The Journal of Supercomputing, 31, 227–248, 2005

Implementation of the SHA-2 Hash Family Standard Using FPGAs

N. SKLAVOS nsklavos@ee.upatras.gr
O. KOUFOPAVLOU
Electrical and Computer Engineering Department, University of Patras, Patras, Greece

Abstract. The continued growth of both wired and wireless communications has triggered the revolution for the generation of new cryptographic algorithms. SHA-2 hash family is a new standard in the widely used hash functions category. An architecture and the VLSI implementation of this standard are proposed in this work. The proposed architecture supports a multi-mode operation in the sense that it performs all the three hash functions (256, 384 and 512) of the SHA-2 standard. The proposed system is compared with the implementation of each hash function in a separate FPGA device. Comparing with previous designs, the introduced system can work in higher operation frequency and needs less silicon area resources. The achieved performance in the term of throughput of the proposed system/architecture is much higher (in a range from 277 to 417%) than the other hardware implementations. The introduced architecture also performs much better than the implementations of the existing standard SHA-1, and also offers a higher security level strength. The proposed system could be used for the implementation of integrity units, and in many other sensitive cryptographic applications, such as, digital signatures, message authentication codes and random number generators.

Keywords: hash function standard, security, cryptography, hardware implementation, SHA-2 standard, AES standard

1. Introduction

In the last years, communications growth has increased dramatically the amount of the transmitted data. In addition, to the raised quantity of information is the increased quality demand for the protection of the transmission channel with high level security strength [13]. In order, these special needs for security to be satisfied sufficiently, new cryptographic algorithms and security schemes have been developed. Lately, a new Advanced Encryption Standard (AES) [3] and a new family of secure hash functions SHA-2 [21] have been published.

Hash functions are a fundamental primitive category in modern cryptography, often informally called one-way hashes [2]. A hash function is a computationally efficient function, which maps binary strings of arbitrary length to binary strings of some fixed length, called hash-values. The main scope of the hash function is to ensure the data integrity in the transmission channel. They are widely spread and many wireless protocols, such as WAP [27] and Hiperlan [10], have specified security layers and cryptographic schemes based on them. Hash functions are also used for the implementation of digital signature algorithms [16, 17], keyed-hash message authentication codes [11] and in random number generator architectures [23].

The Secure Hash Algorithm-1 (SHA-1) [20], is the world's most popular hash function. Unfortunately, the security level of this standard is limited to a level comparable to an 80-bit block cipher. The announced new AES Standard (Rijndael) [1, 3], which is specified in 128-, 192-, and 256-bit keys, drove the demand for a new SHA algorithm offering security comparable to the AES key strengths. On August 26, 2002, NIST announced the Secure Hash Standard 2 [21], which introduces the specifications of three new Secure Hash Algorithms, SHA-2 (256, 384 and 512).

Today, the most complicated cryptographic systems have been implemented in software than in hardware. One major reason is the implementers increased knowledge in software programming, than in hardware design. Software tools are widely spread with low prices, while VLSI CAD commercial tools are only on interest of large companies and specified research groups. Individual users and class projects are restricted to software possibilities. The applications increasing demand for computation power, and the power reduction requirements for portable devices, force us to consider that general-purpose processors are no longer an efficient solution for mobile systems. So, new hardware approaches are needed in order to implement some computational heavy and power consuming functions in order to to meet the current network speed requirements. Such approaches are Application-Specific Integrated Circuits (ASIC) technology and Field Programmable Gate Arrays (FPGAs).

ASIC device is the solution that created better opportunities for implementing real-time and more sophisticated systems. ASICs devices guarantee better performance, with enough small dedicated size. The reliability reaches high limits and the turnaround time is fast. Between the software applications and the ASICs devices there is a middle ground. This area is covered by the FPGAs. These components provide reconfigurable logic and they are commercially available at low prices. These devices vary in capacity and performance. The main disadvantage of them is that they are not suitable for the implementation of large functions. Programmable logic has several advantages over custom-hardware. It is less time-consuming, for the development and the design phase, than the custom-hardware approach.

In our days, reconfigurable computing is a very attractive method for the hardware implementation of systems/algorithms [4, 7–9, 12, 22, 26]. The systems/algorithms are divided into a sequence of hardware implementable objects (Hardware Objects). These types of objects represent the serial behavior of the algorithm and can be executed sequentially. The use of the Hardware Objects offers to the designer/developer a logic on-demand-capability that basically relies on the reconfigurable applied technique. Reconfigurable systems can change their "true" hardware configuration and can support multi-operation modes.

In this paper, an architecture with multi-mode operation for the SHA-2 family standard hardware implementation is proposed. The introduced system supports alternative operation modes. Upon the user needs, it performs the three SHA-2 hash functions (256, 384 and 512). Comparisons of the proposed system, with implementations of each hash function of the SHA-2 standard in a separate hardware device (FPGA) [24] are presented. In this way, a fair and detailed evaluation of the proposed architecture is given.

The covered silicon area of the proposed architecture is almost the same with the covered silicon area of the SHA-2(512) separate implementation [24]. The performance of the proposed system is equal and similar to the performance of the separate implementations

of SHA-2(384) and SHA-2(512) respectively [24]. The performance of the separate implementation SHA-2(256) is slightly higher compared with the performance of the proposed system [24]. Comparing with previous implementations published in [6] the proposed system is 277 and 417% faster. The work of [14] achieves higher throughput compared with the proposed systems at about 3% and 36%, but with lower operation frequency at about 49% times.

The introduced architecture can support efficiently the security needs of all the AES (Rijndael) operation modes, in every type of application. The proposed implementation could substitute efficiently the existing MD5 and SHA-1 hash functions implementations, in every integrity unit [10, 27] and in all the types of the applied security schemes [11, 16, 17, 23]. It provides higher supported security level and better hardware performance. In addition, the proposed system performance is much better than the previous SHA-1 standard works, in both software assembly developments [5, 18] and hardware implementations [6, 23].

The paper is organized as follows: in Section 2 the new SHA-2 hash family standard is introduced. In the next Section the proposed system architecture for the SHA-2 family and the VLSI implementation are presented in detail. The hardware implementation synthesis results are illustrated in the Section 4 and comparisons with other related works are given. Finally, conclusions are discussed in the last Section.

2. Secure hash family standard 2 (SHA-2)

An n-bit hash is a map from arbitrary length messages to n-bit hash values [19]. An n-bit hash function is an n-bit hash, which is one-way and collision resistant. One-way is the function that for a given hash value, it should require work equivalent to 2^n hash computations to find any message that hashes that value [2]. The term collision resistance characterizes the functions that finding two messages, which hash the same value, should require work equivalent to $2^{n/2}$ hash computations. Of course the hash functions architectures are public and commonly known. In the hash computation process, there is no secrecy and no keys, public or private, are used at all. The security is based on the one-way operation of each hash function itself [25].

The SHA-2 standard [21] supersedes the existing SHA-1, FIPS 180-1 [20], adding three new hash functions, SHA-2(256), SHA-2(384), and SHA-2(512), for computing a condensed representation (message digest) of electronic data. The produced message digest ranges in length from 256- to 512-bits, depending on the selected hash function each time. These hash functions enable the determination of a message's integrity: any change to the message will, with a very high probability, results in a different produced message digest.

The three new hash functions, specified in this standard, are called secure because for each one of them, it is computationally infeasible: (1) to find a message that corresponds to a given message digest, or (2) to find two different messages that produce the same message digest. Each hash function operation can be divided in two stages: preprocessing and hash computation. Preprocessing involves padding the input message, parsing the padded data into a number of m-bit blocks, and setting the appropriate initial values, which are used in

Table 1. Secure hash functions specifications.

	Hash functions			
Terms	SHA-1	SHA-2 (256)	SHA-2 (384)	SHA-2 (512)
Input message size (bits)	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Padded data block (bits)	512	512	1024	1024
Word size (bits)	32	32	64	64
Transformation rounds	80	64	80	80
Message digest (bits)	160	256	384	512
Security	80	128	192	256

the hash computation. The hash computation uses the padded data along with functions, constants, and word logical and algebraic operations, to iteratively generate a series of hash values. The produced hash value after a specified number of transformation rounds is equal to the message digest.

The hash functions of the SHA-2 family differ most significantly in the number of security bits that are provided for the hashed input message. Security is directly related to the message digest length. In most of the cases, when a hash function is used in conjunction with another encryption algorithm, there are special demands, which require the use of a hash function with a certain number of security bits. For example, if a message is being signed with a digital signature algorithm that provides 192-bit security, then that signature algorithm requires the use of a secure hash algorithm that provides 192-bit security, SHA-2(384). In Table 1 the hash function specifications are given. With the rule security is defined a birthday attack on a message digest of size n, which finally produces a collision with a work factor of approximately $2^{n/2}$ [15].

Furthermore, in the Appendix the most important technical issues of the SHA-2 standard are presented. The Appendix is a brief presentation of the standard specifications [21] and it is given for better understanding. Due to the limited length of the paper, used constants, generation constants mechanisms, analytical arithmetic and logical functions and padding operations are not included. For more information about the SHA-2 hash family the readers are encouraged to study the standard specifications.

3. Proposed system architecture and VLSI implementation

3.1. Proposed system architecture

The proposed system architecture is illustrated in Figure 1. It performs all the three SHA-2 hash family standard different functions (256, 384 and 512), upon to the user needs.

During initialization phase, the user with the appropriate write commands selects the operation mode. The Control Unit coordinates all the system operations and processes. After the initialization phase, the control unit is totally responsible for the system operation. It defines the proper constants and operation word length, it manages the ROM blocks and it controls all the proper algebraic and digital logic functions for the operation of SHA-2 (256, 384 and 512) hash functions.

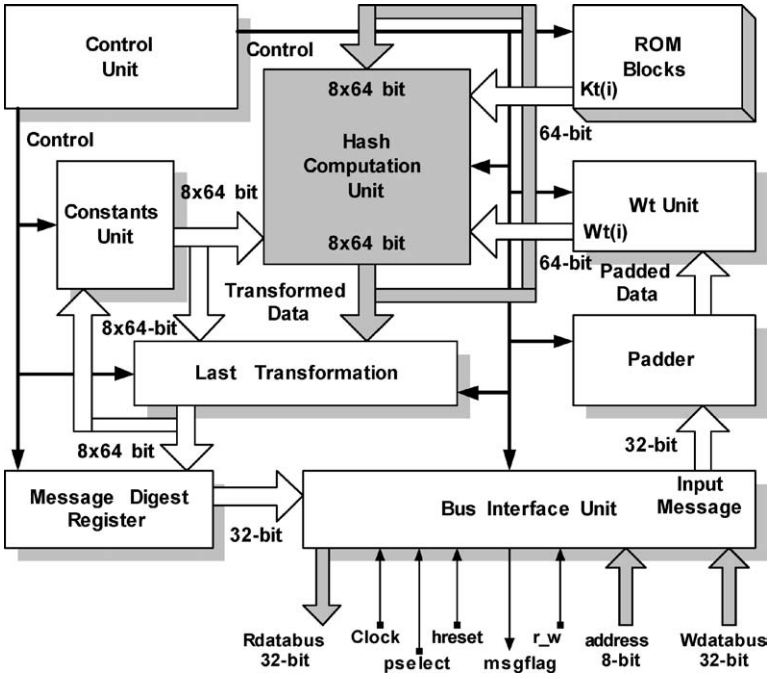


Figure 1. Proposed system architecture.

The Hash Computation Unit is the main datapath component of the system architecture. The specified number of the data transformation rounds, for each one of the SHA-2 hash family functions, is performed in this component with the support of a rolling loop (feedback). The Transformed Data are finally modified in the Last Transformation, which operates in cooperation with the Constants Unit. In this way, the message digest is produced and is stored into the Message Digest Register.

First, the Padder pads the input message and after that the hash computation begins. The Padded Data is a multiple of 512-bit block for the SHA-2(256) and a multiple of 1024-bit block for both the SHA-2(384) and SHA-2 (512). In every data transformation round, based on the padded data, in the *Wt* Unit a new data block, $Wt(i)$, is produced. In the ROM Blocks the specified constants set, $Kt(i)$, of the SHA-2 standard are stored, in order to support the Hash Computation Unit process. A Bus Interface Unit has also been integrated, in order for the proposed system to communicate efficiently with the external environment.

In the following, the system architecture's basic units are described.

3.2. Hash Computation Unit

The Hash Computation Unit architecture is shown in Figure 2. It accepts 8 basic data inputs ($A_{\text{in}}, \ldots, H_{\text{in}}$) and produces 8 basic outputs ($A_{\text{out}}, \ldots, H_{\text{out}}$). The constant input values

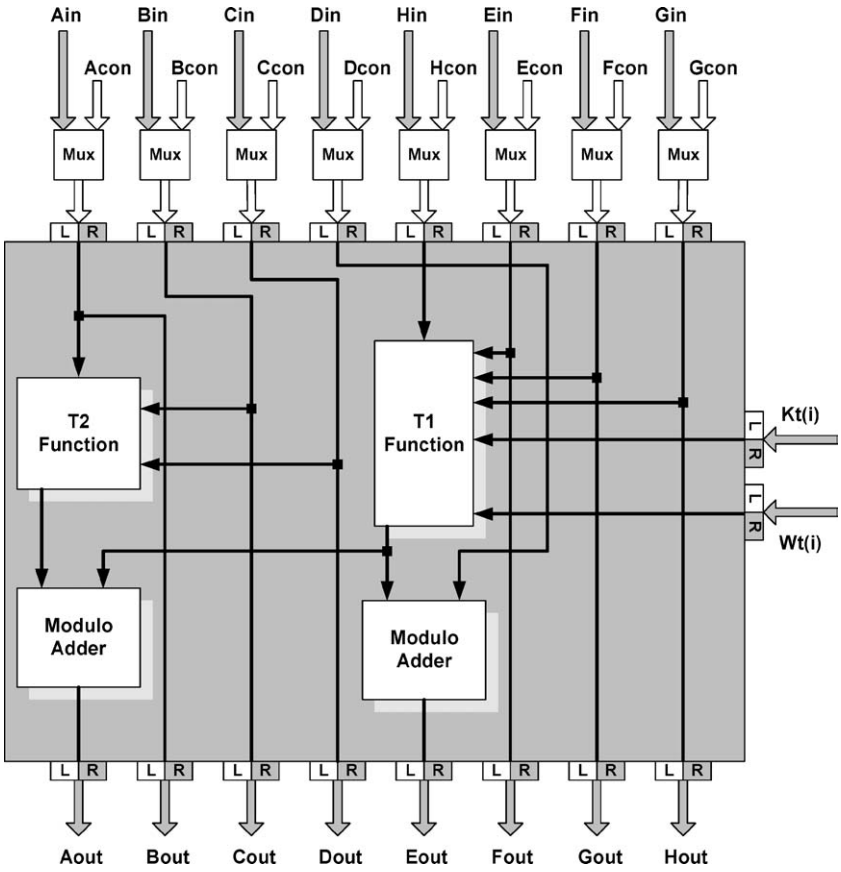


Figure 2. Hash Computation Unit architecture.

(A_{con}, ..., H_{con}), are loaded only during the initialization process. This unit has also two other inputs, $Kt(i)$ and $Wt(i)$. $Kt(i)$ data are provided from the ROM Blocks (Figure 1) and $Wt(i)$ data from the *Wt* Unit. The word length of all the inputs and outputs is equal to 64-bit. In order the proposed architecture to perform properly for all the hash functions of the SHA-2 family standard the word length is divided in two equal parts of 32-bit. The right (R) 32-bit part is used always, while the left (L) 32-bit is "idle" (stuck at hex "00000000") for the SHA-2(256) operation. This is due to the fact that the SHA-2(384) and SHA-2(512) operates with 64-bit word size while the SHA-2(256) operates with 32-bit words.

The data modifications in the Hash Computation Unit are mainly performed by the T1 and T2 functions and the Modulo Adders. Especially, in the Modulo Adder components, modulo addition 2^{32} is performed for the SHA-2(256), and modulo addition 2^{64} for the SHA-2(384) and SHA-2(512).

The proposed architectures for the T1 and T2 functions are illustrated in Figure 3. In these architectures, the $S(n)$ component indicates right rotation of the input data by n-bit.

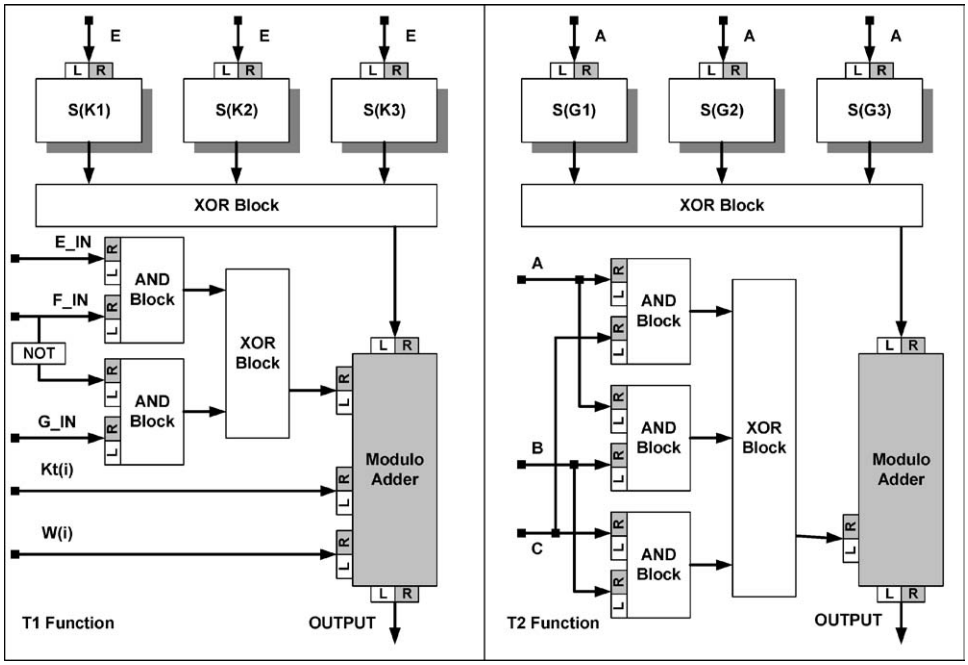


Figure 3. T1 and T2 function architectures.

The specified constants values (K1, K2, K3) and (G1, G2, G3) for T1 and T2 functions, are different for each one of the SHA-2 hash family functions. So, for the three different operation modes of the proposed system architecture, $S(n)$ component rotates the input data by different number of positions. The specified constants values for the three hash function are given in the following Table 2.

3.3. Padder and Wt unit

The input message is padded in the Padder, before the beginning of the hash computation. So, the Padder ensures that the input message length is a multiple of 512-bit block in the case of SHA-2(256), and multiple of 1024-bit block in the cases of SHA-2(384) and SHA-2(512). Padding is a simple operation [21] but it differs in the standard three hash functions.

Table 2. SHA-2 family constants.

SHA-2	K1, K2, K3	G1, G2, G3	J1, J2, J3	L1, L2, L3
(256)	2, 13, 22	6, 11, 25	10, 19, 17	3, 18, 7
(384)	28, 34, 39	14, 18, 41	6, 61, 19	7, 8, 1
(512)	28, 34, 39	14, 18, 41	6, 61, 19	7, 8, 1

This operation can be performed alternatively by software. The hardware implementation has not much area cost (7% of the total system) and ensures higher performance than software. In order to provide a full hardware and autonomous system implementation, without additional software packages requirements, padding is performed in hardware. Of course, in applications with hard limited silicon area, such as Personal Digital Assistants, the software solution is preferable.

The output data of Padder is the input of the *Wt* Unit (Figure 1). In the *Wt* Unit the Padded data are divided into N equal data blocks, $M^1, \ldots, M^N$, and are processed in order. The output data, are produced by using the following two equations:

$$W(i) = M^i, \quad 1 \leq i \leq 16, \tag{1}$$

$$W(i) = \sigma 1 W(i-2) + W(i-7) + \sigma 0 W(i-15) + W(i-16), \quad 16 < i, \tag{2}$$

where, i indicates the number of the transformation round. In every hash function of the SHA-2 family standard, a certain number of transformation rounds is specified, which is determined in the Table 1. The architecture of the *Wt* Unit is illustrated in Figure 4.

The Padded Data consist of 16 data blocks (*Mi*), which are stored, during initialization, in the 16 registers ($R\_1, \ldots, R\_16$). In every clock cycle the stored data into the register $R\_(i)$ are forwarded to the register $R\_(i-1)$. After initialization and during shifting, in every clock cycle the combinational logic circuits $\sigma 0$ and $\sigma 1$ (Figure 4) perform data transformations. As it is shown in Figure 4, functions $\sigma 0$ and $\sigma 1$ of the equation (2) are implemented by using the $S(n)$ and $R(n)$ components. The $S(n)$ indicates right rotation of the input data by

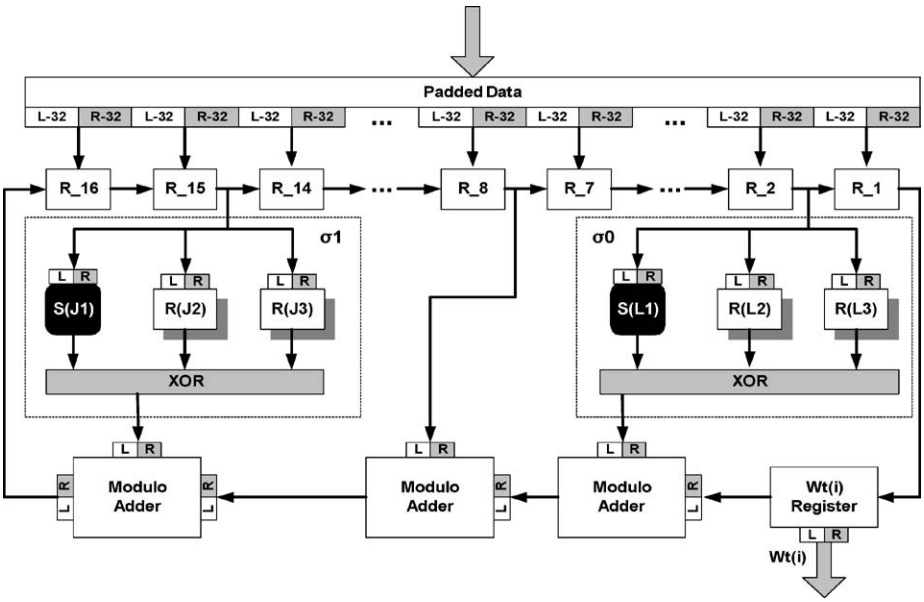


Figure 4. Wt Unit Architecture.

n-bit, while the $R(n)$ component indicates right shift of the input data by n-bit. Both $S(n)$ and $R(n)$ operate with different constants for each one of the hash functions of the SHA-2. The specified rotation and shift constants values are given in Table 2.

The Modulo Adder components perform modulo addition 2^{32} or 2^{64}, in the same way as described in Section 3.2. The produced $Wt(i)$ for each data modification round is stored in the $Wt(i)$ Register and then is loaded to the $Wt(i)$ output.

3.4. ROM Blocks

In the ROM Blocks (Figure 5) are stored the specified constants set support $Kt(i)$, which is different for each one hash function of the SHA-2 family standard. The SHA-2(384) and SHA-2(512) use a sequence of 80 constants equal to 64-bit. These constants represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers (for more details see [21]). The SHA-2(256) uses a sequence of 64 constants equal to 32-bit. These constants represent the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers.

From the 80 defined constants in both SHA-2(384) and SHA-2(512), only the first 64 constants are used in SHA-2(256). Especially, only the left 32-bit parts of each one of these 64 constants are valid data. In order to implement the specified $Kt(i)$ constants set, only two ROM arrays of 80×64-bit are used in total. In this way the operation of each hash function is achieved with minimized allocated area resources.

In the cases of SHA-2(384) and SHA-2(512) operation modes, the ROM_Block_I 32-bit output, is the left part of the $Kt(i)$ output while the right part is the ROM_Block_II 32-bit

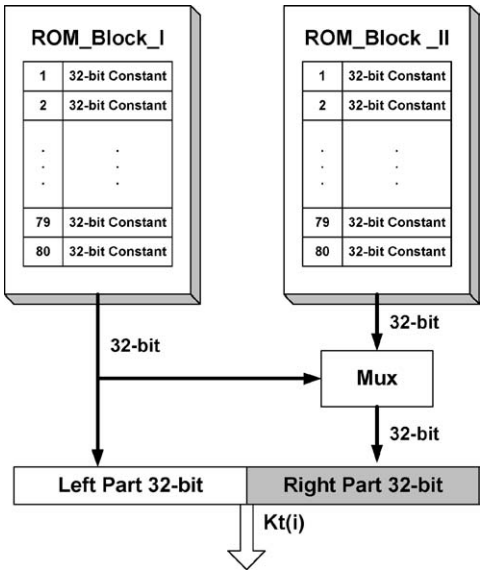


Figure 5. ROM Blocks.

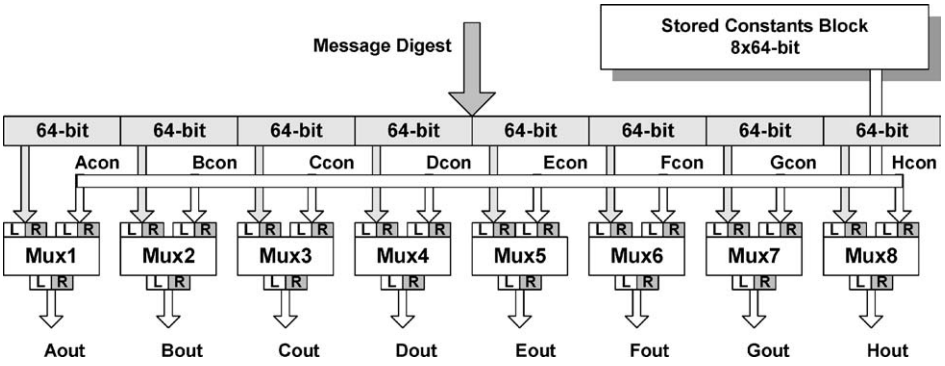


Figure 6. Constants unit architecture.

output. When the proposed system architecture operates in the SHA-2(256) mode, the 32-bit left part of the $Kt(i)$ is "idle" (stuck at hex"00000000"), while the right 32-bit part is the output of the ROM_Block_I.

3.5. Constants unit

The SHA-2 hash family standard specifies 8 × 32-bit constants for the SHA-2(256), and 8 × 64-bit for both the SHA-2(384) and SHA-2(512). These constants are used in the initialization process of the Hash Computation Unit and in the Last Transformation. The proposed architecture for the Constants Unit is illustrated in the Figure 6.

For the SHA-2(384) and SHA-2(512) the specified constants are equal to the first 64 bits of the fractional parts of the square roots of the first 8 prime numbers. For the SHA-2(256) these constants are obtained by taking the first 32 bits of the fractional parts of the square roots of the first 8 prime numbers. The specified constants in SHA-2(256) are the left 32-bit parts of the specified constants of SHA-2(384) and SHA-2(512). For this reason only 8 × 64-bit constants have to been stored in the Constants Unit. The Stored Constants Block of the above architecture has been implemented by using the Look Up Tables (LUTs) technique.

The SHA-2 family hash functions operate on padded data, which are multiples of 512-bit block for SHA-2(256) or 1024-bit block for SHA-2(384 and 512). If a message has more data blocks than one, the values of the used constants have to be refreshed. When the (n) padded data block is processed, the produced message digest is equal to the concatenation of the used constants for the next padded data block $(n + 1)$. These new values are forced to the outputs $(A_{\text{out}}, \ldots, H_{\text{out}})$, through the multiplexers set (Mux1 to Mux8). When a new block, not related to the previous processed padded data, has to be transformed, the used constants are initialized in the predefined square root values $(A_{\text{con}}, \ldots, H_{\text{con}})$. When the proposed system performs as SHA-2(256) the right part (R) of the multiplexers input/output is used only, while in the cases of SHA-2(384) and SHA-2(512) operation modes both right (R) and left part (L) values of these inputs/outputs are used.

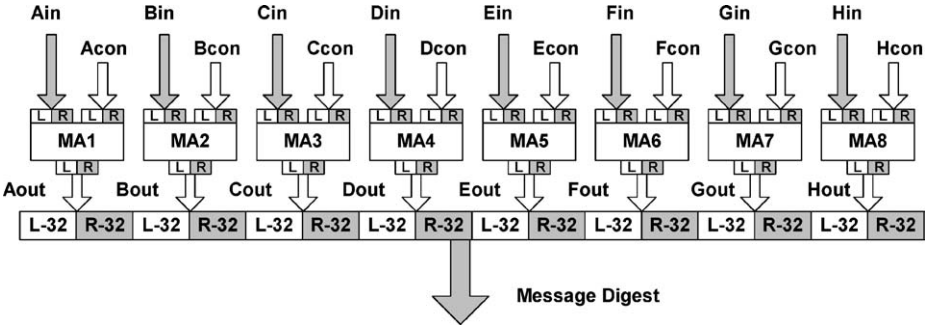


Figure 7. Last Transformation architecture.

3.6. Last Transformation

The Last Transformation proposed architecture is illustrated in Figure 7. The main functions of this unit are the modulo additions between the 8 data inputs ($A_{\text{in}}, \ldots, H_{\text{in}}$) and the 8 input constants values ($A_{\text{con}}, \ldots, H_{\text{con}}$).

The MA component operates modulo addition 2^{32}, for the SHA-2(256) and modulo addition 2^{64}, for the SHA-2(384) and SHA-2(512). The word length of the inputs and output of the MA component is 64-bit. The right 32-bit part (R) is used always while the left 32-bit (L) is "idle" (stuck at hex "00000000") in the case of SHA-2(256) operation. The produced message digest for the SHA-2(256) is 256-bit and consists of the right 32-bit parts of all the MA components outputs. The message digest for the SHA-2(512) is 512-bit and is produced by the concatenation of the 64-bit data output of every modulo adders. The 384-bit message digest for the SHA-2(384) is obtained by concatenating the 64-bit data output of the 6 left MA components ($A_{\text{out}} \| B_{\text{out}} \| C_{\text{out}} \| D_{\text{out}} \| E_{\text{out}} \| F_{\text{out}}$).

4. VLSI implementation synthesis results

The proposed system architecture (Figure 1) was captured by using vhdl, with structural description logic. The code was synthesized, placed, and routed using an FPGA device of XILINX (Virtex v200pq240, speed grade:-6) [28]. The system then was simulated again, and verified considering real time operating conditions, by using the official test vectors, provided by the standard [21]. The tools that were used for the above procedures are: modelsim se/ee plus 5.4e for simulation, Leonardo spectrum 2002a.49 for synthesis and xilinx Foundation 3.1i for placing and routing. The synthesis results of the proposed system are shown in Table 3. In order to have a fair and detailed evaluation, the proposed system implementation is compared with each one the hash functions individual implementation, in different hardware devices (FPGAs) [24]. The synthesis results for these implementations are also shown in the Table 3.

The proposed system has almost the same allocated resources with the separate SHA-2_512 implementation. Of course the SHA-2_256 and SHA-2_384 implementations have

Table 3. Implementation synthesis results.

Hardware device → Implementation ↓	Covered area			Used ROM (bit)	F (MHz)
	CLBs	FGs	DFFs		
SHA-2_256	1060	2120	1651	64 × 32	83
SHA-2_384	1966	3932	3689	80 × 64	74
SHA-2_512	2237	4074	3739	80 × 64	75
Proposed system SHA-2 (256, 384, 512)	2384	4103	3912	80 × 64	74

DFFs: D Flip-Flops, CLBs: Configurable Logic Blocks, FGs: Function Generators.

less covered area at about 55 and 17% respectively. The operating frequency of the proposed system is the same as the frequency of SHA-2_384 implementation. It is also almost the same as the frequency of SHA-2_512 implementations and about 11% less than the 83 MHz of the SHA-2_256 implementation. The estimated power consumption of the proposed system is 49 mW. The separate implementations, SHA-2_256, SHA-2_384 and SHA-2_512, have estimated power dissipation 41, 45, and 46 mW, respectively. These power estimations have been achieved with the power consumption tool provided in [29].

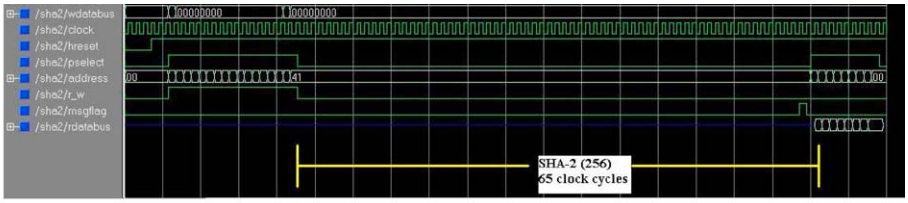
In order for the proposed system to perform efficiently for the specified 64 data transformation rounds, the system in the SHA-2(256) operation mode requires 65 clock cycles. For the SHA-2(384) and SHA-2(512) operations the standard defines 80 data transformation rounds. For these operations, in order for a message digest to be produced, the system requires 81 clock cycles. In Figure 8, for each one of the three operation modes, simulation results for the external bus generation waveforms are presented.

As mentioned above, during initialization the user selects which operation mode will be performed. After, the input message is loaded from the bus interface unit the msgflag indicates when a new message digest is generated. When this signal is turned ON the message digest is read from the user.

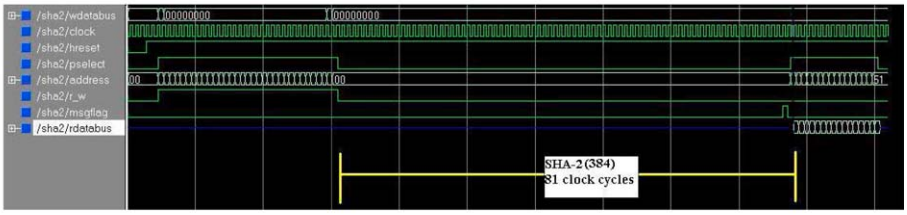
Furthermore, the proposed system is compared, in the terms of the achieved throughput and the area-delay product, with the separate implementations of the SHA-2 family hash functions. These comparisons results are illustrated in Figures 9 and 10 respectively. In Figure 9, the achieved throughput values of the conventional architectures of [24] (SHA-2_256, SHA-2_384, SHA-2_512) and the proposed system are compared. The throughput value of the proposed system, when it operates in SHA-2(256) mode, is 13% less than the throughput of the SHA-2_256 implementation. Also, it is equal and slight lower with the achieved throughputs of the separate implementations, in the cases of SHA-2(384) and SHA-2(512) operation modes.

The area-delay product of the proposed system is worst than the SHA-2_256 and SHA-2_384 area-delay products and almost the same with the SHA-2_512 case.

The main advantages of the proposed system, proved by the Table 3 synthesis results and Figures 9 and 10 comparisons, are that the introduced system allocates almost the same area resources of only the separate SHA-2_512 implementation and performs efficiently for all the three hash functions of the SHA-2 family standard. The proposed system throughput is worst



(A) SHA-2 (256)



(B) SHA-2 (384)

(C) SHA-2 (512)

Figure 8. External bus waveforms.

only in SHA-2(256) operation mode, compared with the respective separate implementation SHA-2_256.

In addition, comparisons of the proposed system implementation with other previous works are illustrated in the Table 4. Since the SHA-2 hash family is a new standard only few related works [6, 14] have been published until now in the technical literature. In Table 4 comparisons with hash function standard (SHA-1) implementations [5, 6, 18, 23] are also given.

Although, in [6] only the SHA-2(256) has been implemented, whilst the proposed system supports the operation of all the hash functions of the SHA-2 family standard, the proposed system achieves better throughput values comparing with both the FPGA and the ASIC implementations of [6], at about 277% and 417% respectively.

The architecture in [6] is similar to a typical digital processor architecture, with shared processing resources and buses. The target of [6] was not performance or area but support of all MD4 hash functions, including SHA-1 and SHA-2(256). This design requires 392 clock cycles to perform the 64 rounds of SHA-2(256). The clock frequency for the FPGA

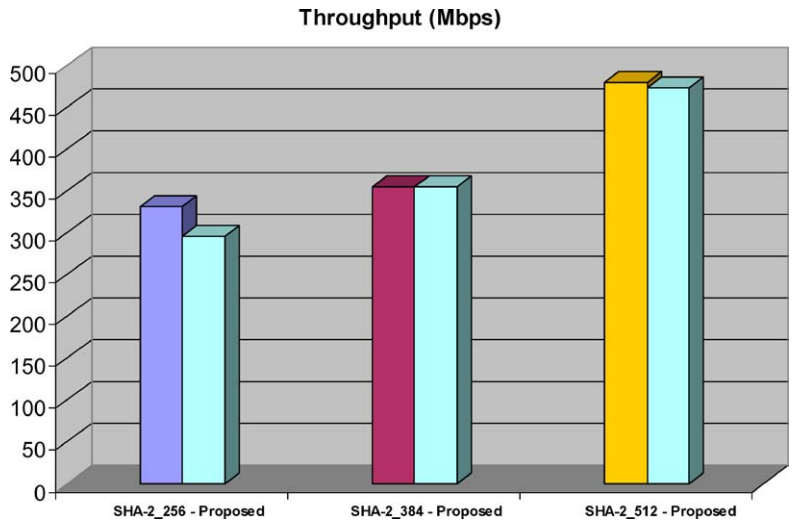


Figure 9. Implementations throughput comparison.

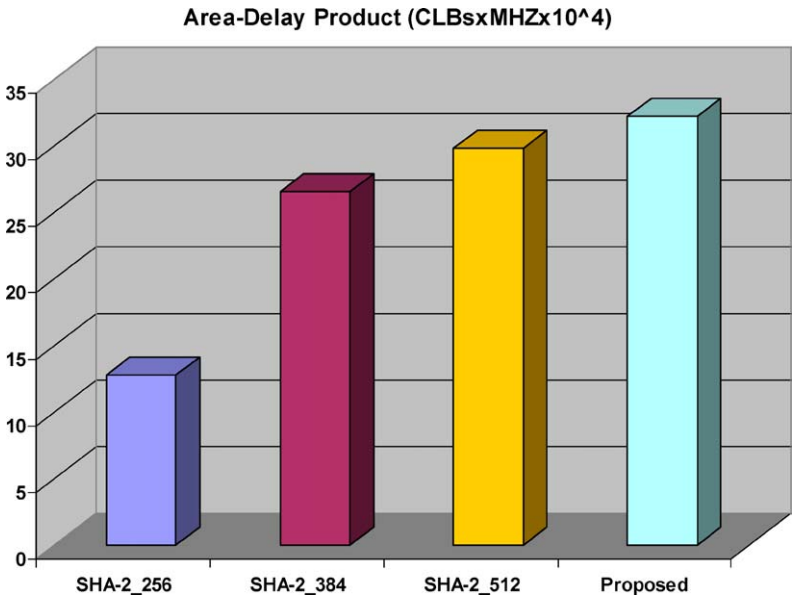


Figure 10. Area-delay product comparison.

Table 4. Implementation comparison results.

Implementations	Covered area	Frequency (MHz)	Data rate (Mbps)
SHA-1 [6]	1004 CLBs 10900 Gates	42.9 FPGA 59 ASIC	119; 86
SHA-1 [23]	2606 CLBs	37	233
SHA-1 [5]	Assembly	90 Soft.	40
SHA-1 [18]	–	N/A Soft. 133 Soft.	4.23; 41.51
SHA-2 (256) [6]	1004 CLBs 10900 Gates	42.9 FPGA 59 ASIC	77; 56
SHA-2 [14] (384, 512)	2914	38 FPGA	479
Proposed system SHA-2 (256, 384, 512)	2384 CLBS	74 FPGA	291, (256) 350, (384) 467, (512)

and the ASIC implementations are 59 and 42,9 MHz respectively. The proposed FPGA implementation is based on the full rolling technique, supports all the three hash functions of SHA-2(256, 384, 512) standards. The proposed system requires only 65 clock cycles in the SHA-2(256) operation mode. In the cases of SHA-2(384) and SHA-2(512) operation modes, 81 clock cycles are required. The clock frequency for all the operation modes of the proposed architecture is equal to 74 MHz. So, its performance (throughput) is 3 and 4 times faster than the implementations in [6]. The shared arithmetic units that are used in the architecture of [6], supported by data and address buses, are a design technique with not very good performance, compared with the proposed system. The better performance of the proposed system is due to the fact that the architecture of [6] requires too many clock cycles (392), compared with the 65 and 81 clock cycles that the proposed system needs. Although, in [6] components can be added and removed from the system easily (scalability) and the system performance can be increased by using more arithmetic units (exploiting parallelism).

The introduced system in [14] utilizes a shift register design approach and look-up tables. It supports only the two hash functions (384 and 512) of SHA-2 standard, while the proposed system operates efficiently for all modes. The achieved throughput of [14] is 479 Mbps and it is higher than the 350 and 467 MHz of the proposed system for SHA-2(384) and SHA-2(512) operation modes respectively. The proposed system requires about 18% less silicon area resources than the design in [14]. Finally, the proposed system architecture frequency is 1.9 times higher than [14]. But, the frequency of the design in [14] can be improved with some optimizations applied in the critical path.

In addition, the proposed system is proved to be better compared with the previous SHA-1 standard software [5, 18] and hardware implementations [6, 23]. But, we cannot go on a detailed "fair" comparison with the previous standard, since these two standards (SHA-1 and SHA-2) have major differences in their specifications.

Using the Performance/Area (Mbps/CLBs) ratio the proposed system is proved better compared with [6] at about 50%. The value of this ratio (0.16) for the system in [6] is almost the same with the one of the proposed system (0.15) for SHA-2(384) operation mode. Finally using this ratio, the proposed system is superior to [14] for SHA-2(512) operation, at about 20%.

5. Conclusions

In this paper, an architecture with multi-mode operation and the VLSI implementation of the SHA-2 hash family is proposed. The system can support efficiently the security needs of all AES (Rijndael) operation modes, with higher offered security level compared with the previous existing standard, SHA-1. Furthermore, this proposed system could substitute the implementations of the existing SHA-1 standard, in all types of applications, such as digital signatures, message authentication codes and random number generators, with better achieved performance and higher supported security level. The introduced system performs efficiently for the three SHA-2 standard functions (256, 384 and 512). The allocated resources of the proposed system are almost the same with the covered area of the separate implementation SHA-2_512. The achieved performance is almost equal to the separate implementations performance. The proposed system covers less area resources compared with previous published implementations [14], and achieves higher operation frequency compared with other related works [6, 14]. In some cases, it also achieves higher performance at about 277 and 417% than other hardware integrations [6]. Finally, the system performs much better compared with the implementations of the hash family standard SHA-1.

Appendix: SHA-2 hash family

1. SHA-256

SHA-256 may be used to hash a message, M, having a length of l bit, where $0 \leq l \leq 2^{64}$. The algorithm uses (1) a message schedule of sixty-four 32-bit words, (2) eight working variables of 32 bits each, and (3) a hash value of eight 32-bit words. The final result of SHA-256 is a 256-bit message digest.

The words of the message schedule are labeled $W0$, $W1$, ..., $W63$. The eight working variables are labeled a, b, c, d, e, f, g, and h. The words of the hash value are labeled $H0^{(I)}$, $H_1^{(I)}$, ..., $H_7^{(I)}$, which will hold the initial hash value, $H^{(0)}$, replaced by each successive intermediate hash value (after each message block is processed), $H^{(i)}$, and ending with the final hash value, $H^{(N)}$. SHA-256 also uses two temporary words, $T1$ and $T2$.

1.1. SHA-256 preprocessing

1. Padding of message M.
2. Parse the padded message into $N512$-bit message blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$
3. Set the initial hash value, $H^{(0)}$.

1.2. SHA-256 hash computation

The SHA-256 hash computation uses functions and constants. Addition (+) is performed modulo 2^{32}. After preprocessing is completed, each message block, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$, is processed in order, using the following steps:

For $i = 1$ to N:
{

1. Prepare the message schedule, $\{Wt\}$:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} 16 \le t \le 63 \end{cases}$$

2. Initialize the eight working variables, a, b, c, d, e, f, g, and h, with the $(i-1)$th hash value:

$$a = H_0^{(i)} b = H_1^{(i)} c = H_2^{(i)} d = H_3^{(i)} e = H_4^{(i)} f = H_5^{(i)} g = H_6^{(i)} h = H_7^{(i)}$$

3. For $t = 0$ to 63:
{

$$T_1 = h + \Sigma_1^{\{256\}}(e) + \mathrm{Ch}(e, f, g) + K_t^{\{256\}} + W_t$$
$$T_2 = \Sigma_0^{\{256\}}(a) + \mathrm{Maj}(a, b, c)$$
$$h = g, g = f, f = e, e = d + T_1, d = c, c = b, b = a, a = T_1 + T_2$$

}

4. Compute the ith intermediate hash value $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)} H_1^{(i)} = b + H_1^{(i-1)} H_2^{(i)} = c + H_2^{(i-1)} H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)} H_5^{(i)} = f + H_5^{(i-1)} H_6^{(i)} = g + H_6^{(i-1)} H_7^{\{i\}} = h + H_7^{(i-1)}$$

}

After repeating steps one through four a total of N times (i.e., after processing $M^{(N)}$), the resulting 256-bit message digest of the message, M, is

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$$

2. SHA-384

SHA-384 may be used to hash a message, M, having a length of l bit, where $0 \le l < 2^{128}$. The algorithm uses (1) a message schedule of eighty 64-bit words, (2) eight working variables of 64 bits each, and (3) a hash value of eight 64-bit words. The final result of SHA-384 is a 384-bit message digest. The words of the message schedule are labeled $W_0, W_1, \ldots, W_{79}$. The eight working variables are labeled a, b, c, d, e, f, g, and h. The words of the hash value are labeled $H_0^{(i)}, H_1^{(i)}, \ldots, H_5^{(i)}$, which will hold the initial hash

value, $H^{(0)}$, replaced by each successive intermediate hash value (after each message block is processed), $H^{(i)}$, and ending with the final hash value, $H^{(N)}$. SHA-384 also uses two temporary words, $T1$ and $T2$.

2.1. SHA-384 preprocessing

1. Pad the message, M according to standard's procedure but in different process than SHA-256.
2. Parse the padded message into N 1024-bit message blocks, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$
3. Set the initial hash value, $H^{(0)}$

2.2. SHA-384 hash computation

The SHA-384 hash computation uses functions and constants. Addition (+) is performed modulo 2^{64}. After preprocessing is completed, each message block, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$, is processed in order, using the following steps:

For $i = 1$ to n:
{

1. Prepare the message schedule, $\{Wt\}$:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 79 \end{cases}$$

2. Initialize the eight working variables, a, b, c, d, e, f, g, and h, with the $(i-1)$th hash value:

$$a = H_0^{(i)} b = H_1^{(i)} c = H_2^{(i)} d = H_3^{(i)} e = H_4^{(i)} f = H_5^{(i)} g = H_6^{(i)} h = H_7^{(i)}$$

3. For $t = 0$ to 79:
{

$$T_1 = h + \Sigma_1^{\{512\}}(e) + \mathrm{Ch}(e, f, g) + K_t^{\{256\}} + W_t$$
$$T_2 = \Sigma_0^{\{512\}}(\alpha) + \mathrm{Maj}(a, b, c)$$
$$h = g, g = f, f = e, e = d + T_1, d = c, c = b, b = a, a = T_1 + T_2$$

}
4. Compute the ith intermediate hash value $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)} H_1^{(i)} = b + H_1^{(i-1)} H_2^{(i)} = c + H_2^{(i-1)} H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)} H_5^{(i)} = f + H_5^{(i-1)} H_6^{(i)} = g + H_6^{(i-1)} H_7^{\{i\}} = h + H_7^{(i-1)}$$

After repeating steps one through four a total of Ntimes (i.e., after processing $M^{(N)}$), the 384-bit message digest is obtained by truncating the final hash value, $H^{(N)}$, to its left-most 384 bits:

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \|$$

3. SHA-512

SHA-512 may be used to hash a message, M, having a length of l bit, where $0 \leq l < 2^{128}$. The algorithm uses (1) a message schedule of eighty 64-bit words, (2) eight working variables of 64 bits each, and (3) a hash value of eight 64-bit words. The final result of SHA-512 is a 512-bit message digest. The words of the message schedule are labeled $W_0, W_1, \ldots, W_{79}$. The eight working variables are labeled a, b, c, d, e, f, g, and h. The words of the hash value are labeled $H_0^{(i)}, H_1^{(i)}, \ldots, H_7^{(i)}$, which will hold the initial hash value, $H^{(0)}$, replaced by each successive intermediate hash value (after each message block is processed), $H^{(i)}$, and ending with the final hash value, $H^{(N)}$. SHA-512 also uses two temporary words, $T1$ and $T2$.

3.1. SHA-512 preprocessing

1. Pad the message, M according to standard's procedure but in different process than SHA-256.
2. Parse the padded message into N 1024-bit message blocks, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$
3. Set the initial hash value, $H^{(0)}$

3.2. SHA-512 hash computation

The SHA-512 hash computation uses functions and constants. Addition (+) is performed modulo 2^{64}. After preprocessing is completed, each message block, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$, is processed in order, using the following steps:

For $i = 1$ *TO* n:

{

2. Prepare the message schedule, $\{Wt\}$:

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \leq t \leq 79 \end{cases}$$

2. Initialize the eight working variables, a, b, c, d, e, f, g, and h, with the (i-1)th hash value:

$$a = H_0^{(i)} b = H_1^{(i)} c = H_2^{(i)} d = H_3^{(i)} e = H_4^{(i)} f = H_5^{(i)} g = H_6^{(i)} h = H_7^{(i)}$$

4. For $t = 0$ to 79:
{

$$T_1 = h + \Sigma_1^{\{512\}}(e) + \mathrm{Ch}(e, f, g) + K_t^{\{256\}} + W_t$$
$$T_2 = \Sigma_0^{\{512\}}(a) + \mathrm{Maj}(a, b, c)$$
$$h = g, g = f, f = e, e = d + T_1, d = c, c = b, b = a, a = T_1 + T_2$$

}
4. Compute the ith intermediate hash value $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)} H_1^{(i)} = b + H_1^{(i-1)} H_2^{(i)} = c + H_2^{(i-1)} H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)} H_5^{(i)} = f + H_5^{(i-1)} H_6^{(i)} = g + H_6^{(i-1)} H_7^{\{i\}} = h + H_7^{(i-1)}$$

After repeating steps one through four a total of Ntimes (i.e., after processing $M^{(N)}$), the resulting 512-bit message digest of the message, M, is

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$$

4. SHA-2 used functions

This section defines the functions that are used by each of the algorithms. SHA-256, SHA-384, and SHA-512 algorithms all use different functions which are described in the next paragraphs.

4.1. SHA-256

SHA-256 uses six logical functions, where each function operates on 32-bit words, which are represented as X, Y, and Z. The result of each function is a new 32-bit word.

$$\mathrm{Ch}(x, y, z) = (\mathrm{X\ AND\ Y})\ \mathrm{XOR}\ ((\mathrm{NOT\ X})\ \mathrm{AND\ Z})$$
$$\mathrm{Maj}(x, y, z) = (\mathrm{X\ AND\ Y})\ \mathrm{XOR}\ (\mathrm{X\ AND\ Z})\ \mathrm{XOR}\ (\mathrm{Z\ AND\ Y})$$
$$\Sigma_0^{\{256\}} = \{\mathrm{RotationRight}^2(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{13}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{22}(\mathrm{X})\}$$
$$\Sigma_1^{\{256\}} = \{\mathrm{RotationRight}^6(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{11}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{25}(\mathrm{X})\}$$
$$\sigma_0^{\{256\}} = \{\mathrm{RotationRight}^7(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{18}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{ShiftRight}^3(\mathrm{X})\}$$
$$\sigma_1^{\{256\}} = \{\mathrm{RotationRight}^{17}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{19}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{ShiftRight}^{10}(\mathrm{X})\}$$

4.2. SHA-384

SHA-384 uses six logical functions, where each function operates on 64-bit words, which are represented as X, Y, and Z. The result of each function is a new 64-bit word.

$$\begin{aligned}
\mathrm{Ch}(x, y, z) &= (\mathrm{X\ AND\ Y})\ \mathrm{XOR}\ ((\mathrm{NOT\ X})\ \mathrm{AND\ Z}) \\
\mathrm{Maj}(x, y, z) &= (\mathrm{X\ AND\ Y})\ \mathrm{XOR}\ (\mathrm{X\ AND\ Z})\ \mathrm{XOR}\ (\mathrm{Y\ AND\ Z}) \\
\Sigma_0^{\{512\}} &= \{\mathrm{RotationRight}^{28}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{34}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{39}(\mathrm{X})\} \\
\Sigma_1^{\{512\}} &= \{\mathrm{RotationRight}^{14}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{18}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{41}(\mathrm{X})\} \\
\sigma_0^{\{512\}} &= \{\mathrm{RotationRight}^{1}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{8}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{ShiftRight}^{7}(\mathrm{X})\} \\
\sigma_1^{\{512\}} &= \{\mathrm{RotationRight}^{19}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{61}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{ShiftRight}^{6}(\mathrm{X})\}
\end{aligned}$$

4.3. SHA-512

SHA-512 uses six logical functions, where each function operates on 64-bit words, which are represented as X, Y, and Z. The result of each function is a new 64-bit word.

$$\begin{aligned}
\mathrm{Ch}(x, y, z) &= (\mathrm{X\ AND\ Y})\ \mathrm{XOR}\ ((\mathrm{NOT\ X})\ \mathrm{AND\ Z}) \\
\mathrm{Maj}(x, y, z) &= (\mathrm{X\ AND\ Y})\ \mathrm{XOR}\ (\mathrm{X\ AND\ Z})\ \mathrm{XOR}\ (\mathrm{Y\ AND\ Z}) \\
\Sigma_0^{\{512\}} &= \{\mathrm{RotationRight}^{28}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{34}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{39}(\mathrm{X})\} \\
\Sigma_1^{\{512\}} &= \{\mathrm{RotationRight}^{14}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{18}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{41}(\mathrm{X})\} \\
\sigma_0^{\{512\}} &= \{\mathrm{RotationRight}^{1}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{8}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{ShiftRight}^{7}(\mathrm{X})\} \\
\sigma_1^{\{512\}} &= \{\mathrm{RotationRight}^{19}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{RotationRight}^{61}(\mathrm{X})\}\ \mathrm{XOR}\ \{\mathrm{ShiftRight}^{6}(\mathrm{X})\}
\end{aligned}$$

References

1. Advanced Encryption Standard, http://csrc.nist.gov, 2002.
2. S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Cryptographic hash functions: A survey. Technical Report 95-09, Department of Computer Science, University of Wollongong, July 1995.
3. J. Daemen and V. Rijmen. AES Proposal: Rijndael. http://www.esat.kuleuven.ac.be/~rijmen/rijndael, 2002.
4. O. Diessel, H. ElGindy, M. Middendorf, H. Schmeck, and B. Schmidt. Dynamic scheduling of tasks on partially reconfigurable FPGAs. *IEE Proc., Comput. Digit. Tech.*, 147(3):181–188, 2000.
5. H. Dobbertin, H., Bosselaers, A., and B. Preneel. RIPEMD-160, a strengthened version of RIPEMD. In *Proc., Fast Software Encryption*, pp. 71–82, Springer-Verlag, 1996.
6. S. Dominikus. A hardware implementation of MD4-family hash algorithms. In *IEEE Proc., International Conference on Electronics Circuits and Systems (ICECS)*, vol. III, pp. 1143–1146, 2002.
7. M. Eisenring and M. Platzner. Synthesis of interfaces and communication in reconfigurable embedded systems. *IEE Proc., Comput. Digit. Tech.*, 147(3):159–165, 2000.
8. J. Goodman and A. P. Chandrakasan. An energy-efficient reconfigurable public-key cryptography processor. *IEEE Journal of Solid-State Circuits*, 36(11):1808–1820, 2001.
9. P. N. Green and M. D. Edwards. Object oriented development method for reconfigurable embedded systems. *IEE Proc., Comput. Digit. Tech.*, 147(3):153–158, 2000.
10. HiperLan2 Global Forum, Hiperlan specifications, www.hiperlan2.com, 2002.

11. HMAC Standard, National Institute of Standards and Technology. The keyed-hash message authentication code (HMAC). http://csrc.nist.gov/publications/fips.htm, 2002.
12. P. James-Roxby, E. Cerro-Prada, and S. Charlwood. Core-based design methodology for reconfigurable computing applications. *IEE Proc., Comput. Digit. Tech.*, 147(3):142–146, 2000.
13. U. Maurer. Cryptography 2000 ± 10. Informatics—10 Years Back, 10 Years Ahead, Lecture Notes in Computer Science, Springer-Verlag, vol. I, pp. 63–85, 2001.
14. M. McLoone, and J. V. McCanny. Efficient single-chip implementation of SHA-384 & SHA-512. In *IEEE Proc., International Conference on Field-Programmable Technology (FTP)*, pp. 311–314, 2002.
15. A. Menezes, P. Van Oorchot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Inc, 1997.
16. National Institute of Standards and Technology (NIST). Digital Signature Standard, FIPS PUB 186-2, http://csrc.nist.gov/publications/fips/fips186-2.htm, 2002.
17. R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM Proc.*, 21(2):120–126, 1978.
18. M. Roe. Performance of block ciphers and hash functions-one year later. In *Proc., Second Intern. Workshop for Fast Software Encryption*, pp. 83–86, 1994.
19. B. Schneier. *Applied Cryptography—Protocols, Algorithms and Source Code in C*. 2nd edition. John Wiley and Sons, 1996.
20. SHA-1 Standard, National Institute of Standards and Technology (NIST). Secure Hash Standard, FIPS PUB 180-1, www.itl.nist.gov/fipspubs/fip180-1.htm, 2002.
21. SHA-2 Standard, National Institute of Standards and Technology (NIST), Secure Hash Standard, FIPS PUB 180-2, www.itl.nist.gov/fipspubs/fip180-2.htm, 2002.
22. N. Shirazi, W. Luk, and P. Y. K. Cheung. Framework and tools for run-time reconfigurable designs. *IEE Proc., Comput. Digit. Tech.*, 147(3):147–152, 2000.
23. N. Sklavos, P. Kitsos, K. Papadomanolakis, and O. Koufopavlou. Random number generator architecture and VLSI implementation. *IEEE International Symposium on Circuits & Systems (ISCAS) Proc*, vol. IV, pp. 854–857, 2002.
24. N. Sklavos and O. Koufopavlou. On the hardware implementations of the SHA-2 (256, 384, 512) hash functions. In *IEEE International Symposium on Circuits & Systems (ISCAS) Proc.*, vol. V, pp. 153–156, 2003.
25. D. R. Stinson. *Cryptography: Theory and Practice*. CRC Press LLC, 1995.
26. R. Stoica, D. Zebulum, R. Keymeulen, T. Tawel, A. Daud, and A. Thakoor. Reconfigurable VLSI architectures for evolvable hardware: From experimental field programmable transistor arrays to evolution-oriented chips, *IEEE Transactions on VLSI*, 9(1):227–232, 2001.
27. WAP Forum: WAP White Paper, www.wapforum.org, 2002.
28. Xilinx, San Jose, California, USA, Virtex, 2.5 V Field Programmable Gate Arrays, www.xilinx.com, 2002.
29. Xilinx, San Jose, California, USA, A simple method of estimating power in XC40000X1/EX/E FPGAs, Application Brief XBRF 014 v1.0, 2002.