Hardware performance analysis of the SHACAL-2 encryption algorithm

M. McLoone

Abstract: A hardware performance analysis of the SHACAL-2 encryption algorithm is presented. SHACAL-2 was one of four private-key algorithms chosen in the New European Schemes for Signatures, Integrity and Encryption (NESSIE) initiative. To the author's knowledge, there has been no previous published research work conducted on hardware SHACAL-2 architectures. Consequently, in this paper, both iterative and pipelined designs are developed and implemented. A fully pipelined encryption SHACAL-2 architecture implemented on a Virtex-II XC2V4000 device achieves a throughput of over 25 Gbit/s. This is one of the fastest encryption algorithm implementations currently available. The iterative encryption architecture operates at 432 Mbit/s on the XC2V500 device. A comparison is provided between SHACAL-2 hardware designs that incorporate carry save adders and designs that include typical full adders. The SHACAL-2 decryption algorithm is also clearly defined in the paper as it was not provided in the NESSIE submission.

1 Introduction

In recent times, data security has become an essential aspect in information technology and communication systems. Government bodies, banks and commercial companies are now realising the need for secure, reliable and high performing encryption algorithms. In 1997, the US National Institute of Standards and Technology (NIST) requested candidates for a new Advanced Encryption Standard (AES) algorithm to replace the private-key Data Encryption Standard (DES) algorithm, which was broken by a brute-force attack [1]. Following a three year rigorous selection process, the Rijndael algorithm was selected to replace DES as the Federal Information Processing Encryption Standard (FIPS). In Europe, a similar initiative, known as New European Schemes for Signatures, Integrity and Encryption (NESSIE), began in January 2000 [2]. NESSIE was a three year project that formed part of the Information Societies Technology (IST) programme of the European Commission. While the AES development effort concentrated solely on private-key algorithms, the aim of the NESSIE project was to present a portfolio of strong cryptographic primitives evaluated in terms of both security and performance. There were 42 original submissions, which included private and public key algorithms, MAC algorithms, hash functions and digital signature algorithms. In the evaluation process researchers attacked the submitted algorithms to find weaknesses and compromise their security. In February 2003, NESSIE recommended 12 of the original submissions and 5 existing standard algorithms to form the portfolio. The following private key block ciphers were selected:

- MISTY1: developed by Mitsubishi Electric Corporation, Japan [3]
- SHACAL-2: developed by Gemplus, France [4]
- Camellia: developed by Nippon Telegraph and Telephone Corporation & Mitsubishi Electric Corporation, Japan [5]
- AES: the US NIST FIPS [6], developed by Vincent Rijmen and Joan Daemen

The SHACAL-2 algorithm is one of two versions of the SHACAL submission to NESSIE. The basic version of the SHACAL algorithm is known as SHACAL-1. It operates on a 160-bit block of data and is based on the secure hash algorithm, SHA-1. The extended version, SHACAL-2 operates on a 256-bit data block and is based on the SHA-256 hash algorithm. In anticipation of the expected increase in use of the AES standard the NIST proposed an expansion of their hash standard, SHA-1 to include three new hash algorithms, SHA-256, SHA-384 and SHA-512. This new FIPS 180-2 standard became effective in February 2003 [7].

Until now there has been no published work on hardware implementations of the SHACAL-2 algorithm. The only research carried out on hardware architectures of the SHACAL-1 algorithm was the work of the author [8]. A fully pipelined SHACAL-1 encryption architecture is described, which is capable of achieving data-rates of up to 17 Gbits/s, while a similar decryption architecture achieves a speed of 13 Gbit/s. Performance metrics for software implementations of the SHACAL-2 algorithm are available. The SHACAL specification provided figures for an implementation on an 800 MHz Pentium III processor [4]. Encryption and decryption can both be performed at a speed of 56 Mbit/s. In this paper, iterative and fully pipelined SHACAL-2 hardware architectures are presented. These architectures are

IEE Proceedings online no. 20059027

doi:10.1049/ip-cds:20059027

Paper first received 1st August 2003 and in revised form 29th September 2004

The author is with the Institute of Electronics, Communications and Information Technology (ECIT), Queen's University Belfast, Queens Road, Queens Island, Belfast BT3 9DT, UK

E-mail: maire.mcloone@ee.qub.ac.uk

implemented on Xilinx Virtex II FPGA devices [9] for demonstration purposes.

2 The SHACAL-2 algorithm

The SHACAL-2 algorithm operates on a 256-bit block of data utilising a key, k, where, $128 \leq k \leq 512$. If $k < 512$, it is appended with zeros to 512-bits in length. The SHACAL-2 key schedule expands the 512-bit key into sixty-four 32-bit subkeys. The initial sixteen subkeys, W_0 to W_{15}, are formed by splitting the input key into sixteen 32-bit data blocks. The plaintext is split into eight 32-bit blocks of data and these are operated on by a compression function, which cycles through 64 iterations. After the final iteration, the eight 32-bit compression function outputs are concatenated to form the ciphertext. The overall algorithm is described below and represented pictorially in Fig. 1.

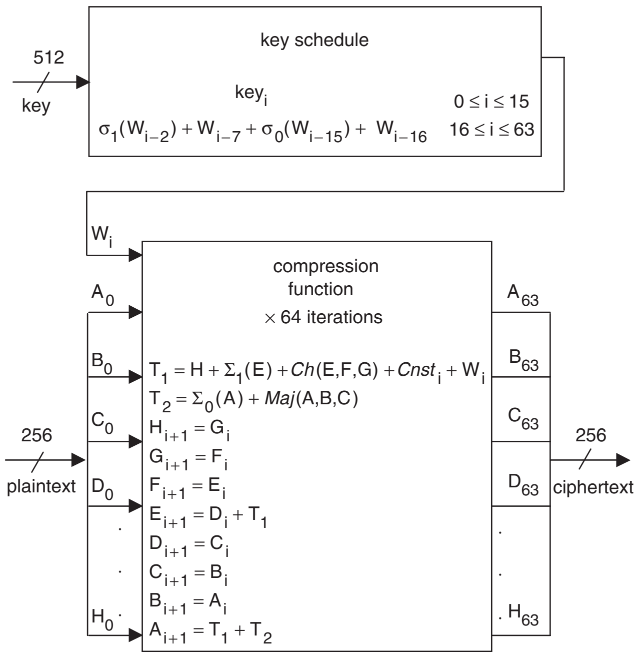

Fig. 1 *SHACAL-2 encryption algorithm*

SHACAL-2 encryption algorithm:
Plaintext: *256-bit data block*
Key: *$128 \leq k \leq 512$*
Ciphertext: *256-bit data block*

- Split 256-bit plaintext into 8×32-bit blocks, A_0, B_0, C_0, D_0, E_0, F_0, G_0 and H_0
- Perform 64 iterations of the following compression function:

$$\begin{aligned}
T_1 &= H + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + Cnst_i + W_i \\
T_2 &= \Sigma_0(A)_i + Maj(A_i, B_i, C_i) \\
H_{i+1} &= G_i \\
G_{i+1} &= F_i \\
F_{i+1} &= E_i \\
E_{i+1} &= D_i + T_1 \\
D_{i+1} &= C_i \\
C_{i+1} &= B_i \\
B_{i+1} &= A_i \\
A_{i+1} &= T_1 + T_2
\end{aligned} \tag{1}$$

- Concatenate A_{63}, B_{63}, C_{63}, D_{63}, E_{63}, F_{63}, G_{63} and H_{63} to form ciphertext
- The 64 constants, *Cnst*$_i$ are as outlined in the SHA-256 specification [7]
- The functions are defined as:

$$\begin{aligned}
\Sigma_1(x) &= ROT_{R-6}(x) \oplus ROT_{R-11}(x) \oplus ROT_{R-25}(x) \\
Ch(x,y,z) &= (x \text{ AND } y) \oplus (x \text{ AND } z) \\
\Sigma_0(x) &= ROT_{R-2}(x) \oplus ROT_{R-13}(x) \oplus ROT_{R-22}(x) \\
Maj(x,y,z) &= (x \text{ AND } y) \oplus (x \text{ AND } z) \oplus (y \text{ AND } z)
\end{aligned} \tag{2}$$

where,
ROT_{R-n} is defined as a 32-bit word rotated to the right by n positions

- The input 512-bit key is expanded to sixty-four 32-bit subkeys, W_i, where

$$W_i = \begin{cases} Key_i \\ \quad 0 \leq i \leq 15 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} \\ \quad 16 \leq i \leq 63 \end{cases} \tag{3}$$

where,

$$\begin{aligned}
\sigma_0(x) &= ROT_{R-7}(x) \oplus ROT_{R-18}(x) \oplus SHF_{R-3}(x) \\
\sigma_1(x) &= ROT_{R-17}(x) \oplus ROT_{R-19}(x) \oplus SHF_{R-10}(x)
\end{aligned}$$

and,
SHF_{R-n} is defined as a 32-bit word shifted to the right by n positions.

2.1 SHACAL-2 decryption

A description of the SHACAL-2 decryption algorithm was not included in the SHACAL algorithm submission to NESSIE. Therefore, it is derived and outlined below. It requires and inverse compression function and an inverse key schedule.

SHACAL-2 decryption algorithm:
Ciphertext: 256-bit data block
Key: $128 \leq k \leq 512$
Plaintext: 256-bit data block

- Split 256-bit ciphertext into 8×32-bit blocks, A_0, B_0, C_0, D_0, E_0, F_0, G_0 and H_0
- Perform 64 iterations of the following inverse compression function:

$$\begin{aligned}
T_1 &= \Sigma_1(F_i) + Ch(F_i, G_i, H_i) + InvCnst_i + InvW_i \\
T_2 &= \Sigma_0(B_i) + Maj(B_i, C_i, D_i) \\
A_{i+1} &= B_i \\
B_{i+1} &= C_i \\
C_{i+1} &= D_i \\
D_{i+1} &= (E_i + T_2) - A_i \\
E_{i+1} &= F_i \\
F_{i+1} &= G_i \\
G_{i+1} &= H_i \\
H_{i+1} &= A_i - (T_1 + T_2)
\end{aligned} \tag{4}$$

- Concatenate A_{63}, B_{63}, C_{63}, D_{63}, E_{63}, F_{63}, G_{63} and H_{63} to form original plaintext
- The constants, *InvCnst*$_i$ are those utilised in encryption, in reverse order

- The functions $\Sigma_1(x)$, $Ch(x,y,z)$, $\Sigma_0(x)$ and $Maj(x,y,z)$ are as defined for the encryption algorithm
- In the inverse key schedule the 512-bit inverse key is expanded to sixty-four 32-bit subkeys, $\mathrm{Inv}W_i$, where,

$$\mathrm{InvW_i} = \begin{cases} \mathrm{InvKey_i} \\ \quad 0 \le i \le 15 \\ \mathrm{InvW_{i-16}} - [\sigma_1(\mathrm{InvW_{i-14}}) + \mathrm{InvW_{i-9}} \\ \quad +\sigma_0(\mathrm{InvW_{i-1}})] \quad 16 \le i \le 63 \end{cases} \tag{5}$$

and $\sigma_0(x)$ and $\sigma_1(x)$ are as defined for the encryption algorithm.

In the decryption process the subkeys created during encryption are used in reverse order. Thus, if the original key is sent to the receiver, it is necessary to wait until all subkeys are created before commencing decryption. However, it is possible to generate the subkeys as they are required during decryption by using an inverse key, *InvKey* [8]. The *InvKey* is created by concatenating the final 16 subkeys generated during encryption, where,

$$InvKey = \{W_{48}, W_{49}, W_{50}, W_{51}, W_{52}, W_{53}, W_{54}, W_{55}, W_{56}, W_{57}, W_{58}, W_{59}, W_{60}, W_{61}, W_{62}, W_{63}\} \tag{6}$$

3 SHACAL-2 hardware architectures

An outline of the SHACAL-2 iterative architecture is shown in Fig. 2. The main components in the design are the compression function and key scheduling blocks. It also comprises a counter and a memory block. Initially, the 256-bit plaintext is split into eight 32-bit data blocks, A to H, and these are input into the compression function. The output A to H values are looped back to form the inputs on subsequent iterations. After 64 iterations, the output values are concatenated to form the ciphertext.

An outline of the SHACAL-2 pipelined architecture is provided in Fig. 3. Similar to the iterative design, the main components in the pipelined architecture are the key schedule and compression function. However, in this architecture the 64 compression functions are unrolled

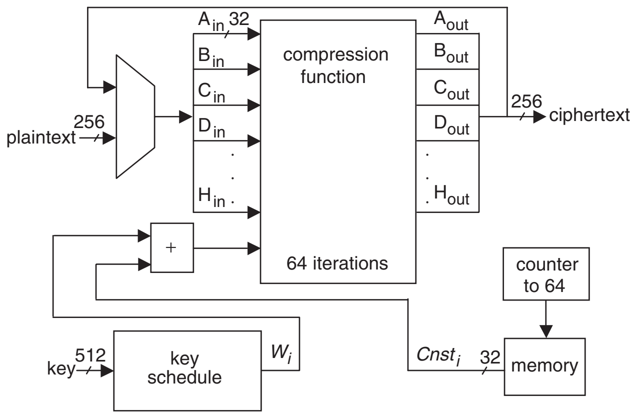

Fig. 2 *Outline of SHACAL-2 iterative encryption architecture*

and pipeline stages placed between each function. Plaintext blocks can be accepted on every clock cycle and after an initial latency, the corresponding ciphertext blocks will appear on consecutive clock cycles. Therefore, very high data throughputs can be achieved utilising this design.

The SHACAL-2 algorithm is derived from the SHA-256 hash algorithm. This algorithm is very similar to the SHA-384 and SHA-512 hash algorithms, but they produce different hash lengths. As described in Section 2, SHACAL-2 encryption consists of a compression function and key schedule. These components also feature in the SHA-256, SHA-384 and SHA-512 algorithms. The author has developed efficient hardware implementations of these components in the context of the SHA-384 and SHA-512 algorithms [10] and therefore, these components are used in the SHACAL-2 algorithm architectures described in this paper.

The key schedule is designed using a 16-stage shift register as illustrated in Fig. 4 [10]. The 512-bit key is loaded into the 32-bit registers over 16 clock cycles. On the next clock cycle, the value of register 15 is replaced with the result of (3). The output, W_i is taken from the output of register 15 and therefore, the subkeys are created as they are required by the compression function. The SHACAL-2 inverse key schedule architecture, depicted in Fig. 5, is

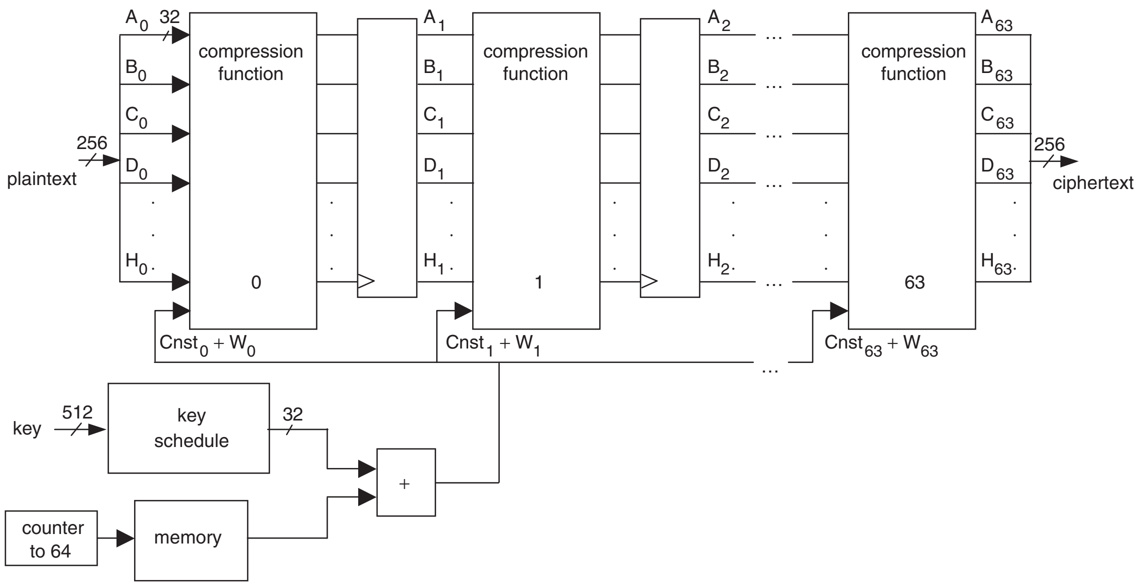

Fig. 3 *Outline of SHACAL-2 fully pipelined encryption architecture*

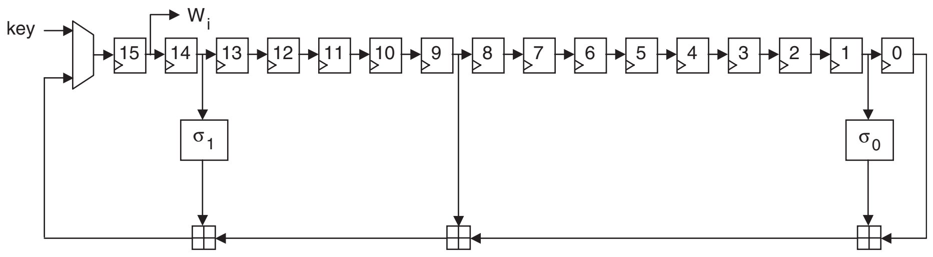

Fig. 4 *SHACAL-2 key schedule architecture*

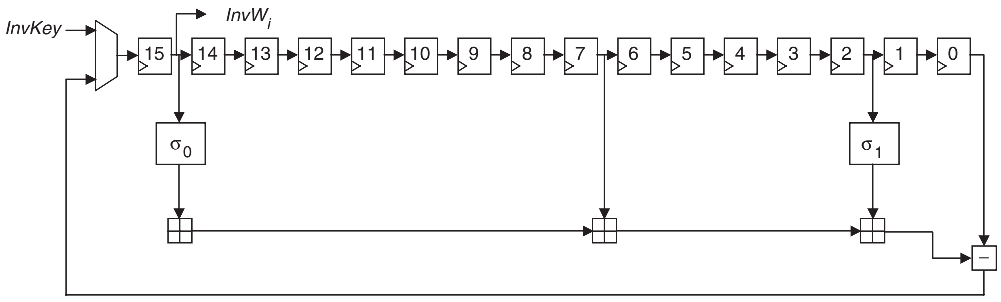

Fig. 5 *SHACAL-2 inverse key schedule architecture*

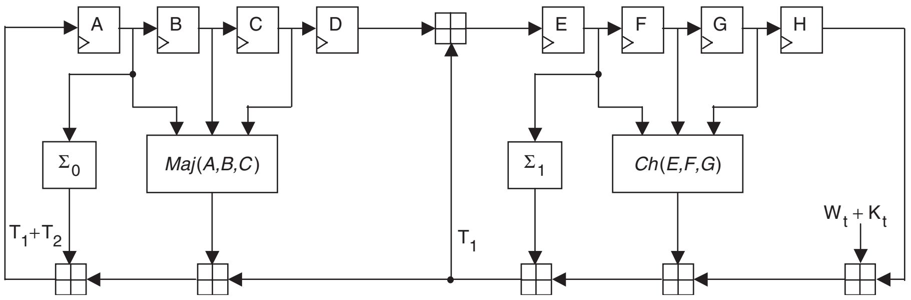

Fig. 6 *SHACAL-2 compression function architecture*

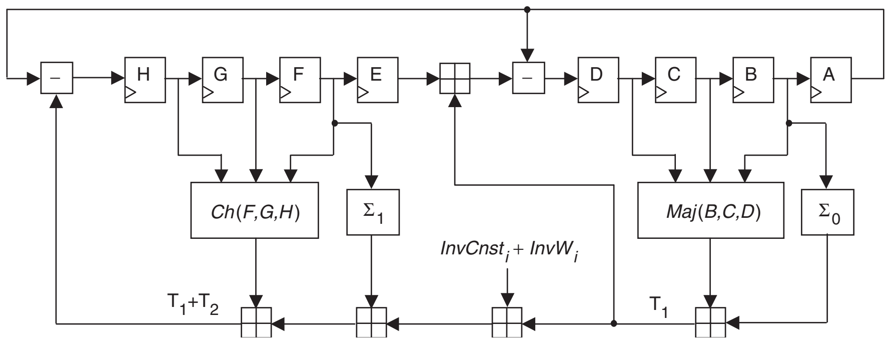

Fig. 7 *SHACAL-2 inverse compression function architecture*

similar to that used for encryption. If the inverse key, *InvKey*, described in Section 2, is utilised instead of the original key, the inverse subkeys, $InvW_i$, can be generated as they are required by the inverse compression function, avoiding a delay of 64 clock cycles.

The compression function architecture, shown in Fig. 6 [10], is designed using eight 32-bit registers that store the updating values of A to H. The SHACAL-2 inverse compression function, illustrated in Fig. 7, is more complicated in that it contains two subtractions and therefore, the SHACAL-2 decryption design is slower than the encryption design.

A memory component is used to store the 64 32-bit constants. On the Virtex II FPGA device, the memory is implemented as a dual-port BlockRAM, as shown in Fig. 8.

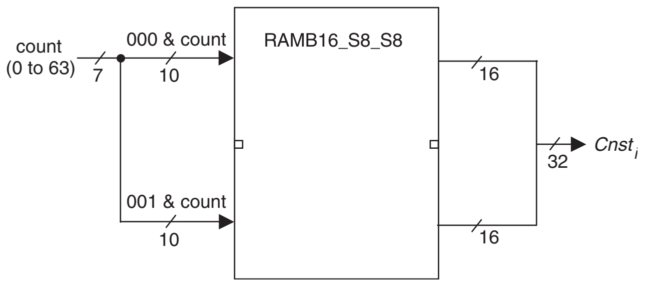

Fig. 8 *SHACAL-2 memory block containing constants*

The counter is used to address the BlockRAM and thus, the constants are output in sequence as required by the compression function.

The critical path of the iterative and pipelined encryption designs occurs in the compression function component in the calculation of

$$A_{i+1} = T_1 + T_2 = H + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + Cnst_i + W_i + \Sigma_0(A_i) + Maj(A_i, B_i, C_i) \quad (7)$$

This path can be reduced in two ways [8]. Firstly, the addition, $Cnst_i$ + W_i, can be performed on the previous clock cycle and thus, is removed from the critical path. The second method involves the utilisation of carry-save adders (CSAs) in preference to the typical full adders (FAs). When using CSAs, carry propagation is avoided until the final addition and therefore, CSAs are beneficial in computations comprising multiple additions. A CSA is implemented using the following equations:

$$S = A \text{ xor } B \text{ xor } C \quad (8)$$

$$C = (A \text{ and } B) \text{ or } (A \text{ and } C) \text{ or } (B \text{ and } C) \quad (9)$$

Figure 9 depicts (7) implemented using CSAs.

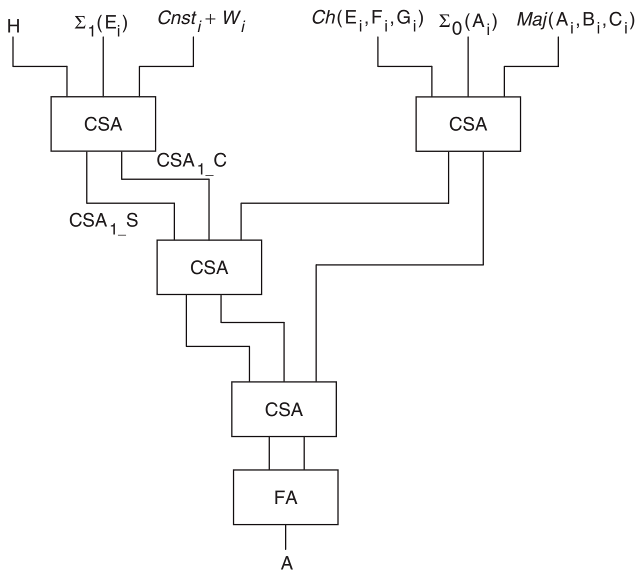

Fig. 9 *CSA implementation of critical path, $A = T_1 + T_2$, for encryption*

The following equation,

$$E_{i+1} = D_i + T_1 = D_i + H + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + Cnst_i + W_i \quad (10)$$

can also be implemented using CSAs. Since there are four additions, 3 CSAs and 1 full adder are required in an implementation. However, since the addition of $H + \Sigma_1(E_i) + Cnst_i + W_i$ is shared with (7), the implementation can be reduced to 2 CSAs and 1 FA as shown in Fig. 10.

When the designs are implemented utilising CSAs, the critical path no longer occurs in the compression function. It now falls in the key schedule component in the calculation of the subkeys, as described in (3).

In the iterative and pipelined decryption architectures, the critical path appears in the inverse compression function in the calculation,

$$H_{i+1} = A_i - [\Sigma_0(B_i) + Maj(B_i, C_i, D_i) + \Sigma_1(F_i) + Ch(F_i, G_i, H_i) + InvCnst_i + InvW_i] \quad (11)$$

This path can be reduced using the two approaches described for the encryption architectures. The addition, $InvCnst_i$ + $InvW_i$, can be performed on the previous clock

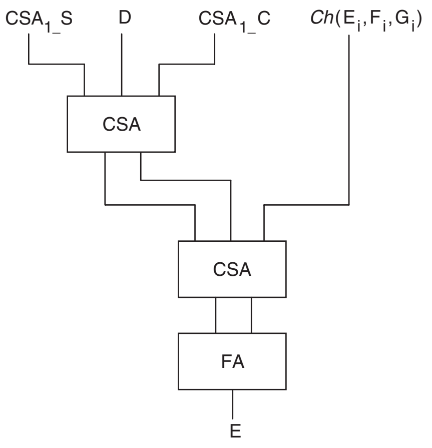

Fig. 10 *CSA implementation of equation, $E_{i+1} = D_i + T_1$, for encryption*

cycle and the remaining equation can be implemented using CSAs and two's complement logic, such that,

$$H_{i+1} = A_i + \overline{\Sigma_0(B_i)} + \overline{Maj(B_i, C_i, D_i)} + \overline{\Sigma_1(F_i)} + \overline{Ch(F_i, G_i, H_i)} + \overline{InvCnst_i + InvW_i} + 5 \quad (12)$$

The implementation of (12) is shown in Fig. 11. The equation

$$D_{i+1} = (E_i + T_2) - A_i = E_i + \Sigma_0(B_i) + Maj(B_i, C_i, D_i) + \overline{A_i} + 1 \quad (13)$$

can also be implemented in a similar manner utilising CSAs and two's complement logic.

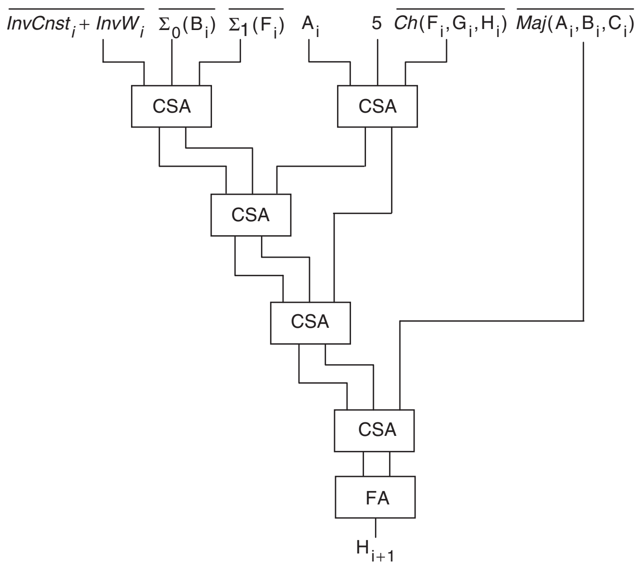

Fig. 11 *CSA implementation of critical path, $H_{i+1} = A_i - (T_1 + T_2)$, for decryption*

4 Performance analysis

To perform a hardware performance analysis of the SHACAL-2 encryption algorithm, the iterative and pipelined designs described in Section 3 were implemented on Xilinx Virtex II FPGAs. They were simulated using Modelsim 5.7b and synthesised using Synplify Pro v7.2 and Xilinx Foundation Series 5.1i. The designs were verified using the test vectors provided with the SHACAL NESSIE submission.

The iterative encryption and decryption architectures were implemented on XC2V500 Virtex-II devices. The

Table 1: Performance of iterative architectures implemented with and without CSAs

Type of design	Device	Data-rate (Mbit/s)	Area	Efficiency (Mbit/s/slices)
Encryption: No CSAs	XC2V500	326	1413 slices, 1 BRAM	0.23
Encryption: CSAs	XC2V500	432	1709 slices, 1 BRAM	0.25
Decryption: No CSAs	XC2V500	287	1460 slices, 1 BRAM	0.20
Decryption: CSAs	XC2V500	299	2029 slices, 1 BRAM	0.15

performance results obtained are described in Table 1. The use of CSAs in the encryption architecture improves the data-rate and although the area is increased, the overall efficiency of the design is also improved. However, this is not true for the decryption architecture. Since its critical path contains a subtraction, there is no significant advantage to utilising CSAs.

The pipelined architectures were implemented on XC2V4000 devices. The encryption architecture implemented using CSAs achieves a throughput of 25.88 Gbit/s with a clock speed of 101 MHz. It requires 23,038 CLB slices and similarly to the iterative designs, 1 BRAM to store the constants. The corresponding figures for the encryption architecture implemented with FAs are 23.5 Gbit/s at a clock speed of 92 MHz, 20,001 slices and 1 BRAM. The decryption architecture was only implemented with FAs as no significant benefit was evident in the performance of the iterative decryption architecture, which included CSAs. The decryption design runs at 21.4 Gbit/s and uses 17,197 CLB slices and 1 BRAM. Similar to the iterative encryption design, CSAs improve the speed of the pipelined encryption architecture at the expense of area. For both the iterative and pipelined designs, the encryption architectures are faster than the decryption architectures. This is because the critical paths in the decryption designs contain subtractions as described in Section 3.

Table 2 provides a summary of pipelined hardware implementations of a number of NESSIE finalist algorithms.

Although it is difficult to compare algorithms implemented on different devices, the SHACAL-2 algorithm is undoubtedly the fastest algorithm when implemented in hardware. The throughput of a hardware implementation depends on where the critical path of the design occurs. In some algorithm architectures this may occur in the non-linear function, but usually it is dependent on the architectural design. For example, in the SHACAL-2 hardware architecture described in this paper, the critical path falls in the nonlinear compression function, but when CSAs are utilised, the critical path is found to lie in the key schedule. In comparison to the other algorithms outlined in Table 2, SHACAL-2 only comprises logical operations, additions, rotations and shifts, all of which are well suited to hardware implementation and, thus, a high throughput is achieved. The other algorithms, with the exception of SHACAL-1, involve operations such as substitution boxes, modular multiplication, modular addition or Galois Field arithmetic. Although these operations can be manipulated for implementation on a hardware device, they will not result in higher throughputs than that achieved by simple logical functions.

It is interesting that SHACAL-2 outperforms SHACAL-1 in terms of performance since it also provides a higher number of bits of security. The underlying SHA-2 algorithm provides 128-bits of security whereas the SHA-1 function only provides 80 bits of security [7]. However, in the fully pipelined hardware architectures of both algorithms, SHACAL-2 only comprises 64 replications of the compression function, while SHACAL-1 consists of 80 replications. This coupled with the fact that SHACAL-2 operates on a block length of 256-bits as opposed to SHACAL-1's 160-bit block length, will result in SHACAL-2 providing a faster hardware architecture. SHACAL-2's 256-bit block length is also longer than the 64-bit/128-bit data blocks operated on by the other algorithms in Table 2.

To the author's knowledge the SHACAL-2 architecture is one of the fastest encryption algorithm implementations published to date. Also, even higher throughput rates are possible if implemented using ASIC technology. However, the pipelined SHACAL-2 architecture presented is also larger in area when compared to the other finalist implementations. Lower area designs are possible while still achieving high throughputs by unrolling the algorithm by a smaller number of compression functions. It is also possible to sub-pipeline the design by placing registers between the additions in the critical path and thus, reducing it to 1 FA. Although this will result in increased latency, the overall data throughput is likely to increase twofold. However, the area of the design will also increase significantly resulting from the additional registers. Therefore, determining which design approach to utilise will depend upon the application requirements and whether speed or area are important.

Table 2: Summary of pipelined hardware implementations of NESSIE finalists

Author	Algorithm	Device	Data-rate (Gbit/s)	Area
Pan *et al.* [11]	IDEA	XC2V1000	6.08	4 221 slices 34 multipliers
Standaert *et al.* [12]	Khazad	XCV1000	9.47	8 800 slices
Beuchat [13]	RC6	XC2V3000	15.2	8 554 slices 80 multipliers
McLoone *et al.* [8]	SHACAL-1	XC2V4000	17.02	13 729 slices
Denning *et al.* [14]	Camellia	XC2V4000	17.4	7 837 slices 88 BRAM
Rouvroy *et al.* [15]	MISTY1	XC2V2000	19.4	6 322 slices
Saggese *et al.* [16]	Rijndael	XCV2000E	20.2	5 810 slices 100 BRAM
This paper	SHACAL-2	XC2V4000	25.88	23 038 slices

5 Conclusion

The SHACAL-2 encryption algorithm is one of four private key block ciphers selected for the NESSIE portfolio of strong cryptographic primitives. The overall objective of the NESSIE initiative is to have a catalogue of algorithms that have been evaluated in terms of both security and performance. In the final report issued on the scheme, it was reported that the SHACAL-2 algorithm proved to be very secure with a large security margin [17]. Also, as the algorithm is derived from the US FIPS SHA-256 hash algorithm, its main components have already been subject to much cryptanalysis. In terms of performance, only figures for software implementations have been published to date. The algorithm performs encryption at a rate of 56 Mbit/s on an 800 MHz Pentium III processor [4]. In this paper, a performance analysis is conducted on hardware architectures of the algorithm. Highly efficient iterative SHACAL-2 encryption and decryption designs are presented. It is shown that higher encryption speeds are achieved if CSAs are used in the design of the compression function. A fully pipelined SHACAL-2 design is also presented, which operates at a phenomenal 25 Gbit/s on a XC2V4000 Virtex-II device and, as such, is one of the fastest encryption algorithm implementations currently available. Thus, with respect to the two main selection criteria, the SHACAL-2 algorithm is an ideal choice for the NESSIE portfolio.

6 Acknowledgments

This research has been funded under a Royal Academy of Engineering/EPSRC post-doctoral research fellowship.

7 References

1 DESCHALL Press Release, http://home.earthlink.net/~rcv007/despr4.htm, 2003
2 NESSIE IST-1999-12324, http://www.cryptonessie.org, 2003
3 Mitsubishi Electric Corporation, 'A description of the MISTY1 encryption algorithm'. Request for Comments (RFC) 2994, http://www.ietf.org/rfc/rfc2994.txt, 2003
4 Handshuh, H., Naccache, D.: 'SHACAL', Gemplus, http://www.gemplus.com/smart/r_d/publications/pdf/HN00shac.pdf, 2003
5 NTT & Mitsubishi Electric Corporation, 'Camellia', http://info.isl.ntt.co.jp/camellia/, 2003
6 Daemen, J., and Rijmen, V.: 'The design of Rijndael: AES-the advanced encryption standard' (Springer-Verlag, 2002)
7 US NIST 'Secure Hash Standard'. Draft FIPS PUB 180-2, May 2001
8 McLoone, M., and McCanny, J.V.: 'Very high speed 17 GbPs SHACAL encryption architecture'. 13th Int. Conf. Field Programmable Logic and Applications-FPL 2003, LNCS 2778, Portugal, 2003 pp. 111–120
9 Xilinx Virtex^{TM}-II Platform FPGAs, http://www.xilinx.com, Sept. 2002
10 McLoone, M., and McCanny, J.V.: 'Efficient single-chip implementation of SHA-384 and SHA-512'. IEEE Int. Conf. Field-Programmable Technology (FPT), Hong Kong, Dec. 2002, pp. 311–314
11 Pan, Z., Venkateswaran, S., Gurumani, S.T., and Wells, B.E.: 'Exploiting fine-grain parallelism present within the international data encryption algorithms using a Xilinx FPGA'. 16th Int. Conf. Parallel and Distributed Computing Systems (PDCS-2003), NV, USA, 2003
12 Standaert, F.X., and Rouvroy, G.: 'Efficient FPGA implementation of block ciphers Khazad and MISTY1'. 3rd NESSIE Workshop, November 2002, http://www.dice.ucl.ac.be/crypto/publications/2002/nessie3.pdf
13 Beuchat, J.L.: 'High throughput implementations of the RC6 block cipher using Virtex-E and Virtex-II devices'. INRIA Research Report, 2003, http://www.ens-lyon.fr/~jlbeucha/publications.html
14 Denning, D., Irvine, J., and Devlin, M.: 'A key agile 17.4 Gbit/sec Camellia implementation'. 14th Int. Conf. Field Programmable Logic and Applications-FPL 2004, LNCS 3203, Antwerp, Belgium, 2004, pp. 546–554
15 Rouvroy, G., Standaert, F.X., Quisquater, J.J., and Legat, J.D.: 'Efficient FPGA implementation of block cipher MISTY1'. The Reconfigurable Architecture Workshop (RAW 2003), Nice, France, 2003
16 Saggese, G.P., Mazzeo, A., Mazzocca, N., and Strollo, A.G.M.: 'An FPGA-based performance analysis of the unrolling, tiling and pipelining of the AES algorithm'. 13th Int. Conf. Field Programmable Logic and Applications - FPL 2003, Portugal, 2003, (*Lect. Notes Comput. Sci.*, **2778**, pp. 292–302)
17 NESSIE Security Report, Version 2.0, https://www.cosic.esat.kuleuven.ac.be/nessie/deliverables/D20-v2.pdf