Comparative Analysis of the Hardware Implementations of Hash Functions SHA-1 and SHA-512

Tim Grembowski[1], Roar Lien[1], Kris Gaj[1], Nghi Nguyen[1],
Peter Bellows[2], Jaroslav Flidr[2], Tom Lehman[2], Brian Schott[2]

[1]Electrical and Computer Engineering, George Mason University, 4400 University Drive, Fairfax, VA 22030
{rlien, kgaj, nnguyen1}@gmu.edu
[2] University of Southern California - Information Sciences Institute
Arlington, VA 22203
{pbellows, jflidr, tlehman, bschott}@east.isi.edu

Abstract. Hash functions are among the most widespread cryptographic primitives, and are currently used in multiple cryptographic schemes and security protocols such as IPSec and SSL. In this paper, we compare and contrast hardware implementations of the newly proposed draft hash standard SHA-512, and the old standard, SHA-1. In our implementation based on Xilinx Virtex FPGAs, the throughput of SHA-512 is equal to 670 Mbit/s, compared to 530 Mbit/s for SHA-1. Our analysis shows that the newly proposed hash standard is not only orders of magnitude more secure, but also significantly faster than the old standard. The basic iterative architectures of both hash functions are faster than the basic iterative architectures of symmetric-key ciphers with equivalent security.

1 Introduction

Hash functions are very common and important cryptographic primitives. Their primary application is their use together with public-key cryptosystems in the digital signature schemes. They are also a basic building block of secret-key Message Authentication Codes (MACs), including the American federal standard HMAC [8]. This authentication scheme appears in two currently most widely deployed security protocols, SSL and IPSec [12, 16]. Other popular applications of hash functions include fast encryption, password storage and verification, computer virus detection, pseudo-random number generation, and many others [13, 16].

Cryptographically strong, collision-free, hash functions are very difficult to design. Tens of them have been proposed, and the majority of them have been broken. Only a few hash functions have gained a wider acceptance, and even fewer have been standardized.

By far the most widely accepted hash function is SHA-1 (Secure Hash Algorithm-1), a revised version of the American federal standard introduced in 1993 [4]. The original version of this function, SHA, was developed by National Security Agency

(NSA), and revised in 1995 for increased security even before any weakness was found in the open research.

SHA-1 was introduced as a federal standard about the same time as an 80-bit secret-key encryption algorithm named Skipjack [5] and the Digital Signature Standard (DSS) [6]. The security parameters of all these standards were chosen in such a way to guarantee the similar level of security, in the range of 2^{80} operations, as required by the best currently known attack.

After introducing a new secret-key encryption standard, AES (Advanced Encryption Standard) [7], with three key sizes, 128, 192, and 256 bits, the security of SHA-1 does not any longer match the security guaranteed by the encryption standard. Therefore, an effort was initiated by NSA to develop three new hash functions, with the security matching the security of AES with 128, 192, and 256 bit key respectively. This effort resulted in the publication of the draft Federal Information Processing Standard, introducing three new hash functions referred to as SHA-256, SHA-384, and SHA-512 [11].

The goal of the project described in this article was to implement the most complex of these new hash functions, SHA-512, in reconfigurable hardware, and to compare its implementation with the implementation of SHA-1, realized in the same technology. Our comparative analysis sought, among the other, answers to the following questions:

- does the increased security of the SHA-512 hash function come at the cost of decreased speed, increased area, or decreased speed to area ratio of the hardware implementations when compared to the SHA-1 hash function;
- how does the speed of the SHA-512 hash function compare to the speed of the corresponding versions of the AES algorithm? Which transformation, encryption or authentication, is faster in hardware? Which transformation requires less area?

Our investigation is a part of the larger project [10] aimed at implementing a hardware accelerator for a new suite of cryptographic algorithms to be used in the IP security protocol, IPSec. The target throughput of this accelerator is 1 Gbit/s for both encryption and authentication. Therefore we are also interested in studying the difficulty of implementing SHA-1 and the newly proposed hash functions at the speed of 1 Gbit/s using the current FPGA devices.

Although multiple commercial and academic implementations of SHA-1 have been reported and validated by NIST [15], we are not aware of any hardware implementation of SHA-512, or its comparison with the implementation of SHA-1 implemented in the same technology, using the same optimization techniques. This article is aimed at filling this gap.

2 Functional Comparison

In Table 1, four investigated hash functions are compared from the point of view of functional characteristics. The security of these hash functions is determined by the size of their outputs, referred to as hash values, *n*. The best known attack against these functions, the "birthday attack", can find a pair of messages having the same hash value with a work factor of approximately $2^{n/2}$. This complexity means that in order to

accomplish equivalent security, hash functions need to have an output twice as long as the size of a key of the corresponding secret-key cipher.

SHA-1 and SHA-256 have many features in common. They both can process messages with the maximum length up to 2^{64}-1 bits, have a message block size of 512 bits, and have internal structure based on processing 32-bit words. SHA-384 and SHA-512 have even more similarities. They process messages with the maximum length up to 2^{128}-1 bits, have a message block size of 1024 bits, and have internal structure based on processing 64-bit words. On top of that, the definition of SHA-384 is almost identical to the definition of SHA-512, with the exception of a different choice of the initialization vector, and a truncation of the final 512-bit result to 384 bits.

All functions have a very similar internal structure, and process each message block using multiple rounds. The number of rounds is the same for SHA-1, SHA-384, and SHA-512, and 20% smaller in SHA-256. The critical path in each round involves multioperand addition. SHA-1 requires two fewer operands per addition than in the remaining three functions.

A notation k+1 used in the table, means that the number of operands to be added is k in all but last round, and k+1 in the last round. Alternatively, a number of operands may be equal to k in all rounds, and an additional simplified round may be introduced for the remaining single addition.

Table 1. Functional characteristics of four investigated hash functions

	SHA-1	**SHA-256**	**SHA-384**	**SHA-512**
Size of hash value	160	256	384	512
Complexity of the best attack	2^{80}	2^{128}	2^{192}	2^{256}
Equivalently secure secret-key cipher	Skipjack	AES-128	AES-192	AES-256
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Message block size	512	512	1024	1024
Word size	32	32	64	64
Number of words	5	8	8	8
Number of digest rounds	80	64	80	80
Number of operands added in the critical path	5+1	7+1	7+1	7+1
Number of constants K_t	4	64	80	80
Round-dependent operations	f_t	None	None	None

The number of different constants is equal to four in SHA-1, and is the same as the number of rounds in all remaining functions. As a result, implementations of SHA-256, SHA-384, and SHA-512 must include a look-up table of constants, K_t, where t=0..number of rounds. SHA-1 is also the only function that contains an operation dependent on the round number t; in all remaining hash functions all rounds perform exactly the same operations.

The following conclusions can be derived from this functional comparison. Hardware implementations of SHA-384 and SHA-512 have exactly the same performance, so only one of them needs to be implemented for the purpose of comparative analysis. Notice that the size of the message block is twice as large in SHA-512 as compared to SHA-1, the number of rounds is the same, and the critical path is only slightly longer in SHA-512. Because of this, SHA-512 (the strongest function) is likely to be significantly faster than SHA-1 (the weakest function), which would be a very positive result if true. The throughput of SHA-256 is likely to be in the same range as a throughput of SHA-1, and smaller than the throughput of SHA-512. Taking into account these estimations, we have decided to implement two of the investigated hash functions, SHA-1 and SHA-512, which lay on the opposite ends of the spectrum in terms of both security and speed, with SHA-1 being the weakest and slowest, and SHA-512 being the strongest and fastest of the four investigated hash functions.

3 Design Methodology

Our target FPGA device was the Xilinx Virtex XCV-1000-6. This device is composed of 12,288 basic logic cells referred to as CLB (Configurable Logic Block) slices, includes 32 4-kbit blocks of synchronous dual-ported RAM, and can achieve synchronous system clock rates up to 200 MHz [17]. This device was chosen because of the availability of a general purpose PCI board based on three FPGA devices of this type. This board is described in detail in Section 5.

The design flow and tools used in our group for the implementation of cryptographic modules in Xilinx FPGA devices are shown in Fig. 1. All algorithms were first described in VHDL, and their description verified through the functional simulation using Active HDL v. 5.1, from Aldec, Inc. Test vectors and intermediate results from the reference software implementations based on the Crypto++ library [1] were used for debugging and verification of VHDL codes. The revised VHDL code became an input to the Xilinx integrated environment ISE 4.1i, performing the automated logic synthesis, mapping, placing, and routing. Tools included in this environment generated reports describing the area and speed of implementation, a netlist used for timing simulation, and a bitstream used to configure an actual FPGA device. All designs were fully verified through behavioral, post-synthesis, and timing simulations, and experimentally tested using the procedure described in Section 5.

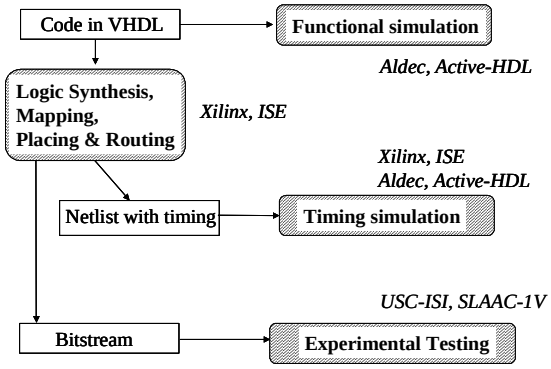

Fig. 1. Design flow and tools used in the development of cryptographic modules

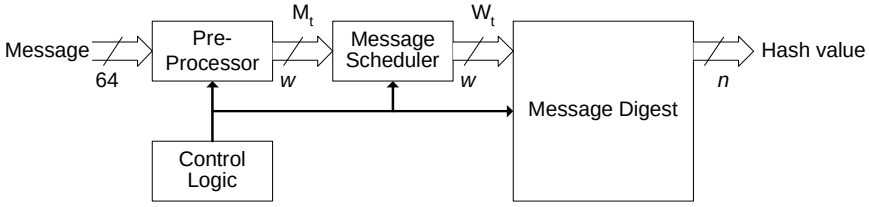

Fig. 2. General block diagram of SHA-1 and SHA-512. For SHA-1, w=32, n=160; for SHA-512, w=64, n=512

4 Hardware Architectures

A general block diagram common for all four hash functions is shown in Fig. 2. Input messages pass first through the preprocessing unit which performs padding and forms message blocks of the fixed length, 512 or 1024 bits, depending on the hash function. The preprocessing unit passes message blocks to the message scheduler unit. In our architecture, message blocks are passed to the message scheduler unit a word at a time, during the first 16 clock cycles used to process each message block. The message digest unit performs the actual hashing. It uses one clock cycle per digest round. In each round, the digest unit processes a new word W_t generated by the message scheduler unit.

The internal structure of the message digests for SHA-1 and SHA-512 are shown in Fig. 3ab. In both functions, input registers are initialized with the constant initialization vector, and are updated with the new value in each round. In SHA-1, four out of five words (A, B, C, and D) remain almost unchanged by a single round. These words are only shifted by one position down. The last word, E, undergoes a complicated transformation equivalent to multioperand addition modulo 2^{32}, with five 32-bit oper-

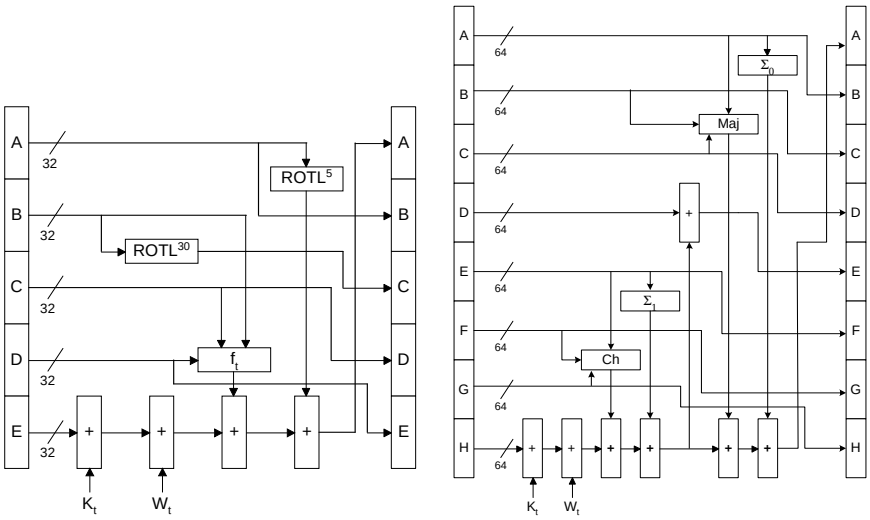

Fig. 3. Functional block diagram of the message digest unit of a) SHA-1, b) SHA-512

ands dependent on all input words, the round-dependent constant K_t, and the message dependent word W_t. The internal structure of the message digest of SHA-512 is similar. The primary differences are as follows: The number of words processed by each round is 8, each word is 64 bits long, and the longest path is equivalent to addition of seven 64-bit operands modulo 2^{64}. These operands depend on seven out of eight input words (all except D), the round-dependent constant K_t, and a message dependent word W_t. Six out of eight input words remain unchanged by a single round.

Our implementations of the message digests are shown in Figs. 4ab. The critical path in each circuit is marked with a thick line. Both circuits use the carry save representation of numbers to speed-up the multioperand addition, and minimize delays associated with carry propagation. The number of operands that need to be processed in each round has been minimized by precomputing the sum $K_t + W_t$ in the preceding clock cycle.

At the same time, the need for an additional round at the end of processing has been eliminated by introducing a conditional addition of the initial value of registers H_0-H_m (m=4 for SHA1, and m=7 for SHA-512) inside of each round. These initial values are added only in the last round of the message digest computations; in all previous rounds zero is added instead. After these two optimizations, the maximum number of operands to be added in each round is 5 for SHA-1 and 7 for SHA-512.

The straightforward use of carry save adders in case of five operand addition would lead to three levels of 3-to-2 carry save adders, followed by a carry propagate adder as shown in Fig. 5a. Instead, we have decided to use a 5-to-3 parallel counter (see Fig. 5b) [14], which reduces the number of binary digits at each position in the sum of five operands from 5 to 3, and has approximately the same delay as a 3-to-2 carry save adder. The operation of the 5-to-3 parallel counter is shown in Fig. 5c, using the dot notation. In this notation, each dot represents a binary digit, 0 or 1 [14]. The 5-to-3

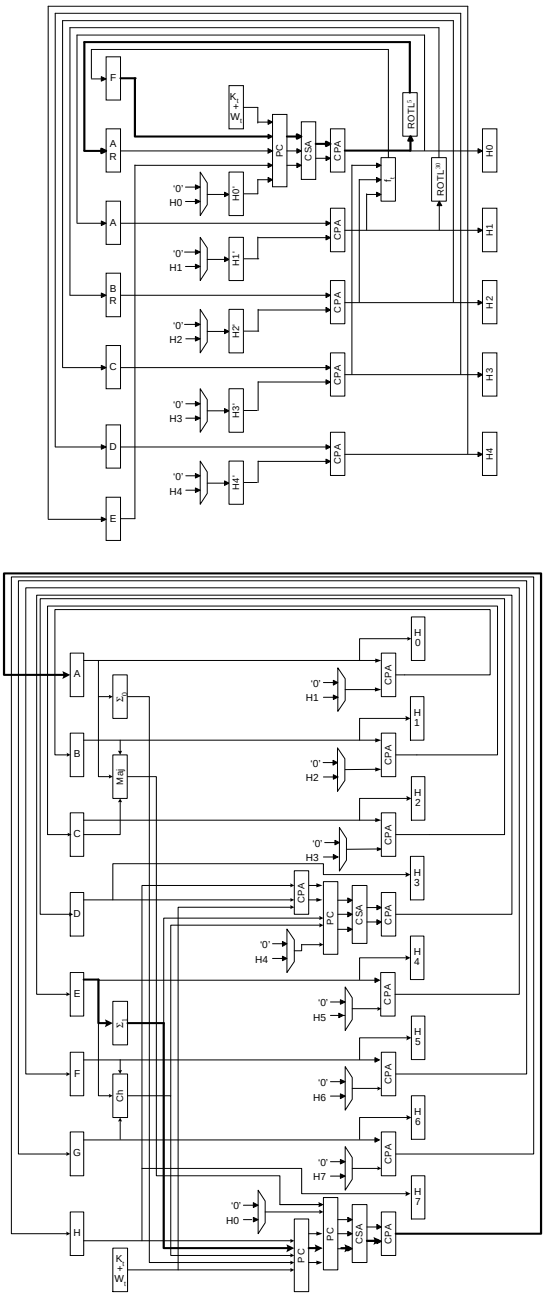

Fig. 4. Our implementations of the message digest units of a) SHA-1, b) SHA-512

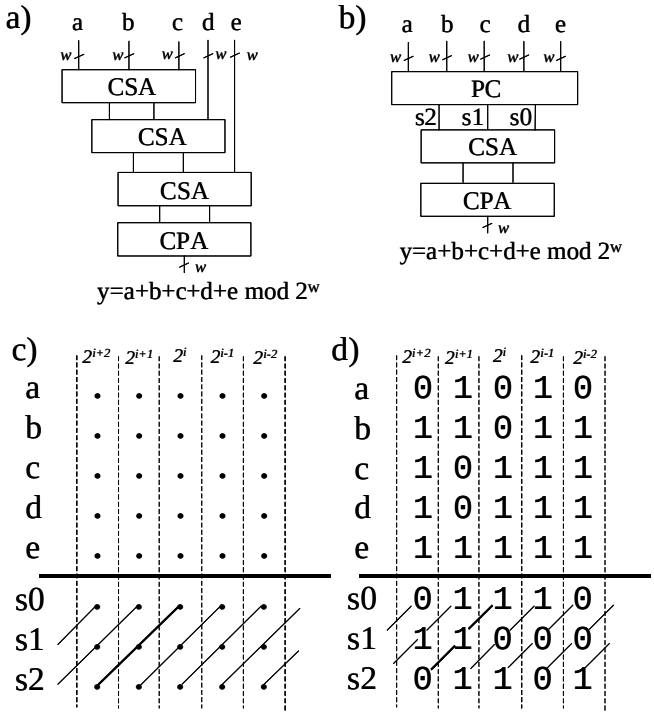

Fig. 5. Using 5-to-3 Parallel Counter. a) adding five *w*-bit numbers using a tree of 3-to-2 carry-save adders, b) adding five *w*-bit numbers using 5-to-3 parallel counter followed by a 3-to-2 carry save adder, c) operation of the 5-to-3 parallel counter in the dot notation, d) example of the operation of the 5-to-3 parallel counter

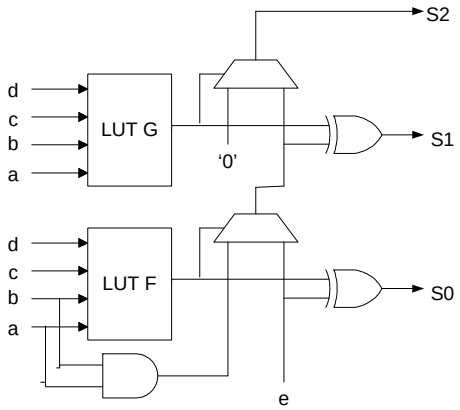

Fig. 6. Using internal structure of a single CLB slice of the Xilinx Virtex FPGA device to implement a bit-slice of a 5-to-3 Parallel Counter (PC)

parallel counter adds five binary digits with the same weight, 2^i, and represents the result using three binary digits with three subsequent weights, 2^i, 2^{i+1}, and 2^{i+2}. An example of the operation of this counter is shown in Fig. 5d. The speed-up comes from the fact that the operation of the parallel counter can be realized in Virtex FPGAs using resources of a single CLB slice as shown in Fig. 6.

In SHA-512, a cascade of two 5-to-3 parallel counters is used to reduce the number of operands from seven to three (see Fig. 4b). As a result, the critical path is longer than in SHA-1 only by two levels of CLB slices (one level for the parallel counter, and one for the $\sum_1$ operation).

Further optimization of the critical path in both circuits has been accomplished by reducing the delays of interconnects. The primary optimization technique used for that purpose was the reduction of the fan-out of control signals by using buffers, duplicating portions of control logic, and placing control logic close to the controlled parts of the execution unit.

The block diagrams of the message scheduling units in SHA-1 and SHA-512 are shown in Fig. 7. Both units generate 80 message dependent words, W_t, t=0..79. The first 16 of these words, $W_0..W_{15}$, is simply the first 16 words of the input message block, $M_0..M_{15}$; the remaining words are computed using a simple feedback function, based on rotations, shifts, and XOR operations. The actual implementation of both functions is given in Fig. 8. Our implementations have been optimized for minimum area, using a shift register mode of CLB slices available in the Xilinx Virtex FPGA devices. Using this mode, a cascade of several one-bit registers, each taking normally a single CLB slice, can be reduced to a single CLB slice implementing the multi-stage shift register with up to 16 stages.

5 Testing Procedure

The experimental testing of our cryptographic modules was performed using the SLAAC-1V hardware accelerator board. The logical architecture of SLAAC-1V is shown in Fig. 9. The three Virtex 1000 FPGAs (denoted as X0, X1, and X2) are the primary processing elements.

About 20% of the resources in the X0 FPGA are devoted to the PCI interface and the board control module. The remaining logic of this device, as well as the entire X1 and X2 FPGAs, can be used by the application developer. The board control module implemented in X0 provides high-speed DMA (Direct Memory Access), data buffering, clock control (including single-stepping and frequency synthesis from 1 to 200 MHz), etc. The current 32 bit 33 MHz control module has obtained DMA transfer rates of over 1 Gbit/s (125 MB/s) between X0 and the host memory, very near the PCI theoretical maximum.

In all our experiments, the X1 FPGA was configured to contain cryptographic modules, while X0 and X2 were used only to facilitate the transfer of data between X1 and the memory of the host computer running Linux.

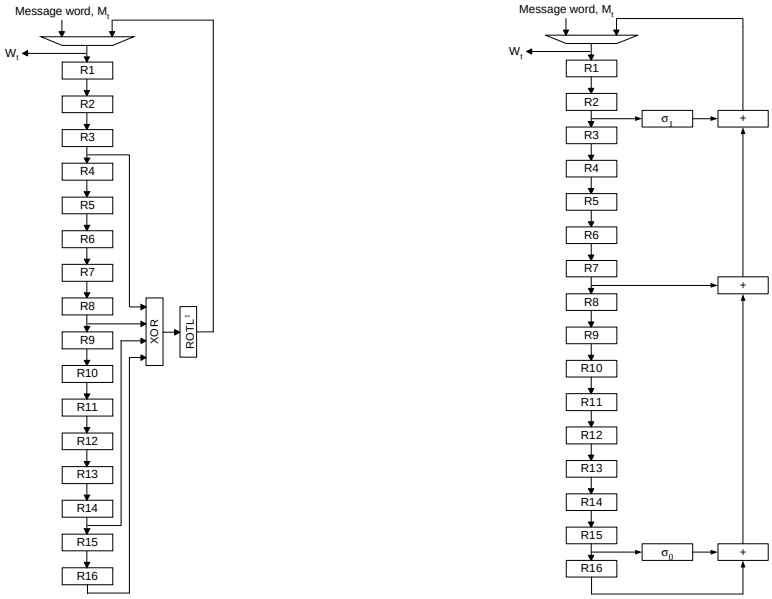

Fig. 7. Functional block diagrams of the message scheduler unit of a) SHA-1, b) SHA-512

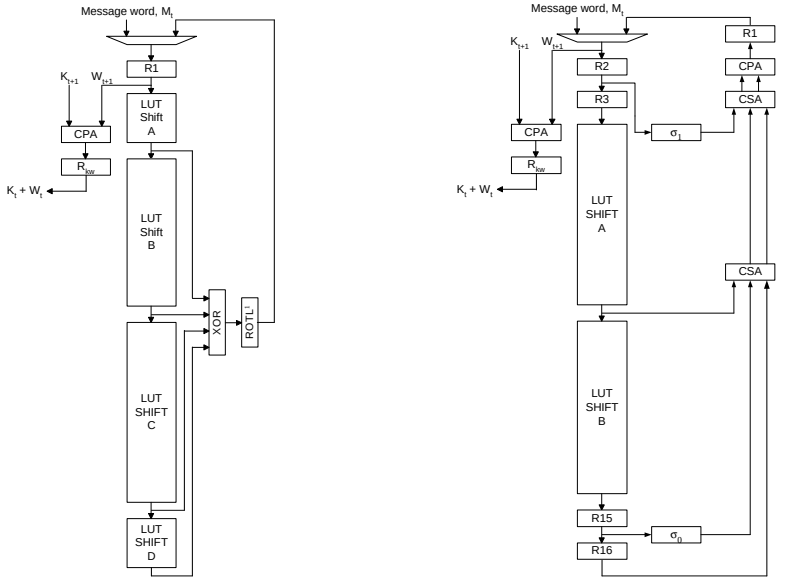

Fig. 8. Our implementations of the message scheduler unit of a) SHA-1, b) SHA-512

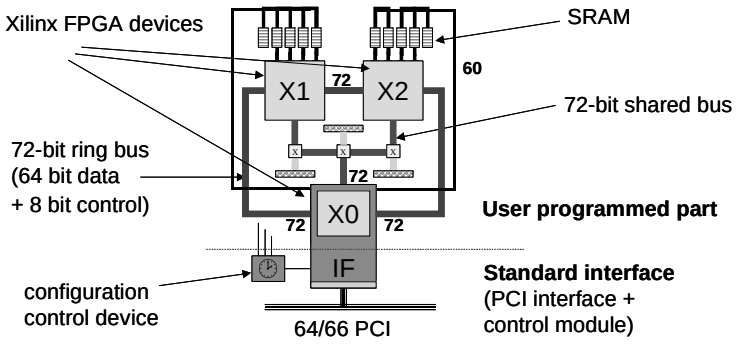

Fig. 9. SLAAC-1V Architecture

The test program written in used the SLAAC-1V APIs and the SLAAC-1V driver to communicate with the board.

Our testing procedure is composed of three groups of tests. The first group attempts to verify the circuit functionality at a single clock frequency. The goal of the second group is to determine the maximum clock frequency at which the circuit operates correctly. Finally, the purpose of the third group is to determine the limit on the maximum encryption and decryption throughput, taking into account the limitations of the PCI interface.

Our first group of tests is based on the NIST recommendations provided in [2]. These recommendations describe the comprehensive suite of three functional tests for SHA-1.

The second test is aimed at determining the maximum clock frequency of the hash function modules. Three megabytes of pseudorandomly generated data are sent to the board for hashing, the result is transferred back to the host and compared with the corresponding output obtained using software implementation of the given hash function based on the Crypto++ library [1]. This procedure is repeated 30 times using the same clock frequency to minimize the effect of input data values on the results of analysis. The next clock frequency is chosen based on the rules of the binary search, i.e., in the middle between two closest earlier identified frequencies giving different test results. The test is repeated until the difference between these two frequencies is smaller than the required accuracy of the measurement (< 0.1 MHz in our tests). The highest investigated clock frequency at which no single processing error is detected is considered the maximum clock frequency. In our experiments, this test was automatically repeated 10 times with consistent results in all iterations.

The third group of tests is an extension of the second group. After determining the maximum clock frequency, we measure multiple times and average the amount of time necessary to process 3 MB of data, taking into account the delay contribution of the 32 bit/33 MHz PCI interface.

6 Results

In Fig. 10, the minimum clock periods of SHA-1 and SHA-512 obtained using static timing analysis and experiment are given. For clock periods determined through static timing analysis, the percentage of the critical path delay used by logic and routing respectively is shown.

Based on the knowledge of the minimum clock period, the maximum data throughput has been computed according to the equation:

*Throughput = Message_block_size / (Clock period * Number_of_rounds)*

Throughput values calculated based on the minimum clock periods obtained using static timing analysis and experiment are shown in Fig. 11. In the same figure, these

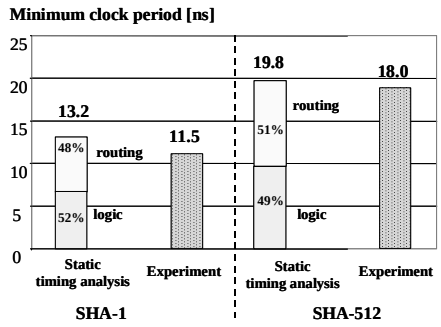

Fig. 10. Minimum clock period of SHA-1 and SHA-512: a) obtained using static timing analysis, b) determined experimentally

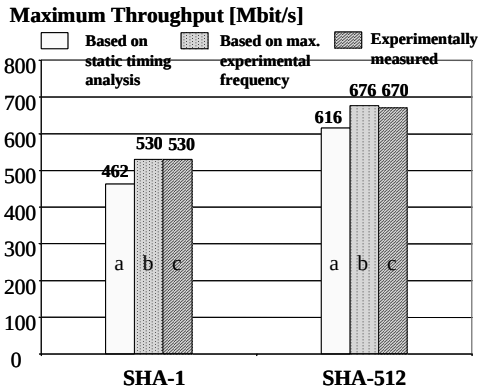

Fig. 11. Maximum throughputs of SHA-1 and SHA-512: a) obtained using static timing analysis, b) calculated based on the experimentally measured maximum clock frequency, c) experimentally measured, including the contributions of the PCI interface

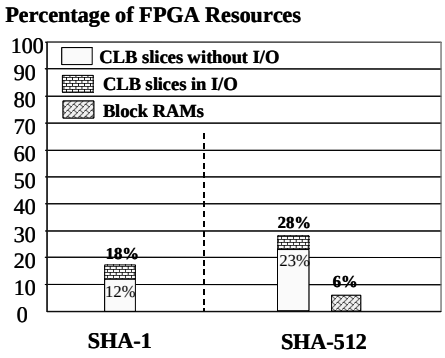

Fig. 12. Percentage of the FPGA resources used by each implementation

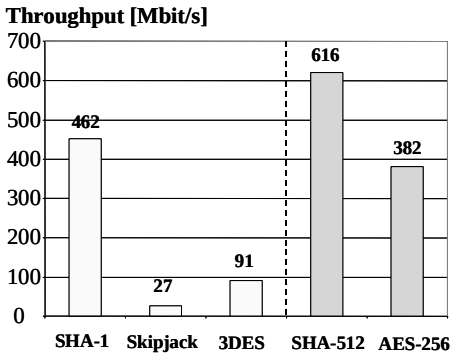

Fig. 13. Comparison of throughputs for the basic iterative architectures of old and new standards in the area of hash functions and symmetric-key ciphers

results are compared with the experimentally measured data throughputs that take into account the delay contributions of the PCI interface. This comparison demonstrates that the PCI interface provides a constant uninterrupted flow of data and has a negligible influence on the data throughput.

Our results confirm our earlier predictions that the design of strong hash functions does not involve any major trade-off between security and performance. To the contrary, the most secure function, SHA-512, is also the fastest of four investigated hash functions.

The percentage of the FPGA resources (CLB slices and Block RAMs) used by implementations of SHA-1 and SHA-512, are shown in Fig. 12. The difference in the number of CLB slices is primarily caused by the difference in the size of input and output registers in the message digest units of both functions (512 bits vs. 160 bits), and the width of the multioperand adders in the critical path of these units (64 bits vs. 32 bits). In SHA-512, two 4 kbit block RAMs are used to store 80 64-bit constants K_t.

7 Possible Extensions

The analysis of our results, reveals a potential for further optimizations. Since a large percentage of the critical path delay (48% in SHA-1 and 51% in SHA-512) is contributed by delays of interconnects, a substantial gain can be accomplished by manual floorplaning and routing. Since these optimizations are specific for a given type of the device, and are not easily transferable to another family of FPGA devices, they have not been attempted at this point.

A further radical improvement of the circuit speed can be achieved by using an unrolled architecture of the message digest unit, in which m (m=2, 4, 5, or 8) digest rounds are implemented as combinational logic and executed in the same clock cycle. As a result, the total number of clock cycles necessary to compute a digest for a single message block is reduced by a factor of m, at the cost of a significantly smaller increase in the delay of the critical path. The preliminary study of this extended architecture for SHA-1 and m=5 indicates that the delay of logic in the critical path increases by a factor smaller than 2.5, leading to the overall increase in the circuit throughput by a factor of 2. This preliminary result needs to be verified, taking into account the delays of interconnects, and will be reported in a future article together with results of a similar optimization of SHA-512.

Another popular way of speeding up the hardware implementations of cryptographic transformations is parallel processing, using either several independent execution units or pipelining. Even taking into account limitations imposed by the area of FPGA devices, pipelining was shown to permit speeding up the implementations of AES and other secret key ciphers by at least an order of magnitude [3, 9]. Taking into account the relatively smaller area required by the basic implementations of SHA-1 and SHA-512, a potential speed-up is even greater in case of hash functions.

Unfortunately, applying parallel processing to hash functions is limited by the fact that only input blocks belonging to *different* messages may be processed in parallel. In this respect, hashing is similar to encryption in the CBC (Cipher Block Chaining) mode and other *feedback* modes of secret-key block ciphers. The processing of the next message block cannot start before the processing of the previous block is fully completed. Additionally, hash functions do not possess any non-feedback modes of operation, such as ECB (Electronic CodeBook) or counter mode. Therefore, pipelining, although possible, is limited by the availability of multiple independent messages that could be processed in parallel. This availability is application specific and may strongly depend on the characteristic of the network traffic, e.g., an average size of packets exchanged in the given communication protocol.

8 Summary

An FPGA implementation of the newly proposed draft hash standard SHA-512 has been developed and compared with the implementation of the old hash standard SHA-1. An effort was made to use exactly the same technology and identical design and optimization techniques. Our implementations based on Xilinx XCV-1000-6 demon-

strate that SHA-512 is 33% faster than the equivalent implementation of SHA-1 according to the static timing analysis, and 26% faster according to the experiment. At the same time, without taking into account an input/output interface, SHA-512 takes almost twice as many CLB slices as SHA-1, and requires two additional 4 kbit Block RAMs. These results have been verified experimentally using the PCI FPGA Board, SLAAC-1V, based on three Xilinx FPGA devices Virtex 1000. Our results prove that the design of a strong hash function does not necessarily involve a trade-off between the hardware speed and cryptographic security. At the same time, the more secure hash function may require substantially more hardware resources.

Further optimizations of both implementations based on loop unrolling are possible and will be reported in a future article. Our research is a part of a larger effort aimed at implementing the newly proposed cryptographic algorithms of IPSec in the form of a giga-bit rate hardware accelerator on a Xilinx FPGA-based PCI card [10].

References

1. Crypto++, free C++ class library of cryptographic schemes, available at http://www.eskimo.com/~weidai/cryptlib.html.
2. Digital Signature Standard Validation System (DSSVS) User's Guide available at http://csrc.nist.gov/cryptval/shs.html
3. Elbirt, A. J., Yip, W., Chetwynd, B., Paar, C.: An FPGA implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. Proc. 3rd Advanced Encryption Standard (AES) Candidate Conference, New York, April 13-14, 2000.
4. http://www.itl.nist.gov/fipspubs/fip180-1.htm
5. FIPS 185, Escrowed Encryption Standard (EES), February 1994.
6. FIPS 186-2, Digital Signature Standard (DSS), February 2000, available at http://csrc.nist.gov/encryption/tkdigsigs.html
7. NIST, FIPS Publication 197, Specification for the Advanced Encryption Standard (AES), November 26, 2001, available at http://csrc.nist.gov/encryption/aes/.
8. FIPS 198, HMAC - Keyed-Hash Message Authentication Code, available at http://csrc.nist.gov/encryption/tkmac.html
9. Gaj, K., and Chodowiec, P.: Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard Using Field Programmable Gate Arrays, Proc. RSA Security Conference - Cryptographer's Track, April 2001.
10. GRIP (Gigabit Rate IP Security) project page, available at http://www.east.isi.edu/projects/GRIP/
11. NIST Cryptographic Toolkit, Secure Hashing, available at http://csrc.nist.gov/encryption/tkhash.html
12. IP Security Protocol (ipsec) Charter - Latest RFCs and Internet Drafts for IPSec, http://ietf.org/html.charters/ipsec-charter.html
13. Menezes, A. J., van Oorschot P. C., and Vanstone S. A.: Handbook of Applied Cryptography, CRC Press, Inc., Boca Raton, 1996.
14. Parhami, B.: Computer Arithmetic: Algorithms and Hardware Design, Oxford University Press, 2000.
15. SHS Validation List, available at http://csrc.nist.gov/cryptval/shs/shaval.htm.
16. Stallings, W.: Cryptography and Network Security, 1999 Prentice-Hall, Inc., Upper Saddle River, New Jersey. 2nd Edition.
17. Xilinx, Inc.: Virtex 2.5 V Field Programmable Gate Arrays, available at www.xilinx.com.