The FPGA Implementation of the RC6 and CAST-256 Encryption Algorithms

M. Riaz and H.M. Heys

Electrical and Computer Engineering
Memorial University of Newfoundland
St. John's, Newfoundland, Canada A1B 3X5
Email: {mohsin, howard}@engr.mun.ca

Abstract

The National Institute of Standards and Technology (NIST) in the U.S. has initiated a process to develop an Advanced Encryption Standard (AES) specifying a private-key encryption algorithm based on a 128-bit block size as a replacement for the Data Encryption Standard (DES). In this paper, we investigate the efficiency of two AES candidates, RC6 and CAST-256, from the hardware implementation perspective with Field Programmable Gate Arrays (FPGAs) as the target technology. Our analysis and synthesis studies of the ciphers suggest that it would be desirable for FPGA implementations to have a simpler cipher design that makes use of simpler operations that not only possess good cryptographic properties, but also make the overall cipher design efficient from the hardware implementation perspective.

1 Introduction

In the recent years, there has been a great need for much improved techniques of securely transmitting and storing information. The field of cryptography encompasses some of these requirements and has been the focus of a growing research effort. Until recently, encryption products were frequently in the form of specialized hardware. For example, encryption devices plugged into the communications line and encrypted all the data going across the line. Although software encryption is becoming more prevalent today, hardware is still the embodiment of choice for many applications.

Speed and security are important issues that play in the favour of the hardware implementation of encryption devices. Encryption algorithms involve many complex operations on the message or plaintext bits. Often these are not the type of operations that are incorporated into a typical desktop computer. The most widely accepted private-key block cipher, the Data Encryption Standard (DES) [1], introduced in 1977, runs inefficiently on general purpose processors. Although some cryptographers have tried to shape their algorithms to suit software implementations, specialized hardware such as an encryption chip will likely emerge as the winner in efficiency. Another key factor that favours the hardware implementation of a block cipher is security. An encryption algorithm being run on a generalized computing machine has no physical protection. On the other hand, hardware encryption devices can be securely encapsulated to prevent this. Other factors that suggest a hardware implementation include cost, lower power consumption, and ease of installation.

The National Institute of Standards and Technology (NIST) has initiated a process to develop a Federal Information Processing Standard (FIPS) for an Advanced Encryption Standard (AES) [2] specifying an encryption algorithm for the twenty-first century as a replacement of DES. In this regard, the agency has announced a request for candidate algorithm nominations of AES. One of the important evaluation criterion concerns the efficiency of the private-key block cipher from the hardware implementation perspective. RC6 [3] and CAST-256 [4] are among the fifteen candidate algorithms that have been presented to the first round of the AES development phase. Both ciphers are modifications of earlier generation ciphers (RC5 [5] and CAST-128 [6]) based on smaller (64-bit) block sizes. Like most proposed private-key block ciphers, RC6 and CAST-256 are clearly designed for efficient implementation in software.

In this paper, we examine the issues associated with the FPGA implementation of RC6 and CAST-256. We chose FPGAs as our target hardware environment because of their usefulness in building *algorithm agile* applications [7], faster turnaround design time, and the ability to easily implement scalable security with variable architectural parameters.

2 Description of the Encryption Algorithms

2.1 The RC6 Cipher

RC6-$w/r/b$, a general version of the RC6 cipher [3], operates on units of four w-bit words, with the encryption consisting of a nonnegative number of rounds r. The user supplies a *primary* key of b bytes, where $0 \leq b \leq 255$ and from this key, the key schedule scheme of the RC6-$w/r/b$ algorithm derives $2r + 4$ subkeys, where each subkey is a w-bit word. These $2r + 4$ subkeys are then stored in the array $S[0,....,2r + 3]$. This array of subkeys is used in both encryption and decryption.

Encryption with the RC6 algorithm is described below. RC6 works with four w-bit registers A, B, C, and D which contain the initial input plaintext as well as the output ciphertext at the end of the encryption. The standard *little endian* convention is used for packing the data bytes into the input/output blocks. The encryption block involves the following basic operations:

$a \oplus b$	bitwise exclusive-or of w-bit words
$a + b$	integer addition modulo 2^w
$a \times b$	integer multiplication modulo 2^w
$a \lll b$	rotate the w-bit word a to the left by the amount given by the least significant $\log_2 w$ bits of b

For the AES implementation of RC6, $w = 32$ and $r = 20$. Each of the four 32-bit registers A, B, C, and D is updated after each round of encryption. At the output of the 20-round encryption the four registers represent the ciphertext. The entire process of encryption in RC6 algorithm is illustrated in Figure 1.

Decryption is similar, but involves reversing the order of the subkeys, replacing left rotations by right ones and replacing addition with subtraction. Since the AES submission requires the cipher to operate on 32-bit words and there should be 20 rounds of encryption/decryption, this paper discusses the issues related to hardware implementation of RC6-32/20/b version. Note that for AES, $b \leq 32$ bytes (i.e. up to 256 bits)

$$
\begin{array}{l}
B = B + S[0] \\
D = D + S[1] \\
\text{for}(i = 1;\ i \leq r\ ;\ i{+}{+}) \\
\quad \{ \\
\qquad t = (B \times (2B + 1)) \lll \log_2 w \\
\qquad u = (D \times (2D + 1)) \lll \log_2 w \\
\qquad A = ((A \oplus t) \lll u) + S[2i] \\
\qquad C = ((C \oplus u) \lll t) + S[2i + 1] \\
\qquad (A, B, C, D) = (B, C, D, A) \\
\quad \} \\
A = A + S[2r + 2] \\
C = C + S[2r + 3]
\end{array}
$$

Figure 1: Encryption with RC6-$w/r/b$

of key are alllowed as *primary* key. For details on the *key scheduling scheme* refer to [3].

2.2 The CAST-256 Cipher

CAST-256 [4] is a private-key block cipher that is a generalization of the basic "Feistel" network [8]. CAST-256 algorithm uses a 128-bit block size and a 256-bit primary key that is used in the algorithm's key schedule scheme to generate two sets of subkeys, each of which is used per round: a 5-bit subkey K_{r_i} is used as a "rotation" key for round i and a 32-bit subkey K_{m_i} is used as a "masking key" for round i. There are a total of 48 rounds of encryption.

Three different 32-bit round functions are used in CAST-256. These round functions are defined as follows:

- Round Function (f_1)
  $I = ((K_{m_i} + D) \lll K_{r_i})$
  $O = ((S_1[I_a] \oplus S_2[I_b]) - S_3[I_c]) + S_4[I_d]$
- Round Function (f_2)
  $I = ((K_{m_i} \oplus D) \lll K_{r_i})$
  $O = ((S_1[I_a] - S_2[I_b]) + S_3[I_c]) \oplus S_4[I_d]$
- Round Function (f_3)
  $I = ((K_{m_i} - D) \lll K_{r_i})$
  $O = ((S_1[I_a] + S_2[I_b]) \oplus S_3[I_c]) - S_4[I_d]$

Here D is the 32-bit data input to the round function, $I_a to I_d$ are the most significant byte through the least significant byte of I, respectively, S_i is the i^{th} substitution box or S-box, and O is the 32-bit output of the round function. Each S-box is a nonlinear mapping of an 8-bit input to a 32-bit output [9]. Moreover, "+" and "-" are addition and subtraction modulo 2^{32} operations, "$\oplus$ " is bitwise exclusive-OR operation, and, finally, "$u \lll v$" is the rotation of u to left by the value indicated by v.

$$
\begin{array}{l}
\text{for}(i = 0;\ i < 6;\ i{+}{+}) \\
\quad \{ \\
\qquad C = C \oplus f_1(D, K_{r_{4i+1}}, K_{m_{4i+1}}) \\
\qquad B = B \oplus f_2(C, K_{r_{4i+2}}, K_{m_{4i+2}}) \\
\qquad A = A \oplus f_3(B, K_{r_{4i+3}}, K_{m_{4i+3}}) \\
\qquad D = D \oplus f_1(A, K_{r_{4i+4}}, K_{m_{4i+4}}) \\
\quad \} \\
\text{for}(i = 6;\ i < 12;\ i{+}{+}) \\
\quad \{ \\
\qquad D = D \oplus f_1(A, K_{r_{4i+1}}, K_{m_{4i+1}}) \\
\qquad A = A \oplus f_3(B, K_{r_{4i+2}}, K_{m_{4i+2}}) \\
\qquad B = B \oplus f_2(C, K_{r_{4i+3}}, K_{m_{4i+3}}) \\
\qquad C = C \oplus f_1(D, K_{r_{4i+4}}, K_{m_{4i+4}}) \\
\quad \}
\end{array}
$$

Figure 2: Encryption with CAST-256

The CAST-256 encryption algorithm is illustrated in Figure 2. The plaintext is stored in four 32-bit input registers A, B, C, and D. There are 48 rounds of encryption. In each round of encryption, a 32-bit *masking* key and a 5-bit *rotation* key is used. The output of the 48-round encryption is contained in the four 32-bit registers A, B, C, and D as the ciphertext.

Decryption is identical to encryption except that the masking and round keys derived from the primary key K are used in the reverse order. Note that the 256-bit primary key can be generated from smaller user keys as outlined in the CAST-256 algorithm specifications [4]. Details of the key scheduling scheme for CAST-256 are also outlined in [4].

3 Hardware Implementation Environment

3.1 FPGAs vs. ASICs

A new development in integrated circuits offers a hardware implementation choice that is much more flexible than Application Specific Integrated Circuits (ASICs): large, fast, reconfigurable gate arrays, popularly, known as Field Programmable Gate Arrays (FPGAs). These devices consist of arrays of configurable logic blocks that implement logical functions of gates and are easily reconfigurable. In contrast, ASICs provide only the functionality needed for a specific task. A well-designed ASIC chip will support a particular application for which it is designed, but not a slightly modified version of the same application introduced after the ASIC design is completed. Furthermore, even if a modified ASIC can be developed, the original hardware is too highly customized to be reused in successive generations. In contrast, the configuration of an FPGA can be easily reprogrammed to accommodate a design modification.

We have chosen FPGAs as the target technology for realizing the RC6 and CAST-256 cryptographic algorithms in hardware because of a number of reasons. Replacing one cryptographic algorithm with another is a trivial matter in software, but it is not in hardware. But at the same time, hardware solutions can offer improved performance in terms of speed, security, and cost. As such the solution to this problem is reconfigurable hardware and FPGAs are the answer. In fact, FPGAs can be used to build algorithm agile applications [7] . The term algorithm agility refers to the fact that the same FPGA can be reprogrammed at run time to support different algorithms. Other key factors that favour the use of FPGAs for hardware implementation of ciphers include faster turnaround design time, scalable security, and variable architectural parameters.

3.2 Development Environment

The design cycle and CAD tools used for the hardware implementation of RC6 and CAST-256 algorithms have been provided by Canadian Microelectronics Corporation (CMC) [10]. The entire design process can be divided into the following three stages:

- Generating the VHDL (IEEE 1076) descriptions of the cipher design, employing different architecture options. The functional VHDL simulation of the design is carried out using the Synopsys VSS simulator version 1998.02 to verify the correct operation of the cryptographic algorithm.
- Gate-level synthesis and logic optimization of the design utilizing Synopsys Design Compiler version 1998.02 to produce a functionally equivalent schematic in hardware.
- Place and route for a specific FPGA device followed by a final verification of the design. The timing simulation data is generated during this design stage which is then used to carry out timing simulation for the final verification of the design.

We chose Xilinx as the FPGA vendor and a XC4000 device family. In particular, we used XC40200XV-9-BG560 as our target device. This particular FPGA has a total of 7056 configurable logic blocks (CLBs), which gives us a baseline with which we can measure FPGA resource consumption. Xilinx Alliance Series version 1.5 is used for place and route.

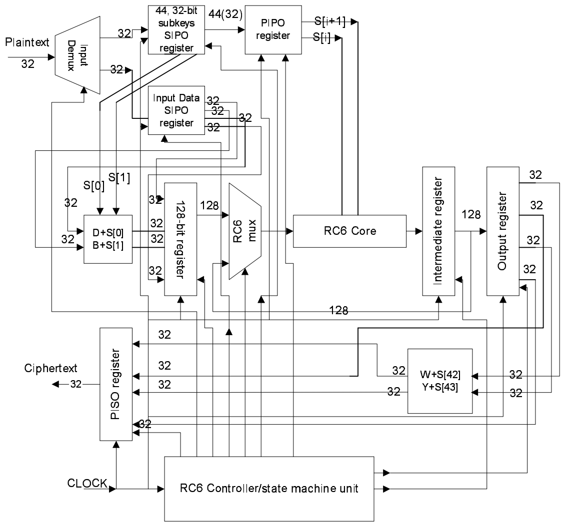

Figure 3: RC6 Encryption in Hardware

4 Cipher Design Issues

4.1 Design of RC6

The block diagram representation of the RC6 encryption realized in hardware is shown in Figure 3. The RC6 core basically consists of four components, namely, a 32-bit adder, a 32×32 "partial" integer multiplier (i.e. the product is modulo 2^{32}), a 32-bit XOR and a 32-bit barrel shifter. There is also an RC6 controller/state machine unit that controls the various modes of operation of the cipher i.e. it determines when the cipher is in the *key-download* mode, the *data-download* mode, the *data-encrypt* mode, the *idle* mode or the *reset* mode. The block diagram for the RC6 encryption also shows the logic (shift registers, multiplexers/demultiplexers, serial-in parallel-out (SIPO), parallal-in serial-out (PISO) and parallel-in parallel-out (PIPO) registers) associated with the datapath of the cipher.

We will now consider the components associated with RC6 core.

32-bit XOR

The timing and FPGA resource reports for the synthesized 32-bit XOR reveal that the maximum delay for data to be available at the output of the 32-bit XOR is equal to 4.88 ns and total number of CLBs used is equal to 16 (0.2 % of FPGA CLB resources).

32-bit Barrel Shifter

One of the major concerns in the design of the RC6 core has to do with the data-dependent rotations. We had to look for an implementation that would take constant time for these rotations, irrespective of the size of the rotation, for otherwise, the RC6 algorithm is vulnerable to the *timing attack* [11] that may lead to breaking the cipher. The solution to this problem is a barrel shifter that can shift any number of bits in one clock cycle. Synthesis results reveal that our implementation of the 32-bit barrel shifter gives a maximum delay of 4.88 ns for the data to be shifted out. However, it uses 369 CLBs (5.2 % of the total FPGA CLBs), which is significantly more than the much slower serial shifter.

32-bit Adder

The implementation of a fast, low complexity 32-bit adder involved the consideration of a number of design choices. We first explored a 32-bit *carry ripple adder* (CRA) implementation which gave a maximum delay equal to 173.21 ns and used 32 CLBs (0.45 % of the total CLBs). Hence, the CRA has a minimal complexity but a large delay.

A *carry lookahead adder* (CLA) is the fastest of all adders, but is not practically feasible for addition of numbers greater than 8 bits, because of the very high complexity and the limitations that arise from potentially high fan-in and fan-out requirements. It is impractical for the 32-bit addition required.

The design of a 32-bit adder using *hierarchical carry lookahead*, also known as *block carry lookahead adder* (BCLA) [12] was also investigated. This design uses eight 4-bit CLAs as the building blocks to take care of high fan-in and fan-out requirements for a bigger pure CLA. Although, the delay was reduced down to 70 ns, the design used a total of 60 CLBs, 0.85 % of the total FPGA resources (an increase in the hardware complexity by a factor of two).

We also considered a pure 32-bit *carry select adder* (CSA) as well as a 32-bit carry select adder using a 4-bit CRA as the basic unit. However, synthesis studies using FPGAs revealed that a *hybrid* design is preferred on the basis of its speed. This *hybrid* design uses a 4-bit CLA as the basic unit in a 32-bit CSA. Our final results show that this implementation choice for the 32-bit adder yields a maximum delay of 39.32 ns (an improvement by a factor of 4.4 over the CRA implementation). The 32-bit hybrid adder, however, takes up 197 CLBs (2.79 % of the FPGA resources).

32×32 "Partial" Integer Multiplier

This component is the most critical of all components in the RC6 encryption core. We need a 32×32 "partial" integer multiplier to compute integer multiplication modulo 2^{32}. Our initial implementation of the multiplier used a behavioural representation. The synthesis results for this implentation of the partial multiplier in FPGAs were not encouraging as it used 551 CLBs (7.8 % of the available CLBs) for the target device. The synthesized multiplier had a maximum delay of 294 ns, which was considered to be unacceptably high. This led to the investigation of structural design options for the multiplier. We have found the *Wallace tree* [13] architecture to be more suitable for the implementation of the partial multiplier in FPGAs than other design options. The Wallace tree implementation reduced the maximum delay for the multiplier to 79 ns (an improvement by a factor of 3.8). However, this implementation increases the hardware complexity of the multiplier to 930 CLBs (13 % of the available CLBs).

4.2 Design of CAST-256

The block diagram of CAST-256 encryption realized in hardware is shown in Figure 4. The CAST-256 core basically consists of a *generic round function* that realizes any of the three round functions f_1, f_2, or f_3 depending on the particular round in progress. The rest of the block diagram, as in the case of RC6 cipher, illustrates the logic associated with the datapath and the control path of the CAST-256 cipher.

Generic Round Function

The generic round function consists of four 32-bit adder/subtractor/excusive-or units, a separate 32-bit XOR, a 32-bit barrel shifter and the four 8×32 S-boxes, namely, S_1, S_2, S_3, and S_4. The CAST-256 controller is used to control the flow of data through each sub-component of the generic round function. The design of the generic round function is modular and purely structural. A decoding logic unit inside the cipher's main controller generates the necessary control signals that decide the flow of data through the generic round function for each round of encryption.

The hardware complexity of the cipher is primarily due to the complex structures of the S-boxes. We have implemented these S-boxes as *table lookups*. Each S-box takes around 411 CLBs and has a maximum delay of 62 ns. The generic round function requires a total of 3037 CLBs (43 % of the available FPGA resources) and has a maximum delay of 202 ns.

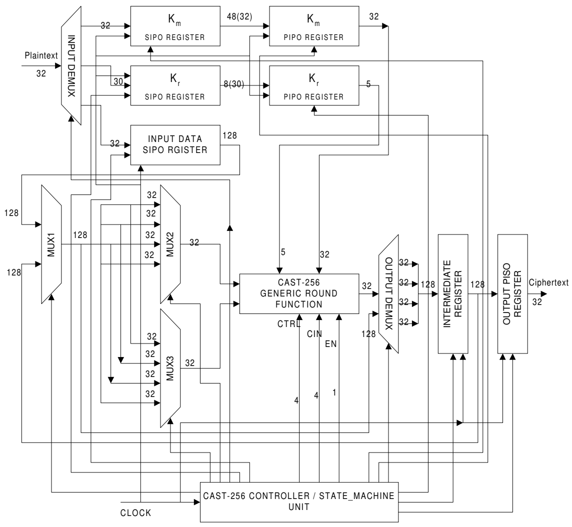

Figure 4: CAST-256 Encryption in Hardware

5 Comparison of RC6 and CAST-256 Ciphers

Oue synthesis studies for implementation of RC6 and CAST-256 encryption in FPGAs yield the following results. (It should be noted that we are assuming that the subkeys for encryption are being generated outside the FPGA using the key schedule scheme for each cipher and that they are being downloaded into the key storage unit designed for each cipher.)

- Neglecting the key setup and algorithm setup times, our design of RC6 encryption in the XC40200 FPGA device yields a data encryption rate of about 37 Mbits/s. The hardware needed for the RC6 encryption is 4944 CLBs for the RC6 core plus 704 CLBs for key storage purposes and another 64 CLBs for storing the 128-bit input data. Besides this, there is also some data flow and control logic overhead (on the order of 750 CLBs). Thus the total FPGA resources required is about 6450 CLBs or 91% of the available CLBs in the target device.
- The hardware implementation of CAST-256 in the XC40200 FPGA device yields a data encryption rate of about 13 Mbits/s for a 48-round implementation, and about 26 Mbits/s for a 24-round implementation. The hardware needed for the CAST-256 en-

cryption is 3037 CLBs for the generic round function plus 768 CLBs for "masking" keys storage and another 120 CLBs for "rotation" keys storage purposes. Besides this, there are also 64 CLBs for storing the 128-bit input data. The data flow and control logic overhead is around 1000 CLBs. Thus a total of about 5052 CLBs or 72% of the available CLBs are required to implement CAST-256.

6 Conclusions

In conclusion, it appears neither RC6 nor CAST-256 are well suited for implementation in the targetted Xilinx FPGA. The hardware complexity is high and the speed low, particularly compared to similar implementations of DES [7].

Our simulation and synthesis studies reveal that multiplication and addition are major bottlenecks as far as speed of encryption in the RC6 cipher is concerned. However, a faster implementation of RC6 can only be achieved at the expense of increasingly large hardware complexity, which implies the use of a high end FPGA device. Moreover, it appears that implementation of RC6 in the targetted FPGA using *pipelining* is found to be impractical from a hardware complexity viewpoint.

CAST-256 encryption in FPGAs is found to be slower than what we can achieve with the RC6 cipher. At the same time, the hardware complexity of CAST-256 cipher is roughly of the same order as RC6. This is because the advantage of not having a multiplication operation is being offset by the use of four S-boxes.

As a consequence of investigating the FPGA implementations of these two private-key block ciphers, we suggest a much simpler cipher design that makes use of simpler operations that not only possess good cryptographic properties, but also make the overall design efficient from the hardware implementation perspective. For example, by finding a suitable design that reduces the hardware complexity, it will be possible to effectively pipeline multiple rounds and thereby increase the cipher speed for FPGA implementations.

References

[1] "National Bureau of Standards - Data Encryption Standard,"FIPS Publication 46, 1977.

[2] NIST Advanced Encryption Standard (AES) Development Effort web site "http://csrs.nist.gov-/encryption/aes/aes-home.htm".

[3] R.L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin, "The RC6 Block Cipher," available at web site "http://theory.lcs.mit.edu/ rivest/rc6.pdf".

[4] C. Adams, "The CAST-256 Encryption Algorithm," available at web site "http://www.entrust.-com/resources/pdf/cast256.pdf".

[5] R.L. Rivest, "The RC5 Encryption Algorithm," *Proceedings of Fast Software Encryption - 2nd International Workshop*, Leuven, Belgium, Springer-Verlag LNCS 1008, pp. 86-96, 1995.

[6] C. Adams, "Constructing Symmetric Ciphers Using the CAST Design Procedure," *Designs, Codes, and Cryptography*, vol. 12, no. 3,pp. 283-316, 1997.

[7] J.-P. Kaps and C. Paar, "Fast DES Implementation for FPGAs and its Application to a Universal Key-search Machine," presented at Workshop in Selected Areas of Cryptography (SAC'98), Kingston, Ont., Aug. 1998.

[8] H. Feistel, W.A. Notz, and J.L. Smith, "Some Cryptographic Techniques for Machine-to-Machine Communications," *Proceedings of IEEE*, Vol. 63, No. 11, pp. 1545-1554, 1975.

[9] C.M. Adams and S.E. Tavares, "Designing S-boxes for Ciphers Resistant to Differential Cryptanalysis," *Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography*, Rome, Italy, pp. 181-190, 15-16 Feb. 1993.

[10] The Canadian Microelectronics Corporation Website: "http:// www.cmc.ca"

[11] H. M. Heys, "A Timing Attack on RC5," presented at SAC'98, Kingston, Ont., Aug. 1998.

[12] Doran, R. W., "Variants on an Improved Carry Lookahead-Adder," *IEEE Trans. on Computers*, Vol. 37, No. 9, pp. 1110-1113, 1988.

[13] Wallace, C. S., "A Suggestion for a Fast Multiplier," *IEEE Trans. on Computer*, Vol EC-13, pp. 14-17, 1964.