2009 International Conference on Reconfigurable Computing and FPGAs

Efficient Technique for the FPGA Implementation of the AES MixColumns Transformation

Solmaz Ghaznavi, Catherine Gebotys, and Reouven Elbaz
Dept. of Electrical and Computer Engineering, University of Waterloo
Waterloo, Canada
sghaznav@ecemail.uwaterloo.ca, {cgebotys, reouven}@uwaterloo.ca

***Abstract*—The advanced encryption standard, AES, is commonly used to provide several security services such as data confidentiality or authentication in embedded systems. However designing efficient hardware architectures with small hardware resource usage and short critical path delay is a challenge. In this paper, a new technique for the FPGA implementation of the *MixColumns* transformation, an important part of AES, is introduced. The proposed *MixColumns* architecture, targeting 4-input LUTs on an FPGA, uses up to 23% less hardware resources than previous research. Overall, incorporating the proposed technique along with block memories for the *SubBytes* transformation in the AES encryption reduces usage of hardware resources by up to 10% and 18% in terms of slices and LUTs, respectively. The improvement is obtained by more efficient resource sharing through expansion and rearrangement of the *MixColumns* equation with respect to the structure of FPGAs. This can be highly advantageous in an FPGA implementation of block cipher modes using AES in many secure embedded systems.**

***Keywords*-AES; MixColumns; FPGA; LUT; architecture;**

I. Introduction

It has been observed that the cost of failure in providing security services, such as confidentiality and authentication, can be very high. For instance, a security survey by the Computer Security Institute (CSI) and Federal Bureau of Investigation (FBI) revealed that just 223 organizations sampled from various industry sectors had lost hundreds of millions of dollars due to computer-related crime [1]. The Advanced Encryption Standard (AES), the focus of this research, can be used to provide security services such as data confidentiality or authentication. Data confidentiality provides protection of data from being disclosed by unauthorized parties. Data authentication is the assurance that the received data has not been replayed or affected by modification, insertion, or deletion, and also the sender is authenticated. AES was standardized by the National Institute of Standards and Technology (NIST) in 2000 and is predicted to be secure well beyond 30 years [2].

A block cipher mode, or mode for short, is a technique for adapting a symmetric-key block cipher algorithm for an application and a message length. NIST currently has recommended 8 modes for the approved block ciphers in a series of special publications [3]. The specification of the AES block cipher, shown in Fig. 1, defines two functions: encryption that generates ciphertext and decryption that produces plaintext. Among the 8 modes recommended by NIST 6 of them solely use encryption. These 6 modes are designed such that encryption is used for generating ciphertext as well as plaintext. For instance, the well-known counter (CTR) mode uses encryption to generate a pseudorandom key stream from a counter, and combines that key stream with the message through a XOR operator to generate ciphertext. To obtain the plaintext message, the ciphertext message is XORed with that same key stream. To do so, the keystream is re-generated with the counter and the same encryption used to produce the ciphertext.

Reducing the usage of FPGA hardware resources results in fitting a system in a smaller device. For instance, using a smaller device can be an important factor in reducing the hardware cost in embedded systems. In applications, such as space application, that have stringent power requirements, a smaller FPGA device size decreases the static power consumption. This can significantly reduce the overall power consumption.

In AES, there are two expensive transformations in terms of computational resources, namely the *MixColumns* and *SubBytes* transformations. The focus of this research is on the *MixColumns* transformation in encryption. Compared to previous work, a *MixColumns* design which uses the smallest number of hardware resources on an FPGA is proposed. More specifically, incorporating the proposed *MixColumns* architecture in the AES encryption results in the smallest number of slices and 4-input Look Up Tables (LUTs) compared to previous research. The improvement is obtained by rearrangement of the *Mixcolumns* equation with respect to the structure of FPGAs.

The rest of the paper is organized as follows. Section II will present a brief overview of AES and the *MixColumns* transformation. In section III, the proposed technique for the *MixColumns* implementation will be introduced. Section IV will provide the existing techniques previously proposed to implement the *MixColumns*. Experimental results of the *MixColumns* and AES encryption on a Virtex 4 FPGA will be shown in section V. Finally, section VI will provide the conclusion.

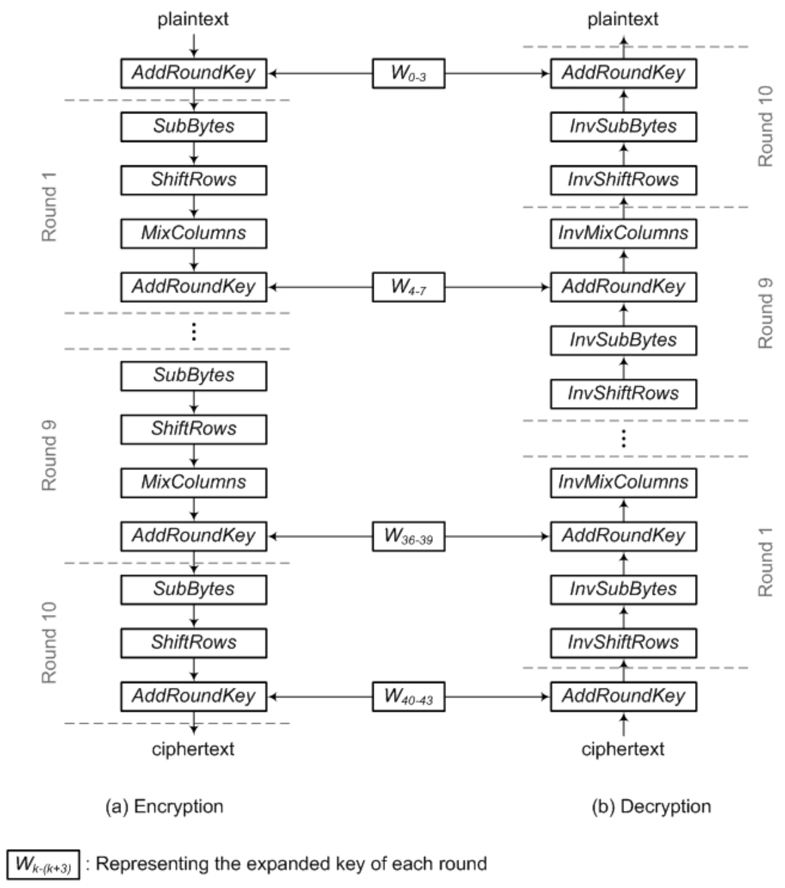

Figure 1. AES algorithm

II. Overview of AES and MixColumns

AES is a block cipher based on the concepts of diffusion and confusion described by Shannon. While most block ciphers follow these principles, few do so as clearly as AES that is a substitution permutation (SP)-network [4]. The objective of confusion is to make the relationship between plaintext, key, and ciphertext as complex as possible. The *SubBytes* transformation provides AES with confusion. Diffusion spreads the influence of plaintext and key throughout ciphertext. The *MixColumns* transformation which modifies the four columns of the state matrix independently ensures diffusion in AES.

A. AES

The AES algorithm for the key size of 128 bits, shown in Fig. 1, consists of 10 rounds. There are 12 and 14 rounds for the key size of 192 and 256 bits, respectively. The round transformations are *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey* [5]. In each round, data is processed as a 4x4 matrix, called the *state*, with 8-bit elements.

B. Mixcolumns

The *MixColumns* transformation is a function whose input is a column of the state (specifically four 8-bit values $[a\ b\ c\ d]$) and produces a corresponding output column (also four 8-bit values $[a'\ b'\ c'\ d']$). It is defined by equation 1 in polynomial representation where all the polynomial coefficients are in $GF(2^8)$. The *MixColumns* transformation is also shown in equation 2 in the equivalent matrix multiplication form. In the result column $[a'\ b'\ c'\ d']$, each 8-bit row depends on all the rows of the input column $[a\ b\ c\ d]$.

$$result{=}(dx^3{+}cx^2{+}bx{+}a)\cdot(3x^3{+}1x^2{+}1x{+}2)\bmod(x^4+1) \quad (1)$$

$$\begin{aligned} result &= \begin{bmatrix} a' \\ b' \\ c' \\ d' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \\ &= \begin{bmatrix} 2a & \oplus & 3b & \oplus & c & \oplus & d \\ a & \oplus & 2b & \oplus & 3c & \oplus & d \\ a & \oplus & b & \oplus & 2c & \oplus & 3d \\ 3a & \oplus & b & \oplus & c & \oplus & 2d \end{bmatrix} \end{aligned} \quad (2)$$

The multiplication by 2 (i.e. times x in the polynomial form) is modulo polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ in $GF(2^8)$. Therefore it can be implemented by three XOR operations and 4 single-bit shifts. The multiplication by 2, which is denoted by $xtime()$, is further described as follows where z is an 8-bit element of the state and z_i refers to bit i ($i \in \{0, ..., 7\}$).

$$\begin{aligned} Input: z &= [z_7\ z_6\ z_5\ z_4\ z_3\ z_2\ z_1\ z_0] \\ Output: z' &= 2z = xtime(z) \qquad (3) \\ z'_0 &= z_7 \\ z'_1 &= z_0 \oplus z_7 \\ z'_2 &= z_1 \\ z'_3 &= z_2 \oplus z_7 \\ z'_4 &= z_3 \oplus z_7 \\ z'_5 &= z_4 \\ z'_6 &= z_5 \\ z'_7 &= z_6 \end{aligned}$$

Consequently, we will have the following equation for the multiplication by 3 (i.e. $xtime(z) \oplus z$) in terms of polynomials.

$$\begin{aligned} Input: z &= [z_7\ z_6\ z_5\ z_4\ z_3\ z_2\ z_1\ z_0] \\ Output: z' &= 3z \\ &= xtime(z) \oplus z \qquad (4) \end{aligned}$$

For example, bit 0 of the first 8-bit row a' of the result in equation 2 is defined as (from equation 3 and 4) $a'_0 = a_7 \oplus b_7 \oplus b_0 \oplus c_0 \oplus d_0$, whereas bit 1 of a' becomes $a'_1 = a_0 \oplus a_7 \oplus b_0 \oplus b_7 \oplus b_1 \oplus c_1 \oplus d_1$.

III. PROPOSED MIXCOLUMNS DESIGN ON AN FPGA

In this research, the *MixColumns* equation is expanded and rearranged so that it uses the smallest number of slices and LUTs on an FPGA that contains 4-input LUTs. The proposed architecture distinguishes 2 different groups of output bits in 8-bit elements of the state column as follows.

- the output bits at positions $\{0, 2, 5, 6, 7\}$ for which the $xtime()$ operation requires only a bit shift (shown in equation 3)
- the output bits at positions $\{1, 3, 4\}$ for which the $xtime()$ operation requires a bit shift and an XOR (shown in equation 3)

The expansion and rearrangement of *MixColumns* in equation 5 is proposed for the first group where a bit position $i \in \{0, 2, 5, 6, 7\}$. This allows $a_i \oplus b_i \oplus c_i \oplus d_i$ to be shared amongst all the 4 bits (a'_i, b'_i, c'_i, d'_i) of the output state column.

$$\begin{aligned} result_i &= \begin{bmatrix} a'_i \\ b'_i \\ c'_i \\ d'_i \end{bmatrix} \\ &= \begin{bmatrix} 2a_i & \oplus & 3b_i & \oplus & c_i & \oplus & d_i \\ a_i & \oplus & 2b_i & \oplus & 3c_i & \oplus & d_i \\ a_i & \oplus & b_i & \oplus & 2c_i & \oplus & 3d_i \\ 3a_i & \oplus & b_i & \oplus & c_i & \oplus & 2d_i \end{bmatrix} \\ &= \begin{bmatrix} a_i \oplus b_i \oplus c_i \oplus d_i \\ a_i \oplus b_i \oplus c_i \oplus d_i \\ a_i \oplus b_i \oplus c_i \oplus d_i \\ a_i \oplus b_i \oplus c_i \oplus d_i \end{bmatrix} \oplus \begin{bmatrix} 2b_i \oplus 2a_i \oplus a_i \\ 2b_i \oplus 2c_i \oplus b_i \\ 2d_i \oplus 2c_i \oplus c_i \\ 2d_i \oplus 2a_i \oplus d_i \end{bmatrix} \\ &= \begin{bmatrix} a_i \oplus b_i \oplus c_i \oplus d_i \\ a_i \oplus b_i \oplus c_i \oplus d_i \\ a_i \oplus b_i \oplus c_i \oplus d_i \\ a_i \oplus b_i \oplus c_i \oplus d_i \end{bmatrix} \oplus \begin{bmatrix} b_{i-1} \oplus a_{i-1} \oplus a_i \\ b_{i-1} \oplus c_{i-1} \oplus b_i \\ d_{i-1} \oplus c_{i-1} \oplus c_i \\ d_{i-1} \oplus a_{i-1} \oplus d_i \end{bmatrix} \qquad (5) \end{aligned}$$

Equation 5 is then mapped to 2 levels of 4-input LUTs of an FPGA shown in Fig. 2(a). The resource sharing amongst 4 rows of the state column where $i \in \{0, 2, 5, 6, 7\}$ is illustrated in Fig. 2(a) where each LUT implements the 4-input XOR function.

Next, the *MixColumns* transformation is expanded and rearranged as shown in equation 6 for the second group where a bit position $j \in \{1, 3, 4\}$. This allows $2a_j \oplus b_j \oplus c_j$ and $a_j \oplus 2c_j \oplus d_j$ to be shared between the (a'_j, d'_j) and (b'_j, c'_j) output bits of the state column, respectively. Fig. 2(b) shows equation 6 mapped to 2 levels of 4-input LUTs for the second group of bits.

There are totally 43 4-input XORs in the proposed *MixColumns* implementation. The 4-input LUTs generating the 4-input XORs had to be manually instantiated to produce the desired LUT schematic in Fig. 2 after synthesis.

$$\begin{aligned} result_j &= \begin{bmatrix} a'_j \\ b'_j \\ c'_j \\ d'_j \end{bmatrix} \\ &= \begin{bmatrix} 2a_j & \oplus & 3b_j & \oplus & c_j & \oplus & d_j \\ a_j & \oplus & 2b_j & \oplus & 3c_j & \oplus & d_j \\ a_j & \oplus & b_j & \oplus & 2c_j & \oplus & 3d_j \\ 3a_j & \oplus & b_j & \oplus & c_j & \oplus & 2d_j \end{bmatrix} \\ &= \begin{bmatrix} 2a_j \oplus b_j \oplus c_j \\ a_j \oplus 2c_j \oplus d_j \\ a_j \oplus 2c_j \oplus d_j \\ 2a_j \oplus b_j \oplus c_j \end{bmatrix} \oplus \begin{bmatrix} 2b_j \oplus d_j \\ 2b_j \oplus c_j \\ b_j \oplus 2d_j \\ a_j \oplus 2d_j \end{bmatrix} \\ &= \begin{bmatrix} a_{j-1} \oplus a_7 \oplus b_j \oplus c_j \\ a_j \oplus c_{j-1} \oplus c_7 \oplus d_j \\ a_j \oplus c_{j-1} \oplus c_7 \oplus d_j \\ a_{j-1} \oplus a_7 \oplus b_j \oplus c_j \end{bmatrix} \oplus \begin{bmatrix} b_{j-1} \oplus b_7 \oplus d_j \\ b_{j-1} \oplus b_7 \oplus c_j \\ b_j \oplus d_{j-1} \oplus d_7 \\ a_j \oplus d_{j-1} \oplus d_7 \end{bmatrix} \qquad (6) \end{aligned}$$

The proposed technique is designed for encryption in this research. Thus it can be beneficial, with respect to hardware

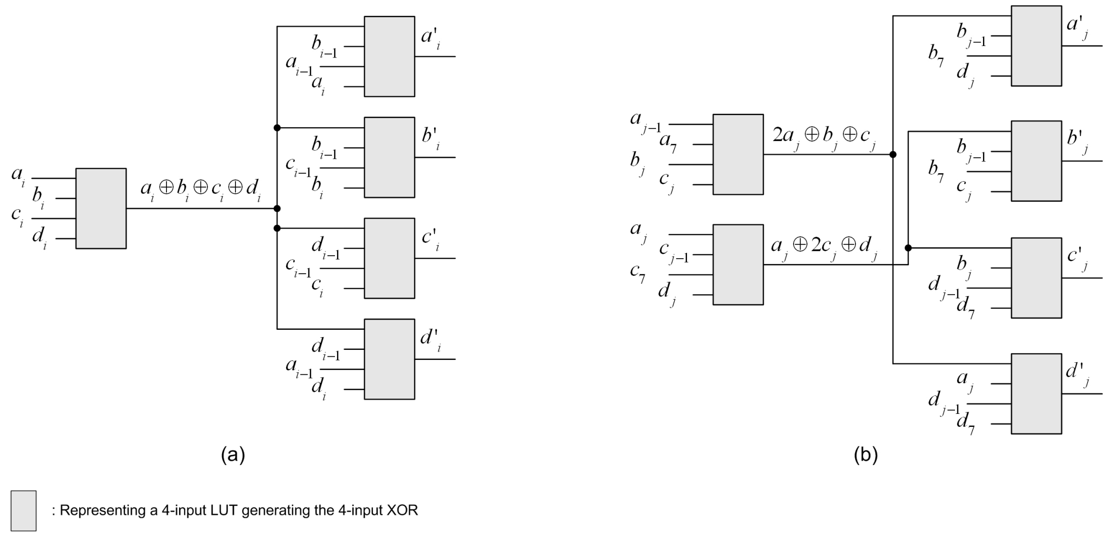

Figure 2. Proposed *MixColumns*: (a) bit position $i \in \{0,2,5,6,7\}$; (b) bit position $j \in \{1,3,4\}$

resource usage, to the modes that do not use the decryption function. Currently, the modes recommended by NIST [2] benefiting from the proposed *MixColumns* are:

- confidentiality modes
  - Cipher FeedBack (CFB)
  - Output FeedBack (OFB)
  - Counter (CTR)
- authenticated encryption modes
  - Counter with Cipher block chaining-Message authentication code (CCM)
  - Galois/Counter Mode (GCM)
- authentication mode
  - Cipher-based MAC (CMAC)

IV. Previous research

Rearrangement of the *MixColumns* with respect to the structure of an FPGA can result in a better optimized design in terms of utilizing hardware resources. Significant research [6], [7], [8], [9], [10] has been done on resource sharing between *MixColumns* and *InvMixColumns*, which is the inverse *MixColumns* in the decryption function [5], however there is limited work on optimizing the *MixColumns* transformation on its own on an FPGA. This is important since several modes (e.g., the NIST approved modes mentioned in section III) do not need the decryption function. In [11], [12], researchers suggested equation 7 where a bit position $m \in \{0,1,...,7\}$. In the left column of equation 7, two bits are XORed and the $xtime()$ is then applied to the XORed result [11], [12].

$$\begin{aligned} result_m &= \begin{bmatrix} a'_m \\ b'_m \\ c'_m \\ d'_m \end{bmatrix} \\ &= \begin{bmatrix} 2\,(a_m \oplus b_m) \\ 2\,(c_m \oplus b_m) \\ 2\,(c_m \oplus d_m) \\ 2\,(a_m \oplus d_m) \end{bmatrix} \oplus \begin{bmatrix} b_m \oplus c_m \oplus d_m \\ a_m \oplus c_m \oplus d_m \\ a_m \oplus b_m \oplus d_m \\ a_m \oplus b_m \oplus c_m \end{bmatrix} \end{aligned} \quad (7)$$

$$\begin{aligned} result_m &= \begin{bmatrix} a'_m \\ b'_m \\ c'_m \\ d'_m \end{bmatrix} \\ &= \begin{bmatrix} a_m\oplus b_m\oplus c_m\oplus d_m \\ a_m\oplus b_m\oplus c_m\oplus d_m \\ a_m\oplus b_m\oplus c_m\oplus d_m \\ a_m\oplus b_m\oplus c_m\oplus d_m \end{bmatrix} \oplus \begin{bmatrix} 2(a_m\oplus b_m) \\ 2(c_m\oplus b_m) \\ 2(c_m\oplus d_m) \\ 2(a_m\oplus d_m) \end{bmatrix} \oplus \begin{bmatrix} a_m \\ b_m \\ c_m \\ d_m \end{bmatrix} \end{aligned} \quad (8)$$

In [13], researchers suggested the original *MixColumns* shown in equation 2. In this approach, $xtime(z)$ and $xtime(z) + z$ are computed for each 8-bit element of the state column. There is also equation 8 used for *MixColumns* architecture [8], [7]. These designs were implemented to compare the experimental results on the FPGA in section V.

Another approach, that is used in software implementations as well, is to combine *SubBytes* and *MixColumns* transformations in one memory [14], [15], [16], [17], [18]. Compared to other methods, this technique requires the largest memory size. For instance, in an implementation with a 128-bit data path, it requires a memory size of 128 $Kbit$.

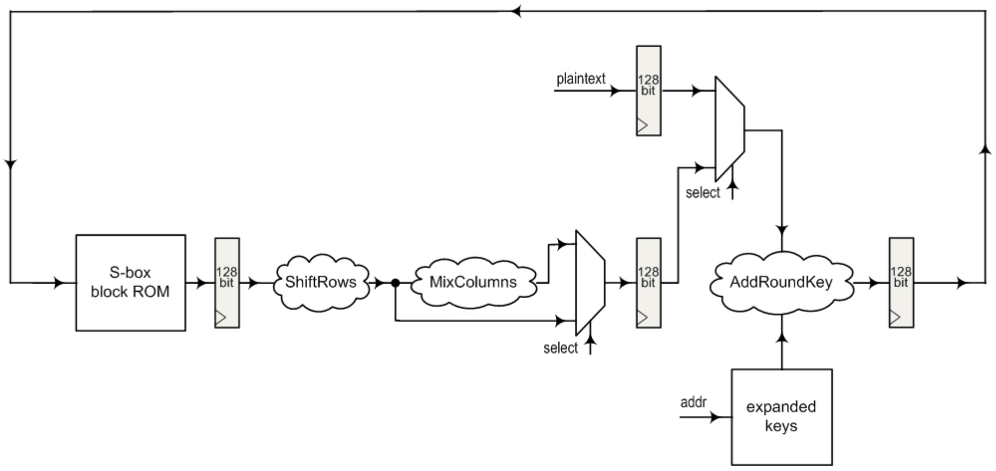

Figure 3. AES encryption data path

This memory size is 4 times larger than that of the 16 parallel S-boxes (32 $Kbit$).

V. Experimental results

The experimental results use the Xilinx Virtex 4 FPGA (target device xc4vlx15-10ff668) and Altera Cyclone (target device EP1C3T144C6). These FPGAs use 4-input LUTs as function generators. The tools used are as follows, ISE 10.1.03 and XPower Analyzer for Xilinx synthesis/implmentation and power estimation, Quaruts II 9.0 for Altera compilation, and ModelSim XE III 6.3c for testing and verification. All designs were synthesized in tables I and II with the synthesis objective set to minimum area. Manual instantiation was used to prevent further modification by the tool and the netlist was verified after place and route. The following two subsections will provide the experimental results after place and route for the *MixColumns* and AES encryption implementations.

A. MixColumns Implementation

Table I shows the number of LUTs with the corresponding LUT savings for the *MixColumns* implementations. The input and output are 32-bit columns of the state. As it can be observed in table I, improvements are more significant in case of the Xilinx synthesizer than Altera synthesizer.

B. AES Encryption implementation

The 128-bit data path shown in Fig. 3 is used for the AES encryption. Since most of the state of the art FPGAs contain block memories, storing the *SubBytes* values in 32 $Kbit$ of block ROM (shown as S-box block ROM in Fig. 3) is a common design choice. For fair comparisons, the proposed *MixColumns* design, [11], [12], [13], [8], and [7] are implemented in the AES data path depicted in Fig. 3. All the *MixColumns* implementations have two levels of LUTs in the data path. The *AddRoundKey*, shown in Fig. 3, has 1 level of LUTs in the logic. The experimental results of the AES encryption implementation are shown in table II. All the implementations use 532 flip flops and 32 $Kbit$ of block ROM in total. The static power consumption is 265 mW.

Due to routing, there are slight variations in the maximum clock frequency, f_{max}. It should also be noted that about 50% and 30% of the critical path delay are related to delays of routing and clock to output of block ROM. Therefore delays of LUTs have much less impact on the critical path delay. Dynamic power consumption is measured at 100 MHz in table II. There is significant hardware resource savings in terms of slices (containing 2 LUTs) and LUTs. As it is shown in table II, the proposed architecture uses 10.77% and 18.30% fewer slices and LUTs, respectively, than [11], [12]. Compared to [13], the proposed implementation reduces the slice and LUT usage by 7.45% and 12.46%, respectively. There is also 7.45% and 14.40% fewer slices and LUTs compared to [8], [7].

VI. Discussion and conclusion

In this research, an FPGA design of the *MixColumns* transformation is proposed which uses less hardware (slices and LUTs) compared to previous research.

Due to more efficient resource sharing, the proposed design for the *MixColumns* transformation provides the smallest hardware usage on an FPGA with 4-input LUTs. Overall, the AES encryption implementation with the proposed *MixColumns* architecture reduces usage of hardware resources i.e. slices and LUTs by up to 10% and 18%, respectively.

This research can be highly advantageous for efficient utilization of hardware resources on FPGAs in modes using encryption of AES. The proposed design can be used to provide security services such as confidentiality or authentication by these modes.

Table I
EXPERIMENTAL RESULTS OF *MixColumns* IMPLEMENTATIONS ON FPGA

	# of LUTs on Virtex4	% of LUT savings on Virtex4	# of LUTs on Cyclone	% of LUT savings on Cyclone
Proposed *MixColumns*	43	-	43	-
Design [11], [12]	56	23.21	51	15.68
Design [13]	54	20.37	48	10.41
Design [8], [7]	55	21.28	51	15.68

Table II
EXPERIMENTAL RESULTS OF AES ENCRYPTION IMPLEMENTATIONS ON FPGA

	# of slices	% of slice savings	# of LUTs	% of LUT savings	f_{max} in MHz	Block ROM size in $Kbit$	Dynamic power in mW
Proposed *MixColumns*	298	-	309	-	167.75	32	708.68
Design [11], [12]	334	10.77	377	18.30	166.22	32	712.48
Design [13]	322	7.45	353	12.46	171.85	32	709.92
Design [8], [7]	322	7.45	361	14.40	166.66	32	712.57

ACKNOWLEDGMENT

Reouven Elbaz is supported by the Postdoctoral Research Fellowship (PDRF 2009-10) of the Government of Canada.

REFERENCES

[1] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: Design challenges," *Trans. on Embedded Computing Sys.*, vol. 3, no. 3, pp. 461–491, 2004.

[2] NIST, "Recommendation for key management part 1," March 2007.

[3] ——, "Current modes," February 2008.

[4] D. Stinson, *Cryptography: Theory and Practice, Third Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, November 2005. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&amp;path=ASIN/1584885084

[5] FIPS197, "Advanced encryption standard," November 2001.

[6] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact rijndael hardware architecture with s-box optimization," in *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*. London, UK: Springer-Verlag, 2001, pp. 239–254.

[7] P. Chodowiec and K. Gaj, "Very compact fpga implementation of the aes algorithm," in *CHES*, 2003, pp. 319–333.

[8] V. Fischer, M. Drutarovsky, P. Chodowiec, and F. Gramain, "Invmixcolumn decomposition and multilevel resource sharing in aes implementations," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, no. 8, pp. 989–992, Aug. 2005.

[9] N. Chen and Z. Yan, "Compact designs of mixcolumns and subbytes using a novel common subexpression elimination algorithm," in *ISCAS*, 2008, pp. 1584–1587.

[10] A. Satoh and S. Morioka, "Unified hardware architecture for 128-bit block ciphers aes and camellia," in *CHES*, 2003, pp. 304–318.

[11] T. Good and M. Benaissa, "Pipelined aes on fpga with support for feedback modes (in a multi-channel environment)," *Information Security, IET*, vol. 1, no. 1, pp. 1–10, March 2007.

[12] X. Zhang and K. Parhi, "High-speed vlsi architectures for the aes algorithm," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, no. 9, pp. 957–967, 2004.

[13] H. Li and Z. Friggstad, "An efficient architecture for the aes mix columns operation," *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 4637–4640 Vol. 5, May 2005.

[14] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat, "Compact and Efficient Encryption/Decryption Module for FPGA Implementation of AES Rijndael Very Well Suited for Small Embedd," in *ITCC 2004, special session on embedded cryptographic hardware*. IEEE Computer Society, 2004, pp. 583–587.

[15] P. Bulens, F.-X. Standaert, J.-J. Quisquater, P. Pellegrin, and G. Rouvroy, "Implementation of the AES-128 on Virtex-5 FPGAs," in *Progress in Cryptology - AfricaCrypt 2008*. Springer, 2008, pp. 16 – 26.

[16] J. Daemen and V. Rijmen, *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

[17] S. Tillich, J. Großschädl, and A. Szekely, "An instructions set extension for fast and memory-efficient aes implementation," in *Communications and Multimedia Security*, ser. Lecture Notes in Computer Science, S. K. u. A. U. Jana Dittmann, Ed. Springer, 2005, pp. 11 – 21.

[18] M. McLoone and J. V. McCanny, "Rijndael fpga implementations utilising look-up tables," *J. VLSI Signal Process. Syst.*, vol. 34, no. 3, pp. 261–275, 2003.