www.ietdl.org

Published in IET Computers & Digital Techniques
Received on 19th May 2014
Revised on 4th September 2014
Accepted on 31st October 2014
doi: 10.1049/iet-cdt.2014.0101

ISSN 1751-8601

High throughput and secure advanced encryption standard on field programmable gate array with fine pipelining and enhanced key expansion

Qiang Liu, Zhenyu Xu, Ye Yuan

School of Electronic Information Engineering Tianjin University, Tianjin 30072, People's Republic of China
E-mail: qiangliu@tju.edu.cn

Abstract: Aiming at protection of high speed data, field programmable gate array (FPGA)-based advanced encryption standard (AES) design is proposed here. Deep investigation into the logical operations of AES with regard to FPGA architectures leads to two efficient pipelining structures for the AES hardware implementation. The two design options allow users to make a trade-off among speed, resource usage and power consumption. In addition, a new key expansion scheme is proposed to address the potential issues of existing key expansion scheme used in AES. The proposed key expansion scheme with additional non-linear operations increases the complexity of cracking keys by up to $2^{(N-1)}$ times for N-round AES. The proposed design is evaluated on various FPGA devices and is compared with several existing AES implementations. In terms of both throughput and throughput per slice, the proposed design can overcome most existing designs and achieves a throughput of 75.9 Gbps on a latest FPGA device. Two parallel implementations of the proposed design can meet the real-time encryption/decryption demand for 100 Gbps data rate. Furthermore, the proposed AES design is implemented on the Zynq xc7z020 FPGA platform, demonstrating its application to image encryption.

1 Introduction

Today's information technologies create large amounts of data in the scale of exabytes every day. Secure transfer and storage of these data become a major concern for businesses. Advanced technologies, such as high-speed Ethernet and storage area network (SAN), provide solutions for fast data transmission and scalable data management. Often, the data are not protected when they are transferred, and thus are under the risk of loss and deliberate compromise [1]. Encryption is a powerful and widely used protection technology. However, as the Ethernet evolves to 40/100 Gbps [2, 3] and the SAN switches and host bus adapters support the bandwidth up to 128 Gbps [1], real-time encryption has become a challenge.

Real-time encryption encrypts or decrypts the data right before the data are sent or loaded without any user intervention. Advanced encryption standard (AES) is a widely used encryption algorithm [4]. It is a symmetric block cipher and is capable of using cryptographic keys of 128, 192 and 256 bits to encrypt and decrypt data. Different lengths of the key lead to different number of rounds of operations that execute in sequential manner for encrypting/decrypting a plaintext. The iterative operations in the AES algorithm have the inherent concurrency that enables pipelined hardware implementation for high throughput.

A substantial progress in the development of AES hardware designs has been made based on field programmable gate arrays (FPGAs) and ASICs [5]. The designs target maximising speed, minimising area or achieving a trade-off between speed and area, by means of techniques such as loop unrolling, pipelining and datapath wordlength customisation. This work aims to be the further progress along the direction of pursuing fast and secure AES on FPGAs.

The motivations behind this work are as follows: First, the new big data applications require higher throughput for real-time encryption to ensure secure transfer and storage. Second, owing to the fact that the increased NRE cost makes low- to mid-volume ASIC production unaffordable, FPGAs are more attractive for AES implementations. Third, pipelining is the widely used technique to increase the throughput of AES hardware implementations. Inserting registers into the combinational logic within and between cipher rounds enables parallel processing of multiple plaintexts. It is desirable to have guidance on how and where to insert the registers for an efficient AES design on different FPGA architectures. Fourth, in addition to the main encrypting operations in AES, key expansion is an equivalent important process, which generates the key for each round of encryption/decryption and affects the security and speed performance of AES. The structural design of key expansion has not been paid much attention yet.

In this paper, we first investigate the elementary operations in AES in detail, to analyse the logic depth of each operation. This is because a great logic depth will increase the signal delay, and thus reduce the maximum clock frequency of the AES hardware implementation. Then, we relate the logic

depth of the operations to FPGA logical architectures in mathematical formulae. These formulae reveal the critical path of the AES hardware design, and provide guidance on pipelining designs. With the aid of the analytical formulae, we propose two pipelining solutions, by dividing or combining the combinational logic of the elementary operations in single AES cipher round into stages, with the goal of achieving balanced and reduced logic depth. As in [6], the first solution focusing on the elementary operation level divides the single AES cipher round into two pipelining stages. As a second solution, the logic in the elementary operations are further divided to make the logic depth of the pipeline stages to be one, which was difficult to accomplish, as mentioned in [5, 7]. With this elaboration, a deep pipelining structure is achieved, further improving AES throughput. The two solutions achieve different design efficiency on different FPGA logical architectures.

Meanwhile, the key expansion module of AES is also studied to find out the structural issues that degrade the security and speed performance of AES. Consequently, a new key expansion scheme is proposed. The present scheme involves more non-linear operations for obfuscation and well matches with the pipeline structure of the encrypting/decrypting datapath of AES. With this new key expansion scheme, the resistance to key cracking and the speed of AES are improved simultaneously.

Therefore the contributions of this paper are as follows:

- Deep investigation into the logical implementation of AES with regard to FPGA architectures. The logic depth of each elementary operation in AES is related to the logical architecture of FPGAs, resulting in two efficient pipelining structures for the AES hardware implementation. In addition, the clock frequency margin is explored to further increase the throughput of the AES implementation (Section 3).
- A new and secure key expansion scheme. The proposed key expansion scheme incorporates additional non-linear operations and increases the complexity of cracking keys by up to $2^{(N-1)}$ times for N-round AES. Also, the new scheme is properly pipelined to match the encrypting/decrypting datapath of AES, leading to improved performance (Section 4).
- Evaluation of the design on various FPGA devices, achieving a throughput of 75.9 Gbps for AES-128. Two parallel duplications of the present pipelined datapath can easily achieve the throughput over 100 Gbps. Moreover, the proposed design improves throughput per slice by up to 8.1 times compared to several existing AES implementations. We also implement the design on the Zynq xc7z020 platform to demonstrate image encryption application (Section 5).

2 Background

Fig. 1 shows the process flow of AES-128, which takes 10 rounds of operations per 128-bit data [4]. In each cipher round, there are four elementary operations, including SubBytes, ShiftRows, MixColumns and AddRoundKeys, which are performed on a two-dimensional array of bytes called state matrix. SubBytes transforms individual bytes of the state matrix into the values stored in a non-linear byte substitution table (Sbox). ShiftRows cyclically shifts the last three rows of the state matrix by different offsets, respectively. MixColumns mixes all the 4 bytes of a column of the state matrix to form a new column. AddRoundKeys is simply the exclusive-OR (XOR) between

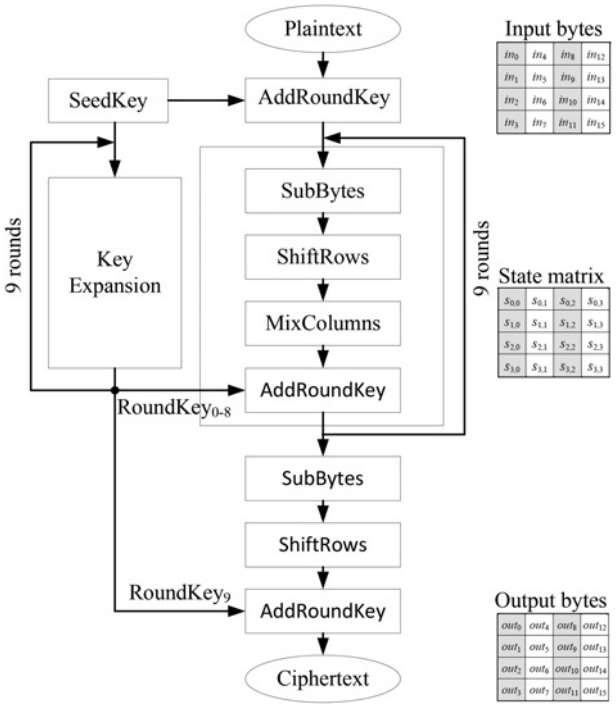

Fig. 1 *Original 128-bit AES encryption flow*

the state matrix and RoundKey, which is 128 bits. After 10 rounds of iterations of the four operations, a plaintext is converted to a ciphertext, which can be decrypted through an inverse flow of Fig. 1.

In Fig. 1, there is also a module parallel to the main encrypting operations. This module called Key Expansion is used to generate from the SeedKey a series of RoundKeys, which are then applied to the state matrix in the AddRoundKeys operation. Although this paper presents an AES-128 structure, the design method is valid for all variants of AES, and thus the generic abbreviation AES is used in the following sections.

According to the AES flow described above, various kinds of implementation of AES on FPGAs have been developed. Table 1 lists several recent FPGA-based AES implementations. There are two main structures: non-pipelined and pipelined.

The non-pipelined designs [8–10] focused on the compact design and are more efficient when AES works in the feedback mode, where the new plaintext needs to be first added with the previously encrypted ciphertext. As shown in Table 1, the speed of the non-pipelined designs is slower than the pipelined designs.

The pipelined designs aim to increase the throughput of AES implementations, and are suitable to AES working in the counter mode, which does not need the previous output to compute the new one. As a result, the iterative operations in AES can be unrolled and registers are inserted to realise pipelining process.

The pipelined designs shown in Table 1 can be distinguished into two design options. The first design option is the number of pipelining stages in each cipher round. In [11–15], various numbers of stages were explored, leading to different throughput. The second design option is the implementation of Sbox in SubBytes. The widely used solutions are: (i) to store the values of Sbox in a lookup table (LUT) and (ii) to compute the values in the composite-field on-the-fly by using multiplicative inverse and affine transformations. The

doi: 10.1049/iet-cdt.2014.0101

Table 1 Existing FPGA-based AES implementations

Design	Platform	Structure	Throughput, Gbps
[8]	Virtex7	non-pipelined	5.3
[9]	XC5VLX50	non-pipelined with LUT-based SubBytes	4.34
[10]	XC5VLX50	non-pipelined with LUT-based SubBytes	3.1
[11]	XC2VP20-7	pipelined with seven stages with composite field SubBytes	21.64
[12]	XC3S2000-5	pipelined with three stages with composite field SubBytes	25.107
[13]	XC6VLX240T	pipelined with six stages with composite field SubBytes	44.047
[14]	XC2V2000-5	pipelined with composite field SubBytes	17.8
[15]	XC5VLX110T	pipelined with three stages and composite field SubBytes	25.89
[16]	XC4VLX40	pipelined AES-GCM with composite field SubBytes	20.61
[7]	XC5VLX85	pipelined AES-GCM with LUT-based SubBytes	41.47
[3]	XC5VSX220	four parallel pipelined AES-GCM with LUT-based SubBytes	119.30

The specific device used in [8] is unknown.

former requires a large storage space and the latter increases computation complexity [7]. The composite-field computation-based Sbox achieved smaller area compared to the LUT-based implementation in ASIC implementations [17, 18]. For FPGAs with four-input LUTs, the composite-field implementation can also reduce area. However, for recent FPGAs with six-input LUTs, the situation is changed, and the LUT-based implementation of Sbox has smaller area and reduced delay. We will see the related experimental results in the following section. Therefore we choose the first solution to implement Sbox. It was mentioned in [12, 14, 15] that the LUT in the first solution limits further partitioning of pipelining stages in each round of AES. This limitation is resolved in the present paper by dividing Sbox into sub-boxes.

Recently, AES operating in the Galois/Counter mode (AES-GCM) was also implemented on FPGAs [7, 16]. The AES-GCM targets high speed authenticated cipher, and contains two main components: an AES engine and a finite-field multiplier over GF(2^{128}). Therefore efficient multiplier designs were also explored in [7, 16]. A pipelined AES-GCM core was duplicated four times on an FPGA to achieve throughput 119.30 Gbps [3]. The present paper only targets the AES design, and the design can be applied to the AES-GCM implementations.

Although research on the FPGA-based AES implementations has been carried out extensively, this paper searches for solutions to further increase throughput as well as provide further resistance against attacks. The differences from the previous works are the following: First, the present paper designs pipelining structures based on the investigation of the combined effects of the combinational logic depth in the AES operations and the FPGA logic structures, leading to improved throughput and resource utilisation. Second, as reported in [3, 7] that the critical path locates in the key expansion module, we explore the structure design of the key expansion module to match the pipeline stages, leading to increased throughput. Third, we also discuss the complexity of the key expansion module and propose a new scheme to increase the complexity in cracking keys.

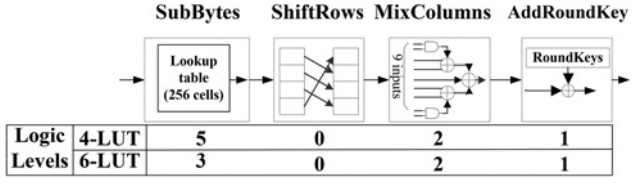

Fig. 2 *Four elementary operations in each round of AES and the logic depth of each operation*

4-LUT: four-input LUT
6-LUT: six-input LUT

3 Design approach for high-throughput AES

As the existing pipelining designs, the proposed structure first uses loop unrolling to expand the multi-round operations in AES and adds registers between rounds, forming a one-pass datapath for plaintexts. In the following, we present the approach to pipelining design of the single AES round. To start with, the combinational logic depth of the elementary operations in AES is analysed. Then two pipelining solutions are provided with different pipelining stage design for each cipher round. Finally, overclocking is explored to further increase the throughput.

3.1 Logic depth analysis

In FPGA-based designs, the logic depth of a combinational logic path between two registers is the number of cascaded LUTs and MUXes. The four elementary operations of AES are shown in Fig. 2, where the operation details are illustrated.

Among the four elementary operations, the SubBytes dominate the performance of AES [18]. As mentioned in Section 2, the existing AES implementations used two different solutions to implement Sbox in SubBytes. We experimentally compared the two solutions. The approach in [17, 18] was followed to implement the composite-field computation-based Sbox on FPGAs. The implementation involves mapping and inverse mapping between the GF(2^8) field and the GF(2^4) field, a number of GF(2^4) multiplications and additions, and affine transformation. In Table 2, the logic resource usage and critical path delay of the composite-field implementation is then compared to the LUT-based Sbox implementation. The results are obtained

Table 2 Comparison of Sbox implementations based on composite-field computation and LUT on FPGAs

Implementation	XC4VLX60 (4-input LUTs)		XC6VLX240T (6-input LUTs)	
	#LUTs	Critical path delay, ns	#LUTs	Critical path delay, ns
composite-field	72	11.14	48	9.14
LUT	128	7.97	32	6.22

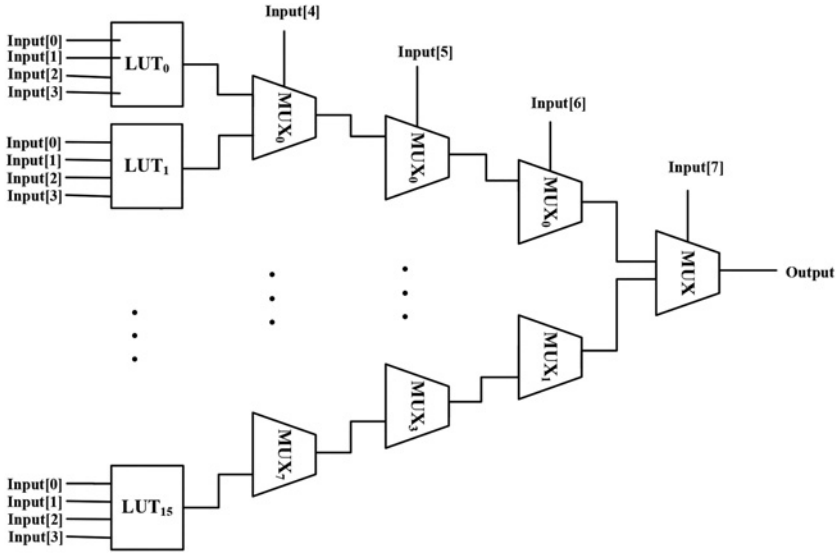

Fig. 3 *SubBytes operation implemented on FPGAs with 4-input LUTs, leading to logic depth 5*

after synthesis, routing and placement. The comparison shows that in the FPGA with six-input LUTs, the LUT-based implementation has less logic resource usage and shorter critical path delay, compared to the composite-field computation-based solution.

Therefore in this work, the Sbox in SubBytes is implemented using a LUT with an 8-bit input to select one output out of 256 words. The logic depth of SubBytes can be determined as follows [6].

$$L_{\text{sb}} = 8 - K + 1 \tag{1}$$

where K is the number of inputs of a LUT. The MixColumn operation involves several XOR and AND operations with nine inputs, as shown in Fig. 2. Therefore its logic depth is

$$L_{\text{mc}} = \lceil (4/K) \rceil + \lceil (3/K) \rceil \tag{2}$$

The operations in ShiftRows are the cross assignments without any logical operation. AddRoundKeys performs bit-wise XOR and thus the logic depth is one.

In the recent FPGAs, each LUT may have 4/5/6 inputs, allowing complex logic functions to implement on one basic logic element and thus reducing the combinational logic depth on FPGAs. For example, in FPGAs where each LUT has four inputs, the operation in SubBytes needs 16 parallel four LUTs and four levels of cascaded MUXes, as shown in Fig. 3, resulting in logic depth equal to 5. When mapped onto FPGAs with six-input LUTs, the SubBytes has logic depth 3 with four 6-LUTs and two levels of cascaded MUXes, as shown in Fig. 4. Similarly, the other elementary operations have the corresponding logic depth, as shown in Fig. 2. Since

$$L_{\text{sb}} \geq L_{\text{mc}} \geq 2, \quad \text{for } 4 \leq K \leq 6 \tag{3}$$

the greatest logic depth is in the SubBytes operation.

When pipelining the operations in single AES round, registers are inserted to divide the operations into stages. The design goal of pipelining is to achieve the balanced and minimum logic depth, that is, logic delay, in different pipeline stages. In addition, design pipelining allows designers to make trade-off between area and throughput for the AES design. More pipelining stages may lead to shorter delay and higher throughput, but require more registers and logic resources. In the following, we propose two pipelining architectures, which are designed based on the logic depth analysis described above and achieve high throughput and low area.

3.2 Two-stage pipelining design

The above analysis shows that the greatest logic depth of the AES operations is in SubBytes. The combined logic depth of the other three operations is not greater than that of SubBytes. Therefore at the elementary operation level, a two-stage pipelining can be designed, as shown in Fig. 5*a*. SubBytes is in one stage, and ShiftRows, MixColumn and AddRoundKeys are combined in another stage. Inserting more registers among the last three elementary operations would not increase the clock frequency, but increase resource usage.

For 4-input LUTs, the two stages have logic depth 5 and 3, respectively. For 6-input LUTs, the logic depth of both stages is 3. The pipelining structure is then duplicated 10 times and connected one after another to form the one-pass datapath. With this structure, on every clock cycle, an input of 128-bit data is used that gives an encrypted/decrypted data

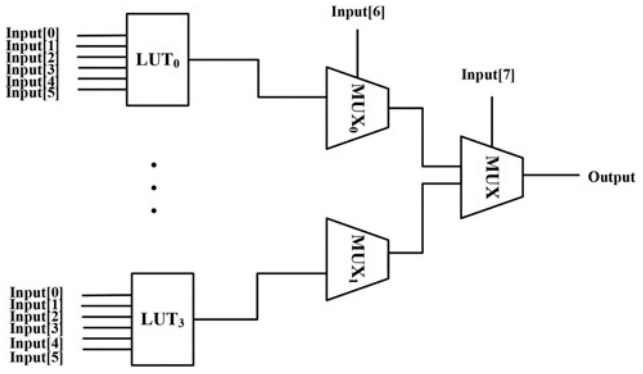

Fig. 4 *SubBytes operation implemented on FPGAs with 6-input LUTs, leading to logic depth 3*

doi: 10.1049/iet-cdt.2014.0101

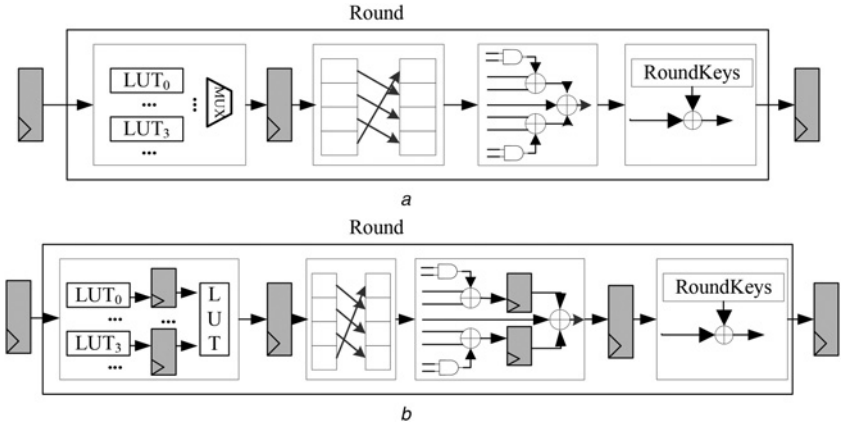

Fig. 5 *Pipelining designs of AES operations on FPGAs with 6-input LUTs*

a Two-stage pipeline for each AES round
b Five-stage pipeline for each AES round

as output. Each input data is processed over 22 cycles for encryption and 41 cycles for decryption through the datapath.

3.3 Deep pipelining design

To further increase the throughput, we need to divide the combinational logic inside the elementary operations. Based on the above analysis, the SubBytes should be tackled first. One way to reduce the logic depth of SubBytes is to partition the Sbox into small LUTs, called sub-Sboxes, and store the middle results into registers. Since the SubBytes operation has different logic depth for four and six LUTs, in the following, we will discuss the designs separately.

For FPGAs with 6-input LUTs, the Sbox is partitioned into four sub-Sboxes, and each contains 64 words. Four registers can be inserted between the LUTs and MUXes in Fig. 4 to break the logic path. Then the four registered results are selected by the two most significant bits of the input. The structure is shown in Fig. 6, where the SubBytes operation is divided into two stages and each stage has logic depth 1. Note that different from Fig. 4, the two-level MUXes are implemented in one 6-input LUT now, leading to reduced logic depth. This change is due to the internal architecture of the logic elements in FPGAs. Within each logic element, the outputs of LUTs are connected to the internal MUXes, whereas the outputs of registers in a logic element are connected to the inputs of LUTs in the next logic element. Therefore in this case inserting registers does not just break combinational logic, but also enables reduction of the original logic depth. In addition, the logic depth of

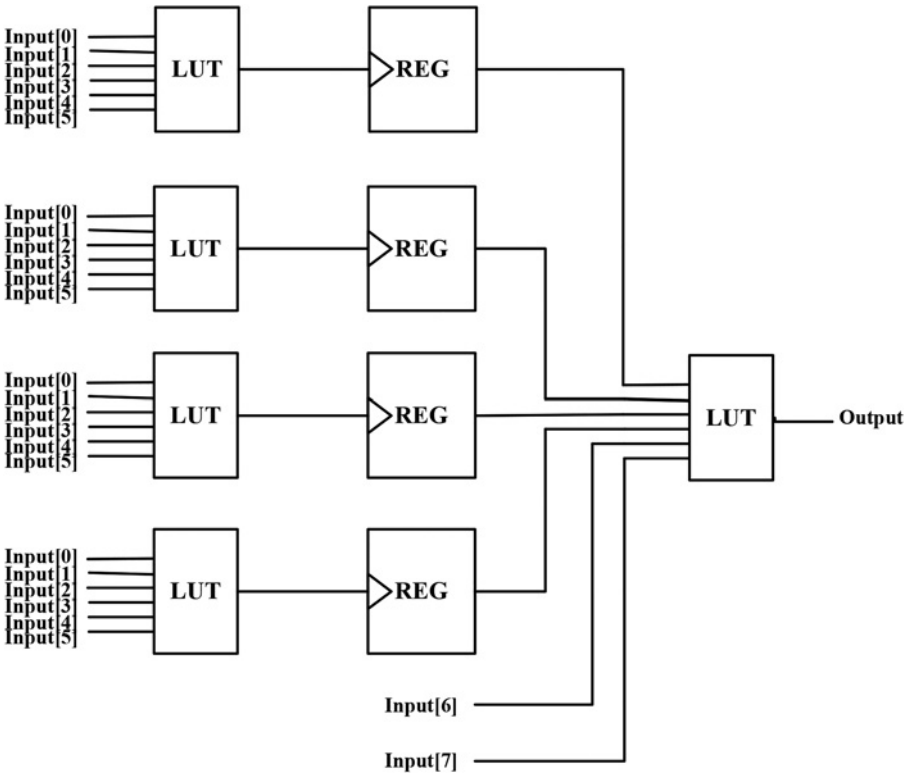

Fig. 6 *SubBytes with four sub-Sboxes is divided into two stages on FPGAs with 6-input LUTs, leading to logic depth 1*

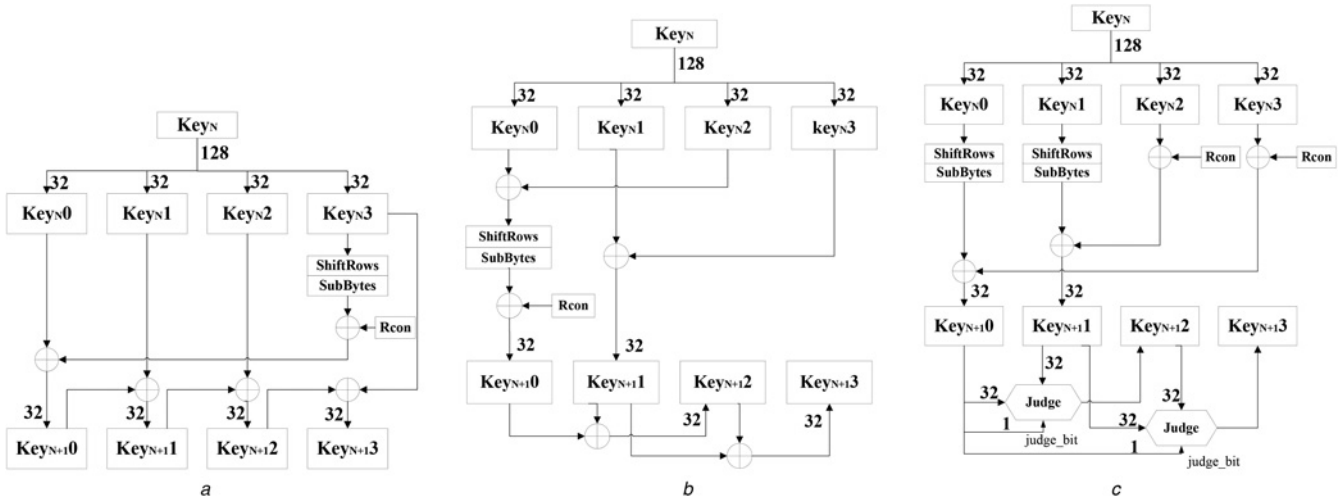

Fig. 7 *Key expansion schemes*

a Original key expansion scheme in AES
b Modified key expansion scheme [20]
c Our proposed key expansion scheme

MixColumn is 2 with 6-input LUTs. It is also necessary to break the logic into two stages. As a result, as shown in Fig. 5*b*, the four elementary operations in AES can be organised in five pipelining stages on FPGAs with 6-input LUTs, with each stage having the minimum logic depth 1. With this pipelining structure, each input data is processed over 50 cycles for encryption and 101 cycles for decryption.

Similarly, for 4-input LUTs, the Sbox should be partitioned into 16 sub-Sboxes. To obtain a single logic depth design, 30 registers need to be inserted between the LUT level and the MUX level and between every two MUX levels in Fig. 3. Together with the further logic division of other three elementary operations, an eight-stage pipelining is constructed for the AES on FPGAs with 4-input LUTs. Owing to the long pipeline, the latency of each data from input to output becomes 79 cycles for encryption and 158 cycles for decryption.

3.4 Exploring conservative timing margin

In the above, we exploit pipelining to reduce path delay and increase the maximum clock frequency for high throughput. Given the fact that the minimum logic depth is achieved, it is not possible to further increase the clock frequency relying on pipelining. To deal with this issue, this paper deploys overclocking. Usually, the maximum clock frequency of a circuit reported by electronic design automation (EDA) tools, which analyse the circuit at the worst case, tends to be conservative [19]. In a practical situation, there could be a margin between the worst-case frequency and the actual operating frequency under which the circuit works correctly. This margin is explored in this paper to further increase the throughput.

Based on the timing results analysed by EDA tools, we gradually increase the operating clock frequency of the AES implementation until errors are observed at the output. The achievable maximum clock frequency may vary on different FPGAs and under different working environments. In our experiment, the clock frequency is further increased 67% compared to the reported value. As a result, the throughput of AES is increased by the same order. This bonus comes at no cost.

Up to now, based on the logic depth analysis, we propose two pipelining solutions: one targeting at the elementary operation level and another targeting at the fine logic level. To realise high-throughput AES, the key expansion module shown in Fig. 1 should also be pipelined to match the structure of the encrypting datapath. Some of the existing AES implementations did not discuss the design of this module, and some reported that the critical path locates in the module but did not provide any improvement solutions. This paper proposes a new key expansion scheme, which increases both performance and security.

4 New key expansion scheme

In the original key expansion scheme shown in Fig. 7*a*, if an attacker has one RoundKey by some means such as power attack, he actually is able to obtain all RoundKeys and the SeedKey, due to the invertibility of the XOR operation. For instance, if the key of round $N+1$ is known, the cracker can derive the subkey Key_N3 from $\text{Key}_{N+1}2$ and $\text{Key}_{N+1}3$, Key_N2 from $\text{Key}_{N+1}2$ and $\text{Key}_{N+1}1$, Key_N1 from $\text{Key}_{N+1}1$ and $\text{Key}_{N+1}0$ and Key_N0 from $\text{Key}_{N+1}0$ and Key_N3. Therefore the whole set of RoundKeys is known and following in the same manner, the SeedKey can be deduced.

The work in [20], which focuses on key expansion scheme design, develops an enhanced key expansion scheme, as shown in Fig. 7*b*. By changing the method of key expansion between rounds and within a round, this scheme makes the crackers to have up to 2^{32} times in order to obtain the key of round N from that of round $N+1$. However, in this scheme the first word and the last word of the expanded RoundKey are the same, as shown here

$$\text{Key}_{N+1}2 = \text{Key}_{N+1}0 \oplus \text{Key}_{N+1}1 \tag{4}$$

$$\begin{aligned}\text{Key}_{N+1}3 &= \text{Key}_{N+1}1 \oplus \text{Key}_{N+1}2 \\ &= \text{Key}_{N+1}1 \oplus \text{Key}_{N+1}0 \oplus \text{Key}_{N+1}1 \\ &= \text{Key}_{N+1}0\end{aligned} \tag{5}$$

doi: 10.1049/iet-cdt.2014.0101

This would make it easier for the crackers to obtain a RoundKey in order to carry out the key deduction.

4.1 New scheme

To increase the difficulty to crack RoundKeys, we come up with a new key expansion scheme based on the method [20]. The present scheme is shown in Fig. 7*c*. In this scheme, $\text{Key}_{N+1}0$ is produced by Key_N0 and Key_N3, and $\text{Key}_{N+1}1$ is produced by Key_N1 and Key_N2. We use two non-linear transformations, including ShiftRows, SubBytes and XOR with Rcon (a constant), to generate $\text{Key}_{N+1}0$ and $\text{Key}_{N+1}1$. The added non-linearity reduces the correlation between RoundKeys, and makes it more difficult for attackers to crack the keys just by observing and analysing the regular pattern in each round.

After $\text{Key}_{N+1}0$ and $\text{Key}_{N+1}1$ are obtained, two judgement modules are added in the proposed scheme to calculate $\text{Key}_{N+1}2$ and $\text{Key}_{N+1}3$. This is inspired by [21], where RoundKeys are formed based on the even and odd positions of generated words. The operations in the judgement modules are

$$\begin{aligned}&\text{If(judge\_bit} == 1)\\&\quad \text{out} = \text{operand\_1} \oplus (\text{operand\_2} \ll 16)\\&\text{else}\\&\quad \text{out} = \text{operand\_1} \oplus (\text{operand\_2} \ll 8)\end{aligned}$$

where judge_bit is the judgement input, operand_1/2 are the two inputs, out is the output of the judgement module and $\ll$ represents the left rotation.

We can choose any two bits of $\text{Key}_{N+1}0$ as the judgement inputs of the two judgement modules. The rotation operation is introduced to prevent $\text{Key}_{N+1}3$ from being equal to $\text{Key}_{N+1}0$, as in [20]. These two extra judgement modules increase the difficulties for crackers to recover the original keys, as discussed later in this section.

To match the pipeline design of the elementary operations, the key expansion module also should be pipelined accordingly. When using a two-stage pipeline, in the proposed key expansion scheme in Fig. 7*c*, two sets of ShiftRows, SubBytes and XORs between Rcon and $\text{key}_{N2}/\text{key}_{N3}$ are placed in one pipelining stage, and the remaining XORs and two judgement modules are placed in the second stage. According to the analysis in Section 3, the first stage has the logic level $L_{\text{sb}} = 8 - K + 1$ and the second stage has the logic level $\lceil (4/K) \rceil$. This pipeline structure matches the two-stage structure in the encryption datapath with the greatest logic depth in SubBytes. In contrast, the pipeline of the key expansion scheme [20] is hardly matched to the proposed two-stage pipelining structure, because the XOR operation before SubBytes in Fig. 7*b* adds one more logic level to the first pipeline stage. The result section will show the performance degradation caused by this mismatch.

When the encrypting/decrypting datapath is in a deeper pipeline, the SubBytes operation in the key expansion is also divided accordingly and inserting more registers among the remaining operations to match the number of pipelining stages, ensuring that the RoundKeys are generated timely when they are needed in the encrypting/decrypting process.

4.2 Security analysis

Now consider how to derive the RoundKeys of the present key expansion scheme if the attackers know the key of round $N+1$. With just Key_{N+1} the attackers cannot directly derive any subkeys of Key_N because of the modified expansion method illustrated in Fig. 7*c*. They have to speculate about some subkeys in order to derive Key_N.

(a) *Case 1:* If they first speculate on Key_N0, then they can obtain Key_N3 from Key_N0 and $\text{Key}_{N+1}0$. Afterwards, they have to guess Key_N1 or Key_N2, because Key_N3 is calculated from Key_N1 and Key_N2, neither of which is known. Therefore, in a worst case they need to guess 2^{64} times in total to derive Key_N, given each subkey 32 bits.
(b) *Case 2:* However, insightful attackers can first guess Key_N1, then they can calculate Key_N2 from Key_N1 and $\text{Key}_{N+1}1$. Afterwards, obtaining either of Key_N0 and Key_N3 makes another known. To determine either of Key_N0 and Key_N3, they should guess the judgement bit of Key_N0 to determine what formation of the Key_N1 or Key_N2 is used in the calculation. As a result, the complexity in deriving Key_N from Key_{N+1} is 2^{33}. To finally obtain the SeedKey, the attackers need to guess up to M times

$$M = 2^{64} \times 2^{33(N-1)}, \quad 1 \le N \le 10 \tag{6}$$

where 2^{64} is the worst-case number of speculations needed to derive the SeedKey from the first RoundKey, because there is no relation between four words in the SeedKey, and only two words of the first RoundKey are calculated from the SeedKey. Therefore the present scheme is M times more secure than the original scheme, and is $2^{(N-1)}$ times more secure than the

Table 3 Comparison of original non-pipelined, two-stage pipelining and deep-pipelining AES designs

Design	Platform	LUT size	#Slices	Fmax, MHz	Throughput, Gbps	Mbps/Slice	Power, W	Gbps/W
original AES	XC7VX690T	6	486	322.58	3.44 (1×)	7.08 (1×)	1.04	3.30 (1×)
two-stage	XC7VX690T	6	3436	516.80	66.10 (19.22×)	19.20 (2.71×)	3.59	18.42 (5.58×)
five-stage	XC7VX690T	6	4339	593.12	75.92 (22.07×)	17.50 (2.47×)	4.64	16.36 (4.96×)
original AES	XC6VLX240T	6	335	323.73	3.45 (1×)	10.31 (1×)	4.83	0.72 (1×)
two-stage	XC6VLX240T	6	3121	501.00	64.10 (18.58×)	20.54 (1.99×)	8.08	7.93 (11.01×)
five-stage	XC6VLX240T	6	3900	573.39	73.39 (21.27×)	18.81 (1.82×)	9.52	7.71 (10.71×)
original AES	XC5VSX240T	6	340	305.90	3.26 (1×)	9.60 (1×)	5.22	0.63 (1×)
two-stage	XC5VSX240T	6	3579	347.58	44.49 (13.65×)	12.43 (1.29×)	7.56	5.89 (9.35×)
five-stage	XC5VSX240T	6	4444	439.17	56.21 (17.24×)	12.65 (1.32×)	9.96	5.64 (8.95×)
original AES	XC4VLX60	4	1975	192.68	2.06 (1×)	1.04 (1×)	2.15	0.96 (1×)
two-stage	XC4VLX60	4	19 727	285.71	36.57 (17.75×)	1.85 (1.78×)	9.26	3.95 (4.11×)
eight-stage	XC4VLX160	4	38 511	454.55	58.18 (28.24×)	1.51 (1.45×)	26.3	2.21 (2.30×)

Table 4 Comparison of the proposed deep-pipelining designs and existing FPGA-based AES implementations

Design	Platform	#Pipeline stages	Bitwidth, bits	#Slices	Fmax, MHz	Throughput, Gbps	Mbps/Slice
[8]	Virtex7	Non-pipelined	128	2444	456.00	5.30	2.17
our design	XC7VX690T	5	128	4339	593.12	75.92	17.50
[13]	XC6VLX240T	6	128	5927	319.29	40.87	6.90
our design	XC6VLX240T	5	128	3900	573.39	73.39	18.81
[3]	XC5VSX240T	2	128	14 799	233.00	119.30	8.06
our design	XC5VSX240T	5	128	4444	439.17	56.21	12.65
[15]	XC5VLX110T	3	128	8896	202.26	25.89	2.91
our design	XC5VLX110T	5	128	4445	403.39	51.63	11.62
[7][1]	XC5VLX85	2	128	4628	324.00	41.47	8.96
our design	XC5VLX85	5	128	4447	420.35	53.80	12.10
[22]	XC4VLX60	4	128	10 756	312.50	40.00	3.72
[16]	XC4VLX40	2	128	16 378	161.00	20.61	1.26
[7][2]	XC4VLX60	3	128	7712	285.00	36.48	4.73
our design	XC4VLX160	8	128	38 511	454.55	58.18	1.51

scheme in [20], where $M = 2^{64} \times 2^{32(N-1)}$.

The cost of increasing security is the amount of extra resources needed by the additional non-linear operations and the two judgement modules. The overhead will be evaluated in the following section.

5 Experimental results

To evaluate the proposed AES designs, they are synthesised, placed and routed using Xilinx ISE 14.2 on different FPGA devices. To start with, the two-stage pipelining design and the deep pipelining design are compared thoroughly, in terms of throughput, power consumption, throughput per area and throughput per power in Table 3. Then the deep-pipelined design is compared with several existing AES implementations, as shown in Table 4. Third, the new key expansion scheme is evaluated from the performance and area overhead aspects. Finally, an image encryption application is implemented to demonstrate the real-time processing capability of the proposed AES design and the benefit of exploring the conservative timing margin on throughput improvement.

In the following results, the maximum clock frequency (Fmax) is reported by a Xilinx Timing Analyser. Fmax is related to the path delay, as discussed in Section 3. The throughput demonstrates the impacts of pipelining and clock frequency. In addition, the number of slices (#Slices) used in the designs is shown as the measurement of implementation area, including used registers and LUTs. To better understand the design efficiency, throughput per resource usage (Mbps/slice) and throughput per power consumption (Gbps/W) are also reported. The Fmax and #Slices are obtained after synthesis and place&route, and the power consumption is reported by an Xpower Analyser.

5.1 Comparison of pipelining architectures

In Table 3, different pipeline architectures of AES are compared, including the original one without pipelining, two-stage pipelining and deep pipelining, on various FPGA devices. Looking through the table, we could make the following observations. First, in terms of area and power consumption, the original AES design without pipelining shows its strength. The pipelined designs consume more logic resources and power, owing to duplicated hardware pieces for the unrolled round operations and the large amount of inserted registers. Second, in terms of clock frequency and throughput, the deep-pipelined architecture, that is, the five-stage and eight-stage designs, as expected, achieves the best result. Compared to the original design, the throughput is improved by up to 28 times. This is because all timing paths are reduced to logic depth one. Third, in terms of throughput per slice and throughput per WATT, the two-stage pipelining architecture achieves the best design efficiency. The area efficiency and power efficiency are increased by up to 2.7 and 11 times, respectively, when compared with the original design. We can see that different pipelining architecture designs of AES are characterised by different area, power consumption and throughput. This provides designers great opportunity to explore AES design and make decisions on design trade-off. Therefore designers should select the right architecture for their AES implementation according to specific design goals.

In addition, the pipelined designs are more efficient on the FPGAs with 6-input LUTs, compared with 4-input LUTs. Implemented onto 4-input LUTs, the deep-pipelined design with eight stages is too large to fit into the same FPGA as the original and two-stage pipelined designs. Moreover, as shown in the last column of Table 3, the pipelined designs are effective in improving the power efficiency of the AES implementation, up to 11 times.

5.2 Proposed deep-pipelining design against existing designs

To further evaluate the proposed designs, we also compare our deep-pipelining designs with several existing AES hardware implementations. The results are shown in Table 4, where the performance results of the existing designs were reported respectively in the published works. The number of pipelining stages of different designs, as the most important design parameter, is shown in the table. The number of pipelining stages determines the system performance, area and latency. The latency of a 128-bit data passing through the AES encryption datapath can be calculated based on the stage number as $10 \times$ #Pipeline stages. The bit width of datapath is also a characteristic of the AES design and determines the required logic resources and I/O pins. The datapath of all designs compared in Table 4 is 128 bits.

From the table we can see that our design has the throughput higher than most existing designs on corresponding FPGA devices. The highest throughput we achieve is 75.92 Gbps on XC7VX690T with the clock frequency of 593.12 MHz. The slice usage is only 4.0% of

doi: 10.1049/iet-cdt.2014.0101

the total available slices on the device. Two parallel duplications of the present datapath design will easily achieve the throughput over 100 Gbps.

Compared to the non-pipelined design [8], which generates a result in every 11 cycles, our design improves the throughput by 14.32 times with the cost of 1.78 times more slice usage. Compared with [13], we achieve higher throughput and less use of slices. In [13], the AES implementation is composed of masked six-stage pipeline. The mask increases security as well as resource usage. The design [3] shows the highest throughput in Table 4, because it implemented four parallel encryption datapaths. The results of other designs shown in the table are based on one encryption datapath. On the same FPGA as in [3], two parallel instances of our design will achieve the similar performance with much less resource usage.

Our design is faster and smaller than [15], because the present design uses LUTs to implement LUT-based SubBytes, while [15] performs SubBytes in the composite field with more complex combinational logic. There are two pipelined AES designs in [7], and we label in this paper the two-stage pipelining design as [7^1] and the three-stage pipelining design as [7^2]. Our design with the new key expansion design is faster than [7^1], where the key expansion is the performance bottleneck. On the Virtex-4 FPGAs, the throughput of our design is faster than [22, 16, 7^2], but requires a larger FPGA to accommodate the eight-stage pipeline.

In terms of resource efficiency, our design on FPGAs with 6-input LUTs overruns all other designs, as shown in Table 4. This outcome stems from the fact that the elementary operations in AES are analysed in detail and a fine pipeline datapath is properly constructed with necessary registers inserted. On the FPGAs with 4-input LUTs, the resource efficiency of our deep-pipelined design is not promising, due to the large resource usage. Our two-stage pipelining design could be a better choice in this situation, although its efficiency is still worse than that of design [7^2]. The design [7^2] uses Block RAMs to implement the LUT in SubBytes and thus saves slices. However, our design does not require Block RAMs.

5.3 Key expansion scheme evaluation

To evaluate the new key expansion scheme, we also implement the proposed AES design with the original key expansion scheme and the key expansion scheme [20],

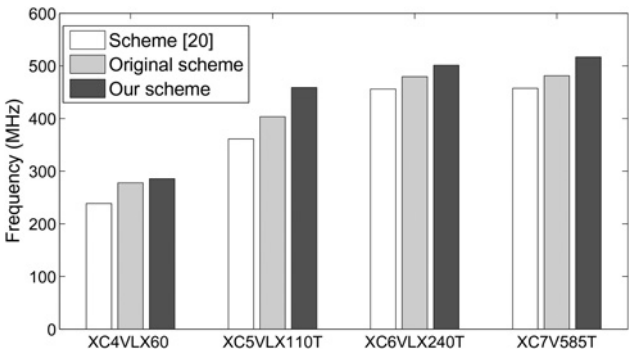

Fig. 8 *Performance comparison of three key expansion schemes*

Original scheme in the AES standard [4], the enhanced scheme [20] and our proposed scheme. The figure demonstrates that the AES design with our proposed key expansion scheme is superior to the other two schemes in performance

respectively. The results are shown in Fig. 8, where all designs are in the two-stage pipelining on four FPGA devices. The design adopting the new key expansion shows improved performance compared to ones with the other two, in particular up to 1.27 times faster than [20]. This is because the greatest logic depth in the key expansion scheme [20] is one level more than that of the proposed new scheme. This result is in accordance with the discussion in Section 4, and demonstrates that the proposed new scheme for key expansion increases speed of the AES implementation while enhancing the complexity of key cracking.

The cost of the improvement of speed and security is the extra resource required by the new scheme. To evaluate the area overhead, we isolate the key expansion scheme from the encrypting/decrypting datapath, and implement the original key expansion scheme in Fig. 7*a* and the new proposed key expansion scheme in Fig. 7*c*, respectively. On the XC7VX690T device, the original scheme needs 84 slices and the new scheme requires 121 slices. This area overhead is about 7.6% of the original AES design.

5.4 Image encryption application

Image encryption is an important and effective technique to protect image security, and has been a focus in the research of information security. In this experiment, we use the proposed AES design to perform image encryption for two

Fig. 9 *Image encryption example*

With the properly increased clock frequency, the AES can encrypt and decrypt the image correctly (b)
Further increasing the clock frequency leads to erroneous image (c)
a Original image
b Image after encryption and decryption by AES with the actual Fmax, where the image encryption error rate is 0%
c Image after encryption and decryption by AES with the clock frequency higher than the actual Fmax, where the image encryption error rate is 2.6%

purposes. First, we want to evaluate the real-time processing performance of the proposed AES design. We carry out the experiment on the Xilinx Zynq xc7z020 platform. The image under encryption is the Lena image in the 128 × 128 JPEG format as shown in Fig. 9*a*. Second, we want to show the potential of conservative timing margin to further increase the AES performance.

In the experiment, the image is first converted to a HEX format with 642 × 128 bits. To fully test the processing capability, we store the HEX data in on-chip block RAMs and each 128-bit is sent to the encryption/decryption datapath every clock cycle. The encrypted and decrypted data are also stored on-chip. To validate the correctness of the encrypting and decrypting operations, the encrypted-then-decrypted image is compared with the original image. Since the image is processed in the HEX format, the comparison of the two images is carried out 128 × 128 bits. If any bit of the 128-bit data is wrong, the whole data is considered corrupt due to the encryption and decryption process. We define the ratio of the number of corrupt data over the total number of data after encryption and decryption as image encryption error rate.

The maximum clock frequency of the deep-pipelining design on the FPGA device is 325 MHz as reported by the post-layout timing analyser. We gradually increase the clock frequency until errors are found, that is after encryption and decryption the image encryption error rate is not zero. In this way, we obtain the actual Fmax 545 MHz in our lab environment, which increases 67% over the conservative frequency while does not introduce any errors. Fig. 9*b* shows the correct outcome after AES encryption and decryption with the actual Fmax, where the image encryption error rate is 0%. Therefore the processing performance of the AES implementation is 69.76 Gbps, and encrypting the Lena image only needs 1.2 µs. When the clock frequency is over 545 MHz, the image after encryption and decryption is shown in Fig. 9*c*, where we hardly distinguish the lady, and the image encryption error rate is 2.6%.

6 Conclusions

Targeting on encryption/decryption of high speed data at 100 Gbps data rate, this paper proposed FPGA-based AES designs. These designs were elaborated with fine and deep pipelining, and this new key expansion scheme improved both throughput and security of the AES algorithm. The design approach started with the deep analysis of combinational logic in the critical operations involved in AES, to gain insight into the path delay. Based on the investigation of logic depth, with respect to different FPGA structures, pipelining structures were properly designed, leading to balanced pipeline stages with minimum logic delay. In addition, a new key expansion scheme is proposed to address the potential issues of existing key expansion scheme used in AES. The new scheme increased the complexity of key cracking and the speed of AES.

The proposed design achieved a throughput of 75.9 Gbps using a single pipeline on a latest FPGA device. Two parallel implementations of the proposed design can meet the real-time encryption/decryption demand for 100 Gbps data rate.

7 Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under grant no. 61204022, and by the Natural Science Foundation of Tianjin under grant no. 12JCYBJC30700.

8 References

1 Tate, J., Beck, P., Ibarra, H.H., Kumaravel, S., Miklas, L.: 'Introduction to storage area networks and system networking'. 2012 [Online]. Available at: http://www.ibm.com/redbooks
2 D'Ambrosia, J.: '40 Gigabit Ethernet and 100 gigabit Ethernet: the development of a flexible architecture [commentary]', *IEEE Commun. Mag.*, 2009, **47**, (3), pp. S8–S14
3 Henzen, L., Fichtner, W.: 'FPGA parallel-pipelined AES-GCM core for 100G Ethernet applications'. Proc. ESSCIRC, September 2010, pp. 202–205
4 National Institute of Standards and Technology (NIST), Information Technology Laboratory (ITL): 'Advanced encryption standard (AES)', in: Federal Information Processing Standards (FIPS) Publication 197, September 2001
5 Gaj, K., Chodowiec, P.: 'FPGA and ASIC implementations of AES'. Cryptographic Engineering (Springer, 2009), pp. 235–294
6 Liu, Q., Xu, Z., Yuan, Y.: 'A 66.1 Gbps single-pipeline AES on FPGA'. Proc. Int. Conf. Field-Programmable Technology, December 2013, pp. 378–381
7 Zhou, G., Michalik, H., Hinsenkamp, L.: 'Improving throughput of AES-GCM with pipelined karatsuba multipliers on FPGAs'. Reconfigurable computing: architectures, tools and applications (Springer, Berlin/Heidelberg), ser. Lecture Notes in Computer Science, 2009, pp. 193–203
8 Hussain, U., Jamal, H.: 'An efficient high throughput FPGA implementation of AES for multi-gigabit protocols'. Proc. Int. Conf. Frontiers of Information Technology, December 2012, pp. 215–218
9 Rais, M.H., Qasim, S.M.: 'Efficient hardware realization of advanced encryption standard algorithm using Virtex-5 FPGA', *Int. J. Comput. Sci. Netw. Secur.*, 2009, **9**, (9), pp. 59–63
10 Rais, M.H., Qasim, S.M.: 'A novel FPGA implementation of AES-128 using reduced residue of prime numbers based S-Box', *Int. J. Comput. Sci. Netw. Secur.*, 2009, **9**, (9), pp. 305–309
11 Hodjat, A., Verbauwhede, I.: 'A 21.54 Gbits/s fully pipelined AES processor on FPGA'. IEEE Proc. Int. Symp. Field-Programmable Custom Computing Machines, April 2004, pp. 308–309
12 Good, T., Benaissa, M.: 'AES on FPGA from the fastest to the smallest'. Cryptographic hardware and embedded systems (Springer, Berlin/Heidelberg), ser. Lecture Notes in Computer Science, 2005, pp. 427–440
13 Wang, Y., Ha, Y.: 'FPGA-based 40.9-Gbits/s masked AES with area optimization for storage area network', *IEEE Trans. Circuits Syst. II: Express Briefs*, 2013, **60**, (1), pp. 36–40
14 Järvinen, K.U., Tommiska, M.T., Skyttä, J.O.: 'A fully pipelined memoryless 17.8 Gbps AES-128 encryptor'. Proc. Int. Symp. Field Programmable Gate Arrays, ser. FPGA '03, New York, NY, USA, 2003, pp. 207–215
15 Reddy, R.S.S.K., Praneeth, P.: 'VLSI implementation of AES crypto processor for high throughput', *Int. J. Adv. Eng. Sci. Technol.*, 2011, **6**, (1), pp. 22–26
16 Zhou, G., Michalik, H., Hinsenkamp, L.: 'Efficient and high-throughput implementations of AES-GCM on FPGAs'. Proc. Int. Conf. Field-Programmable Technology, December 2007, pp. 185–192
17 Hodjat, A., Verbauwhede, I.: 'Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors', *IEEE Trans. Comput.*, 2006, **55**, (4), pp. 366–372
18 Mathew, S.K., Sheikh, F., Kounavis, M., *et al.*: '53 Gbps native GF$(2^4)^2$ composite-field AES-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors', *IEEE J. Solid-State Circuits*, 2011, **46**, (4), pp. 767–776
19 Shi, K., Boland, D., Constantinides, G.: 'Accuracy-performance tradeoffs on an FPGA through overclocking'. IEEE Proc. Int. Symp. Field-Programmable Custom Computing Machines, April 2013, pp. 29–36
20 Hu, L., Yuan, W., Yu, M.T., Chu, J.F., Liu, F.: 'One-way property strategy and improvement of key generation algorithm of Rijndael', *J. JILIN Univ. (Eng. Technol. Edn.)*, 2009, **39**, (1), pp. 137–142
21 Saberi, I., Shojaie, B., Salleh, M.: 'Enhanced key expansion for AES-256 by using even–odd method'. Proc. Int. Conf. Research and Innovation in Information Systems, November 2011, pp. 1–5
22 Chen, T., Huo, W., Liu, Z.: 'Design and efficient FPGA implementation of Ghash core for AES-GCM'. Proc. Int. Conf. Computational Intelligence and Software Engineering, December 2010, pp. 1–4