A High Performance Sub-Pipelined Architecture for AES

Hua Li and Jianzhou Li
Department of Mathematics and Computer Science
University of Lethbridge
Canada T1K 3M4

Abstract

In this paper, an efficient sub-pipelined architecture for AES is proposed. It can do both encryption and decryption with well evenly divided three-stage pipeline. The three-stage pipelined key expansion module generates the corresponding subkeys concurrently for encryption or decryption. The design can operate in CBCk mode and process three blocks of data simultaneously. The proposed architecture is simulated in Verilog HDL and implemented using Xilinx Virtex II FPGA device. The comparison indicates that our design has a relatively low area and high throughput up to 1.57Gbits/s.

Keywords: AES, sub-pipelined architecture, FPGA, cryptography

1 Introduction

The Advanced Encryption Standard (AES) cryptosystem, founded in 2000 by NIST and developed by Joan Daemen and Vincent Rijmen [1, 2, 3], is in a state of constant improvement, especially when considering hardware implementations. For example, ASIC designs for AES [4, 5, 6, 7] and FPGA implementations of AES [8, 9, 10, 11] are proposed to speed up the performance.

AES is a symmetric block cipher that performs four operations on a 128-bit block of data for a certain number of repetitions. These operations are Inv-/SubBytes, Inv-/ShiftRows, Inv-/MixColumns and AddRoundKey [2, 3]. In this paper, we propose a high performance sub-pipelined AES architecture for 128-bit cipher key. It is a three-stage pipelined design which processes three blocks of data in parallel and provides a throughput of 1.57Gbits/s with a throughput per area rate of 0.765Mbps/slice when implemented in Xilinx Virtex II FPGA. Compared with previous work, the proposed design can do encryption and decryption in the same module, and the throughput is faster than [9, 10]. In addition, the design can easily be extended to reconfigurable architecture for 128-, 192-, and 256-bit cipher keys with only modification on key generation module.

This paper begins with a brief introduction of AES operations in section 2. The main focus of the paper is presented in section 3, where the new architecture is proposed. Section 4 analyzes the proposed design and compares it with previous relevant work. Section 5 then concludes this paper.

2 AES operations

In AES, all the intermediate results of the 128-bit block as well as the input and the output block are called states. The most intuitive way for AES operation is to picture each state as a 4×4 matrix of bytes which are filled in the matrix column by column. At each stage of the transformation between plaintext and ciphertext, the block of data is transformed from its current state to next new state, depending on which operation is used.

SubBytes operation is a nonlinear substitution that performs on each byte of the state using S-box which contains a permutation of all possible 256 8-bit values. The S-box is constructed by first calculating the multiplicative inverse of each byte in $GF(2^8)$ with the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ and then applying an affine transformation expressed as $b' = b_i \oplus b_{(i+4)\ mod\ 8} \oplus b_{(i+5)\ mod\ 8} \oplus b_{(i+6)\ mod\ 8} \oplus b_{(i+7)\ mod\ 8} \oplus c_i$ for $0 \leq i < 8$, where b_i is the i^{th} bit of the byte, and c_i is the i^{th} bit of a constant byte c with the hexadecimal value $\{63\}$. The inverse of this operation, InvSubBytes, consists of applying the inverse of the affine transformation followed by taking the same multiplicative inverse in $GF(2^8)$.

The Inv-/ShiftRows operation is the cyclic shifting of each row of the state to the left on encryption or to the right on decryption. With the top row being identified as row 0, and the bottom row as row 3, the offset of each row shifted corresponds to the row number. For instance, the top row does not shift at all, where the next row shifts one byte in the supposed direction.

Inv-/MixColumns operation treats each column of the state as a four-term polynomial over $GF(2^8)$ and transforms each column to a new one by multiplying it with a constant polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ modulo $x^4 + 1$. The inverse MixColumns operation is a multiplication of each columns with $b(x) = a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$ modulo $x^4 + 1$.

The AddRoundKey operation is only a simple logical XOR of the current state with a round key that is generated by the key expansion. XOR operation is its own inverse.

Because the sequence of transformations for decryption is different from that for encryption, two different modules are needed for applications that require both encryption and decryption. As the result of the commutativity of InvShiftRows and InvSubBytes operations, and of the distributivity of InvMixColumns over the AddRoundKey operation, Equivalent Inverse Cipher in Fig. 1 is used where encryption and decryption are in the same order using appropriate forward or inverse operations [2, 3]. In this method, an additional InvMixColumns operation must be applied to all round keys for decryption with the exception of initial and last one before they are added to the states. The normal round in Fig. 1 repeats nine times for 128-bit cipher key AES (AES-128), eleven times for 192-bit cipher key AES (AES-192) and thirteen times for 256-bit cipher key AES (AES-256) respectively.

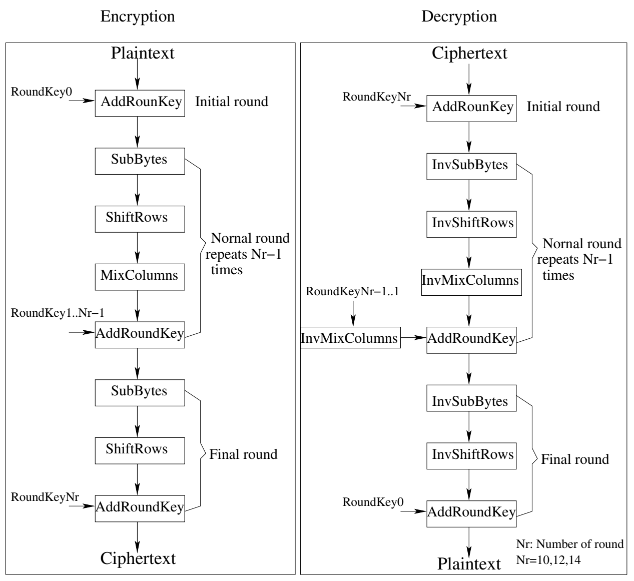

Figure 1. AES algorithm flow for encryption/decryption

In key expansion, the initial 128-, 192- or 256-bit cipher keys need to be expanded to respective eleven, thirteen or fifteen 128-bit round keys in order to perform AddRoundKey operation for each round of encryption. The pseudocode for the key expansion algorithm is described in the following [2].

```
$for(RC[1] = $‘01’$, i = 2;\ i < 11;\ i++)$
    $\{RC[i] = RC[i-1] *$ ‘02’;
    $Rcon[i] = (RC[i],$ ‘00’, ‘00’, ‘00’$);\}$
$for(i = 0;\ i < Nk;\ i++)$
    $W[i] = (key[4*i], key[4*i+1], key[4*i+2], key[4*i+3]);$
i=Nk;
$while(i < 4*(Nr+1))$
$\{$   $temp = W[i-1];$
    $if(i\ mod\ Nk == 0)$
        $temp = SubWord(RotWord(temp)) \oplus Rcon[i/Nk];$
    $else\ if\ (Nk > 6\ and\ i\ mod\ Nk == 4)$
        $temp = SubWord(temp);$
    $W[i] = W[i-4] \oplus temp;$
    $i = i + 1;$
$\}$
```

In the key expansion algorithm, Nk is the number of 32-bit words comprising the cipher key. Nk can be 4, 6 or 8. Nr is the number of rounds which equals to 10, 12, or 14 for AES-128, AES-192 and AES-256 respectively. Every four keys of initial byte array key[0..(4×Nk-1)] are concatenated into initial 32-bit word keys W[0..(Nk-1)]. The (Nr+1) 128-bit round keys are obtained sequentially from every four words in the word array W[0..(4×Nr+3)]. For words in positions that are a multiple of Nk, RotWord which is a one-byte circular left shift on the words and SubWord which applies the S-box operation on each byte of its input word are performed, and the result is XORed with corresponding Rcon from Rcon[1:10]. For AES-256, the words on positions i which satisfies that (i-4) is a multiple of Nk need only to perform Subword operation. The first Nk-word key corresponds to the cipher key and all subsequent Nk-word keys are derived recursively from their respective predecessors. The same serial Nk-word keys are used in reversed order for decryption and all these keys can be derived from the last round of Nk-word key using the inverse operations [2, 3].

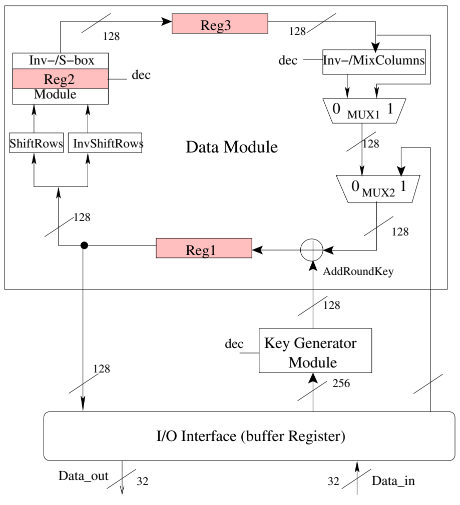

Figure 2. The three-stage sub-pipelined architecture for AES

3 Design of sub-pipelined architecture for AES

The Equivalent Inverse Cipher is adopted in our design, so that the operations on both encryption and decryption can be applied in the same order using the appropriate forward or inverse operations determined by the signal "dec". Fig. 2 illustrates the proposed three-stage sub-pipelined architecture which iterates 10, 12, 14 times to perform encryption/decryption for 128-, 192-, and 256-bit cipher key length. The sub-pipeline structure, in which each round separated by inner pipelining register, can decrease the delay of the pipeline stages and substantially increase the throughput. Three pipelined stages are applied in our design to achieve more even critical path and higher working frequency.

The proposed architecture mainly consists of three modules. The first module is *Data Module* where the blocks of data flow continuously along the path. The second one is *Key Generator Module* where corresponding round keys are provided for Data Module simultaneously. The third one is *I/O Interface Module* that deals with data input and output.

The CBCk mode, as the expansion of CBC (Cipher Block Chain) mode, can also be implemented in the proposed sub-pipelined architecture. In CBCk mode, the ith plaintext is XORed with the $(i-k)$th $(k \geq 3)$ ciphertext before encryption is implemented.

3.1 Data Module

Data Module in Fig. 2 includes Inv-/SubBytes, Inv-/ShiftRows, Inv-/MixColumns and AddRoundKey operations. For Inv-/S-box operation, we use an efficient combinational circuit proposed by J. Wolkerstorfer et al. [4]. In this method, the finite field arithmetic of Inv-/S-box in $GF(2^8)$ is transformed into equivalent one in the field of $GF(2^4)$ which is more suitable for hardware implementation. For the implementation of Inv-/MixColumns, we apply our proposed architecture [12] where both forward and inverse MixColumns can be summed up to efficient calculation of the products of input byte with bytes $\{01\}$, $\{02\}$, $\{04\}$ and $\{08\}$, which proves to have short critical path, small gate count and versatility. Inv-/ShiftRows is a simple permutation of bytes, and requires no logic gates. Because Inv-/S-box has a multiplexor of its own, we can use this multiplexor to select ShiftRows or Inverse ShiftRows as the input corresponding to encoding or decoding with no additional overhead to the function of Inv-/S-box. The AdRoundKey is a simple XOR operation. Moreover, two additional multiplexors (MUX1 and MUX2) are used in *Data Module*. MUX1 is used to bypass Inv-/MixColumns in the final round of encryption/decryption. MUX2 is used for loading the data blocks into the datapath in the initial round.

Due to the different complexity of the four operations in AES, a pipelined and parallel architecture is proposed to achieve a balanced critical path and a high clock frequency. The proposed architecture is three-stage pipelined with one register *Reg1* between output of AddRoundKey and input of Inv-/ShiftRows operation, one register *Reg2* embedded into the Inv-/S-Box, which makes the stage of the critical path in Inv-/S-Box roughly equals to that in Inv-/MixColumns, and the last register *Reg3* between output of Inv-/S-box and input of Inv-/MixColumns. Thus the pipeline of the critical path in *Data Module* is well balanced. It is also noted that both the Inv-/ShiftRows and the half of the Inv-/SubBytes operations are computed in the same stage, because the Inv-/ShiftRows operation does not require gates, and offers no significant slowing down to the overall clock rate.

The architecture works in a sub-pipelined iterative mode. The *Key Register* in *Key Generator Module* (Fig. 3) is initialized with the cipher key by I/O Interface. Then *Key Generator Module* begins to produce the round keys continuously for *Data Module*. Because *Data Module* has three-stage pipelines, three blocks of plain/cipher text can be processed along the data path simultaneously. 30 clock cycles are required to encrypt/decrypt three blocks of data for 128-bit cipher key (10 rounds), and 36 clock cycles are needed for 192-bit key (12 rounds) and 42 clock cycles are required for 256-bit key (14 rounds). In the last round, the Inv-/MixColumns is skipped by multiplexor MUX1 and *Key Generator Module* begins to load initial key from *Key Register* or *Final Key Register* into *Register1* (Fig. 3) for encryption or decryption, guaranteeing that the next initial AddRoundKey operation can be performed synchronously. *Key Generator Module* provides each corresponding round key for continuous three clock cycles since three blocks of data use the same round key in each loop.

3.2 Key Generator Module

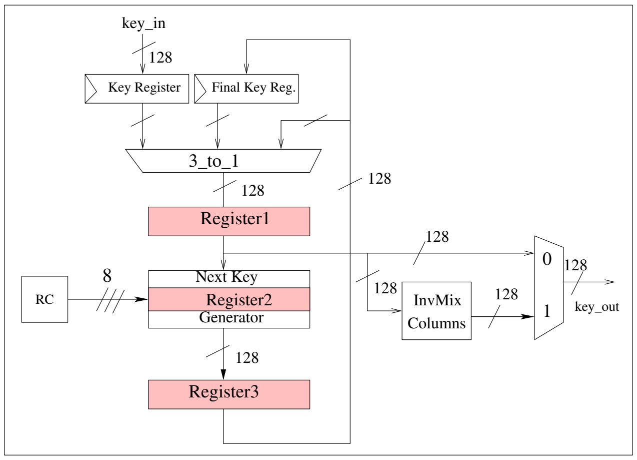

Figure 3. The logic diagram of *Key Generator Module*

In this subsection, the detailed architecture of 128-bit cipher key expansion is proposed. The key expansion is capable of presenting the keys in a forward or reverse order for encryption or decryption. The logic diagram of the proposed *Key Generator Module* is illustrated in Fig. 3. The cipher key is loaded initally from *I/O Interface Module* and stored in *Key Register*. This value is reloaded only when cipher key is changed. *Final Key Register* is used to store the last round of key as the initial key for key expansion in decryption.

In order to synchronously provide every round key for *Data Module*, *Key Generator Module* also works in a three-stage sub-pipelined iterative mode, where one of the three levels of pipeline registers is placed inside *Next Key Generator Module* (Fig. 4) to balance the critical path. Since three blocks of data in *Data Module* require the same round key in each loop, *Key Generator Module* generates the same round key for three times in a pipelined fashion. When *Key Generator Module* begins working, the initial key stored in *Key Register* (for encryption) or *Final Key Register* (for decryption) is loaded into *Register1* in the first three consecutive clock cycles to make sure that the three-stage pipeline starts with the same initial values. Then each round key is produced three times in three consecutive clock cycles to meet the pace of *Data Module*.

Note that, in the last iterative loop, the last round key is loaded into *Final Key Register* for the generation of decryption round keys. This value is reloaded only when the cipher key changes.

Furthermore, the Equivalent Inverse Cipher demands that each round key is transformed by Inverse MixColumns operation on decryption. Thus an additional InvMixColumns is required for *Key Generator Module* (Fig. 3). The InvMixColumns does not need many gates and has no significant effect on the overall complexity. It should be noted that the round keys for encryption, and the initial and last round key for decryption do not require the InvMixColumns operation. We use a multiplexor to select if the InvMixColumns is needed.

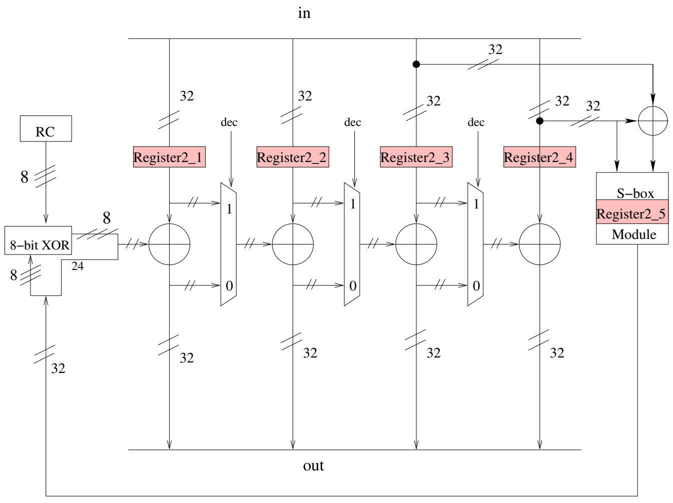

Figure 4. The logic diagram of *Next Key Generator Module*

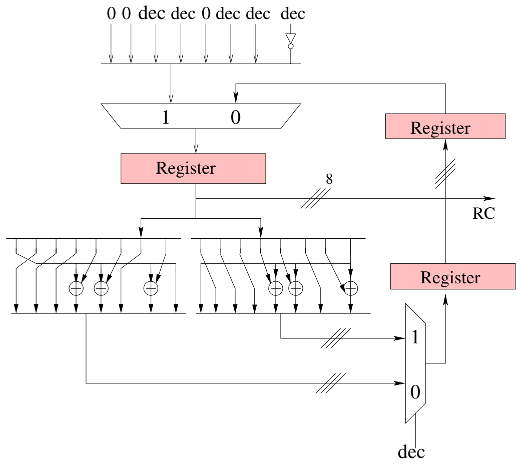

Figure 5. The logic diagram of Round Constant (RC)

The logic diagram of *Next Key Generator Module* is illustrated in Fig. 4. It is the main part to realize the key

Table 1. Performance of proposed AES implementation

Number of Slices	2052
Critical path(ns)	7.36
Freq.(MHz)	135.7
Latency (cycles)	31
Throughput(Mbits/s)	1570
Mbps/Slice	0.765

expansion algorithm. The "8-bit XOR" is 8-bit XOR operation of Round Constant (RC) with the leftmost eight bits of output from S-box (the remaining rightmost three bytes are unchanged). Both SubWord and RotWord operations in the key expansion algorithm are implemented in S-box Module. The registers from "Register2_1" to "Register2_5" consists of the second level of pipeline registers, i.e. *Register2* in Fig. 3.

The logic diagram of RC is shown in Fig. 5. The initial value of RC[1]={01h}, when dec=0; and RC[10]={36}, when dec=1. This module generates RC values sequentially on encryption from{01}, and the same values in a reverse order from {36} on decryption. *RC Module* also works in a three-stage pipelined iterative mode.

4 Performance evaluation and comparison

The design is simulated by Verilog HDL, synthesized with Xilinx Synthesis Technology (XST) and implemented on Xilinx FPGA Virtex II device. It works at a system clock of 135MHz, and the throughput reaches 1.57Gbits/s. The critical path of the Inv-/MixColumns is 7 gate equivalent, and each of two stages of S-box operation is also roughly 7 gate equivalent, so the proposed pipelined architecture of AES for both encryption and decryption is well balanced. Table 1 summarizes the performance of the proposed AES implementation.

It can be seen that our design can do both encryption and decryption with key agile scheduler, and the throughput is faster than [9, 10]. In addition, the performance of throughput/slice is also better than [9, 10]. Note that the AES implementation by N. Weaver et al. [11] uses the sub-pipelined structure with five stage pipelines for both encryption core and corresponding 128-bit subkey generation core. It seems comparable to our design with 1,750 Mbit/s throughput, 770 slices and 2.3 throughput/slice. However, it uses 10 4K-bit BlockRAMs to realize the S-box. If the 40k-bit BlockRAMs are converted to CLB slices for fair comparison considering that we do not use RAM in our circuit design, then the number of slices will be overly increased and therefore throughput/slice decreased in [11]. Furthermore, the design of [11] implements only encryption. If we only implement encryption, then the Equivalent Inverse Cipher along with other inverse operations can be removed from the architecture and the complexity of hardware can be greatly reduced.

A fully pipelined architecture is used by M. McLoone et al. [8] with 3,239 Mbit/s throughput, 7,576 slices and 0.43 throughput/slice. We can see that even though the throughput is faster than our design, it is paid by more hardware area cost. The throughput/slice performance in our design is much better than in [8].

5 Conclusion

A high performance three-stage sub-pipelined architecture AES is proposed in this paper. In the design, both encryption and decryption are implemented in the same module, and the round keys can be generated in forward or inverse order simultaneously. Compared with previous FPGA implementations, the proposed designs have a relatively low area and high throughput performance. The design can be used in applications such as digital video recorders and PCMCIA card etc., where high speed/area ratio is required. Furthermore, the design can also be extended to accommodate the reconfigurable architecture for variable key option with little modification.

Acknowledgment

This research work is supported by the funding of National Science and Engineering Research Council of Canada (NSERC) and the University of Lethbridge Research Fund (ULRF).

References

[1] J. Daemen and V. Rijimen, The Design of Rijndael, Springer-Verlag, 2002.

[2] NIST, "Federal Information Processing Standard 197, The Advanced Encryption Standard (AES)," http://csrc.nist.gov/publications/fips/fips197/fips197.pdf, 2001.

[3] W. Stallings, Cryptography and Network Security, Third Edition, Prentice Hall, 2003.

[4] J. Wolkerstorfer, E. Oswald and M. Lamberger, "An ASIC Implementation of the AES SBoxes," CT-RSA 2002, LNCS2271, pp. 67-78, 2002.

[5] A. Satoh and S. Morioka and K. Takano and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," Proc. Advances in Cryptology, ASIACRYPT, pp. 239–254, 2001.

Table 2. Performance comparison of similar FPGA implementations for AES

Different FPGA implementations	with key scheduler	encryption or decryption	Throughput (Mbits/s)	Number of slices	Throughput/slice
A. Elbirt et al. [9] Virtex, 2 stages	N	encryption	949.1	4871	0.19
P. Chodowiec et al. [10] Virtex, 8 stage	N	encryption	1,265	2000	0.63
Our design Virtex II, 3 stage	Y	both	1,570	2,052	0.765

[6] S. Mangard, M. Aigner and S. Dominikus, "A Highly Regular and Scalable AES Hardware Architecture," IEEE Transaction On Computers, VOL. 52, NO. 4, pp. 483-491, 2003.

[7] N.S. Kim, T. Mudge and R. Brown, "A 2.3Gb/s Fully Integrated and Synthesizable AES Rijndael Core," Custom Integrated Circuits Conference (CICC), pp. 193-196, San Jose, CA, September 2003.

[8] M. McLoone and J.V McCanny, "High Performance Single-Chip FPGA Rijndael Algorithm Implementation," CHES 2001, LNCS 2162, pp. 65-76, 2001.

[9] A. Elbirt, W. Yip, B. Chetwynd and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," IEEE Trans. of VLSI Systems, NO. 9, VOL. 4, pp. 545-557, August 2001.

[10] P. Cholowiec, P. Khuon and K. Gaj, "Fast implementations of secret key block ciphers using mixed inner- and outer-round pipelining," Proceedings of the International Symposium on Field Programmable Gate Arrays, February 2001.

[11] N. Weaver and J. Wawrzynek, "High performance, compact AES implementations in Xilinx FPGA," http://www.cs.berkeley.edu/∼nweaver/sfra/rijndael.pdf.

[12] H. Li, Z. Friggstad, "An Efficient Architecture for the AES Mix Columns Operation," IEEE International Symposium on Circuits and Systems, Kobe, Japan, May 2005.