International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:2, No:2, 2008

Efficient Pipelined Hardware Implementation of RIPEMD-160 Hash Function

H. E. Michail, V. N. Thanasoulis, G. A. Panagiotakopoulos, A. P. Kakarountas, and C. E. Goutis

***Abstract*—**In this paper an efficient implementation of Ripemd-160 hash function is presented. Hash functions are a special family of cryptographic algorithms, which is used in technological applications with requirements for security, confidentiality and validity. Applications like PKI, IPSec, DSA, MAC's incorporate hash functions and are used widely today. The Ripemd-160 is emanated from the necessity for existence of very strong algorithms in cryptanalysis. The proposed hardware implementation can be synthesized easily for a variety of FPGA and ASIC technologies. Simulation results, using commercial tools, verified the efficiency of the implementation in terms of performance and throughput. Special care has been taken so that the proposed implementation doesn't introduce extra design complexity; while in parallel functionality was kept to the required levels.

***Keywords*—**Hardware implementation, hash functions, Ripemd-160, security.

I. Introduction

NOWADAYS many applications like the Public Key Infrastructure (PKI) [1], IPSec [2], Secure Electronic Transactions (SET) [3], and the 802.16 [4] standard for Local and Metropolitan Area Networks incorporate authenticating services. All these applications pre-suppose that an authenticating module that includes a hash function is nested in the implementation of the application. Moreover, digital signature algorithms like DSA [5] that are used for authenticating services like electronic mail, electronic funds transfer, electronic data interchange, software distribution, data storage etc are based on using a critical cryptographic primitive like hash functions. Hashes are used also to identify files on peer-to-peer filesharing networks for example, in an ed2k link. Furthermore, hashing cores are also essential for security in networks and mobile services, as in SSL [6], which is a Web protocol for establishing authenticated and encrypted sessions between Web servers and Web clients. They are also the main modules that exist in the HMAC implementations that produce Message Authentication Codes (MAC's) [7].

This work was supported by the project PENED 2003 No 03ED507, which is funded in 75% by the European Union- European Social fund and in 25% by the Greek state-Greek Secretariat for Research and Technology.

H. E. Michail, V. N. Thanasoulis, G. A. Panagiotakopoulos, A. P. Kakarountas and C. E. Goutis are with the Department of Electrical & Computer Engineering, University of Patras, Patras, GR-26500, Greece (e-mail: michail@ece.upatras.gr, kakarountas@ieee.org, goutis@ece.upatras.gr; corresponding author: Harris E. Michail phone: 0030697487335; e-mail: michail@ece.upatras.gr).

Taking into consideration the rapid evolution of the communication standards that include message authenticity, integrity verification and non-repudiation of the sender's identity, it is obvious that hash functions are very often used to most popular cryptographic schemes like those in IPSec in conjunction with other cryptographic primitives. All the pre-mentioned applications which incorporate hash functions are being used more and more widely lately. In many of these cryptographic schemes the performance of the incorporated hash functions determines the performance of the whole security scheme.

From the latter it is also quite clear that all applications that incorporate hash functions are addressing numerous users-clients and thus throughput increase is the prior requirement, particularly for the corresponding server of these services. This is because the cryptographic system, especially from the server part, has to reach the highest degree of throughput in order to satisfy immediately and securely all requests for service from the users-clients.

In many of these cryptographic schemes, the throughput of the incorporated hash functions specifies the throughput of the system. High-speed calculation of the hash functions is strongly related to the streamlined communication of the two subscribers of the latter mentioned applications. Especially in applications that transmission and reception rates are high, any latency or delay on calculating the digital signature of the data packet leads to degradation of the network's quality of service.

The latter mentioned facts were a strong motivation for the proposition of a novel cryptographic algorithm strong in cryptanalysis. RIPEMD proposal was developed in the framework of the EU project RIPE (Race Integrity Primitives Evaluation). In addition to the demanded high security level, the need for effective security implementation is a significant factor for the selection of a hardware implementation. In this work we propose an efficient hardware implementation of Ripemd-160 hash function. Our implementation is structured in such way so that it can be used in a variety of applications, maintaining the flexibility of similar software constructions. The cryptographic core maintains the functionality of the algorithm, keeps the area small enough with achieving both high operating frequency and throughput (primary due to the 4-pipeline stage that has been applied), as required by most portable communication devices.

This paper is organized as follows: In section 2 the Ripemd-160 hash function is presented. In section 3 follows the

presentation of the cryptographic core. The synthesis results are discussed in section 4. Finally, in section 5 follow the conclusions derived from the proposed implementation.

II. RIPEMD-160 Hash Function

The fixed length of each of the processed blocks is 512 bits. As it is described in [8] Ripemd-160 hash function has a bitsize of the hash-result and chaining variable of 160 bits (five 32-bit words). Ripemd-160 uses two parallel processes of five rounds, with sixteen operations for each round (5 x 16 operations for the process).The inputs for in every round is five bit words of data, the message word X_i and a 32 bit constant K_i.

Unfolding the expressions of a_t, b_t, c_t, d_t, e_t that describes the five input words, it is observed that b_{t-1}, c_{t-1} , d_{t-1} , e_{t-1} values are assigned directly to outputs c_t, d_t, e_t, a_t respectively. In Eq. (1) the expressions of a_t, b_t , c_t , d_t , e_t are defined.

$$\begin{aligned} e_t &= d_{t-1} \\ d_t &= ROL_{10}(c_{t-1}) \\ c_t &= b_{t-1} \\ b_t &= e_{t-1} + ROL_s[\, f_t(b_{t-1}, c_{t-1}, d_{t-1}) + a_{t-1} + X_i + K_j \,] \\ a_t &= e_{t-1} \end{aligned} \tag{1}$$

where $ROL_x(y)$ represents cyclic shift (rotation) of word y to the left by x bits and $f_t(z, w, v)$ denotes the non-linear function which depends on the round being in process.

From Eq.(1), it is derived that the maximum delay is observed on the calculation of the b_t, value. Obviously the critical path consists of three addition stages as it can be seen observing Fig. 1 and a multiplexer via which the values pass each time to/and feed the operation block.

RIPE-MD can also be used in the HMAC implementation in order to authenticate both the source of a message and its integrity without the use of any additional mechanisms.

III. RIPEMD-160 Core

The proposed system architecture is illustrated in the following Fig. 2. This implementation is suitable for any system that uses the Ripemd-160 hash function, constrained only by the assumption that the start_counter1 signal is applied one clock cycle before the final padded message is available in the 512-bit input block_in of Ripemd-160 core. Concurrently to the arrival of the 512-bit input block_in the signal start_round1 has to be applied and 80 clock cycles later the hash value is pending on the 160-bit output ripemd-160 hash value of the Ripemd-160 core.

The Ripemd-160 hash function consists of two parallel processes of five rounds, with sixteen transformations for each round (5 x 16 transformations for the process). All the rounds in the processes are similar but each one performs a different operation on five 32-bit inputs. The data (input message) are processed 16 times in each transformation round resulting in 80 transformations performed in total per process. Each operation is performed in every time instance. In each process

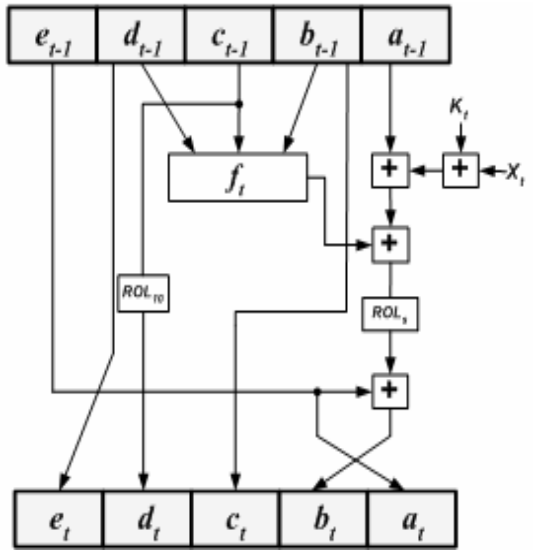

Fig. 1 RIPEMD - 160 operation block of a round

a certain 32-bit word X_i must be supplied to the transformation round. This 32-bit X_i results from a specific process on the 512-bit input block_in through appropriate permutations on the initial 16 X_i word blocks that are exported from 512-bit input block. The computation of each X_i (beyond the first 16 X_i that are computed by a simple split of the 512-bit input block_in) takes place in "X_i' permutation" unit as it is shown in Fig. 1. The implemented Ripemd-160 core has five primary inputs h_0, h_1, h_2, h_3, h_4 where the initial values are supplied. These can be the ones specified by the standard [7] or others that have occurred from process of the former block of the message.

In order to apply pipeline stages to the Ripemd-160 core, ten 160-bit registers have to be used between the data transformation rounds. In Fig. 2, these registers are implanted at the end of each transformation round and this technique ensures that five 512-bit data blocks can be processed at the same time and finally a 160-bit message digest is produced every 16 clock cycles.

All multiplexers are controlled by the Count_16 component. The Count_16 component also arranges when the 16 X_i's - that are pending in the input of the register and are related to the process in the next round - will be stored in the register.

For the five messages that can be concurrently processed, 2*5*16 X_i's (160 X_i's for every message) must be computed and stored. Instead of this, for every message only 2*16 X_i's are computed and saved, those used in the currently processed transformation round for the two processes. So, this way only 2*5*16 X_i's (32 X_i's for every message) must be computed and stored.

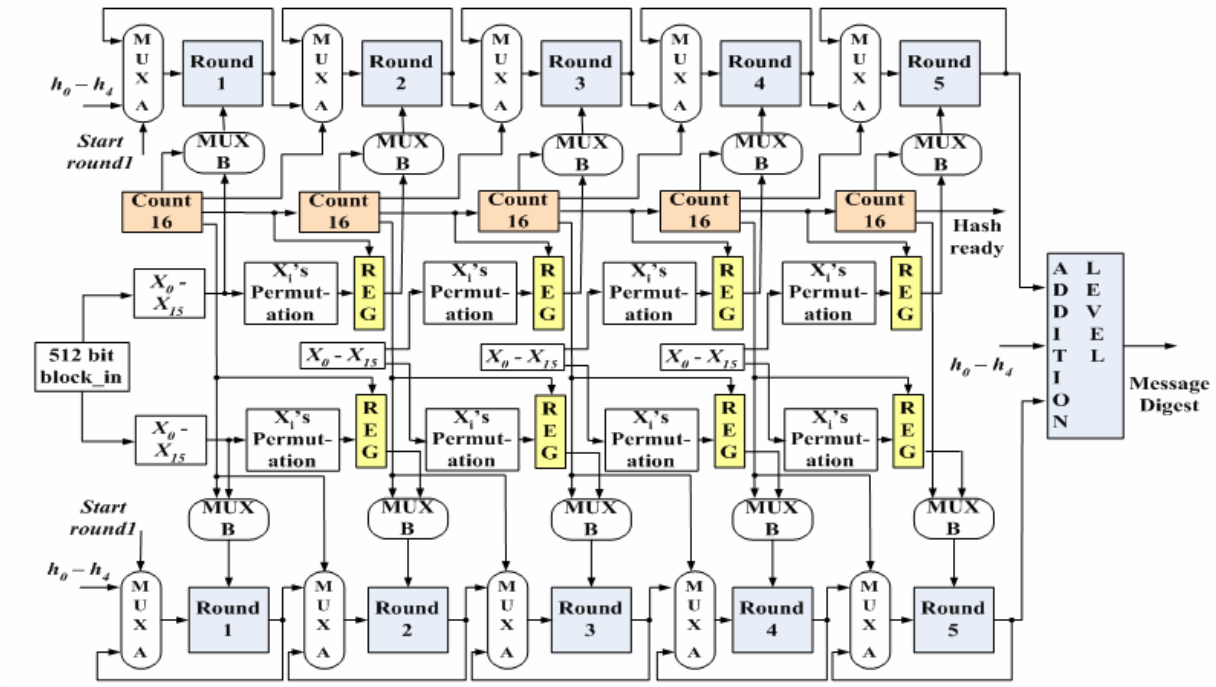

Fig. 2 Ripemd-160 Core

The X_i's for the next transformation round of every message are computed and pending in the "X_i' permutation" units. When the message proceeds to the next transformation round the X_i's needed in the new round are stored in registers. These registers supply the new round with the 16 X_i's and at the same time the 16 X_i's required for the next round, for the same message, are computed and pending on the corresponding "X_i' permutation" unit.

Implanting registers at the end of every transformation round and considering the fact that the process in every round lasts 16 clock cycles leads to the following conclusion; the implemented Ripemd-160 core can be supplied with a new 512-bit input block_in if only 16 clock cycles have past from the last time instance that the Ripemd-160 core was again supplied. This leads to a maximum throughput limit for the Ripemd-160 core which is determined by the fact that at each processing instance only 5 different messages at most can be concurrently at process. During this 16 clock cycles the 512-bit input block_in must remain stable in order for the Ripemd-160 core to function properly. This way we save up time (1 clock cycle) and hardware that would be spent if it was decided to store in registers the 512-bit input block_in. This wouldn't improve at all the implementation and it doesn't incur any design or functional problems. The 512-bit input block_in is kept stable to the output of the unit that is responsible for the padding procedure and will be mentioned to the next section.

The five Count_16 components are responsible for the synchronization of all procedures that have to be done in order to obtain the hash value. Each one is enabled only when in the corresponding transformation round there is a message that is being processed. When the process in that round reaches the end the Count_16 component arranges that the process will continue to the next round. The five Mux_A multiplexers for each process ensure that every time the process of a message ends in one round, the process will be continued in the next round. The five Mux_B multiplexers also for each process, ensure at any time instance that the correct X_i is supplied to each operation of the round during the process of a message.

The presented implementation illustrates all the necessary implementation details and it is also indented to serve as a reference implementation to compare with in order to propose some improvements in its architecture which correspondingly are going to increase the performance of the whole RIPE-MD hashing core.

IV. Synthesis Results

The latter presented hashing core was captured in VHDL and was fully simulated and verified using the Model Technology's ModelSim Simulator. Verification of the designs' operation was achieved using a large set of test vectors, apart from the test example proposed by the standards.

The synthesis tool used to port VHDL to the targeted technologies was Synplicity's Synplify Pro Synthesis Tool. Simulation of the designs was also performed after synthesis, exploiting the back annotated information that was extracted from the synthesis tool. Further evaluation of the designs was performed using the prototype board for the Xilinx Virtex device family.

Probing of the FPGA's pins was done using a logic analyzer. No scaling frequency technique was followed, selecting one master clock for the system, which was driven in the FPGA from an onboard oscillator. The behavior of the implementation was verified exploiting the large capacity of the FPGA device. Thus, a BIST unit was developed which was responsible for providing predefined inputs to the RIPEMD and monitoring of the output response. This allowed additionally to cope with high output throughput. Further evaluations of the designs were performed using the prototype boards for the Xilinx Virtex device family.

The achieved operating frequency, the required integration area and the corresponding throughput in three different FPGA families are pictured in Table I. The throughput it is calculated from Eq.2.

$$Throughput = \frac{\#bits \cdot f_{operation}}{\#operations} \quad (2)$$

where #bits is equal to the number of bits processed by the hash function, #operations corresponds to the required clock cycles between successive messages to generate each Message Digest and $f_{operation}$ indicates the maximum operating frequency of the circuit.

The results of Table I give a rough estimation of the performance of the proposed implementation in various FPGA platform boards. When placement and routing take place using vendor's tools, a more realistic performance of the RIPE-MD implementation arises that slight a bit from the reported results stated in Table I. The implementation is synthesizable in most FPGA families, resulting in a reusable, general purpose implementation.

TABLE I
IMPLEMENTATION SYNTHESIS RESULTS

Technology	Area (CLBs)	Frequency (MHz)	Throughput (Gbps)
Virtex	1798	51.6	1.65
Virtex-E	1856	64.6	2.08
Virtex-II	1985	74.6	2.38

V. CONCLUSION

In this paper, an efficient hardware implementation of the Ripemd-160 hash function has been presented. The introduced implementation is targeting general purpose applications and therefore can be integrated in a wide range of systems requiring safety, such as systems for digital signatures, data integrity and message authentication. Our implementation maintains the functionality of the algorithm, keeps the area small enough with acceptable operating frequency and throughput. All this make the application of this implementation successful in communication protocols such as IPsec, WAP and security schemes in general. This implementation is also intended to serve as a reference one for future optimized implementations of RIPE-MD algorithm providing as well all low-level implementation details.

ACKNOWLEDGMENT

This work was supported by the project PENED 2003 No 03ED507, which is funded in 75% by the European Union-European Social fund and in 25% by the Greek state-Greek Secretariat for Research and Technology.

REFERENCES

[1] RFC 2510 - Internet X.509 PKI - Certificate Management Protocols, available at www.ietf.org/rfc/rfc2510.txt
[2] SP800-77 , Guide to IPSec VPN's, National Institute of Standards and Technology (NIST).
[3] Secure Electronic Transactions: An Overview, available at www.davidreilly.com/topics/electronic_commerce/essays/secure_electronic_transactions.html
[4] Johnston D, Walker J, Overview of IEEE802.16 Security, IEEE Security and Privacy, May-June 2004.
[5] FIPS 186, (DSS), Digital Signature Standard Federal Information Processing Standard, (FIPS), Publication 180-1, NIST, US Dept of Commerce.
[6] Introduction to SSL, available at http://docs.sun.com/source/816-6156-10/contents.htm
[7] FIPS 198, The Keyed-Hash Message Authentication Code (HMAC) Federal Information Processing Standard, (FIPS), Publication 180-1, NIST, US Dept of Commerce.
[8] H. Dobbertin, A. Bosselaers, B. Preneel, RIPEMD-160: A Strengthened Version of RIPEMD, 18 April 1996.