Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA

Umer Farooq, M. Faisal Aslam

Journal of King Saud University – Computer and Information Sciences (2016) xxx, xxx–xxx

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA ☆

Umer Farooq [a,*], M. Faisal Aslam [b]

[a] LIP6, Universite Pierre et Marie Curie, 4 Place Jussieu, 75005 Paris, France
[b] IAET, UOAS, Bremerhaven, Germany

Received 6 October 2015; revised 22 December 2015; accepted 10 January 2016

KEYWORDS

Cryptography;
Embedded security;
AES;
FPGA;
Exploration

Abstract Over the past few years, cryptographic algorithms have become increasingly important. Advanced Encryption Standard (AES) algorithm was introduced in early 2000. It is widely adopted because of its easy implementation and robust security. In this work, AES is implemented on FPGA using five different techniques. These techniques are based on optimized implementation of AES on FPGA by making efficient resource usage of the target device. Experimental results obtained are quite varying in nature. They range from smallest (suitable for area critical application) to fastest (suitable for performance critical applications) implementation. Finally, technique making efficient usage of resources leads to frequency of 886.64 MHz and throughput of 113.5 Gb/s with moderate resource consumption on a Spartan-6 device. Furthermore, comparison between proposed technique and existing work shows that our technique has 32% higher frequency, while consuming 2.63× more slice LUTs, 8.33× less slice registers, and 12.59× less LUT-FF pairs.

1. Introduction

During the past few years, cryptographic algorithms have been extensively used to fight security threats (Stallings, 2010). Secure systems are of tremendous importance nowadays. Secure transmission and storage of data are needed for all types of solutions, ranging from area sensitive embedded devices to massively parallel high performance computing devices. Such diversified area/performance requirements motivated us to explore different implementation techniques for Advanced Encryption Standard (AES) cryptographic algorithm. AES is an example of symmetric key cryptographic algorithm and today it is used in many cryptographic applications. AES algorithm (also known as Rijndael cipher algorithm) was accredited as the new commercial cryptographic algorithm in 2001 to replace aging Data Encryption Standard (DES) algorithm (Advanced Encryption Standard, 2001). Hardware-based implementation of AES algorithm is very

☆ This document is a collaborative effort.
* Corresponding author.
E-mail addresses: umer.farooq@lip6.fr (U. Farooq), maslam@studenten.hs-bremerhaven.de (M.F. Aslam).
Peer review under responsibility of King Saud University.

ELSEVIER Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2016.01.004

Please cite this article in press as: Farooq, U., Aslam, M.F. Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA Journal of King Saud University – Computer and Information Sciences (2016), http://dx.doi.org/10.1016/j.jksuci.2016.01.004

important as it is more secure, faster, and consumes less power as compared to its software-based implementation.

For hardware-based implementation of AES, Field Programmable Gate Arrays (FPGAs) are an attractive choice (Saggese et al., 2003; Elbirt et al., 2001). FPGAs were initially used as a glue logic. But during the past few years, they have seen a tremendous growth both in terms of market and capability and now they are used to implement complex applications. Recent cryptographic applications, ranging from fully pipelined parallel architectures to low power low cost implementations, are now increasingly implemented on FPGAs. The use of FPGAs has expanded because of the fact that they require less time to market as compared to their Application Specific Integrated Circuit (ASIC) counterpart (Kuon and Rose, 2006). Also, the development process of FPGAs is much more effective and requires less cost as compared to ASIC design. Furthermore, the reconfigurable nature of FPGAs gives the designer the ability to modify the initially implemented algorithm whereas no such provision exists for ASICs. This feature can be used to configure the device at run time. It can also be used to address the flaws in already implemented design and to optimize the algorithm for a fixed set of requirements (Dandalis and Prasanna, 2004).

Significant amount of work has been done lately regarding efficient AES implementation on FPGA. For example, the authors in Rouvroy et al. (2004) present a sequential implementation of AES algorithm on FPGA that results in a design that is well suited for small embedded applications. Implementation in Van Dyken and Delgado-Frias (2010) focusses on the design of low power FPGA-based encryption schemes. These schemes try to achieve best power results without compromising the throughput of the design. Three implementation schemes are presented which are compared in terms of area, and power consumption rates. Similarly, authors in Hoang and Nguyen present an efficient implementation of AES algorithm on FPGA that results in high throughput and it is well suited for applications requiring high speed and performance. Furthermore, authors in Soliman and Abozaid (2011), Gielata et al. (2008) and Qu et al. (2009) explore pipelining, subpipelining, and loop unrolling techniques to increase the frequency and throughput of AES implementation on FPGA. The work in Soliman and Abozaid (2011) used inner pipeline at two, three, or four stages on Xilinx Virtex-5 FPGA (Xilinx, 2014a) and achieved a maximum throughput of 73.737 Gb/s, maximum frequency of 576.07 MHz. Folded parallel architecture is used by Rahimunnisa et al. (2014) to obtain high throughput. The concept of folding is used to improve the area utilization while maintaining high throughput. They have achieved a maximum frequency of 505.5 MHz on a Xilinx Virtex-6 device. More recent work (Farashahi et al., 2014) presents a high speed hardware implementation of AES algorithm on Xilinx Virtex-5 device. They have achieved a throughput of 86 Gb/s and a maximum theoretical frequency of 671.524[1] MHz.

The main contribution of the work presented in this paper is that, here, we explore different implementation techniques. Depending upon the optimization applied, the results of implemented techniques range from the smallest to fastest implementation. Smaller implementation techniques require minimal resources and they are more suitable for embedded applications. On the other hand, high throughput techniques appear to be more resource hungry but they are suitable for high performance applications where throughput is a bigger concern than the amount of resources being used. As described in Section 2, AES is an iterative algorithm and we make use of this iterative approach to optimize its implementation. Furthermore, when considering FPGA hardware, resource mapping can play an important part as it can greatly influence the efficiency of a design. Benefits of an otherwise well optimized algorithm would be lost. Highlight of this work is that techniques presented here make use of efficient algorithm implementation as well as efficient resource mapping to explore their impact on architecture resource usage. Our results using Xilinx Spartan-6 (Xilinx, 2014b), Virtex-5 devices show that by carefully optimizing the algorithm implementation and by efficiently mapping the hardware resources, small area and high throughput can be achieved for different techniques. Furthermore, when compared with recent state-of-the-art results, our optimal technique gives better or comparable throughput while using significantly less FPGA resources.

The remainder of the paper is organized as follows. In Section 2, an overview of AES encryption algorithm is provided. In this section, brief description of different modules of AES algorithm is presented. Section 3 presents key optimization and resource mapping techniques for AES algorithm. Further, a discussion is presented in this section on how these techniques can result in good area and delay trade-offs. Experimental results are discussed in Section 4 demonstrating how different exploration techniques result in different area and performance values. Paper is finally concluded in Section 5 with a summary of some ideas that can be explored in future.

2. Overview of AES Rijndael design

In 2000, National Institute of Standards and Technology (NIST) selected Rijndael (Daemen and Rijmen, 2000) as the new Advanced Encryption Standard (AES) in order to replace aging Data Encryption Standard (DES). AES was selected because of its robust security properties and simple implementation both in software and hardware. AES is an iterative round based symmetric key block cipher that supports key size of 128, 192, 256 bits and block size of 128 bits. The use of larger key sizes increases the cryptographic strength but requires that greater number of iterative rounds be performed. In this work we focus on AES implementation with key size of 128 bits as it is sufficient for most purposes and is most commonly used.

2.1. Cipher module

Implementation of AES on hardware can be mainly divided into two modules: one is cipher module and other is key expansion module. Cipher module is responsible for performing encryption/decryption on the data while key expansion module is responsible for preparing the key that is required for each round of cipher. In case of 128 bit key, cipher module performs ten rounds of substitution and permutation to transform the input data to ciphered data. For the first 9 rounds of encryption, cipher module makes use of SubByte, ShiftRow, MixColumn, AddRoundKey operations and for final round

[1] Theoretical frequency limited by maximum available frequency of FPGA which is 550 MHz

Please cite this article in press as: Farooq, U., Aslam, M.F. Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA Journal of King Saud University – Computer and Information Sciences (2016), http://dx.doi.org/10.1016/j.jksuci.2016.01.004

Mixcolumn operation is skipped to complete the encryption process. Based on the original key, key expansion module calculates key that is used in each round by aforementioned operations. Fig. 1 shows standard structure of AES algorithm. It can be seen from the figure that different functions of cipher module combined with key expansion module perform the encryption on input data through an iterative process. Input data in AES is often represented as 4×4 bytes array and it is termed as "state". A brief discussion on different operations of cipher module and key expansion module is recalled (Advanced Encryption Standard, 2001) here.

SubByte function performs a non-linear transformation independently on each byte of the input state. This transformation is performed by substituting each byte of the state with a value from substitution box (also termed as S-box). There are 16 parallel S-boxes each with 8 inputs and 8 outputs. The S-box operation is the only nonlinear transformation of AES algorithm. It is an invertible operation and can be used for decryption process too. The construction of S-box is achieved by combining two transformations. The first transformation is performed by taking the multiplicative inverse in the finite field $GF(2^8)$ where an all zero bit input is mapped to itself. In the second part, affine transformation is performed over GF(2). The input is an 8 bit vector in GF(2) and it is multiplied by a constant 8 by 8 bit matrix M and then added to 8 bit vector C. Both constant matrices M and C are given below.

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

The ***ShiftRow*** function performs byte wise circular shifts on last three rows of the state. In this function, first row is not rotated, but second, third, and fourth rows are rotated by one, two, three bytes respectively. This rotation provides diffusion property of the AES algorithm.

MixColumn function separately operates on each of the four columns of states treating each column as a four-term polynomial. The columns are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)$ given by

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

Above function can also be written as matrix multiplication as $s'(x) = a(x)xs(x)$ or in matrix form

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

AddRoundKey is the final cipher function and is used to mix key information with the data that are being operated upon. Inputs of this function are 16 byte state and 16 byte key which is obtained from key expansion algorithm. Its output is a simple bit wise XOR operation between current round state and current round expanded key.

2.2. Key expansion module

Main purpose of this function is to calculate round key for each round of the cipher based on original key. As shown in Fig. 1, this module comprises of three simple sub-modules SubWord, RotWord, and RoundConst. SubWord is a function that takes a four byte input word and applies S-box to each of four bytes to produce output word. RotWord takes a word, performs cyclic permutation, and returns the word. RoundConst function contains a round constant array that performs a bitwise XOR function. Round constant array contains values given by $[x_{i-1}, \{00\}, \{00\}, \{00\}]$ with x_{i-1} being powers of x (x is denoted as $\{02\}$) in the field $GF(2^8)$. KeyExpansion is an important function and depending upon the target architecture its implementation can be performed either by using computing resources or by using on-chip memory resources that eventually leads to different results.

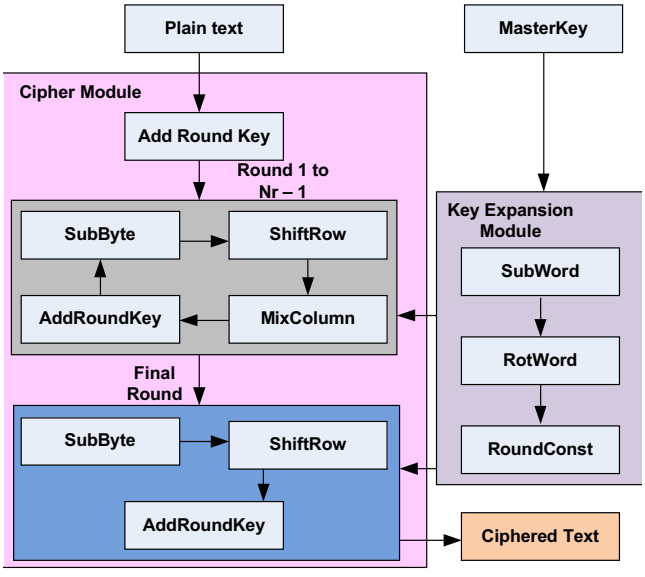

Figure 1 Standard overview of AES algorithm.

3. Proposed AES algorithm implementation

Optimizations performed in the AES algorithm and different resource mapping options are discussed in this section. Different combinations of these optimizations and resource mapping options then lead to five exploration techniques.

3.1. Optimized implementation

Based on the explanation of AES design given in Section 2, it can be seen that its implementation can be mainly divided into two separate but dependant parts: one is key expansion module while the other a cipher module. Cipher module is an essentially iterative looping structure and classic loop optimization technique like 'loop unrolling' are applied to optimize its implementation in terms of speed. Loop unrolling is a technique that replaces a looping structure with N copies of that looping body, hence reducing the total loop iterations by N

times (i.e. inter-round optimization). Loop unrolling in the cipher module can be combined with key expansion routine to achieve fully parallel implementation of AES algorithm. Key expansion routine can be intuitively splitted into smaller key expansion modules that would be placed along unrolled round operations of cipher module (termed as online key expansion). Loop unrolling of cipher module combined with splitted key expansion modules (i.e. intra-round optimization), result in fully parallel implementation of AES algorithm on FPGA. This parallel implementation should result in an improved throughput of AES algorithm. This performance advantage, however, comes with a price as unrolling and splitting will increase the required FPGA resources.

3.2. FPGA resource mapping

It is well known that implementation decisions have significant impact on the area and performance of target architectures. In this work, we chose Spartan-6 family of Xilinx. Like many Xilinx FPGAs, Spartan-6 device is a two dimensional mesh-based FPGA. It contains a mesh of Look-Up Tables (LUTs)/Configurable Logic Blocks (CLBs)[2]. LUTs are mainly used for implementation of combinational and sequential logic. They can also be configured as memory elements. These LUTs are connected to each other through programmable routing fabric and connections to the external world are made through programmable I/O pads. Apart from LUTs, Spartan-6 device also contains a dedicated amount of Block RAMS (BRAMS) that we use for different implementation techniques in this work.

CLBs are main source for implementation of AES computation operations. Majority of AES algorithm operations (e.g. AddRoundKey, MixColulmn, ShiftRow etc) can be implemented using CLBs. But the operations like SubByte that involve S-box can be implemented using one of following options.

- Block RAM (BRAM): For SubByte operation (used both in cipher module and key expansion module), S-box is normally required that replaces original values with values from S-box. The values in the S-box are predefined and these values can be stored in BRAMS. BRAMS in Spartan-6 device are dedicated memories and S-box values can be loaded into them at the time of configuration.
- Logic Blocks: CLBs in Spartan-6 device can be used both as computation resource as well as storage resource. S-box values for SubByte operations can be directed to be stored in CLBs through different coding techniques in VHDL. Main advantage of using CLBs as memory is that synthesis tool can better optimize area and delay constraints during synthesis procedure.

3.3. Exploration techniques

Based on the discussion presented in Section 3.1 and 3.2, we have implemented AES algorithm in five different ways. These five exploration techniques give the results ranging from smallest area to fastest implementation. A brief description of these techniques is as follows:

[2] Terms LUT, CLBs are used interchangeably in this paper

3.3.1. CB-KB-S

In this technique, cipher module's S-box is implemented in BRAM and key expansion module's S-box is also implemented in BRAM. Two modules are executed in a serial way: first key expansion is performed and then cipher module is executed. This is the simplest implementation technique. This technique gives us the best area results but at the same time it gives the worst delay results. Implementation results obtained from this technique suggest that this technique is useful for embedded applications requiring area critical results.

3.3.2. CB-KB-P

In this technique, cipher module's S-box is implemented in BRAM and key expansion module's S-box is also implemented in BRAM. Contrary to the first implementation technique, in this technique loop unrolling is performed in cipher module and key expansion is performed online. Online key expansion improves performance in a great way, i-e instead of waiting for the complete key expansion,we have enabled this module to provide the cipher key for each stage of cipher module. This will eventually minimize the critical path of whole design. Because of the parallel operation, this technique gives better delay results but poor area results as compared to 'CB-KB-S' exploration technique.

3.3.3. CB-KC-S

In this technique, cipher module's S-box is implemented in BRAM and key expansion module's S-box is implemented using CLBs. Further, two modules are executed in a serial way: first key expansion is performed and then cipher module is executed. In this implementation, since KeyExpansion S-box is implemented using CLBs, number of BRAMs are less but more number of CLBs would be required compared to CB-KB-S technique. Further, S-box implementation in CLBs would lead to better delay results.

3.3.4. CB-KC-P

In this technique, cipher module's S-box is implemented in BRAM and key expansion module's S-box is implemented using CLBs. Moreover, in this technique loop unrolling is performed in cipher module and key expansion is performed online. Because of the parallel operation, this technique gives good delay results but poor area results as compared to 'CB-KB-S' exploration technique.

3.3.5. CC-KC-S

In this technique, both cipher and key expansion modules' S-box are implemented using CLBs. Implementation of S-box using CLBs leads to best possible delay results with no BRAMs at all. But this technique requires much more CLBs than all other exploration techniques. Further discussion on the exploration results of these exploration techniques is presented in Section 4.

4. Results and analysis

4.1. Experimental setup

In this work, we have implemented 128 bit key length AES algorithm using five different exploration techniques

Please cite this article in press as: Farooq, U., Aslam, M.F. Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA Journal of King Saud University – Computer and Information Sciences (2016), http://dx.doi.org/10.1016/j.jksuci.2016.01.004

(described in Section 3.3). Implementation of AES algorithm is performed on xc6s1x150-3-fgg900 device which is a member of Xilinx Spartan-6 family. The VHDL core of the design is synthesized, placed and routed using Xilinx ISE 13.2 release where explicit directives were used to determine the mapping of S-box and loop unrolling was achieved through parallel processes for cipher and key expansion module. The Xilinx tool was used to measure number of slice registers, number of slice LUTs, maximum frequency, and critical path delay for each exploration technique. From these values, we also calculated theoretical throughput and efficiency of each design using Eqs. (2) and (3) respectively. As a final measure of quality of design, we also compare total resource usage and delay of each implementation scheme that gives an idea about the area-delay tradeoff. As discussed in Section 1, most of existing work uses Virtex-5 as target device. So, we also implement our best technique on this device to have a fair comparison between our work and existing work.

$$T_{\text{put}} = \frac{Number\ of\ processed\ bits}{Critical\ path\ delay} \tag{2}$$

$$Efficiency = \frac{T_{\text{put}}}{Total\ resources} \tag{3}$$

4.2. Experimental results

Experimental results of five exploration techniques are shown in Figs. 2–4b. As it can be seen from these figures that exploration techniques are expressed as 'CX-KY-Z' where X, Y indicate the implementation of cipher module and key expansion module's S-box either in BRAM (B) or in CLB (C) respectively. As far as 'Z' is concerned, it indicates whether AES is implemented in serial (S) or parallel (P) manner. For example, an exploration technique with name 'CB-KC-P' indicates that cipher module S-box is implemented in BRAM, key expansion S-box is implemented in CLBs and loop unrolling is applied to have a parallel execution.

Fig. 2a gives the number of slice registers used by different exploration techniques. It can be seen from this figure that techniques employing parallelism, require more slice registers as compared to techniques employing serial implementation. This is because of the fact that at each stage of cipher module, registers are required to synchronize the cipher key expansion with cipher module to ensure safe operation. Without registers it can provide invalid/wrong key to cipher module during online operation. Similarly, Fig. 2b gives the number of slice LUTs used by each technique under consideration. As expected, techniques using BRAMs for their S-box implementation require less number of LUTs as compared to techniques using combinational blocks for the S-box implementation. Furthermore, it can be seen from the figure that CC-KC-S technique requires the most number of LUTs as it is implementing both its cipher module, key expansion module's S-box using combinational blocks.

In order to perform the performance analysis of different exploration techniques, we have also measured the frequency and critical path delay results and these results are presented in Fig. 3a. In this figure, solid line indicates the frequency results while dashed line gives the delay results for different

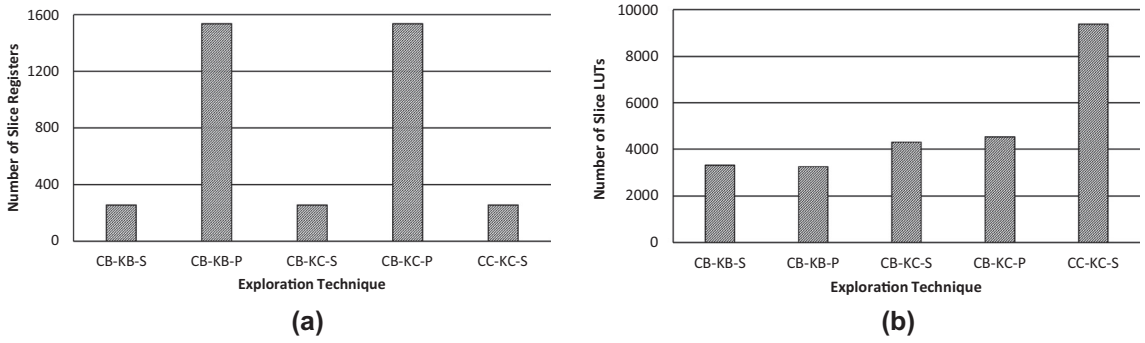

Figure 2 (a) Slice registers used; (b) Slice LUTs used.

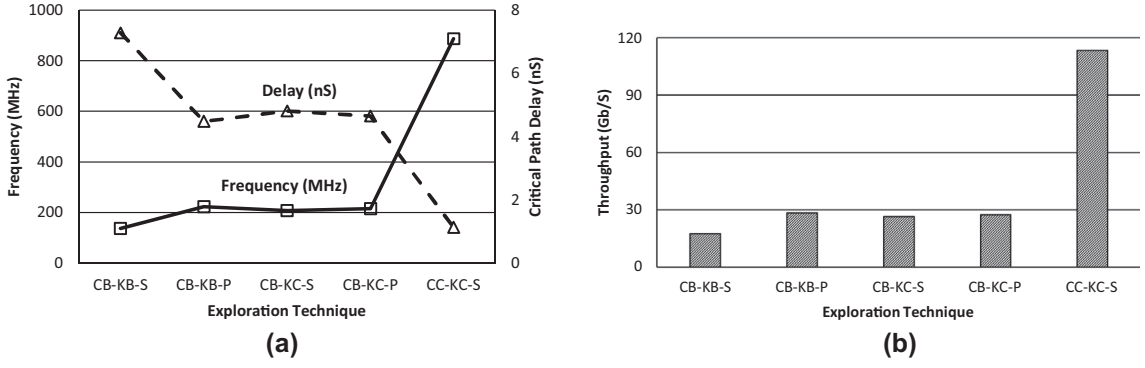

Figure 3 (a) Frequency-delay comparison; (b) Throughput comparison.

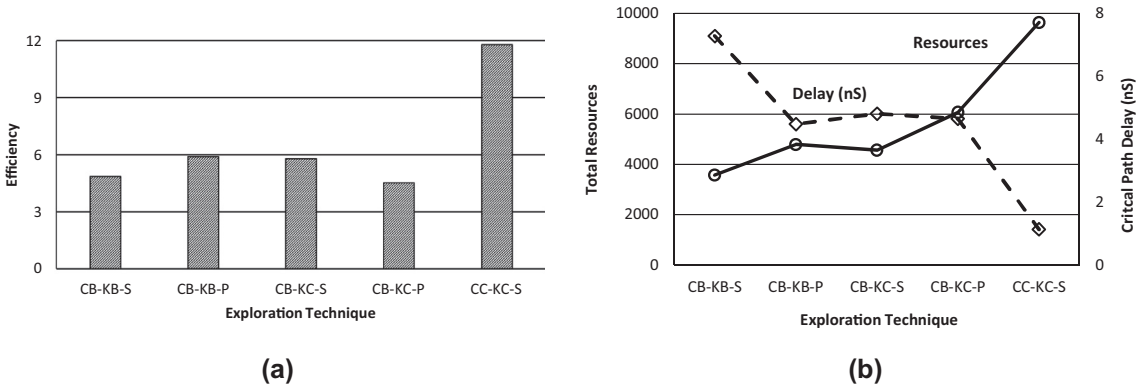

Figure 4 (a) Efficiency comparison; (b) Area-delay comparison.

exploration techniques under consideration. Furthermore, we have also computed the throughput (ref Eq. (2)) of each implementation and relevant results are shown in Fig. 3b. It can be seen from both figures that in general when designs are implemented using parallelism, they lead to higher frequency which eventually results in better throughput - see the comparison between CB-KB-S and CB-KB-P, CB-KC-S and CB-KC-P. There is one exception however, in CC-KC-S technique, there is no parallelism. But implementation of both S-box in CLBs results in better frequency and throughput results although it requires more number of slice LUTs as compared to other exploration techniques. Better frequency results are obtained in this case because of the fact that here communication delay between CLBs and BRAMs is removed that has resulted in smaller critical path delay eventually leading toward higher throughput compared to other techniques under consideration.

Based on our experimentation, we have also computed the efficiency (ref Eq. (3)) of each exploration technique and it can be seen from Fig. 4a that CC-KC-S technique gives us the best efficiency results. This is because of the fact that compared to other designs, this technique uses least amount of registers, only twice as many slices and no BRAMs at all. Further, this technique gives best frequency results that eventually gives best efficiency results. As a final measure of the quality of design, we have also performed area-delay tradeoff analysis of all the techniques and relevant results are shown in Fig. 4b. In this figure, area is the sum of number of all used resources (shown as solid line) and delay is calculated using frequency results of implemented techniques (shown as dashed line). Results in Fig. 4b show that again, in our case, CC-KC-S gives the best area-delay trade-off. This is because of the fact that compared to CB-KC-P, this design consumes only twice as many slices with smallest number of registers and no BRAMs at all. Further, this design gives best frequency results, leading to smallest critical path delay and eventually resulting in lowest area-delay product.

4.3. Comparison results

For the sake of completeness, we have also compared results of our best technique (i.e. CC-KC-S) with recent state-of-the-art results. For that purpose, we implemented our design on Xilinx Virtex-5 FPGA, since most of the recent work uses this FPGA as target device. We have compared the results of our technique and recent work against a number of area and performance parameters. For example, Fig. 5 shows the comparison for number of slice LUTs used by different techniques. In this figure, x-axis gives reference number and device used by each technique while y-axis gives total number of slices used by each technique. It can be seen from the figure that (Farashahi et al., 2014) uses least number of slice LUTs and when compared to our implementation, we consume 61.6%, 62% more resources on Virtex-5, Spartan-6 devices respectively. However, the comparison of LUT-FF pair usage in Fig. 6 shows that among existing work, 4-stage technique of Farashahi et al. (2014) uses least number of LUT-FF pairs and compared to this technique, our proposed technique uses

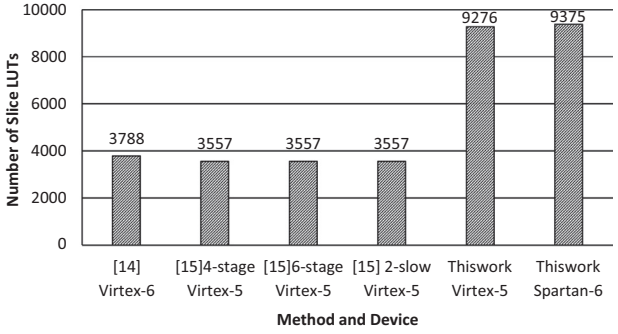

Figure 5 Slice LUTs comparison between existing and proposed work.

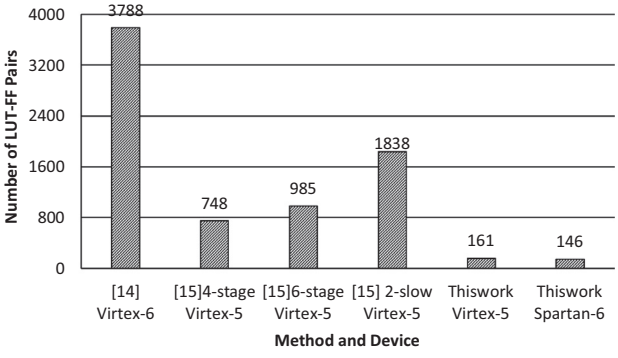

Figure 6 LUT-FF comparison between existing and proposed work.

78.5%, 80.5% less number of LUT-FF pairs on Virtex-5, Spartan-6 devices respectively.

For a complete picture of resource usage, we also perform a comparison between number of slice registers used by different techniques. These comparison results are shown in Fig. 7. It can be seen from the figure that 4-stage technique of Farashahi et al. (2014) uses least number of registers among the state-of-art techniques and when compared to this technique, our proposed technique uses 68.44%, 68.31% less slice registers on Virtex-5, Spartan-6 devices respectively. Results presented in Figs. 5–7 suggest that although, our proposed technique consumes more slice LUTs but it consumes much less LUT-FF pairs, slice registers when compared to best results of recent state-of-art techniques.

Since frequency of an implementation scheme gives insight of the critical path delay of the implementation, we also perform a frequency comparison between different implementation techniques. Frequency comparison of different techniques is shown in Fig. 8. It can be seen from the figure that 2-slow technique of Farashahi et al. (2014) gives the best frequency results among existing techniques. When compared to the proposed technique, our implementation on Virtex-5 is only 4% slower while our implementation on Spartan-6 is 32% faster than 2-slow technique of Farashahi et al. (2014) while using fewer number of available FPGA resources (ref Figs. 5–7). Finally, we also perform a throughput comparison between existing techniques and the proposed technique. This comparison is presented in Fig. 9. It can be seen from the figure that 2-slow technique of Farashahi et al. (2014) gives the best throughput results among existing techniques. When compared to the proposed technique, our implementation on Virtex-5 gives only 4% less throughput while our implementation on Spartan-6 gives 32% more throughput than 2-slow technique of Farashahi et al. (2014).

Detailed comparison results are presented in Figs. 5–9. It can be seen from these figures that 4-stage technique of Farashahi et al. (2014) gives best overall resource usage results among the existing techniques. However, when compared, our

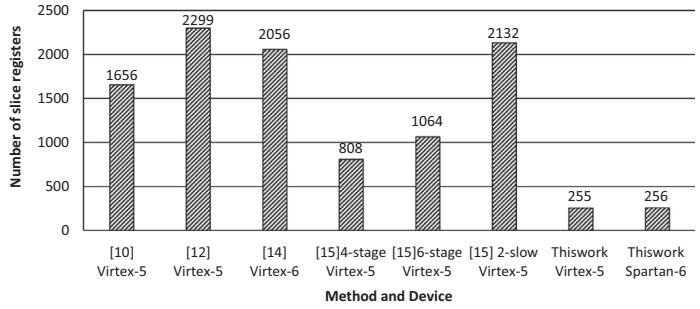

Figure 7 Slice registers comparison between existing work and proposed technique.

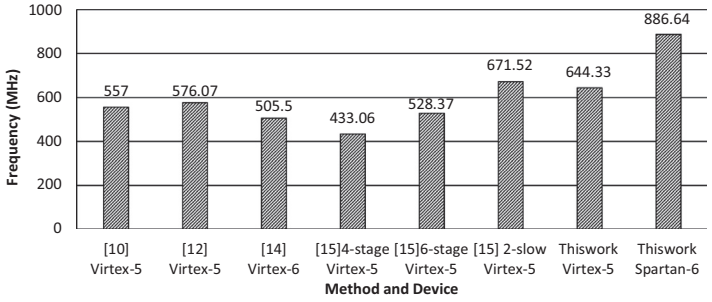

Figure 8 Frequency comparison between existing work and proposed technique.

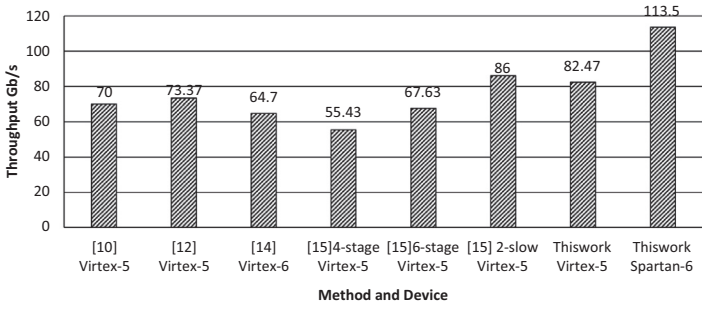

Figure 9 Throughput comparison between existing work and proposed technique.

technique gives better resource usage results (except number of slice LUTs) while giving much better frequency and throughput results (4-stage technique of Farashahi et al. (2014) is 51% slower than our implementation on Spartan-6). Similarly, when we compare the best frequency results of existing technique (i.e. 2-slow of Farashahi et al., 2014), our technique gives comparable (in case of Virtex-5 implementation) or better results (in case of Spartan-6 implementation) while consuming much less device resources.

5. Conclusion

This paper presents a study that explored five different implementation techniques for AES algorithm. In these techniques, we have made use of optimizations like loop unrolling that introduced parallelism in our design. Further, different FPGA resource mappings have lead to different results. Based on experimentation, it is seen that generally sequential implementations lead to better area results but poor performance results. Also, it was observed that parallelism leads to better delay results but poor area results as more registers are required. Moreover, it was observed that efficient usage of computing resources (i.e. CLBs) of FPGAs leads to better performance results and frequency as high as 886.64 MHz can be achieved in certain scenario. Finally the comparison of five techniques has shown that the one with efficient CLB usage (i.e. CC-KC-S) requires least BRAMs, registers and gives best results in terms of frequency and throughput eventually leading to best area-delay trade-off. Furthermore, comparison between the best proposed technique and existing work shows that the proposed techniques makes good use of available resources and gives better tradeoff as far as the resource usage and throughput of the design is concerned.

In future, it can be interesting to explore the effect of more device features of Spartan-6 device. Furthermore, it will be interesting to integrate some dynamic reconfiguration mechanism so that AES algorithm implementation may be more optimized.

References

Advanced Encryption Standard (AES), 2001.

Daemen, J., Rijmen, V., 2000. The block cipher Rijndael, 1820, 277–284.

Dandalis, A., Prasanna, V.K., 2004. An adaptive cryptographic engine for internet protocol security architectures. ACM Trans. Des. Autom. Electron. Syst. 9, 333–353.

Elbirt, A., Yip, W., Chetwynd, B., Paar, C., 2001. An FPGA-based performance evaluation of the AES bolck cipher candidate algorithm finalists. IEEE Trans. Very Large Scale Integr. VLSI Syst. 9, 545–557.

Farashahi, R.R., Rashidi, B., Sayedi, S.M., 2014. FPGA based fast and high-throughput 2-slow retiming 128-bit AES encryption algorithm. Microelectron. J. 45, 1014–1025.

Gielata, A., Russek, P., Wiatr, K., 2008. AES hardware implementation in FPGA for algorithm acceleration purpose. In: International Conference on Signals and Electronic Systems. ICSES '08, pp. 137–140.

Hoang, T., Nguyen, V.L. An efficient FPGA implementation of the advanced encryption standard algorithm. In: 2012 IEEE RIVF International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), pp. 1–4.

Kuon, I., Rose, J., 2006. Measuring the gap between FPGAs and ASICs. In: Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays. ACM, New York, NY, USA, pp. 21–30.

Qu, S., Shou, G., Hu, Y., Guo, Z., Qian, Z., 2009. High throughput, pipelined implementation of AES on FPGA. In: International Symposium on Information Engineering and Electronic Commerce. IEEC '09, pp. 542–545.

Rahimunnisa, K., Karthigaikumar, P., Rasheed, S., Jayakumar, J., SureshKumar, S., 2014. FPGA implementation of AES algorithm for high throughput using folded parallel architecture. Secur. Commun. Netw. 7, 2225–2236.

Rouvroy, G., Standaert, F.X., Quisquater, J.-J., Legat, J., 2004. Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications. In: Proceedings of the International Conference on Information Technology: Coding and Computing. ITCC 2004, vol. 2, pp. 583–587.

Saggese, G., Mazzeo, A., Mazzocca, N., Strollo, A., 2003. An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm. 2778, 292–302.

Soliman, M.I., Abozaid, G.Y., 2011. {FPGA} implementation and performance evaluation of a high throughput crypto coprocessor. J. Parallel Distrib. Comput. 71, 1075–1084.

Stallings, W., 2010. Cryptography and Network Security: Principles and Practice, fifth ed. Prentice Hall.

Van Dyken, J., Delgado-Frias, J.G., 2010. FPGA schemes for minimizing the power-throughput trade-off in executing the advanced encryption standard algorithm. J. Syst. Archit. 56, 116–123.

Xilinx, 2014a. Virtex-5, <http://www.xilinx.com/>.

Xilinx, 2014b. Spartan-6, <http://www.xilinx.com/products/>.

Please cite this article in press as: Farooq, U., Aslam, M.F. Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA Journal of King Saud University – Computer and Information Sciences (2016), http://dx.doi.org/10.1016/j.jksuci.2016.01.004