International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463
Vol. 3 Issue 2, February-2014, pp: (353-357), Impact Factor: 1.252, Available online at: www.erpublications.com

An FPGA implementation of SHA3 using keccak function for 512 bit encryption

Ms. Sheetal Deshmukh[1], Ms. Apeksha Sakhare[2]
Department of computer science & engineering, G. H. Raisoni College of Engineering, Nagpur, Maharashtra, India

Abstract: SHA-3 (Secure hash algorithm-3), originally known as Keccak is a cryptographic hash function selected as the winner of the NIST hash function competition. Hash functions have many applications in cryptography mainly in digital signatures and message authentication codes and in network security. Implementation of the main building block (compression function) for five different SHA-3 candidates on reconfigurable hardware is presented. Keccak is a hash function that based on the sponge construction. Keccak hash function has been submitted to SHA-3 competition. In this paper has implemented "SHA-3 512" hash function and high-throughput core designed to work in high clock frequency dedicated to ASIC or expensive FPGA (Spartan 3). Because in many systems for the entire chip the clock frequency is fixed.so even if the hash core can reach a high frequency it has to be clocked at a lower frequency.

Keywords: SHA-3, Keccak 512 bit , High throughput core, Cryptography, Xilinx ISE13.1.

1. INTRODUCTION

Security has become a crucial aspect in the design and use of computer systems and networks. Whether one is designing a wireless communication system, web application, or network protocol, addressing security is an essential engineering criterion. Though a well-designed system is built from a multitude of components, the use of cryptography as a building block is almost unanimous. Cryptography is used to address many security issues, the most pertinent of which are confidentiality, integrity, and authentication. Cryptography encompasses. The design of (cryptographic) primitives, basic building blocks, and protocols/schemes that use these building blocks to construct complex security

systems. Dually, cryptanalysis entails the analysis and evaluation of cryptographic algorithms, including
Primitives and protocols. In this paper only focus on the implementation and analysis of two kinds of cryptographic primitives: stream ciphers and hash functions.The importance of hash functions, in modern cryptography is clearly proven by the different applications and multi-purposes that these are used, in cryptographic protocols. First, digital signatures are the first application of cryptographic secured hash functions. Hash functions serve a dual role in signature schemes: they expand the domain of messages that can be signed by a scheme and they are an essential element of the scheme's security.

Second, message authentication code (MAC) is a keyed hash function satisfying certain cryptographic properties. Third, a common method of client authentication is to require the client to present a password previously registered with the server. Storing passwords of all users on the server poses an obvious security risk. Although, the server need not to know the passwords—it may store their hashes (together with some salt to frustrate dictionary attacks) and use the information to match it with the hashes of alleged passwords. And last, one more usage of hash functions is to "destroy" any structure that may exist in the input, while preserving most of its entropy. Validity of using hash functions for entropy extraction is not based on their cryptographic properties but rather on our belief that a good hash function destroys most of the dependencies that may exist in the bits of its input.

One of the important criteria for the hash function selection is its efficiency on the hardware implementation. So, a comparison in term of implementation is much useful. The implementations are categorized into FPGA and standard-cell ASIC implementations. For FPGA implementation, it is desirable to compare implementations on the same target device or on devices of the same FPGA family. For ASIC implementation, the minimal gate length of the process should be agreed. For comparisons in many modules and different applications, three different hardware implementations are used (the fully autonomous implementation, implementation with external memory and implementation of core functionality) and are described, in the next paragraphs of this section. Candidate algorithms that meet the minimum acceptability requirements are going to comparing, computational efficiency, based on the security, memory requirements, software and hardware suitability, flexibility, simplicity, and licensing requirements.
The security level provided by each submitted algorithm as compared to other submissions (of the same hash length), including first and second preimage resistance, resistance and collision resistance, to generic attacks. Also, if other security factors raised by the public comments during the evaluation process, including attacks which demonstrate that

the actual security of the algorithm is less than the strength claimed by the submitter. This keynote talk deals with the evaluation of computational efficiency, applicable to hardware implementations. Computational efficiency essentially refers to the throughput of an implementation of the software. The memory required for hardware and software implementations will be considered during the evaluation process. Memory requirements will include factors such as gate counts for hardware implementations, and RAM requirements and code size for software implementations. Algorithms with greater flexibility that meet the needs of more users are preferable. For example ''flexibility'' include, the algorithm parameterization (can accommodate additional rounds), parallel implementations of the algorithm in order to achieve higher performance efficiency and efficiency algorithm implementations for wide variety of platforms, including constrained environments such as smart cards. The algorithms will be judged according to relative simplicity of design.

1.1 Stream Ciphers

Stream ciphers are cryptographic algorithms that transform a stream of plaintext messages of varying bit-length into cipher text of the same length, usually by generating a key stream that is then XORed with the plaintext. Using a shared secret key, stream ciphers can be used to provide confidentiality, i.e., restrict access to secret data to the parties in possession of the key by encrypting the plaintext secret data. In general, stream ciphers have very strong security properties, use few resources and high throughput thus making them ideal for mobile applications; well-known examples of stream ciphers include the RC4 cipher used in 802.11 Wireless Encryption Protocol, E0 cipher used in Bluetooth protocol, and the SNOW 3G cipher used by the 3GPP group in the new mobile cellular standard.

1.2 Hash Functions

Like hash functions, stream ciphers are important cryptographic primitives. However, hash functions transform arbitrary-length input messages into fixed-length message digests. They are used in many applications in commitment schemes, digital signatures and message authentication codes. To this end they are required to satisfy different security properties. These security properties include.

i) Preimage resistance, i.e., given f (x) it is infeasible to find x,
ii) second preimage resistance, i.e., given x it is infeasible to find x1 _ x : f (x) = f (x1), and
iii) collision resistance, i.e., it is infeasible to find x, x1 : x1 _ x and f (x) = f (x1).Informally, a hash function is collision resistant if it is practically infeasible to find two distinct messages m1 and m2 that produce the same message digest.

2. PROBLEM DEFINITION

In previous design BLAKE hash function, JH hash function, Skein Hash function and Ghrostl hash function is implemented. All these hash function required more rounding for encryption as shown in table1. Our objective is to provide more encryption using less no of rounding i.e more permutation has to done, in 5 candidates of sha-3 finalist. Keccak hash function provides more encryption in less rounding i.e for 512 bit encryption 10 rounds will be required. To design and implement SHA-3 algorithm using keccak hash function. Cryptography has to be done for 512 bits. Design has to be done using HDL coding which has to support high throughput core
.

3. OBJECTIVE

Main objective is to design SHA3 algorithm done by candidate keccak3 for 512 bit cryptography. Verilog coding for proposed design is done in Xilinx ISE tool. Optimization Target is one of the most important decisions to make in order to develop a fair comparison. The possible choices include Maximum Throughput, Minimum Area, Maximum Throughput to Area Ratio, Minimum Latency, etc. All of the aforementioned targets can be used to make a comparison. Out of them, we have selected Maximum throughput to Area Ratio as our criteria of choice. In this selection has advantages over other possible choices. First, it is practical, as hardware cores are typically applied in situations, where the size of the processed data is significant and the speed of processing is essential. Secondly, throughout the entire design process this optimization criterion is a very reliable guide. At every junction where the decisions must be made, it starting from the choice of a high-level hardware architecture and down to the choice of the particular FPGA tool options, this criterion facilitates the decision process, leaving very few possible paths for further investigation.

4. METHODOLOGY

Architecture of the core

The architecture is of the whole core given in the following figure :

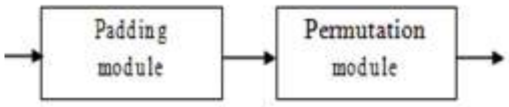


Fig 1: The architecture of the whole core

The width of the user input is far less than 576 bit. So the padding module uses a buffer to assemble the user input. If the buffer grows full, the padding module notices the permutation module its output is valid. Then the permutation module begins calculation, the buffer cleared, the padding module waiting for input simultaneously. In the high throughput core, two rounds are done per clock cycle. The round constant module is implemented by combinational logic, saving resource than block RAM, because most bits of the round constant is zero.

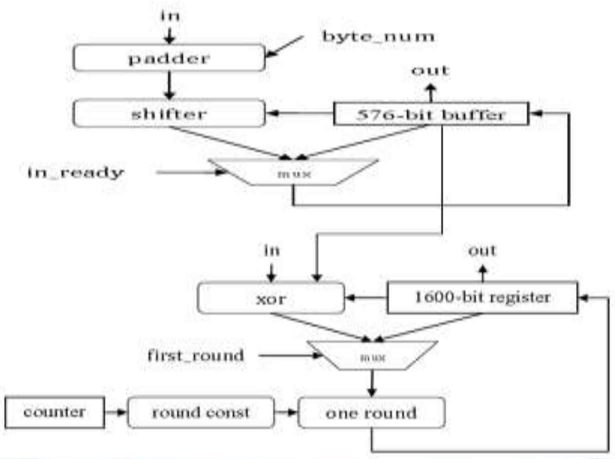


Fig .2 Architecture of padding and permutation module

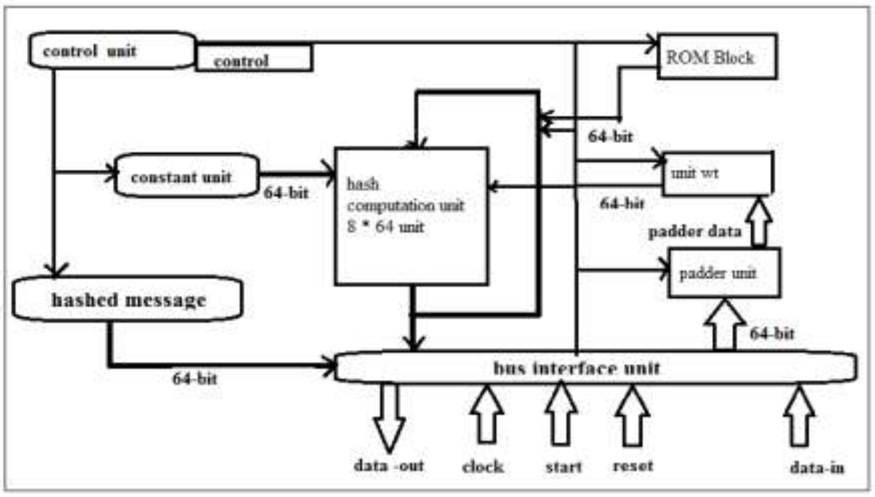


Fig. 3 Architecture of the hash function keccak 512 bit

5. Simulations

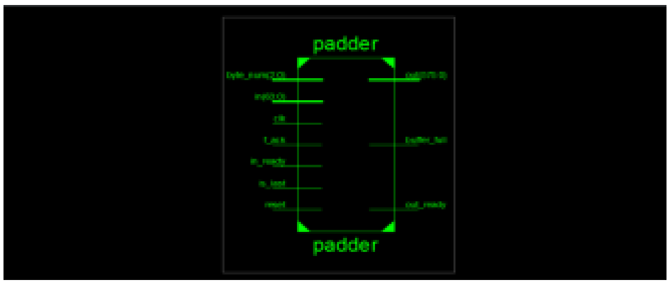


Fig 4. RTL view of the Padder module

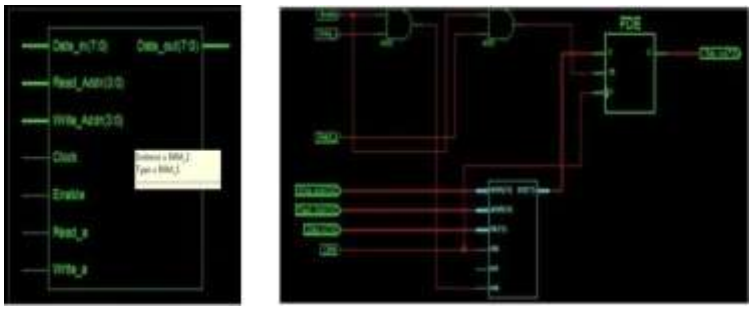
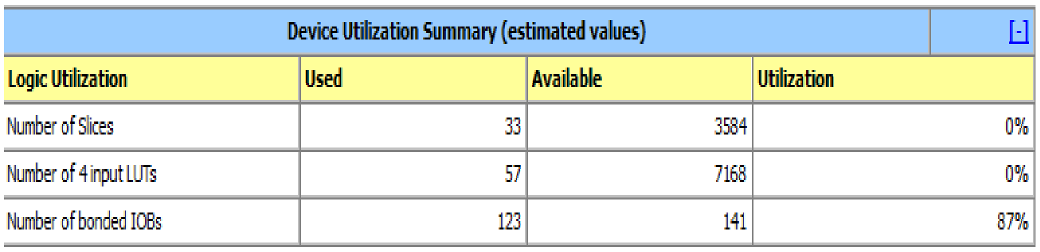


Fig. 5 Top level entity and RTL of RAM

In this table number of slices, total numbers of LUT'S, number of bonded IOB'S are present. For design of padder module only 57 LUT'S and 33 slices are required.



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	33	3584	0%
Number of 4 input LUTs	57	7168	0%
Number of bonded IOBs	123	141	87%

Fig. 6 synthesis report for Padder module

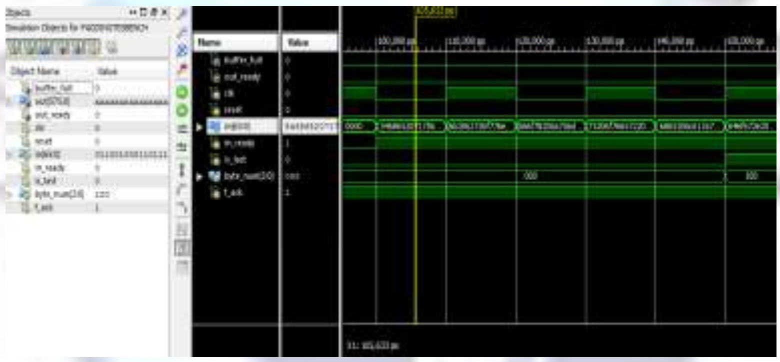


Fig. 7 waveform of the padder module

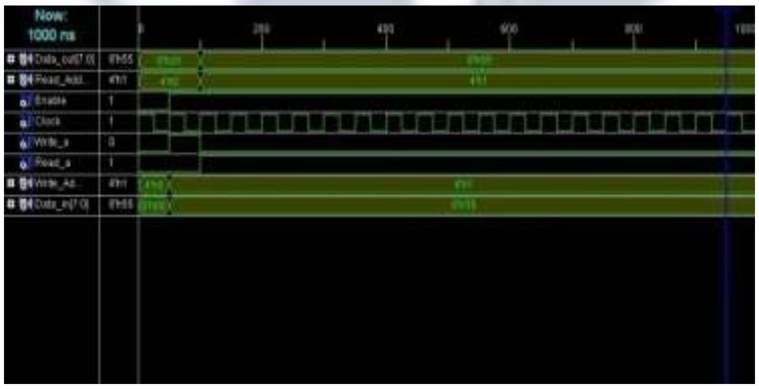


Fig. 8 waveform of the RAM module

6. CONCLUSION

Keccak hash function hardware implementations are described in this paper. For implementation of this function use spartan-3 FPGA device in this paper provide more encryption using less no of rounding i.e more permutation has to done, in 5 candidates of sha-3 finalist. Keccak hash function provides more encryption in less rounding i.e for 512 bit encryption 10 rounds will be required to design and implement SHA-3 algorithm using keccak hash function.

REFERENCES

[1]. Fatma Kahri, Belgacem Bouallegue, Mohsen Machhout and Rached Tourki Electronics and Micro-Electronics Laboratory "An FPGA implementation of the SHA-3: The BLAKE Hash Function"(2012) (E. µ. E. L) Faculty of Sciences of Monastir, Tunisia Kahrifatma@gmail.com.

[2]. National Institute of Standard and Technology (NIST), "Cryptographic hash algorithm competition", 2007, available on lineat http://www.nist.gov/itl/csd/ct/hash_competition.cfm.

[3]. A. H. Namin& M. A. Hasan (2010 a), Implementation of the Compression Function for Selected SHA-3 Candidates on FPGA. Retrieved Feb. 25th, 2010, from University of Waterloo, Department of Electrical and Computer Engineering.

[4]. [4] Schorr (2010), Performance Analysis of a Scalable Hardware FPGA Skein Implementation. Retrieved February 2010,_from Kate Gleason College of Engineering Department of Computer Engineering Rochester, New York.

[5]. J. Elbirt (2009), Understanding and Applying Cryptograph and Data Security. Book ISBN 978-1-4200-6160-4 (alk. paper).

[6]. Regenscheid, R. Perlner, S. Chang, J. Kelsey, M. Nandi, & S. Paul (2009), Status Report on the First Round of the SHA- 3 Cryptographic Hash Algorithm Competition. Retrieved September 2009, from National Institute of Standards and Technology, U.S. Department of Commerce.

[7]. C. Rechberger (2010), Second-Preimage Analysis of Reduced SHA-1, from KatholiekeUniversiteit Leuven, Department of Electrical Engineering.

[8]. Imad Fakhri Alshaikhli, Mohammad A. Ahmad, Hanady Mohammad Ahmad (2012). "Protection of the Texts Using Base64 and MD5." JACSTRVol 2, No 1 (2012)(1): 12.

[9]. E. Andreeva, B. Mennink, B. Preneel& M. Skrobot (2012), Security Analysis and Comparison of the SHA-3 Finalists BLAKE, Grostl, JH, Keccak, and Skein. from KatholiekeUniversiteit Leuven.

[10].Belgium. E. B. Kavun& T. Yalcin (2012), On the Suitability of SHA-3 Finalists for Lightweight Applications. from Horst Görtz Institute, Chair of Embedded Security, Germany.