FPGA Implementations of the RC6 Block Cipher

Jean-Luc Beuchat

Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon,
46, Allée d'Italie, F–69364 Lyon Cedex 07,
Jean-Luc.Beuchat@ens-lyon.fr

Abstract. RC6 is a symmetric-key algorithm which encrypts 128-bit plaintext blocks to 128-bit ciphertext blocks. The encryption process involves four operations: integer addition modulo 2^w, bitwise exclusive or of two w-bit words, rotation to the left, and computation of $f(X) = (X(2X+1)) \bmod 2^w$, which is the critical arithmetic operation of this block cipher. In this paper, we investigate and compare four implementations of the $f(X)$ operator on Virtex-E and Virtex-II devices. Our experiments show that the choice of an algorithm is strongly related to the target FPGA family. We also describe several architectures of a RC6 processor designed for feedback or non-feedback chaining modes. Our fastest implementation achieves a throughput of 15.2 Gb/s on a Xilinx XC2V3000-6 device.

1 Introduction

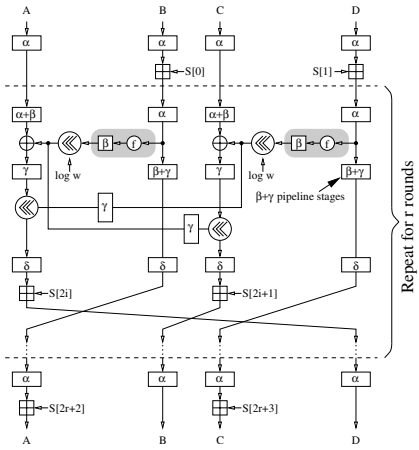
In 1997, the National Institute of Standards and Technology (NIST) initiated a process to specify a new symmetric-key encryption algorithm capable of protecting sensitive data. RSA Laboratories submitted RC6 [9] as a candidate for this Advanced Encryption Standard (AES). NIST announced fifteen AES candidates 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 NP 18033 project (via the Swedish ISO/IEC JTC 1/SC 27 member body[1]) and the Cryptrec project initiated by the Information-technology Promotion Agency in Japan[2].

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 [9] for details). These values (called round

[1] http://www.din.de/ni/sc27

[2] http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html

P.Y.K. Cheung et al. (Eds.): FPL 2003, LNCS 2778, pp. 101–110, 2003.



(a) Pipelined datapath of RC6.

```
entity rc6_f is
  port (
    D : 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 <= D;
  d1 <= D (30 downto 0) & '1';
  p  <= d0 * d1;
  Q  <= p (31 downto 0);
end behavioral;
```

(b) Straightforward implementation of $f(X)$.

Fig. 1. Encryption with RC6.

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 (Figure 1a):

- 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 several hardware architectures of RC6 using Virtex-E and Virtex-II field programmable gate arrays (FPGA). Virtex-E and Virtex-II Configurable Logic Blocks (CLB) provide functional elements for synchronous and combinational logic. Each CLB includes respectively two (Virtex-E) and four (Virtex-II) slices containing basically two 4-input look-up tables (LUT), two storage elements, fast carry logic dedicated to addition and subtraction, and two dedicated AND gates (referred to as MULT_AND) which improve the efficiency of multiplier implementation. Furthermore, Virtex-II devices embed many 18-bit $\times$ 18-bit multiplier blocks (also referred to as MULT18x18 blocks) supporting two independent dynamic data input ports: 18-bit signed or 17-bit unsigned. Arithmetic operators dedicated to FPGAs should therefore involve such building blocks.

This paper is organized as follows: Section 2 describes several architectures of a RC6 processor. We then investigate various implementations of $f(X)$ and show that the choice of an algorithm depends on the target FPGA family (Section 3). Finally, Section 4 digests our main results and compare them with recent works on RC6.

2 Architecture of a RC6 Processor

RC6 encrypts plaintext in fixed-size 128-bit blocks. However, messages will often exceed 128 bits and a simple solution, known as Electronic Codebook (ECB) mode, consists in partitioning the plaintext into 128-bit blocks and encrypting each independently. This ECB mode has a major drawback in that identical ciphertext blocks imply identical plaintext blocks and is therefore inadvisable if the secret key is reused for more than one message. More sophisticated chaining modes bring a solution to this problem. For instance, in the Cipher Block Chaining (CBC) mode, a feedback mechanism causes the jth ciphertext block to depend on the first j plaintext blocks and an n-bit initialization vector. Since the entire dependency on preceding blocks is contained in the previous ciphertext block [6], all blocks must be processed sequentially (CBC decryption can however be performed in parallel). This property forbids to pipeline the computation path and implies a slightly different hardware architecture of the block cipher with a lower throughput. The counter (CTR) mode, a non-feedback mode described for example in [3], could remedy the situation if it becomes a standard as recommended in [5]. It is also possible to pipeline the processor in feedback modes if we accept the decomposition of the data stream into d separately encrypted messages, where d is the pipeline depth [11]. Also note that RC6 involves forty 32-bit $\times$ 32-bit unsigned multipliers. The implementation of the 20 rounds is therefore only possible on rather large and expensive FPGAs.

Consequently, the hardware architecture of a RC6 processor depends as well on the required chaining mode and the target FPGA. We adopt here a design methodology initially proposed for the hardware implementation of the IDEA block cipher [7, 11]: the simplest RC6 processor contains a single round, the input round, and the output round (Figure 2a). This architecture is tailored to feedback chaining modes: a single plaintext block is encrypted at a time and we can provide a new input block after 21 clock cycles.

Assume now that a non-feedback chaining mode is required or that the data stream is decomposed into several separately encrypted messages. In order to shorten the critical path, each round has a parametric number of internal pipeline stages (parameters α, β, γ, and δ on Figure 1a). Figure 2b 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. Finally, Figure 2c illustrates an architecture with full loop unrolling dedicated to high throughput implementations of the RC6 block cipher.

In addition to the RC6 computation path, each processor contains a subkey memory implemented on CLBs and a control unit. The latter simply consists in a token associated with each plaintext block. This token indicates the validity of the data and selects the correct subkeys in iterative architectures. We have written a C program which generates a structural VHDL description of such a RC6 processor according to several parameters (partial or full loop unrolling, inner-round pipeline stages, and outer-round pipeline stages). Some examples are freely available at `http://www.ens-lyon.fr/~jlbeucha`.

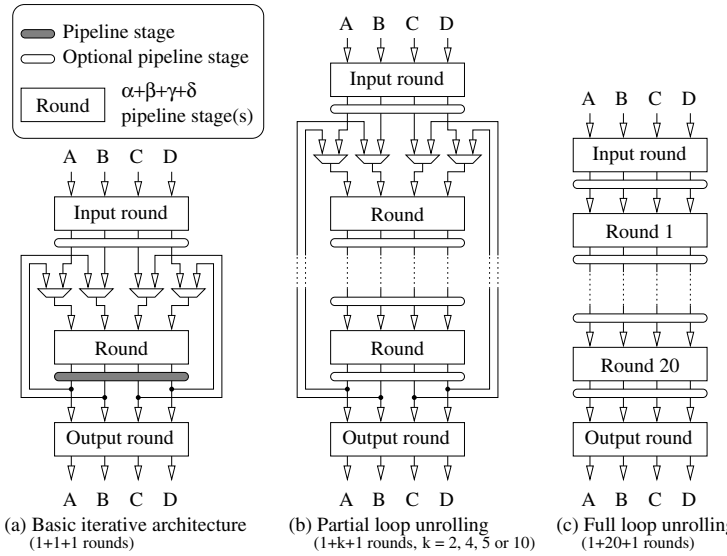


Fig. 2. Some architectures of a RC6 processor.

3 Computation of $f(X)$

The computation of $f(X)$ is the critical arithmetic operation of the block cipher. Therefore, both area and delay of a RC6 processor are closely related to the hardware operator carrying out $f(X) = (X(2X+1)) \bmod 2^w$. In this section, we investigate and compare a method involving an array of AND gates and carry-propagate adders (CPA), and three algorithms dedicated to FPGA embedding small multiplier blocks. In the following, $X_{q:p}$ denotes $\sum_{i=p}^{q} x_i 2^i$.

3.1 Adder-Based Algorithm

This first algorithm is based on a standard method for squaring described for instance in [8]. Let us consider the problem of computing $f(X)$ when X is a 8-bit unsigned integer. As shown in Figure 3, the partial products can be significantly simplified before performing their addition according to 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 well-known 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 $2x_3 x_2$.

Let us formalize the algorithm sketched out in this example. If w is even, the computation of $f(X)$ involves the addition of $\frac{w}{2}$ partial products PP_i defined as follows[3]:

[3] A proof of correctness is provided in [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
x_3x_2	0	x_2	0	0	0	0	0
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
$x_3\bar{x}_2$	0	x_2	0	0	0	0	0
x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0

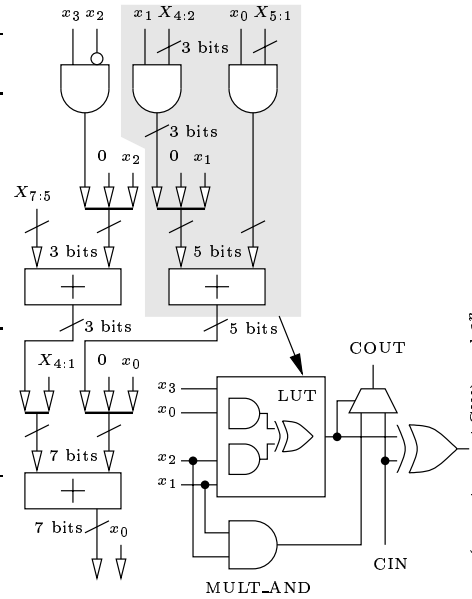


Fig. 3. Algorithm 1: computation of $f(X) = (X(2X+1)) \bmod 2^8$ with AND gates and carry-propagate adders.

$$\mathrm{PP}_i = \begin{cases} \displaystyle\sum_{j=0}^{w-1} x_j 2^j & \text{if } i = \frac{w}{2} - 1, \\ x_{i+1}\bar{x}_i 2^{w-1} + x_i 2^{w-3} & \text{if } i = \frac{w}{2} - 2, \\ x_i 2^{2i+1} + \displaystyle\sum_{j=2i+3}^{w-1} x_{j-i-2} x_i 2^j & \text{if } 0 \le i \le \frac{w}{2} - 3. \end{cases}$$

The above equation allows to automatically generate the VHDL code of the partial product generator for any even w. Synthesis tools are then able to put to good use the MULT_AND gates in order to generate and add partial products using the same logic resources as a simple multioperand tree adder (Figure 3).

3.2 Multiplier-Based Algorithms

A straightforward algorithm reported in [10] consists in writing the VHDL code depicted by Figure 1b. Since $w = 32$ and Virtex-II devices embed 17-bit $\times$ 17-bit unsigned multipliers, commercial tools like Synplify Pro or XST (Xilinx Synthesis Technology) resort to the well-known divide-and-conquer scheme [8] in order to build the operator (Figure 4).

Each $f(X)$ operator based on this scheme involves three multiplier blocks and two carry-propagate adders. Consequently, a RC6 processor with full loop unrolling requires $2r \cdot 3 = 120$ MULT18x18 blocks and fits in a XC2V4000 device. Let us define the lower and higher words of the operand X as follows:

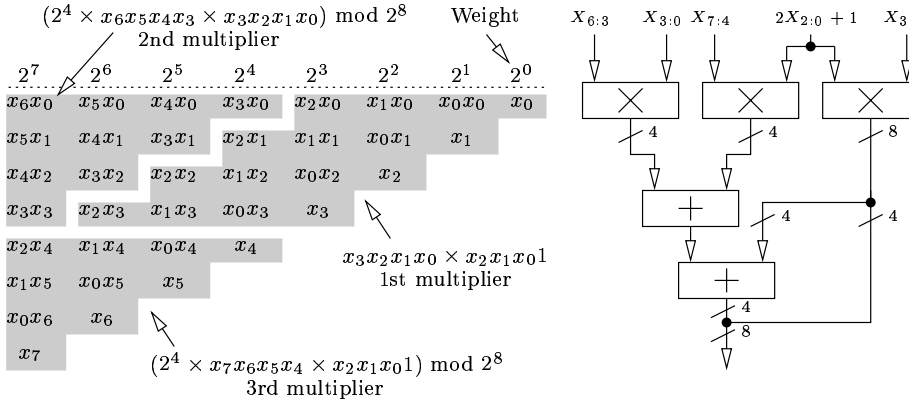


Fig. 4. Algorithm 2: computation of $f(X) = (X \cdot (2X+1)) \bmod 2^w$ with a divide-and-conquer strategy.

$X_L = \sum_{i=0}^{w/2-1} x_i 2^i$ and $X_H = \sum_{i=0}^{w/2-1} x_{n+i} 2^i$. A solution to reduce the amount of 17-bit $\times$ 17-bit unsigned multipliers, and therefore the price of the processor, is to evaluate $f(X)$ according to:

$$\begin{aligned} f(X) &= (2X^2 + X) \bmod 2^w = (2 \cdot (2^{w/2} X_H + H_L)^2 + X) \bmod 2^w \\ &= (2 \cdot (2^w X_H^2 + 2^{1+w/2} X_H X_L + X_L^2) + X) \bmod 2^w \\ &= (2^{2+w/2} X_H X_L + 2X_L^2 + X) \bmod 2^w. \end{aligned}$$

Figure 5 illustrates how this algorithm works. In this example, we assume that $w = 8$ and that 4-bit $\times$ 4-bit multiplier blocks are available. Since $(2 \cdot 2^7 x_0 x_6 + 2 \cdot 2^7 x_2 x_4) \bmod 2^8 = 0$, we can discard these terms. For $w = 32$, this third algorithm involves a 16-bit squarer, a 14-bit $\times$ 14-bit multiplier, a 14-bit CPA, and a 31-bit CPA.

Remember that Virtex-II devices embed 17-bit $\times$ 17-bit unsigned multipliers. In the following, we describe how to put to good use the most significant bit of their input ports in order to further reduce the area of the $f(X)$ operator. Consider again the computation of $f(X)$ with $w = 8$ (Figure 6). The trick consists in performing the *rectangular* multiplication $(2 \cdot X_{3:0} + 1)X_{3:0}$ and allows to replace the $(w-1)$-bit CPA by a $w/2$-bit CPA.

3.3 Comparison of the Four Algorithms

Table 1 summarizes the main characteristics of the four $f(X)$ operators previously described[4]. From these results, we can conclude that:

[4] The VHDL code was synthesized and implemented on Virtex-E and Virtex-II devices with Synplify Pro 7.0.3 and Xilinx Alliance Series 4.1.03i. All input and output signals were routed through the D-type flip-flops available in the Input/Output blocks of Virtex-E or Virtex-II devices. No specific constraints were given to the synthesis tools and it should be possible to improve the results.

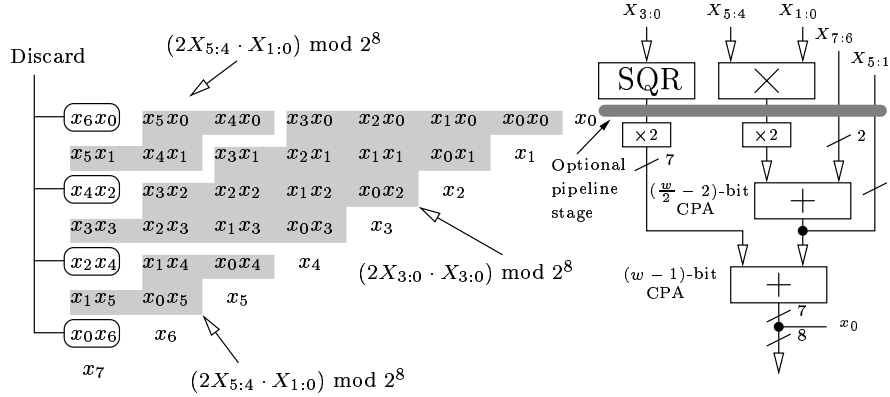


Fig. 5. Algorithm 3: computation of $f(X)$ with a squarer, a multiplier, and two carry-propagate adders.

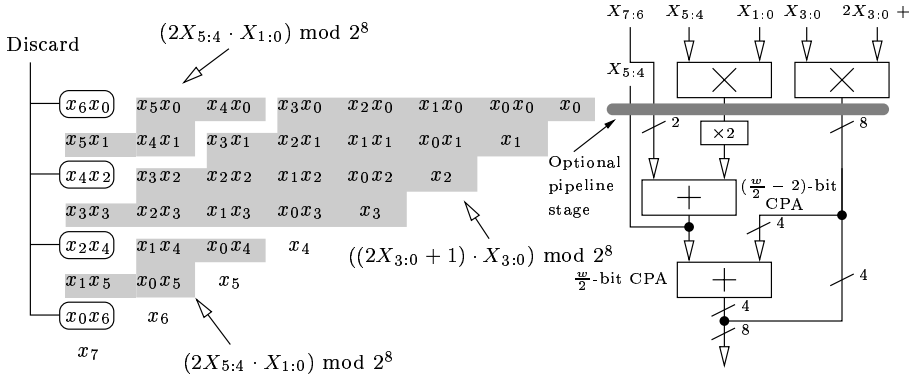


Fig. 6. Algorithm 4: computation of $f(X) = (X(2X+1)) \bmod 2^8$ with two multipliers and two carry-propagate adders.

- On a Virtex-II device, Algorithm 4 leads to the smallest circuit.
- The adder-based operator (Algorithm 1) is the best alternative for the Virtex-E family. This result is not surprising since current synthesis tools are unable to build efficient multipliers from a high-level VHDL description such that shown in Figure 7. We also notice that Algorithm 3 and Algorithm 4 lead to the same circuit area: the fact that both multipliers and adders are now implemented on CLBs explains this result.

In order to improve the frequency of the RC6 processor, our VHDL code generator is able to insert optional pipeline stages in the $f(X)$ operator (parameter β on Figure 1a). For Virtex-II devices, we take advantage of the internal pipeline stage of each MULT18x18 block. Unfortunately, the VHDL coding style depends on both the chosen algorithm and the synthesis tools:

```
entity rc6_f is
  port (
    D   : 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 sqr_q  : std_logic_vector (31 downto 0);
  signal mult_q : std_logic_vector (27 downto 0);
  signal add_q  : std_logic_vector (31 downto 0);
begin  -- behavioral
  sqr_q  <= D (15 downto 0) * D (15 downto 0);
  mult_q <= D (29 downto 16) * D (13 downto 0);
  add_q (17 downto 0)  <= D (17 downto 0);
  add_q (31 downto 18) <= D (31 downto 18) +
                          mult_q (13 downto 0);
  Q (0)           <= add_q (0);
  Q (31 downto 1) <= add_q (31 downto 1) + sqr_q (30 downto 0);
end behavioral;
```

(a) VHDL code for Algorithm 3.

```
entity rc6_f is
  port (
    D   : 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 mult1_q : std_logic_vector (32 downto 0);
  signal mult2_q : std_logic_vector (27 downto 0);
  signal add_q   : std_logic_vector (15 downto 0);
begin  -- behavioral
  mult1_q <= (D (15 downto 0) & '1') * D (15 downto 0);
  mult2_q <= D (29 downto 16) * D (13 downto 0);
  add_q (15 downto 2) <= D (31 downto 18) +
                         mult2_q (13 downto 0);
  add_q (1 downto 0)  <= D (17 downto 16);
  Q (15 downto 0)  <= mult1_q (15 downto 0);
  Q (31 downto 16) <= mult1_q (31 downto 16) + add_q;
end behavioral;
```

(b) VHDL code for Algorithm 4.

Fig. 7. Two VHDL descriptions of the $f(X)$ operator.

Table 1. Comparison of several $f(X)$ operators.

	XCV50E-6		**XC2V40-6**			
	Slices	**Delay [ns]**	**Pipeline**	**Slices**	**Mult. blocks**	**Delay [ns]**
Algorithm 1	**134**	$\sim$ **18**	–	148	–	$\sim$ 12
Algorithm 2	274	$\sim$ 20	–	18	3	$\sim$ 12
Algorithm 3	229	$\sim$ 20	–	25	2	$\sim$ 12
			1 stage	41	2	$\sim$ 8
Algorithm 4	230	$\sim$ 20	–	**17**	**2**	$\sim$ **12**
			1 stage	**25**	**2**	$\sim$ **8**

- For Algorithms 3 and 4, it sometimes suffices to insert a register after each multiplication. For instance, Synplify Pro is able to infer registered multipliers (option `-pipe 1` in the synthesis script). Synthesis tools also provide the designer with libraries containing the basic building blocks of a given FPGA family. It is therefore possible to instantiate a pipelined MULT18x18 block in the VHDL code, instead of expressing the multiplication operator.
- However, if the multiplier does not fit in a single MULT18x18 block, current synthesis tools are unable to simultaneously apply the divide-and-conquer scheme and the retiming algorithm. It is therefore impossible to automatically pipeline the VHDL description of Algorithm 2 depicted on Figure 1. The solution consists here in performing the divide-and-conquer scheme by hand: the VHDL description will then contain three 16-bit $\times$ 16-bit multipliers.

The VHDL code illustrated on Figure 7 leads however to poor results on Virtex-E devices. A structural description of the operator (partial product generation, carry-propagate adders, and registers) gives better results.

4 Implementation Results

Table 2 summarizes the main characteristics of several RC6 processors for Virtex-E and Virtex-II devices. For non-feedback chaining modes, processors with full-

Table 2. Some RC6 processors for Virtex-II and Virtex-E devices. CAD tools: Synplify Pro 7.0.3 and Xilinx Alliance Series 4.1.03i (∗) or ISE 5.1.03i (†).

	Device	Algo	Rounds	Pipeline				Slices	Mult. blocks	Throughput [Gb/s]
				α	β	γ	δ			
Non-feedback chaining modes	XCV2000E-6†	1	1 + 20 + 1	1	2	1	1	19198 (99%)	–	∼ 10.6
	XCV300E-6∗	1	1 + 1 + 1	1	2	1	1	2068 (67%)	–	∼ 0.5
	XC2V3000-6∗	4	1 + 20 + 1	1	1	1	0	8554 (59%)	80 (83%)	∼ 15.2
	XC2V3000-6†	4	1 + 20 + 1	1	1	1	0	10288 (71%)	80 (83%)	∼ 14.2
	XC2V3000-6∗	1	1 + 10 + 1	1	1	1	0	7456 (52%)	0 (0%)	∼ 4.8
	XC2V1000-6∗	4	1 + 10 + 1	1	1	1	0	4391 (85%)	40 (100%)	∼ 7.4
	XC2V500-6∗	4	1 + 5 + 1	1	1	1	0	2365 (76%)	20 (62%)	∼ 3.9
	XC2V250-6∗	4	1 + 4 + 1	1	1	1	0	1534 (99%)	16 (66%)	∼ 2.8
Feedback chaining modes	XCV300E-6∗	1	1 + 1 + 1	0	0	0	0	1709 (55%)	–	∼ 0.16
	XCV300E-6∗	4	1 + 1 + 1	0	0	0	0	1902 (61%)	–	∼ 0.15
	XCV400E-6∗	1	1 + 4 + 1	1	0	0	0	3932 (81%)	–	∼ 0.16
	XC2V1000†	4	1 + 1 + 1	0	0	0	0	1560 (30%)	4 (10%)	∼ 0.29
	XC2V1000†	4	1 + 4 + 1	1	0	0	0	2902 (56%)	16 (40%)	∼ 0.34

loop unrolling achieve high throughputs. The area and the critical path however depend on the synthesis tools: we have obtained better results with Symplify Pro 7.0.3 and Xilinx Alliance 4.1.03i than with ISE 5.1.03i. Also note that 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.

The basic iterative architecture (Figure 2a) seems to be the best one for feedback chaining modes: it requires less slices than systems with partial loop unrolling and achieves the same throughput. As the rounds are combinational, the critical path increases and we obtain very low encryption rates.

A NSA team has implemented RC6 with semi-custom ASICs based on a 0.5 μm CMOS library [10]. Using the architecture depicted by Figure 2c with a pipeline stage between two consecutive rounds and Algorithm 2 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 2 [2, 4]. 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 and to achieve a throughput of 13.1 Gbits/s. While the throughput is close to ours, this solution is more expensive and requires a more complex PCB.

5 Conclusions

In this paper, several architectures of the RC6 block cipher for Virtex-E and Virtex-II FPGAs have been described. We have also investigated four algorithms computing $f(X)$, which is the critical arithmetic operation of the block cipher. Our experiments indicate that both the choice of an algorithm and the VHDL

coding style are strongly related to the target FPGA family. Our VHDL generator allows to quickly explore a wide parameter space and to determine the best architecture for a given set of constraints (feedback or non-feedback chaining mode, FPGA device, ...).

Acknowledgments

The author would like to thank the "Ministère Français de la Recherche", 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.
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. M. Dworkin. Recommandation for Block Cipher Modes of Operation, 2001. NIST Special Publication 800-38A.
4. 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 at `http://ece.gmu.edu/crypto/publications.htm`.
5. H. Lipmaa, P. Rogaway, and D. Wagner. Comments to NIST concerning AES Modes of Operations: CTR-Mode Encryption.
6. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.
7. 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, 1999.
8. B. Parhami. *Computer Arithmetic.* Oxford University Press, 2000.
9. R.L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin. The RC6 Block Cipher, 1998.
10. 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.
11. 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, 1994.