A Low-cost and High Efficiency Architecture of AES Crypto-engine

Yan Qing. Zhong*, Jian Ming. Wang**, Z. F. Zhao**, D.Y. Yu**, L. Li**
**Vinno Technologies Inc.
* Beijing Normal University
Beijing, China

Abstract—**The growing market of WPAN has led to an increasing demand of security measures and devices for protecting the user data transmitted over the open channels. Advanced Encryption Standards (AES) is the basic security approach for WPAN. To meet the low cost, low power feature and high security demand of WPAN, a low cost, high efficient AES core is proposed in this paper. To achieve low cost, methods of integration and resource sharing are used in designing a very low-complexity architecture, especially in (inverse) byte substitution ((inv) SubBytes) modules and (inverse) mix column ((inv) MixColumn) modules, etc. Further more, AES Encryptor and Decryptor is integrated into a full functional crypto-engine. This very low-cost and high efficiency AES core of IEEE 802.15.4-2006 is designed and emulated on Xilinx FPGA. Simulation results show that this kind of design can be used in resource critical applications, such as smart card, PDA and mobile phones.**

Keywords-AES; S-box; Sub-Byte; WPAN

I. INTRODUCTION

Compared with other communication technologies, wireless access is the most competitive. However, efficient security is not easy to be provided. In wireless network, invalid access is much easier for hostile devices and more difficult to be detected than in those based on cable connection. Thus security is a key problem of 802.15.4-2006.

IEEE 802.15.4-2006 Standards is the amended version of IEEE 802.15.4b known as ZigBee, which defines the MAC and PHY layer of a low rate, low cost and low power WPAN. In its MAC sub layer, AES is chosen to be the basic security approach. Until now, many AES software cores were designed, several papers were published. However, in software approach, the secret key is vulnerable to attacks and relies on underlying programs. In addition, achievable speed by software implementation is not acceptable for internet applications such as routers. This leads to hardware design of AES where parallel processing, pipelining and higher level of security is possible. Therefore, hardware systems provide more superior performance with higher throughput [1].

AES is a complex computational algorithm, which requires lots of resource and power consumption in application. Before this implementation, many high-speeds, high throughput hardware designs of AES were also available. However, most of them didn't take area into account enough. This may lead to the contradiction between low power consumption featured application and hardware design. As a solution, we applied many methods of reduce saving, such as resource sharing and integration, to simplify our design. AES Encrypter and Decrypter was integrated into a whole functional core, which can run in both encryption and decryption mode. To make the AES core most suitable for WPAN, other methods of resource reducing and power saving are also used. Emulations on Xilinx ISE and Modelsim show that: both low cost and high efficiency are achieved in AES crypto-engine (referred to as VAES in the following).

This paper is organized as follows. Section 2 presents an overview of AES algorithm. Section 3 discusses methods to reduce the hardware cost and power consumption, as well as the hardware solution of AES system. In section 4, the hardware consumption and test result is provided. Section 5 concludes the paper.

II. OVERVIEW OF AES ALGORITHM

An AES system is a symmetric-key system in which the sender and receiver of a message share a single, common key, which is used to encrypt and decrypt the message. An AES algorithm can be divided into two main blocks: encryption/decryption and key schedule. Scheduled key should be ready no later than encryption/decryption operation starts. Therefore, key schedule operation often takes place no later than encryption and decryption occurs. Encryption and Decryption are inverse operation. For both encryption and decryption process, 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 (ShiftRow) of the State array by different offsets, 3) mixing the data within each column of the State array (MixColumn), and 4) adding a Round Key (AddRoundKey) to the State [2]. The basic processing unit of data is block or state, which can be 128 bits, 192 bits or 256 bits. To achieve the highest throughput, WPAN uses 128 bits block.

AES is referred to as AES-128 in following sections if not mentioned. Fig. 1 illustrates the processing of AES encryption.

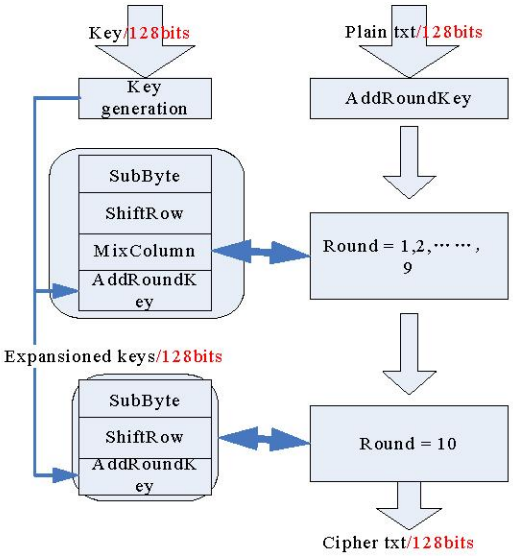

Figure 1: AES-128 operation and transformations of round function

A. *The Handling of Key Schedule*

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 4*11 words: the algorithm requires an initial set of 4 words, and each of the 10 rounds requires 4 words of key data. The resulting key schedule consists of a linear array of 4-byte words, denoted [Wi], with i in the range $0 < i < 44$.

The expansion of the input key into the key schedule proceeds according to Fig. 2:

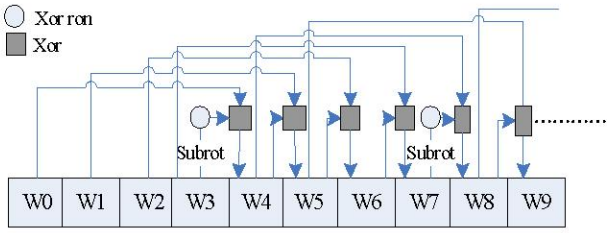

Figure 2: Key Expansion routine

Subrot: SubByte((rotword(w[i])).

B. *The Handling of AES Cipher and Decipher*

For full understanding, a simple introduction for each of these 4 transformations is described as follows:

1) SubByte and inv-SubByte: SubByte is the most complex and cost-required part in both cipher and decipher. There are mainly 2 ways to implement it: 1) use substitution table, as mentioned in [1]; 2) first take the multiplicative inverse in the finite field (GF(2^8)), then apply affine transformation to it. There are also other ways to design SubByte function, as in [8], but they might be too slow to meet our speed requirement.

2) ShiftRow and inv-ShiftRow: Linear diffusion process, operating on individual rows. Depending on the row location, offset of left shift varies from zero to three bytes in AES encryption operation, as in Fig. 3. In decryption process, the operation is reversed.

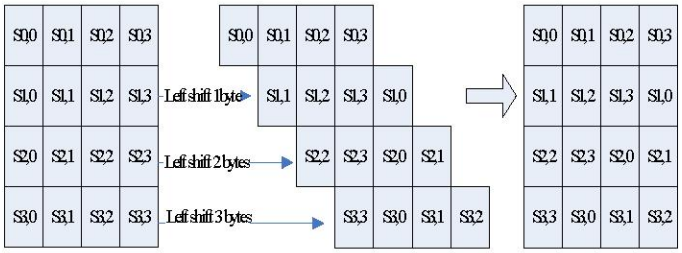

Figure 3: Shift row operation of AES encryption.

3) MixColumn and inv-MixColumn: Matrix multiplication over GF (2^8) (GF is short for Galois Field). Column vector is multiplied with a fixed matrix where the bytes are treated as a polynomials rather than numbers.

In encryption process, the columns are multiplied with a matrix of the same size; while in decryption process, the columns are multiplied with another matrix.

4) AddRoundKey: perform a byte-oriented xor between state and key. This operation is reversible, the same in both cipher and decipher.

III. The Hardware Solution of Low-Cost and High Efficient AES

AES algorithm is quite complex and resource consuming, at the same time WPAN requires a low-cost, low-power, low-complexity design. If AES is accomplished without optimization, it might cost a large extra amount of area, which wouldn't be accepted by WPAN compatible transceiver chip. To achieve the required security level without sacrificing too much resource, many different kinds of resource saving methods are investigated. To achieve this, an important issue is to integrate Encrypter and Decrypter together and it could function between encryption and decryption mode. Other resource sharing and resource re-using methods are also applied. Finally, the hardware architecture of AES system is proposed.

After a full investigation to encryption and decryption process, we found that we can adjust the sequence of decryption and make it the same as sequences of encryption [5]. By doing this, resource sharing between encryption and decryption becomes possible. Resource sharing is achieved by firstly combined transformations and inverse transformations within rounds; then integrates cipher and inverse cipher by adjusting the inverse transformations' order. After that, other resource re-using and power saving methods are also applied to the AES crypto-engine. At last, low cost hardware implementation architecture is given.

A. *Integration of Transformations within Rounds*

1) Resource Sharing between SubByte and InvSubByte unit:

SubByte and inv-SubByte are the most crucial parts in our design. As discussed in section 2.2, if the first method is used to design SubByte function, two different tables are needed. This denotes that two different ROMs (256*8 bits) are required to store the tables [1]. Moreover, for a parallel architecture of AES design, it usually needs several tables. For example, in a high-speed implementation of AES-128, 20 S-box modules and 16 inverse S-box modules are demanded. In which, there are 16 S-box modules used in implementation of function SubBytes. Four S-box modules are used in implementation of function KeyExpansion. And 16 inverse S-box modules are used in implementation of function InvSubBytes. In this case, a substantial amount of hardware resource will be occupied if SubBytes and InvSubBytes use respective tables in encryption and decryption. It is desirable to obtain a simplified way so as to reduce the hardware complexity. The second SubByte method – "affine + public S-box" is chosen to design SubByte module. A public S-box, which can be used in both SubByte and inv-SubByte function, is used to substitute the GF (2^8) inverse operation. Further discussions on this issue will be provided in section C.

Therefore, SubByte can be implemented by S-box+ affine and inv-SubByte by inv-affine + S-box. In this sense both SubByte and inv-SubByte functions are fulfilled by only one ROM table, which will reduce a substantial amount of resources. The architecture of integrated unit is as follows. In implementation, a control signal is used to indicate encryption/decryption mode.

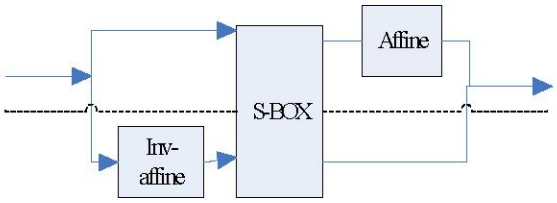

Figure 4: The integrated unit of subbyte and inv-subbyte

2) Integration of Mix column and Inv Mix column function: The function of mix/inv-mix column can be described by the following equation:

$$\text{Encryption output} = [2,3,1,1] * (b0, b1, b2, b3)^T; \quad (1)$$

$$\text{Decryption output} = [E,B,D,9] * (b0, b1, b2, b3)^T; \quad (2)$$

In [1], multiplied by 2 can be implemented by function xtime():

$$Out[7:0] = \{in[6:4], in[3:0] \text{xor} \{in[7], in[7], '0', in[7]\}, in[7]\}$$

Only 4 xor gates are needed to fulfill multiply 2, much lower than multiplier. If all the multiplication operation could be done by xtime(), then lots of gates can be saved. In this way, we decompose the equation above to be:

Encryption output Eoutput:

$$\text{Eoutput} = 2*(b0+b1)+b1+b2+b3 \quad (3)$$

$$= \text{xtime}(b0 \text{ xor } b1) \text{xor } b1 \text{ xor } b2 \text{ xor } b3 \quad (4)$$

Decryption output Doutput:

$$\text{Doutput} = 4*(2*(b0+b1)+2*(b2+b3)+(b0+b2)) + 2*(b0+b1)+b1+(b2+b3) \quad (5)$$

$$= \text{xtime}(\text{xtime}(b0 \text{ xor } b1) \text{xor xtime}(b2 \text{ xor } b3) \text{ xor}(b0 \text{ xor } b2)) \text{ xor Encryption output} \quad (6)$$

Thus, one byte MixColumn and inv MixColumn can be merged according to equation (6). Subsequently, to realize the overall integration of 4-byte MixColums and InvMixColumns transformations, first integrate four 1-byte MixColumns and InvMixColumns transformations, but only shift input data left cyclically (Fig. 8). The control signal En/dec determines the output data from MixColumns or InvMixColumns transformation. When work in encryption mode, the circuit runs in MixColumns mode, and outputs the data for encryption. On the contrary, the operation works in InvMixColumns transformation mode, and outputs the data for decryption [6].

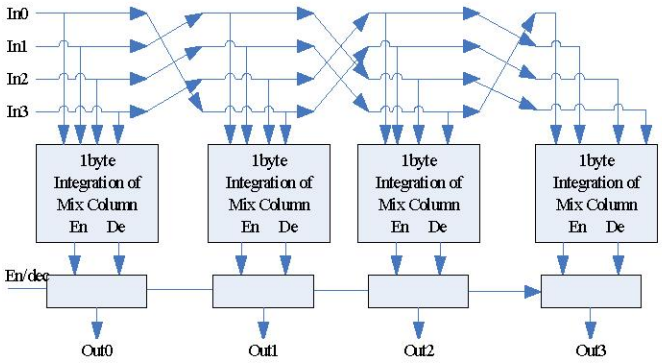

Figure 5: Integration of mix column and inv mix column [6].

B. *Resource Sharing of Encryption and Decryption*

To share resource between encryption and decryption, the decryption transformation order should be adjusted in accord with that of encryption. In deriving the equivalent structure of the inverse cipher, we make use of two properties of the component transformations.

First, the order of ShiftRow and ByteSub is indifferent. ShiftRow simply transposes the bytes and has no effect on the byte values. ByteSub works on individual bytes, independent of their position.

Second, the sequence

AddRoundKey(State,RoundKey);

InvMixColumn(State);

Can be replaced by:

InvMixColumn(State);

AddRoundKey(State,InvRoundKey).

with InvRoundKey obtained by applying InvMixColumn to the corresponding RoundKey. This is based on the fact that for a linear transformation A, A(x+ k) = A(x) + A(k) [4].

Now the same structure to implement both encryption and decryption can be applied to AES crypto-engine. The structure and equivalent structure of decryption are shown in following figure:

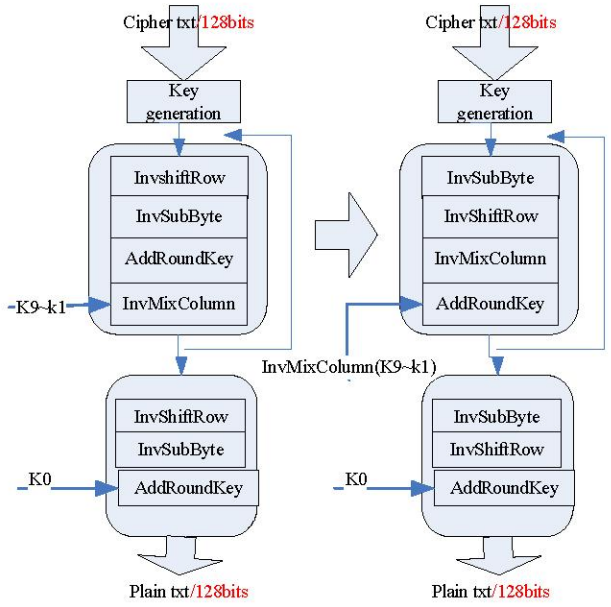

Figure 6: The inverse cipher process (the left one) and equivalent inverse cipher process

C. *Other Resource Sharing and Power saving methods*

In addition, different ways to save resources are adapted.

1) The re-use of SubByte component: As mentioned above, SubByte is used in both encryption and key expansion. Since key expansion is conducted before enc/dec module working, reuse of SubByte component between enc/dec module and key expansion module is possible. Separate SubByte component from the enc/dec module to be an independent component, start it by control signals. When key expansion is working, SubByte the key words, otherwise SubByte states.

2) The design of shift row and inv-shift row function: According to the algorithm, shift row function only changes the row orders, instead of value.

If register the state and shift states row by row, the algorithm will become very complex as well as resource consuming. In implementation, we just store the state before row shifting, and then outputs bytes according to the order of new state after row shifting/ inv-row shifting. By doing this, resource is reduced and speed is improved.

D. *The Hardware Architecture of AES*

1) Key Expansion: Key Expansion routine is the same for both cipher and inverse cipher; however, the usage in encryption and decryption is different. In encryption process, AES uses keys according to the schedule order; while in decipher, AES uses keys in reverse order.

Two options exist for designing key schedule function. One option is to generate all the required sub-keys in advance or at the very beginning. This design requires a huge buffer for storing all the expanded keys. Second option is to generate sub-key in parallel with the encryption round. The other method requires much less buffer space, since only one 128 bit sub-key is produced and stored. Plus, the cold start latency of sub-key generation is smaller for this method is much less. Unfortunately, the decryption unit uses the sub-keys in reverse order, forcing pre-generation of all sub-keys [3].

Because we are going to integrate encrypter and decrypter into one module, the second architecture of key expansion should be used. Therefore, one 16*11 bytes RAM is needed to pre-store scheduled keys. To enhance resource sharing, RAM can be the public storage so that it can be shared with other components in WPAN chip. According to the standard, the component is designed as following:

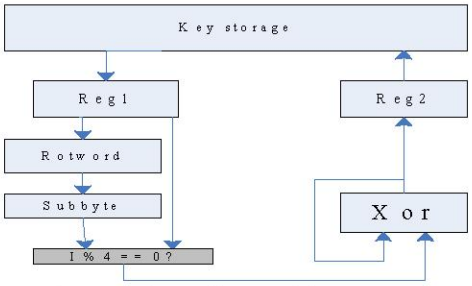

Figure 7: The architecture of key expansion

Reg1 is used to store W [i-4], Reg2 is used to store W [i-1]. Firstly read key word W [i-4] from key storage into Reg1, if i%4 ==0, conduct the rotword, SubByte operation and xor it with ron value; else go straight; after that, xor it with reg2, store the value of xor into reg2 and key storage.

2) Encryption and Decryption Operation: For a WPAN transceiver chip usually contains both encryption and decryption component and the two components do not run at the same time, both encryption and decryption mode can be integrated into a whole architecture. An encryption/decryption mode indication flag is used to indicate the operation mode. When enc/dec_direction flag indicates encryption mode, perform in encryption mode and output cipher txt; on the contrary, perform in decryption mode and output plain txt. The data path is as follows:

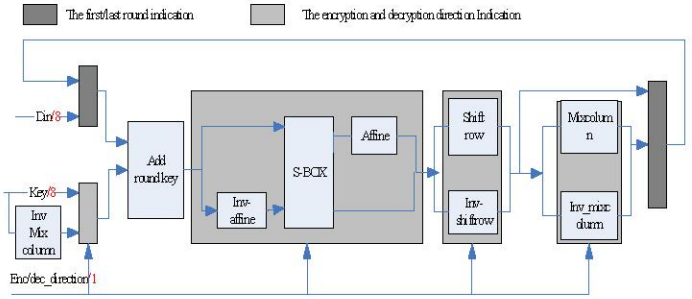

Figure 8: The integrated encryption and decryption module

Set flags such as first/last round、enc/dec_direction, handle data by the indication of these flags according to the algorithm.

As a combination of above methods, the whole hardware

architecture of AES algorithm is illustrated by the following figure:

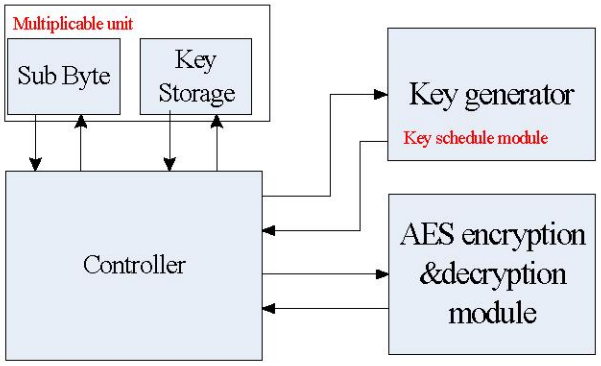

Figure 9: The overall architecture of AES crypto-engine
Subbyte module and key storage module are reused between key generator and AES encryption&decryption module. The enc/decryption process is controlled by controller.

IV. Performance

We implement VHDL coded AES module in Xilinx vertex II xc2v2000 series fg676-4 device. The synthesis result shows that the critical path was 7.801ns using low speed grade -4. The maximum operation clock frequency can reach up to 128 Mhz. The process data path width is 8 bits. It takes about 14us to complete an AES-128 stand-alone en/decryption on 50 Mhz clock frequency. The integrated design of AES-128 encryption and decryption results in a simply hardware of 12, 480 gate counts and a 256*8 bit rom with key expander included.

Compared to other standard AES cores, this AES core (VAES core) is much more resource saving and more user friendly. The following table is the resource and data throughput comparison between VAES core with GRAES designed by Gaiser Research and AES IP Core designed by ErSt Electronic GmbH. VAES core greatly saves the implementation cost by 14% and 19%, respectively; while with little sacrifice on data throughput.

TABLE I: Comparison between VAES and Other AES Cores

Core	Resource	Data throughput@ Maximum frequency
GRAES	14,500 gates	127 Mbps@125 Mhz
AES IP core	834 slices	1059Mbps@91 Mhz
VAES	675 slices(12,480 gates)	55Mbps@128 Mhz

VAES core was tested under Model Sim 6.0 environment. The AES core was tested as following process sequences: first, input test vectors [5] and control signals, start the engine. Second, check outputs and compare them with test vectors. The AES crypto-engine manifests a good performance through several times of testing.

V. Conclusion

In the innovative hardware implementation of AES, we integrated AES encryption and decryption operation into a single module, which can switch between encryption and decryption when required. By applying following methods, lots of resource and power was saved.

1. Integrate sub-modules of encryption and decryption, and then integrate AES en/decryption function into a full functional module which work under both encryption and decryption mode.

2. Furthermore, re-use (inv) SubByte module between key schedule and en/decryption module. The re-use of (inv) SubByte can save 50% resource than not re-used.

3. Clock gating and operand isolation are applied into our design to reduce the power of AES module.

The methods used for resource reducing, such as resource sharing, component re-using and integration, can also be applied in other hardware module designs. With the development of WPAN network and other low-cost, low range networks, the integrated low cost, low power AES crypto-engine will become more and more useful.

Reference

[1] Song J. Park, "Analysis of AES hardware implementations," Available at: http://islab.oregonstate.edu/koc/ece679/project/2003/park.pdf.

[2] J. Dijmen and V. Rijmen, "AES Proposal: Rijndael," NIST AES Proposal, June 1998. Available at: http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf.

[3] MooSeop Kim, Juhan Kim, and Yongje Choi, "Low power circuit architecture of AES Crypto module for wireless sensor network," transactions on engineering, computing and technology V8 October 2005 ISSN 1305-5313.

[4] A. Satoh, S. Morioka, K. Takano, S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," ASIC CRYPT 2001, Lecture Notes in Computer Science 2248, Springer, 2001, pp. 239-254.

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

[6] Hsin-Chung Wan, Chih-Hsiu, Lin An-Yeu Wu, "Design and implementation of cost-efficient AES cryptographic engine," Available at: http://www.eng.ntu.edu.tw/eng/chinese/bulletin/n88/n88-6.pdf

[7] Namin Yu, Howard M. Heys, "Investigation of compact hardware implementation of the Advanced Encryption Standard," Department of Electrical and Computer Engineering Memorial University of Newfoundland.

[8] V. Rijmen, "Efficient implementation of the Rijndael S-box". Available at: http://www.iaik.tugraz.at/research/krypto/AES/old/~rijmen/rijndael/sbox.pdf.

[9] Tsung-Fu Lin, Chih-Pin Su, Chih-Tsun Huang, and Cheng-Wen Wu, "A High-Throughput Low-Cost AES Cipher Chip", in Proc. 3rd IEEE Asia-Pacific Conf. ASIC, Taipei, Aug. 2002, pp. 85-88.

[10] Chih-Chung Lu and Shau-Yin Tseng, "Integrated Design of AES (Advanced Encryption Standard) Encryptor and Decryptor," Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors, Page: 277.