Date: Wed, 10 May 2000 19:44:00 +0200
From: Piotr Mroczkowski <pmrocz@mamut.isi.wat.waw.pl>
Organization: WAT
X-Mailer: Mozilla 4.7 [en] (Win98; I)
X-Accept-Language: en
To: AESround2@nist.gov
Subject: the article - Rijndael

[See attached document]

Implementation of the block cipher Rijndael using Altera FPGA

Piotr Mroczkowski
Military University of Technology
Warsaw
Poland
pmrocz@mamut.isi.wat.waw.pl

1. Introduction.

The cipher Rijndael is one of the five finalists of the Advanced Encryption Standard. The algorithm has been designed by Joan Daemen and Vincent Rijmen and its specification is given in [1]. It is a block cipher. The length of the block and the length of the key can be specified to be 128, 192, 256 bits. In this paper we present the hardware implementation with 128-bit blocks and 128-bit keys, using FPGA (Field Programmable Gate Arrays). In this variant the cipher consists of 10 rounds.

2. The FPGA implementation of the cipher Rijndael.

2.1 The encryption and decryption units.

The encryption algorithm has been designed this way that the generation of the subkey and the round calculations can be parallely executed. In the first order it executes the round number zero (which is the EXOR input data with the main key) and it is calculated the subkey to the round number one. Then it is executed the round and it is calculated the subkey to the next round. The advantage of this design is the fact that we do not need to store the subkeys; they are currently calculated. The Figure 1 shows the encryption unit of the algorithm Rijndael.

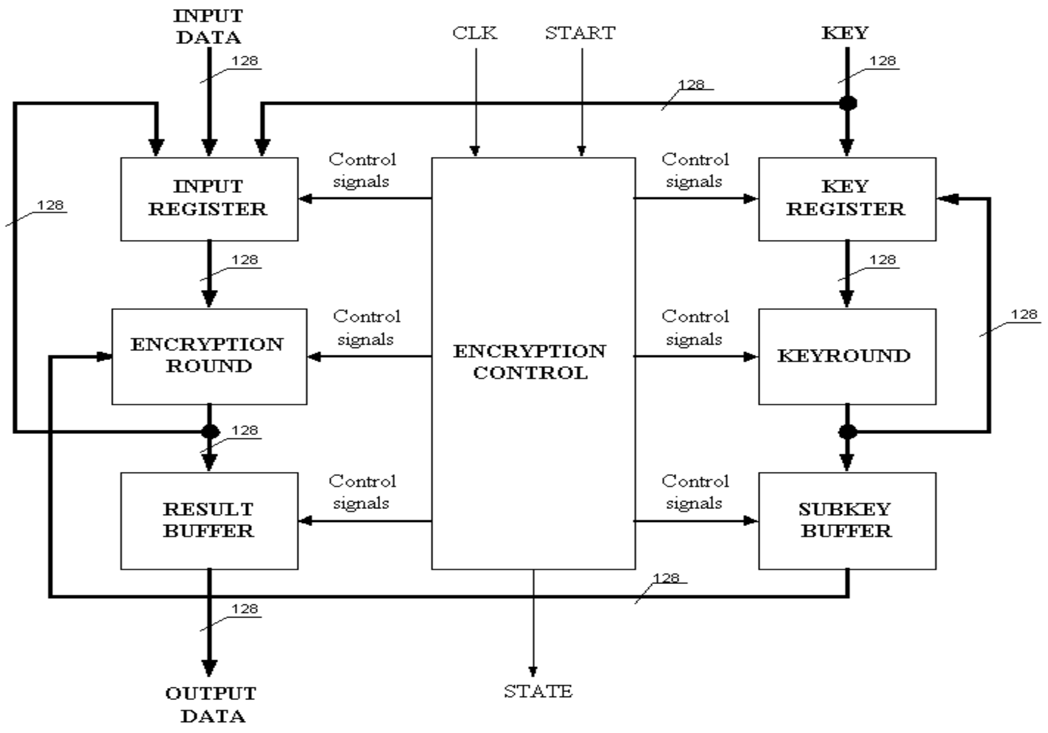


Figure 1. The encryption unit.

The decryption algorithm has been designed in the similar way. In the first order it is calculated the tenth subkey (the one witch has been used in the final round of the encryption round), then there are executed simultaneously the calculations of the inverse of the final round and the generation of the subkey for the next decryption round. The decryption unit is depicted in Figure 2.

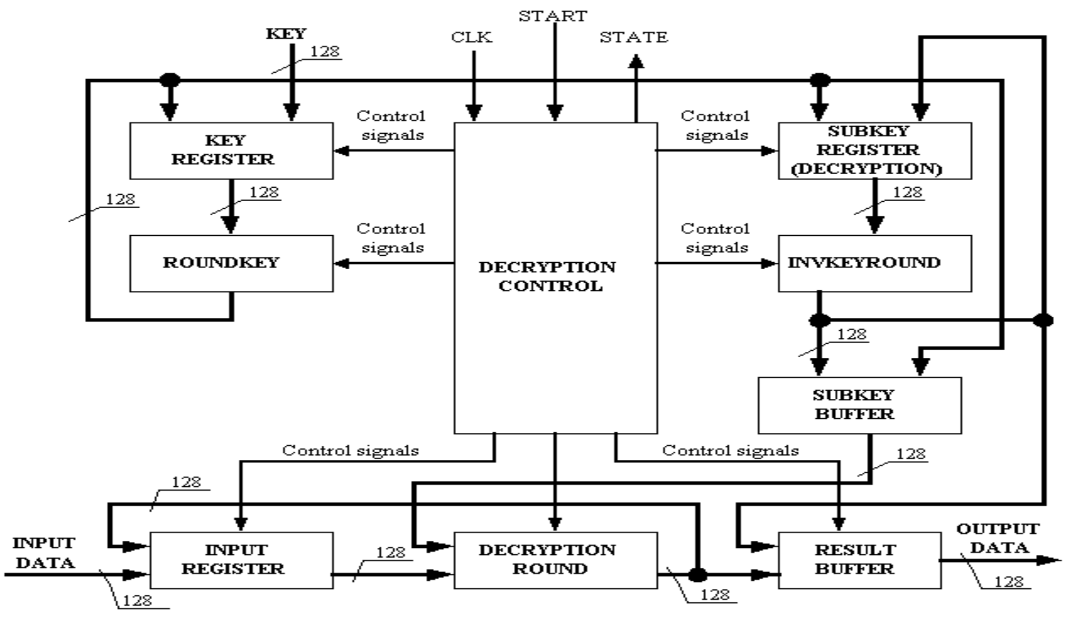
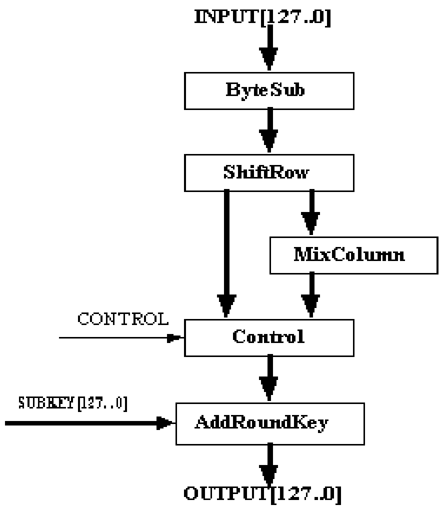


Figure 2. The decryption unit.

2.2 The implementations of the round transformations.

The basic encryption round contains the following transformations: ByteSub, ShiftRow, MixColumn, AddRoundKey and the final encryption round does not contain the MixColumn transformation. The implementation of the round has been designed this way that it can work as the basic round or as the final round. The unit of the encryption round is depicted in Figure 3.

The basic decryption round (which is the inverse of the basic encryption round) contains the following transformations: AddRoundKey, InvMixColumn, InvShiftRow, InvByteSub and the decryption round number one (which is the inverse of the final encryption round) does not contain the InvMixColumn transformation. The implementation of the decryption round has been designed in the similar way as the encryption one and its unit is depicted in Figure 4.



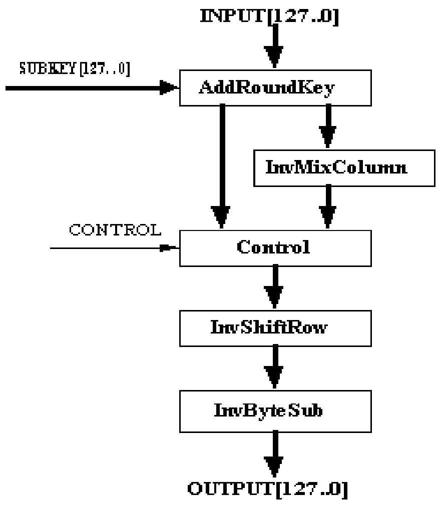


Fig. 3. The encryption round. **Fig. 4. The decryption round.**

The non-linear ByteSub (resp. InvByteSub) transformations contains 16 parallely working S-boxes (resp. inverse S-boxes). The 128-bit input block is divided into 16 bytes. Each byte states is the input to the S-box (resp. inverse S-box) and the output is also a byte. The outputs of all S-boxes (resp. inverse S-boxes) are concatenated to constitute the output of the ByteSub (resp. InvByteSub) transformation.

The S-box transforms the input byte to the inverse byte in the sense of the arithmetic in the finite field GF(2^8) (the zero byte is transformed to the zero byte) and then it is subjected to the affine transformation. The inverse S-box transforms first the input byte according to the inverse of this affine transformation and then the inversion in the field GF(2^8) is applied.
We have implemented the S-box (resp. inverse S-box) using the build in EAB (Embedded Array Block) memory which emulate the ROM memory with the configuration of 256×8 bits. The realisation of the S-box needs one EAB block, i.e. 2048 bits. The access time to the memory implemented this way is about 18 ns.
In the ShiftRow (resp. InvShiftRow) transformation the 128 bit input block is divided into 16 bytes denoted 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 realisations of these transformations shift the byte according to the units depicted in Figures 5 and 6.

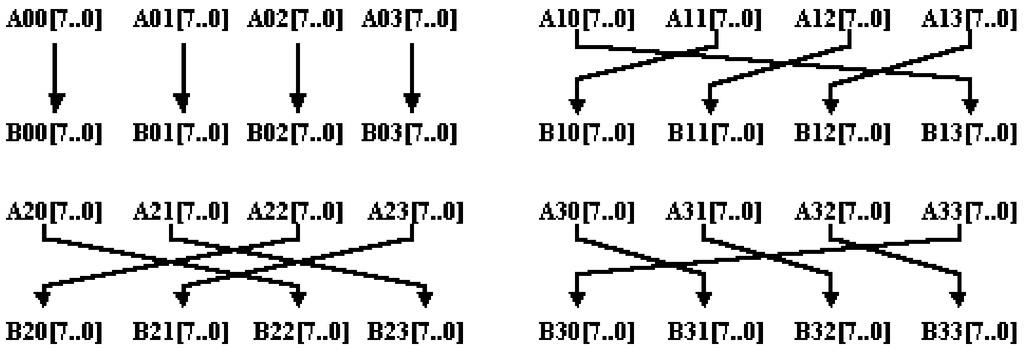


Figure 5. The ShiftRow transformation.

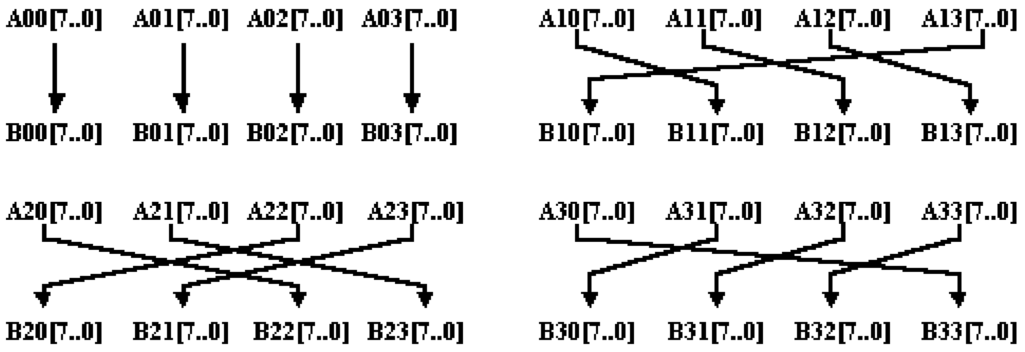


Figure 6. The InvShiftRow transformation.

In the MixColumn (resp. InvMixColumn) transformation the 128 bit input block is divided into 16 bytes denoted Aij[7..0], where $i,j \in \{0,1,2,3\}$, and the output bytes are denoted Bij[7..0]. The bytes Aij[7..0], when the index i is fixed, 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 three. This polynomial is multiplied by the fixed polynomial c(x) (given in the specification [1]) modulo the polynomial x^4+1. In the case of decryption the inverse polynomial d(x) is used; $c(x)d(x) \equiv 1 \bmod (x^4+1)$. The result of this modular multiplication represents the column Bij[7..0] (the same index i) of the transformed state. The implementation of these transformations can be realised as the bit oriented EXOR operations.
The AddRoundKey transformations take the block text and EXOR it with the subkey of the given round.
The logical block denoted KeyRound (resp. InvKeyRound) calculates the subkeys of succeeding rounds of the encryption (resp. decryption) algorithm. It works according to the signals from the logic block EncryptionControl (resp. DecryptionControl) and the values of the input subkey. The functional description of the logical block KeyRound is depicted in Figure 7 and the logical block InvKeyRound is depicted in Figure 8.

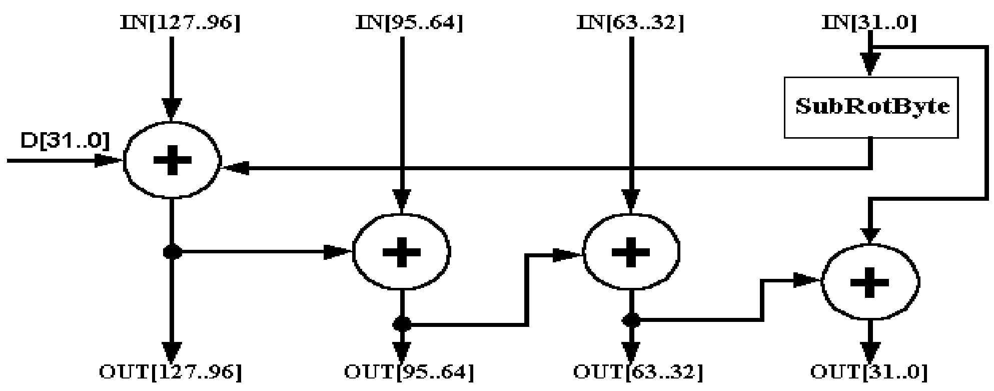


Figure 7. The logical block KeyRound.

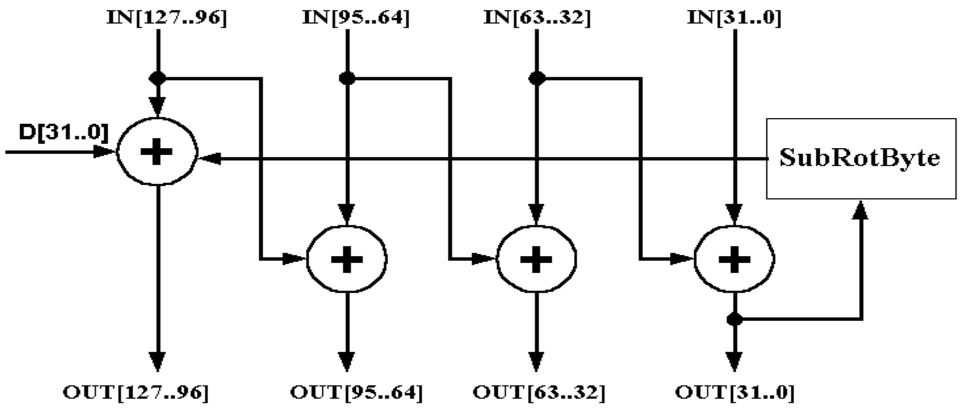


Figure 8. The logical block InvKeyRound.

The round constant D[31..0] takes the values depending on the round number and the values are given in Table1.

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

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

The logical block SubRotByte has 32 inputs and 32 outputs. It realises two operations: RotByte and SubByte. The RotByte transformation is a cyclic bytes rotation in 31 bit word on one byte position to the left. The SubByte transformation is an application of four parallel S-boxes to the input 32-bit word.

2.3 Result of implementation.

The encryption and decryption algorithm have been implemented in two separable chips denoted EPF10K250AGC599-1 (the chips of series EPF10K250A have 20 block of the EAB memory which can be used to implement the ROM in configuration of 256×8 bits) of the firm Altera. The Table 2 given the results of logic synthesis:

The logical block	Input Pins	Output Pins	Bidir Pins	Memory Bits	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
The other logic blocks	-	-	-	0	473
Decryption	**258**	**129**	**0**	**40960**	**2885**

Table 2. The results of the logic synthesis of encryption and decryption.

The frequency of the used external clock decides about the speed of encryption and decryption, but it must be adapted to possibilities of chips realising the algorithm. The minimal period of clock for the encryption chip is 22 ns. (45,45 kHz), for the decryption chip 24 ns. (41, 46 kHz). The encryption or decryption is performed in 21 clock cycles. I have achieved the speed of 268 Mb/s for encryption and 248 Mb/s for decryption for the Rijndael algorithm with 128-bit encrypted blocks and 128-bit key (Table 3).

Implementation	The encryption speed	The decryption speed
The software implementation (ANSI C)	27Mb/s	27Mb/s
The software implementation (Visual C++)	70,5Mb/s	70,5Mb/s
The hardware implementation (Altera)	**268Mb/s**	**248Mb/s**

Table 3. The speed of encryption and decryption.

3. Conclusions.

The cipher Rijndael seems to be very suitable for hardware implementations. The achieved speed is about four times the one reported in software. The further program can be eventually obtained by introducing the pipeline architectures.

4. Acknowledgements.

I would like to sincerely thank Vincent Rijmen for valuable comments and discussions on the implementations of the cipher Rijndael. I am grateful to Janusz Szmidt for encouragement in the work and the help in the redaction of this article. I thank Piotr Bora for valuable comments during the realisation at the project.

Bibliography:

[1] J. Daemen, V. Rijmen, „AES Proposal: Rijndael”, http://www.esat.kuleuven.ac.be/~rijmen/rijndael/, September 1999r.
[2] V. Fischer, „Realization of the Round 2 AES Candidates using Altera FPGA”, http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html, March 2000r.
[3] K. Gaj, P. Chodowiec, „Implementations of the AES Candidate Algorithms using FPGA Devices” Technical Report, George Mason University, April 2000r.
[4] J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti, E. Roback, „Status Report on the First Round of the Development of the Advanced Encryption Standard,” NIST report, August 1999r.
[5] V. Rijmen, the private communications, vincent.rijmen@esat.kuleuven.ac.be.