AES Algorithm Implementation using ARM Processor

T.Ravichandra Babu
Department of ECE
Sri Vasavi Engg.College
JNTUK University, Kakinada
Andhra Pradesh, India.

K.V.V.S.Murthy
Adhitya Educational Institutions.
JNTUK University,
Kakinada-East Godavari (Dist.)
Andhra Pradesh, India.

G.Sunil
Department of ECE
Sri Vasavi Engg.College.
JNTUK University, Kakinada
Andhra Pradesh, India.

ABSTRACT
The AES encryption/decryption algorithm is widely used in modern consumer electronic products for security. To shorten the encryption/decryption time of plenty of data, it is necessary to adopt the algorithm of hardware implementation; however, it is possible to meet the requirement for low cost by completely using software only. How to reach a balance between the cost and efficiency of software and hardware implementation is a question worth of being discussed. In this paper, we implemented the AES encryption algorithm with hardware in combination with part of software using the custom instruction mechanism provided by the ARM7 with keil platform. we explored various combinations of hardware and software to realize the AES algorithm and discussed possible best solutions of different needs.

General Terms

Algorithm, Embedded Systems and Applications

Keywords

Cryptography, AES, custom instruction, ARM processor

1. INTRODUCTION
With the rapid development of the Internet and e-Commerce, portable memory devices such as USB Disk, SD Card… are becoming increasingly popular; how to prevent the encryption system from being decrypted has become an important issue. As the length of the data block of the Data Encryption Standard (DES) algorithm is 64 bits only, and the length of the key is 56 bits only, it can no longer meet today's system needs.

Hence, National Institute of Standard and Technology (NIST) launched a campaign to solicit new encryption algorithm. After a string of evaluations, Rijmen's algorithm of Vincent Rijmen became the new coding standard and has replaced the existing symmetrical coding standard (DES)[4].

The AES algorithm is a round-based encryption/decryption algorithm and each round includes 4 operations: *AddRoundKey, ShiftRows, SubBytes* and *MixColumn*s. To shorten the encryption/decryption time of plenty of data, it is necessary to adopt the algorithm of hardware implementation; however, it is possible to meet the requirement for low cost by using software only.

In recent years, there have been plenty of literatures on hardware/software implementation of the AES algorithm. They can be divided into 3 types: (1) full software implementation on low cost devices that do not require high speed, (2) full hardware implementation SubBytes is the computation that requires most hardware. (3) Software/hardware co-design .implement part of the algorithm using hardware and the remaining algorithm using software so as to reach a balance between the cost and efficiency. To mix its computation, the hardware is turned into custom instruction to support the software, which is a feasible method. How to reach a balance between the cost and efficiency of software and hardware implementation is a question worth of being discussed. In this paper, we implemented the AES encryption algorithm with hardware in combination with part of software using the custom instruction mechanism provided by the ARM7 with Keil platform. We explored various combinations of hardware and software to realize AES algorithm and discussed possible best solutions of different needs. In the next section, we will briefly review the AES algorithm.

2. AES ALGORITHM

The Advanced Encryption Standard is a symmetric block cipher. The data block size is fixed to be128 bits, while the key length can be 128, 192 or 256 bits. The AES is a round-based algorithm. The number of rounds Nr is 10, 12, or 14, when the key length is 128, 192 or 256 bits, respectively. Each round of AES algorithm performs the three transformations: AddRoundKey, SubBytes, and ShiftRows. Except the final round, each round also performs Mixcolumns. The key used in each round, called as round-key, is generated from the initial key by a separate key scheduling module.

The 128 bit data block is divided into 16 bytes, which are represented by a 4X4 matrix of bytes. The entries are denotes by *S0,0, S0,1, S0,2, S0,3, S1,0,S1,1, S1,2, S1,3, S2,0, S2,1, S2,2, S2,3,S3,0 , S3,1, S3,2,* $S_{3,3}$. The matrix represents a state *S*. All the four transformations map an input state to an output state. The AddRoundKey involves only one bit-wise XOR operation between the state *S* and the round key. The ShiftRows cyclically shifts k bytes to the left on k^{th} row of the state matrix, k=0~3. The position changes to *S0,0, S0,1, S0,2, S0,3, S, S1,2, S2,2, S2,3, S2,0, S2,1, S3,1, S3,2, S33,* $S_{3,0.}$ The MixColumn uses each column of the state matrix as a polynomial over GF(2^8)and

multiples them modulo x+1 with a polynomial a(x) ={03}x + {01}x + {01}x + {02}.

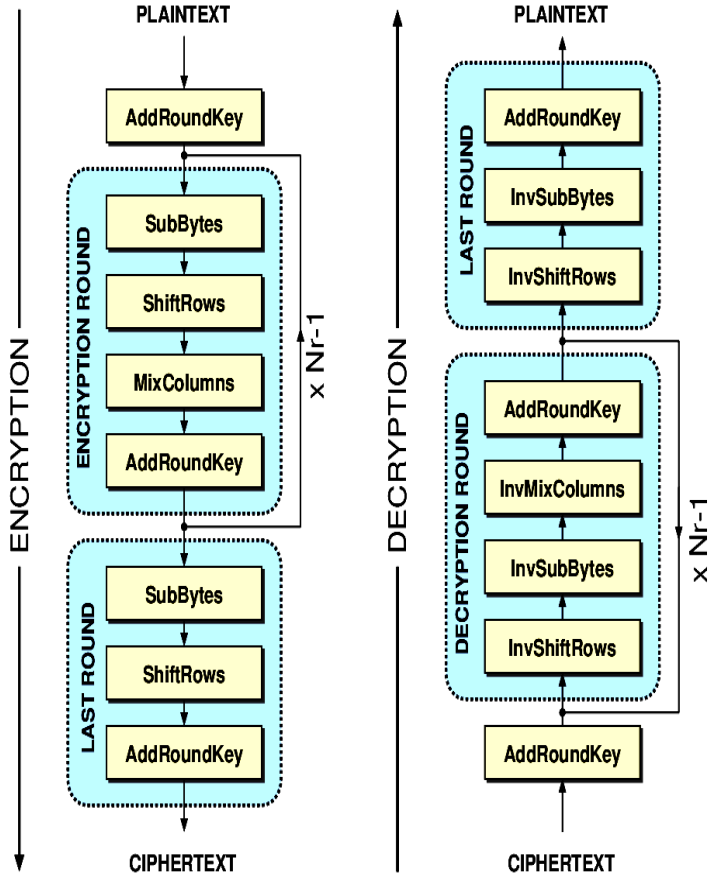


Figure 1.AES Encryption / Decryption flow

The SubBytes is a nonlinear transformation, which substitutes each byte of the state with its multiplicative inverse in $GF(2^8)$ and then performs an affine transformation. The irreducible polynomials $m(x) = x^8 + x^4 + x^3 + x + 1$ is used in the AES algorithm to construct $GF(2^8)$. The affine transformation consists of a bitwise matrix multiplication with a fixed 8x8 binary matrix followed by XOR with $\{63\}_h$. The module performing the SubBytes transformation is called as *SBOX*.

3. ROUNDKEY GENERATION

There are two main approaches to generate the round key used in the AES process. Keys can be generated on-the-fly by a concurrently executing data path that computes the next round key during the time the actual data path completes computing the current AES round. The second alternative is to pre-compute all roundkeys and store them in a roundkey memory.
A critical point in the implementation of a cryptographic system is the "key setup time" which is defined as the amount of time required to start cryptographic operations after a new cipherkey has been provided. On-the-fly key generators can be designed in a way to completely eliminate any latency overhead when changing cipherkeys, at least for encryption. For decryption, the first roundkey that is required is the last roundkey that has been used for encryption. Since the key expansion uses recursion, there is no simple way to obtain the last roundkey directly from the cipherkey. This must be done by computing all roundkeys for the encryption. The last roundkey so obtained can be used as an initial vector for the inverse key schedule.

The AES-128 mode requires 10 roundkeys with 128 bits. An on-the-fly key generator of a flexible AES implementation that supports both encryption and decryption for all standard key lengths needs to be able to store 256 bits of cipherkey, 128 bits for the roundkey, and finally 128 bits for the last roundkey. This is more than one fourth of the total amount of storage that is needed for all roundkeys. Consequently, pre-computing all roundkeys is not always a bad decision.

4. OPTIMISATION FOR ARM PROCESSOR

ARM is the leading provider of 32-bit embedded RISC microprocessors with almost 75% of the market. ARM offers a wide range of processor cores based on a common architecture [5] [3], delivering high performance together with low power consumption and system cost.

ARM processors implement a load/store architecture. Depending on the processor mode, 15 general purpose registers are visible at a time. Almost all ARM instructions can be executed conditionally on the value of the ALU status flags. Load and store instructions can load or store a 32-bit word or an 8-bit unsigned byte from memory to a register or from a register to memory.

The ARM arithmetic logic unit has a 32-bit barrel shifter that is capable of shift and rotate operations. The second operand to all ARM data-processing and single register data-transfer instructions can be shifted before data processing or data transfer is executed, as part of the instruction. The amount by which the register should be shifted may be contained in an immediate field in the instruction, or in the bottom byte of another register. When the shift amount is specified in the instruction, it may take any value from 0 to 31, without incurring any penalty in the instruction cycle time.

At first sight, the key expansion defined for AES , does not look hardware intensive. After all, only four *SubBytes* operations are required per AES round. However, the additional flexibility required to support all three key lengths results in a very cumbersome and slow implementation. For faster implementations with large parallel data paths, the critical path through the key generator is usually longer than the actual data path. For small implementations that use a data path of 32 bits or less, more area is required to implement a key generator than the actual data path.

5. HARDWARE & SOFTWARE IMPLEMENTATIONS

We will firstly describe main considerations in the hardware implementation and then in software implementation. The AddRoundKey operation involves only one bit-wise XOR operation. The MixColumn operation can be also implemented

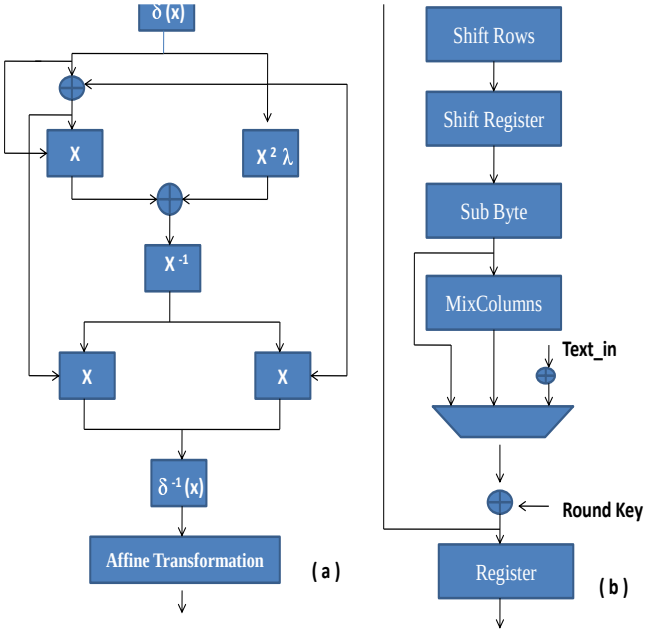
with XOR gates only [7]. The ShiftRows operation can be realized by wring. There are two approaches for designing S-Box circuits: (1) Table lookup and (2) Combinational circuit. The former uses ROM or RAM to store the table. In the latter design, the inversion in $GF(2^8)$ is the most complicated operation. To reduce the hardware complexity, the composite field arithmetic is exploited [11], by that the original inversion in $GF(2^8)$ is mapped to operations in composite field $GF((2^4)^2$. Basically, an element $a \in GF(2^8)$ is represented as a linear polynomial $a_h x+a_l$ with coefficients in $GF(2^4)$. Let us take the eight bits of $a \in GF(2^8)$ as $\{a_1, a_{2,}..., a_7\}$ and the four bits for a_h (a_l) as $\{a_{h3,}..., a_{h0}\}$ $(\{a_{l3,}..., a_{l0}\})$. Then the mapping can be computed as follows [3]:

$a_l = (a_{l0} = a_c \oplus a_0 \oplus a_5, a_{ll} = a_l \oplus a_2, a_{12} = a_A, a_{l3} = a_2 \oplus a_4, a_h = (a_{h0} = a_c \oplus a_5, a_{hl} = a_A \oplus a_c, a_{h2} = a_B \oplus a_2 \oplus a_3, a_{h3} = a_B)$,
where $a_A = a_1 \oplus a_7$, $a_B = a_5 \oplus a_7$, $a_C = a_4 . \oplus a_6$.

The inversion of $a_h x+a_l$ requires modular reduction to guarantee that the result is also a two-term polynomial. The irreducible polynomial $n(x) = x^2 + \{1\}x + \{e\}$ is used. Let '×' be multiplication. The inversion can be derived as follows:
$(a_h x+a_l)^{-1} = (ah \otimes d)x+(a_h \oplus a_l) \otimes d$,
where $d = ((ah^2 \otimes \{e\}) \oplus (a_h \otimes a_l) \oplus al^2)^{-1}$

Those operations can be reduced to the bit-wise logical AND and XOR functions. In this work, the operation x^2 mod m(x) and x{e} are merged into the following logic implementation:

Figure 2.Hardware implementation of (a) SBOX and (b)AES algorithm



$q0=a_1 \oplus a_2$, $q_1=a_0$, $q_2=a_B \oplus a_3$, $q_3=a_B$,
where $a_B=a_0 \oplus a_1$.

In the software part, AddRoundKey and SubBytes are based on individual bytes and it does not matter on how the data is arranged in the memory. However, since ShiftRows manipulate data in one row while MixColumn in one column, it is impossible for the two operations to read 4 bytes at one time. Since MixColumn involves more algorithmic actions, the original state matrix is transposed for simplifying MixColumn operations in paper [8]. However, this requires the modification of the key generation procedure. The approach in papers [9,10] combined the SubBytes and MixColumn as an extended SBox table. The extended SBox table are 32-bit, 256 word tables. generated by concatenating four values Si,j×{03}, Si,j×{02}, Si,j×{01} and S_{ij}×{01} of each SBox table output Si,j.

In this paper, SubBytes and Mix Columns are executed separately.

5.1. The Mixcolumn Transformation

The MixColumn transformation makes use of arithmetic operations

Table 1: Look-up tables used by different versions

Version	Encryption	Decryption
V1	Sbox	InvSbox
V1T	Sbox	InvSbox
V2	Sbox + Enc. table	InvSbox + Dec. table

in the finite field $GF(2^n)$. We assume that the reader has a basic background of Galois Fields, but for completeness we recall that addition in $GF(2^n)$ is equivalent to a simple bitwise XOR, while multiplication is obtained by reducing the result of standard multiplication (with XOR as sum) modulo a fixed polynomial. This polynomial must be irreducible to preserve the algebraic structure of field. In the MixColumn transformation, each column of the State is considered as a polynomial with coefficients in $GF(2^8)$, and multiplied modulo $x^4 + 1$ with a fixed polynomial $\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$, co prime to the modulo. Assuming that the column before transformation consists of the bytes (b0, b1, b2, b3), each byte representing a polynomial in $GF(2^8)$, the transformed column bytes (c0, c1, c2, c3) are computed as follows:

$C_0 = \{02\} \circ b_0 \oplus \{03\} \circ b_1 \oplus \{01\} \circ b_2 \oplus \{01\} \circ b_3$
$C_1 = \{01\} \circ b_0 \oplus \{02\} \circ b_1 \oplus \{03\} \circ b_2 \oplus \{01\} \circ b_3$
$C_2 = \{01\} \circ b_0 \oplus \{01\} \circ b_1 \oplus \{02\} \circ b_2 \oplus \{03\} \circ b_3$
$C_3 = \{03\} \circ b_0 \oplus \{01\} \circ b_1 \oplus \{01\} \circ b_2 \oplus \{02\} \circ b_3$

Where denotes polynomial multiplication in $GF(2^8)$ defined by the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$, and denotes simple XOR at byte level. Multiplication by {02} in $GF(2^8)$ can be implemented at byte level with a left shift followed by a conditional bitwise XOR with {1b}. Multiplication by larger coefficients can be implemented with repeated multiplications by {02} and XORs with previously calculated results.

6. CUSTOM INSTRUCTIONS

In this paper, we use ARM7 Processor to implement AES algorithm using custom hardware instructions. The advantages of custom instructions include the reduction of instruction sequence and the speed acceleration by hardware.

With the ARM processor development kits, we can convert one hardware circuit into a custom instruction and put it in the instruction set of the CPU. Depending on the data amount and execution, we can get the ARM supports four types of custom instructions: combinatorial, multi-cycle extended and register file. In this we are implementing AES algorithm using the custom instructions are ARM processor and keil compiler software with both hardware and software combinations.

7. EXPERIMENTAL RESULTS

We explored design space with a parameterized synthesizable design. Relevant programmable parameters include:

- ✓ *SW*, *TSBOX* or *GSBOX*: A user can choose software table(SW), pre-store hardware table (TSBOX), generating transformation by combinational logic to implement SBOX (GSBOX), which is realized by composite field arithmetic as stated in the third section.
- ✓ Number of SBOX: If using TSBOX or GSBOX, a user can choose how many SBOX to implement: 1, 4, 8 or 16.
- ✓ *MixColumn*: A user can choose whether to implement it using hardware.
- ✓ *ShiftRow+AddRoundkey*: A user can choose whether to implement it using hardware.

The custom instructions are implemented with ARM7 Processor development kit. In the experiment, ARM7 processor hardware in combination with software i.e. embedded C language developed using keil software. The time is measured for running32 packets of data, each having 128 bits. The round keys are pre-calculated using the same implementation (either look-up table or combinational logic) as used in the data path. After the round keys have been prepared, the 32 packets are encrypted sequentially.

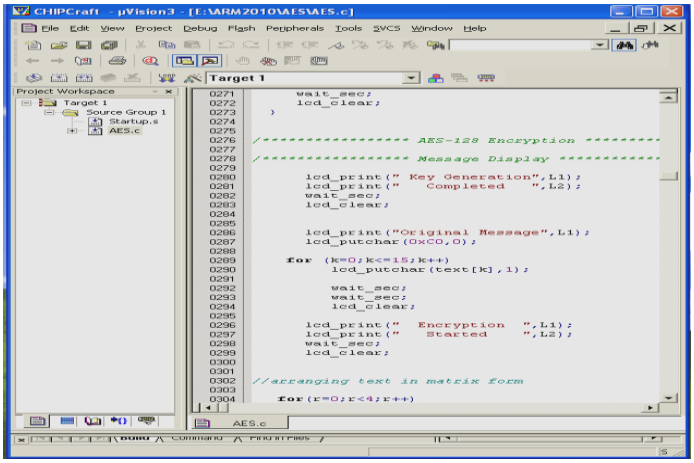


Figure 3.AES algorithm source code

The source code which is developed in embeddedC language the fig 3 represents example for the source code of AES encryption and decryption algorithm.
After developing the source code ,burn the programming into the ARM processor by using the flash burner called Philips flash utility V2.2.3.

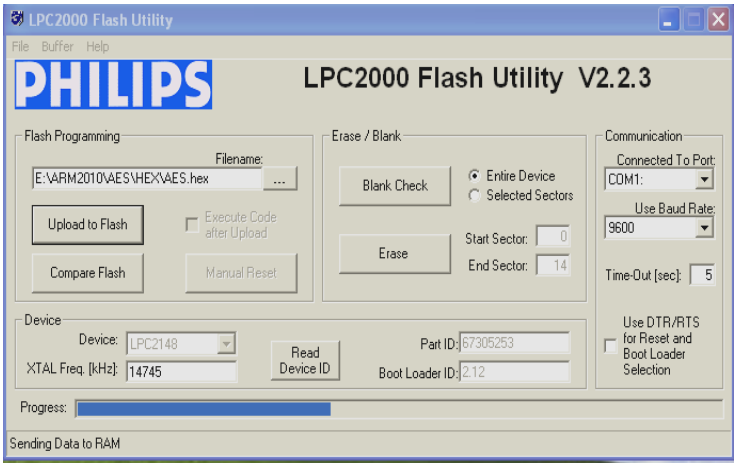


Figure 4. Burning process

The fig 4 represents the example of the dumping(burning) the program into the ARM processor.

When the burning process is completed, go to the hyper terminal and transmit the data. The fig 5(A) & (B) represents the hyper terminal link to transmit the data to the processor.

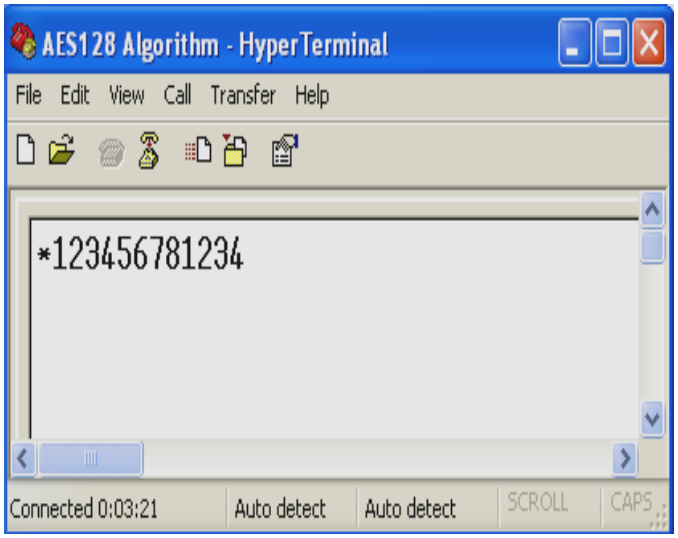


Figure 5(A).Hyper Terminal

Figure 5 (B).LCD display data

When the data is received then the key generation process is calculated. The number of rounds is depending up on the size of the data. The number of rounds Nr is 10, 12, or 14, when the key length is 128, 192 or 256 bits, respectively.

The fig 6 represents the examples of AES key generation method. Here the generated key length is depend on the data block size. For example The number of rounds is 10, 12, or 14, when the key length is 128, 192 or 256 bits, respectively

Figure 6.AES key generation

When the key generation is completed; the first method in AES algorithm i.e. encryption method is started. Here the original data which transmitted in the form of encrypted. The fig 7 represents the example of the encryption form.

Figure 7. Encryption process

After completion of the encrypted form then we perform the decryption form of the encrypted data. At finally the original message is received. The fig 8 represents the example of final received original data. By using this process we can provide high security for transmitting the data.

Figure 8.Received original message window

Earlier this algorithm is implemented on ALTERA Nios II platform, with a parameterized synthesizable design. But when we are using ARM processor it can provide more security & accuracy for data transmission

8. ACKNOWLEDGMENTS

The authors would like to thank Dr.Syed S Basha, Chairman of The RISE, for the invaluable support during the development of this work.

9. CONCLUSION

The AES encryption/decryption algorithm is widely used in modern consumer electronic products for security. In this paper, we have implemented the AES encryption and decryption algorithm with hardware in combination with part of software using the custom instruction mechanism provided by the ARM7 With a language of embedded C using of keil platform , we explored various combinations of hardware and software to realize the AES algorithm and discussed possible best solutions of different needs.

REFERENCES

[1] ARM Architecture. Reference Manual ARM DDI 0100D, ARM Limited, Feb 2000.

[2] Arm Ltd. website. http://www.arm.com

[3] ARM7. Data Sheet ARM DDI 0020C, ARM Limited, Dec 1994.

[4] AN 188: Custom Instructions for the Nios Embedded Processor, Altera Corporation

[5] B. Gladman. A Speci cation for Rijndael, the AES Algorithm. Available at http://fp.gladman.plus.com, May 2002

[6] G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti and S. Marchesin, "Efficient Software Implementation of AES on 32-bit Platforms," *CHES 2002, LNCS 2523*, pp. 159–171, 2003.

[7] Intel Strong ARM SA-1110 Microprocessor. Developer's Manual 278240-003, Intel Corporation, Jun 2000.

[8] I. Verbauwhede, *et al., Design and Performance Testing of a 2.29-GB/s Rijdael Processor, IEEE JSSC*, vol. 38,no. 3, pp569-572, 2003

[9] Intel Ltd. website.http://www.intel.com.

[10] J. Daemen and V. Rijmen. Rijndael, the Advanced Encryption Standard. Dr. Dobb's Journal, 26(3):137{139, Mar. 2001.

[11] K. Nadehara, M. Ikekawa and I. Kuroda, "Extended Instructions for the AES Cryptography and their Efficient Implementation," *Signal Processing Systems,*

2004. SIPS 2004. IEEE Workshop on, pp. 152- 157, 13-15 Oct. 2004

[12] NIST, "Advanced Encryption Standard(AES)," FIPS PUBS 197, Nov. 2001

[13] Satoh, S. Morioka, K. Takano and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," *ASIACRYPT 2001, LNCS, vol. 2248*, pp. 239-254, 2001

[14] S. Tillich, J. Großsch¨adl and A. Szekely, "An Instruction Set Extension for Fast and Memory-Efficient AES Implementation," *J. Dittmann, CMS 2005, LNCS 3677*, pp. 11–21, 2005.

[15] X. Zhang and K. K. Parhi: High-Speed VLSI Architectures forthe AES Algorithm, *IEEE Transactions on VLSI Systems*, vol.12, Issue 9, pp. 957-967, Sept.2004