Two Methods of Rijndael Implementation in Reconfigurable Hardware

Viktor Fischer[1] and Miloš Drutarovský[2]

[1] Laboratoire Traitement du Signal et Instrumentation,
Unité Mixte de Recherche CNRS 5516, Université Jean Monnet,
Saint-Etienne, France
fischer@univ-st-etienne.fr

[2] Department of Electronics and Multimedial Communications,
Technical University of Košice,
Park Komenského 13, 041 20 Košice, Slovak Republic
Milos.Drutarovsky@tuke.sk

Abstract. This paper presents an evaluation of the Rijndael cipher, the Advanced Encryption Standard winner, from the viewpoint of its implementation in a Field Programmable Devices (FPD). Starting with an analysis of algorithm's general characteristics a general cipher structure is described. Two different methods of Rijndael algorithm mapping to FPD are analyzed and suitability of available FPD families is evaluated. Finally, results of proposed mapping implemented in Altera FLEX, ACEX and APEX FPD are presented and compared with the fastest known Xilinx FPGA implementation. Results obtained are significantly faster than that of other implementations known up to now.

1 Introduction

Since 1997 the National Institute of Standards and Technology (NIST) has been working with the international cryptographic community to develop an Advanced Encryption Standard (AES). One of requirements given by the NIST on AES candidates [1] was the possibility of their efficient hardware implementation [2]. Compared with software-based solution, hardware implementation offers superior performance and significantly higher system security. Implementation in Field Programmable Devices (FPD)[1] adds to these two parameters a possibility to modify the algorithm in the field. Several papers dealing with implementation of AES candidates in reconfigurable hardware have been published so far. Some of them give only estimation of these parameters [3], while others present results based on implementation in FPGA [4], [5], [6]. Although some authors (e.g. [4], [5]) analyze the possibility to increase the speed using pipeline structures, the use of these structures in current cryptography is limited, because they are not

[1] There are several vendors of FPD. These vendors use different names for their FPD – e.g. Field Programmable Gate Arrays (FPGA) by Xilinx and Complex Programmable Logic Devices (CPLD) by Altera. FPD abbreviation is used as common name for all of them.

suitable for encryption/decryption in the most common feedback modes such as Cipher Block Chaining mode, Cipher Feedback Mode, and Output Feedback Mode. All of above mentioned papers present results of implementation in Xilinx FPGA (mostly in a high performance Virtex family [7]) and the final AES NIST report [2] is based on these results. Some research groups [8], [9], [10], [11] presented also results of the implementation of AES candidates in Altera FLEX logic devices [12]. In October 2, 2000 NIST has decided to propose Rijndael cipher [13] as the Advanced Encryption Standard and it is expected that Rijndael will be used by U.S. Government and, on voluntary basis, by the private sector. Based on this decision our further optimization effort was concentrated to the Rijndael algorithm and performance results presented in [9], [14] have been significantly improved in our new implementations. This improvement is based on different method of algorithm mapping, better VHDL encoding and usage of Altera low-cost ACEX[2] [15] and high-performance APEX [16] FPD families. This paper evaluates two different methods of the Rijndael cipher implementation from the viewpoint of its hardware mapping into high performance Altera FPD and it is organized as follows. A brief overview of Rijndael cipher algorithm and its basic building blocks is given in Section 2. In Section 3 aspects of proposed methods are discussed and different solutions from the viewpoint of the FPD embedded memory occupation and speed are presented. The limitations and implementation results of VHDL design for advanced Altera FPD are described in Section 4. In Section 5, the results obtained for both methods are compared and some comparisons with the fastest known implementations are made. Finally, in Section 6, possible future work is described and concluding remarks are made.

2 Rijndael Cipher Overview

2.1 Basic Algorithm Characteristics

Rijndael is a block cipher using 128, 192 and 256-bit input/output blocks and keys [13]. The sizes of data blocks and keys can be chosen independently. Number of rounds depends on both of these parameters and it is given in [13]. In the next analysis 128 bits for both I/O block and user key are assumed. Therefore, the cipher in all presented configurations operates in $N_r = 10$ rounds.

Encryption and Decryption Algorithms. Encryption and decryption (in the following text referred as *standard decryption*) algorithms for 128-bit input/output block and 128-bit user key are depicted in Fig. 1a and Fig. 1b, respectively. Round keys K_0 to K_{10} are obtained by the expansion of the user key, following the algorithm, described bellow.

As it can be seen, the cipher core is composed mostly of operations that are easy to implement in a reconfigurable hardware: byte rotation (permutation), byte substitution and bit-wise addition modulo 2 (XOR). The only exception is the *MixColumn* function and its inverse (*InvMixColumn* function), that involve

[2] Low-cost FPD are optimal for many practical cost-sensitive cryptographic applications.

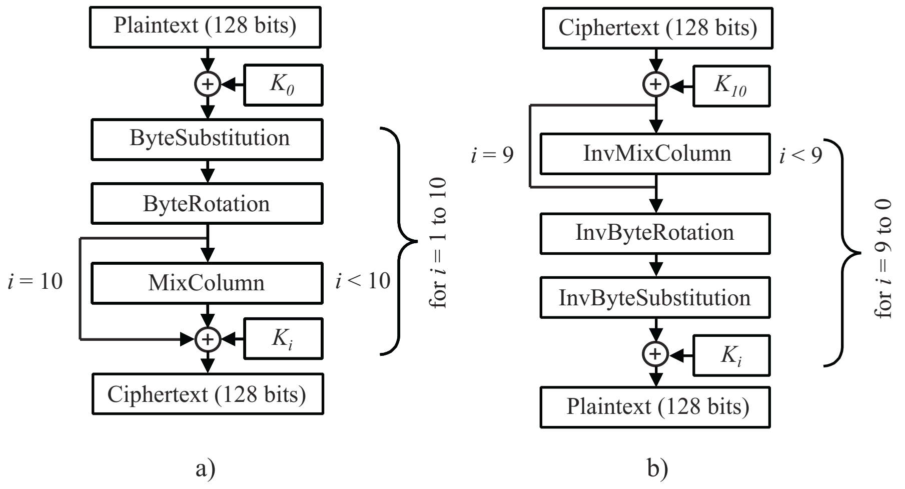

Fig. 1. Structure of Rijndael cipher a) encryption algorithm b) standard decryption algorithm

matrix multiplication on 32-bit blocks in Galois field $GF(2^8)$. Byte substitution operation uses 8×8-bit S-boxes (*byte substitution boxes*). There is one type of S-boxes for encryption and another one for decryption. Both of them are applied byte-wise on the whole 128-bit block.

Key Scheduling. Round keys K_i are derived from the user key by means of the key schedule. It consists of two components: key expansion and round key selection. Total number of round keys is equal to $N_r + 1$. The key expansion algorithm (see Fig. 2) uses bit-wise additions modulo 2 of 32-bit values obtained from user key combined with byte substitution, the byte rotation and round constants (*RCons*) addition. The order of round key calculation is the same for both encryption and decryption, although decryption uses round keys in reverse order.

Difference between Encryption and Decryption. Standard encryption and decryption algorithms use different ordering of basic operations. Moreover, basic operations for decryption are different (they implement inverse versions of basic encryption operations). As a consequence of this fact resource sharing between encryption and decryption logic is very limited.

Modification of the Order of Operations. In the table-lookup implementation it is essential that the only nonlinear step (inverse byte substitution) is the first transformation in a round and that the rows are shifted before MixColumn is applied. In the standard decryption algorithm inverse byte substitution is the last operation in the round. It is shown in [13] that order of inverse byte substitution and inverse byte rotation can be changed (both operation are byte oriented).

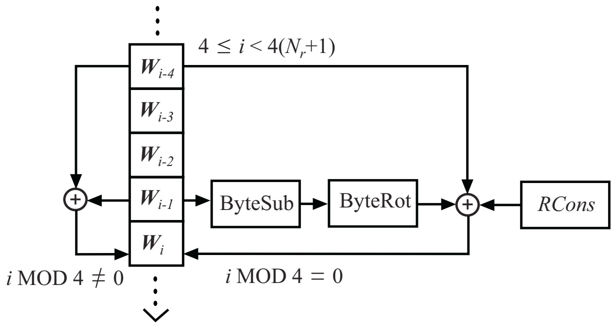

Fig. 2. Key expansion algorithm

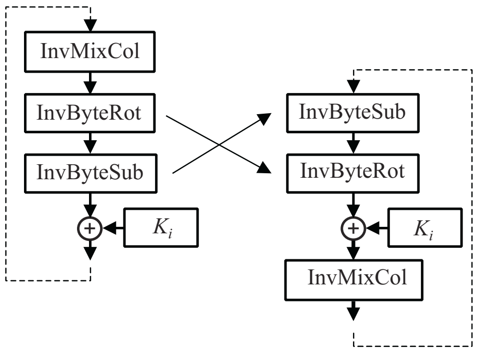

Fig. 3. Round modification for decryption

This feature is depicted in Fig. 3. Since InvMixCol is a linear transformation, the following equation is valid

$$InvMixColumn(d \oplus K) = InvMixColumn(d) \oplus InvMixColumn(K) \ . \quad (1)$$

Using properties described above, the standard Rijndael decryption algorithm can be transformed into its *modified form* described in Fig. 4. Comparing Fig. 3 and Fig. 1a it can be found that order of operation of encryption and modified decryption algorithms is the same (although significance of each operation is different). Moreover, round keys have to be inverted by InvMixColumn function with the exception of the first and last round keys.

2.2 Classification of Basic Cipher Operations and Choice of Technology

Rijndael has a relatively simple structure, while most of operations can be easily implemented in FPD. Efficient implementation of Rijndael algorithm requires ability to implement the following basic cipher operations:

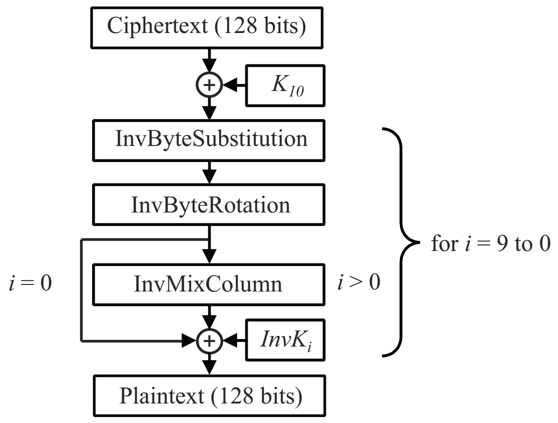

Fig. 4. Modified decryption algorithm

Bit-wise Addition Modulo-2 (XOR). This operation is easily realisable in FPD using input lookup table of *Logic Element* (LE) of Altera FPD or Configurable Logic Block (CLB) of Xilinx FPD. XOR operation with two to four inputs can be implemented in each LE or CLB slice (1/2 of CLB).

Fixed Rotation (Byte Permutation). Also this operation can be easily implemented but in this case routing resources are used. Cell interconnections can be reordered in a very simple way to implement rotations in both directions. Byte permutation order is different for encryption and for decryption.

8×8-bit *S*-boxes. Rijndael cipher uses 2 types of fixed 8×8-bit S-boxes: S-box $S[x]$ for encryption and inverse S-box $S^{-1}[x]$ for decryption. For memory limited implementations both S-boxes can be efficiently computed using the algorithm described in [13]. Actual design choice depends on features of FPD family. 8×8-bit S-boxes should preferably be realised using large embedded memories, because combinatorial function would occupy many resources (input LUTs of LE or CLB). Dedicated embedded memory blocks are ideal for implementing S-boxes. We have used them in implementations based on Altera devices.

In general, S-boxes can be implemented as lookup tables using dedicated embedded memories or within a set of small memories of LEs or CLBs configured as memory elements. Actual design choice depends on features of FPD family. 8×8-bit S-boxes should preferably be realised using large embedded memories, because combinatorial function would occupy many resources (input LUTs of LE or CLB). Dedicated embedded memory blocks are ideal for implementing S-boxes and they were used in implementations based on Altera devices.

The size of required memory depends on number of bytes that should be substituted in one clock period. If the whole 128-bit word should be processed in one period, 16 identical 8×8-bit S-boxes have to be used for encryption and for decryption. This requires the total memory capacity of 65536[3]. Since all operations of the round can be executed in parallel during one clock period, the algorithm can be executed in at least $N_r + 1$ clock periods.

MixColumn and InvMixColumn Functions. There is a quite important difference between encryption and decryption. MixColumn function

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} , \tag{2}$$

where $\bullet$ represents the multiplication in $GF(2^8)$ using the primitive polynomial $m(x) = x^8 + x^4 + x^3 + x^1 + 1$ and $X_i, Y_i \in GF(2^8)$, is replaced by its inverse – InvMixColumn

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \bullet \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} . \tag{3}$$

Matrix Multiplications in $GF(2^8)$. These operations constitute the main obstacle in efficient implementation of this cipher in programmable devices. Authors of Rijndael propose a method using so called *XTime* function [13] to solve this problem. This 8-bit function can be easily implemented also in FPD and the matrix multiplication represents XOR operations applied on the outputs of this function. There is also another possibility to realise matrix multiplication. Since square matrices in MixColumn (2) and InvMixColumn functions (3) contain constant elements (polynomials in $GF(2^8)$), it can be shown [17], that this multiplication can be replaced by several XOR ($\oplus$) operations that are simple to implement in FPD. For example, operation

$$Y = 03 \bullet X, \qquad \text{for } X, Y \in GF(2^8) \tag{4}$$

represents multiplication in $GF(2^8)$ using primitive polynomial $m(x)$. It can be implemented using following bit-wise XOR operations

$$\begin{array}{ll} y_7 = x_7 \oplus x_6 & y_6 = x_6 \oplus x_5 \\ y_5 = x_5 \oplus x_4 & y_4 = x_7 \oplus x_4 \oplus x_3 \\ y_3 = x_7 \oplus x_3 \oplus x_2 & y_2 = x_2 \oplus x_1 \\ y_1 = x_7 \oplus x_1 \oplus x_0 & y_0 = x_7 \oplus x_7 \oplus x_0 \end{array} . \tag{5}$$

This way matrix multiplication can be replaced by several XOR operations.

[3] In some FPD families that include large dedicated embedded memory blocks (e.g. 4 kbit blocks in Altera FLEX 10KE and ACEX 1K) it make no sense to use compact S-box and inverse S-box representation based on the affine transform [13].

Key Scheduling. The key scheduling is different for both encryption and decryption. Encryption round keys are used in normal order and can be computed on-the-fly. During standard decryption encryption round keys are used in reverse order and so they cannot be computed on-the-fly. For modified decryption depicted in Fig. 4, additional InvMixColumn function have to be applied on encryption round keys [13]. Round keys can be calculated easily from the user key using operations as XOR and rotation on 32-bit data. So the key schedule computation is very fast. Since round key preparation for modified decryption algorithm is more complex (application of InvMixColumn function on encryption round keys), *decryption latency* (cipher preparation time) could be higher than that of encryption. Encryption and decryption use $(N_r + 1)$ 128-bit keys, so the RAM capacity should be at least 1408 bits.

Choice of FPD Technology. From the above analysis it follows that critical operations from the point of view of their implementation in FPD are byte substitution and matrix multiplication. While fast byte substitution necessitates the presence of huge and fast memory blocks, matrix multiplication needs high fan-in combinatorial parts and significant count of global interconnections. Altera FLEX, ACEX and APEX families seem to fulfil better the first condition. On the contrary, Xilinx VIRTEX family offers more interconnection flexibility and more convenient combinatorial part of CLB (two LUTs per CLB). Since the speed of byte substitution operation seemed to be dominant in overall cipher speed, we have selected Altera FPD to implement the algorithm.

3 Methods of Rijndael Mapping to FPD

The speed and FPD resource requirements of Rijndael cipher mapping depends on the method used for actual mapping into available FPD resources. This section analyzes two mapping methods optimized for FPD with large embedded memory blocks (EMB), e.g. *Embedded Array Block* (EAB) in FLEX and ACEX devices or *Embedded System Block* (ESB) in APEX devices. Two types of cipher core configurations in feedback mode based on basic iterative architecture without loop unrolling are assumed: a *fast configuration* and an *economic configuration.* For both configurations it is assumed that encryption and/or decryption round keys are precomputed and stored in the EMBs.

The cipher architecture in the fast configuration is shown in Fig. 5. One round of the cipher is implemented as a mixture of combinational logic and access to EMBs, supplemented with a single register and a multiplexer. In the first clock cycle, complete input block of data (128 bits) is fed through a multiplexer, and stored in the register. In each subsequent clock cycle, one round of the cipher is evaluated, the result is fed back to the circuit through the multiplexer, and stored in the register. Therefore encryption and decryption can be made in 11 clock cycles.

The cipher architecture in the economic configuration is very similar to that in the fast configuration. The only difference is that it uses cipher core with

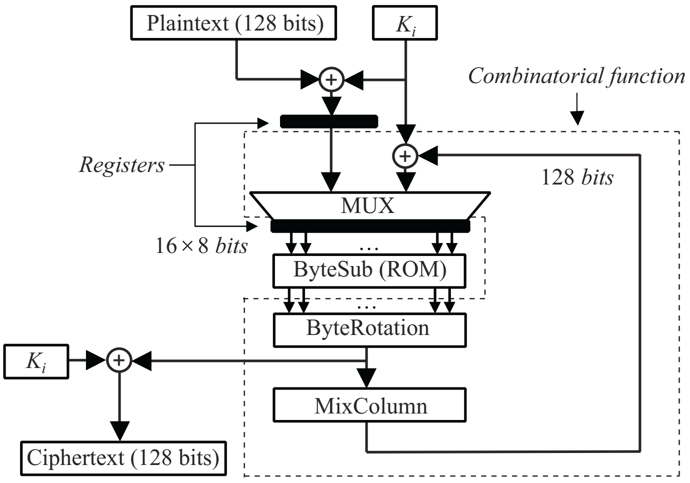

Fig. 5. Cipher architecture in the fast configuration

resource (especially EMBs) sharing. Internal data block, the 128-bit cipher state, is processed in 64 (32) bit sub-blocks in 2 (4) subsequent clock cycles. One round of cipher is executed in 2 (4) clock cycles and complete encryption/decryption can be made in 22 (44) clock cycles. The economic configuration needs 2 (4) times less S-boxes than the fast configuration.

3.1 Mapping Based on 8 × 8-Bit S-boxes

This approach was used in all known FPD implementations of Rijndael algorithm since it has minimal memory requirements. For the fast configuration (see Fig. 5) it uses 16 identical 8×8-bit S-boxes. Algorithm mapping for encryption is based on block diagram described in Fig. 1a and for decryption on that in Fig. 1b. It is clear that the logic for encryption and decryption is different and cannot be shared. Encryption and decryption S-boxes are also different. Since EAB in FLEX 10KE and ACEX 1K families contains 4096 bits RAM/ROM bits, two S-boxes, one for encryption and one for decryption, occupy exactly one EAB. Derivation of the cipher structure in economic configuration is straightforward and contains only some additional multiplexers and counters.

We shall now discuss some aspects of MixColumn and InvMixColumn transformations implementation. The complexity of these transformations is very different from the point of view of their implementation in FPD. Each of 32 output bits of the MixColumn block is a function of 5 or 7 input bits. On the contrary, InvMixColumn's output bits depend on 11 to 19 (!) input bits. Since LE is optimized for implementation of 4-input logic functions, these large combinatorial functions have to be implemented in several levels, e.g. 5 or 7-input functions in two levels and 17 or 19-input functions in 3 levels. This multilevel logic slows down significantly the final cipher speed, especially in the decryption logic. We have studied the possibilities to adapt function implementation to the structure of available logic cells. While APEX family offers the possibility to use high fan-in product term logic, this possibility could not be exploited, because product term logic is not suitable for XOR function mapping. Therefore we have tried to take advantage of another feature of Altera FPD families - the fast carry chain interconnections of neighboring logic cells. Although these interconnections are designed to implement fast arithmetic functions, they can also be used for wide logic functions implementations. Advantage of this method is that signal transitions via carry chain are several times (up to four times) faster than the transitions through complete logic cell. Disadvantage of the method lies in the fact that only neighboring cells can be interconnected. Unfortunately, matrix multiplication in MixColumn, but especially in InvMixColumn transformations represents a huge logic function with a lot of interconnections. For this reason, the use of carry chain for multiplication implementation has brought some speed improvement (up to 20 %), but it did not attained our expectations. Other negative aspect of the use of carry chains is their vendor specific character. Nevertheless, we can conclude that the utilization of carry chains in Altera FPD stays useful and we use them as often as possible in our cipher implementations.

3.2 Mapping Based on 8 × 32-Bit *T*-boxes

This approach was originally proposed for 32-bit processors [13]. From the point of view of memory requirements, it is less attractive than method based on 8 × 8-bit *S*-boxes, since in the worst case it uses 4-times more embedded memory. This is clear disadvantage of this approach. On the other hand, in FPD with 4-kbit EABs it uses just 2-times more EABs. Since the high performance FPD (e.g. APEX devices) include relatively large embedded memories, these FPD can be used for mapping fast cipher configuration based on larger 8 × 32-bit *T*-boxes. Features and advantages of FPD implementation based on *T*-boxes are described in this section.

T-boxes approach combines *S*-boxes and the MixColumn operation for the encryption process into four 8 × 32-bit tables [13]

$$T_0[a] = \begin{bmatrix} S[a] \bullet 02 \\ S[a] \\ S[a] \\ S[a] \bullet 03 \end{bmatrix} \qquad T_1[a] = \begin{bmatrix} S[a] \bullet 03 \\ S[a] \bullet 02 \\ S[a] \\ S[a] \end{bmatrix}$$
$$T_2[a] = \begin{bmatrix} S[a] \\ S[a] \bullet 03 \\ S[a] \bullet 02 \\ S[a] \end{bmatrix} \qquad T_3[a] = \begin{bmatrix} S[a] \\ S[a] \\ S[a] \bullet 03 \\ S[a] \bullet 02 \end{bmatrix} \tag{6}$$

These tables with 256 4-byte word entries make up 4 Kbytes of total space. Using these tables, the complete round transformation for a 32-bit block can be expressed as [13]

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j-1}] \oplus T_2[a_{2,j-2}] \oplus T_3[a_{3,j-3}] \oplus K_j \tag{7}$$

where K_j is the round key in round j. Since MixColumn operation is not performed in the last round of encryption algorithm, the last round have to be specially handled: S-boxes have to be used instead of T-boxes. Fortunately, S-boxes can be easily extracted from T-boxes: since all $T_i[a]$, $i = 0, 1, 2, 3$ boxes contain in some rows direct $S[a]$ values, we can get substitution result by combining T-boxes outputs of selected bytes (where S-box output value has not been multiplied by the constants 02 or 03).

In order to use T-boxes approach for decryption, the cipher structure described in Fig. 1b have to be modified. This implementation aspect has been anticipated in the design of Rijndael cipher [13]. The modified structure of decryption algorithm (see Fig. 4) is the same, as the structure of encryption algorithm, therefore T-box approach shown in Fig. 6 can be directly used also for decryption with the exception that new set of inverse T^{-1}-boxes must be used:

$$T_0^{-1}[a] = \begin{bmatrix} S^{-1}[a] \bullet 0E \\ S^{-1}[a] \bullet 09 \\ S^{-1}[a] \bullet 0D \\ S^{-1}[a] \bullet 0B \end{bmatrix} \qquad T_1^{-1}[a] = \begin{bmatrix} S^{-1}[a] \bullet 0B \\ S^{-1}[a] \bullet 0E \\ S^{-1}[a] \bullet 09 \\ S^{-1}[a] \bullet 0D \end{bmatrix}$$
$$T_2^{-1}[a] = \begin{bmatrix} S^{-1}[a] \bullet 0D \\ S^{-1}[a] \bullet 0B \\ S^{-1}[a] \bullet 0E \\ S^{-1}[a] \bullet 09 \end{bmatrix} \qquad T_3^{-1}[a] = \begin{bmatrix} S^{-1}[a] \bullet 09 \\ S^{-1}[a] \bullet 0D \\ S^{-1}[a] \bullet 0B \\ S^{-1}[a] \bullet 0E \end{bmatrix} \tag{8}$$

Since none row of T_j^{-1}, $j = 0, 1, 2, 3$ contains unmodified $S^{-1}[a]$ values, extraction of $S^{-1}[a]$ values from T^{-1}-boxes must be done by multiplication of selected row by the multiplicative inverse in $GF(2^8)$ of the corresponding constant ($0E^{-1} = E5$, $09^{-1} = 4F$, $0D^{-1} = E1$, $0B^{-1} = C0$) according to equations

$$S^{-1}[a] = 09^{-1} \bullet [S^{-1}[a] \bullet 09] = 4F \bullet [S^{-1}[a] \bullet 09] \tag{9}$$

similarly

$$S^{-1}[a] = E5 \bullet [S^{-1}[a] \bullet 0E] = S^{-1}[a] = C0 \bullet [S^{-1}[a] \bullet 0B] \tag{10}$$

Multiplication by these constants in $GF(2^8)$ can be represented using following bit-wise XOR operations

$$S^{-1}[x] = E5 \bullet X \rightarrow \begin{array}{l} s_7^{-1} = x_7 \oplus x_5 \oplus x_4 \oplus x_2 \oplus x_1 \oplus x_0 \\ s_6^{-1} = x_7 \oplus x_6 \oplus x_4 \oplus x_3 \oplus x_1 \oplus x_0 \\ s_5^{-1} = x_6 \oplus x_5 \oplus x_3 \oplus x_2 \oplus x_0 \\ s_4^{-1} = x_5 \oplus x_4 \oplus x_2 \oplus x_1 \\ s_3^{-1} = x_7 \oplus x_5 \oplus x_3 \oplus x_2 \\ \mathbf{s_2^{-1}} = \mathbf{x_6 \oplus x_5 \oplus x_0} \\ \mathbf{s_1^{-1}} = \mathbf{x_7 \oplus x_5 \oplus x_4} \\ s_0^{-1} = x_6 \oplus x_5 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 \end{array} \tag{11}$$

$$S^{-1}[x] = 4F \bullet X \rightarrow \begin{aligned} \mathbf{s_7^{-1}} &= \mathbf{x_7 \oplus x_4 \oplus x_1} \\ \mathbf{s_6^{-1}} &= \mathbf{x_6 \oplus x_3 \oplus x_0} \\ \mathbf{s_5^{-1}} &= \mathbf{x_5 \oplus x_2} \\ \mathbf{s_4^{-1}} &= \mathbf{x_4 \oplus x_1} \\ s_3^{-1} &= x_7 \oplus x_4 \oplus x_3 \oplus x_1 \oplus x_0 \\ s_2^{-1} &= x_7 \oplus x_6 \oplus x_4 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 \\ s_1^{-1} &= x_6 \oplus x_5 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 \\ \mathbf{s_0^{-1}} &= \mathbf{x_5 \oplus x_2 \oplus x_0} \end{aligned} \tag{12}$$

$$S^{-1}[x] = C0 \bullet X \rightarrow \begin{aligned} s_7^{-1} &= x_7 \oplus x_6 \oplus x_4 \oplus x_1 \oplus x_0 \\ s_6^{-1} &= x_7 \oplus x_6 \oplus x_5 \oplus x_3 \oplus x_0 \\ s_5^{-1} &= x_6 \oplus x_5 \oplus x_4 \oplus x_2 \\ s_4^{-1} &= x_7 \oplus x_5 \oplus x_4 \oplus x_3 \oplus x_1 \\ \mathbf{s_3^{-1}} &= \mathbf{x_3 \oplus x_2 \oplus x_1} \\ s_2^{-1} &= x_7 \oplus x_6 \oplus x_4 \oplus x_2 \\ s_1^{-1} &= x_7 \oplus x_6 \oplus x_5 \oplus x_3 \oplus x_1 \\ s_0^{-1} &= x_7 \oplus x_5 \oplus x_2 \oplus x_1 \end{aligned} \tag{13}$$

Since equations (11)-(13) enable to get the same $S^{-1}[x]$ value, all output bit values s_i^{-1}, $i = 0, 1, \ldots, 7$ of $S^{-1}[x]$ can be computed as combinatorial logic function with maximally 3 logical inputs (chosen from equations (11)-(13) and typed in bold face) implemented within one LE. This function is implemented in a "Multiplication elimination" block depicted in Fig. 6.

4 Results of Implementation in Altera FPD

To map Rijndael algorithm into Altera FPD, the VHDL-based design methodology has been used. It should be stressed, that all presented results have been obtained using timing analysis and implementation reports generated by Altera MAX+PLUS II v.9.6 and QUARTUS v.2000.5 development tools. The results of mapping based on 8×8-bit S-boxes are summarized in Tables 1-3 and results for 8×32-bit T-boxes in Tables 4-5.

Table 1. Fast configuration with 16 S-boxes and 128-bit data blocks in APEX 20KE200-1 (using 50 ESBs = 98% of total ESB count)

Logic Elements Used						Speed (Mbits/s)		
Encrypt		Decrypt		Both		Enc	Dec	Both
LE	%	LE	%	LE	%			
1257	15	1738	21	2493	30	964	694	612

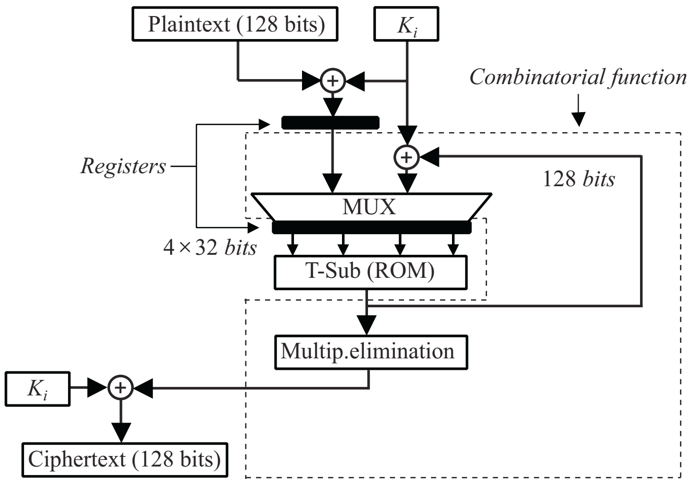

Fig. 6. Cipher architecture based on T-boxes approach

Table 2. Fast configuration with 16 S-boxes and 128-bit data blocks in FLEX 10KE200-1 (using 24 EABs = 100% of total EAB count)

Logic Elements Used						Speed (Mbits/s)		
Encrypt		Decrypt		Both		Enc	Dec	Both
LE	%	LE	%	LE	%			
1265	12	1801	18	2530	25	570	505	451

Table 3. Economic configuration with 8 S-boxes and 64-bit data blocks in ACEX 1K100-1 (using 12 EABs = 100% of total EAB count)

Logic Elements Used						Speed (Mbits/s)		
Encrypt		Decrypt		Both		Enc	Dec	Both
LE	%	LE	%	LE	%			
1461	29	2006	40	2923	59	282	238	212

Table 4. Fast configuration with 16 T-boxes and 128-bit data blocks in APEX 1K400-1 (using 86 ESBs = 82% of total ESB count)

Logic Elements Used						Speed (Mbits/s)		
Encrypt		Decrypt		Both		Enc	Dec	Both
LE	%	LE	%	LE	%			
529	3	529	3	845	5	750	750	750

Table 5. Economic configuration with 4 T-boxes and 32-bit data blocks in ACEX 1K50-1 (using 10 EABs = 100% of total EAB count)

Logic Elements Used						Speed (Mbits/s)		
Encrypt		Decrypt		Both		Enc	Dec	Both
LE	%	LE	%	LE	%			
824	29	824	29	1213	42	115	115	115

5 Discussion

5.1 Comparison of Two Methods of Rijndael Implementation

Analyzing results given in previous section we can present next advantages of the method using S-boxes approach:

– lower memory requirements for S-boxes implementation,
– no latency during encryption/decryption changing,
– very fast encryption, but significantly slower decryption.

As disadvantages of this method we can name:

– low-level of resources sharing,
– high count of logic elements used.

The second method based on T-boxes brings following advantages:

– faster overall cipher speed (for both encryption and decryption),
– high level of resources sharing, due to the symmetry of encryption/decryption
– very few logic elements used, because matrix multiplication is realized using look-up tables.

The disadvantages of the second method are:

– relatively high latency when changing encryption to decryption and vice versa – T-boxes have to be generated from S-boxes stored in one EMB (this latency can be reduced to zero, if double amount of EMBs is used),
– double (or quadruple) memory needs for T-boxes implementations (one T-box has 8 kbits, while one S-box has 2 kbits).

We can conclude that the first method could be better for applications, where only encryption algorithm is used. On the contrary, the second method should give better results if both encryption and decryption have to be fast. In the economic version (where commutation latency is acceptable) T-boxes can be computed from S-boxes stored in one EMB after each direction commutation. In the fast version separate T-boxes can be used for both encryption and decryption. This will reduce commutation latency to zero.

5.2 Comparison with Known FPD Implementations

Several Rijndael cipher implementations have been published up to now. Table 6 gives the FPD implementation results of encryption/decryption speed in the feedback cipher mode published in [4] - ELB, [6] - DAN, [8] - GAJ and [10] - MUT. For comparison NSA implementation in 0.5 μm ASIC [8] is included as well. Figure 7 compares known implementations in low-cost Altera FPD. It can be seen, that our 16 S-boxes implementation is the fastest implementation of Rijndael cipher in low-cost Altera FPD. T-boxes approach permits to implement the Rijndael cipher in as small circuit as ACEX 1K50,leaving almost 60 % of resources free! As it can be seen in Fig. 8, the encryption/decryption in our fast configuration based on T-boxes implementation is more than 80 % faster than the fastest FPD implementation known to us. It can also be seen that S-boxes approach for comparable Altera FLEX and Xilinx VIRTEX families gives similar results.

Table 6. Results of Rijndael implementations

Logic Elements Used	Speed Mbits/s
Fast (T-boxes, 128 bit blocks)	**750**
Fast (S-boxes, 128 bit blocks)	**612**
NSA	606
GAJ	414
DAN	353
ELB	300
MUT	248
Economic (S-boxes, 64 bit blocks)	212
Economic (T-boxes, 32 bit blocks)	115

6 Conclusions

In this paper we have evaluated the Rijndael cipher from the point of view of its implementation in reconfigurable hardware. The implementation results given in the previous sections depend significantly on the used FPD family. The Altera ACEX FPD have been found to be an excellent solution for very fast Rijndael cipher implementation in the reconfigurable hardware. Presented new solution based on T-boxes allows implement Rijndael cipher with the same high speed of encryption and decryption. On the other side, low-cost ACEX FPD family is suitable for cost-sensitive encryption applications. Future development will include integration of circuits for key exchange based on public-key schemes. Although current implementation uses only 128-bit keys, extension to larger keys (192 and 256 bits) requires just minor algorithm modifications and allows reach higher security with only minimal additional development effort.

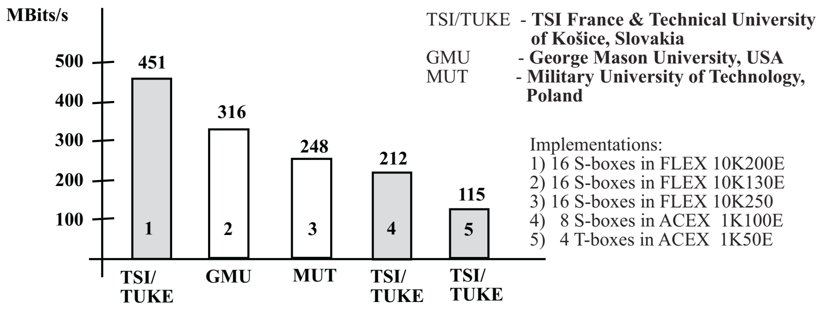

Fig. 7. Comparison of known Rijndael cipher implementations in Altera FLEX 10K and ACEX 1K FPD

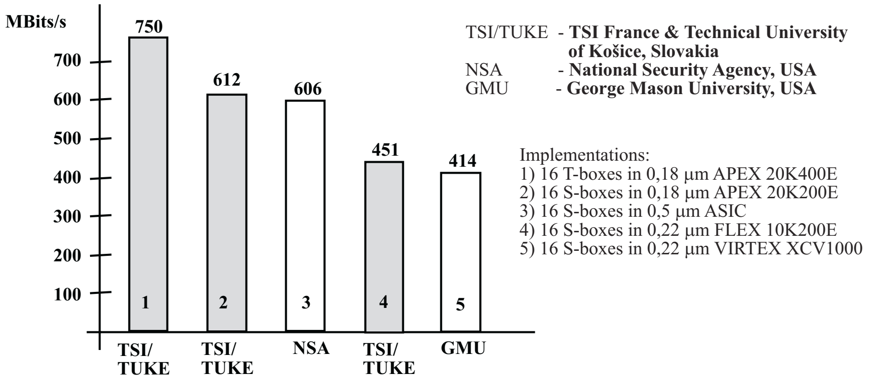

Fig. 8. Comparison of fastest known Rijndael cipher implementations in feedback mode for different FPD and ASIC

References

1. Advanced Encryption Standard. http://www.nist.gov/aes/
2. Nechavatal, J. at al.: Report on the development of the Advanced Encryption Standard (AES). NIST [1], October (2000) 1–116
3. Weaver, N., Wawrzynek, J.: A Comparison of the AES Candidates Amenability to FPGA Implementation. Proc. of The Third Advanced Encryption Standard Candidate Conference, NIST, Gaithersburg, MD, April 13-14, (2000) 28–39
4. Elbirt, A. at al.: An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. Proc. of The Third Advanced Encryption Standard Candidate Conference, NIST, Gaithersburg, MD, April 13-14, (2000) 13–27
5. Gaj, K., Chodowiec, P.: Comparison of the hardware performance of the AES candidates using reconfigurable hardware. Proc. of The Third Advanced Encryption Standard Candidate Conference, NIST, Gaithersburg, MD, April 13-14, (2000) 40–56.

6. Danalis, A., Prasanna, V., Rolim, J.: A Comparative Study of Performance of AES Final Candidates Using FPGAs. Submission for The Third AES Candidate Conference, New York, March 21, 2000 available at [1]
7. Virtex series FPGAs. http://www.xilinx.com/products/virtex.com
8. Gaj, K., Chodowiec, P.: Hardware performance of the AES finalists-survey and analysis of results. Available at http://ece.gmu.edu/crypto/
9. Fischer, V.: Realization of the Round 2 Candidates using Altera FPGA. Submitted for The Third Advanced Encryption Standard Candidate Conference, New York, March 21, (2000), available at [1]
10. Bora, P., Czajka, T.: Implementation of the Serpent Algorithm Using Altera FPGA Devices. Public Comments on AES Candidate Algorithms-Round 2, available at [1]
11. Mroczowski, P.: Implementation of the block cipher Rijndael using Altera FPGA. Public Comments on AES Candidate Algorithms-Round 2, available at [1]
12. FLEX 10KE Embedded Programmable Logic Family. http://www.altera.com
13. Daemen, J., Rijmen, V.: AES Proposal: The Rijndael Block Cipher. Version 2, September (1999) 1–45, available at [1]
14. Fischer, V. Realisation of the RIJNDAEL Cipher in Field Programmable Devices. Proceedings of DCIS 2000, Montpellier, November (2000) 312–317
15. ACEX 1K Programmable Logic Family. http://www.altera.com
16. APEX 20K Programmable Logic Family. http://www.altera.com
17. Chodowiec, P., Gaj, K. Implementation of the Twofish Cipher Using FPGA Devices. Technical Report, George Mason University, July (1999) 1–24, available at http://ece.gmu.edu/crypto/