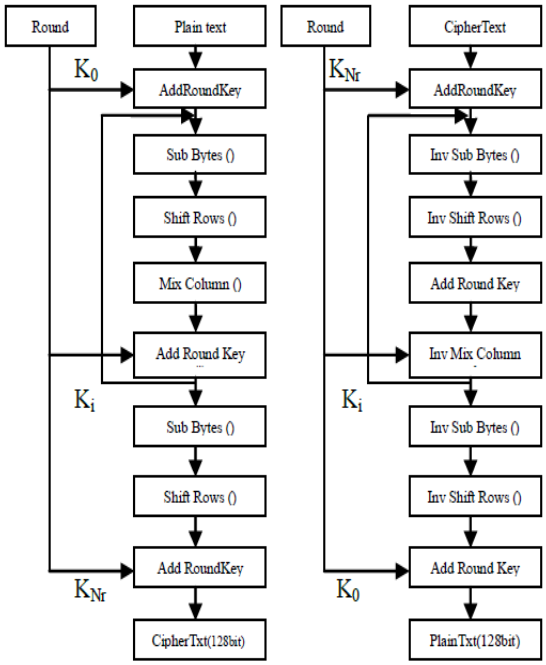

Figure 2 – AES Encryption & description

The AES cipher is described as a pseudo code in Figure 2. [1] As shown in the pseudo code, all the Nr rounds are identical with the exception of the final round which does not include the *MixColumns* transformation. The array with represents the round keys that are generated by the key expansion routine.

```
Cipher(byte PlainText[4*Nb], byte
CipherText[4*Nb], word w[Nb*(Nr+1)])
begin
```

```
byte state[4,Nb]
state = in
AddRoundKey(state, w[0, Nb-1])
for round = 1 step 1 to Nr–1
SubBytes(state)
ShiftRows(state)
MixColumns(state)
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
```

Figure 3 – AES128 Cipher Pseudo Code

After an initial round key addition, a round function consisting of four different transformations sub-bytes, shift-rows, mix-columns, and add-round-key are applied to the data block in the encryption procedure and in reverse order with inverse transformations in Decryption procedure. But last round in encryption contains only sub bytes, shift rows and add round key. Last round in decryption contains only inverse sub bytes, inverse shift rows and add round key. Four transformations in a round function are examined and optimally designed to achieve efficient implementation.

A. SubByte/Inv SubByte transformations

Subbyte transformation is a non linear byte substitution. This can be done by using two methods.

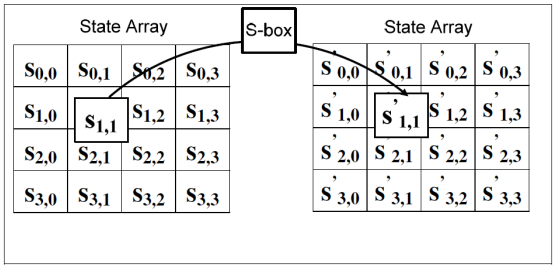

Figure 4 – AES128 Sub Bytes

In LUT based approach, the unbreakable delay of lookup tables is greater than the other logic. By using LUT method it is difficult to use sub pipeline structure with two pipeline stages, which prevents the further speedup. An alternative method is to use combinational logic, which is faster than the LUT and can also be divided into two pipeline stages, allowing further speedup. In non LUT method sub bytes can be implemented by finding multiplicative inverse followed by affine transform. Similarly inverse sub bytes implemented by using inverse affine transform followed by multiplicative inverse. Here multiplicative inverse is common; by taking this advantage we can implement a single structure for both subbytes and inverse

B Shift Rows

ShiftRows is a simple shifting transformation. First row of the state is kept as it is, while the second, third and fourth rows cyclically shifted by one byte, two bytes and three bytes to the left, respectively. In the InvShiftRows, the first row of the State does not change, while the rest of the rows are cyclically shifted to the right by the same offset as that in the ShiftRows.

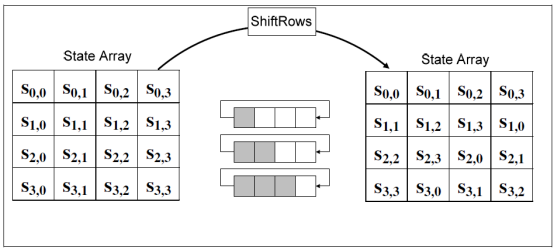

Figure 5 – AES128 Shift Rows

C. MixColumn/InvMixColumn transformation

The MixColumns() transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over GF(28) and multiplied modulo x4 + 1 with a fixed polynomial a(x), given by a(x) = {03}x3 + {01}x2 + {01}x + {02} .

The function *xtime* is used to represent the multiplication with _02‘, modulo the irreducible polynomial m(x)= x8 + x4 + x3 + x + 1. Implementation of function *xtime*() includes shifting and conditional xor with _1B‘. Fig. 4 shows the mixed column module.

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.

D. Add Roundkey

Add RoundKey involves only bit-wise XOR operation. After every round output of the mixcolumn is added with round key.

The round key values are added to the columns of the state in the following way

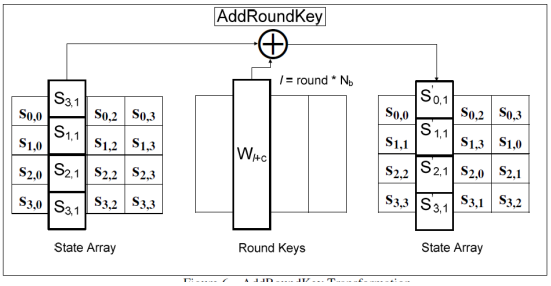

Figure 6 – AES128 Add Round Key

During the *AddRoundKey* transformation, the round key values are added to the *State* by means of a simple *Exclusive Or* (XOR) operation. Each round key consists of Nb words that are generated from the KeyExpansion routine.

By inverting the encryption structure one can easily derive the decryption structure. However, the sequence of the transformations will be different from that in encryption. This feature prohibits resource sharing between encryptors and decryptors.

Key Expansion

In the AES algorithm, the key expansion module is used for generating round keys for every round. There are two approaches to provide round keys. One is to pre-compute and store all the round keys, and the other one is to produce them on-the-fly. First approach consumes more area. In second approach, the initial key is divided into *Nk* words (key0, key1,…, keyNk-1) which are used as initial words. With the help of these initial words rest the words are generated iteratively. It can be computed that is 4, 6, or 8, when the key length is 128, 192 or 256-bit, respectively. Each round key has 128 bits, and is formed by concatenating four words.

The AES algorithm requires four words of round keys for each encryption round. That is total of 4*(Nr + 1) round keys considering the initial set of keys required for the first *AddRoundKey* transformation. All the round keys are derived from the cipher key itself.

According to the Federal Information Processing Standards (FIPS) Publication 197 [1], there is no restriction on the cipher key selection, as no week cipher key has been identified for the AES algorithm. The expansion of the cipher key into the round keys is performed by the KeyExpansion algorithm as shown in the pseudo code in Figure 7. [1]

The key expansion procedure can be described by the pseudo code listed below

```
KeyExpansion(byte CipherKey[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
word temp
i = 0
while (i < Nk)
w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
i = i+1
end while
i = Nk
while (i < Nb * (Nr+1)]
temp = w[i-1]
if (i mod Nk = 0)
temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
else if (Nk > 6 and i mod Nk = 4)
temp = SubWord(temp)
end if
w[i] = w[i-Nk] xor temp
i = i + 1
end while
end
```

Figure 7 – AES Key Expansion Pseudo Code

In the above pseudo code, the array represents the round keys that are generated by the KeyExpansion routine and Nk represents the size of the cipher key. Depending on the version of the AES algorithm, Nk=4, 6 or 8. The first Nk words of the expanded key are filled with the cipher key

Results

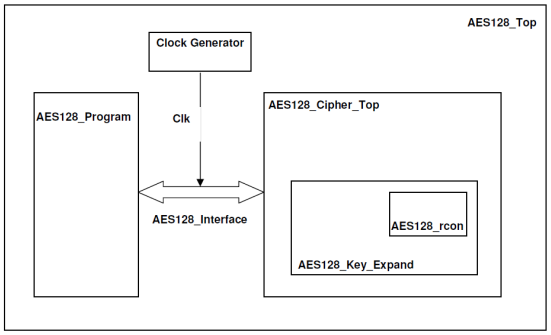

Figure 8 – AES128 Test Bench Infrastructure

The AES architecture was implemented using *Verilog HDL*, and simulated using *Mentor Graphics Modelsim.* Here we implemented two types of designs. AES is pipelined implementations.

Conclusion

In this paper, we presented a efficient pipeline AES architecture which includes both encryption and decryption. Also sub pipelining architecture helped us to get higher throughput than earlier implementations. The design is modeled using *Verilog HDL* and simulated with the help of *Model sim*. Synthesis is done by using Xilinx ISE 9.10

References

[1] J.Daemen and V.Rijmen, AES Proposal: Rijndael, AES algorithm submission,‖ September 3, 1999, available: ttp://www.nist.gov/CryptoToolkit.

[2] Draft FIPS for the AES,‖ available from: http://csrc.nist.gov/encryption.aes , February 2001

[3] Advanced Encryption Standard (AES), Nov. 26, 2001.

[4] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA implementation and

performance evaluation of the AES block cipher candidate algorithm finalist. presented at Proc. 3rd AES Conf. (AES3). [Online]. Available: http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html

[5] V. Fischer and M. Drutarovsky, "Two methods of Rijndael implementation in reconfigurable hardware," in Proc. CHES 2001, Paris, France,May 2001, pp. 77–92.

[6] A.M.Deshpande, M.S.Deshpande and .N.Kayatanavar,"FPGA Implementation of AES Encryption and Decryption"IEEE Inter.Conf.Cont,Auto,Com,and Ener., vol.01,issue04, pp.1-6,Jun.2009