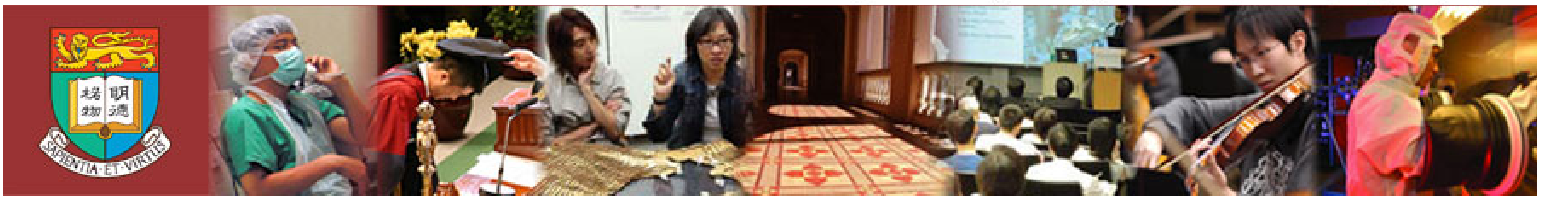

A UNIFIED ARCHITECTURE OF MD5 AND RIPEMD-160 HASH ALGORITHMS

Chiu-Wah Ng, Tung-Sang Ng and Kun-Wah Yip

Department of Electrical & Electronic Engineering, The University of Hong Kong
Pokfulam Road, Hong Kong
Email: {cwng, tsng, kwyip}@eee.hku.hk

ABSTRACT

Hash algorithms are important components in many cryptographic applications and security protocol suites. In this paper, a unified architecture for MD5 and RIPEMD-160 hash algorithms is developed. These two algorithms are different in speed and security level. Therefore, a unified hardware design allows applications to switch from one algorithm to another based on different requirements. The architecture has been implemented using Altera's EPF10K50SBC356-1, providing a throughput over 200Mbits/s for MD5 and 80Mbits/s for RIPEMD-160 when operated at 26.66MHz with a resource utilization of 1964LC.

I. INTRODUCTION

A hash function computes a fixed length output called the message digest from an input message of various lengths. The MD5 message digest algorithm [1], developed by Ron Rivest at MIT, accepts a message input of various lengths and produces a 128-bit hash code. It has been one of the most widely-used hash algorithms. However, it has been indicated in [2] that there is a security threat in the algorithm. Furthermore, a 128-bit hash output may not offer sufficient security protection in the near future. To provide higher security protection, RIPEMD-160 has been proposed by Dobbertin, Bosselaers and Preneel [2]. RIPEMD-160 accepts the same input format as that of MD5, and produces a 160-bit output.

It is easy to find that the structures of the two algorithms are quite similar. It follows that they can be combined together to give one hardware design that can perform the two hash functions. This approach has the following advantages. Firstly, the unified design is a resource-efficient implementation when different hash algorithms are needed in applications. Applications can switch to either algorithm based on different requirements. Second, since MD5 is still the most widely-used hash algorithm, upgrading the current implementation in the future to RIPEMD-160 is much easier with a unified hardware architecture.

Motivated by these observations, in this paper we develop a unified architecture for MD5 and RIPEMD-160. Comparison with other unified architectures indicates that the proposed architecture is area-efficient.

II. REVIEW OF HASH ALGORITHMS

MD5 and RIPEMD-160 share the same flow of operation. First, the input message is padded and divided into data blocks of length 512 bits. Each data block is treated as 16 32-bit words. The algorithms iteratively processes each data block. For the first data block, an initial value is used to compute an intermediate result. The intermediate result, called the chaining variable, is then updated according to the input data block and the previous result. After all iterations are done, the final chaining variable is the hash value.

A. MD5

Figure 1 shows the basic operation of MD5 [3]. The algorithm consists of 4 main rounds with each round applying the basic operation 16 times. In the figure, $F(B,C,D)$ is a nonlinear bit-wise function of 3 inputs. There are 4 different functions for the 4 rounds. The index i represents each step, and $X[i]$ represents one message word. The orders of the 16 words for processing are different for different rounds. In the figure, $K[i]$ is a 32-bit constant chosen from a fixed table containing 64 constants stated in the specification [1] and $S[i]$ is a 5-bit input that controls the left rotation of the input sum. After a total of 64 basic operations, the chaining variable is updated by directly adding the 4 variables A, B, C and D with the current content in the chaining variable.

B. RIPEMD-160

Figure 2 shows the basic operation of RIPEMD-160. RIPEMD-160 consists of 5 main rounds with each round applying the basic operation 16 times. There are two parallel lines in each step, and five different non-linear functions $F(B,C,D)$ corresponding to the 5 rounds. In the figure, $X[i]$ is the input word, $K[i]$ is one of the ten 32-bit constants in the algorithm, $S[i]$ corresponds to the 4-bit control for the left rotation operation, and Rol^{10} denotes a 10-bit left cyclic shift operation. After a total of 80 basic operations, the chaining variable is updated by adding the left five variables A, B, C, D, E, the five right variables A', B', C', D', E', and the current content in the chaining variable.

III. UNIFIED ARCHITECTURE DESIGN, AND ITS FGPA IMPLEMENTATION

It is apparent from Section 2 that MD5 and RIPEMD-160 share many common parts, but there are also certain differences, as listed below.

1. The input order for the word $X[i]$ and the number of shifts $S[i]$ are different.
2. There are 64 constants for $K[i]$ in MD5 but there are only 10 constants in RIPEMD-160.
3. RIPEMD-160 has an extra variable E (E') for the 160-bit output.
4. The nonlinear functions F of the two algorithms are different.
5. The MD5 requires a 5-bit cyclic shifter while RIPEMD-160 needs a 4-bit one.
6. There is an extra cyclic shift operation (Rol^{10}) in RIPEMD-160.

In order to unify the two algorithms for FPGA implementation, we modify the basic hardware structure of MD5 (Figure 1) with the following modifications and additions.

1. Two separate look-up tables (LUTs) are used to store the constants and numbers of shifts.
2. All the input orders are stored in a LUT to index the message words in different steps.
3. The 128-bit chaining variable is expanded to 160 bits.
4. A 10-bit cyclic shifter, which is required in RIPEMD-160, is added.
5. Two additional nonlinear functions are added. [Four and five nonlinear functions are required for MD5 and RIPEMD-160, respectively. Out of these nine functions, only six are distinct.]
6. Multiplexers are added.

The resultant unified architecture is shown in Figure 3.

The design has been modeled using VHDL and implemented in FPGA. The target device is an Altera EPF10K50SBC356-1. The aim is to obtain a resource-efficient implementation. Therefore, only one of the two lines of RIPEMD-160 is implemented in the FPGA, resulting in a saving of nearly half of the logic elements. This arrangement is used as a trade-off between area and speed. Compact hardware implementation can be obtained by the following approach.

Modern FPGAs have numerous memory resources besides logic elements. The hash algorithms take 512-bits data as the input. A dual-port RAM in the FPGA can be used to store this large amount of data to allow both external and internal access of data. Memory elements configured as ROM are used to store the constants, numbers of shifts and word-input orders. For constants lookup, a ROM with 6-bit input address and 32-bit output is used for the MD5 algorithm as it specifies that 64 32-bits words and a ROM with 4-bit input address is required for storing the 10 constants of RIPEMD-160. For the shift-number lookup, since there are 160 steps in RIPEMD-160, a ROM with 8-bit address lines must be used. These constants fill up the lower 196 words of the ROM, thus leaving the upper 64 words for the variable shift in MD5. This arrangement allows simple decoding logic to select the appropriate constants for the 2 algorithms. In order to accommodate the larger shift specified in the MD5 algorithm, a ROM with 5-bit output is used. For the message selection sequence, the same approach is used and a ROM with 8-bit input address and 4-bit output is used. As LUTs are used in the aforementioned manner, a counter can be used to index the appropriate values during each step. It simplifies the control logic design. A compact implementation thus results. Table 1 summarizes the memory resource utilization.

Since memory lookup is slow and in this case the message words are accessed by first accessing the message selection table and then the dual-port RAM, the execution time is increased. To speed up, the execution of the basic operation is pipelined into two stages. First, an operand fetch is performed. The appropriate constants are looked up in memory. Then an execution phase for the actual operation is performed. Figure 4 shows an example of the execution of MD5.

With additional circuits for initialization and output data processing, the final hardware design is shown in Figure 5.

IV. PERFORMANCE EVALUATION AND COMPARISON

The design has been synthesized, placed and routed on Altera's EPF10K50SBC356-1 using Altera's Max+PlusII 10.0. The maximum frequency is 26.66 MHz. The number of logic cells used is 1964 (68%) and that of memory bits is 5376 (13%). The throughput of the hash algorithms can be calculated as follows:

Throughput (Mbits/s) = 512 x freq (MHz) / no. of clk cycles.

The throughput performance is shown in Table 2. Several implementations of hash algorithms can be found in the literature. However, each of them uses different FPGA devices and hence a comparison of the performance is difficult. As a rough comparison, Table 3 and Table 4 summarize the results. In [4], a universal hardware module is designed for the MD4 family, including MD5, RIPEMD-160, SHA-1 and SHA-256. Although this module can combine 4 different algorithms, it requires multiple clock cycles to execute a step. In the design presented herein, each step can be executed in a single clock cycle. In addition, the module of [4] requires significantly more resources (1004 CLBs) when compared with the proposed design. In [5], only MD5 is implemented. We can compare the performance of our design with the performance results reported in [5]. In [6], the implementation combines MD5, SHA-1 and HAS-160 and thus requires significant resources. It is apparent from Table 4 that the unified architecture requires only slightly more resources than the single MD5 implementation but with a comparable performance.

Table 1. Memory utilization of the proposed design.

MD5 constants	64 x 32 =	2048 bits
RIPEMD-160 constants	16 x 32 =	512 bits
Shift amount	256 x 5 =	1280 bits
Message selection	256 x 4 =	1024 bits
Dual port ram		512 bits
	Totals:	5376 bits

Table 2. Performance of the proposed design.

	MD5	RIPEMD-160
Clock cycles	66	162
Throughput	206Mbits/s	84Mbits/s

V. CONCLUSION

In this paper, a unified architecture for MD5 and RIPEMD-160 has been designed and implemented in FPGA. It has been shown that the implementation is resource-efficient and preserves the single clock-cycle structure. With the present FPGA implementation, the hardware design is suitable for applications that require low to medium throughput such as Ethernet networks.

REFERENCES

[1] R. Rivest, “The MD5 Message-Digest Algorithm,” *RFC 1321*, Apr. 1992.

[2] H. Dobbertin, A. Bosselaers and B. Preneel, “RIPEMD-160: A Strengthened Version of RIPEMD, Fast Software Encryption,” *LNCS 1039*, pp. 71-92, Springer-Verlag, 1996.

[3] W. Stallings, *Cryptography and Network Security, 2nd ed.*, Now York: Prentice-Hall, 1997.

[4] S. Dominikus, “A hardware implementation of MD4-family hash algorithms,” *Proc. 9th Int. Conf. on Electronics, Circuits and Systems*, vol. 3, pp. 1143-1146, 2002.

[5] J. Deepakumara, H. M. Heys and R. Venkatesan, “FPGA implementation of MD5 hash algorithm,” *Proc. 2001 Canadian Conf. on Electrical and Computer Engineering*, vol. 2, pp. 919-924, 2001.

[6] Y. K. Kang, D. W. Kim, T. W. Kwon and J. R. Choi, “An efficient implementation of hash function processor for IPSEC,” *Proc. 2002 IEEE Asia-Pacific Conf. on ASIC*, pp. 93-96, 6-8 Aug. 2002.

Table 3. Performance comparison of different designs.

Algo.	This design		[4]		[5]		[6]	
	Cyc.	Thru.	Cyc.	Thru.	Cyc.	Thru.	Cyc.	Thru.
MD5	66	206	206	107	65	165	65	142
RIPEMD-160	162	84	337	65	N/A	N/A	N/A	N/A

Table 4. Comparison of resource utilization.

	This design	[4]	[5]	[6]
FPGA Vendor	Altera	Xilinx	Xilinx	Altera
Resource util.	1964 LC	1004 CLB	880 slices	10573 LE

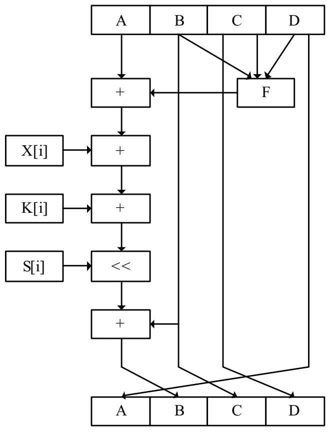

Figure 1. Basic MD5 operation.

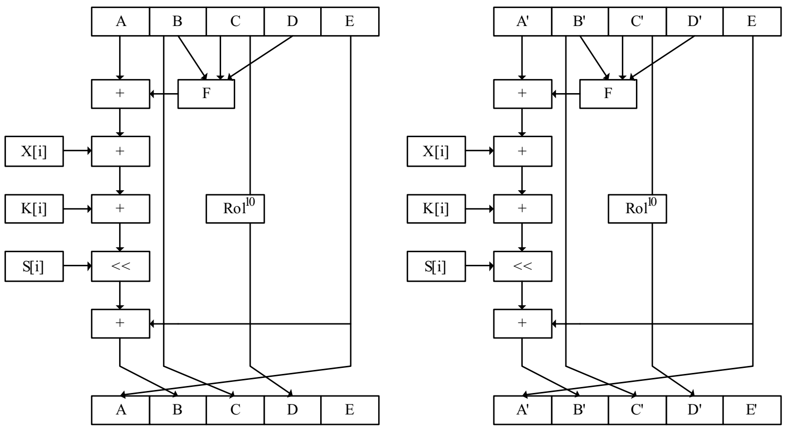

Figure 2. Basic RIPEMD-160 operation.

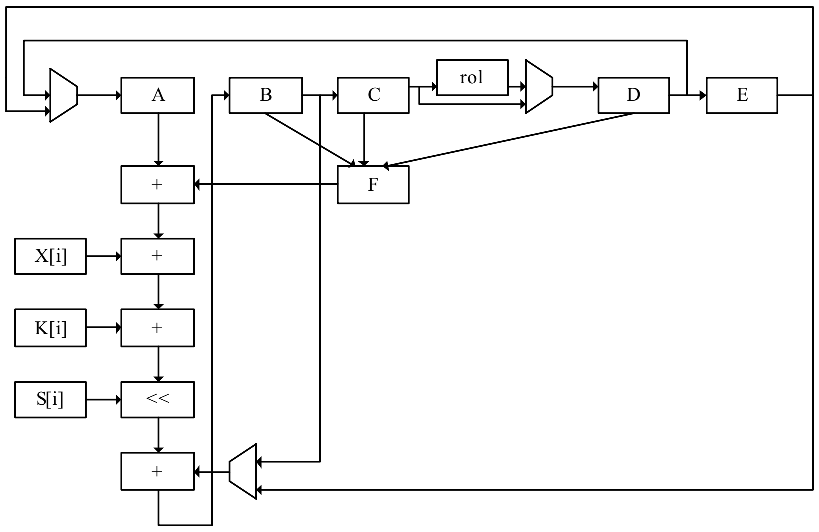

Figure 3. Basic operation of the unified architecture.

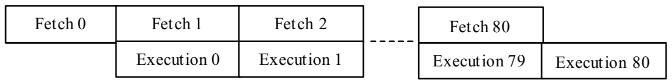

Figure 4. Execution of the MD5 algorithm: an example.

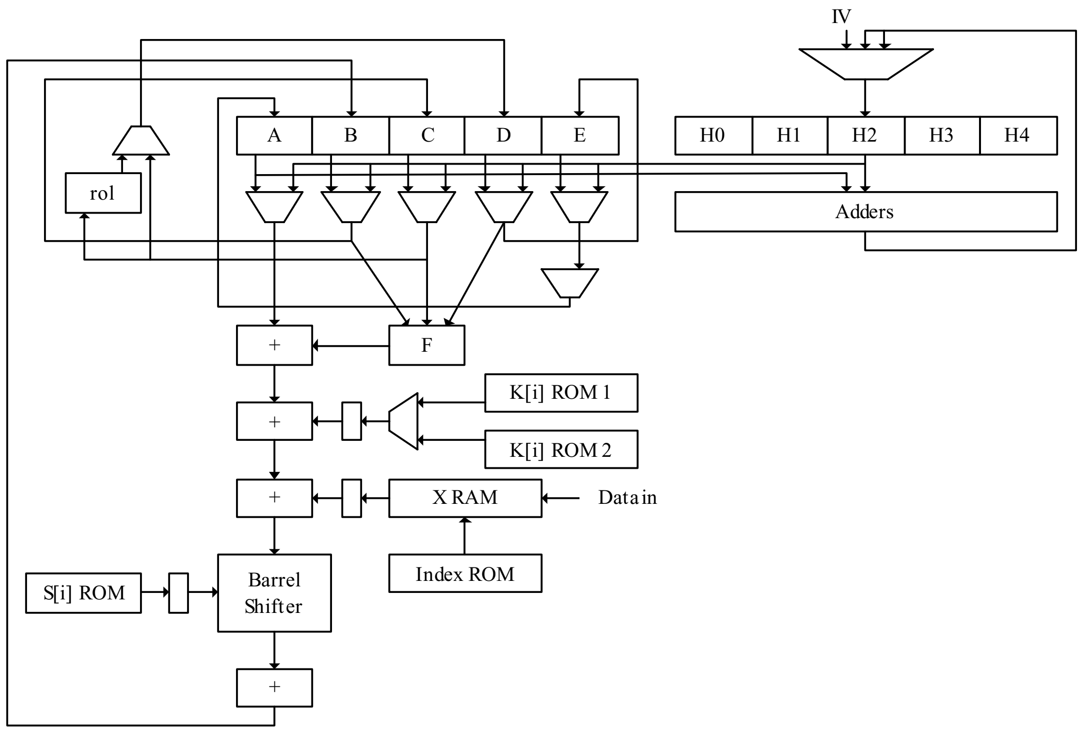

Figure 5. Hardware design of the unified architecture.