High Throughput Implementations of the RC6 Block Cipher Using Virtex-E and Virtex-II Devices

Jean-Luc Beuchat

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

High Throughput Implementations of the RC6 Block Cipher Using Virtex-E and Virtex-II Devices

Jean-Luc Beuchat

No 4495

July 2002

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4495--FR+ENG

High Throughput Implementations of the RC6 Block Cipher Using Virtex-E and Virtex-II Devices

Jean-Luc Beuchat

Thème 2 — Génie logiciel
et calcul symbolique
Projet Arénaire

Rapport de recherche n ° 4495 — July 2002 — 13 pages

Abstract: This short paper is devoted to the study of effective hardware architectures for the RC6 block cipher using Virtex-E and Virtex-II FPGA devices. The key point of the implementation is the design of an arithmetic operator computing $f(X) = (X(2X+1)) \bmod 2^w$. Significant speed and area improvements are obtained by taking full advantage of the small multiplier blocks available in Virtex-II devices.

Key-words: FPGA, RC6 block cipher, computer arithmetic, cryptography.

(Résumé : tsvp)

This text is also available as a research report of the Laboratoire de l'Informatique du Parallélisme `http://www.ens-lyon.fr/LIP`.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (France)
Téléphone : 04 76 61 52 00 - International : +33 4 76 61 52 00
Télécopie : 04 76 61 52 52 - International : +33 4 76 61 52 52

Implantations haut débit de l'algorithme de chiffrage par blocs RC6 sur des circuits Virtex-E et Virtex-II

Résumé : Cet article est consacré à l'étude d'architectures performantes pour l'algorithme de chiffrage par blocs RC6 à l'aide de circuits FPGA des familles Virtex-E et Virtex-II. La principale difficulté consiste à implanter efficacement un opérateur calculant $f(X) = (X(2X+1)) \bmod 2^w$. Des gains importants en surface et en vitesse sont obtenus en utilisant les petits blocs de multiplication disponibles sur les circuits Virtex-II.

Mots-clé : FPGA, algorithme de chiffrage par blocs, RC6, arithmétique des ordinateurs, cryptographie.

In 1997, the National Institute of Standards and Technology (NIST) initiated a process to specify a new symmetric-key encryption algorithm capable of protecting sensitive data. RSA Laboratories submitted RC6 [7] as a candidate for this Advanced Encryption Standard (AES). NIST announced fifteen AES candidates at the First AES Candidate Conference (August 1998) and solicited public comments to select five finalist algorithms (August 1999): MARS, RC6, Rijndael, Serpent, and Twofish.

Though the algorithm Rijndael was eventually selected, RC6 remains a good choice for security applications and is also a candidate for the NESSIE project, the NP 18033 project, and the Cryptrec project initiated by the Information-technology Promotion Agency in Japan.

A version of RC6 is more exactly specified as RC6-$w/r/b$, where the parameters w, r, and b respectively express the word size (in bits), the number of rounds, and the size of the encryption key (in bytes). Since all actual implementations are targeted at $w = 32$ and $r = 20$, we use RC6 as shorthand to refer to RC6-32/20/b. A key schedule generates $2r + 4$ words (w bits each) from the b-bytes key provided by the user (see [7] for details). These values (called round keys) are stored in an array $S[0, \ldots, 2r + 3]$ and are used in both encryption and decryption. The encryption algorithm involves four operations:

- Integer addition modulo 2^w (denoted by $X \boxplus Y$).
- Bitwise exclusive or of two w-bit words (denoted by $X \oplus Y$).
- Computation of $f(X) = (X(2X + 1)) \bmod 2^w$, where X is a w-bit integer.
- Rotation of the w-bit word X to the left by an amount given by the $\log_2 w$ least significant bits of Y (denoted by $X \lll Y$).

Note that the decryption process requires moreover integer subtraction modulo 2^w and rotation to the right. As the algorithm is similar to encryption, we will not consider it here.

In this paper, we study effective hardware architectures of this block cipher using Virtex-E and Virtex-II field programmable gate arrays (FPGAs). In section 1, we investigate various implementations of $f(X)$ and show that the choice of an algorithm depends on the target FPGA family. We also describe architectures of RC6 processors. Section 2 digests our main results and compare them with recent works on RC6.

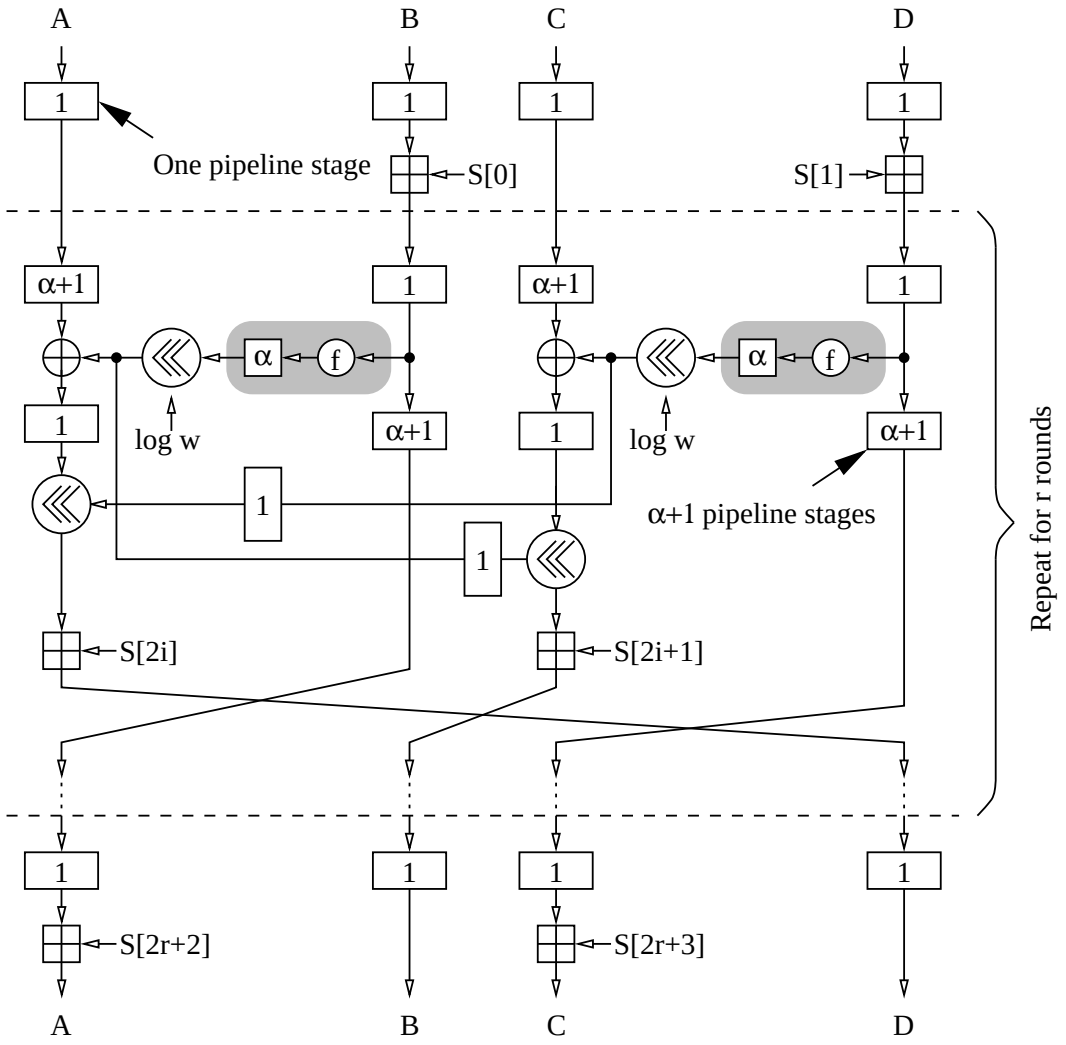


Figure 1: Encryption with RC6.

1 Hardware implementation

Designing an efficient operator dedicated to $f(X)$ is the key point of the hardware implementation and depends on the target FPGA resources. After a brief overview of useful features of Virtex-E and Virtex-II devices, we describe and evaluate three

algorithms computing $f(X)$. We propose finally some architectures of a RC6 processor.

1.1 Some Features of the Virtex-E and Virtex-II Devices

Virtex-E and Virtex-II configurable logic blocks (CLBs) provide functional elements for synchronous and combinatorial logic. Each CLB includes respectively two (Virtex-E) or four (Virtex-II) slices containing basically two 4-input look-up tables (LUT), two storage elements, and fast carry logic dedicated to addition and subtraction.

A Virtex-II device also embeds many 18×18 two's complement multipliers (the MULT18x18 blocks), each of them supporting two input ports 18-bit signed or 17-bit unsigned wide. Furthermore, each multiplier has an internal pipeline stage. Surprisingly, this feature is poorly documented in the Virtex-II data sheet and synthesis tools seem unable to automatically deal with it. The MULT18x18S component, available in the Virtex-II library of Synplify Pro, allows us to write VHDL multipliers that take advantage of this pipeline characteristic.

1.2 Dedicated operators for $f(X) = (X(2X+1)) \bmod 2^w$

1.2.1 Algorithm 1

An artless approach consists in writing the VHDL code depicted by Figure 2. When the size of the operands is strictly greater than 17, tools like Synplify use the well-known divide-and-conquer approach (see for example [6]) in order to synthesize unsigned multipliers for the Virtex-II family. Consequently, Synplify allocates three MULT18x18 blocks to carry out the product $(X(2X+1)) \bmod 2^w$ (Figure 3).

1.2.2 Algorithm 2

Consider the problem of computing $f(X) = (X(2X+1)) \bmod 2^w$ when X is a 8-bit unsigned integer. As shown in Figure 4, the partial products can be significantly simplified before performing their addition using the identities $x_i x_i = x_i$ and $x_i x_j + x_j x_i = 2x_i x_j$. Finally, based on the relation $x_i x_j + x_i = 2x_i x_j + x_i \bar{x}_j$ we remove $x_3 x_2$ and x_2 from the leftmost column and replace them by $x_3 \bar{x}_2$. As $f(X)$ is computed modulo 2^8, we ignore the term $2x_3 x_2$.

```
entity rc6_f is
  port (
    X : in  std_logic_vector (31 downto 0);
    Q : out std_logic_vector (31 downto 0));
end rc6_f;
architecture behavioral of rc6_f is
  signal d0 : std_logic_vector (31 downto 0);
  signal d1 : std_logic_vector (31 downto 0);
  signal p  : std_logic_vector (63 downto 0);
begin -- behavioral
  d0 <= X;
  d1 <= X (30 downto 0) & '1';
  p  <= d0 * d1;
  Q  <= p (31 downto 0);
end behavioral;
```

Figure 2: Algorithm 1: naive implementation of $f(X)$.

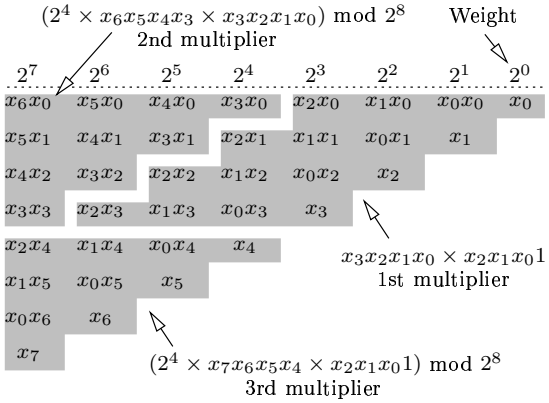


Figure 3: Synthesizing the code illustrated by Fig. 2 with the divide-and-conquer strategy; in this example $w = 8$.

Let us formalize the algorithm sketched in this example. If w is even, the computation of $f(X)$ involves the addition of $\frac{w}{2}$ partial products PP_i defined by:

$$\mathrm{PP}_i = \begin{cases} \displaystyle\sum_{j=0}^{w-1} x_j 2^j & \text{if } i = \frac{w}{2} - 1 \\ x_{i+1}\bar{x}_i 2^{w-1} + x_i 2^{w-3} & \text{if } i = \frac{w}{2} - 2 \\ \displaystyle\sum_{j=2i+3}^{w-1} \left(x_{j-i-2} x_i 2^j + x_i 2^{2i+1} \right) & \text{otherwise} \end{cases} \tag{1}$$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
x_6	x_5	x_4	x_3	x_2	x_1	x_0	1
x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
x_6x_0	x_5x_0	x_4x_0	x_3x_0	x_2x_0	x_1x_0	x_0x_0	x_0
x_5x_1	x_4x_1	x_3x_1	x_2x_1	x_1x_1	x_0x_1	x_1	0
x_4x_2	x_3x_2	x_2x_2	x_1x_2	x_0x_2	x_2	0	0
x_3x_3	x_2x_3	x_1x_3	x_0x_3	x_3	0	0	0
x_2x_4	x_1x_4	x_0x_4	x_4	0	0	0	0
x_1x_5	x_0x_5	x_5	0	0	0	0	0
x_0x_6	x_6	0	0	0	0	0	0
x_7	0	0	0	0	0	0	0
x_5x_0	x_4x_0	x_3x_0	x_2x_0	x_1x_0	0	x_0	0
x_4x_1	x_3x_1	x_2x_1	0	x_1	0	0	0
$\mathbf{x_3x_2}$	0	x_2	0	0	0	0	0
$\mathbf{x_3}$	0	0	0	0	0	0	0
x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
x_5x_0	x_4x_0	x_3x_0	x_2x_0	x_1x_0	0	x_0	0
x_4x_1	x_3x_1	x_2x_1	0	x_1	0	0	0
$\mathbf{x_3\bar{x}_2}$	0	x_2	0	0	0	0	0
x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0

Figure 4: Algorithm 2: computation of $f(X) = (X(2X + 1)) \bmod 2^8$ with AND gates and a few carry-propagate adders.

Equation (1) allows us to automatically generate a VHDL operator for any even value of w. A proof of this algorithm is available in [1].

1.2.3 Algorithm 3

Remember that the divide-and-conquer requires three embedded multipliers in order to implement $f(X)$ on a Virtex-II device. Consequently, this method leads to a small operator in terms of slice number. Though algorithm 2 allows simplifications of the partial products, it doesn't make use of the Virtex-II embedded multipliers and generates larger circuits. We propose here an algorithm combining the benefits of these two methods. Consider again the computation of $f(X)$ with $w = 8$ (Figure 5). The trick consists in performing the *rectangular* multiplication $(2 \cdot X_{3:0} + 1)X_{3:0}$, where $X_{q:p}$ denotes $\sum_{i=p}^{q} x_i 2^i$. Examine now the remaining terms of the partial products:

- The product $X_{5:4}X_{1:0}$ appears two times; therefore, we compute it with a single embedded multiplier and left-shift the result.

- As the weight of $2x_0x_6$ and $2x_2x_4$ is strictly greater than 2^7, we discard these terms.
- Finally, we add the two products and the higher half of X modulo 2^8.

For $w = 32$, this method requires a 16×17 multiplication, a 14×14 multiplication, and two additions. Formally, this algorithm is described by:

$$\begin{aligned}
&(X(2X+1)) \bmod 2^w \\
&= \left(\left(\sum_{i=0}^{\frac{w}{2}-1} x_i 2^i \cdot \left(\sum_{i=0}^{\frac{w}{2}-1} x_i 2^{i+1} + 1 \right) \right) \bmod 2^w + \right. \\
&\quad \left(2 \cdot \sum_{i=0}^{\frac{w}{2}-3} x_i 2^i \cdot \sum_{i=\frac{w}{2}}^{w-3} x_i 2^{i+1} \right) \bmod 2^w + \\
&\quad \left. \sum_{i=\frac{w}{2}}^{w-1} x_i 2^i \right) \bmod 2^w. \qquad (2)
\end{aligned}$$

The proof of this equality is straightforward and will not be addressed here.

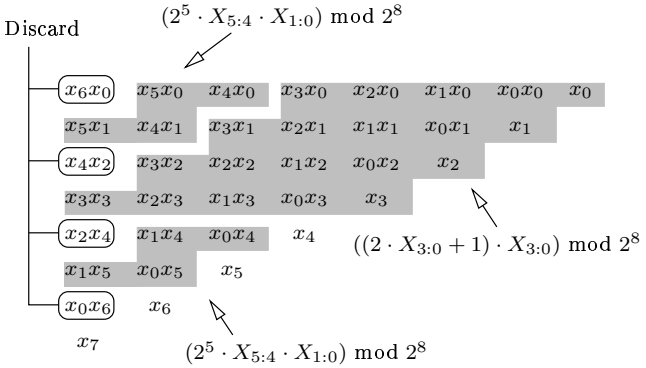


Figure 5: Algorithm 3: computation of $f(X) = (X(2X+1)) \bmod 2^8$ with two embedded multipliers and two carry-propagate adders.

1.3 Comparison of the algorithms

We have written a C library which generates VHDL descriptions of the operators described above. The VHDL code was synthesized using Synplify Pro 7.0.3 and

implemented on several Virtex-E and Virtex-II devices using Xilinx Alliance Series 4.1.03i. Table 1 summarizes the results of some experiments.

Device	Algorithm 1			Algorithm 2		
	Slices	MULT18x18	Delay [ns]	Slices	MULT18x18	Delay [ns]
XCV1000E-6	288	–	18.8	181	–	16.5
XC2V1000-6	18	3	11.0	193	0	10.9

Device	Algorithm 3		
	Slices	MULT18x18	Delay [ns]
XCV1000E-6	233	–	19.9
XC2V1000-6	17	2	10.7

Table 1: Comparison of several $f(X)$ operators for Virtex-E and Virtex-II devices.

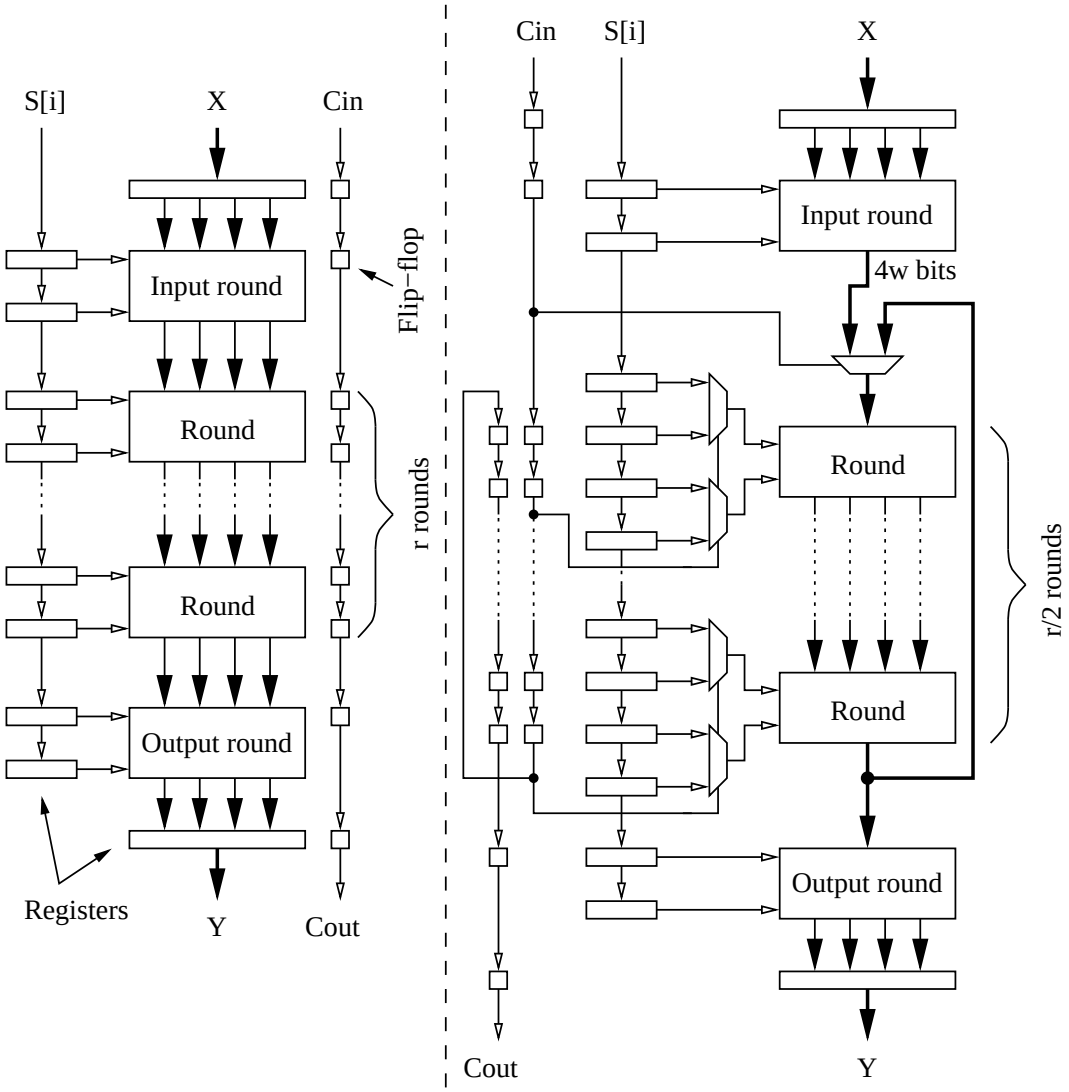
Algorithm 2 is the best choice for the Virtex-E family. On a Virtex-II device, algorithm 3 leads to the smallest and fastest circuit. As the complete RC6 pipeline involves 40 $f(X)$ operators, this method saves up to 40 MULT18x18 blocks compared with algorithm 1. Note that our VHDL generators allow to pipeline the operators.

1.4 Architecture of the RC6 processor

Now that we have efficient $f(X)$ operators, the design of a cipher round is straightforward: integer addition modulo 2^w takes advantage of fast-carry logic, a barrel shifter achieves the rotations, and LUTs implement bitwise exclusive or. In order to shorten the critical path, we insert registers as illustrated by Figure 1. Note that the depth of the pipeline depends on the latency α of the $f(X)$ operator ($\alpha = 0$ or $\alpha = 1$).

Figure 6a shows a first architecture consisting of the input round, $r = 20$ rounds, and the output round. The $2r + 4$ round keys are stored in w-bit registers. A shift register implements the control unit. A token indicates the validity of the data on the corresponding pipeline stage. However, this approach involves 40 $f(X)$ operators and will only fit into large FPGAs.

Figure 6b depicts an iterative architecture with partial loop unrolling and pipelining. The circuit implements k rounds (k is an integer divisor of the total number of rounds r), the input round, and the output round. This methodology was inspired by hardware implementations of the IDEA block cipher described in [9] and [5]. The control unit requires a minor update in that the token addresses now the round key memory. At the price of a lower throughput, this approach permits the implementation of RC6 in smaller and cheaper devices.



(a) Full loop unrolling and pipelining (b) Partial loop unrolling and pipelining

Figure 6: Two architectures of a RC6 processor.

2 Results

A VHDL generator allows us to specify the parameters w and r, the latency α of the $f(X)$ operator and the number of rounds physically implemented on the FPGA.

This tool is useful to study the trade-off between the choice of a device and the throughput. Table 2 summarizes some results. In all our experiments, the $f(X)$ operator has an internal pipeline stage and the latency of a round is therefore equal to three. XC2V500 and XC2V250 devices have not enough I/Os to deal with 128-bit words. Our solution consists in defining 64-bit input and output ports and spending two clock cycles for data transmission.

Device	Algo	Rounds (# iterations)	Slices	MULT 18x18S	Delay [ns]	Throughput [Gb/s]
XCV1600E-8	2	20 (1)	14110 (90%)	–	13.2	9.7
XCV1000E-8	2	10 (2)	7157 (46%)	–	13.2	4.8
XC2V3000-6	3	20 (1)	8554 (59%)	80 (83%)	8.4	15.2
XC2V3000-6	2	10 (2)	7456 (52%)	0 (0%)	13.3	4.8
XC2V1000-6	3	10 (2)	4391 (85%)	40 (100%)	8.6	7.4
XC2V500-6	3	5 (4)	2365 (76%)	20 (62%)	8.2	3.9
XC2V250-6	3	4 (5)	1534 (99%)	16 (66%)	8.9	2.8

Table 2: Characteristics of our RC6 processors.

Table 3 digests results published by some other researchers. A NSA team has implemented RC6 with semi-custom ASICs based on a 0.5 μm CMOS library [8]. Using the architecture depicted by Figure 6a with a pipeline stage between two consecutive rounds and algorithm 1 to compute $f(X)$, the NSA team reports a throughput of 2.2 Gbits/s.

Gaj et al. have proposed an architecture similar to Figure 6 [3][2]. The main differences lie in the $f(X)$ operator and in the number of pipeline stages per cipher round (3 in our case versus 28 in their system). However, four XCV1000-6 devices are required to implement the algorithm with full loop unrolling. While the throughput is close to ours, this solution is more expensive and requires a larger area on the PCB.

J.-O. Haenni has studied software implementations of RC6 for Itanium and G4 processors [4]. The code was written in assembly language to benefit from the potential of multimedia instructions. FPGA and ASIC approaches clearly outperform optimized software solutions.

3 Conclusions

In this paper, improved architectures of the RC6 block cipher for Virtex-E and Virtex-II FPGAs have been described. Our dedicated VHDL generators provide a

Reference	**Technology**	**Throughput [Gb/s]**
NSA team [8]	0.5 μm CMOS	2.2
Gaj and Chodowiec [3]	XCV1000-6 (4 devices)	13.1
Haenni [4]	Itanium (733 MHz) G4 (450 MHz)	0.33 0.47
Best solution of this paper (see table 2)	XC2V3000-6	15.2

Table 3: Results of some other researchers.

wide parameter space exploration: choice of the dedicated $f(x)$ operator, number of pipeline stages per round, and number of rounds physically implemented on the FPGA. Significant speed and area improvements are obtained by taking full advantage of Virtex-II MULT18x18 dedicated multiplier blocks.

Acknowledgments

The author would like to thank the "Ministère Français de la Recherche" (grant # 1048 CDR 1 "ACI jeunes chercheurs"), the Swiss National Science Foundation, and the Xilinx University Program for their support.

References

[1] J.-L. Beuchat. *Etude et conception d'opérateurs arithmétiques optimisés pour circuits programmables.* PhD thesis, Swiss Federal Institute of Technology Lausanne, 2001. Available from http://www.ens-lyon.fr/~jlbeucha.

[2] P. Chodowiec, P. Khuon, and K. Gaj. Fast Implementations of Secret-Key Block Ciphers Using Mixed Inner- and Outer-Round Pipelining. In *Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 94–102, 2001.

[3] K. Gaj and P. Chodowiec. Fast implementation and fair comparison of the final candidates for Advanced Encryption Standard using Field Programmable Gate Arrays. In *Proc. RSA Security Conf. - Cryptographer's Track*, pages 84–99. Springer-Verlag, 2001. Available from http://ece.gmu.edu/crypto/publications.htm.

[4] J.-O. Haenni. *Architecture EPIC et jeux d'instructions multimédias pour applications cryptographiques*. PhD thesis, Swiss Federal Institute of Technology Lausanne, 2002.

[5] E. Mosanya, C. Teuscher, H. F. Restrepo, P. Galley, and E. Sanchez. CryptoBooster: A Reconfigurable and Modular Cryprographic Coprocessor. In C. K. Koc and C. Paar, editors, *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems, CHES'99, Worcester, MA*, volume 1717 of *Lecture Notes in Computer Science*, pages 246–256. Springer-Verlag, Berlin, Heidelberg, 1999.

[6] B. Parhami. *Computer Arithmetic*. Oxford University Press, 2000.

[7] R.L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin. The RC6 Block Cipher. Available from http://www.rsasecurity.com/rsalabs/rc6, 1998.

[8] B. Weeks, M. Bean, T. Rozylowicz, and C. Ficke. Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms. Technical report, National Security Agency, 2000. Available from http://csrc.nist.gov/encryption/aes/round2/r2anlsys.htm.

[9] R. Zimmermann, A. Curiger, H. Bonnenberg, H. Kaeslin, N. Felber, and W. Fichtner. A 177 Mbit/s VLSI Implementation of the International Data Encryption Algorithm. *IEEE Journal of Solid-State Circuits*, 29(3):303–307, March 1994.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399