

Design Strategies and Modified Descriptions to Optimize Cipher FPGA Implementations: Fast and Compact Results for DES and Triple-DES

Gaël Rouvroy, François-Xavier Standaert,
Jean-Jacques Quisquater, Jean-Didier Legat
{rouvroy,standaert,quisquater,legat}@dice.ucl.ac.be

UCL Crypto Group
Laboratoire de Microelectronique
Université catholique de Louvain
Place du Levant, 3, B-1348 Louvain-La-Neuve, Belgium

Abstract. In this paper, we propose a new mathematical DES description that allows us to achieve optimized implementations in term of ratio $Throughput/Area$. First, we get an unrolled DES implementation that works at data rates of 21.3 Gbps (333 MHz), using Virtex-II technology. In this design, the plaintext, the key and the mode (encryption/decryption) can be changed on a cycle-by-cycle basis with no dead cycles. In addition, we also propose sequential DES and triple-DES designs that are currently the most efficient ones in term of resources used as well as in term of throughput. Based on our DES and triple-DES results, we also set up conclusions for optimized FPGA design choices and possible improvement of cipher implementations with a modified structure description.

Keywords: cryptography, DES, FPGA, efficient implementations, design methodology.

1 Introduction

The rapid growth of secure transmission is a critical point nowadays. We have to exchange data securely at very high data rates. Efficient solutions, with huge data rate constraints, have to be hardware implemented and flexible in order to evolve with the permanent changes in norms. FPGA (Field Programmable Gate Arrays) implementations of the triple-Data Encryption Standard (triple-DES) efficiently meet these constraints. Triple-DES is based on three consecutive DES¹. As detailed in [6–8, 3], DES is very well suited for FPGA solution. Accurately, we have first to focus our attention to achieve very fast DES designs with limited hardware resources.

Some high-speed DES hardware implementations have been published in the literature. These designs unroll the 16 DES rounds and pipeline them. Patterson [7] made a key-dependent data path for encryption in an FPGA which produces a bitstream of about 12 Gbps. Nevertheless the latency to change keys is tenth of milliseconds. A DES implementation is also downloadable from FreeIP [6] and encrypts at 3.05 Gbps. Last known implementations were announced by Xilinx company in [3, 8]. They were FPGA implementations of a complete unrolled and pipelined DES encryptor/decryptor. The 16-stage and 48-stage pipelined cores could achieve data rates of, respectively, 8.4 Gbps and 12 Gbps². The 48-stage pipelined version could also produce a throughput of 15.1 Gbps on Virtex-II. It also allowed us to change the plaintext, the key and the encryption/decryption mode on a cycle-by-cycle basis.

In this paper, based on our new mathematical DES description [17], we finally get also an optimized complete unrolled and pipelined DES design that encrypts with a data rate of 21.3 Gbps with 37 cycles of latency³. This design is the best currently known one in term of ratio $Throughput/Area$.

¹ Without intermediate IP and $IP-1$ permutations.

² These results were obtained with Virtex-E technology.

³ Using Virtex-II technology.

Concerning sequential designs, some DES and triple-DES FPGA implementations have been published in the literature. These designs encrypt or decrypt most of time every 16 cycles. In academic literature, we mainly found the paper of C. Paar [16] with a DES data rate of 402.7 Mbps. The most recent publication (October, 2001) comes from K. Gaj et al. [15]. They propose a tripe-DES design with a bitstream of 91 Mbps with less resources, using Virtex1000 component. For commercial Xilinx IP cores, different DES and triple-DES implementations are also available from Helion Technology, CAST, and InSilicon [10–14]. The best triple-DES one in term of ratio $Throughput/Area$ comes from CAST and gives a throughput of 668 Mbps on Virtex-II FPGA, using 790 slices (January, 2002).

Concerning our DES and triple-DES sequential implementations, we get designs that can encrypt or decrypt every 18 cycles. Our triple-DES design uses 604 slices and produces a throughput of 917 Mbps. This solution is the best sequential one known nowadays in term of resources used as well as in term of throughput.

Based on our DES and triple-DES results, we also set up conclusions for accurate design choices. We also mention the importance of cipher modified structure descriptions.

The paper is organized as follows: section 2 refers the Data Encryption Standard; section 3 explains our previous published new mathematical DES description; section 4 describes our unrolled and pipelined DES implementations and proposes a comparison with the previous Xilinx implementations; section 5 details our sequential DES and triple-DES implementations and also shows comparisons; our design strategies to optimize cipher FPGA implementations are explained in section 6; finally, section 7 concludes this paper.

2 The DES algorithm

In 1977, the Data Encryption Standard (DES) algorithm was adopted as a Federal Information Processing Standard for unclassified government communication. It is still largely in use. DES encrypts 64-bit blocks with a 64-bit key, of which only 56 bits are used. The other 8 bits are parity bits for each byte. The algorithm counts 16 rounds. DES is very famous and well known. For detailed information about DES algorithm, refer to the standard [5] or our preceding work [17].

3 New mathematical DES description

The original DES description is not optimized for FPGA implementations regarding the speed performance and the number of LUTs used. It is not an efficient way to fit as well as possible into the Virtex CLBs. In order to achieve a powerful design, we explain a new mathematical description of DES previously detailed in our work [17]. The aim is to optimize its implementations on FPGA, reducing resources used as well as the critical path of one DES round.

An FPGA is based on slices composed by two 4-bit input LUTs and two 1-bit registers. Therefore, an optimal way to reduce the LUTs use is to regroup all the logical operations to obtain a minimum number of blocks that take 4-bit input and give 1-bit output. In addition, it is important to mention that all permutation and expansion operations (typically P , E , IP , IP^{-1} , PC^{-1} and PC^{-2}) do not require additional LUTs, but only wire crossings and fanouts (pure routing).

First, we transform the round function of the enciphering computation. This transformation has no impact on the computed result of the round. Figure 1 shows a modified round representation, where we move the E box and the XOR operation.

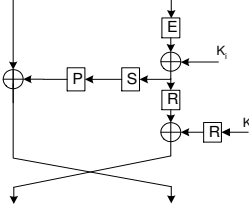


Fig. 1. Modified description of one DES-round.

This involves the definition of a new function denoted R (like reduction):

$$\begin{aligned}
 R &= E^{-1}, \\
 \forall x, R(E(x)) &= x, \\
 \exists y \mid E(R(y)) &\neq y.
 \end{aligned} \tag{1}$$

Now, if we change all the enciphering parts of DES with this modified round function and if we combine the E and XOR block with the left XOR block of the previous round, we get the **A** architecture detailed in Figure 2. Another more efficient solution is to move the R and XOR of the right part of the round into the left XOR operator of the next round. As a result, we obtain the **B** architecture shown in Figure 2.

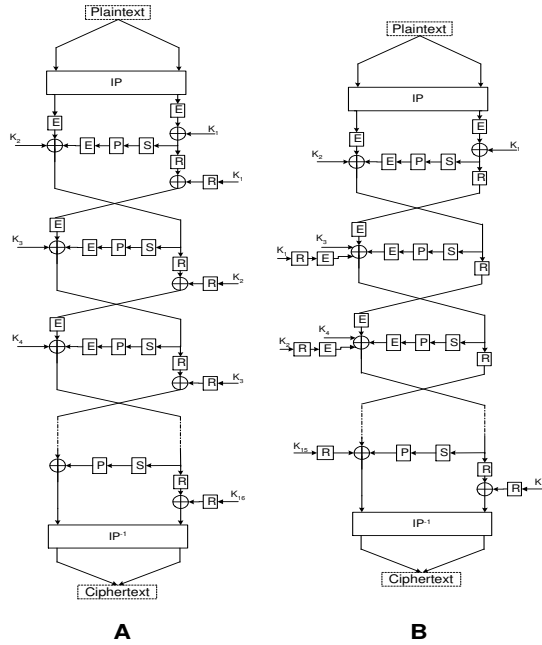


Fig. 2. Two modified descriptions of the DES algorithm.

In the B arrangement, the first and last rounds are quite different from intermediate ones. Therefore, we obtain an irregular DES description. In addition, we increase the number of E and R blocks, which will not alter the number of LUTs consumed. We also keep exactly the same number of S-boxes that is the expensive part of the architecture. Concerning the modulo two sum operators, we really decrease their number. We spare 15×32 2-bit XORs comparing with the A architecture⁴. We can directly conclude that this design will consume less logic than traditional implementations.

⁴ The design exactly counts 14×48 4-bit XORs, 1×48 3-bit XORs, 1×48 2-bit XORs, 1×32 3-bit XORs and 1×32 2-bit XORs.

Therefore, modifying the original DES description according to the B architecture of Figure 2, we are able to significantly reduce the number of resources used. In addition, we also reduce the critical path of one round function. These transformations allow us to obtain very efficient DES FPGA implementations as detailed in the next sections.

4 Our unrolled and pipelined DES implementations

To be speed efficient, we propose designs that unroll the 16 DES rounds and pipeline them, based on the B mathematical description of Figure 2. In addition, we implemented solutions that allow us to change the plaintext, the key and the encryption/decryption mode on a cycle-by-cycle basis, with no dead cycle. We can achieve very high data rates of encryption/decryption.

The left part of Figure 3 illustrates how the critical path, in our solution, is hugely decreased. We only keep one S-boxes operator and one XOR function (= two LUTs + one $F5$ function + one $F6$ function)⁵. With this solution we obtain a 1-stage pipeline per round. Due to the irregular structure of our design, we have to add an additional stage in the first round. To be speed efficient for implementation constraints, we also put a 2-stage pipeline respectively in the input and in the output. This approach allows an additional degree of freedom for the place and route tool. As mentioned in the figure, first and last registers are packed into IOBs. Therefore, we obtain a 21-stage pipeline.

In the right part of Figure 3, we put an extra pipelined stage in each round in order to limit the critical path to only one S-box. As a consequence, we finally get a 37-stage pipeline. Table 1 shows our 21-stage and 37-stage pipelined results.

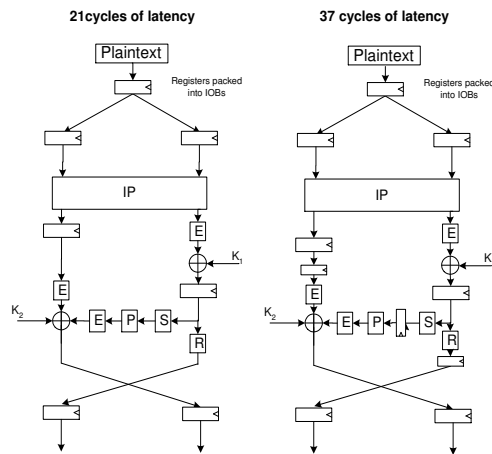


Fig. 3. Our two unrolled and pipelined DES designs.

Comparing to the preceding efficient Xilinx implementations [3, 8], our 21-stage gives better results in terms of speed and logical resources. Nevertheless, it is quite more registers consuming. For the 37-stage pipeline, we use less LUTs again. We also reduce the registers needed. This is due to the fact that we only have a 2-stage pipeline per round. In addition, this design uses shift registers for the key schedule calculation. In Virtex FPGAs, *SRL16* blocks can directly implement a 16-bit shift register into one LUT. This design uses 484 extra LUTs for the key schedule calculation⁶.

⁵ E and R operators do not increase the critical path.

⁶ Estimation of the corresponding Xilinx implementation gives the use of about 900 LUTs for shift registers. No accurate values are given in [3, 8].

Pipeline	Xil. 16-stage	Xil. 48-stage	Our 21-stage	Our 37-stage
LUTs used	4216	4216	3775	3775
Registers used	1943	5573	2904	4387
Frequency in XCV300-6	100 MHz	158 MHz	127 MHz	175 MHz
Data rate in XCV300-6	6.4 Gbps	10.1 Gbps	8.1 Gbps	11.2 Gbps
Frequency in XCV300E-6	132 MHz	189 MHz	176 MHz	258 MHz
Data rate in XCV300E-6	8.4 Gbps	12.0 Gbps	11.2 Gbps	16.5 Gbps
Frequency in XC2V1000-5	/	237 MHz	227 MHz	333 MHz
Data rate in XC2V1000-5	/	15.1 Gbps	14.5 Gbps	21.3 Gbps

Table 1. Final results of our pipelined implementations.

The reason why we have a better speed result for the 37-stage pipeline is quite strange. Obviously, in their design, they do not put registers into IOBs and an additional pipelined stage before and after encryption. Without such registers, the critical path is in the input and output paths.

In addition, after checking and simulating their available source code on the web, we found two errors. First, they forgot to put a 1-stage pipeline after the XOR between key and R part. Actually, Xilinx implemented this 1-stage pipeline but they sent the XOR directly between key and R part into S-boxes, instead of the corresponding registered value. They also forgot to register the key just before the XOR function. Therefore, their critical path is quite longer. Finally, their solutions do not implement a correct DES that can encrypt every cycle.

To conclude, we propose two efficient and different solutions in terms of space and data rate. Depending on environment constraints, we really believe that one of the two designs should be well appropriate. Especially for high ratio $Throughput/Area$, the 37-stage pipelined solution is very efficient in terms of speed and slices used. Table 2 compares ratio $Throughput/Area$ between 48-stage Xilinx implementation [3, 8] and our 37-stage implementation. We directly see the significative improvement.

DES Version	Xil. 48-stage	Our 37-stage
Slices used	3900	2965
Data rate	15.1 Gbps	21.3 Gbps
Throughput/Area (Mbps/slices)	3.87	7.18

Table 2. Comparisons with Xilinx implementation on Virtex-II-5

5 Our sequential DES and triple-DES implementations

To be space efficient, we propose one DES and triple-DES sequential design, based on the B mathematical description of Figure 2. Instead of keeping a critical path of one LUT (as the best one of our pipelined designs), we prefer to limit the critical path to two LUTs (+ one $F5$ function + one $F6$ function). Indeed, if we limit the critical path to one LUT, we will encounter some critical problems with the routing part of the control signals. For example, to select the input data of a round with MUX functions, we will need 64 identical control signals. An unique signal will be generated with the control part. Then, this signal will be extended to all MUX operators. The propagation delay, of this high fanout signal, will approximatively correspond to one LUT delay. Therefore, we will have an critical path of about two LUTs. So, our sequential DES and triple-DES are designed to have a critical path of two LUTs.

First, we obtain one DES design that encrypts/decrypts, with possible different keys, every 18 cycles with no dead cycles. Our solution proposes almost the same interface as commercial cores [10, 12, 14]. Indeed, they propose modules with 16 cycles for encryption/decryption. Nevertheless,

we get better throughput because of higher work frequency.

Figure 4 explains the key scheduling part for the computation of K1 and K2, needed in the data path. The selection of the input key is carried out with XOR operations. To control the correct functionality, all registers have to be cleared in the right cycles. The *reset_regKIN* and *reset_K1* signals ensure this constraint.

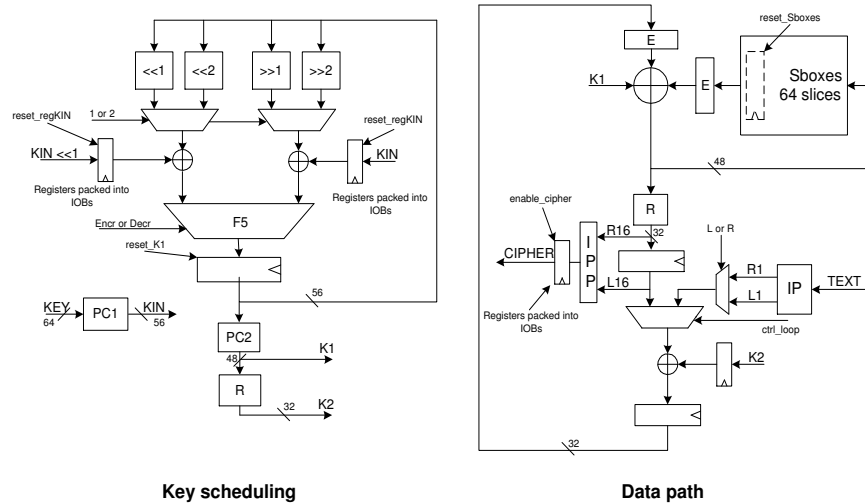


Fig. 4. Our sequential DES encryptor/decryptor.

As shown on the B scheme of Figure 2, due to the irregular new DES description, we also need additional XOR operators in the first and last round. To carry out these operations, we use the data path part where we clear the output registers of S-boxes, in the first and 18th cycles. This is done using the *reset_Sboxes* signal. If we decide to spare these XOR operators, we loose 2 cycles for the encryption/decryption. This is why we get a 18-cycle encryptor/decryptor.

In addition, we mainly focus our attention to maximize the utilization of resources in a slice. For example, we prefer to register the S-boxes outputs instead of other signals because it does not consume additional slices. If we do not this, using these slice registers for other signals will not allowed. Therefore, we will waste these registers. To manage all the signals, we develop a control unit that produces registered signals.

On the other hand, we also obtain one triple-DES design that encrypts/decrypts, with possible different keys, every 18 cycles with no dead cycles. This circuit is obtained using three of our DES sequential implementations. Our solution proposes almost the same interface as commercial cores [10, 11, 13]. Indeed, they mainly propose modules with 16 cycles for encryption/decryption. Nevertheless, we get better throughput because of higher work frequency.

Table 3 summarizes our result for DES and triple-DES sequential designs. In table 4 and 5, we compare our sequential DES and triple-DES with other existing implementations, in term of ratio *Throughput/Area*.

Sequential	DES	triple-DES
LUTs used	365	1103
Registers used	202	672
Slices used	189	604
Latency (cycles)	20	58
Output every (cycles)	1/18	1/18
Freq. in XCV300E-6	176 MHz	165 MHz
Freq. in XC2V1000-5	274 MHz	258 MHz

Table 3. Final results of our sequential DES and triple-DES implementations.

Sequential	DES	CAST	Helion	Ours
Slices used	238	\simeq 450		189
Throughput (Mbps)	816	640		974
Throughput/Area (Mbps/slices)	3.43	1.42		5.15

Table 4. Comparisons with other sequential DES implementations on Virtex-II -5.

Sequential	3-DES	CAST	Gaj et al.	Ours
Slices used	790	614		604
Throughput (Mbps)	668	91 (XCV1000)		917
Throughput/Area (Mbps/slices)	0.85	0.15		1.51

Table 5. Comparisons with other sequential triple-DES implementations on Virtex-II -5.

6 Design strategies to optimize cipher FPGA implementations

FPGAs allow computing in parallel with high work frequencies. Nevertheless, wrong uses of their CLB structure can dramatically increase the speed efficiency as well as the resources used. Accurate slice mapping will permit to get fast and compact designs.

In this section, we set up conclusions for accurate cipher FPGA⁷ designs. Our conclusions are directly based on our DES and triple-DES implementations. We also mention the possible improvement of cipher implementations with a modified structure description. We propose systematic design rules to achieve very fast and compact cipher FPGA implementations, depending on the environment constraints. The following methodology is proposed:

1. Analyze and modify the mathematical cipher description:

Most block ciphers have a regular structure with a number of identical repeated round functions, depending on different subkeys. Subkeys are also generated using a number of identical repeated keyround functions. This regular and repeated round structure allows achieving efficient FPGA designs. Regular cipher structures permit very suitable sequential designs. Nevertheless, original cipher descriptions are not always optimized regarding the speed performance and number of slices used. An FPGA is based on slices composed by two 4-bit input LUTs and two 1-bit registers. Therefore, an optimal way to reduce the LUTs used is to regroup all the logical operations to obtain a minimum number of blocks that take 4-bit input⁸ and give 1-bit output.

Therefore, the aim of this step is to reduce the number of LUTs used as much as possible, modifying the structure description without changing the mathematical functionality. The following methodology has to be applied:

- (a) In round and keyround descriptions, try to move and/or inverse and/or merge blocks as well as possible.

⁷ For Virtex technologies.

⁸ Sometimes 5-bit input.

- (b) Ignore round and keyround descriptions, and try to regroup blocks from different rounds. Try to keep a regular structure as much as possible for sequential designs.
- If no improvements are possible, go to the next point.

2. **Choose your design based on data rate constraints:**

Depending on the data rate constraints, adopt one of the following designs:

- (a) For more than ± 10 Gbps constraints, choose a complete unrolled and pipelined implementation.
- (b) For less than ± 1 Gbps constraints, choose a sequential design.
- (c) For other data rate constraints, choose an uncomplete unrolled and pipelined design or multiple sequential designs.

3. **Fit as well as possible into CLB slices:**

- (a) **For a complete unrolled and pipelined design**, it is important to achieve huge data rates. Therefore, we will adopt designs where we limit the critical path inside one slice (one LUT + one $F5$ function + one $F6$ function at most). Left part of Figure 6 shows some typical slice uses for one slice critical path designs.
- (b) **For sequential design**, it is important to achieve compact design. Due to control signals, it is better to limit the critical path to two LUTs (+ one $F5$ function + one $F6$ function). Indeed, if we limit the critical path to one LUT, we will encounter some critical problems with the routing part of the control signals. For example, to select the input data of a round with MUX functions, we will need β^9 identical control signals. A unique signal will be generated with the control part. Then, this signal will be extended to all MUX operators. The propagation delay, of this high fanout signal, will approximatively correspond to one or more LUT delay.

Therefore, we recommend to limit the control signals to one LUT and routing. The routing part generates additional delays. With an average 60-fanout signal, the delay corresponds to more and less one LUT delay. For the data path part, if we do not use fanout more than three, we can limit the critical path to two LUTs. Figure 5 summarizes our advices.

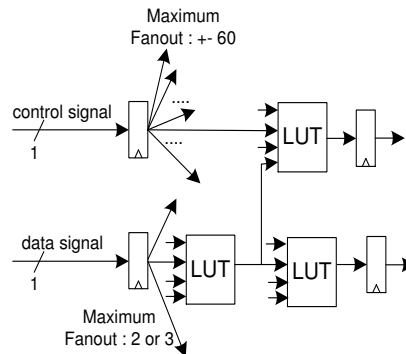


Fig. 5. Recommended architecture for sequential designs.

Figure 6 shows some typical slice uses for two slice critical path designs. We show how to use slice flip flops to register input signals. Figure 6 does not illustrate all the slice configuration possibilities. It is again important to mention that we need to describe the VHDL code at a slice level in order to avoid bad slice mapping and extra slice uses.

⁹ Typically β equals 64, 128 or 256.

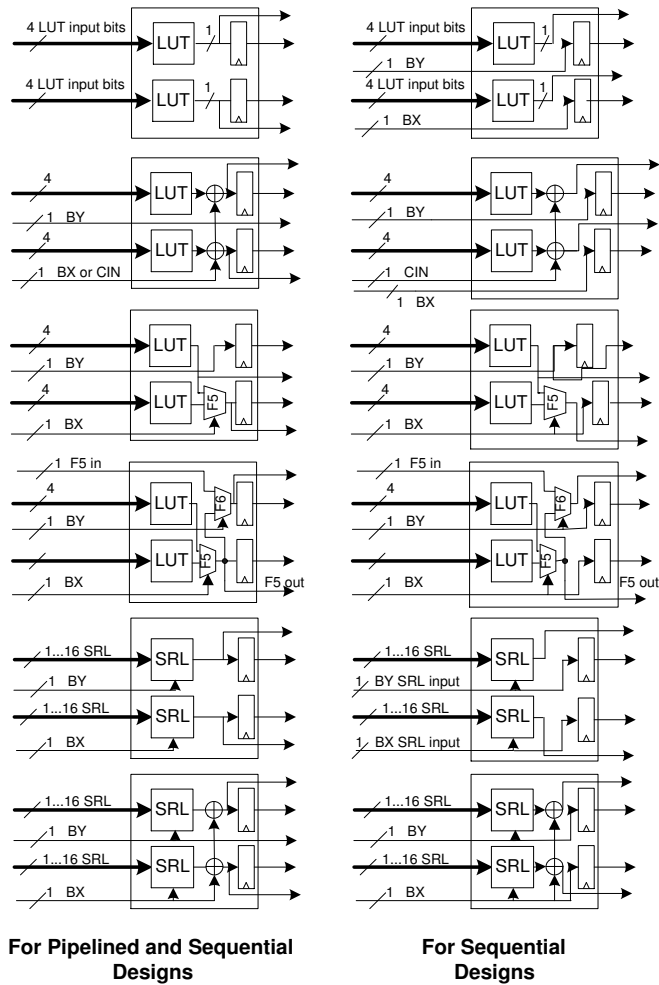


Fig. 6. Slice uses for different designs.

- (c) For an uncomplete unrolled and pipelined design or multiple sequential designs, previous advices are always valid depending on the pipelined or sequential choice.

4. Deal with input and output buffers:

To achieve good frequency results, input and output registers have to be packed into IOBs. In addition, we also recommend to put one more register stage in the input and output paths. This approach allows an additional degree of freedom for the place and route tool.

The previous explained rules set up a methodology to implement cipher algorithms. We do not affirm that this methodology is the most optimized systematic method. Nevertheless, it gives efficient results for DES and triple-DES algorithms.

7 Conclusion

We propose a new mathematical DES description that allows us to achieve optimized FPGA implementations. We get one complete unrolled and pipelined DES, and sequential DES and tripe-DES implementations. Finally, we get the best DES and triple-DES FPGA implementations known nowadays in term of ratio *Throughput/Area*. The fastest DES gives a data rate of 21.3 Gbps (333 MHz). In this design, the plaintext, the key and the mode (encryption or decrytion) can be changed on a cycle-by-cycle basis with no dead cycles. Our sequential triple-DES computes ciphertxts every $\frac{1}{18}$ cycle and produces a data rate of 917 Mbps, using 604 slices.

Based on our DES and triple-DES results, we also set up conclusions for optimized cipher FPGA designs. We recommend to modify the mathematical cipher description. Depending on our data rate constraints, we advice to choose unrolled and pipelined designs or sequential designs. We also show how to fit in slices as well as possible and how to deal efficiently with input and output buffers.

References

1. J.M. Rabaey. *Digital Integrated Circuits*. Prentice Hall, 1996.
2. Xilinx. Virtex 2.5V field programmable gate arrays data sheet. available from <http://www.xilinx.com>.
3. Xilinx, V. Pasham and S. Trimberger. High-Speed DES and Triple DES Encryptor/Decryptor. available from <http://www.xilinx.com/xapp/xapp270.pdf>, Aug 2001.
4. B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., second edition, 1996.
5. National Bureau of Standards. *FIPS PUB 46*, The Data Encryption Standard. U.S. Departement of Commerce, Jan 1977.
6. FreeIP. <http://www.free-ip.com/DES/index.html>.
7. C. Patterson. High performance DES encryption in Virtex FPGAs using Jbits. In *Proc. of FCCM'01*, IEEE Computer Society, 2000.
8. S. Trimberger, R. Pang and A. Singh. A 12 Gbps DES encryptor/decryptor core in an FPGA. In *Proc. of CHES'00*, LNCS, pages 156–163. Springer, 2000.
9. M. Davio, Y. Desmedt, M. Fossprez, R. Govaerts, J. Hulsbosch, P. Neutjens, P. Piret, J.J. Quisquater, J. Vandewalle and P. Wouters. Analytical Characteristics of the DES. In David Chaum, editor, *Advances in Cryptology - Crypto '83*, pages 171–202, Berlin, 1983. Springer-Verlag.
10. Helion Technology. High Performance DES and Triple-DES Core for XILINX FPGA. available from <http://www.heliontech.com>.
11. CAST, Inc. Triple DES Encryption Core. available from <http://www.cast-inc.com>.
12. CAST, Inc. DES Encryption Core. available from <http://www.cast-inc.com>.
13. inSilicon. X_3 DES Triple DES Cryptoprocessor. available from <http://www.insilicon.com>.
14. inSilicon. X_DES Cryptoprocessor. available from <http://www.insilicon.com>.
15. P.Chodowiec, K. Gaj, P. Bellows and B. Schott. Experimental Testing of the Gigabit IPsec-Compliant Implementations of RIJNDAEL and Triple DES Using SLAAC-1V FPGA Accelerator Board. In *Proc. of ISC 2001: Information Security Workshop*, LNCS 2200, pp.220-234, Springer-Verlag.
16. J.P. Kaps and C. Paar. Fast DES Implementations for FPGAs and Its Application to a Universal Key-Search Machine. In *Proc. of SAC'98 : Selected Areas in Cryptography*, LNCS 1556, pp. 234-247, Springer-Verlag.
17. G. Rouvroy, FX. Standaert, JJ. Quisquater, JD. Legat. Efficient Uses of FPGA's for Implementations of DES and its Experimental Linear Cryptanalysis. Accepted for publication on April 2003 in IEEE Transactions on Computers, Special CHES Edition.