Efficient FPGA Implementation of Block Cipher MISTY1

Gael Rouvroy, Francois-Xavier Standaert
Jean-Jacques Quisquater, Jean-Didier Legat
Universite catholique de Louvain, Microelectronic Laboratory
Place du Levant, 3, 1348 Louvain-La-Neuve, Belgium
rouvroy,standaert,quisquater,legat@dice.ucl.ac.be

Abstract

NESSIE is a 3-year research project (2000-2002). The goal of the project is to put forward some algorithms to obtain a set of the next generation of cryptographic primitives. In order to achieve this objective, the project needs to evaluate mathematical security levels and software/hardware implementations. This paper investigates the significance of an FPGA implementation of the block cipher MISTY1. Reprogrammable devices such as FPGA's are highly attractive solutions for hardware implementations of encryption algorithms. A strong focus is placed on a high throughput circuit which completely unrolls all the MISTY1 rounds and pipelines them in order to increase the data rate. Our design allows us to change the plaintext and the key on a cycle-by-cycle basis with no dead cycles. The final core implementation can work at a data rate up to 19.4 Gbps (303 MHZ)[1].

1. Introduction

The NESSIE project[2] is about to put forward a portfolio of strong cryptographic primitives that has been obtained after an open call and been evaluated using a transparent and open process. These primitives include block ciphers, stream ciphers, hash functions, MAC algorithms, digital signature schemes, and public-key encryption schemes. The technical analysis used in determining which of the NESSIE candidates will be selected as a standard block cipher includes efficiency testing of both hardware and software implementations of candidates algorithms.

The encryption algorithm MISTY1 is a 64-bit block cipher with a 128-bit key and a variable number of rounds. Mitsuru Matsui, the designer (1996), recommends a 8-round version. It is a Feistel cipher. MISTY1 is designed on the basis of the theory of provable security against differential and linear cryptanalysis. In addition, it allows us to realize high speed encryption on hardware platforms as well as on software environments [2]. The detailed specification can be found in [2, 3]. In this paper, we study the efficiency of MISTY1 encryption for an FPGA implementation. Our fast core is detailed and compared to AES RIJNDAEL, SERPENT, KHAZAD and previous MISTY1 circuits to determine the efficiency of our MISTY1 FPGA implementation.

The paper is organized as follows: Section 2 describes the FPGA technology used and the synthesis/implementation tools; Section 3 gives a short mathematical description of MISTY1; MISTY1 implementation is detailed in section 4; Section 5 compares our MISTY1 design to FPGA implementations of AES RIJNDAEL, SERPENT, KHAZAD and previous MISTY1; Section 6 concludes this paper.

2. Hardware description

In this section, we briefly describe the structure of a VIRTEX FPGA as well as the synthesis and implementation tools that were used to obtain our results.

Configurable Logic Blocks (CLB's): The basic building block of the VIRTEX logic block is the logic cell (LC). A LC includes a 4-input function generator, carry logic and a storage element. The output from the function generator in each LC drives both the CLB output and the D input of the flip-flop. Each VIRTEX CLB contains four LC's, organized in two similar slices. Figure 1 shows a detailed view of a single slice. Virtex function generators are implemented as 4-input look-up tables (LUT's). Beside its operation as a function generator, each LUT can provide a 16×1-bit synchronous RAM. Furthermore, the two LUT's within a slice can be combined to create a 16×2-bit or 32×1-bit synchronous RAM or a 16×1-bit dual port synchronous RAM. The VIRTEX LUT can also provide a 16-bit shift register.

[1] This bitstream is obtained with a VIRTEXII2000bg575-6 device.

[2] NESSIE: New European Schemes for Signature, Integrity, and Encryption.

* This work has been funded by the Walloon region (Belgium) through the research project TACTILS.

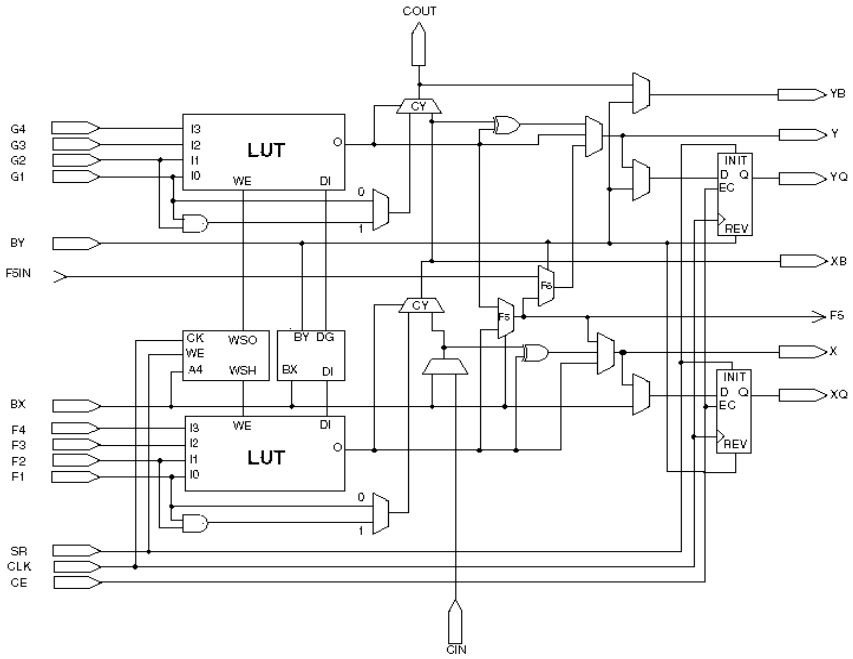


Figure 1. The VIRTEX slice.

The storage elements in the VIRTEX slice can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches. The D inputs can be driven either by the function generators within the slice or directly from slice inputs, bypassing function generators.

The F5 multiplexer in each slice combines the function generator outputs. This combination provides either a function generator that can implement any 5-input function, a 4:1 multiplexer, or selected functions of up to nine bits. Similarly, the F6 multiplexer combines the outputs of all four function generators in the CLB by selecting one of the F5-multiplexer outputs. This permits the implementation of any 6-input function, an 8:1 multiplexer, or selected functions up to 19 bits. The arithmetic logic also includes a XOR gate that allows a 1-bit full adder to be implemented within an LC. In addition, a dedicated AND gate improves the efficiency of multiplier implementations.

Finally, VIRTEX FPGA's incorporate several large RAM blocks. These complement the distributed LUT implementations of RAM's. Every block is a fully synchronous dual-ported 4096-bit RAM with independent control signals for each port. The data widths of the two ports can be configured independently.

Hardware targets: For our implementations, we used VIRTEX and VIRTEXII technologies. We chose these technologies in order to allow relevant comparisons with the best-known FPGA cipher implementations. In this paper, we compare the number of LUT's, registers and slices used. We also evaluate the delays and frequencies thanks to our implementation tools (post-map and post place-and-route frequencies). The synthesis was performed with FPGA Express 3.6.1 (SYNOPSYS) and the implementation with XILINX ISE-4. Our circuits were described using VHDL.

3. MISTY1 block cipher algorithm

The MISTY1 cipher is an iterated block cipher, with a Feistel structure, that operates on a 64-bit block with a 128-bit key and with a variable number of rounds n. We describe the algorithm with $n = 8$, as recommended in [2, 3]. In the following subsections, we illustrate the data randomizing part and the key scheduling part with their different components.

3.1. Data randomizing part

Figure 3 shows the data randomizing part of MISTY1[3]. The 64-bit plaintext P is divided in two 32-bit parts. Both parts are transformed into the 64-bit ciphertext using bitwise XOR operations, sub-functions FO_i $(1 \leq i \leq (n = 8))$ and sub-functions FL_i $(1 \leq i \leq (n + 2 = 10))$.

[3] Where registers needed for efficiency purposes are already mentioned.

3.1.1 FO_i function:

Figure 2 shows the structure of FO_i[4]. This function split the input into two 16-bit strings. Then, it transforms both strings into the output with bitwise XOR operations and sub-functions FI_{ij} ($1 \leq j \leq 3$). KO_{ij} ($1 \leq j \leq 4$) and KI_{ij} ($1 \leq j \leq 3$) are the left j-th 16 bits of KO_i and KI_i, respectively.

3.1.2 FI_{ij} function:

Figure 2 also shows the structure of FI_{ij}. The input is divided into two parts: a 9-bit string and a 7-bit string. These strings are transformed into the output using bitwise XOR operations and substitutions tables S_7 and S_9. In the beginning and the end of FI_{ij} function, the 7-bit string is zero-extend to 9 bits, and in the middle part, the 9-bit string is truncated to 7 bits eliminating its highest two bits (MSB). KI_{ij1} and KI_{ij2} are the left 7 bits and the right 9 bits of KI_{ij}, respectively.

3.1.3 FL_i function:

The structure of FL_i function is also shown in Figure 2. The 32-bit input is divided into two equal parts. The function transforms both parts into the output with bitwise AND, OR and XOR operations. KL_{ij1} ($1 \leq j \leq 2$) is the left j-th 16 bits of KL_i.

3.1.4 S_7 and S_9 substitution functions:

For the selection of S_7 and S_9 substitution functions, Matsui considers three criteria:

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

Based on these criteria, for the S_7 substitution function, Matsui chooses the following mathematical description:

$y_0 = x_0 + x_1x_3 + x_0x_3x_4 + x_1x_5 + x_0x_2x_5 + x_4x_5 + x_0x_1x_6 + x_2x_6 + x_0x_5x_6 + x_3x_5x_6 + 1$
$y_1 = x_0x_2 + x_0x_4 + x_3x_4 + x_1x_5 + x_2x_4x_5 + x_6 + x_0x_6 + x_3x_6 + x_2x_3x_6 + x_1x_4x_6 + x_0x_5x_6 + 1$
$y_2 = x_1x_2 + x_0x_2x_3 + x_4 + x_1x_4 + x_0x_1x_4 + x_0x_5 + x_0x_4x_5 + x_3x_4x_5 + x_1x_6x_3x_6 + x_0x_3x_6 + x_4x_6 + x_2x_4x_6$
$y_3 = x_0 + x_1 + x_0x_1x_2 + x_0x_3 + x_2x_4 + x_1x_4x_5 + x_2x_6 + x_1x_3x_6 + x_0x_4x_6 + x_5x_6 + 1$
$y_4 = x_2x_3 + x_0x_4 + x_1x_3x_4 + x_5 + x_2x_5 + x_1x_2x_5 + x_0x_3x_5 + x_1x_6 + x_1x_5x_6 + x_4x_5x_6 + 1$
$y_5 = x_0 + x_1 + x_2 + x_0x_1x_2 + x_0x_3 + x_1x_2x_3 + x_1x_4 + x_0x_2x_4 + x_0x_5 + x_0x_1x_5 + x_3x_5 + x_0x_6 + x_2x_5x_6$
$y_6 = x_0x_1 + x_3 + x_0x_3 + x_2x_3x_4 + x_0x_5 + x_2x_5 + x_3x_5 + x_1x_3x_5 + x_1x_6 + x_1x_2x_6 + x_0x_3x_6 + x_4x_6 + x_2x_5x_6$

Based on the same above criteria, the S_9 function is also defined as a logical description:

$y_0 = x_0x_4 + x_0x_5 + x_1x_5 + x_1x_6 + x_2x_6 + x_2x_7 + x_3x_7 + x_3x_8 + x_4x_8 + 1$
$y_1 = x_0x_2 + x_3 + x_1x_3 + x_2x_3 + x_3x_4 + x_4x_5 + x_0x_6 + x_2x_6 + x_7 + x_0x_8 + x_3x_8 + x_5x_8 + 1$
$y_2 = x_0x_1 + x_1x_3 + x_4 + x_0x_4 + x_2x_4 + x_3x_4 + x_4x_5 + x_0x_6 + x_5x_6 + x_1x_7 + x_3x_7 + x_8$
$y_3 = x_0 + x_1x_2 + x_2x_4 + x_5 + x_1x_5 + x_3x_5 + x_4x_5 + x_5x_6 + x_1x_7 + x_6x_7 + x_2x_8 + x_4x_8$
$y_4 = x_1 + x_0x_3 + x_2x_3 + x_0x_5 + x_3x_5 + x_6 + x_2x_6 + x_4x_6 + x_5x_6 + x_6x_7 + x_2x_8 + x_7x_8$
$y_5 = x_2 + x_0x_3 + x_1x_4 + x_3x_4 + x_1x_6 + x_4x_6 + x_7 + x_3x_7 + x_5x_7 + x_6x_7 + x_0x_8 + x_7x_8$
$y_6 = x_0x_1 + x_3 + x_1x_4 + x_2x_5 + x_4x_5 + x_2x_7 + x_5x_7 + x_8 + x_0x_8 + x_4x_8 + x_6x_8 + x_7x_8 + 1$
$y_7 = x_1 + x_0x_1 + x_1x_2 + x_2x_3 + x_0x_4 + x_5 + x_1x_6 + x_3x_6 + x_0x_7 + x_4x_7 + x_6x_7 + x_1x_8 + 1$
$y_8 = x_0 + x_0x_1 + x_1x_2 + x_4 + x_0x_5 + x_2x_5 + x_3x_6 + x_5x_6 + x_0x_7 + x_0x_8 + x_3x_8 + x_6x_8 + 1$

Both substitution boxes are defined as ROM tables in [2]. To optimize the number of logic cells used for FPGA implementation, we prefer to implement S_7 and S_9 functions directly as logical expressions. With enough pipelined stages, we keep under control the critical path of the design.

3.2. Key scheduling part

Figure 4 shows the key scheduling part of MISTY1[5]. K_i ($1 \leq$ i ≤ 8) is the left i-th 16 bits of the secret input key K. $K_i^{'}$ ($1 \leq i \leq 8$) corresponds to the output of FI_{ij} where the input of FI_{ij} is assigned to K_i and the key KI_{ij} is set to $K_{(i+1)mod8}$. The assignment between key scheduling subkeys K_i/ $K_i^{'}$ and the round subkeys KO_{ij}, KI_{ij}, KL_{ij} is defined in Table 1, where i equals to $(i-8)$ when $(i > 8)$.

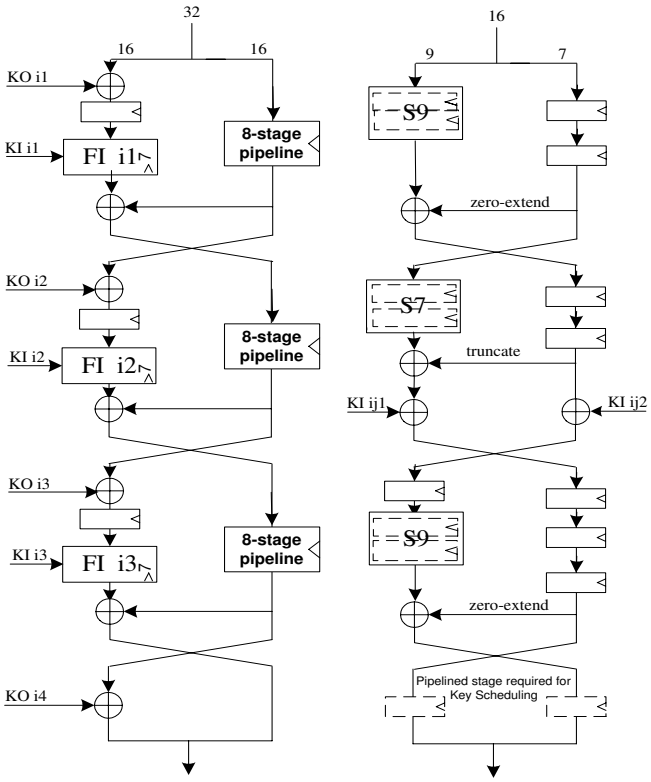
This concludes the mathematical description of MISTY1 algorithm. The next section explains our FPGA design choices in order to be efficient in term of speed and resources used.

[4] Where registers needed for efficiency purposes are already mentioned.

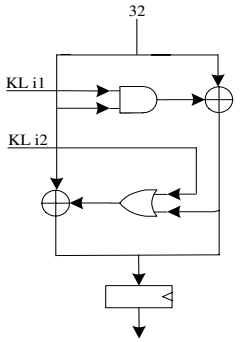
[5] Where registers needed for efficiency purposes are already mentioned.

Encrypt Round	KO_{i1}	KO_{i2}	KO_{i3}	KO_{i4}	KI_{i1}	KI_{i2}	KI_{i3}	KL_{i1}	KL_{i2}
Key round	K_i	K_{i+2}	K_{i+7}	K_{i+4}	K'_{i+5}	K'_{i+1}	K'_{i+3}	$K_{\frac{i+1}{2}}(odd.i)$ $K'_{\frac{i}{2}+1}(even.i)$	$K'_{\frac{i+1}{2}+6}(odd.i)$ $K_{\frac{i}{2}+4}(even.i)$

Table 1. Subkeys distribution.



FO_i function **FI_{ij} function**



FL_i function

Figure 2. FO_i, FI_{ij}, FL_i functions.

4. MISTY1 FPGA implementation

FPGA's are very efficient devices and they are suitable for high work frequencies. Results available in the public literature sometimes mention encryption rates comparable with software ones. We believe that these performances can be greatly improved using today's technology as soon as inherent constraints of FPGA's are taken into account. According to VIRTEX FPGA technology, we have the opportunity to implement a 4-input function within a LUT. Within a slice, we can also use two additional XORs, one F5 and one F6 functions. In order to optimize the frequency of the design, we choose to limit the critical path to only one 4-input LUT and route. So we do not use additional XORs and F5, F6 functions which will probably increase the critical path[6].

Based on this delay constraint, we modify the mathematical algorithm description to regroup a maximum number of functions in a minimum number of 4-input LUT's.

4.1. S_7 and S_9 implementations

For S_7 and S_9 implementations, we use the logical expressions in place of substitution tables in order to reduce the number of logic cells used. The logical functions have to be enough pipelined to keep the critical path to only one 4-input LUT and route. For S_7 and S_9, we obtain two 2-stage pipelined versions. Table 2 shows the results that we obtain after synthesis.

Component	Nbr of LUT's	Nbr of FF's	Nbr of pipelined stages
S_7	45	45	2
S_9	44	35	2

Table 2. S_7 and S_9 synthesis results.

[6]Although these function could be used to reduce the number of logic cells consumed.

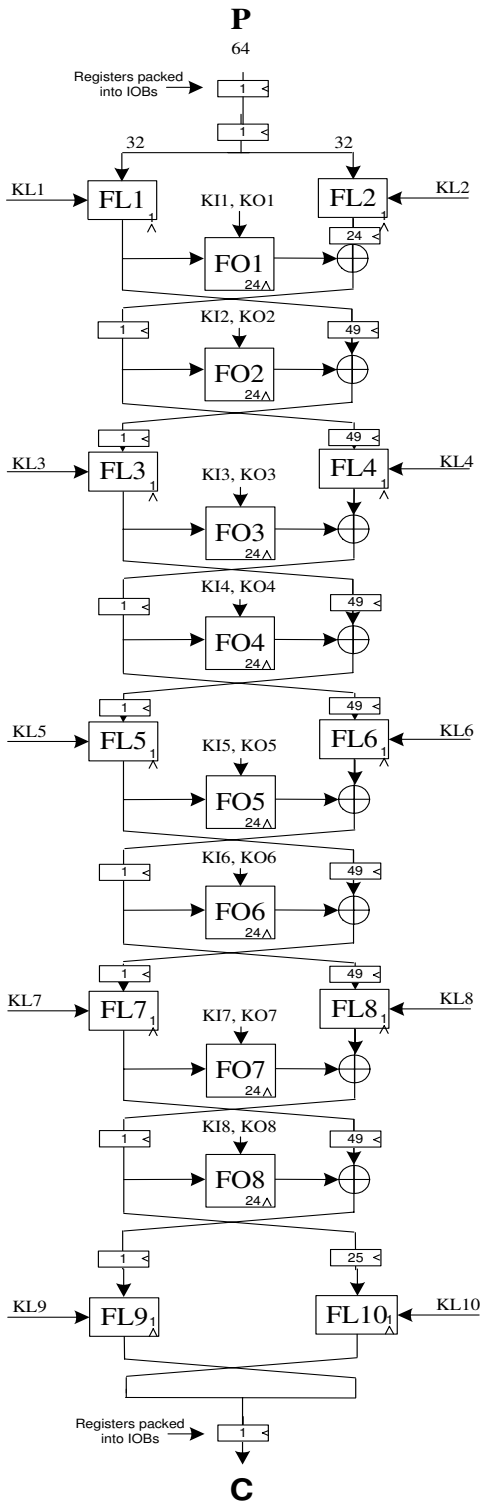


Figure 3. The data randomizing part of MISTY1.

4.2. FO_i, FI_{ij}, FL_i implementations

Figure 2 details how we implemented FO_i, FI_{ij}, FL_i functions in order to keep a critical path of only one 4-input LUT and route. As mentioned on the figure, we have to put an additional output pipelined stage into FI_{ij} function only for the key scheduling part. We do it to be speed efficient.

Table 3 shows the results that we obtain after synthesis.

Component	Nbr of LUT's	Nbr of FF's	Nbr of pipelined stages
FO_i	655	671	24
FI_{ij}	172	181	7
FL_i	32	32	1

Table 3. FO_i, FI_{ij}, FL_i synthesis results.

4.3. The data randomizing part of MISTY1

For the same delay constraints, we obtain the design detailed in Figure 3. We put additional registers for input and output bits that we pack into IOBs to increase our speed performance. We finally get a 208-stage pipelined design.

4.4. The key scheduling part of MISTY1

In order to change the secret key dynamically, we choose to also implement the key scheduling in hardware. This allows us to change the encryptor with a new key on a cycle-by-cycle basis with no dead cycles. Figure 4 shows the key scheduling part of MISTY1. Additional registers for input key bits are also packed into IOBs in order to increase performances.

The assignment between key scheduling subkeys K_i/ $K_i^{'}$ and the round subkeys KO_{ij}, KI_{ij}, KL_{ij} is defined in Table 1. We do the same in hardware putting the correct number of pipelined stages for every round subkeys. Therefore, to achieve the key distribution, we use 16-bit shift registers, every one fitting in one LUT. Figure 5 represents the subkeys distribution.

Table 4 summarizes the synthesis results that we obtain for the complete key scheduling part.

Nbr of LUT's	Nbr of FF's	Nbr of pipelined stages
4800	5016	208

Table 4. Key scheduling synthesis results.

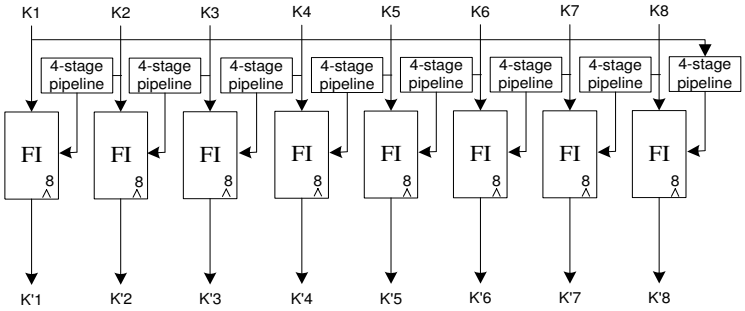


Figure 4. Key Scheduling.

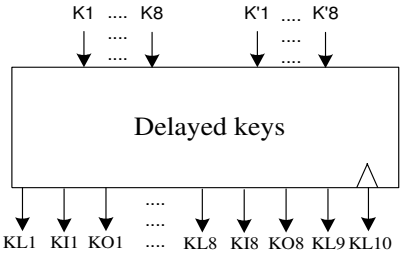


Figure 5. Subkeys distribution.

4.5. The complete cipher

The implementation of the complete MISTY1 cipher is the combination of the data randomizing part and the key scheduling part from the precedent descriptions. Our design allows us to change the plaintext and the key on a cycle-by-cycle basis with no dead cycles. The results are summarized in Table 5 for VIRTEX1000bg560-6 and VIRTEXII2000bg575-6 devices[7]. We propose a post-map and a post-implementation estimated frequency. Second one considers the routing delays.

From Table 5, we observe very high frequencies after mapping phase but the final designs only run at 159/303 MHz. Comparing with post-map frequencies, we see that we spend to much time in the routing part. Thanks to the FloorPlanner tool, we can conclude that:

1. It is not so easy to deal with additional routing delays. They are no systematic tools to prevent these delays. All we can do, is to locate the problem and to redesign the global circuit. Nevertheless, routing problems come usually from high fanout.

2. Trying to reduce the critical path to only one 4-input LUT is not always the best choice. Indeed, if a big part of the critical path is due to route, it is better to use the additional XORs, F5 and F6 can reduce the number of logic cells used without increasing the critical path.

Anyway, the resulting design is very efficient and suitable for FPGA, as proved in the next section.

5. Comparison with AES RIJNDAEL, SERPENT, KHAZAD and previous MISTY1

In order to evaluate our implementation results and the hardware suitability of MISTY1, we compare them with similar results obtained with the Advanced Encryption Standard RIJNDAEL, SERPENT, KHAZAD and previous MISTY1 designs [4, 7]. We chose RIJNDAEL because of its status of new encryption standard and SERPENT because it seems that it was the best AES candidate regarding FPGA implementations. However, comparisons with KHAZAD seems to be more relevant because it was implemented using the same methodology as we used in this paper.

In [4], the Xilinx VIRTEX1000bg560-4 was selected as the target device for evaluation of AES candidates. Table 6 compares RIJNDAEL[8], SERPENT, MISTY1 and KHAZAD encryption circuits in terms of hardware cost, frequency and throughput for VIRTEX1000bg560-6 component. The hardware cost in LUT and registers is replaced by a number of slices. We also investigate the ratio Throughput/Area which is a good measurement of hardware efficiency.

[7]The latency corresponds to the number of pipelined stages.

[8]Result every 2.1 clk edges means that we get a new output cipher every 2.1 cycles (on average).

Devices	Nbr of LUT	Nbr of registers	Nbr of slices	Latency (cycles)	Output every (cycles)	Frequency (MHz) Post-map	Frequency (MHz) Post-implementation
VIRTEX1000	11160	11920	6322	208	1	204	159
VIRTEXII	11160	11920	6322	208	1	352	303

Table 5. Implementations of MISTY1.

Type	Nbr of slices	Result every (clk edges)	Estimated frequency(Mhz)	Throughput (Mbits/s)	Throughput/Area ($\frac{Mbits/s}{slices}$)
RIJNDAEL	10992	2.1	31.8	1938	0.18
SERPENT	9004	1	38	4860	0.54
Previous MISTY1	8386	1	140	8960	1.07
Fast KHAZAD	8800	1	148	9472	1.07
Low area KHAZAD	7175	1	123	7872	1.09
Our MISTY1	**6322**	**1**	**159**	**10176**	**1.61**

Table 6. Comparisons with RIJNDAEL, SERPENT, KHAZAD and previous MISTY1.

6. Conclusions

We propose an optimized FPGA implementation of the block cipher MISTY1. We finally get a design that can operate up to 303 MHz (19.4 Gbps) using 6322 slices of the FPGA. Upon comparison, our MISTY1 implementation offers better results than those reported for RIJNDAEL, SERPENT, KHAZAD and previous MISTY1 in [4, 7]. MISTY1 gives big advantages in terms of hardware cost: it has the Feistel structure and low-cost substitution boxes. Its key scheduling is also less expansive. Looking at the final results, the ratio $Throughput/Area$ illustrates that MISTY1 is very efficient and suitable for FPGA implementation. It seems that reconfigurable hardware implementations will not be the bottleneck for the selection of MISTY1 as a NESSIE cipher.

References

[1] Xilinx. Virtex 2.5V field programmable gate arrays data sheet. Available from http://www.xilinx.com.

[2] M. Matsui New Block Encryption Algorithm MISTY. In *Proc. of FSE'97*, LNCS, Springer, 1997.

[3] M. Matsui Supporting Document of MISTY1. Available from http://www.cosic.esat.kuleuven.ac.be/nessie/

[4] A.J.Elbirt et Al, An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. The Third Advanced Encryption Standard (AES3) Candidate Conference, April 13-14 2000, New York, USA.

[5] M.McLoone et al, High Performance Single Ship FPGA Rijndael Algorithm Implementations. In *Proc. of CHES2001*: The Third International CHES Workshop, Lecture Notes In Computer Science, LNCS2162, pp 65-76, Springer-Verlag.

[6] K. Gaj et al, Comparison of the Hardware Performance of the AES Candidatesusing Reconfigurable Hardware. The Third Advanced Encryption Standard (AES3) Candidate Conference, April 13-14 2000, New York, USA.

[7] FX. Standaert, G. Rouvroy et al, Efficient FPGA Implementations of Block Ciphers KHAZAD and MISTY1. In the proceedings of Third NESSIE Workshop.

[8] G. Rouvroy et al, Efficient Uses of FPGAs for Implementations of DES and Its Experimental Linear Cryptanalysis. Accepted for publication in IEEE Transactions on Computers, Special CHES Edition, VOL.52, NO. 4, April 2003.

[9] FX. Standaert, G. Rouvroy et al, Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES Rijndael. Accepted for publication in the proceedings of FPGA 2003.