# FPGA IMPLEMENTATION OF MD5 HASH ALGORITHM

*Janaka Deepakumara, Howard M. Heys and R. Venkatesan*
*Faculty of Engineering and Applied Science*
*Memorial University of Newfoundland*
*St.John's, NF, Canada A1B 3X5*
Email:{janaka,howard,venky}@engr.mun.ca

**ABSTRACT**

*In information security, message authentication is an essential technique to verify that received messages come from the alleged source and have not been altered. A key element of authentication schemes is the use of a message authentication code (MAC). One technique to produce a MAC is based on using a hash function and is referred to as an HMAC. Message Digest 5 (MD5) is one of the algorithms, which has been specified for use in Internet Protocol Security (IPSEC), as the basis for an HMAC. The input message may be arbitrarily large and is processed in 512-bit blocks by executing 64 steps involving the manipulation of 128-bit blocks. There is an increasing interest in high-speed cryptographic accelerators for IPSEC applications such as Virtual Private Networks. As we shall show in the paper, it is reasonable to construct cryptographic accelerators using hardware implementations of HMACs based on a hash algorithm such as MD5. Two different architectures, iterative and full loop unrolling, of MD5 have been implemented using Field Programmable Gate Arrays (FPGAs). The performance of these implementations is discussed.*

## 1. INTRODUCTION

Data integrity assurance and data origin authentication are essential security services in financial transactions, electronic commerce, electronic mail, software distribution, data storage and so on. The broadest definition of authentication within computing systems encompasses identity verification, message origin authentication and message content authentication. In IPSEC, the technique of cryptographic hash functions is utilized to achieve these security services.

### 1.1 Hash Functions

Hash functions compress a string of arbitrary length to a string of fixed length. They provide a unique relationship between the input and the hash value and hence replace the authenticity of a large amount of information (message) by the authenticity of a much smaller hash value (authenticator)[1]. In recent years there has been an increased interest in developing a Message Authentication Code (MAC) derived from a hash code. Among the many reasons behind this are that cryptographic hash functions such as MD5 and SHA-1 generally execute faster in software than symmetric block ciphers such as DES. The software for hash functions is widely available and there are no export restrictions from the United States or other countries for cryptographic hash functions. Hence, there are many applications of MD5, SHA-1 and other hash functions to generate MACs. The method to implement the MAC for IP security has been chosen as hash-based MAC or HMAC, which uses an existing hash function in conjunction with a secret key. The HMAC algorithm is specified for an arbitrary FIPS-approved cryptographic hash function. With minor modification, HMAC can easily replace one hash function with another [2].

### 1.2 Message Digest 5 (MD5) Algorithm

MD5 [3] is a message digest algorithm developed by Ron Rivest at MIT. It is basically a secure version 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. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit message digest of the input. This is mainly intended for digital signature applications where a large file must be compressed in a secure manner before being encrypted with a private (secret) key under a public key cryptosystem.

Assume we have an arbitrarily large message as input and that we wish to find its message digest. The processing involves the following steps.

(1) Padding
The message is padded to ensure that its length in bits plus 64 is divisible by 512. That is, its length is congruent to 448 modulo 512. Padding is always performed even if the length of the message is already congruent to 448 modulo 512. Padding consists of a single 1-bit followed by the necessary number of 0-bits.

(2) Appending length
A 64-bit binary representation of the original length of the message is concatenated to the result of step (1). (Least significant byte first). The expanded message at this level will exactly be a multiple of 512-bits. Let the expanded message be represented as a sequence of $L$ 512-bit blocks $Y_0$, $Y_1, \ldots, Y_q, \ldots, Y_{L-1}$ as shown in Figure 1 [4]. Note that in the figure, IV and CV represent initial value and chaining variable respectively.

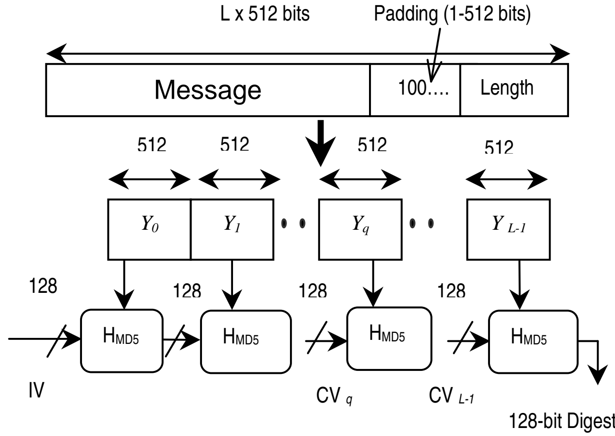


Figure 1. Generation of message digest

(3) Initialize the MD buffer
The variables IV and CV are represented by a four–word buffer (ABCD) used to compute the message digest. Here each A, B, C, D is a 32-bit register and they are initialized as IV to the following values in hexadecimal. Low-order bytes are put first.

Word A: 01 23 45 67
Word B: 89 AB CD EF
Word C: FE DC BA 98
Word D: 76 54 32 10

(4) Process message in 16-word blocks
This is the heart of the algorithm, which includes four "rounds" of processing. It is represented by $H_{MD5}$ in Figure 1 and its logic is given in Figure 2. The four rounds have similar structure but each uses different auxiliary functions *F, G, H* and *I.*

$$F(X,Y,Z) = (X \wedge Y) \vee (\overline{X} \wedge Y)$$

$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \overline{Z})$$

$$H(X,Y,Z) = X \oplus Y \oplus Z$$

$$I(X,Y,Z) = Y \oplus (X \vee \overline{Z})$$

where $\vee$, $\wedge$, $\oplus$ and $^{-}$ represent the logical OR, AND, XOR and NOT operations, respectively. Each round consists of 16 steps and each step uses a 64-element table $T$ [1 ... 64] constructed from the sine function. Let $T[i]$ denote the $i$-th element of the table, which is equal to the integer part of $2^{32}$ times abs(sin($i$)), where $i$ is in radians. Each round also takes as input the current 512-bit block ($Y_q$) and the 128-bit chaining variable ($CV_q$). An array $X$ of 32-bit words holds the current 512-bit $Y_q$. For the first round the words are used in their original order. The following permutations of the words are defined for rounds 2 through 4:

$$\rho_2(i) = (1 + 5i) \bmod 16$$
$$\rho_3(i) = (5 + 3i) \bmod 16$$
$$\rho_4(i) = 7i \bmod 16$$

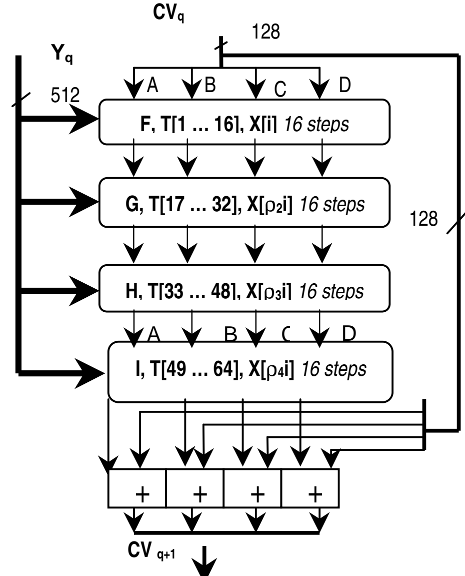


Figure 2. Compression function $H_{MD5}$

The output of the fourth round is added to the input of the first round ($CV_q$) to produce $CV_{q+1}$.

(5) Output
After all $L$ 512-bit blocks have been processed, the output from $L^{th}$ stage is the 128-bit message digest.
Figure 3 shows the operations involved in a single step. The additions are modulo $2^{32}$. Four different circular shift amounts (S) are used each round and are different from round to round. Each step is of the following form [4]:

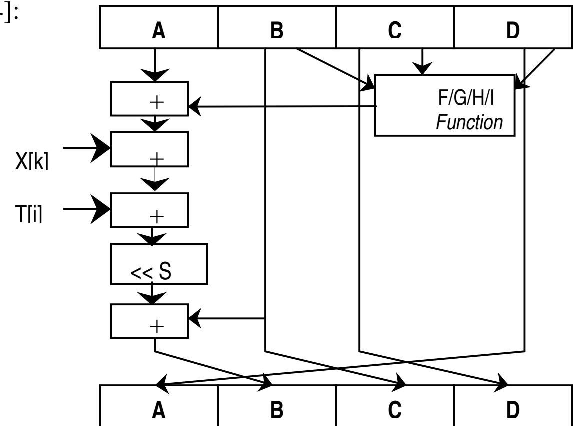


Figure 3. Operations in a single step of MD5

$$A \leftarrow D$$

$$B \leftarrow B + ((A + Func(B, C, D) + X[K] + T[I]) << s)$$

$$C \leftarrow B$$

$$D \leftarrow C$$

### 1.3 FPGA Implementation

Re-configurable devices such as FPGAs are a highly attractive option for hardware implementations as they provide the flexibility of dynamic system evolution as well as the ability to easily implement a broad range of algorithms.

Most hash functions are targeted at software implementations. The advantages of software implementations are ease of use, ease of upgrading, portability and flexibility. However a hardware implementation has more physical security by nature, as it can not easily be modified by an attacker. On the other hand the speed of a software implementation is restricted to the speed of the computing platform and there are vulnerabilities for viruses and other complications due to system failures.

The main features of hash functions are the relatively easy computations making both software and hardware implementations practical. FPGAs offer many advantages over Application Specific Integrated Circuits (ASICs). Short time to market, high flexibility including capability for frequent modifications of hardware, low development cost and low cost of the final product are some. FPGAs have the potential for fast, low cost, reprogramming and experimental testing of a large number of various architectures and revised versions of the same architecture [5]. For this implementation the target device was selected as the Xilinx Virtex FPGA family. The Virtex FPGA delivers high performance, high speed, and high capacity programmable logic solutions. The abundance of routing resources permits the Virtex family to accommodate large and complex designs. The Virtex gate array comprises some main configurable elements: Configurable Logic Blocks (CLBs), Input /Output Blocks (IOBs), look-up tables and block select RAMs. Each CLB contains four logic cells organized in two similar slices [6]. The top-level design was described in VHDL and the available Xilinx core generator modules were utilized wherever applicable. Xilinx Alliance 3.1i and Foundation 3.1i tools were used for synthesizing and implementation. VSS and Foundation EDIF simulators were used for functional and timing simulations.

## 2. MD5 IMPLEMENTATION

MD5 algorithm is a block-chained hashing algorithm. The hash for a block depends on both the block data and the hash of its preceding block. As a result, blocks can not be hashed in parallel. Each step consists of four additions, three component logical operations, two table lookups and one rotation. The tree of operations can be optimized by performing operations, which involve items not dependent on the previous step, early. According to Figure 3, the item that depends on the previous step is word B and hence the result of logical operation has a considerable delay. The optimized tree of operation (assuming each operation takes one unit time) will be as given in Figure 4. According to this one time unit step can be reduced [7].

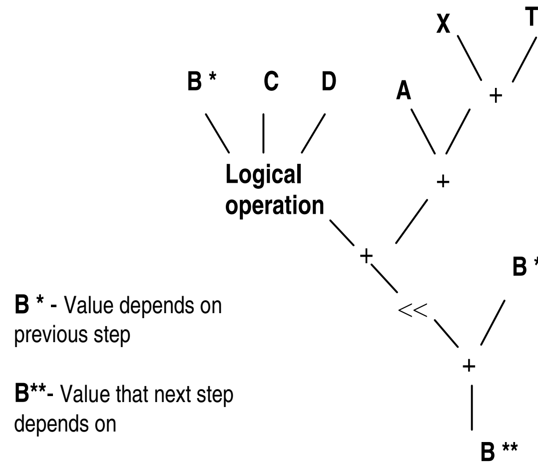


Figure 4. Optimized operation tree

The following architectural options were investigated and implemented:

- iterative looping (Iterate_MD5)
- full loop unrolling (Fullun_MD5)

Both architectures were implemented at behavioral level in VHDL, simulated, synthesized and functionally simulated. After verifying the functionality, the design underwent the translation, mapping, placing and routing (PAR), timing and configuration stages of the flow engine. The functionality of the PAR implementations is then re-simulated with back-annotated timing using the same test vectors used in functional simulation, verifying that the implementation of the design is successful. In both designs, it is assumed that the first two aspects of the algorithm have already been performed and the input of message blocks can be controlled according to the state machine states.

### 2.1 Iterative Looping Architecture

.

By implementing a generic step of the MD5 algorithm, a looping architecture with 64 iterations would seem to provide the greatest area optimized solution. The block diagram of the iterative design is shown in Figure 5.

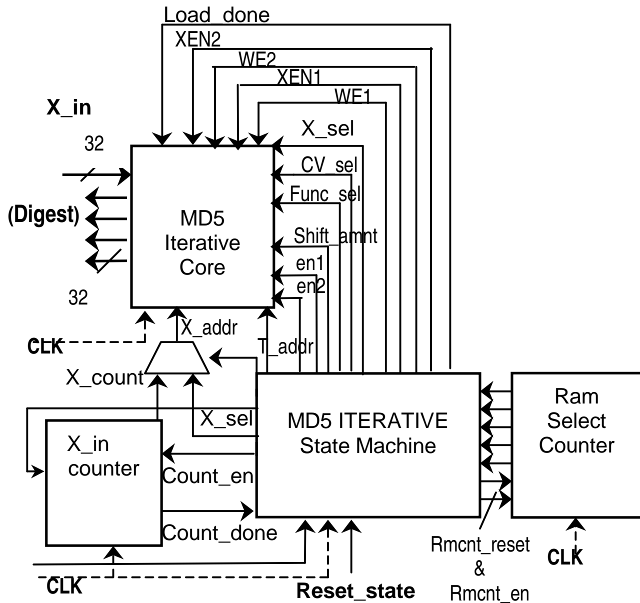


Figure 5. Block diagram of MD5 iterative design (Iterate_MD5).

A few additional multiplexers and a barrel shifter have to be used to perform the selection of the round function and the variable shifting in each round. The state machine has 68 states including three states required for initializing and loading the very first block to the core. In subsequent block operations the state machine utilizes 65 states. The main feature of this design is the loading of message blocks in parallel with computation. The two RAMs can be utilized to load the next block while the present block is being used in computation. This eliminates the loading time. The 512-bit message block is loaded to the core using a 32-bit bus. The "Reset_state" signal initiates the state machine and the counters. Then with the "Start" signal the function gets started. The initial vectors are loaded in parallel to the input register and to a buffer. The initial vectors as well as the chaining variables are kept in this buffer until the 64$^{th}$ step to get added with the last result to form the chaining variable for the next block. Initially the first block is loaded to the XRAM1 using the addresses given by the X_in counter. After that the state machine starts to provide addresses for reading of XRAM1. Using the first 16 addresses provided by the state machine, the next block is written to XRAM2. After the 64$^{th}$ step, XRAM2 is read. During the first 16 steps of processing the second block, the third block is written to XRAM1. This reading and writing of RAMs alternates in every 64 clock cycles. Subsequent blocks utilize the previous chaining variable as their initial values. The generic step is shown in Figure 6.

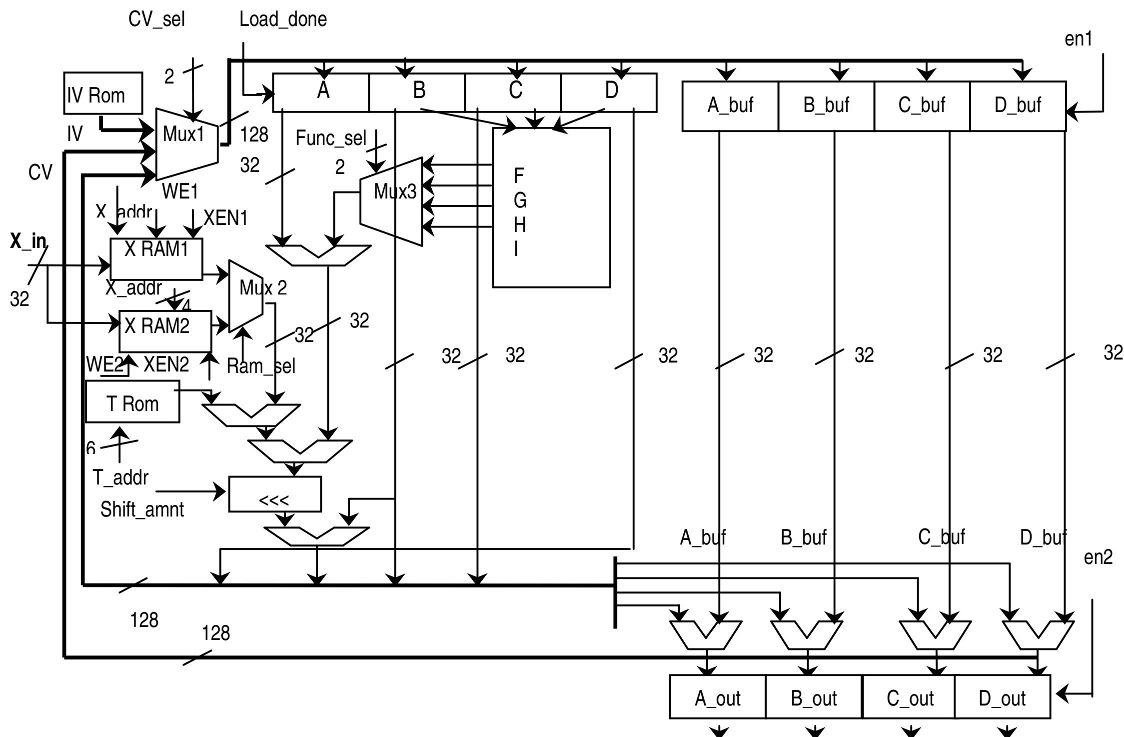


Figure 6. MD5 iterative core

### 2.2 Full Loop Unrolling Architecture

The full loop unrolled architecture has a 64-step combinational logic core as shown in Figure 7. In this architecture all the elements of each step are implemented as combinational logic. The barrel shifter has been removed by direct wiring of appropriate shifted bits in each step.

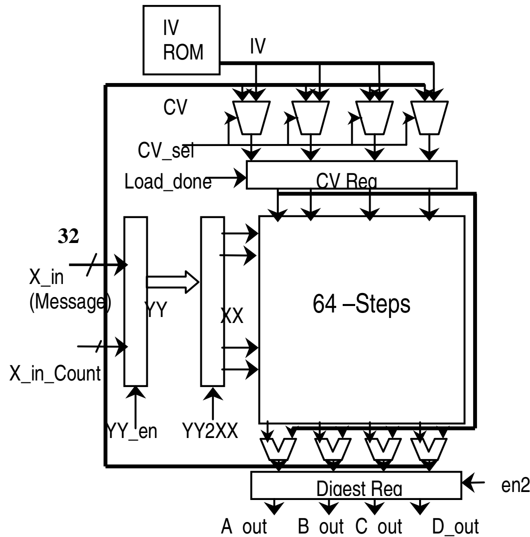


Figure 7 MD5 full loop-unrolling core

.

The use of double buffering (XX and YY) eliminates the loading time from the critical timing path. The next block is loaded during the computation of the present block. IV ROM provides the initialization vector for the first step. The "load_done" signal makes the initialization vector and the chaining variables available for the first block and for the subsequent blocks respectively. During computation of the digest for a block, the next block is stored in buffer YY and after the computation the "YY2XX" signal gets high and hence XX obtains the new input for the next computation. The block diagram of the complete design is given in Figure 8. In addition to the core, the other main components are the state machine which has four states, X_in counter used for loading the blocks to the core and Wait counter utilized to count the number of cycles for the combinational logic delay of the computation.

Similar to the iterative design, the "Reset_State" signal initiates the state machine and the X_in counter. The initialization vectors are taken into the register CV_Reg. With the "Start" signal, the initial block is loaded to buffer YY and right after that "YY2XX" signal loads it to buffer XX and the computation is commenced. During computation, the next block is loaded to buffer YY. When all the blocks in the message are processed, "en2" signal makes the digest available at the output of register, Digest_Reg.

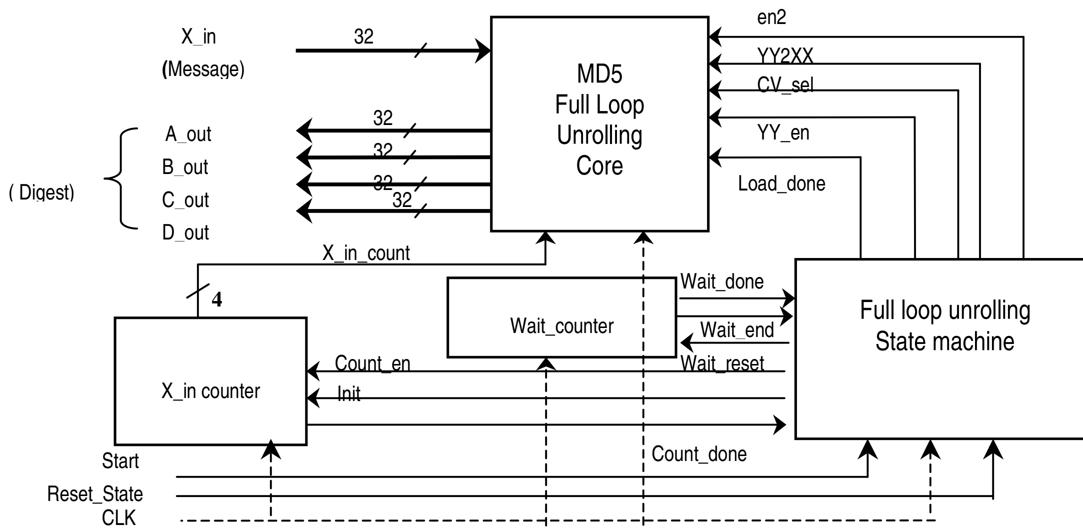


Figure 8. Block diagram of full-loop-unrolling design (Fullun-MD5).

## 3. PERFORMANCE EVALUATION

Both designs were synthesized and placed and routed on the Virtex V1000FG680–6 target device with clock rate up to 200 MHz.

In the case of the iterative design, the utilization of the external IOBs was 161 out of 512 (31%) and the block RAM usage was 2 out of 32 (6%). The number of slices used for this architecture was significantly low. It was 880 out of 12288 (7%) and from this the barrel shifter utilized 288 (2%). There is 4% utilization of three state buffers (TBUFs). According to the timing simulation the maximum frequency of the design was 21 MHz. Hence, the expected throughput is (512 x 21M)/65 = 165 Mbps.

For the full-loop-unrolled design, the utilization of slices was 4763 out of 12288 (38%). The utilization of external IOBs was similar to that of the iterative design. The number of TBUFs has been reduced to 2%. Timing simulations show that the maximum delay for a computation of a chaining variable is 1444.75 ns. The controller can run at 71.4 MHz. Since there is no delay for loading except for the first block, the expected throughput is (512)/(1.445 μs)= 354 Mbps

The summary is given in Table 1.

| Architecture | % Slices utilization | Frequency | Throughput |
|---|---|---|---|
| Iterative | 6 % | 21 MHz | 165 Mbit/s |
| Full loop unrolling | 38% | 71.4 MHz | 354 Mbit/s |

Table 1.

According to the performance measurements on software implementations given in [7], the throughput has been less than 100 Mbps. DEC Alpha (190 MHz) has given a throughput of 87-100 Mbps.

## 3. CONCLUSION

The significance of the hardware implementation of the MD5 algorithm has been examined. Two architectures have been studied for both area utilization and speed with FPGAs as the target device. It is clear that both architectures can be easily fitted to a single device. Although the inherent nature of the MD5 structure does not allow parallel hash operations of blocks, hardware implementations can obtain a significant throughput to cater to some of currently available IP bandwidths. FPGA implementations would therefore be suitable as components in cryptographic accelerators.

The device utilization of iterative design is significantly small. The unused resources can be utilized to implement several cores in the same device and thereby processing several messages in parallel. This would be an attractive feature for a cryptographic accelerator. Although the utilization was fairly high, two full loop-unrolling designs could be fitted into a single FPGA device. Hence there is a possibility of processing two messages in parallel. As well, for both architectures there is a possibility of implementing the complete HMAC by implementing other necessary HMAC components, utilizing the unused resources of the FPGA.

The obtained results can be further improved by using the latest FPGA devices such as Virtex II family. The Virtex devices provide better performance than the previous generation of FPGAs achieving synchronous system clock rates of 200 MHz [6]. The latest devices however can provide more than 400 MHz clock speeds as well as more resources. Further, using timing constraints it is likely that the delays in the critical paths can be reduced.

## REFERENCES


[1]. B. Preneel, "Cryptographic Primitives for Information Authentication- State of the Art in Applied Cryptography", Lecture Notes in Computer Science vol. 1528, Springer-Verlag Berlin Heidelberg NY 1998.

[2]. National Institute of Standards and Technology, *The Keyed-Hash Message Authentication Code (HMAC)*, Federal Information Processing Standards Publication # HMAC, 2001.

[3]. R. Rivest, *The MD5 Message-Digest Algorithm,* RFC 1321, MIT LCS & RSA Data Security, Inc., April 1992.

[4]. W. Stallings, *Cryptography and Network Security*, Second edition. Prentice Hall, 1997.

[5]. K. Gaj and P. Chodowiec, "Comparison of the Hardware Performance of the AES Candidates Using Configurable Hardware", http://csrc.nist.gov/encryption/aes/round2/.

[6]. Xilinx Inc., Virtex 2.5V *Field Programmable Gate Arrays*, 2000.

[7]. J. Touch, *Report on MD5 Performance*, RFC 1810, June 1995.