Regular paper

Implementation of the block cipher Rijndael using Altera FPGA

Piotr Mroczkowski

Abstract — A short description of the block cipher Rijndael is presented. Hardware implementation by means of the FPGA (field programmable gate array) technology is evaluated. Implementation results compared with other hardware implementations are summarized.

Keywords — block cipher, Rijndael, Altera FPGA.

1. Introduction

It has been announced recently that the cryptographic algorithm named Rijndael is the winner of the Advanced Encryption Standard competition. This international contest was organized by the National Institute of Standards and Technology. In September 1997, NIST opened a formal call for algorithms. A group of fifteen AES candidate algorithms were announced in August 1998. Next, all algorithms were subject to assessment process performed by various groups of cryptographic researchers throughout the world. In August 2000, NIST selected five algorithms: Mars, RC6, Rijndael, Serpent, Twofish as the final competitors. These algorithms were subject to further analysis prior to the selection of the best algorithm for the AES. Finally, on October 2, 2000, NIST announced that the Rijndael algorithm was the winner.

The primary criteria chosen by NIST to appoint the winner for the AES included security, efficiency in hardware and software, flexibility. The most important measure was the resistance against all known and unknown attacks, but after a thorough research it appeared that all algorithms considered in second phase were robust. The results of software implementation were also comparable. Under such circumstances the efficiency of hardware implementation seemed to be an important factor of the overall score.

Hardware implementations are designed and coded in hardware description language (for example AHDL – Altera hardware description language) and may be done using the FPGA devices. Altera's devices (FLEX 10K) consist of thousands of universal building blocks (called macrocells), dedicated memory blocks (called embedded array blocks – EABs) connected by means of programmable interconnectors. Block ciphers seem to fit extremely well the characteristics of the FPGAs. The fine-granularity of FPGA matches very well the operations required by block algorithms such as bit-permutations, bit-substitution, look-up table reads and boolean functions. The EABs are suitable for implementing large S-boxes (such as 8×8 S-boxes used in Rijndael).

2. Description of the Rijndael cipher

The Rijndael algorithm, which falls into the block cipher category, has been designed by Joan Daemen and Vincent Rijmen and its specification is given in [1].

The length of the block and the length of the key can be independently specified to 128, 192 and 256 bits. The structure of the variant of encryption algorithm with 128-bit length of the block and the key is presented in Fig. 1.

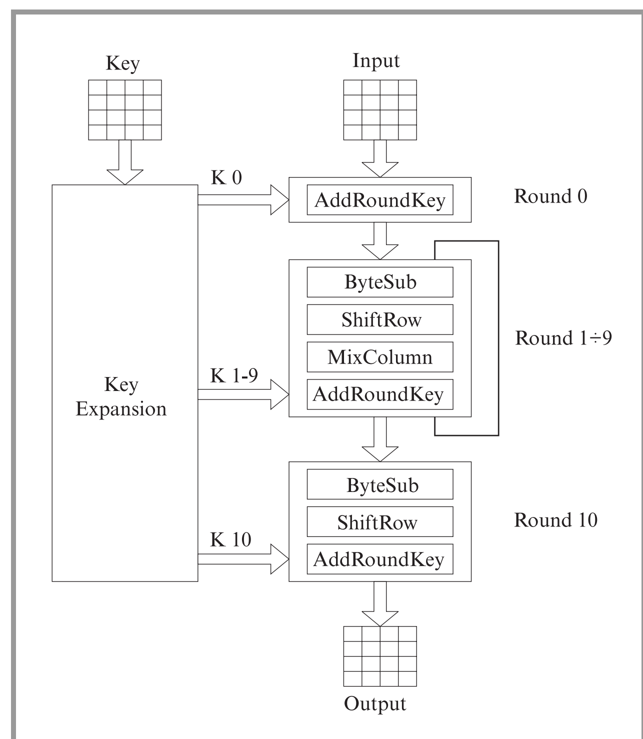

Fig. 1. The encryption algorithm.

An input block of data and the intermediate cipher results are represented by a square matrix with 4×4 of byte dimension (so-called the state). The state is presented in Fig. 2.

The cipher input bytes are mapped onto the state bytes in the order $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{1,0}$, $a_{1,1}$, $a_{1,2}$, $a_{1,3}$, ... At the end of the cipher operation, the cipher output is extracted from the state by taking the state bytes in the same order.

Every round except the initial (Round 0) and final (Round 10) ones consists of four transformations:

1. ByteSub – a single nonlinear transformation, which is applied to each byte of the data.

2. ShiftRow – which cyclically reorders the bytes of row.
3. MixColumn – a linear transformation applied to columns of the matrix.
4. AddRoundKey – which mixes the round key and the intermediate data.

Prior to the first round the transformation AddRoundKey is performed by using the main key as the round key (Round 0). Next, nine basic rounds (Round $1 \div 9$) consisted of all four transformations are performed. In the final round (Round 10) the transformation MixColumn is skipped.
The decryption algorithm with the 128-bit data and key option is presented in Fig. 3.

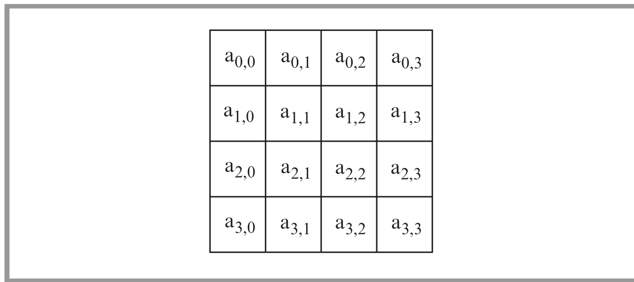

Fig. 2. Example of the state.

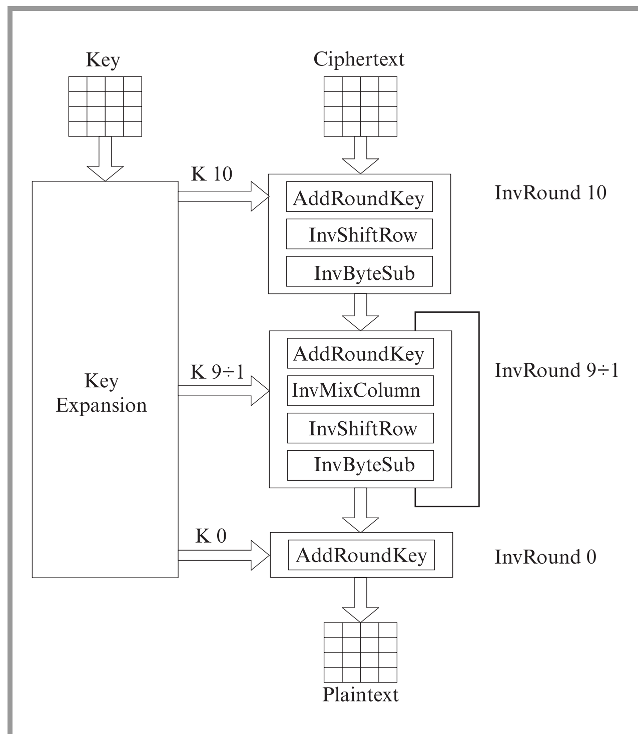

Fig. 3. The decryption algorithm.

In the first step of the decryption algorithm the inversion of the final encryption round is performed (InvRound 10), next nine inversions of the basic encryption rounds (InvRound $9 \div 1$), and in the last step the transformation AddRoundKey is calculated (InvRound 0). In the decryption round all transformations of the encryption round are inverted in the reverse order.

2.1. ByteSub (InvByteSub) transformation

The ByteSub transformation is the byte substitution, operating on each of the state bytes independently. Each byte is considered as representing coefficients of a polynomial of degree less then 8 over GF(2^8). Firstly, we calculate the inversion of this polynomial modulo $(x^8+x^4+x^3+x+1)$, then we multiply the result by a fixed matrix and add a fixed polynomial (an affine transformation). The affine transformation is defined by:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

The inversion and affine transformations create a substitution table (S-box).
The InvByteSub transformation is obtained by the inverse, affine mapping followed by taking the inversion over GF(2^8).
The inverse affine transformation and inversion create an inverse substitution table (InvS-box).
The effect of the ByteSub (InvByteSub) transformation on the state is presented in Fig. 4.

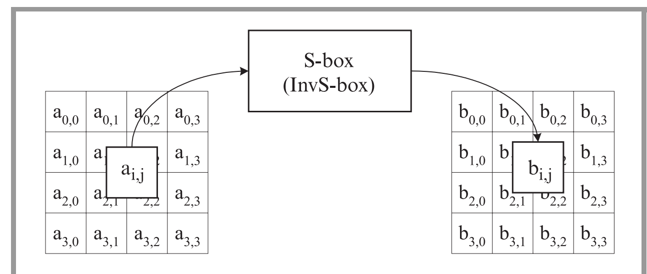

Fig. 4. The ByteSub (InvByteSub) transformation.

2.2. ShiftRow (InvShiftRow) transformation

In the ShiftRow transformation the bytes in the rows of the state are cyclically shifted over different offsets according to the following rule. Bytes in the first row are not shifted, in the second are shifted by 1 byte, in the third by 2 bytes, and in the last one by 3 bytes to the left (Fig. 5).
In the InvShiftRow, bytes in the first row are not shifted, in the second are shifted by 3 bytes, in the third over 2 bytes and in the last one by 1 byte to the left (Fig. 6).

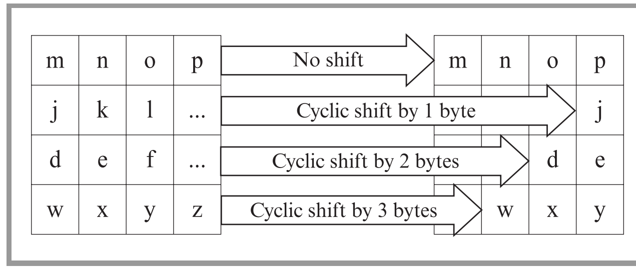

Fig. 5. The ShiftRow transformation.

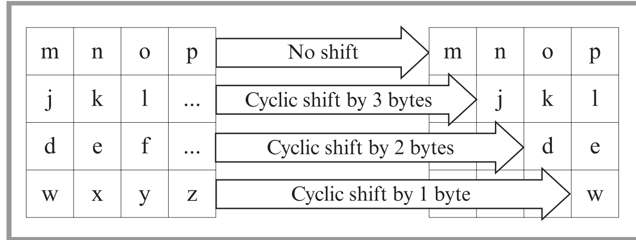

Fig. 6. The InvShiftRow transformation.

2.3. MixColumn (InvMixColumn) transformation

In the MixColumn transformation, the bytes located in the columns of the state, are considered as coefficients of polynomials of degree less then 4 over GF(2^8) field and multiplied modulo (x^4+1) with a fixed polynomial $c(x)$ = '03'x^3 + '01'x^2 + '01'x + '02', where '03' denotes a hexadecimal value. Figure 7 illustrates the effect of the MixColumn transformation on the state.

In the InvMixColumn transformation, the polynomials of degree less then 4 over GF(2^8), which coefficients are the elements in the columns of the state, are multiplied modulo (x^4+1) by a fixed polynomial $d(x)$ = '0B'x^3 + '0D'x^2 + '09'x + '0E', where '0B', '0D,'09', '0E' denote hexadecimal values. Figure 7 illustrates the effect of the InvMixColumn transformation on the state.

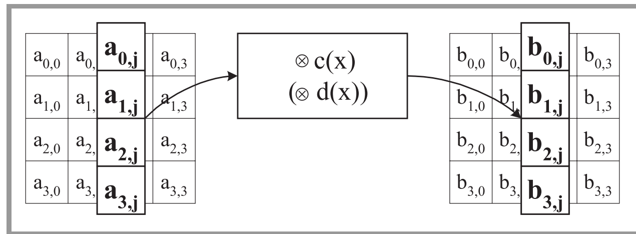

Fig. 7. The MixColumn (InvMixColumn) transformation.

2.4. The AddRoundKey transformation

In the AddRoundKey mapping, the 128-bit round key, which is derived from the key by the KeyExpansion algorithm, is bytewise XORed with the state.

2.5. Key Expansion

The round key (subkey) of each round is derived from the main key using the KeyExpansion algorithm [1]. The encryption (decryption) algorithm needs eleven 128-bit subkeys, which are denoted K0 ÷ K10 (the first subkey K0 is the main key). The round keys are derived as follows: let us denote the bytes of the expanded key by $B_0, B_1, B_2, \ldots, B_{175}$ where the main key is $B_0, B_1, B_2, \ldots, B_{15}$ (K0 = $B_0, B_1, \ldots, B_{15}$). Then the expanded key is derived from the formulae:

$$B_n=\begin{cases} B_{n-16}\oplus \mathrm{SubByte}(B_{n-3})\oplus RC\left[\frac{n}{16}\right], & \text{if (n mod 16)} = 0;\\ B_{n-16}\oplus \mathrm{SubByte}(B_{n-3}), & \text{if (n mod 16)} \in \{1,2\};\\ B_{n-16}\oplus \mathrm{SubByte}(B_{n-7}), & \text{if (n mod 16)} = 3;\\ B_{n-16}\oplus B_{n-4}, & \text{otherwise;}\end{cases}$$

where:
SubByte(B) is a function that returns a byte, which is the result of applying the ByteSub transformation for one byte;
$RC[i]$ – an element over GF(2^8), which represents round constants and is defined by:
$RC[1]$ = '01';
$RC[i] = x\cdot(RC[i-1]) = x^{(i-1)}$.
Round keys are taken from the expanded key in the following way: the first subkey consists of the first 16 bytes (K0 = $B_0 \ldots B_{15}$), the second one of the following 16 bytes (K1 = $B_{16} \ldots B_{31}$), and so on.

3. Field programmable gate array implementation of the Rijndael cipher

3.1. Encryption and decryption units

The encryption algorithm implementation is designed to perform the subkey generation and the round calculations in parallel. Firstly, the initial (Round 0) round (input data

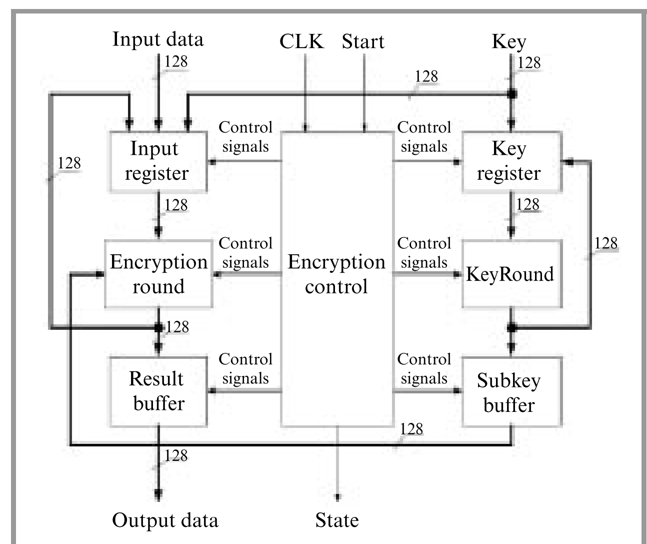

Fig. 8. The encryption unit.

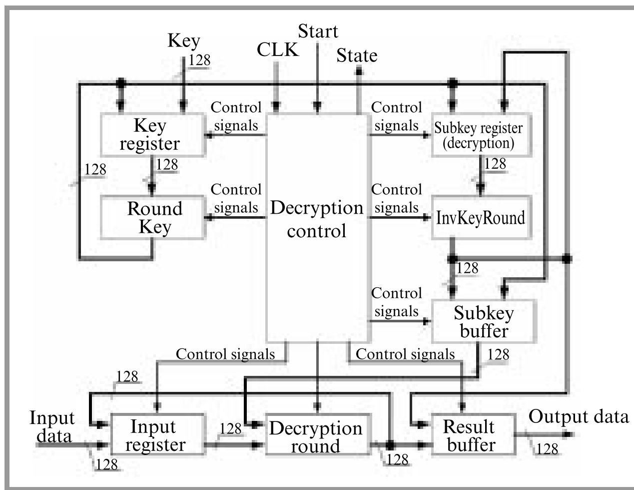

Fig. 9. The decryption unit.

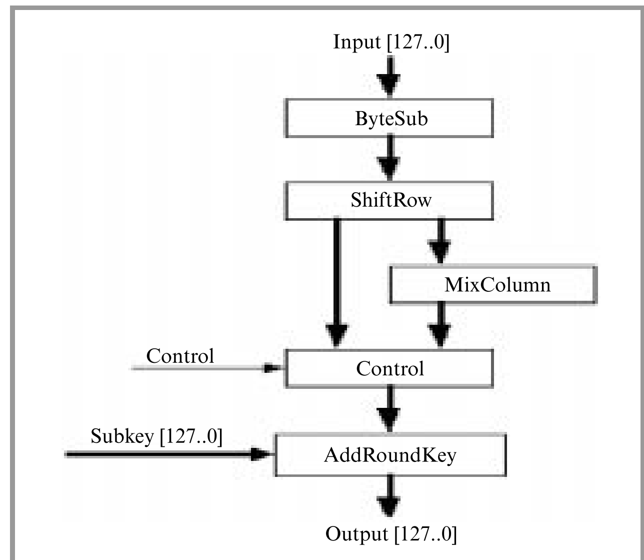

Fig. 10. The encryption round.

EXOR-ed with the main key) is performed and the subkey for the round number one is calculated. Then the round transformations are performed and the subkey for next round is calculated. The advantage of such design is that there is no need for storing the subkeys; they are calculated on the fly and discarded after using. The encryption unit is presented in Fig. 8.

The decryption algorithm is implemented in the similar way. Firstly, the tenth subkey (K10) is calculated, then calculations of the inversion of the final round (InvRound 10) and the subkey generation for the next decryption round (K9) are performed simultaneously. The decryption unit is presented in Fig. 9.

3.2. Round transformation implementations

Basic encryption round consists of the following transformations: ByteSub, ShiftRow, MixColumn, AddRoundKey (the final encryption round does not contain the MixColumn transformation). The round implementation is designed to work both in basic round mode and in the final round mode. The encryption round scheme is presented in Fig. 10.

The basic decryption round (which is the inversion of the basic encryption round) consists of the following transformations: AddRoundKey, InvMixColumn, InvShiftRow, InvByteSub. The first decryption round (InvRound 10 – which is the inversion of the final encryption round) does not contain the InvMixColumn transformation. The decryption round implementation is designed in similar way as the encryption round and its scheme is presented in Fig. 11.

The nonlinear ByteSub (InvByteSub, respectively) transformation contains 16 S-boxes (InvS-boxes, respectively), working in parallel. The 128-bit input block is divided into 16 bytes. Each byte forms the input data of the S-box (InvS-box, respectively). The byte outputs of all S-boxes

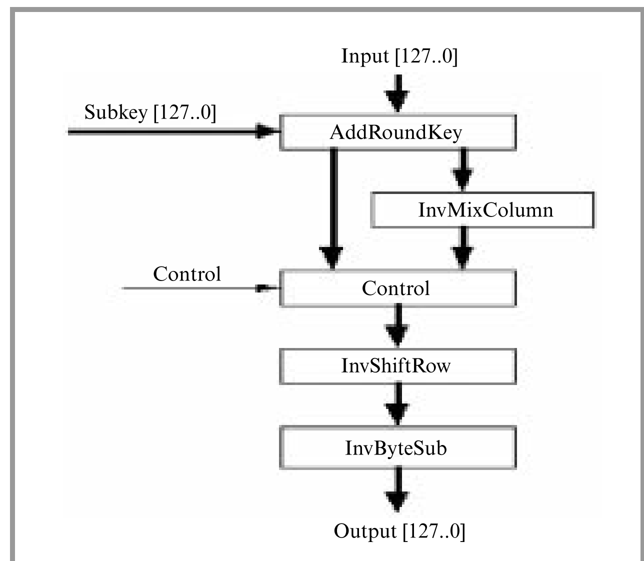

Fig. 11. The decryption round.

(InvS-boxes, respectively) are then concatenated and form the output of the ByteSub (InvByteSub, respectively) transformation.

The S-box transforms the input byte to the inverse byte by performing the arithmetic operation defined over the finite field GF(2^8) (the value '00' is mapped in '00') and then forms the input for the affine transformation. For inverse transformation, the process runs in the opposite direction. The S-box (InvS-box, respectively) was implemented by using the build-in EAB memory which emulate the ROM memory with the configuration of 256×8 bits. The implementation of the S-box needs one EAB block, i.e. 2048 bits. The access memory time in this implementation is approx. 18 ns.

In the ShiftRow (InvShiftRow, respectively) transformation, the 128-bit input block is divided into 16 bytes denoted as *Aij*[7..0], where *i,j* $\in \{0,1,2,3\}$. The bytes *Aij*[7..0] are the elements of the table representing the intermediate state of encrypted (or decrypted) block. The output of the ShiftRow (InvShiftRow) transformation is composed of the bytes *Bij*[7..0], where *i,j* $\in \{0,1,2,3\}$. The implementations of these transformations perform the byte shift, as shown in Figs. 12 and 13.

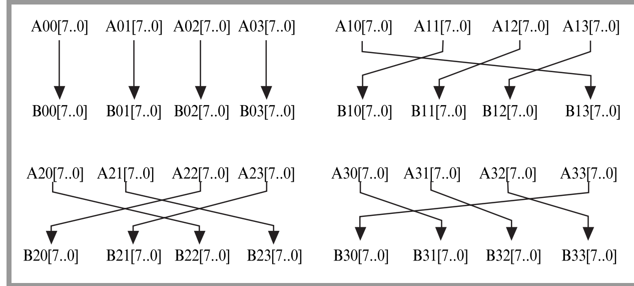

Fig. 12. The ShiftRow transformation.

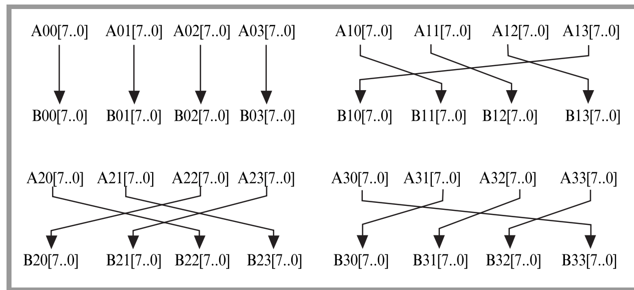

Fig. 13. The InvShiftRow transformation.

In the MixColumn (InvMixColumn, respectively) transformation, the 128-bit input block is divided into 16 bytes denoted as *Aij*[7..0], where *i,j* $\in \{0,1,2,3\}$, and the output bytes are denoted as *Bij*[7..0]. The bytes *Aij*[7..0], while the index j is fixed and $i \in \{0,1,2,3\}$, correspond to the column of the table representing the intermediate state of the transformed block and they are viewed as the coefficients of polynomial over the field GF(2^8) of degree smaller then four. This polynomial is multiplied by the fixed polynomial $c(x)$ = '03'x^3 + '01'x^2 + '01'x + '02' modulo (x^4+1). In the case of decryption the inverse polynomial $d(x)$ is used: $d(x)$ = '0B'x^3 + '0D'x^2 + '09'x + '0E'. The results of this modular multiplication form the column *Bij*[7..0] (index j is fixed and $i \in \{0,1,2,3\}$) of the transformed state. These transformations were implemented as bit-oriented EXOR operations.

The AddRoundKey transformations have the block text as the input value and EXOR-ed it with the value of the subkey of the given round.

The logical unit denoted as KeyRound (InvKeyRound, respectively) calculates the subkeys of subsequent rounds of the encryption (decryption, respectively) algorithm. It is controlled by signals from the logic unit EncryptionControl (DecryptionControl, respectively) and input values of the subkey. The functional description of the logical units KeyRound and InvKeyRound are depicted in Figs. 14 and 15, respectively.

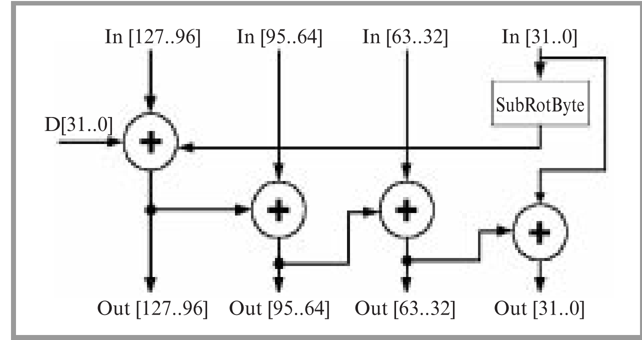

Fig. 14. The logical unit KeyRound.

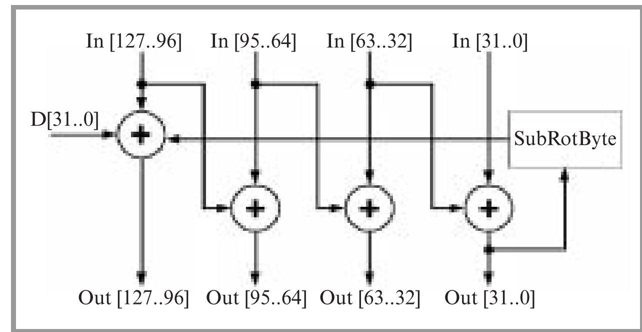

Fig. 15. The logical unit InvKeyRound.

The round constants D[31..0] (see Figs. 14 and 15) have values depending on the round number, as presented in Table 1.

Table 1
The values of the constants D[31..0]

The encryption round number	The decryption round number	D[31..0](hex)
1	10	01000000
2	9	02000000
3	8	04000000
4	7	08000000
5	6	10000000
6	5	20000000
7	4	40000000
8	3	80000000
9	2	1B000000
10	1	36000000

The logical unit SubRotByte has 32 inputs and 32 outputs. It performs two operations: RotByte and SubByte. The RotByte transformation is a cyclic bytes rotation in 32-bit word by one byte position to the left. The SubByte transformation consists of four parallel S-boxes applied to the 32-bit input word.

3.3. Implementation results

The encryption and decryption algorithms have been implemented in two separable chips denoted as EPF10K250AGC599-1 (this type of Altera chip has 20 blocks of the EAB memory which can be used to implement 256×8 bit ROM configuration). The results of logic circuit synthesis are given in Table 2.

Table 2
The logic circuit synthesis results for encryption and decryption process

The logical unit	Input pins	Output pins	Bidir pins	Memory [bit]	LCs
Encryption round	–	–	–	32768	388
KeyRound	–	–	–	8192	138
The other logic blocks	–	–	–	0	506
Encryption	**258**	**129**	**0**	**40960**	**1032**
Decryption round	–	–	–	32768	798
InvKeyRound	–	–	–	8192	139
RoundKey	–	–	–	0	1475
The other logic blocks	–	–	–	0	473
Decryption	**258**	**129**	**0**	**40960**	**2885**

It appears from the table that the decryption algorithm is more complicated than the encryption one (see also Figs. 8 and 9). In result, the logic circuit synthesis for the decryption unit requires over twice as many macrocells the encryption unit. It is worth to observe that before the main part of decryption process the tenth subkey should be calculated. The RoundKey unit performed this operation requires ca 1475 macrocells. Moreover, the decryption round needs twice as many macrocells as the encryption round because the polynomial $d(x)$ used in InvMixColumn transformation is more complicated then the polynomial $c(x)$ used in MixColumn transformation.
The speed of encryption and/or decryption implementation depends mainly on the frequency of the external clock applied to the chip. According to the characteristics the minimal clock cycle is 22 ns (45.45 MHz) for the encryption chip and 24 ns (41.66 MHz) for the decryption chip. The encryption or decryption operation is performed in 21 clock cycles. In the implementation of the Rijndael algorithm with 128-bit encrypted blocks and 128-bit key presented here, the speed of 268 Mbps for encryption and 248 Mbps for decryption has been achieved (Table 3).

Table 3
The speed of encryption and decryption process

Hardware implementation (Altera)	The encryption unit [Mbps]	The decryption unit [Mbps]
The process speed	268	248

Hardware implementations of the Rijndael, based on Altera FLEX FPGA devices were also developed by two other groups working independently: Microsonic [2] and GMU (George Mason University) [3]. The results of implementing Rijndael using FLEX 10K130E and FLEX 10K250A devices are summarized in Fig. 16.

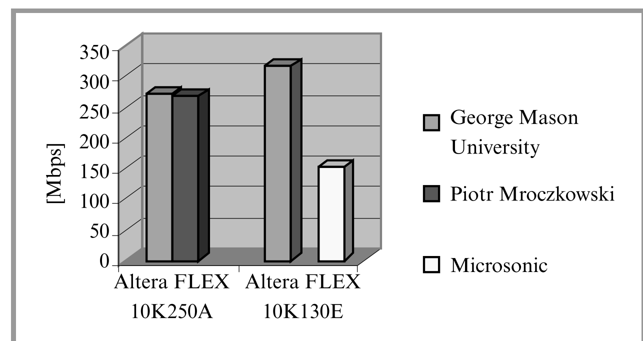

Fig. 16. Comparison of chosen hardware implementations of the Rijndael cipher.

The Figure shows similar performance achieved by GMU group and the Author. However, this is not the case when comparing performance of implementations using FLEX 10K130E devices achieved by Microsonic and GMU group.

4. Conclusions

The Rijndael cipher seems to be very suitable for hardware implementations using Altera FLEX 10K devices. Especially, the EABs fit very well for implementing large S-boxes, such as 8×8 S-boxes used in Rijndael. The achieved speed is about four times greater than for software implementations reported. The further progress can be possibly made with the pipeline architecture.

References

[1] J. Daemen and V. Rijmen, „AES proposal: Rijndael". Sept. 1999. [Online]. Available WWW: http://www.esat.kuleuven.ac.be/∼rijmen/rijndael/

[2] V. Fischer, „Realization of the round 2 AES candidates using Altera FPGA". March 2000. [Online]. Available WWW: http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html

[3] K. Gaj and P. Chodowiec, *Implementations of the AES Candidate Algorithms using FPGA Devices*. Technical Report. George Mason University, April 2000.

[4] J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti, and E. Roback, *Status Report on the First Round of the Development of the Advanced Encryption Standard*. NIST Report, August 1999.

[5] V. Rijmen, the private communications,
e-mail: vincent.rijmen@esat.kuleuven.ac.be.

Piotr Mroczkowski was born on May 20, 1975. He received the M.Sc. degree in computer systems and cryptology from Military Academy of Technology, Warsaw, Poland, in 2000. He is currently working in Military Communication Institute.

e-mail: mroczkow@wil.waw.pl pmrocz@wp.pl
Military Communication Institute
05-130 Zegrze Płd., Poland