Hardware Implementation of the 64-bit MISTY1 Block Cipher

Paris Kitsos

VLSI Design Laboratory
Electrical and Computer Engineering Department
University of Patras, Patras, Greece
pkitsos@ee.upatras.gr

Abstract: An alternative architecture and VLSI implementation of the 64-bit NESSIE proposal, MISTY1 block cipher, is presented in this paper. For this implementation, FPGA device was used. This architecture is suitable for applications with high throughput requirements. A throughput of up to 7.2 Gbps can be achieved at a clock frequency of 96 MHz. The main characteristic of this implementation is that uses RAM blocks that are embedded in the FPGA device in order to implement the necessary by the algorithm S-boxes.

Keywords: MISTY1, block cipher, cryptography, NESSIE, unrolled architecture.

1 Introduction

Due to the rapid development of wireless standards, the security subject in mobile communications has gained more importance. Many attempts have taken place in order to establish qualitative cryptography methods. The New European Schemes for Signatures, Integrity, and Encryption (NESSIE) project [1] had as a goal to establish a portfolio of strong cryptographic primitives of various types. For block ciphers a third security level, *normal-legacy*, has been specified, which means a block size of 64 bits instead of 128 (the AES [2] does not specify smaller block size than 128-bit). It is interested in 64-bit block ciphers which are more secure and efficient than the ones presently used. In February 2003, it was announced that the 64-bit block cipher included in the NESSIE portfolio is MISTY1 [3, 4]. This cipher is designed in order to provide high-level security against differential and linear cryptanalysis.

In the third NESSIE workshop [1], a paper for some NESSIE proposal algorithms was presented [5]. The main purpose of this work was the evaluation of these algorithms in terms of hardware implementation performance. In this evaluation, only the encryption mode of operation was implemented and not the decryption one.

Except of this work, for the implementation of the MISTY1 block cipher some other implementations have been published [6, 7]. The proposed work in [6] is exactly the same one as the MISTY1 implementation proposed in [5]. As previously mentioned, these implementations do not support the decryption mode of the cipher. In [7], two MISTY1 software implementations on a Digital Alpha processor were proposed. Nevertheless, it is well known that the software implementations are much slower than the hardware ones.

In this paper, an architecture and efficient VLSI implementation of the 64-bit NESSIE proposal MISTY1 block cipher is proposed. This alternative design implement both encryption and decryption modes in the same hardware module. Is suitable for applications with high throughput requirements. The main feature of this implementation is the unrolling of the cipher rounds in a 75-stage pipeline. Due to the increased critical path delay of the logical expressions, for the S-boxes implementation, the RAM blocks embedded in the FPGA device are used.

The rest of the paper is organized as follows: In section 2, the MISTY1 block cipher is introduced. In section 3, the proposed hardware implementation is presented and explained in detail. Performance analysis and comparison results with other works are given in section 4, while section 5 concludes the paper.

2 MISTY1 Block Cipher

The MISTY1 [3, 4] block cipher operates with 64-bit block size plaintext and 128-bit secret key. The respective 64-bit ciphertext is produced after a number of n rounds, where n is a multiple of four. In [8] a number n=8 is recommended for use in real applications. Two are the main parts of the MISTY1 block cipher, the *data randomizing* and the *key scheduling*. In the following these two parts are described.

2.1 MISTY1 Data Randomizing Part

The MISTY1 data randomizing part for n=8 is shown in Fig.1. It consists of 8 identical stages (rounds) with an additional substage (subround).

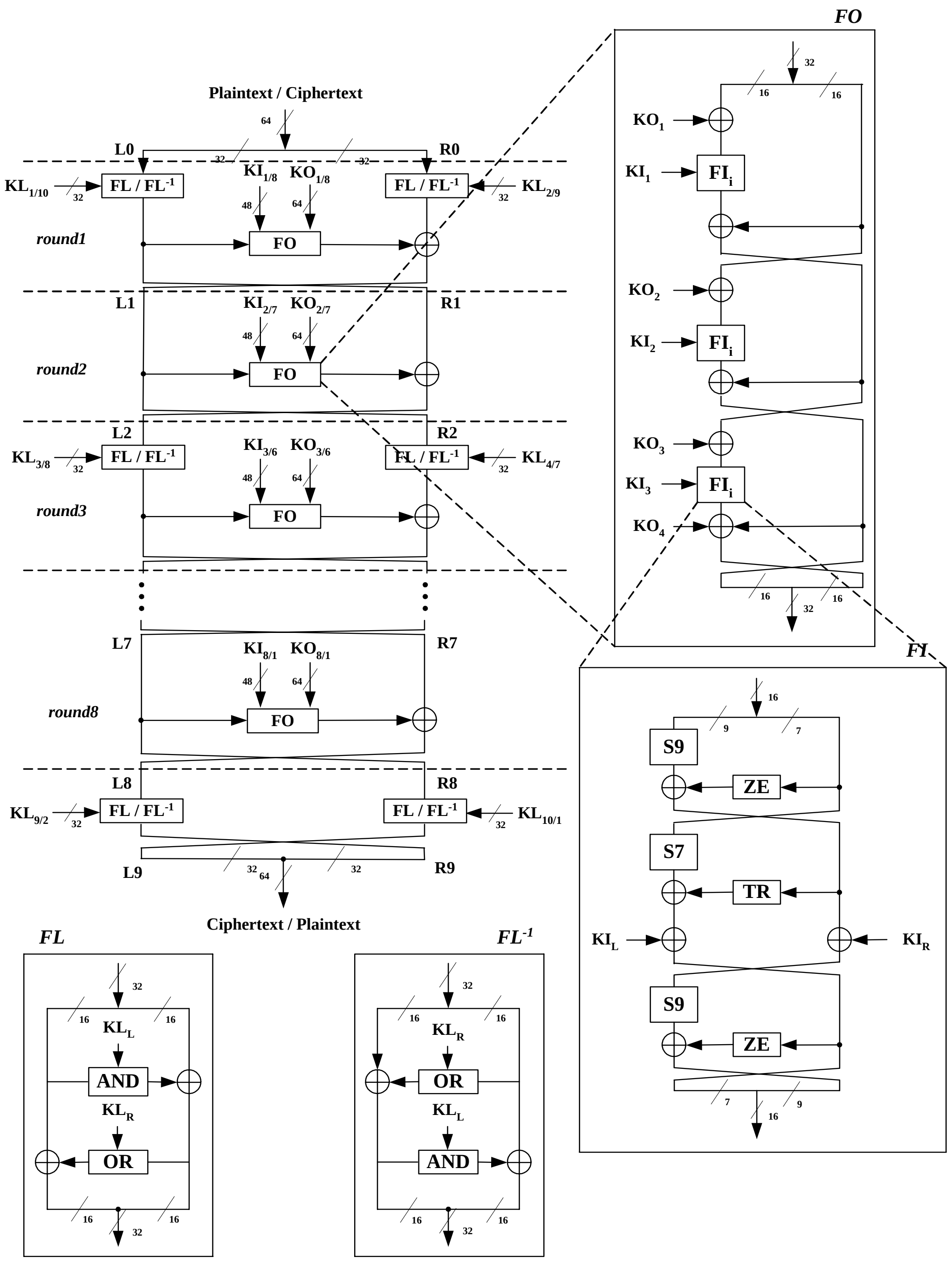

Fig. 1. Data Randomizing Part

In the encryption mode operation, the 64-bit plaintext is transformed into the 64-bit ciphertext by applying bitwise XOR operations and the sub-functions FO_i $(1 \le i \le 8)$ and FL_i $(1 \le i \le 10)$. In the beginning the 64-bit plaintext is divided into two 32-bit strings, the left and the right one. The subfunction FO_i uses a 48-bit sub-key KI_i and a 64-bit sub-key KO_i . The subfuntion FL_i uses a 32-bit sub-key KL_i. The output of each round (stage) is produced by the following equations.

For the odd rounds (i = 1, 3,…, 7) :

Right string: $R_i = FL(L_{i-1}, KL_i)$ and

Left string: $L_i = FL(R_{i-1}, KL_{i+1}) \oplus FO(L_i, KO_i, KI_i)$

For the even rounds (i = 2, 4,…, 8) :

Right string: $R_i = L_{i-1}$ and

Left string: $L_i = R_{i-1} \oplus FO(L_i, KO_i, KI_i)$.

For the last round (i = 9) :

Left string: $R_9 = FL(L_8, KL_9)$ and

Right string: $L_9 = FL(R_8, KL_{10})$.

The final 64-bit ciphertext is produced from the concatenation of L_9 and R_9.

The decryption mode operation of MISTY1 is similar to the encryption mode. The only differences are the reverse order of the sub-keys and the replacement of the function *FL* by the function FL^{-1}. Similarly to the encryption mode, in the decryption one the 64-bit ciphertext is divided into the left and right 32-bit strings, which are transformed into the 64-bit plaintext by applying bitwise XOR operations and the sub-functions FO_i $(8 \geq i \geq 1)$ and FL_i $(10 \geq i \geq 1)$. The output of each round is described with the same equations as in encryption if the FL function is replaced by the FL^{-1}. The resulting plaintext is produced by the concatenation of the final left and right 32-bit strings that are produced by the last subround.

The structure of the FL function is shown in Fig. 1. The 32-bit data is split into two 16-bit halves. KL_L is the left and KL_R is the right part of the KL 32-bit sub-key respectively. After AND, OR, and XOR operations between the data and the sub-key a 32-bit string is produced. In the decryption mode the FL^{-1} function is used instead of the FL one.

The 32-bit input data of function FO is split into two 16-bit strings (Fig. 1). Then, these strings are correlated with KO_j $(1 \leq j \leq 4)$ and KI_j $(1 \leq j \leq 3)$ by using bitwise XOR operations and the sub-functions FI. KO_j and KI_j are the left j-th 16 bits of KO and KI, respectively.

The 16-bit input data of the function FI is split into two 9-bit and 7-bit strings (Fig. 1). After transformations, bitwise XOR operations and the usage of the substitution tables (S-boxes) S7 and S9, the output string is produced. At the beginning and at the end of FI_j function, the 7-bit string is zero-extended (through the ZE module). The ZE adds two zero bits in front of the 7-bit string, and in the middle part, the 9-bit string is truncated (through the TR module) to 7 bits. The TR module truncates the two most significant bits of the 9-bit string. KI_L and KI_R are the left 7 bits and the right 9 bits of KI, respectively.

The two S-boxes (S7 and S9) have been designed so that they can be easily implemented in combinational logic as well as by a look-up-tables. Three criterias are considered in order to select these S-boxes.

- Their average differential/linear probability must be minimal,
- Their delay time in hardware is as short as possible,
- Their algebraic degree is high, if possible.

2.2 MISTY1 Key Scheduling Part

MISTY1 has a 128-bit key *K*, which is sub-divided into eight 16-bit sub-keys K_1, K_2, …, K_8 where $K=K_1||K_2||K_3||K_4||K_5||K_6||K_7||K_8$ (|| symbolizes concatenation). From these sub-keys a second set of sub-keys, K_i' ($1 \leq i \leq 8$) is produced as is shown in Fig. 2.

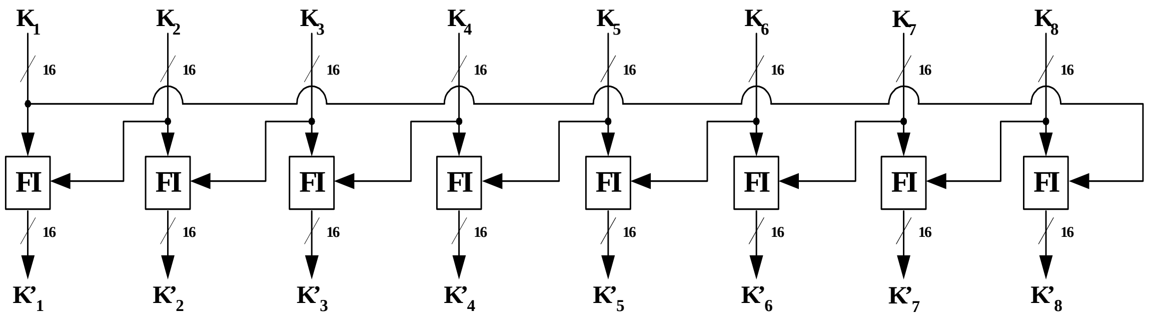

Fig. 2. Second Set of Sub-keys

In Table I, the sub-keys that are used in each round are shown.

Table I. Round Sub-keys Mapping Table

Round	***KO_1***	***KO_2***	***KO_3***	***KO_4***	***KI_1***	***KI_2***	***KI_3***	***KL_L***	***KL_R***
1	K_1	K_3	K_8	K_5	K'_6	K'_2	K'_4	K_1	K'_7
2	K_2	K_4	K_1	K_6	K'_7	K'_3	K'_5	K'_3	K_5
3	K_3	K_5	K_2	K_7	K'_8	K'_4	K'_6	K_2	K'_8
4	K_4	K_6	K_3	K_8	K'_1	K'_5	K'_7	K'_4	K_6
5	K_5	K_7	K_4	K_1	K'_2	K'_3	K'_8	K_3	K'_1
6	K_6	K_8	K_5	K_2	K'_3	K'_4	K'_8	K'_5	K_7
7	K_7	K_1	K_6	K_3	K'_4	K'_5	K'_1	K_4	K'_2
8	K_8	K_2	K_7	K_4	K'_5	K'_6	K'_2	K'_6	K_8
9	-	-	-	-	-	-	-	K_5	K'_3

3 Hardware Implementation

Usually each proposal of a new algorithm is accompanied with a software implementation in a commonly used language. But, it is well known that the software implementations are much slower than the hardware ones. In this section, the alternative hardware implementation of the MISTY1 block cipher is proposed. This implementation is based on the following Eight Round Architectures. As previously mentioned, the two main parts of the MISTY1 block cipher are the data randomizing and the key scheduling. In Fig. 3, the data randomizing part of the eight rounds architecture is illustrated. By using this architecture both encryption and decryption operations can be performed. This is opposed to the [5, 6], where only the encryption process is considered and implemented.

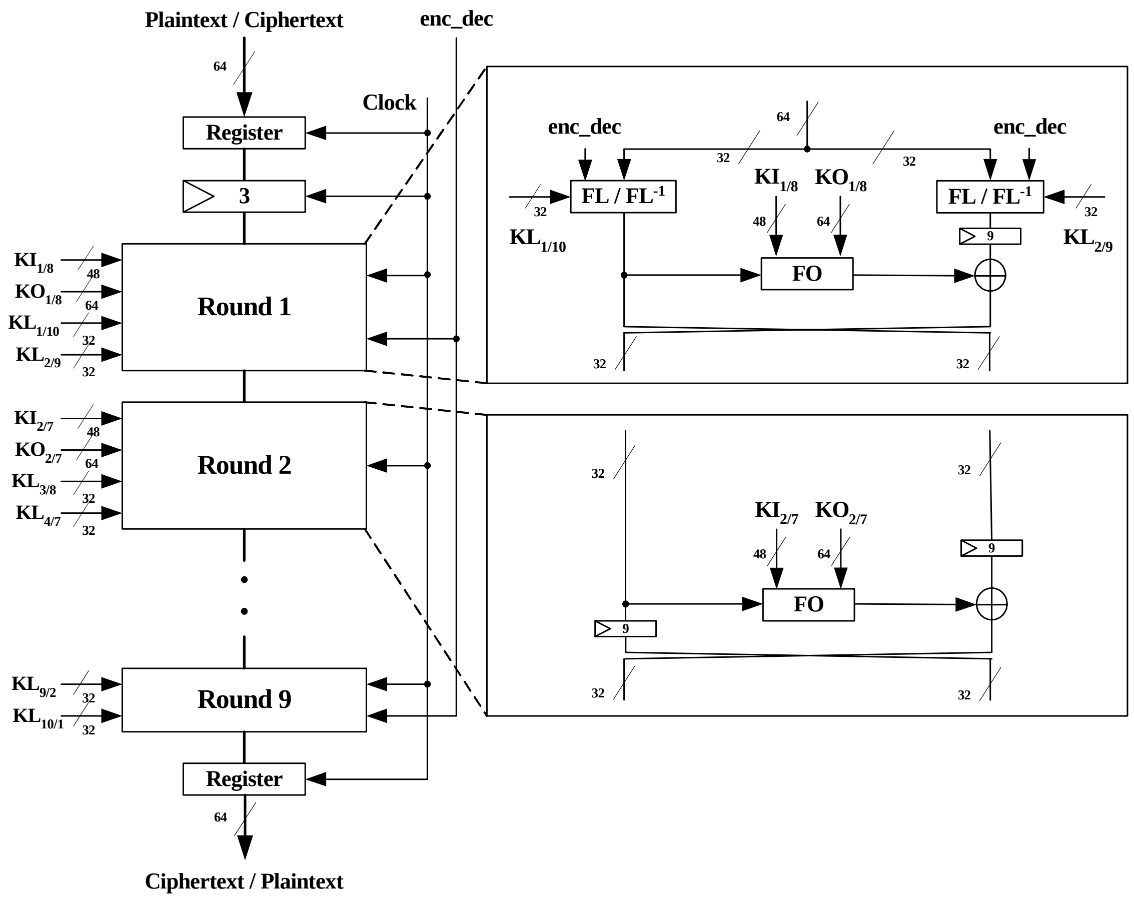

Fig. 3. Eight Rounds Data Randomizing Part Architecture

Compared with previous designs, an alternative architecture for the S-boxes implementation is introduced. For the implementation of the S-boxes (S7 and S9), RAM blocks are used instead of logical expressions. Thus a, significant reduction of the critical path delay is achieved. In order to synchronize the S-boxes operations (ROM-based operations) three 1-stage pipeline registers have been inserted in the right branch of the architecture, one for each corresponding ROM (Fig. 4a).

The structure of all the odd and even rounds are identical. For synchronization reasons, pipeline registers are inserted between the functional units. So, after the first input register a pipeline register with 3-stage delay is added. These pipeline registers are inserted in order to synchronize the key generation part with the data randomizing part. The explanation is given in the following. The insertion of the pipeline registers (9-stage delay) in the odd and even rounds architectures results in an architecture with a 75 pipeline stages.

Fig. 4b, indicates the insertion places of the added pipeline registers in the FO function architecture. Due to the FI function architecture (Fig. 4a) that uses three 1-stage pipeline registers, for the synchronization of the FI functions in FO function architecture, 3-stage pipeline registers are needed.

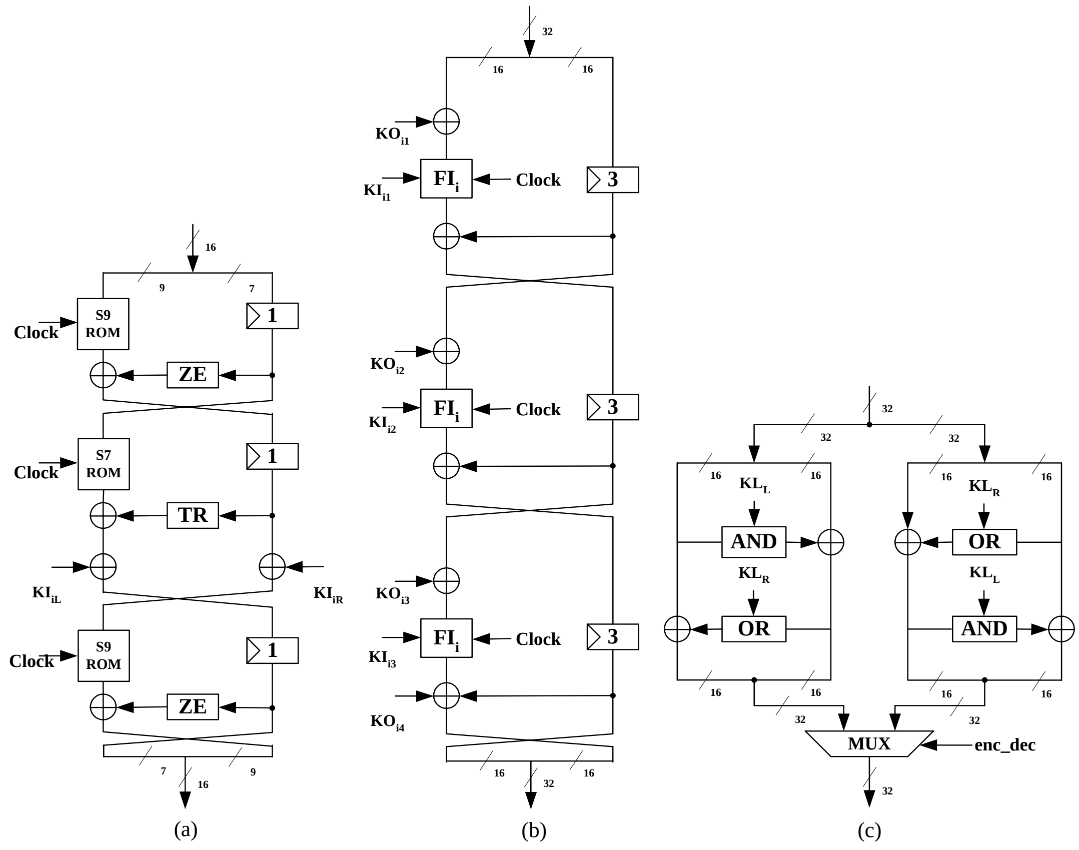

Fig. 4. Architecture of the (a) FI, (b) FO, and (c) FL Functions

The structure of the MISTY1 decryption procedure is similar to the encryption one. In order the proposed architecture to be also suitable for decryption operation, the process of reversing the sub-keys order and replacing the function *FL* with FL^{-1} are needed. The first one is performed in the proposed MISTY1 key scheduling part and is described in the following. The latter one is achieved by designing a unit, which implements both *FL* and FL^{-1} functions. The value of the control signal (*enc_dec)* in conjuction with the inserted multiplexer (Fig. 4c) selects the proper output.

The MISTY1 key scheduling part architecture is shown in Fig. 5. The proposed scheme allows on-the-fly computation of the sub-keys. The sub-key input in each *FI* function is delayed with the usage of the 2-stage pipeline registers. This is necessary because inside the *FI* unit the KI_L and KI_R sub-keys are entered with a 2-stage delay after the latch of the key value. So, the second array of the sub-keys, K_i', is generated with a 3-stage delay. In order to synchronize the key generation part with the data randomizing part, an additional 3-stage pipeline register is inserted after the input register and before the start of the data randomizing part (Fig. 3).

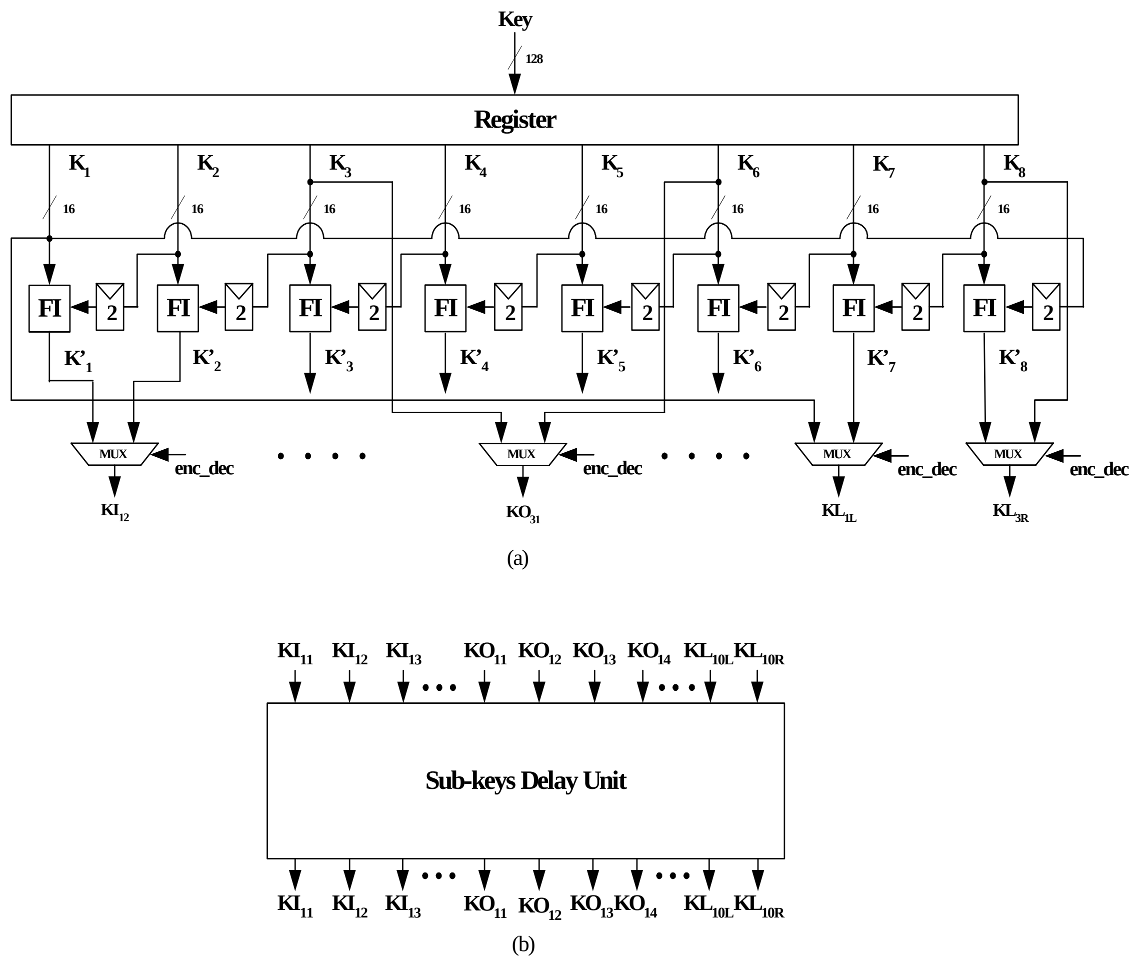

Fig. 5. Eight Rounds Key Scheduling Part Architecture

In order the same architecture to be used for both encryption and decryption operations, multiplexers (controlled by the enc_dec signal) are added. For example, during the encryption operation the value of the sub-key KO_{31} has the same value as K_3, while during the decryption operation the same value as K_6. So, with the usage of a 2-input 16-bit multiplexer the proper value is selected. Moreover, a Sub-keys Delay Unit (Fig. 5b) is necessary in order to add additional delays. In this unit, 16-bit shift registers are used, so as in each round to add the sub-keys with the proper delay.

4 Measurements and Comparisons

The proposed architectures and implementations were captured by using VHDL, with structural description logic. The encryption and decryption operation were verified by using the test vectors provided by the NESSIE submission package [1]. The VHDL codes of the two designs were synthesized by using FPGA devices of Xilinx [9]. The correct functionality of the two hardware implementations was verified with simulations. The measurements of the performance analysis are shown in Table II. In the same table, measurements from other designs are added (only the faster software implementation is added [7]).

Table II. Performance Analysis Measurements

Architecture	**Process**	**FPGA Device**	**CLB Slices**	**F MHz**	**Throughput (Mbps)**	**Throughput/ Area (Mbps/Area)**
[5]	Encryption	XCV1000 BG560-6	8386	140	8960	1.07
[6]	Encryption	XCV1000 BG560-6	6322	159	10176	1.6
[6]	Encryption	XCVII2000 BG575-6	6322	303	19392	3
[7]	Encryption/ Decryption	Software	-	500	288	-
Proposed Eight Round	Encryption/ Decryption	XCV1000 BG560-6	4735	96	7200	1.5
Proposed Eight Round	Encryption/ Decryption	XCVII3000 BF957-6	4039	168	12600	3.3

The architectures proposed in [5] and [6] are identical, but the authors have reported better implementation performance results in [6] than in [5]. Most probably this difference is due to better VHDL code synthesis of [6].

For the hardware implementation of the proposed MISTY1 eight round architecture two Xilinx Virtex devices (XCV1000BG560-6 and XCVII3000BF957-6) were selected. The proposed implementation did not fit in the Virtex-II device, which is used in [6], because the proposed implementation uses more embedded RAM blocks than the available ones in this device. The XCV1000BG560-6 device has 128K bits of embedded RAM (BlockSelectRAM+), divided in 32 RAM blocks that are separated from the main body of the FPGA. The XCVII3000BF957-6 device has 1728K bits of embedded RAM (BlockSelectRAM+), divided in 96 RAM blocks. For the proposed design 79K bits of embedded RAM are used in order to map the necessary for the block cipher S-boxes.

For the addition of the sub-keys delays (necessary in the key scheduling part) a 16-bit shift register is used. The Virtex architecture is well suited to easily implement, fast, and efficient shift register by the usage of the SRL16 feature [10]. Whit this feature, shift registers are implemented without using flip-flop resources. The SRL16 is used to implement a progressive delay line, thereby saving logic resources and producing the highest performance [10]. So, each 16-bit shift register is implemented by one Look-Up-Table (LUT). The implementation in the XCV1000BG560-6 device achieves an operation frequency of 96 MHz and a throughput of 7.2 Gbps, while the implementation in the XCVII3000BF957-6 device achieves a frequency of 168 MHz and a throughput of 12.6 Gbps. The critical path delay is determined by the S-box S9 (RAM block) delay.

The proposed implementation supports encryption and decryption in the same dedicated FPGA device. In order to incorporate this feature, a large number of 2-input 16-bit multiplexers are used. In addition, extra FL^{-1} functions and 2-input 32-bit multiplexers are necessary. So, in the performance comparisons with other architectures that support only encryption [5, 6], this feature must be considered. Of course, the penalty is the minor increase of the allocated hardware resources.

Looking at the Table II, the Throughput/Area ratio illustrates that MISTY1 is efficient and suitable for FPGA implementation. Finally, the proposed Eight Round Architecture implementations in XCVII3000 BF957-6 FPGA is more efficient, compared with the implementation in [6] in the same FPGA family, as the Throughput/Area measurements show.

5 Conclusions

An efficient VLSI architecture for the design and implementation of the MISTY1 block cipher has been presented. In contrast to previous implementation, the proposed one supports encryption and decryption. The rounds are unrolled and RAM blocks embedded in the FPGA are used for the implementation of the S-boxes. A 75-stage pipeline is inserted that achieves a throughput of 72 Gbps at an operation frequency of 96 MHz. This architecture can be applied in high-speed applications.

References

[1] "NESSIE. New European Schemes for Signatures, Integrity, and Encryption", https://www.cosic.esat.kuleuven.ac.be/nessie/

[2] "Advanced Encryption Standard Development Effort," http://www.nist.gov/aes, 2000.

[3] Mitsuru M. Specification of MISTY1 – a 64-bit Block Cipher. *New European Scheme for Signatures, Integrity, and Encryption (NESSIE) Project*. On line available at https://www.cosic.esat.kuleuven.ac.be/nessie/workshop/submissions.html

[4] Mitsuru M. New block encryption algorithm MISTY. *In Fast Software Encryption 1997*, vol. 1267 of LNCS, pages 54-68. Springer-Verlag, 1997.

[5] Standaert F-X, Rouvroy G, Quisquater J-J, Legat J-D. Efficient FPGA Implementations of Block Ciphers KHAZAD and MISTY1. *In Proceedings of the Third NESSIE Workshop*, November 6-7 2002, Munich, Germany.

[6] Rouvroy G, Standaert F-X, Quisquater J-J, Legat J-D. Efficient FPGA Implementation of Block Cipher MISTY1. *In Proceedings of the 10th Reconfigurable Architectures Workshop (RAW 2003)*, April 22, Nice, France.

[7] Nakajima J and Matsui M. *Fast Software Implementations of MISTY1 on Alpha Processors*. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E82-A, No. 1, pp. 107-116, January 1999.

[8] Matsui M *Supporting Document of MISTY1*. version 1.1. On line available at https://www.cosic.esat.kuleuven.ac.be/nessie/workshop/submissions.html

[9] Xilinx Inc., San Jose, Calif., “Virtex, 2.5 V Field Programmable Gate Arrays,” 2001, www.xilinx.com

[10] LeonardoSpectrum, Synthesis and Technology Manual. Software Release v2001.1, February 2001.