Optimisation of the SHA-2 Family of Hash Functions on FPGAs

Robert P. McEvoy, Francis M. Crowe, Colin C. Murphy and William P. Marnane
Department of Electrical & Electronic Engineering,
University College Cork, Ireland
{*robertmce, francisc, cmurphy, liam*}@*rennes.ucc.ie*

Abstract

Hash functions play an important role in modern cryptography. This paper investigates optimisation techniques that have recently been proposed in the literature. A new VLSI architecture for the SHA-256 and SHA-512 hash functions is presented, which combines two popular hardware optimisation techniques, namely pipelining and unrolling. The SHA processors are developed for implementation on FPGAs, thereby allowing rapid prototyping of several designs. Speed/area results from these processors are analysed and are shown to compare favourably with other FPGA-based implementations, achieving the fastest data throughputs in the literature to date.

1. Introduction

In today's modern world of e-mail, internet banking, online shopping, and other sensitive digital communications, cryptography has become a vital tool for ensuring the privacy of data transfers. Hash functions operate at the root of many popular cryptographic methods in current use, such as the Digital Signature Standard (DSS), Transport Layer Security (TLS) and Internet Protocol Security (IPSec) protocols, numerous random number generation algorithms, encryption algorithms, all-or-nothing transforms, and password storage mechanisms.

Hash functions map messages of arbitrary length to a string of fixed length, called the 'message hash' or 'message digest'. This compression process is known as 'hashing'. In 2002, the National Institute of Standards and Technology (NIST) published the Secure Hash Standard [9], which detailed three new Secure Hash Algorithms SHA-256, SHA-384, and SHA-512. Since then, SHA-224 has been added to the standard, forming the 'SHA-2' family of hash functions. The SHA-2 family supersede the SHA-1 algorithm, whose security has been damaged by recent attacks [5].

This paper investigates architectures for hardware acceleration of the SHA-2 algorithms on FPGAs. FPGAs are suitable for use as cryptographic accelerators due to their low cost (relative to ASICs) and their flexibility when adopting security protocol upgrades. The reconfigurability of FPGAs also allows rapid prototyping of various VLSI designs.

This paper is organised as follows. The following section gives an overview of the algorithms in the SHA-2 standard. Section 3 surveys known techniques in the literature for hardware optimisation of SHA-2 operations. Novel SHA-2 architectures are presented in Section 4, whose speed/area performance results are discussed in Section 5. Conclusions are given in Section 6.

2. The Secure Hash Algorithms

Full descriptions of the SHA-224, -256, -384 and -512 algorithms can be found in the official NIST standard [9]. SHA-256 produces a 256-bit message hash; SHA-224 a 224-bit message hash etc. An overview of SHA-256 is given here, and then the differences between SHA-256 and the other members of the SHA-2 family are outlined. The SHA-256 algorithm essentially consists of 3 stages: (i) message padding and parsing; (ii) expansion; and (iii) compression.

2.1. Message Padding and Parsing

The binary message to be processed is appended with a '1' and padded with zeros until its length $\equiv 448 \bmod 512$. The original message length is then appended as a 64-bit binary number. The resultant padded message is parsed into N 512-bit blocks, denoted $M^{(1)}$, $M^{(2)}$, …, $M^{(N)}$. These $M^{(i)}$ message blocks are passed individually to the message expander.

2.2. Message Expansion

The functions in the SHA-256 algorithm operate on 32-bit words, so each 512-bit $M^{(i)}$ block from the padding stage is viewed as 16 32-bit blocks denoted

Proceedings of the 2006 Emerging VLSI Technologies and Architectures (ISVLSI'06)
0-7695-2533-4/06 $20.00 © 2006 IEEE

$M_t^{(i)}$, $0 \leq t \leq 15$. The message expander (also called the message scheduler) takes each $M^{(i)}$ and expands it into 64 32-bit W_t blocks, according to the equations:

$$\sigma_0(x) = ROT_7(x) \oplus ROT_{18}(x) \oplus SHF_3(x) \quad (1)$$

$$\sigma_1(x) = ROT_{17}(x) \oplus ROT_{19}(x) \oplus SHF_{10}(x) \quad (2)$$

$$W_t = \begin{cases} M_t^i & 0 \leq t \leq 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} & 16 \leq t \leq 63 \end{cases} \quad (3)$$

where the function $ROT_n(x)$ denotes a circular rotation of x by n positions to the right, whilst the function $SHF_n(x)$ denotes the right shifting of x by n positions. All additions in the SHA-256 algorithm are modulo 2^{32}.

2.3. Message Compression

The W_t words from the message expansion stage are then passed to the SHA compression function, or the 'SHA core'. The core utilises 8 32-bit working variables labelled A, B, ..., H, which are initialised to predefined values $H_0^{(0)}$–$H_7^{(0)}$ (given in [9]) at the start of each call to the hash function. Sixty-four iterations of the compression function are then performed, given by:

$$\begin{array}{ll} T_1 = H + \sum_1(E) + Ch(E,F,G) + K_t + W_t & \\ T_2 = \sum_0(A) + Maj(A,B,C) & \\ H = G & G = F \\ F = E & E = D + T_1 \\ D = C & C = B \\ B = A & A = T_1 + T_2 \end{array} \quad (4)$$

where

$$Ch(x,y,z) = (x\ AND\ y) \oplus (\bar{x}\ AND\ z) \quad (5)$$

$$Maj(x,y,z) = (x\ AND\ y) \oplus (x\ AND\ z) \oplus (y\ AND\ z) \quad (6)$$

$$\sum\nolimits_0(x) = ROT_2(x) \oplus ROT_{13}(x) \oplus ROT_{22}(x) \quad (7)$$

$$\sum\nolimits_1(x) = ROT_6(x) \oplus ROT_{11}(x) \oplus ROT_{25}(x) \quad (8)$$

and the inputs denoted K_t are 64 32-bit constants, specified in [9]. After 64 iterations of the compression function, an intermediate hash value $H^{(i)}$ is calculated:

$$H_0^{(i)} = A + H_0^{(i-1)}, H_1^{(i)} = B + H_1^{(i-1)}, \ldots, H_7^{(i)} = H + H_7^{(i-1)}$$

The SHA-256 compression algorithm then repeats and begins processing another 512-bit block from the message padder. After all N data blocks have been processed, the final 256-bit output, $H^{(N)}$, is formed by concatenating the final hash values:

$$H^{(N)} = H_0^{(N)} \ \&\ H_1^{(N)} \ \&\ H_2^{(N)} \ \&\ \ldots\ \&\ H_7^{(N)}$$

2.4. SHA-2 Algorithm Differences

Apart from having different output hash lengths, the only other difference between SHA-224 and SHA-256 is the set of K_t constants used. Similarly, the SHA-384 and SHA-512 algorithms differ only in their hash lengths and initial conditions. The SHA-512 algorithm has a similar structure to the SHA-256 algorithm, where: (i) it processes messages in blocks of 1024 bits rather than 512 bits; (ii) it uses 64-bit operations instead of 32-bit operations; (iii) the K_t constants are different; (iv) it iterates its compression function 80 times rather than 64 times; and (v) Equations (1), (2), (7) and (8) have different degrees of rotation and shifting, but are otherwise similar in structure.

3. Known SHA Optimisation Techniques

Several hardware-based SHA-2 designs have appeared in the literature [1-4, 6-8, 10, 12-13]. The longest data path (critical path) in the SHA core is the calculation of working variable A, which involves addition (modulo 2^{32} for SHA-224/256, modulo 2^{64} for SHA-384/512) of 7 operands (see Equation (4)). The following techniques have been proposed to speed up calculations in the SHA core:

- Using Carry-Save Addition (CSA) [2, 3, 4, 6, 7]. CSA separates the sum and carry paths, thus minimising the delay caused by traditional carry propagation. Since CSAs accept 3 input operands, A can be calculated in the SHA core using just 5 CSAs (as in [3]).

  A further 2-operand addition is required following the carry-save adders, to recombine the sum and carry paths. This extra addition stage is generally a fast carry look-ahead addition (CLA). In this work, the platform being targeted is a Xilinx™ Virtex II™ FPGA. By investigating different types of adders, it was found that the FPGA's built-in Carry-Propagate Adder (CPA) actually has a lower delay than a CLA on this platform. Hence, the designs in this paper use CPAs to terminate chains of CSAs and recombine the sum and carry paths, rather than CLAs.

- Unrolling [1, 6]. An unrolled architecture implements multiple rounds of the core compression function in combinational logic, thereby reducing the number of clock cycles required to compute the hash. For example, if the core was unrolled once, then the hash should be calculated in half the number of clock cycles. This comes at the cost of an increase in area and a decrease in clock frequency.

- (Quasi-) Pipelining. Dadda et al. [2, 3, 7] have proposed fast pipelined SHA-2 architectures, which use registers to break the long critical path within the SHA core. Pipelining is not trivial, however, since registers cannot be added at will (due to the inherent feedback in the SHA algorithm). External control circuitry is required to enable the registers correctly. Nevertheless, these 'quasi-pipelined' designs achieve very short critical paths, allowing message hashes to be calculated at high frequencies and high data throughputs.

- Delay Balancing. Dadda et al. also investigated using delay balancing in conjunction with CSA [3]. A CLA adder is used to combine the sum and carry paths as discussed above, but the sum and carry signals are first registered. This moves the CLA adder out of the critical path, but requires an extra register and accompanying control circuitry.
- Moving addition to the message expansion stage [13]. In this architecture, the first addition in the critical path (i.e. $K_t + W_t$) is moved to the message expansion stage, since both K_t and W_t are available before the other operands in Eq. (4). However, quasi-pipelining [2, 3, 7] achieves similar segregation of the operands, where the resultant paths are even shorter.
- Use of Block RAM (BRAM) for storage of constants [8]. Reconfigurable hardware devices such as FPGAs often have on-board memories which can be pre-loaded. Storing the K_t constants in these memories frees up space in the device which can then be used to implement extra logic. The free space also leads to improved routing and, thus, a general speed-up in circuit operation.
- Use of a Parallel Counter. 5-to-3 Parallel Counters (PCs) are used as adders in [4] instead of 3-to-2 CSA adders. The PCs reduce the number of bits at each position in the sum from 5 to 3. Equation (4) is realised in [4] with a cascade of two 5-to-3 PCs, followed by a CSA and a carry-propagate adder (CPA), giving a reduction in the critical path.

This paper investigates combining some of these optimisation techniques in order to increase the data throughput. In particular, the designs presented incorporate CSA, use of BRAM, quasi-pipelining, and unrolling. To the best of the authors' knowledge, this is the first time a pipelined-unrolled architecture for the SHA-2 family has been presented.

4. Hardware Architectures

4.1. Dadda et al.'s Constructions

In [2, 3], Dadda et al. investigate several quasi-pipelined architectures for the SHA-2 core. These architectures, originally proposed for ASIC implementation, were captured using VHDL and synthesised for an FPGA implementation (see Section 5). Of these designs, it was found that the scheme illustrated in Figure 1 gave the shortest critical path and, therefore, could be operated at the highest frequency.

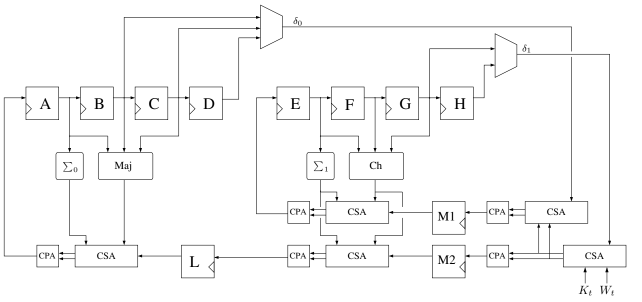


Figure 1. Dadda et al.'s [2] SHA-2 Core

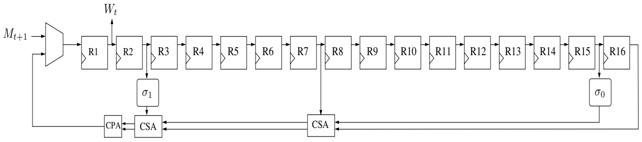


Figure 2. Basic SHA-2 Message Expander [2]

As well as registers to store the working variables A–H, the quasi-pipelined core (hereafter referred to as the "basic" core) includes 3 more registers (M1, M2, L) and 2 multiplexers. The select lines for the multiplexer outputs (δ_0, δ_1) and the clear signals for all the registers are controlled by external circuitry, as described in [2]. Since registers E–H receive their updates one clock cycle before registers A–D, the hash results $H_4^{(i)}$–$H_7^{(i)}$ are ready one clock cycle before $H_0^{(i)}$–$H_3^{(i)}$. It was found that the critical path in Figure 1 was from the multiplexer select line for δ_1 to the output of register M1.

In [2], an architecture for the message expander was presented which uses CSAs to reduce the number of required adders. Since the critical path in this design (see Figure 2) is shorter than that of the core, it is the core that determines how fast the overall hash algorithm can be executed. The authors also present a design employing delay balancing to further shorten the critical path in the message expander. However, this scheme is unsuitable for unrolling.

4.2. Combined Unrolled-Pipelined Design

This work investigates combining Dadda et al.'s fast quasi-pipelined architecture with the unrolling technique, with the aim of increasing the achieveable data throughput. Since the SHA processors were being designed for implementation on a reconfigurable platform, this allowed six different combinations to be investigated with relative ease.

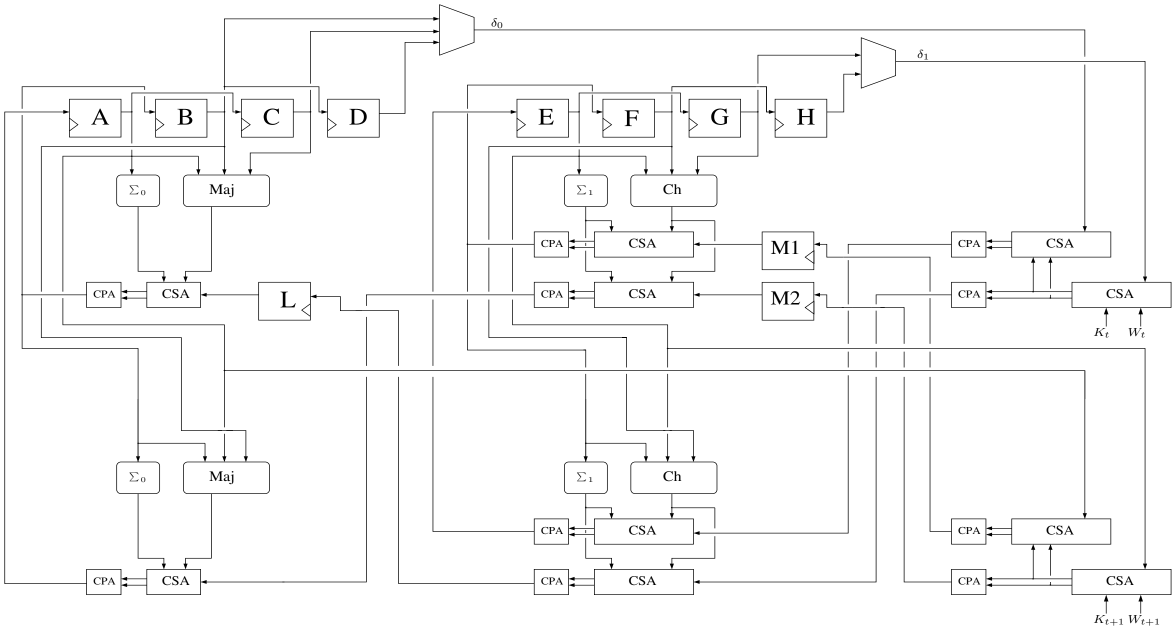


Figure 3. 2x-Unrolled-Pipelined SHA-2 Core

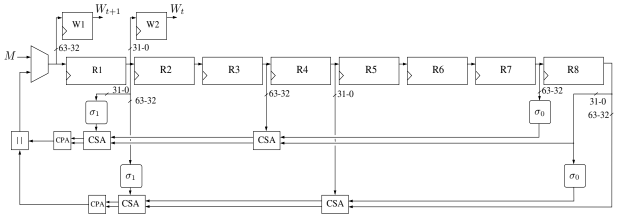


Figure 4. 2x-Unrolled SHA-2 Message Expander

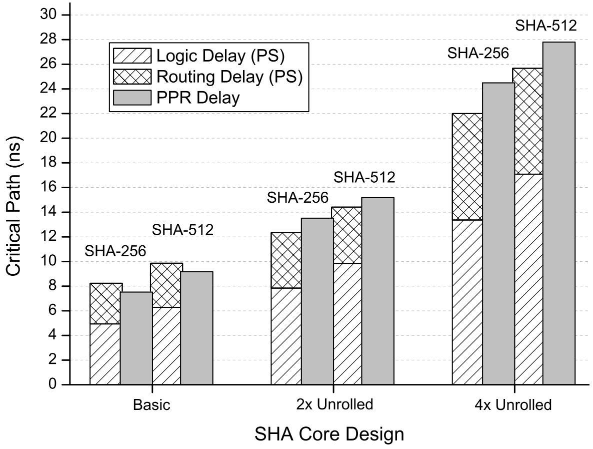
For both SHA-256 and SHA-512, basic (i.e. not unrolled), once unrolled ('2x'), and three times unrolled ('4x') cores were implemented.

Core. The quasi-pipelined SHA core (Figure 1) was unrolled by adding extra combinational logic to the datapaths, such that 2 (or 4) calculations of Equation (4) could be performed in one clock cycle. Figure 3 shows the corresponding 2x-unrolled architecture that was developed. Although the logical functions from the basic core (Figure 1) are duplicated, the registers and multiplexers are not. The registers are also loaded differently in the unrolled core than in the basic core. This unrolled circuit can only be used once the pipeline has been initially filled, i.e. once registers A–D have received their first values via the protocol in [2]. While the pipeline is being filled (and emptied), the basic core of Figure 1 must be used. The K_t constants are stored in BRAM to conserve space on the FPGA, and are addressed via a counter.

Expander. Since the 2x-unrolled core requires two W_t words to be available simultaneously, alterations to the basic message expander (Figure 2) were required, as shown in Figure 4. Firstly, the chain of 16 registers was modified to a chain of 8 registers of twice the size, e.g. for the SHA-256 message expander the registers changed from 32-bit to 64-bit. The multiplexer was also altered (in the SHA-256 case) to select 64-bit words rather than 32-bit words. This was necessary so that two new 32-bit W_t words would be produced on each clock cycle. Secondly, the combinational logic was duplicated so that two calculations of Equation (3) could be performed in one clock cycle. The results of these calculations are concatenated and fed back to the multiplexer. Finally, to prevent the message expander from contributing to the critical path in the core, the W_t words are registered before being passed to the unrolled SHA core. Note again that this unrolled circuit can only be used once the pipeline in the core has been filled.

Padding block. Most SHA designs in the literature assume that the message padding stage of the algorithm can be performed in software. However, here we aim to produce fast stand-alone implementations of the hashing algorithm, which could be used in constrained, embedded, or System-on-a-Chip (SoC) environments. Therefore, a message padding block was also included to implement the padding and parsing described in Section 2.1. This was realised as a synchronous finite state machine (FSM).

Figure 5. Critical Paths for SHA-2 Designs



Control Circuitry. External circuitry was designed to control the register load/clear signals in the core and the multiplexer select lines, as well as controlling the movement of data between the message padder, message expander and core. A second FSM was utilised for this purpose. The state machine works together with a counter to iterate the compression function (Equation (4)) appropriately, and to add the current hash value to the previous hash value. All control signals from the FSM were registered to ensure that they could not contribute to the critical path.

5. Results and Discussion

Six different SHA processors were captured at the RTL level using VHDL. For SHA-256, one processor had a basic quasi-pipelined core (as in [2]), one had a 2x-unrolled core, and another had a 4x-unrolled core. Three similar architectures were also developed for SHA-512 (and by extension SHA-224 and SHA-384). The design was synthesised using Xilinx ISE^{TM} tools v5.1 for implementation on a Xilinx Virtex II^{TM} *xc2v2000-bf957* FPGA. Post Synthesis (PS) and Post Place and Route (PPR) results for the critical paths in each SHA processor are displayed in Figure 5. The PS results are given in order to convey which proportions of the critical path are due to logic delay, and which to routing.

It is clear from Figure 5 that the critical path inside the core of each SHA processor increases with the degree of unrolling. In the basic designs (i.e. SHA processors based on cores from [2]), the PPR results are actually better than the PS results. This is because the synthesis tool only provides an estimate of the delay of the critical path. Since the percentage area occupied on the FPGA by the basic SHA processors is low, the PPR routing delay is not as high as was estimated by the synthesis tool.

In the 2x- and 4x-unrolled designs the PPR critical paths are longer than the PS estimates, as expected. The unrolled designs are larger than the basic designs and, as the FPGA device fills, it becomes more difficult for the Place and Route tool to route the signals along short paths. In the 4x-unrolled designs, approximately one third of the delay in the critical path is due to routing. It is reasonable to expect that these timing results could be further reduced by employing manual routing of the signals in the critical path.

The maximum clock frequency at which the processors can be operated is related to data throughput by:

$$Throughput = \frac{Block\ Size \times Max.\ Frequency}{\#\ Clock\ Cycles} \quad (9)$$

The block size for SHA-256 is 512 bits while the block size for SHA-512 is 1024 bits. Although the unrolled designs process messages in fewer clock cycles than the basic designs, the longer critical path in the unrolled designs means that the maximum clock frequency decreases. If the factor by which clock frequency decreases is less than that by which the number of clock cycles decreases, then an overall increase in data throughput will be achieved.

As noted in Section 2, the SHA-256 algorithm executes in 64 clock cycles, whilst SHA-512 executes in 80 clock cycles. Since the quasi-pipelined architectures include start-up and wind-down clock cycles, the basic quasi-pipelined processors execute the algorithms in 68 and 84 clock cycles respectively. In the new pipelined-unrolled designs presented in this paper, one must wait for the pipeline in the core to fill before activating the unrolled architecture. Similarly, at the end of a compression cycle, the unrolled architecture cannot be used, because two different halves of the intermediate hash $H^{(i)}$ appear in two consecutive clock cycles. Therefore, during start-up and wind-down, the control circuitry 'switches off' the unrolled section of the core and reverts to the basic configuration of Figure 1.

The number of clock cycles required by the compression algorithm in each architecture is shown in Table 1, along with the data throughput results (TP) attained from the six SHA-2 processors. The results show that unrolling the quasi-pipelined SHA-256 design provides no data throughput advantage. As the core is unrolled (from basic to 2x), the critical path length increases by a factor of 1.8. This is larger than the factor by which the number of clock cycles

Table 1. SHA-2 Processor Results

Design	Freq. (MHz)	Clk Cycles	TP (Mbps)	Area (slices)	TP/ Area
SHA-256					
Basic	133.06	68	**1009**	1373 (12%)	0.735
2x-unrolled	73.975	38	996.7	2032 (18%)	0.491
4x-unrolled	40.833	23	908.9	2898 (26%)	0.314
SHA-512					
Basic	109.03	84	1329	2726 (25%)	0.488
2x-unrolled	65.893	46	**1466**	4107 (38%)	0.357
4x-unrolled	35.971	27	1364	5807 (54%)	0.235

decreases (1.79). Therefore, it is the basic quasi-pipelined SHA-256 design (of [2]) that performs best. However, in the SHA-512 processor, the best throughput results were obtained from the 2x-unrolled design. Going from a basic quasi-pipelined SHA-512 design to the 2x-unrolled design reduces the number of clock cycles by a factor of 1.83, but the critical path increases by a factor of only 1.66. Therefore, an overall increase in throughput is obtained, at the cost of a 51% area increase. Note that here the area in FPGA CLB slices refers to the area of the entire processor, i.e. padding block, expander, compressor and control circuitry. Although the efficiency (TP/Area) of the 2x-unrolled SHA-512 processor is not as high as that for the basic processor, it may still be of use in high speed applications where area is not constrained.

Throughput comparisons amongst FPGA-based SHA-2 designs have recently been drawn in the literature [10, 12]. The highest reported data throughput for SHA-256 was 693 Mbps [13], and 1034 Mbps for SHA-512 [6], both targetting Virtex-E FPGAs, and using 1261 and 3517 slices respectively. For completeness, synthesis of our designs on the Virtex-E platform resulted in a (basic) SHA-256 processor with a throughput of 812 Mbps (1340 slices), and a (2x-unrolled) SHA-512 processor with a throughput of 1243 Mbps (3597 slices). In particular, these improvements arise due to the fast new quasi-pipelined core of [2], implemented for the first time on FPGAs.

Therefore, to the best of the authors' knowledge, the FPGA-based SHA-2 architectures presented in this paper achieve the best data throughput results to date. By comparison, software implementations using functions from the MIRACL cryptographic library [11] achieved average throughputs of only 59.8 Mbps for SHA-256 and 27.7 Mbps for SHA-512. Software implementations were run on a 3 GHz Pentium 4 processor, with 1 GB RAM. This highlights the considerable advantage of using VLSI hardware implementations to accelerate cryptographic algorithms and protocols.

6. Conclusion

In this paper, several hardware optimisation techniques for the SHA-2 hash family were explored. A new architecture was proposed for the SHA-2 core, which combines a fast quasi-pipelined design with unrolling. This is the first time a pipelined-unrolled SHA design has been investigated in the literature. This new design was then applied to both the SHA-256 and SHA-512 algorithms by constructing VHDL modules for SHA processors. These full processors include hardware for the SHA core, message scheduler, and message padder, thereby facilitating easy reuse in a VLSI SoC environment. Six processor designs were presented in the paper, each investigating different degrees of unrolling. Finally, the performances of these designs were considered from data throughput and area perspectives, and compared with other leading VLSI designs in the literature that target reconfigurable hardware. It was shown that the data throughput rates achieveable by the SHA processors presented in this paper out-perform these other designs.

Acknowledgement

This research was supported by the Embark Initiative, operated by the Irish Research Council for Science, Engineering and Technology (IRCSET).

References

[1] F. Crowe, A. Daly, and W. Marnane. Single-chip FPGA implementation of a cryptographic co-processor. In *Proceedings of the International Conference on Field Programmable Technology, FPT 2004*, pages 279–285, December 2004.

[2] L. Dadda, M. Macchetti, and J. Owen. An ASIC design for a high speed implementation of the hash function SHA-256 (384, 512). In *ACM Great Lakes Symposium on VLSI*, pages 421–425. ACM, 2004.

[3] L. Dadda, M. Macchetti, and J. Owen. The design of a high speed ASIC unit for the hash function SHA-256 (384, 512). In *DATE 2004*, pages 70–75. IEEE Computer Society, 2004.

[4] T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, T. Lehman, and B. Schott. Comparative analysis of the hardware implementations of hash functions SHA-1 and SHA-512. In *ISC*, volume 2433 of *Lecture Notes in Computer Science (LNCS)*, pages 75–89. Springer, 2002.

[5] A. K. Lenstra. Further Progress in Hashing Cryptanalysis (white paper). `http://cm.bell-labs.com/who/akl/hash.pdf`, February 2005.

[6] R. Lien, T. Grembowski, and K. Gaj. A 1 Gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512. In *CT-RSA 2004*, volume 2964 of *LNCS*, pages 324–338. Springer, 2004.

[7] M. Macchetti and L. Dadda. Quasi-pipelined hash circuits. In *Proceedings of the 17th IEEE Symposium on Computer Arithmetic, ARITH-17*, pages 222–229. IEEE, 2005.

[8] M. McLoone and J. McCanny. Efficient single-chip implementation of SHA-384 and SHA-512. In *Proceedings of the International Conference on Field Programmable Technology, FPT 2002*, pages 311–314, December 2002.

[9] NIST. Secure Hash Standard, FIPS PUB 180-2, 2002.

[10] A. Satoh and T. Inoue. ASIC-Hardware-Focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS. In *ITCC (1)*, pages 532–537. IEEE Computer Society, 2005.

[11] Shamus Software Ltd. Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL). `http://indigo.ie/~mscott/`.

[12] N. Sklavos and O. Koufopavlou. Implementation of the SHA-2 Hash Family Standard Using FPGAs. *The Journal of Supercomputing*, 31:227–248, 2005.

[13] K. K. Ting, S. C. L. Yuen, K.-H. Lee, and P. H. W. Leong. An FPGA based SHA-256 processor. In *FPL*, volume 2438 of *LNCS*, pages 577–585. Springer, 2002.