A Low-Cost Implementation of Advance Encryption Standard

Tien-Luan Vu, Van-Quy Quach, Duy-Hieu Bui, Xuan-Tu Tran

SIS Laboratory, VNU University of Engineering and Technoglogy, 144 Xuan Thuy, Hanoi, Vietnam

E-mail: {luanvt_56, quyqv_56, hieubd, tutx}@vnu.edu.vn

Abstract The fast development of the Internet raises the concerns about the secure of the sensitive information such as passwords, bank accounts, personal information... Advanced Encryption Standard (AES) is currently considered as best symmetric-key block cipher which can be used to protect these data. AES has been proved to takes years to break. However, AES implementations in software require more computations and time to encrypt and decrypt the data. To improve the throughput and reduce the power consumption of the AES crypto system, in this work, we proposed a combined and low-cost AES encryption and decryption hardware architecture supporting all key-lengths as specified in the AES Standard [6]. The design was modeled in VHDL and successfully synthesized using Xilinx Virtex 5 FPGA Chip 5VSX50TF with maximum frequency of $92MHz$ and the maximum throughput of $300Mbps$.

Keywords AES, DES, FPGA

1. Introduction

Since the early days of communication, humans had been interested in how to secure information among parties especially in the field of military. This leads to the advent of many security methods and the most popular way is encrypt the messages with a secret key. Only the parties having the correct key can access the message. There are many cryptography systems based on this model, for example, substitution cipher, transposition cipher, Caesar crypto system and Affine crypto system... However, with the fast development of the computer systems, these weak cryptography systems are easily broken.

Nowadays, with the popularity of the shared network and the Internet, the confidentiality of the information is even more important. The sensitive data such as passwords, bank accounts, bank transaction, personal information, confidential documents... should not be exposed to untrusted parties. That's why in 2001, AES was standardized by US National Institute of Standard and Technology (NIST) and then included in ISO/IEC 18033-3 to replace the DES crypto system with the more secure algorithms and to utilize the feature of modern processors. AES has the block-size of 128-bits (doubled the block size of DES) and it supports the key length of 128-bits, 192-bits, and 256-bits. AES is now used in a wide variety of applications including HTTPS protocol, disk encryption, smart-card and so on. With the correct usage, AES has been proven to be secure for a wide range of applications. However, because of its big block size and its long key length, AES implementations in software are insufficient and lead to the reduction in the system throughput especially in the embedded environment. This also increases the power consumptions of the system.

In this work, we proposed a combined and low-cost AES encryption and decryption architecture which supports all the modes specified in the AES standard. In this architecture, the encryption and decryption procedures are integrated into one single module. The proposed architecture is modeled in VHDL and it is successfully synthesized into hardware using Xilinx ISE 14.7 and the Xilinx FPGA chip XC5VSX50T with the maximum operating frequency of $92MHz$ and the maximum throughput of $300Mbps$.

2. Introduction to AES

2.1. Block Cipher

Block cipher is a symmetric key cipher operating on a fixed-length group of bits, so called blocks. The modern design of the block cipher is based on the concept of the iterated product cipher which carries out encryption in multiple rounds, each of which uses a different sub-key derived from the original key. Block cipher has outstanding advantages such as simplicity, high speed and efficiency. In addition, in block cipher, the transmission in one cipher text block have no effect on other blocks. For these reason, block cipher is widely used in application sensitive with delay and mobile application.

Nowadays, there are a number of block ciphers which were developed and applied in many different fields, for examples, Lucifer (1969), DES (1977), Madryga (1984), NewDES (1985), AES (2001)... In most of them, the round transformation uses the Feistel network, therefore, typical part of the bits of the immediate state are simply transposed (unchanged to another position). However, in AES algorithm, substituted-permuted network is used, so every bit of the state is treated in the similar way and they are permuted and substituted together. Hence, the non-linearity and diffusion of AES is increased. In addition, the large key-length with a large number of the round functions also improves the safety of this algorithm.

2.2. AES algorithm

2.2.1. The overall structure of AES

AES is a symmetric block cipher which has the block size of 128 bits. It supports three key lengths of 128, 192, and 256 bits and contains respectively 10, 12, and 14 rounds for each type of the key lengths.

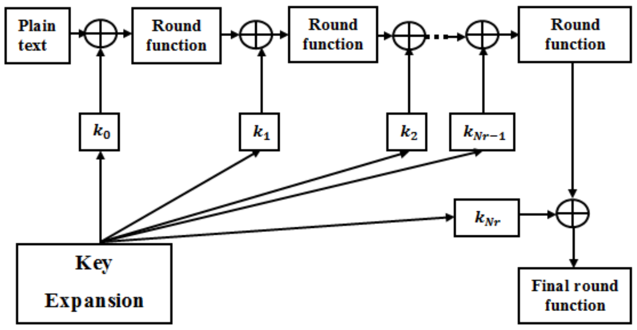

Figure 1: The AES algorithm.

As shown in Figure 1, the AES algorithm is described in both the encryption and decryption procedures. Each of the encryption or decryption procedures consists of *Nr* rounds depending on the key length. A typical round function processes the state matrix in four main transformations: *SubBytes, ShiftRows, MixColumns* and *AddRoundKey* (in the encryption) or *InvSubBytes, InvShiftRows, InvMixColumns* and *AddRoundKey* (in the decryption). At the start of the algorithm, the plain text or the cipher text and the cipher key are XORed together by executing the *AddRoundKey* transformation. In addition, the last round of the algorithm does not contain *MixColumns* or *InvMixColumns* as the typical rounds. *KeyExpansion* recieves cipher key, then expands throughout the *Nr* rounds in the algorithm. The decryption procedure is executed in the reversed order in comparison with the encryption procedure. The details of these transformations are explained in the following sub-sections.

2.2.2. Round Function

SubBytes and InvSubBytes

Each byte in the state matrix is replaced with its corresponding byte value in a look-up table called S-box or Inverse S-box described in the AES Standard [6]. The S-box (or the Inverse S-box) values for each byte is obtained by determining the multiplicative inverse over *GF*(2^8) followed by an affine transformation.

ShiftRows and InvShiftRows

The bytes in each row of the state matrix are cyclically shifted by a certain offset. Row n is cyclically shifted by $n - 1$ bytes. The first row is left unchanged. Each row is shifted to the left in the encryption or to the right in the decryption.

MixColumn and InvMixColumn

In this transformation, each 4-byte column of the state matrix is considered as a four-term polynomial over *GF*(2^8) and multiplied modulo $x^4 + 1$ with a fix polynomial *c(x)* or *d(x)* in the encryption or the decryption respectively. Note that *c(x)* and *d(x)* are multiplicative inverse.

AddRoundKey

Each byte in the state matrix is added to the corresponding byte of the Round Key in $GF(2^8)$. The Round Key is the processed key matrix corresponding to the current round.

2.2.3. Key Expansion

The implementation of the *Key expansion* procedure is illustrated in Algorithm 1. The *Subword* function is similar to *SubByte* function. The *Rotword* function performs a cyclic permutation to the left. *Rcon* is the round constants defined in the AES Standard [6].

The first *Nk* words of the expanded key are filled with the Cipher Key. After that, each word *w[i]* is equal to the XOR of the previous word *w[i-Nk]* and the word of *Nk* position earlier, *w[i-Nk]*. If the expanded sub-keys are in position that is multiple of Nk, a transformation is applied to *w[i-1]* followed by an XOR with an *Rcon[i]* (the round constant, which are defined in the AES Standard

Algorithm 1: Pseudo code for Key Expansion

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

[6]). This transformation consists of a cyclic shift of the bytes (*Rotword*), follow by a substitution of all four bytes (*Subword*). Otherwise, *Subword* is applied to *w[i-1]*.

3. Related works

Because of the secure properties, AES has gained a great deal of attention with many researcher to implement it securely. Nowadays, there are two main trends to implement the AES standard in hardware which is the high performance applications and the low cost design.

The high performance applications focus on maximizing the speed of the overall system which can be achieved at the cost of the hardware resources. Following this trend, in [5], Trang *et al* used the lookup tables to implement the S-Box function with 128-bit data path. This architecture obtains a very high speed (1188*Mbps*) with very low latency (13-clock cycle for encryption and 25 clock cycle for decryption). In addition, Wang and his colleagues in [7] propose a multi-core AES architecture which can achieves an throughput of 1830 Mbps with 3-AES cores. It supports the original AES algorithm and also provides flexibility to configure the parameters of each of the four transformations defined in the AES algorithm. However, this architecture has a large occupied area (27561 slices). Hernandez in [4] uses a true bit-serial approach, incorporate most of the AES algorithm complexity in a controller, components are reused. Therefore, this design can obtain a small area and a throughput of more than 7*Gbps*.

The low cost design aims at minimizing the usage of hardware resources at the cost of execution speeds. The low cost designs are useful in small devices such as mobile phones, smart card etc. Some typical works in this trend are the work in [2] by Chodowiec *et al* and the work in [1] by Adib *et al.* The design in [1] uses Dual-Port RAMs to storing all the results of the fix operations and Digital Clock Manager to optimize the execution time. In addition, this works uses 32-bit words that helps to decrease the memory cost and the area of this design. This architecture only consumes 326 slices plus 3 Block Random Access memories. However, its throughput is only 270*Mbps*. In [2], Chodowiec *et al* use new way to implement MixColumns and InvMixColumns transformations using shared logic resource to decrease the cost of memory usage. In addition choosing 32-bit data path to perform AES algorihm also optimize the area of the system. By following this methods. this work only uses small resources of 222 Slices and 3 Block RAMs, but it can encrypt and decrypt data of 160*Mbps* which satifisfies the needs of most embedded application.

In this paper, we decided to design a low cost application with all cases of the key length specified in the AES standard. Our design uses 32-bit input, and RAMs in *SubByte* and *InvSubByte* transformation. The proposed architecture will be presented in the next section.

4. Proposed Architecture

In this work, we propose an combined AES architecture which can do both encryption and decryption with all key lengths specified in the AES Standard. Our proposed architecture uses 32-bit data path and the iterative loop method to save the hardware resources. There are three main components in our architecture: the state transformation module, key expansion module and the control module. The proposed architecture are described in Figure 2.

The inputs to the system are the plain text for the encryption or the ciphered text for the decryption (*input*), the secrete key (*key_in*), the key mode (which indicates the different length of the encryption key *key_mode*), the operating mode (*enc/dec*) and the enable signal (*enable*). The output is the encrypted messages (in the encryption mode) or the plain text (in the decryption mode). The inputs and the secrete key are further divided into 32-bit blocks. The system will

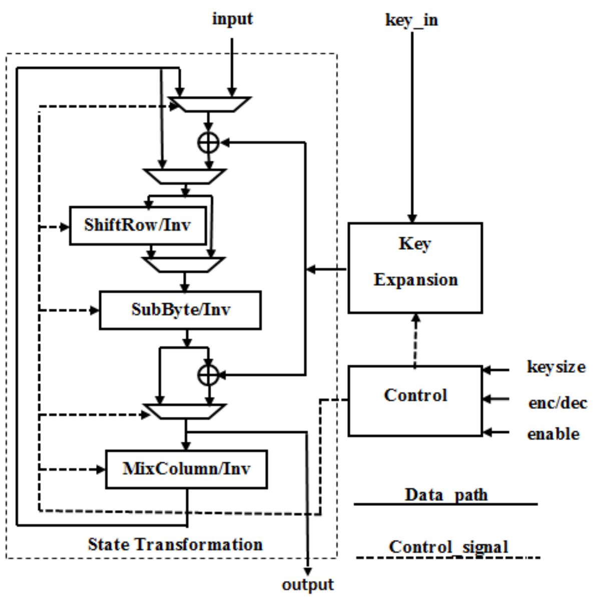

Figure 2: Our proposed architecture.

process each 32-bit block at a time. Based on the operating modes and the key length, the control module will create the correct operation for the state transformation and the key expansion modules. The secrete key is expanded on the fly in the encryption mode, but in decryption mode, expanded keys are stored in a RAM before be sent to the round functions. Along with the expanded key, the input will be encrypted/decrypted using the round function.

4.1. Round Function implementation

The round function is implemented such that it can be used for both encryption mode and decryption mode. Because of that, each pair of the forward and inverse transformations is implemented in the same stage of the state functions. In the encryption mode, *ShiftRows*, *SubBytes*, *MixColumns* are executed respectively. Otherwise, the data is brought through *InvShifRows*, *InvSubBytes*, and *InvMixColumns* to get the plain text. The detailed data flow of the round function is described in Figure 3.

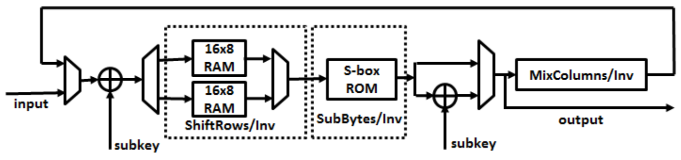

Figure 3: The Round Functions Architecture.

In the encryption mode, comparing with the standard algorithm, *ShiftRows* and *SubByte* are reordered. This modification does not affect the correctness of the algorithm because both *ShiftRows* and *SubBytes* operate on single byte. The details of each state round function will be shown as follows.

ShiftRows and InvShiftRows implementation

In the standard alogrithm, *ShiftRows* and *InvShiftRows* require 128-bit state for transformations, but our data path is only 32-bit, therefore, the storage for 128-bit intermediate processing is needed. To implement the *ShiftRows* and the *InvShiftRows*, we use two 16×8 RAMs. The state information of the AES algorithm will be stored in these two RAMs. The *ShiftRows* and *InvShiftRows* are done by reading the first RAM by the diagonal and writing to the second RAM by column. After each round of transformations, the data will be read from the previously written RAM in the previous round and the new transformed data are written to the RAM which has just been read from.

SubBytes and InvSubBytes implementation

SubBytes and *InvSubBytes* transformations typically substitute a single byte by applying a multiplicative inverse and affine transformation in GF(2^8). In this design, four 512×8 ROMs are used to implement both *SubBytes* and *InvSubBytes* transformations. Each ROM stores a pre-compute substitution value of the 8-bit data in both *SubBytes* and *InvSubBytes* transformations. This takes advantages over the on-fly calculation technique to decrease the latency and the complexity of the design. Because there are four bytes are substituted at a time, our architecture needs 4 ROMs for this stage.

MixColumns ann InvMixColumns implementation

In these transformations, the state's column represented in the form of four-term polynomial will be multiplied with the polynomial *c(x)* or the polynomial *d(x)*. These polynomials are predefined in the AES Standard: the polynomial *c(x)* for *MixColumns* and the polynomial *d(x)* for *InvMixColumns*. Both of the polynomial *c(x)* and the polynomial *d(x)* are multiplicative inverse. The equations of $c(x)$ and $d(x)$ are described in Equation 1 and Equation 2. All the coefficients are in the hex format in the Galois Field $GF(2^8)$.

$$c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{1}$$

$$d(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \tag{2}$$

The implementation of *MixColumns* is fairly simple because all coefficients of the polynomial *c(x)* are small. In contrast, the *InvMixColumns* implementation is far more complex because *d(x)*'s coefficients are much larger than *c(x)*'s coefficients. Chodowiec *et al.* in

[2] proposed a method to utilize the resource sharing technique for these functions. It derives as follow:

$$c(x) \otimes d(x) = \{01\} \tag{3}$$

Multiplying both side of above equation with *d(x)*, obtain:

$$c(x) \otimes d^2(x) = d(x) \tag{4}$$

where

$$d^2(x) = \{04\}x^2 + \{05\} \tag{5}$$

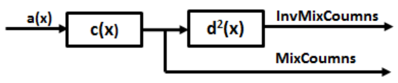

Figure 4: Sharing logic in *MixColmns* and *InvMixColumns*.

Two coefficients of the polynomial *d^2(x)* is much smaller than the coefficients of the polynomial *d(x)*. In addition, two other coefficients are equal to zero. Figure 4 shows the implementation of the *MixColmns* and the *InvMixColumns* in our design.

4.2. Key Expansion

Our design supports all three key lengths in both the encryption mode and the decryption mode. Thus the key expansion implementation needs to flexibly adapt to these features. The expanded key values are computed by combining both the on-the-fly and pre-computed methods. In encryption mode, the on-the-fly computation of the round key paralleling with state transformation is efficient because it takes the advantage of changing key fast with low delays. On the other hand, in the decryption mode, round keys are applied for state transformation in reverse order, from the final expanded key value to the first original value. Therefore, in the decryption mode, the pre-computed round keys are used.

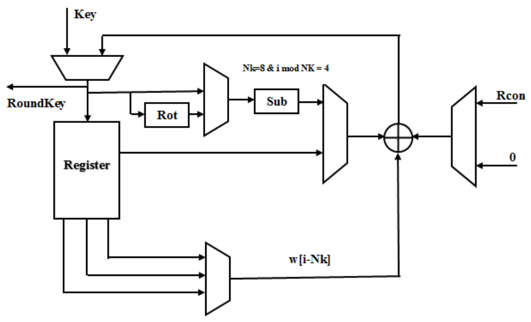

Figure 5: The Key Expansion Block.

The structure of the *Key Expansion* module is shown in Figure 5. This consists *RotWord* and *SubWord* implementations. *RotWord* is simply a circular right shift operation. *SubWord* is similar to the *SubBytes* transformation so four additional *S-box* ROMs are required. Additionally, a shift register of length 7 is used to store 3, 5 or 7 previous words depending on the key length. These words are used to compute the next round key as specified in the AES Standard. In the worst case of the decryption mode, we will need to store the 256-bit key and fourteen 128-bit expanded key of it. This leads to the need of 64 32-bit words to store the expanded key and the original key. The key words are sent directly to the state transformation module in encryption mode, while, in the decryption mode, the key words are pre-computed and stored in a 64×32 RAM before it is used by the state transformation module.

5. Implementation results

Table 1: Utilization result

Logic utilization	Used	Available	Utilization(%)
Number of occupied slices	1134	8160	13
Number of slice registers	2544	32640	7
Number of LUTs	3820	32640	11

The proposed architecture is modeled in VHDL, simulated using ModelSim, and synthesized by Xilinx ISE 14.7. The implementation results are presented in Table 1. Our architecture occupies 13% (1134 slices) of the resources of the FPGA chip XC5VSX50T with about 2607 slice registers and about 3820 LUTs. The maximum operating of the design is about 92*MHz* and its maximum throughput is nearly 300*Mbps*.

Table 2 shows the comparison of our architecture with the works in [1], [2] and [7]. It is clear that our implementation is not the best one in term of the hardware cost when compared with the works in [1] and [2]. However, these two designs partly support the AES Standard. Our implementation have the highest operating frequency among four works. The maximum throughput of our work is approximately 300*Mbps* which is much higher than the works in [1] and [2], however, it is much smaller than the one in [7] which is a multi-core architecture. Our implementation takes the least number of cycles to complete the encryption among these works.

6. Conclusion

The Advanced Encryption Standard (AES), from its announcement in 2001, has become the de-facto data encryption standard. It is used in a wide variety of applications including security for communication, disk encryption, files and data encryption and so

Table 2: Implementation result

Design	[1]	[2]	[7]	Our
Device	XC3S 500E	XC2S 3000	XC2V 6000	XC5V XS50T
Mode	Enc	Enc/Dec	Enc/Dec	Enc/Dec
Keylength	128	128	128/192/ 256	128/192/ 256
Slices	326 + 3 BRAM	222 + 3 BRAM	27561	1134
Frequency (*Mhz*)		60	49.75	92
Cycles	80	40	41/45/49	40/44/48
Througput (*Mbps*)	270	166	1830	300
Efficiency (*Mbps/Slice)*	0.828	0.747	0.06	0.265

on. In this work, we have designed a low-cost AES encryption and decryption architecture. This design supports both encryption and decryption in a single data path with all key lengths specified in the AES standard. It uses a 32-bit data path, reordered processing in the round function and the share resources and some on-fly calculations to minimize the hardware cost and increase the execution speed. Our design is modeled in VHDL, simulated and verified using ModelSim from Mentor Graphics, and then synthesized and implemented using Xilinx ISE 14.7 on the FPGA chip XC5VSX50T. The implementation results show that the proposed AES architecture has a small area footprint of nearly 1134 slices and the maximum frequency of 92*MHz*. This leads to the maximum system throughput of nearly 300*Mbps*. In the near future, we would like to integrate our AES encryption and decryption modules into a complete system-on-a-chip to fully evaluate the performance of the proposed architecture.

References

[1] Samir El Adib and Naoufal Raissouni. Aes encryption algorithm hardware implementation architecture: Resource and execution time optimization. *International Journal of Information and Network Security (IJINS)*, 1(2):110–118, 2012.

[2] Pawe Chodowiec and Kris Gaj. Very compact fpga implementation of the aes algorithm. In ColinD. Walter, etinK. Ko, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 319–333. Springer Berlin Heidelberg, 2003.

[3] Chih-Peng Fan and Jun-Kui Hwang. Implementations of high throughput sequential and fully pipelined aes processors on fpga. In *Intelligent Signal Processing and Communication Systems, 2007. ISPACS 2007. International Symposium on*, pages 353–356, Nov 2007.

[4] Orlando J. Hernandez, Thomas Sodon, Michael Adel, and Nathan Kupp. A Low Cost Advanced Encryption Standard (AES) Co-processor Implementation. *Journal of Computer Science & Technology*, pages 8–14, 2008.

[5] Trang Hoang and Van Loi Nguyen. An efficient fpga implementation of the advanced encryption standard algorithm. In *The 2012 IEEE International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, pages 1–4, Feb 2012.

[6] National Institute of Standards and Technology. Advanced encryption standard. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`, 2001.

[7] Mao-Yin Wang, Chih-Pin Su, Chia-Lung Horng, Cheng-Wen Wu, and Chih-Tsun Huang. Single- and Multi-core Configurable AES Architectures for Flexible Security. *IEEE Trans. Very Large Scale Integr. Syst.*, 18(4):541–552, April 2010.