Design and Implementation of a SHA-1 Hash Module on FPGAs

Kimmo Järvinen

Otakaari 5A, Espoo
FIN-02150, Finland
GSM:+358-40-7384675
kimmo.jarvinen@hut.fi

November 25, 2004

Abstract

This technical report presents an efficient implementation of the commonly used hash algorithm SHA-1. The SHA-1 algorithm is widely used in various public-key cryptography algorithms, and therefore efficient hardware implementation of SHA-1 is of great importance. A thorough presentation of the implementation techniques is presented. The design was implemented on a Xilinx Virtex-II XC2V2000-6 FPGA device, and it required 1275 slices, operated at a clock frequency of 117.6 MHz achieving a throughput of 734 Mbps, respectively. The design is compared to a published design of MD5 hash algorithm and their performance and logic requirements are compared. The SHA-1 design is also compared to other open-literature FPGA-based SHA-1 implementations, and it is concluded that it is among the fastest and smallest SHA-1 FPGA implementations.

© Kimmo Järvinen, 2004, kimmo.jarvinen@hut.fi

1 Introduction

This report describes an efficient hardware implementation of the SHA-1 hash algorithm [7] which is a commonly used algorithm in cryptography. The implementation was designed using similar methods that were used in the implementation of the MD5 hash algorithm [12] which is to be published in [9]. The design is called SIG-SHA-1, where SIG is an acronym for the Signal Processing Laboratory at Helsinki University of Technology. SIG-SHA-1 is made in order to compare hardware implementations of SHA-1 and MD5. The design was used also in the evaluation of a combined MD5/SHA-1 module described in [8]. SIG-SHA-1 is a straightforward implementation of the SHA-1 specifications available in [7], and it performs well in both required area and performance.

Field Programmable Gate Arrays (FPGAs) are almost ideal candidates for implementation platforms of cryptographic algorithms, because they combine the speed of hardware with the flexibility of software. Several benefits of cryptographic algorithms on FPGAs are listed and analyzed in [16].

In the implementation described in this report, FPGA devices manufactured by Xilinx are used. Xilinx Virtex-II XC2V2000-6 FPGA device is used as an implementation platform for the presented design. Virtex-II device family offers both fast performance and large logic resources [17].

Hash algorithms, also commonly called as message digest algorithms, are algorithms generating a unique fixed-length bit vector for an arbitrary-length message $\mathcal{M}$. The bit vector is called the hash of the message and it is here denoted as $\mathcal{H}$. The hash can be considered as a fingerprint of the message.

There are several essential features that a hash algorithm must have. First, $\mathcal{H}$ must be easy to compute for every given $\mathcal{M}$. Second, it must be hard to compute $\mathcal{M}$ when $\mathcal{H}$ is given. Third, it must be hard to find another message $\mathcal{M}'$ which has the same $\mathcal{H}$ as $\mathcal{M}$. [13] Here, the term 'hard' means computationally infeasible.

Secure Hash Algorithm (SHA) is described in the National Institute of Standards and Technology's (NIST) Federal Information Processing Standard (FIPS) 180-2: Secure Hash Standard (SHS) [7]. SHS describes the following algorithms: SHA-1 (SHA-160), SHA-256, SHA-385 and SHA-512, where the number is the length of the hash $\mathcal{H}$ in bits. In this report, only SHA-1 (SHA-160) is considered. SHA-1 is widely used in various public-key cryptographic algorithms, e.g. in Digital Signature Algorithm (DSA) [6].

This report is organized as follows: first, the SHA-1 algorithm is introduced in Section 2. Design and implementation of the SHA-1 module is considered in Section 3 and the results of the implementation are presented in Section 4. Short comparisons to both MD5 and other published SHA-1 implementations is given in Section 5. Finally, conclusions are made in Section 6.

2 SHA-1 Algorithm

SHA-1 is a part of the FIPS 180-2: Secure Hash Standard [7]. It is very widely used in public-key cryptography, especially in message authentication schemes.

SHA-1 calculates a 160-bit $\mathcal{H}$ for a b-bit $\mathcal{M}$. The algorithm consists of the following steps:

1. **Appending Padding Bits**
   The b-bit $\mathcal{M}$ is padded in the following manner: a single 1-bit is added into the end of $\mathcal{M}$, after which 0-bits are added until the length of the message is congruent to 448, modulo 512.

2. **Appending Length**
   A 64-bit representation of b is appended to the result of the above step. Thus, the resulted message is a multiple of 512 bits.

3. **Buffer Initialization**
   Let H_0, H_1, H_2, H_3 and H_4 be 32-bit hash value registers. These registers are used in the derivation of a 160-bit hash $\mathcal{H}$. At the beginning, they are initialized as follows:

$$\begin{array}{rcl} H_0 & = & \text{x}''67452301'' \\ H_1 & = & \text{x}''efcdab89'' \\ H_2 & = & \text{x}''98badcfe'' \\ H_3 & = & \text{x}''10325476'' \\ H_4 & = & \text{x}''c3d2e1f0'' \end{array} \tag{1}$$

4. **Processing of the message (the algorithm)**
   The algorithm which is used for processing of the padded message is described next. First, the padded message needs to be divided into 512-bit blocks, denoted here as M_j where $j \geq 0$ is the index of the block. The algorithm processes one M_j at once, starting from M_0, until all M_j have been processed.

   Five 32-bit registers, A, B, C, D and E are defined. At the beginning of processing of each M_j their values are set as follows: $A \leftarrow H_0$, $B \leftarrow H_1$, etc.

   The algorithm consists of 80 steps. Let t denote the index of a step, i.e. $0 \leq t \leq 79$. First, a 32-bit message block W_t is derived for every step t from the 512-bit message block M_j using a message schedule. For $t < 16$, W_t is simply the tth 32-bit word of M_j. When $t \geq 16$, W_t are derived recursively with the following formula:

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1 \tag{2}$$

where $\lll$ denotes circular shift to the left by s bits and $\oplus$ is a logical xor-operation. Let K_t be a constant value for step t. The values of K are set as follows:

$$K_t = \begin{cases} \text{x}''5a827999'' & 0 \le t \le 19 \\ \text{x}''6ed9eba1'' & 20 \le t \le 39 \\ \text{x}''8f1bbcdc'' & 40 \le t \le 59 \\ \text{x}''ca62c1d6'' & 50 \le t \le 79 \end{cases} \tag{3}$$

A function $F(X,Y,Z)$ depending on the step t is defined as follows

$$F(X,Y,Z) = \begin{cases} (X \wedge Y) \oplus (\neg X \wedge Z) & 0 \le t \le 19 \\ X \oplus Y \oplus Z & 20 \le t \le 39 \\ (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z) & 40 \le t \le 59 \\ X \oplus Y \oplus Z & 60 \le t \le 79 \end{cases} \tag{4}$$

where $\wedge$, $\oplus$ and $\neg$ are bitwise logical and, xor and complement, respectively.

The message is processed for $0 \le t \le 79$ with the following function, which is here called the SHA-1 step function:

$$T = (A \lll 5) + F(B,C,D) + W_t + K_t + E \tag{5}$$

where $+$ denotes an addition modulo 2^{32}. After each step, the values of the registers are set as follows:

$$\begin{aligned} A &\leftarrow T \\ B &\leftarrow A \\ C &\leftarrow B \lll 30 \\ D &\leftarrow C \\ E &\leftarrow D \end{aligned} \tag{6}$$

Finally, when all 80 steps have been processed, the following operations are performed:

$$\begin{aligned} H_0 &\leftarrow H_0 + A \\ H_1 &\leftarrow H_1 + B \\ H_2 &\leftarrow H_2 + C \\ H_3 &\leftarrow H_3 + D \\ H_4 &\leftarrow H_4 + E \end{aligned} \tag{7}$$

If all M_j have been processed, the algorithm is terminated. Otherwise, the algorithm is processed with M_{j+1}.

5. **Output**
   When all M_j have been processed with the above algorithm, the 160-bit hash $\mathcal{H}$ of $\mathcal{M}$ is available in H_0, H_1, H_2, H_3 and H_4.

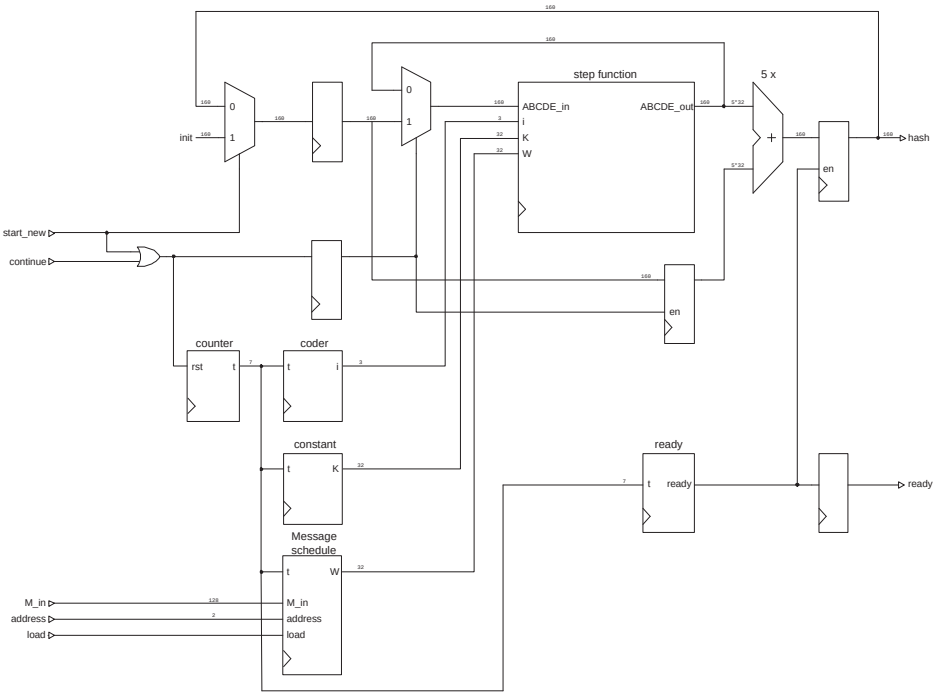

Figure 1: A top-level block diagram of SIG-SHA-1

3 Implementation

The goal of the design of SIG-SHA-1 was to make an implementation comparable to the SIG-MD5-I design presented in [9] so that the logic requirements and performance of MD5 and SHA-1 could be easily compared. The iterative structure was chosen in order to make a compact structure which could be used in the evaluation of the combined MD5/SHA-1 block introduced in [8].

The top-level architecture used for SIG-SHA-1 implementation is almost similar to the architecture used for SIG-MD5 implementations in [9]. A block diagram of SIG-SHA-1 is presented in Figure 1. SIG-SHA-1 implements only the steps 3–5 presented in Section 2, because padding of $\mathcal{M}$ is fast to perform also with software, and thus it does not require hardware acceleration.

The critical path of the implementation includes the step function block and the multiplexer in front of it. Thus, an efficient implementation of the step function is essential for a high performance hardware implementation of SHA-1. The calculation of the algorithm can be speeded up by unrolling several steps as performed in [2], for example. However, it was decided that this approach was not used in the SIG-SHA-1 implementation, because of the increase in area requirements and, especially, because of the reduced comparability to the combined ar-

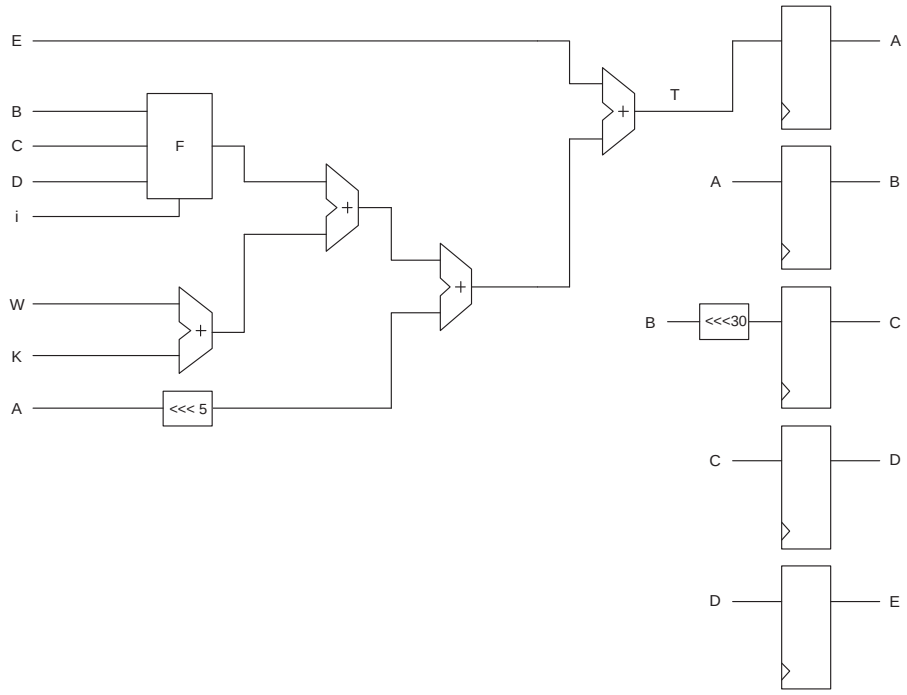

Figure 2: A block diagram of the step function

chitecture of [8] and SIG-MD5-I.

A block diagram of the step function block is presented in Figure 2 where `F` implements the functions of Equation (4) and `i` is the index of the function derived from the step t. The logic requirements of the step function are the following: four 32-bit adders, 5 32-bit registers and `F` requires four 32-bit bitwise logical operations selected by a multiplexer. Cyclical left shifts by constant values 5 and 30 do not require any logic, as they are performed simply by reorganizing the bit vector.

The constant block in Figure 1 contains the values of constant K_t. It may be implemented using dedicated memory blocks of the FPGA device, e.g. Block-RAMs in Xilinx' devices. However, in SIG-SHA-1 it was implemented on slices in order to guarantee straightforward comparison to SIG-MD5-I.

The message schedule block implements the SHA-1 message schedule described in Section 2. A 512-bit message block M_j is loaded into the block with `M_in`, `address` and `load` signals. The width of `M_in` can be chosen between 32 and 512 bits, and for the implementations presented in this report the width was chosen to be 128 bits. The message schedule block includes a 16x32-bit shift register and additional logic implementing Equation (2).

The five adders in Figure 1 are used for calculating the additions of Equa-

tion (7). The counter is a 7-bit counter, which counts the value of t from 0 to 79. The coder derives the index i of the functions of Equation (4) from t, i.e.

$$i = \begin{cases} 0 & \text{if } 0 \leq t \leq 19 \\ 1 & \text{if } 20 \leq t \leq 39 \\ 2 & \text{if } 40 \leq t \leq 59 \\ 3 & \text{if } 60 \leq t \leq 79 \end{cases} \quad (8)$$

The ready block determines when the calculation of the steps is finished, i.e. $t = 79 = 1001111_2$. The leftmost multiplexer in Figure 1 is used for initializing the hash value registers. The initial values are set when a derivation of a new hash value is started with `start_new`, i.e. when M_0 is processed. If $j \geq 1$ in M_j, the values from previous algorithm round are used, and the derivation is began with the `continue` signal. The other multiplexer is used for controlling the iterative loop. For the first step, $t = 0$, the initial values or the values from the previous algorithm round are taken (`start_new` or `continue`), otherwise values from previous iteration step are used.

When the `ready` signal is high, a processing of M_j is finished, and M_{j+1} can be loaded into the design. If all M_j have been processed, the hash $\mathcal{H}$ of the message $\mathcal{M}$ is ready in `hash`.

The above architecture was written in VHDL and it required 580 lines of code. Having the experience of implementing MD5, the design of SHA-1 was simple and straightforward.

4 Results

The architecture presented in Section 3 was implemented on a Xilinx Virtex-II XC2V2000-6 FPGA device. The logic synthesis was performed with Synplify 7.3.4 and implementation, including translation, mapping, place & route and timing, was performed with Xilinx ISE 6.2. Aldec Active-HDL 6.2 was used for project management and simulations.

Virtex-II XC2V2000-6 includes logic resources of 10,752 slices. A Virtex-II slice consists of two 4-to-1-bit Look-Up Tables (LUTs), two flip-flops and some additional logic [17].

Implementation results of the SIG-SHA-1 design on Virtex-II XC2V2000-6 are presented in Table 1, where the throughput and throughput per slice (TPS) values are calculated with the following equations [5]:

$$\text{throughput} = \frac{\text{block size} \times \text{clock frequency}}{\text{clock cycles per block}} \quad (9)$$

Table 1: Implementation results of SIG-SHA-1 on Virtex-II XC2V2000-6

Slices	1,275
Equivalent gate count	25,467
Max. clock frequency	117.5 MHz
Latency of a SHA-1 round	698 ns
Throughput	734 Mbps
TPS	0.576 Mbps / slice

and

$$\text{TPS} = \frac{\text{throughput}}{\text{slices}}. \tag{10}$$

where block size is 512 bits and clock cycles per block is 82.

Based on the values presented in Table 1 it is stated that SHA-1 can be efficiently implemented on FPGAs with minimal logic resources. The performance of SIG-SHA-1 is sufficient for most imaginable applications, but if the throughput falls short for certain applications, similar methods that were used in [9] for increasing the throughput of MD5 can be used also for SHA-1. That is, parallel SHA-1 blocks can be added so that several SHA-1 calculations can be processed simultaneously in parallel. Throughput can be also increased by unrolling several SHA-1 steps and then pipelining the design so that a different SHA-1 calculation can be simultaneously processed in every pipeline stage. These approaches increase the throughput of the design, but they do not decrease the delay of a single SHA-1 calculation. If the delay of a single calculation needs to be reduces, unrolling of several steps may be used.

5 Comparisons

In this section, the SIG-SHA-1 design presented in the previous sections is compared to other relevant implementations. First, SHA-1 and MD5 are compared in Section 5.1. The comparison is straightforward, because similar implementation techniques as well as target devices were used. Second, SIG-SHA-1 designs are compared to other published FPGA-based implementations in Section 5.2. First of all, it must be stated that SIG-SHA-1 was not designed to be the fastest or most compact SHA-1 implementation. It was implemented merely to compare SHA-1 and MD5 and to provide a rightful reference point for the design of a compact and combined MD5/SHA-1 architecture presented in [8].

5.1 Comparison of SHA-1 and MD5

When the SIG-MD5-I design presented in [9] was synthesized with Synplify 7.3.4 instead of Xilinx Synthesis Tool (XST) 6.2 used in the original paper, significant enhancement in performance was attained. These new results on Xilinx Virtex-II XC2V2000-6 are the following: SIG-MD5-I requires 1235 slices, operates at a clock frequency of 101.9 MHz, achieves a throughput of 791 Mbps, and one algorithm round requires 647 ns to be completed. TPS value of SIG-MD5-I is 0.640 Mbps / slice.

When the results of Table 1 are compared to the above MD5 results, it can be seen that almost similar amount of area is required from the target device. Although SHA-1 generates a 160-bit hash value $\mathcal{H}$ instead of 128 bits created with MD5, only small increase in required area is witness. This is mainly because of the simpler structure of the SHA-1 step function compared to MD5.

The simpler step function also results in a shorter critical path, thus allowing higher maximum clock frequency. Although the clock frequency is higher, the performance figures, i.e. throughput and TPS, are smaller compared to SIG-MD5-I. The reason for this is that SHA-1 requires computation of 80 steps instead of 64 computed in MD5. Thus, a slower overall performance is achieved.

As a conclusion for the comparison of SIG-SHA-1 and SIG-MD5-I, it is stated that both SHA-1 and MD5 can be implemented with almost similar logic requirements and there is no major difference in achieved performance of the algorithms. In addition, almost similar implementation techniques can be used. Even some of the VHDL code can be re-used in the design of the other algorithm.

SHA-1 and MD5 have a similar general structure and they share many common resources. Therefore, a design combining both into a single compact design exploiting similarities of the algorithms very efficiently was designed. It was concluded that the algorithms can be combined in a very compact fashion with only small reduction in performance. Details of this design are to be found in [8].

5.2 Comparison to Other Published FPGA-based Implementations

In this section, SIG-SHA-1 is compared to other published FPGA-based SHA-1 implementations. Designs included into the comparison are the following: Diez et al. [2], Dominikus [4], Kang et al. [10], Kitsos et al. [11], Selimis et al. [14], Sklavos et al. [15], and Zibin et al. [18]. A summary of these designs in presented in Table 2.

First, it must be mentioned that this comparison can be considered only as suggestive, because of the variety of different target devices used in different implementations. Thus, exact comparison of different implementation techniques

Table 2: Published FPGA-based SHA-1 Implementations

Design	Device	Slices	Clock (MHz)	Tput (Mbps)
SIG-SHA-1	Virtex-II 2V2000-6	1275	117.5	734
Diez [2]	Virtex-II 2V3000	1550	38.6	900
Dominikus [4][1]	Virtex-E V300E	1004?	42.9	119
Kang [10][2]	Altera EP20K1000E-3	10573 (LE)	18	114
Kitsos [11]	Virtex V300	2506	47	n.a.
Selimis [14]	Virtex V150	518	82	518
Sklavos [15][3]	Virtex-II 2V500	2245	55	1339
Zibin [18]	Altera EP1K100-1	1662 (LE)	43.1	269

cannot be made fairly only based on the performance figures presented in Table 2. In general, area requirements do not change dramatically when a Xilinx' device family is changed to another. However, severe enhancement in performance usually occurs when a newer device family or even only a faster device in the same family is used.

Comparison of logic requirements between Xilinx slices and Altera's logic elements (LE) is not straightforward. Rough estimates can be calculated by assuming that a slice equals two LEs. That is because a slice includes two LUTs, two registers and certain additional logic [17], whereas an LE consists of one LUT, one register and additional logic [1].

The fastest published SHA-1 implementation is, to the author's knowledge, design by Sklavos et al. published in [15]. Surprisingly, it implements, in addition of SHA-1, also the RIPEMD hash algorithm [3]. Efficient unrolling of steps was used in order to increase throughput.

The smallest published implementation is published by Selimis et al. in [14], and it required only 518 slices from the target device. It is not known, whether it requires additional memory in addition to the logic resources or not, but it most likely does.

More exotic hash implementations include designs by Dominikus and Kang et al. Dominikus presented a general hash processor which can be used for SHA-1, SHA-256, MD5 and RIPEMD hash algorithms [4]. It requires only a minimal amount of area and has a competent performance, at least if the rather old device family is taken into account. However, it is a general processor architecture, and therefore it naturally falls short in performance if compared to algorithm specific implementations.

Kang et al. designed a hash implementation for SHA-1, HAS-160 and MD5

[1] A hash processor. Also MD5, SHA-256 and RIPEMD included

[2] Also MD5 and HAS-160 included

[3] Includes also RIPEMD

algorithms in [10]. HAS-160 is a hash algorithm developed by Korea Telecommunications Technology Association [10], and it is not widely used. The structure of HAS-160 is similar to SHA-1 [10] and they were combined together in the design, but MD5 was included merely as a separate block. A more efficient combination of the MD5 and SHA-1 algorithms is presented in [8].

In [2], Diez et al. presented designs of MD5 and SHA-1, of which only the SHA-1 design is considered in this report. It used two step unrolling for achieving faster performance. Kitsos et al. presented in [11] a SHA-1 implementation which was used in an implementation of the Digital Signature Algorithm (DSA) [6], and therefore the SHA-1 design is only a small part of the paper and it is not considered in detail. Zibin and Ning presented an implementation of SHA-1 in [18]. The implementation was performed in a traditional fashion, and the structure of the architecture is quite similar to the architectures presented in this report and in [11], for example.

SIG-SHA-1 is among both fastest and smallest implementations. Thus, it is a very competent implementation of SHA-1. However, as it does not contain any novel methods to implement SHA-1, there is no need to publish it in any refereed journal or conference. In addition, all the studies of implementation techniques performed in [9] for MD5 can be generalized also for SHA-1.

6 Conclusions

An efficient design of the widely used hash algorithm, SHA-1, was presented. The design was a straightforward and easy-to-understand implementation of the SHA-1 specifications presented in [7]. The design is called as SIG-SHA-1, where SIG is an acronym for the Signal Processing Laboratory at Helsinki University of Technology, where the design work was performed.

SIG-SHA-1 was compared to SIG-MD5-I, which is an implementation of the other commonly used hash algorithm MD5, published in [9]. It was concluded that almost similar amount of area is required from the target device for both designs. SIG-SHA-1 operates at a higher clock frequency than SIG-MD5-I, but SIG-MD5-I achieves higher throughput values, because of the structures of the algorithms. Anyhow, both of the implementations achieve very high performance with minimal logic requirements.

SIG-SHA-1 is among the fastest and most compact published FPGA-based SHA-1 implementations. There are both faster and smaller implementations, but SIG-SHA-1 offers a good balance between performance and required area.

References

[1] Altera, Corp. *APEX 20K Programmable Logic Device Family*, March 2004. Version 5.1. URL: `http://www.altera.com/literature/ds/apex.pdf` (visited September 21, 2004).

[2] J.M. Diez, S. Bojanić, Lj. Stanimirović, C. Carreras, and O. Nieto-Taladriz. Hash Algorithms for Cryptographic Protocols: FPGA Implementations. *Proceedings of the 10th Telecommunications Forum TELFOR'2002, Belgrade, Yugoslavia*, November 26 – 28, 2002.

[3] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A Strengthened Version of RIPEMD. In *Fast Software Encryption*, pages 71 – 82, 1996.

[4] S. Dominikus. A Hardware Implementation of MD4-Family Hash Algorithms. *Proceedings of 9th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2002), Dubrovnik, Croatia*, Vol. 3:1143 – 1146, September 15 – 18, 2002.

[5] A.J. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 9:545 – 557, August 2001.

[6] Federal Information Processing Standards. Digital Signature Standard (DSS). *FIPS PUB 186-2*, January 27, 2000. URL: `http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf`, (visited September 21, 2004).

[7] Federal Information Processing Standards. Secure Hash Standard. *FIPS PUB 180-2*, August 1, 2002. With changes, February 25, 2004, URL: `http://www.csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`, (visited September 21, 2004).

[8] K. Järvinen, M. Tommiska, and J. Skyttä. Compact Combined MD5 and SHA-1 Hash Module. *Submitted to the 25th IEEE International Conference on Distributed Computing Systems, ICDCS 2005, Columbus, Ohio, USA*, June 6 – 9, 2005.

[9] K. Järvinen, M. Tommiska, and J. Skyttä. Hardware Implementation Analysis of the MD5 Hash Algorithm. *Accepted to the Thirty-eighth Annual Hawai'i International Conference on System Sciences, HICSS'38, Software Technology Track, Mobile Computing Architectures, Design and Implementation*, January 3 – 6, 2005.

[10] Y.K. Kang, D.W. Kim, T.W. Kwon, and J.R. Choi. An Efficient Implementation of Hash Function Processor for IPSEC. *Proceedings of the IEEE Asia-Pacific Conference on ASIC*, pages 93 – 96, August 6 – 8, 2002.

[11] P. Kitsos, N. Sklavos, and O. Koufopavlou. An efficient implementation of the digital signature algorithm. *Proceedings of 9th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2002), Dubrovnik, Croatia*, 3:1151 – 1154, September 15 – 18, 2002.

[12] R.L. Rivest. The MD5 Message-Digest Algorithm. *RFC 1321, MIT Laboratory for Computer Science and RSA Data Security, Inc.*, April 1992.

[13] B. Schneier. *Applied Cryptography*. John Wiley & Sons, 2nd edition, 1996.

[14] G. Selimis, N. Sklavos, and O. Koufopavlou. VLSI Implementation of the Keyed-Hash Message Authentication Code for the Wireless Application Protocol. *Proceedings of the 10th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2003), Sharjah, United Arab Emirates*, 1:24 – 27, December 14 – 17, 2003.

[15] N. Sklavos, G. Dimitroulakos, and O. Koufopavlou. An Ultra High Speed Architecture for VLSI Implementation of Hash Functions. *Proceedings of the 10th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2003), Sharjah, United Arab Emirates*, 3:990 – 993, December 14 – 17, 2003.

[16] T. Wollinger, J. Guajardo, and C. Paar. Security on FPGAs: State of the Art Implementations and Attacks. *ACM Transactions on Embedded Computing Systems, Special Issue on Security and Embedded Systems*, 3:534 – 574, 2004.

[17] Xilinx, Inc. *Virtex-II Platform FPGAs: Complete Data Sheet*, June 24, 2004. URL: `http://direct.xilinx.com/bvdocs/publications/ds031.pdf`, (visited: September 21, 2004).

[18] D. Zibin and Z. Ning. FPGA Implementation of SHA-1 Algorithm. *Proceedings of the 5th International Conference on ASIC*, 2:1321 – 1324, October 21 – 24, 2003.