

MD5 Hash Algorithm Hardware Realization on a Reconfigurable FPGA Platform

I.N. Tselepis¹, M.P. Bekakos¹, A.S. Nikitakis¹ and E.A. Lipitakis²

Abstract – MD5 is one of the most important hash algorithms today. It has been designed by R. Rivest in 1992 and it is considered as a standard in hash function design. In this paper, the hardware implementation of the MD5 algorithm on reconfigurable devices, such as FPGAs, is investigated. These hardware devices provide high performance at low cost, which makes them suitable for cryptographic and cryptanalytic purposes. The high performance of the implementation is the main goal of the design presented herein.

Index Terms – Cryptography, FPGA, VHDL

I. INTRODUCTION

The need of secure internet communication has given rise to message authentication as an essential technique to verify the integrity of received data. Every cryptographic system, e.g., e-mail applications, bank transactions, etc, uses hash functions for message authentication purposes. With the advent of public key cryptography and digital signature schemes, cryptographic hash functions gained much more prominence. In IPSec hash functions are utilized to achieve data integrity assurance, data origin and message content authentication.

Using Hash Functions it is possible to produce a fixed length fingerprint that depends on the whole message and ensures that it has not been altered during an insecure transmission. Through the hashing procedure message length is effectively reduced and as a consequence the overall computation time involved in the digital signing procedure. For example, in a public key cryptosystem, e.g., RSA, a large message must be compressed in a secure way through a hashing procedure, before being encrypted with a private (secret) key.

MD5 is a message digest or otherwise a hash algorithm, developed by Ron Rivest at MIT. It is an extension of his previous algorithm, MD4, which is a little faster than MD5. This has been the most widely used secure hash algorithm, particularly in Internet-standard message authentication.

MD5 has also been proposed as the default authentication option in IPv6. The algorithm input is a message of arbitrary length and the algorithm output is a 128-bit message digest (or hash-value) of the input. The algorithm involves the following four steps:

- Append Padding Bits
- Append Length
- MD Buffer Initialization
- Process Message in 16-Word Blocks

II. APPEND PADDING BITS

The message is ‘padded’ (extended) so that its length, in bits, is congruent to $448 \bmod 512$. That is, the message is extended so that it is just 64 bits short of being a multiple of 512 bits long. Even if the message length is already congruent to $448 \bmod 512$, padding is still performed. Padding is performed as follows: a single ‘1’ bit is appended to the message, and then ‘0’ bits are appended, so that the bit length of the padded message becomes congruent to $448 \bmod 512$. In all, at least one bit and at most 512 bits are appended.

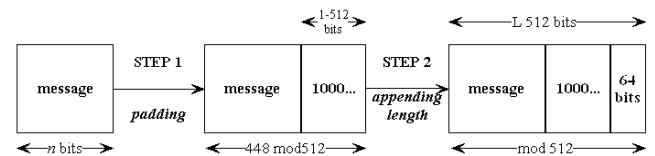


Figure 1. The first two steps of MD5.

III. APPEND LENGTH

A 64-bit representation of the original message length is appended to the result of the previous step. In the unlikely event that the message length is greater than 264, only the low-order 64 bits of the length are used. At this point the resulting message has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let $M[0 \dots N-1]$ denote the words of the resulting message, where N is a multiple of 16. Figure 1 shows the procedure of the initial two algorithmic steps.

¹ Laboratory of Digital Systems, Department of Electrical and Computer Engineering, Democritus University of Thrace, 67100, Xanthi, Hellas. {itselepi, mbekakos, anikitak}@ee.duth.gr

² Advanced Computational Mathematics Research Group, Department of Informatics, Athens University of Economics & Business, 76 Patision Street, Athens 104 34, Hellas. eal@aueb.gr

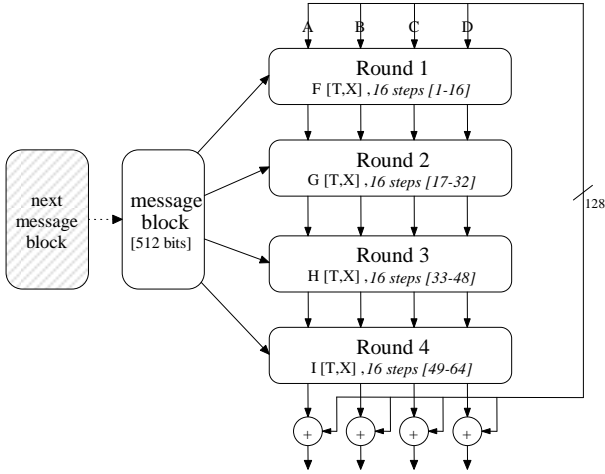


Figure 2. The four-round process.

IV. MD BUFFER INITIALIZATION

A four-word buffer (A, B, C, D), called MD buffer, is used to compute the message digest. Each of A, B, C and D is a 32-bit register. These registers are initialized to the following values in hexadecimal, with low-order bytes first:

$$\begin{aligned}
 \text{word A: } & 01233456 \\
 \text{word B: } & 89ABCDEF \\
 \text{word C: } & FEDCBA98 \\
 \text{word D: } & 76543210
 \end{aligned} \tag{1}$$

V. PROCESS MESSAGE IN 16-WORD BLOCKS

This is the essential part of the algorithm consisted of *four* processing ‘rounds’. The *four* rounds have similar structure, but each uses different round function *F*, *G*, *H* and *I*. The whole *four* round process is depicted in Figure 2, where

$$\begin{aligned}
 F(x, y, z) &= (((x) \& (y)) \mid ((\sim x) \& (z))) \\
 G(x, y, z) &= (((x) \& (z)) \mid ((y) \& (\sim z))) \\
 H(x, y, z) &= ((x) \wedge (y) \wedge (z)) \\
 I(x, y, z) &= ((y) \wedge ((x) \mid (\sim z)))
 \end{aligned} \tag{2}$$

and \wedge represents XOR, $\&$ represents AND, \mid represents OR, \sim represents NOT.

Each round consists of 16 steps and each step uses a 64-element table T [1 ... 64] constructed from the function

$$T(i) = 4294967296 \cdot \text{abs}[\sin(i)] \tag{3}$$

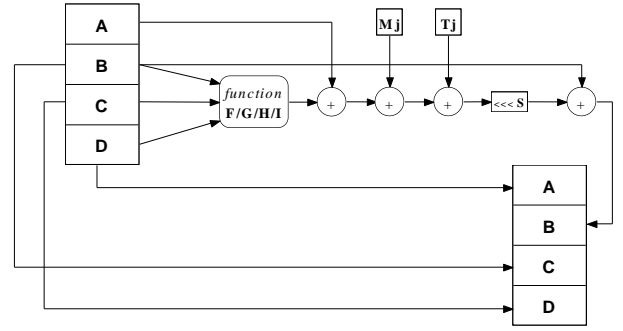


Figure 3. Single-step operations.

VI. FPGA HARDWARE REALIZATION OF MD5

Reconfigurable devices such as FPGAs are an attractive option for hardware implementations, as they provide performance combined with low cost and flexibility. With FPGAs, a dynamic development and immediate testing of a system is possible.

Hash functions use easy computations, making thus hardware implementations practical. FPGAs offer high flexibility, the capability for frequent modifications, low development and purchase cost. FPGAs also offer faster development time than ASICs and can be used as rapid prototyping devices in VLSI design.

For this implementation, as target device was selected the Xilinx Virtex II FPGA family. It provides high performance combined with high capacity programmable logic. The implementation described in VHDL using the Modelsim tool. The Leonardo Spectrum tool was used for the synthesis, placing and routing.

The MD5 Core consists of three main components: The *interface_module*, the *md5_main_core* and the *state_machine* (FSM), as shown in Figure 4.

The *interface_module* provides the main interface between the user and the rest of the system. The signal *hashing_en* is high, when user has data to hash. If the state machine is ready to accept data then the *ack_data* signal is returned to the user; otherwise user has to wait. The *interface_module* accepts the 512-blocks in 32-bit words. After the last 32-bit of the last block is read, the *interface_module* performs the padding and the length appending of the message.

The *state_machine* (FSM) controls the function of the whole system. It defines all the necessary states to complete the MD5 procedure and controls the data flow between the *interface_module* and the *MD5_main_core*. The FSM is utilized as a combination of Moore and Mealy architecture. There has been an effort for the outputs of the FSM to be unlocked in order to reduce the latency.

The *md5_main_core* performs the main part of the algorithm. It consists of *one* 4-bit counter (*message_counter*), *two* 512-bit RAMs the *main_loop_core* and the *chaining_mux* component. The *message_counter* determines the write-addressing to the YY and XX RAMs. The data passes from YY to XX RAM and the YY RAM is then ready to accept the next 512-bits of data. Each 512-bits of data are used by the *main_loop_core* component in order to be processed. The 4-bit counter also gives a done signal (*message_count_done*) to the *state_machine* after loading is complete.

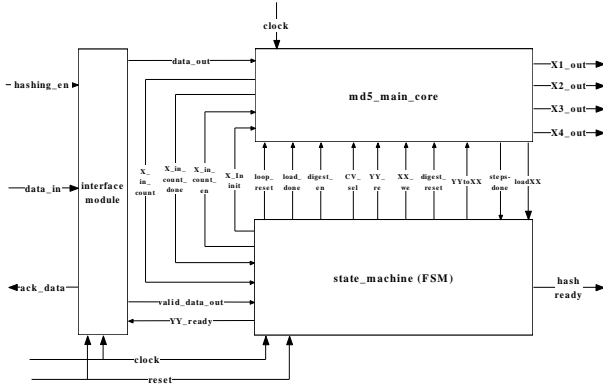


Figure 4. The complete MD5 core.

The *chaining_mux* component (cf. Figure 5) holds the intermediate hash-values of each processed 512-bit block. It also initializes the MD buffer (cf. Figure 2) at the start of the procedure. After the last 512-bit block is been processed, the *state_machine* sends the signal *digest_en* to *chaining_mux* to pass the hash-value to output pins (X1_out to X4_out).

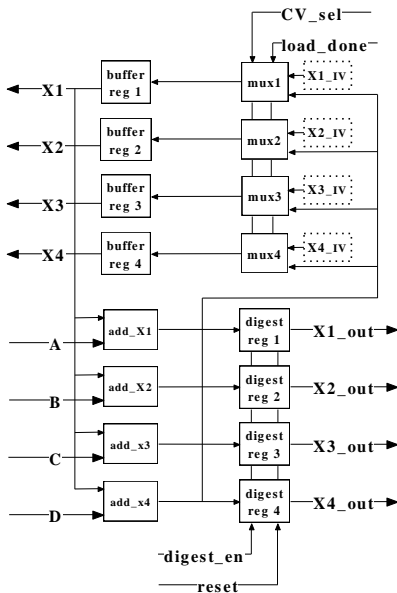


Figure 5. The Chaining_Mux component.

The *main_loop_core* (Figure 6) performs the 64-steps of the algorithm in a single clock cycle per step. Each 512-bits

block requires 65 clock cycles to be processed, excluding the loading of the first block. The *loop_counter* is a 6-bit up-counter which provides these steps to the *computational_block*. The *computational_block* provides the combinational logic for each single step operation. The *rotate_LUT* defines the amount of the rotation needed in each step. The *Xk_LUT* addresses XX_RAM and defines the next 32-bit word of the current block to be processed. Finally, *T_LUT* gives the constant $T[i+1]$ to the *computational_block*, as needed for each single step operation.

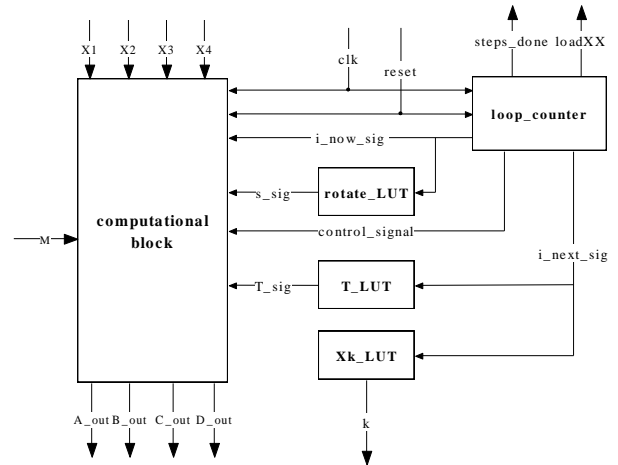


Figure 6. The main_loop_core component.

VII. PERFORMANCE EVALUATION

Design simulation performed using the Modelsim environment. In order to provide all the essential simulations for the verification of the MD5 Core, the appropriate *test_bench* in VHDL was created. The previous design was synthesized, placed and routed on the *2v250fg256* Xilinx Virtex II FPGA device using the Leonardo Spectrum environment. The following chart shows the utilization of the above device resources. The clock frequency achieved using this device was 84 MHz.

SOURCES	USED	AVAILABLE	UTILIZATION
IOs	170	172	98.84%
Function Generators	1341	3072	43.65%
CLB Slices	671	1536	43.68%
Dffs or Latches	666	3588	18.56%

According to the performance measurement from the software implementation given in rfc-1810, the maximum throughput was less than 100Mbps. The MD5 implementation presented herein, since there was no delay for loading except for the first block, exhibited the following performance:

$$\frac{512 \text{ bit}}{65 \text{ Hz}} = 7,87 \text{ bit/Hz} = 7,87 \text{ Mbit/MHz} ,$$

which gives **661,1 Mbps** throughput at 84 MHz.

VIII. MULTIPLE CORES FOR IMPROVING PERFORMANCE

MD5 algorithm is a block-chained hashing algorithm, which implies that the hashing result of one block initializes the round procedure of the next block. As a result, blocks neither can be hashed in parallel, nor a pipelining technique can be applied. On the other hand, many applications require the hashing of small messages.

To improve the performance further in such situations, many cores for processing different messages in parallel can be used. For example, in cryptanalytic attacks, such as password cracking, the MD5 core should be used in the arrangement shown in Figure 7. Cryptanalytic attacks on hash functions are a method to examine the security supposing provided. To perform these attacks, great computation speed is needed. This performance can be obtained through FPGA hardware implementations of hash functions.

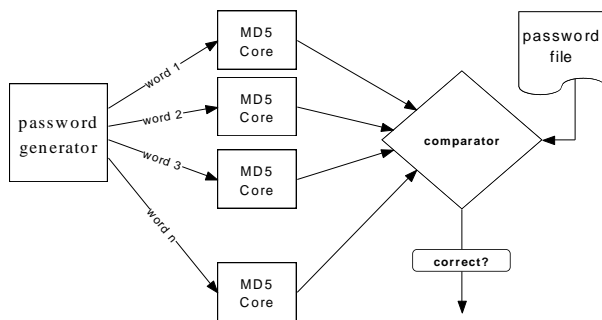


Figure 7. A password cracker consisting of several MD5 Cores.

IX. CONCLUSIONS

The significance of the hardware implementation of the MD5 algorithm has been investigated. FPGA devices provide performance combined with low cost and flexibility. This makes them suitable for high performance cryptographic and cryptanalytic application, thus offering a great advantage when compared to software or other hardware implementations, such as ASICs. Although MD5 algorithm does not allow hashing of different blocks in parallel, such hardware implementations can be used for the parallel hashing of different messages. Hence, a significant throughput can be obtained.

REFERENCES

- [1] Brown S. and Rose J., FPGA and CPLD Architectures: A Tutorial, IEEE Design & Test of Computers, vol. 13, no. 2, pp. 42-57, 1996.
- [2] Deepakumara J., Heys H.M. and Venkatesan R., FPGA Implementation of MD5 Hash Algorithm, Canadian Conference

on Electrical and Computer Engineering, 2001, vol. 2, pp. 919-924, 2001.

- [3] Dobbertin Hans, Cryptanalysis of MD5 Compress, German Information Security Agency, May 2 1996.
- [4] Rivest R., The MD5 Message-Digest Algorithm, RFC 1321, MIT LCS & RSA Data Security, Inc., April 1992.
- [5] Schneier Bruce, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edition, John Wiley and Sons, 1996.
- [6] Sjöholm Stefan and Lindh Lennart, VHDL for designers, Prentice Hall, 1997.
- [7] Touch J., Report on MD5 Performance, RFC 1810, June 1995.
- [8] Vanstone S.A., Menezes A.J., Oorschot P.C., Handbook of Applied Cryptography, CRC Press, 1997.