IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-3, Issue-5 E-ISSN: 2347-2693

Implementation of SHA on FPGA

Ankit Anand[1*], Pushkar Praveen[2] and Shruti[3]

[1]*Computer Science, ABES Institute of Technology, Ghaziabad, UP, India*
[2]*Electronics & Communication, RIET,* Gr.Noida, UP, India
[3]*Computer Science, BITS, Bhopal, MP, India*

www.ijcseonline.org

Received: Apr/29/2015 Revised: May/07//2015 Accepted: May/20/2015 Published: May/30/ 2015

Abstract— In this paper, an FPGA based SHA1 core is designed and implemented using the hardware description language VHDL. Hash functions are the most important cryptographic algorithms and used in the several fields of communication integrity and signature authentication. These functions produce a fixed-size fingerprint or hash value for a variable length (very long) message. The hash function SHA-1, Secure Hash Algorithm, is examined in order to find the common constructs that can be used to implement it using hardware blocks of the FPGA. As a result, a hash core supporting SHA-1 and having a standard single bit SPI is proposed. The hardware is described using VHDL and verified on Xilinx FPGA.

Keywords— ASIC, Digital Signature, FPGA, Message Digest, SHA, RTL, VHDL

I. INTRODUCTION

In this paper, hardware implementation of SHA-1 hash function on FPGA is described. The design is described and modeled using a hardware description language, namely VHDL. The recent advancements in the wireless communications area and personal communications systems have made providing information security a more and more important subject. Cryptographic algorithms need to meet specific information security requirements such as data integrity, confidentiality and data origin authentication.

Hash functions are the most important cryptographic algorithms and used in several fields of communication integrity and signature authentication. These functions are operations that take a variable length of input and produce a compressed, fixed length representation of that input. This condensed representation of an arbitrary long input is referred to as a message digest or hash value. The size of the hash value is fixed depending on the hash function being used. The security of a hash algorithm is directly related to the message digest length. A lot of hash functions have been developed till now, and MD5, SHA-1, SHA-256, SHA-384 and SHA-512 are the most popular among them. The oldest of them is the MD5 hash function. This function was developed in 1991 and has an output digest size of 128 bits. In 1993, the ensuing researches on developing more secure hash functions gave birth to a more secure hash function SHA-1 which provides an output hash of 160 bits. In 2002, in order to match security levels offered by other cryptographic algorithms, NIST developed three new hash functions: SHA-256, SHA-384 and SHA-512. These hash functions were standardized with SHA-1 as SHS (Secure Hash Standard). A 224-bit hash function SHA-224, based on SHA-256, has been added to SHS in 2004. Hash calculations are mainly composed of three sections. In the first section, the incoming message is padded and fixed size message blocks are prepared respective to the particular hash function being applied. After these padding operations, the message schedule is created. In this state, message block is further divided into sub blocks to be used in each round of the hash calculation technique. In the hash calculation process, message digest is computed after a fixed number of iterations respective to the algorithm are carried out, using:

I. Algorithm specific constants

II. Message words prepared by the message or word scheduler

III. The chaining or connecting variables

Hash functions can be implemented in hardware and/or software. Though, as security and throughput requirements of the systems increase, it has been observed that software implementations cannot provide the required security and throughput performance. As a result, it is preferred to implement the hash algorithms in hardware.

II. HASH FUNCTION

A. *Introduction to Hash Function*

A hash function is a sort of operation that takes an input and produces a fixed-size string which is called the hash value. The input string can be of any length depending on the algorithm used. The produced output is a condensed representation of the input message or document and usually called as a message digest, a digital fingerprint or a checksum. The size of the message digest is fixed depending on the particular algorithm being used. The security of a hash function is directly related to the message digest length. Pre-image resistance, second pre-image resistance and collision resistance are very important characteristics of any hash function.

I. **Pre-image resistance** (one-way property): For all specified hash values it is computationally very hard to find an input message having that particular hash value.

II. **Second pre-image resistance**: Given an input message m_1, it is computationally very hard to find another input message m_2 such that hash (m_1) = hash (m_2).

III. **Collision resistance**: It is computationally very hard to find any two different inputs that have the same hash value.

B. *Applications of Hash Functions*

The most common use fields of hash functions are verifying data integrity, providing password authentication and generating digital signatures with DSA in applications such as electronic mail, electronic funds transfer, software distribution and data storage which require data integrity assurance and data origin authentication.

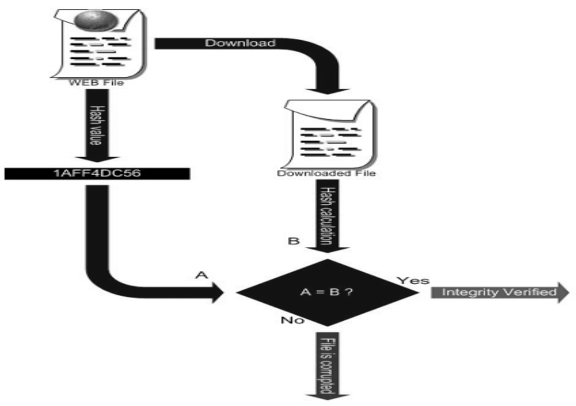

Figure 1: Verifying Data Integrity

The application and verification of a digital signature are illustrated below in Figure 2.2. If a hash function were not used, the recipient would not be sure that the data integrity is protected. Since hash functions are one way functions, any change in the document will change the signature and the signature would not be validated. As a result, when the signature is validated, the recipient makes sure that the document is not altered. Another benefit of digital signatures is the authentication of the source of the messages. Since private key used in the encryption process belongs to a specific user, a valid signature shows that the message is sent by that user.

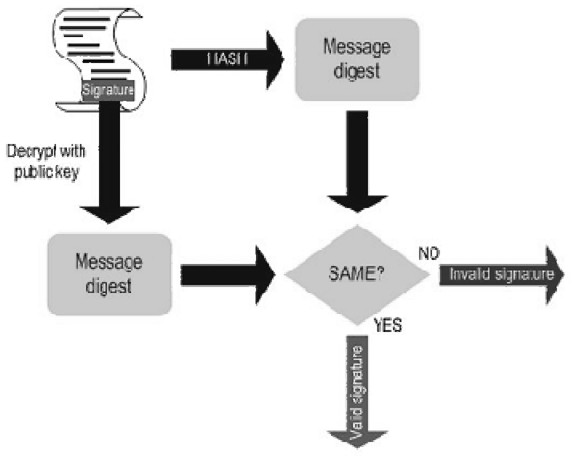

Figure 2: Verification of a Digital Signature

C. Hash Computation Flow

Every hash computation process consists of two stages. The first stage is the preprocessing stage. In this stage the message is padded, parsed into n blocks and the chaining variables are initialized. In the second stage, hash calculation is done. In the hash calculation stage, constants, functions and word operations specific to the hash function are used. Hash calculation generates a message schedule from the padded message and uses that schedule, along with functions, constants and word operations to iteratively generate a series of hash values. The final hash value gen ıe

mes .

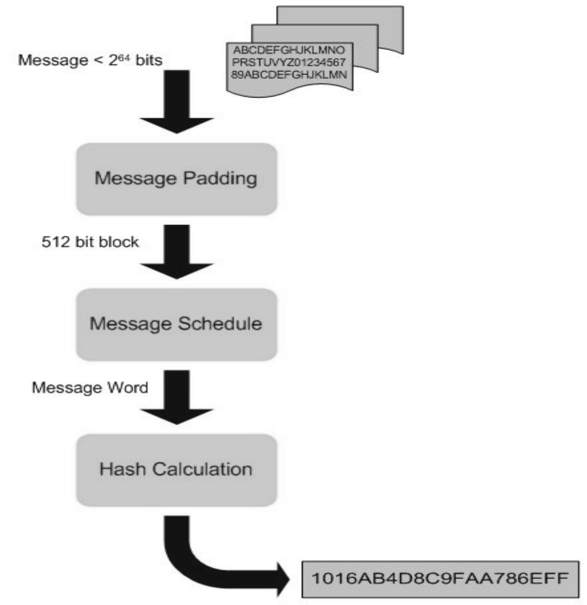

Figure 3: General Hash Computation Flow

D. SHA1 Functions

SHA1 uses three different logical functions. These functions operate on 32 bit words and each has three parameters. These functions are:

1. *Ch(x,y,z) = (x . y)* $\oplus$ *(~x . y)*

   This function is used in first 20 rounds of SHA1 calculations. The architecture of this function is illustrated in Figure 2-5.

2. *Parity(x,y,z) = x* $\oplus$ *y* $\oplus$ *z*

   This function is used in second and last 20 rounds of SHA1 calculations. The architecture of this function is illustrated in Figure 2-6.

3. *Maj(x,y,z) = (x . y)* $\oplus$ *(x . z)* $\oplus$ *(y . z)*

This function is used in third 20 rounds of SHA1 calculations. The architecture of this function is illustrated in Figure.

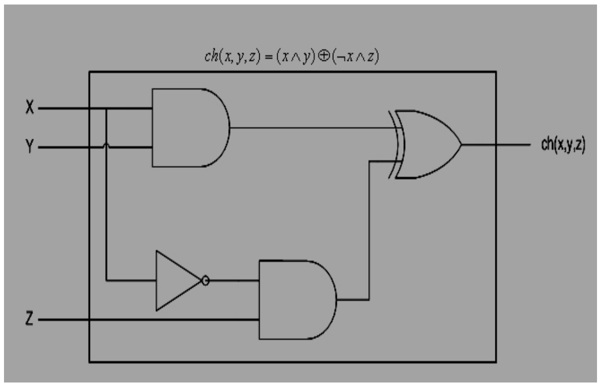

Figure 4: Ch Function Architecture

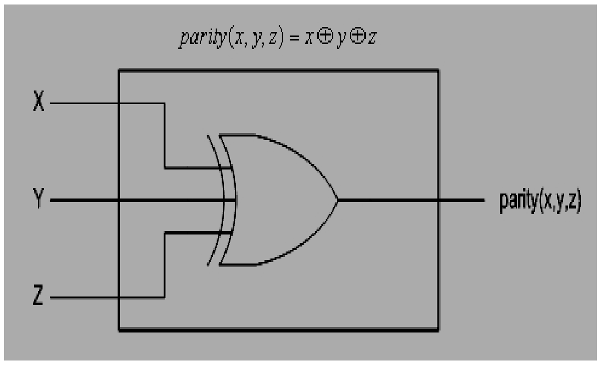

Figure 5: Parity Function Architecture

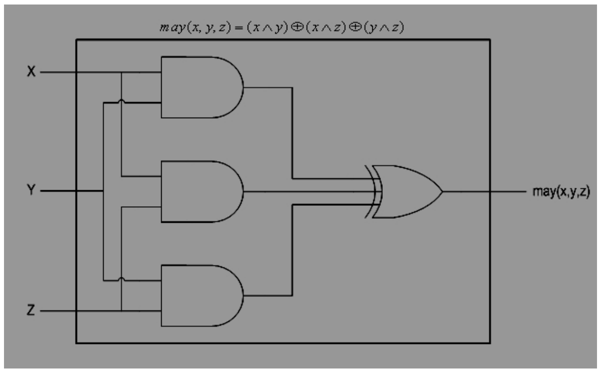
$may(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
X
Y
Z
may(x,y,z)

Figure 6: Maj Function Architecture

These functions are listed below in Table 2-1 according to the SHA1 round number.

SHA1 Functions	Round number, t
f(b,c,d) = Ch(b,c,d)	$0 \le t \le 19$
f(b,c,d) = Parity(b,c,d)	$20 \le t \le 39$
f(b,c,d) = Maj(b,c,d)	$40 \le t \le 59$
f(b,c,d) = Parity(b,c,d)	$60 \le t \le 79$

Table 2-1 SHA1 Functions

1) SHA1 Constants

There are four constants which are used in SHA-1 computations. These are given in Table 2-2

SHA1 Constants	Round number, t
5A827999	$0 \le t \le 19$
6ED9EBA1	$20 \le t \le 39$
8F1BBCDC	$40 \le t \le 59$
CA62C1D6	$60 \le t \le 79$

Table 2-2 SHA1 Constants

2) SHA1 Computation Flow

SHA-1 computation is composed of two stages, preprocessing stage and hash calculation stage. In the preprocessing stage, message is padded, divided into 16 32-bit sub blocks and message schedule is prepared.

Message Padding: Suppose that the length of the message, M, is l bits. Append the bit "1" to the end of the message, followed by *k* zero bits, where k is the smallest, non-negative solution to the equation $l + 1 + k \equiv 448 (\bmod 512)$. Then append the 64-bit block that is equal to the number l expressed using a binary representation. For example, the (8-bit ASCII) message "abc" has length 8*x*3 = 24, so the message is padded with a one bit, then 448 – (25 +1) = 423 zero bits, and then the message length, to make the 512-bit padded message.

Setting the initial hash value: The 160-bit initial hash value $H^{(0)}$ is composed of five 32-bit words which are shown in table.

Hash Calculation: SHA1 may be used to hash a message, M, having a length of l bits, where $0 \le l \le 2^{64}$. The algorithm uses:

A message schedule of 80 x 32-bit words. The words of the message schedule are labeled $W_0, W_1, \ldots\ldots\ldots\ldots, W_{80}$.

Five working variables of 32-bits each. The working variables are labeled as: A, B, C, D and E.

A hash value of five 32-bit words. The words of the hash value are labeled as: $H0^{(i)}$, $H1^{(i)}$, $H2^{(i)}$, $H3^{(i)}$, $H4^{(i)}$ which will hold the initial hash value $H^{(0)}$, replaced by each intermediate hash value (after each message block is processed) $H^{(i)}$ where i denotes the number of 512 bit block being processed in the message M, and ending with the final hash value, $H^{(N)}$ where N is the number of the last 512 bit block in the message M. A single temporary word, T. Previously defined constants which are labeled K_t, where t is the round number. The calculation is carried out as follows:

The message schedule is prepared, i.e. the message word that is going to be used in that round is prepared. This computation is done as described in the following formula:

$$W_t = M_t^i \qquad 0 \le t \le 15$$
$$W_t = ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \qquad 16 \le t \le 79$$

In the above formula M^i_t denotes the tth 32-bit message word of the ith 512-bit message block in the message M. The 5 working variables A, B, C, D and E that are going to be used in the computation are prepared as follows:

$A = H_0^{(i-1)}$

$B = H_1^{(i-1)}$

$C = H_2^{(i-1)}$

$D = H_3^{(i-1)}$

$E = H_4^{(i-1)}$

After these initializations, the final values of the working variables for that round are calculated as described below:

$T = S\hat{}5(A) + f(t; B,C,D) + E + W_t + K_t$
$E = D$
$D = C$
$C = S\hat{}30(B)$
$B = A$
$A = T$

As the final step, intermediate hash values are calculated as described below:

$H_0^{(i)} = A + H_0^{(i-1)}$
$H_1^{(i)} = B + H_1^{(i-1)}$
$H_2^{(i)} = C + H_2^{(i-1)}$
$H_3^{(i)} = D + H_3^{(i-1)}$
$H_4^{(i)} = E + H_4^{(i-1)}$

After 80 rounds the hash value of the incoming 512 bit message block is obtained. Basic SHA-1 computation flow described above is shown below in Figure

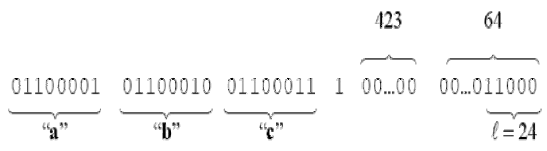

Figure 7: Message padding

$H0^{(0)}$	$H1^{(0)}$	$H2^{(0)}$	$H3^{(0)}$	$H4^{(0)}$
67452301	EFCDAB89	98BADCFE	10325476	C3D2E1F0

Table 2-5 Initial Hash Value for SHA-1

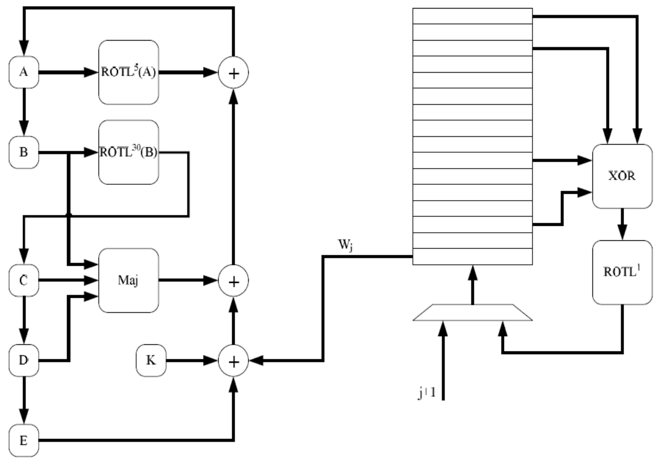

Figure 8: SHA1 Computation Flow

III. HARDWARE IMPLEMENTATION

SHA-1 hash function is implemented in a FPGA. The design is described and captured using a hardware description language Verilog and implemented on Xilinx FPGA. At first step, core design on FPGA concept is examined and the design modules that are going to be implemented are determined. Verilog description of the hash core is written and synthesized using Xilinx ISE 9.2i. The implementation is done for Xilinx's Spartan2 series XC2S200-PQ208 FPGA. To verify the generated Verilog design description, and simulate the design, ModelSim 6.5c is used. The steps in the implementation process are described below:

1. Synthesis: In the synthesis process the syntax of the design is checked and the written VERILOG descriptions are converted to the common constructs on the FPGA such as multiplexers, flip flops, BRAMs etc.
2. Implement design: Before implementation, the constraint file is written to define hardware I/O connections. The implementation constraints file includes timing constraints, package pin assignments and area constraints. Implementing the design means translating, mapping, placement and routing of the design into the targeted Xilinx device. In this process, logical design file generated in the synthesis process, is converted into a native circuit description (NCD file). This file contains hierarchical components used to develop the design and the Xilinx primitives.
3. Generate programming file: In order to generate programming file, the design should have been implemented for the selected FPGA device. This process generates the ".bit" file required to program the FPGA.

4. Configure the device: This process is the process where the FPGA is programmed. FPGA is programmed using Xilinx's Platform Cable USB.

IV. Result

The device utilization summary of the design after synthesis is given below in table:

Logic Utilization	Used	Available	Utilization
Number of Slices	952	2352	40%
Number of Slices Flip Flops	1632	4704	34%
Number of 4 input LUTs	119	4704	2%
Number of bonded IOBs	1693	140	1227%
Number of GCLKs	2	4	50%

Table 4-1 Device Utilization Summary after Synthesis (Spartan2)

The simulation results are also given. The throughput of the design was calculated using LeonardoSpectrum LS2009a_6. The delay results are given below using Xilinx ISE 9.2i.

Timing Parameter	Value
Minimum Period	12.543 ns
Minimum Input Arrival Time before Clock	10.421 ns
Maximum Output Required Time after Clock	6.89 ns
Maximum Frequency	79.329 MHz

Table 4-3 Timing Report after Synthesis (Spartan2)

V. Conclusion

In this paper, a hash core having the capability of performing SHA1 calculations is specified, analyzed and implemented using the hardware description language Verilog. The hash core proposed in this paper can be used in various applications as like providing password authentication, verifying data integrity and generating digital signatures for both data origin authentication and verifying the content of the document easily. The designed hash function core has serial interface that makes communication with the external units such as a personal computer possible. So it provides a great flexibility since it enables remote control of the hash function core. In the market and in the literature an implementation that enables serial communication with the device has not been found.

The throughput of the proposed architecture is less than the present implementations however the proposed implementation has a serial communication interface which makes the design easy to use and consumes less area also high speed and high throughput. The hash function core described using the Verilog can be implemented on to the Xilinx Virtex4 FPGA and there can be some modification to increase the throughput further and also improve the speed of operation.

VI. References

[1] Bowman, M. Debray, S. K., and Peterson L. L, Reasoning about naming systems,1993. .

[2] Ding, W. and Marchionini G. A Study on Video Browsing Strategies. Technical Report. University of Maryland at College Park, 1997.

[3] Fröhlich, B. and Plate J, The cubic mouse: a new device for three-dimensional input. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,2000.

[4] Tavel, P. Modeling and Simulation Design. AK Peters Ltd, 2007.

[5] Sannella, M. J. Constraint Satisfaction and Debugging for Interactive User Interfaces. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington,1994.

[6] Forman G. An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. 3 (Mar. 2003), 1289-1305.

[7] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE.

[8] Y.T. Yu, M.F. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions", Journal of Systems and Software, 2005, in press.

[9] Spector, A. Z. 1989. Achieving application requirements. In Distributed Systems, S. Mullender

[10] Bruce Schneier, "Applied Crptography", John Wiley and Sons, Inc. Press, 1996.

[11] NIST, "Secure Hash Standard", FIPS PUB 180-1, May 1993.