Hardware Design of Standard Hash Algorithm HAS-160

Choong-Mo Youn and Beom-Geun Lee, *Member, KIMICS*

Abstract—This paper is about the hardware implementation of the Hash algorithm, HAS-160, which is widely used for Internet security and authentication. VHDL modeling was used for its realization and the operation speed has been increased by the optimized scheduling of the operations required for step operations.

Index Terms—HAS-160, Hash algorithm

I. INTRODUCTION

Innovative advances in communication technology have resulted in many changes in our lives. Internet banking, electronic commercial trading, cyber stock trading and such similar things have become universal activities and hence, important transactions involving private information and money are conducted online. In addition, those types of transactions will also be ported out on personal mobile devices/communications such as personal digital assistants (PDA), cellular phones, and IMT 2000.

To implement these services, not only advancements in transmission media but also in encryption technology and digital signature technologies are required. However, in the face of constant security attacks that undermine existing computer systems and encryption technologies, higher level of encryption technologies are continuously required. But this results in higher costs and requires higher operation speeds for the devices to comply with the higher technological complexity. To cope with these difficulties, hardware-based encryption algorithm should be implemented[6][7].

Hash algorithm is used for the purpose of verifying data integrity, authentication, non-repudiation, and reuse prevention in Internet-based transactions. This is an indispensable component in Internet-based financial transactions, electronic commercial trade, and peer-to-peer (P2P) services. It has been implemented in software in various communication devices and security-related platforms but has never been implemented in hardware. In this paper, HAS-160, the standard Korean hash algorithm, is modeled in VHDL and implemented in FPGA to verify its performance.

Manuscript received October 10, 2005.

Choong-Mo Yun is with the School of Information Technology, Seoil College, Seoul, 131-701, Korea (Tel: +82-2-490-7408, Fax: +82-2-490-4783, Email: 5477choong@hanmail.net)

Beom-Geun Lee is with the Department of Electronic Engineering, Kyunghee University

II. HAS-160 Algorithm

HAS-160 algorithm is an algorithm for generating final 160-bit output data by processing the input data of arbitrary length to block data whose size is 512 bits[1]. The hash algorithm should satisfy the following conditions.

1) It is impossible to get an input value to the given output value by computation.
2) It is impossible, for a given input, to find another input which generates the same output as the given input.
3) It is impossible to find two different input messages which generate the same output by computation.

In this algorithm, the whole input with 512-bit blocks is entered, and fills the remaining with "0" so that the length of the last block is 448 bits. The last 64 bits should be filled by computing the message length before filling the remaining with "0." Figure 1 shows the structure of the HAS-160[2].

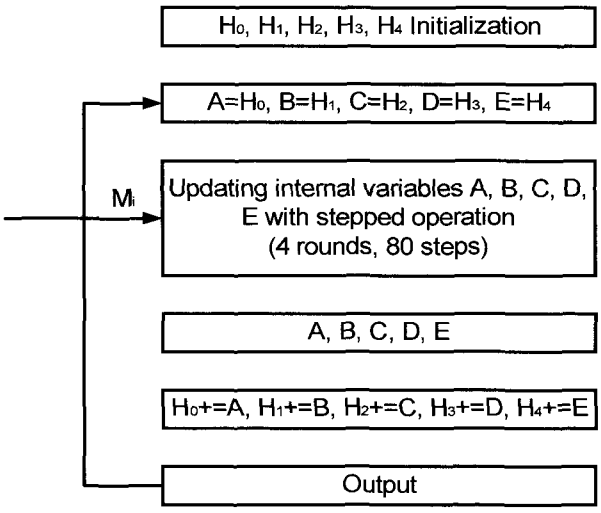

Fig. 1 Structure of HAS-160

The chain variable, H_n, used in HAS-160 is initialized such that H_0 = 67452301, H_1 = efcdab89, H_2 = 98badcfe, H_3 = 10325476 and H_4 = c3d2e1f0. The step operation is performed through 4 rounds and 80 steps and the message variable, M_i, is processed during the operation. The message variable, M_i, transforms the 512-bit messages to 16 32-bit messages according to the little-endian convention. From the 16 words, the 4 message variables, X[16], X[17], X[18], X[19] are generated. The generated words are transformed to XOR values of message variables

used in 0–4, 6–9, 11–14, and 16–19 step operations in each round. The value of the message variables used in each step operation is shown in Table 1.

Table 1. Message variables used in step operations

i	Round 1	Round 2	Round 3	Round 4
0	X[18]	X[18]	X[18]	X[18]
1	X[0]	X[3]	X[12]	X[7]
2	X[1]	X[6]	X[5]	X[2]
3	X[2]	X[9]	X[14]	X[13]
4	X[3]	X[12]	X[7]	X[8]
5	X[19]	X[19]	X[19]	X[19]
6	X[4]	X[15]	X[0]	X[3]
7	X[5]	X[2]	X[9]	X[14]
8	X[6]	X[5]	X[2]	X[9]
9	X[7]	X[8]	X[11]	X[4]
10	X[16]	X[16]	X[16]	X[16]
11	X[8]	X[11]	X[4]	X[15]
12	X[9]	X[14]	X[13]	X[10]
13	X[10]	X[1]	X[6]	X[5]
14	X[11]	X[4]	X[15]	X[0]
15	X[17]	X[17]	X[17]	X[17]
16	X[12]	X[7]	X[8]	X[11]
17	X[13]	X[10]	X[1]	X[6]
18	X[14]	X[13]	X[10]	X[1]
19	X[15]	X[0]	X[3]	X[12]

Figure 2 shows the structure of the step operation.

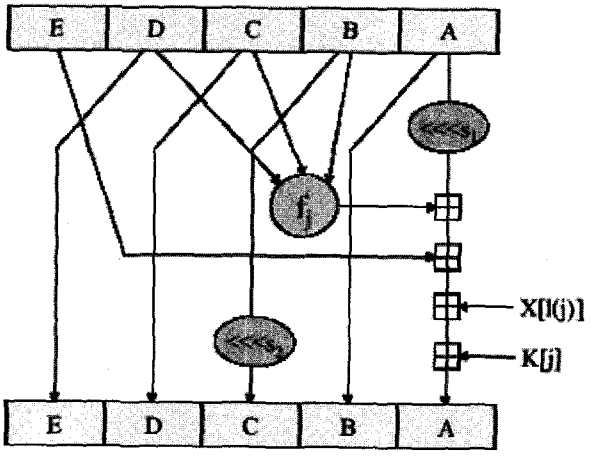

Fig. 2 Step operation

In Figure 2, S_1 and S_2 are the circular shifts used in the $j_{th}(1\leq j\leq 80)$ step operation and the amount of the circular shift is redefined in each of the 4 rounds and 80 steps. The circular shift of the S_1 is: $S_1(j) = 5, 11, 7, 15, 6, 13, 8, 14, 7, 12, 9, 11, 8, 15, 6, 12, 9, 14, 5, 13$ ($0\leq j\leq 19$, $20\leq j\leq 39$, $40\leq j\leq 59$, $60\leq j\leq 79$). And that of S_2 is: $S_2(j)=10$ ($0\leq j\leq 19$), $S_2(j)=17$ ($20\leq j\leq 39$), $S_2(j)=25$ ($40\leq j\leq 59$) and $S_2(j)=30$ ($60\leq j\leq 79$). f_j of Figure 2 is the Bool function used in each step operation.

After the 4-round and 80-step operations for a 512-bit data block are finished, the chain variables H_0–H_4 are updated such that the A, B, C, D, E values are added to the variables: H_0+=A, H_1+=B, H_2+=C, H_3+=D, H_4+=E.

III. IMPLEMENTATION

Figure 3 shows the entire block diagram of the HAS-160.

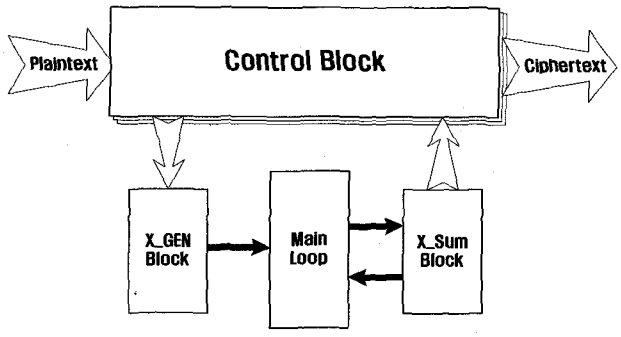

Fig. 3 System block diagram

The system is composed of four functional blocks, and the control block that gets the data input is used to control the current state and transfers the data from the external to the X_Gen block. It outputs the computed hash value to the external and controls the whole system's operations and input/output.

In the X_Gen block, the new data block is generated from the data inputted from the control block according to 4 rounds and 80 steps and is transferred to the Main_Loop block.

Figure 4 shows the X_Gen block.

In the X_Gen block, there are 16 registers for storing the data inputted from the external and 4 registers for storing the data generated in each round from them.

Figure 5 shows the Min_Loop block.

Three clocks are required to perform the stepped operation in the Main_Loop block. First, the step operation is initiated after initializing A, B, C, D, E by getting the initializing constants from the X_Sum block in Figure 3 And the message variables required for each step is inputted from the X_Gen block. In the Main_Loop block processing step operation, 4 32-bit addition operation, one logical operation, and 2-bit cyclic operation. At this time, the maximum-delay path is identified. The scheduling is done by distributing the clock used for each operation to minimize the maximum-delay path.

End_State_Data value is outputted in the X_Sum block from the 512-bit message through 80 steps. If the input data is composed of many 512-bit messages, the 80 steps are processed again by getting the initialization constant from the X_Sum block. The initialization constant at this time is the cumulatively operated value from the previous 512 bits.

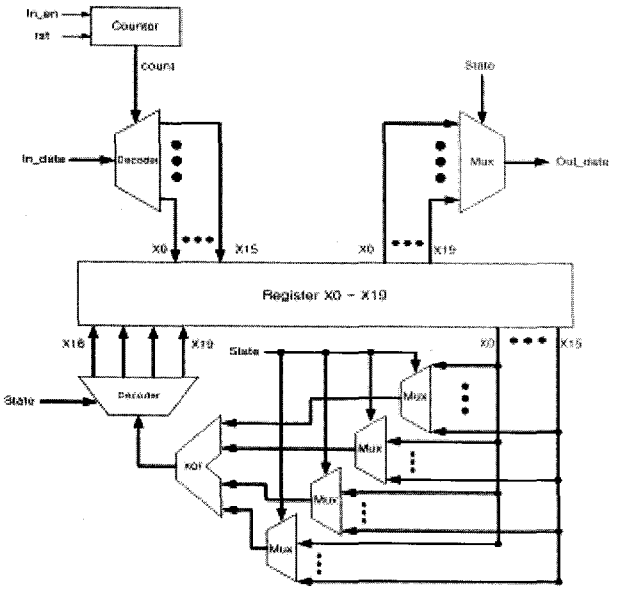

Fig. 4 Detailed structure of the X_Gen Block

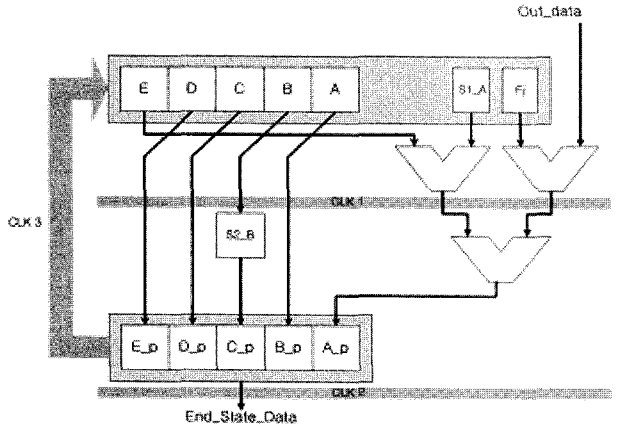

Fig. 5 Detailed structure of the Main_Loop Block

The X-Sum block generates the initialization constant Reset_Data and transfers the Setup_Data to the Main_Loop block. Here, the initialized values are the hexadecimal constants in the HAS-160 hash algorithm: H0 = 67452301, H1 = efcdab89, H2 = 98badcfe, H3 = 10325476 and H4 = c3d2e1f0

Figure 6 shows the detailed structure of the X_Sum block.

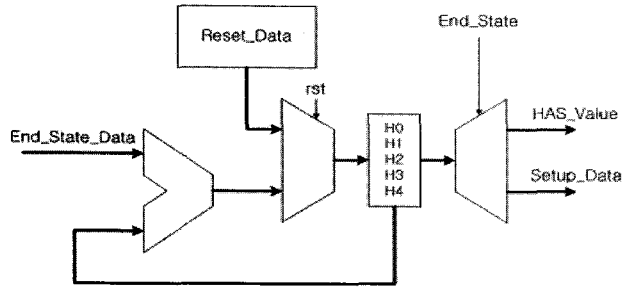

Fig. 6 Detailed structure of the X_Sum Block

160-bit data are generated by cumulatively operating the End_State_Data that is the output of the Main_Loop block through 80-step operations. The generated hash value, HAS_Value is transferred to the control block. If there is an additional 512-bit data block, the Main_Loop block is initialized again with the previous output result of the 80 operations.

If there is no more message, End_State signal is generated from the control block and the final hash value, HAS_Value is transferrd to the control block. The transferred hash value is outputted through a 32-bit bus by the control block.

Figure 7 is the whole structure of the composition diagram.

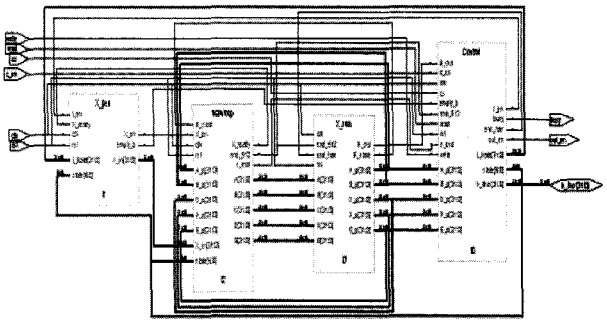

Fig. 7 Composition diagram of the Top Block

The composition operation is performed in the APEX-II device in Altera and the resulting processing speed is 110 MHz. The detailed composite result is in Table 2.

Table 2. Detailed Composition result

Total LUTs	2494 of 16640 (14%)
Logic resources	3007 ATOMs of 16640 (18%)

16640 mean the number of total LUT of APEX-II device in Table 2. 2494 mean the number of used LUT of APEX-II device in Table 2.

Three clocks are required for each step operation and a total of 240 clocks are required for the stepped operation since 80 steps are needed to process 512 bits.

If the input message is 512 bits, a total of 2.345μs is needed for clocks for stepped operations, final addition, and generation of message variables [16], X[17], X[18], X[19], which corresponds to a processing speed of 218.3 Mbps.

Table 3 shows the speed of the hash algorithm according to the implementation system.

Table 3. Performance according to MD5 system

MD5 execution capability		Speed
Software	Intel Pentium 90 NeXTStep[3]	44 Mbps
	Sun SPARC-20/71 Sun OS 4.1.3[3]	57 Mbps
Hardware	Xilinx Virtex[4]	165 Mbps
	ALTERA APEX20k[5]	208 Mbps

HAS-160 has a more complicated structure and processes more steps compared with that of the MD5 having 4 rounds and 64 steps. However, the performance is 218.3 Mbps which is better than that of the hardware realization.

IV. CONCLUSION

In this paper, the Korean standard hash algorithm, HAS-10, is modeled with VHDL and verified with FPGA. The processing of addition, logical operation, and bit cyclic operation required for step operations is modeled separately and composed later. The step operations have the structure of performing operations by the internal 3-step processing.

REFERENCES

[1] TTAS.KO-12.0011/R1 (2000): Hash function Algorithm Standard (HAS-160)
[2] Douglas R. Stinson, *Cryptography Theory and Practice,* CRC, 1995.
[3] J. Touch, "Report on MD5 performance" RFC 1810, June 1995.
[4] Janaka Deepakumara, Howard M. Heys and R. Venkatesan, "FPGA IMPLEMENTATION OF MD5 HASH ALGORITHM" *Technical Paper*, Engineering and Applied Science Memorial University of Newfoundland St. John Canada, 2001.
[5] Warwick Ford, *Computer Communications Security*, Prentice–Hall International Inc, 1994.
[6] Man Young Rhee, *Cryptography and Secure Communications*, McGraw–Hill, 1994.
[7] Bruce Schneier, *Applied Cryptography,* John Wiley & Sons Inc. 1996.

Choong-Mo Youn
He received the M.S degree in Electronic Engineering from Dankook University in 1990 and Ph.D degree in Electronic Engineering for Chongju University in 2000. From 1992 to present, he is a professor, School of Information Technology Seoil College in Korea. His research interests include Digital system and CAD, Network.

Beom-Geun Lee
He received the M.S degree in Electronic Engineering from Chongju University in 1997. Since 1997 to now, he has been Ph.D course student in Electronic Engineering, Kyunghee University Suwon, Korea. His research interests include Computer system and security.