VLSI Implementation of High-speed SHA-256

Ling Bai *, Shuguo Li

Abstract — *To accelerate the speed of iterative computation for the existing SHA-256 algorithm, using 7-3-2 array compressor is proposed to reduce the critical path delay in this paper. The frequency of the proposed scheme is 1.7 times higher than other VLSI implementations under the same process. In addition, the paper designs a new universal architecture for implementing SHA-2 algorithms. The design is synthesized with the slow library of Synopsys Design Compiler in SMIC 0.18 μm CMOS process. Its function has been verified sufficiently on FPGA. Furthermore, compared with the existing SHA-256 core, the results of the ASIC synthesis and FPGA verification are more preferable. This design is convenient to implant into SoC and embedded system with the versatile architecture and high clock frequency*[1].

Index Terms — SHA-256, 7-3-2 compressor, critical path, FPGA

1. Introduction

Hash functions are used to create digital signatures, which are also known as "cryptographic hash". The functions are divided into two families: the MD families and SHA families. SHA family includes SHA-1 and SHA-2. Because much more broken hazard exists in MD5 and SHA-1, SHA-2 is developed by the National Institute and Technology in 2002. And the government of the United States has announced that SHA-2(including SHA-256, 384, 512) will be applied roundly step by step before the year 2010.

As a kind of Hash function, Security Hash Algorithm-256(SHA-256) can be applied to many aspects, such as digital signature, checking the integrity of data, and identity validation. SHA-256 generates a 256-bit message digest for a message whose length is shorter than 2^{64} bits.

For software realization of SHA-256 takes too much CPU source, VLSI implementation has been considered, as well as FPGA. The canonical scheme of SHA-256 algorithm reaches the operating frequency no more than 100MHz by the CMOS 0.18um process. To solve the problem, this paper describes an improved scheme, which focuses on decreasing the delay on the critical data path, increases the speed of SHA-256 nearly twice using the same process. This advanced SHA-256 algorithm IP core has wide applications, such as high speed data integrity assurance and data authentication security.

[1] This work was supported by the National Natural Science foundation of China (No.60476015), national High-Tech Research and Development Program of China (No.2006AA01Z418), and Ministry Commission Project of China (No. 20054500224).

Ling Bai* is now with the Institute of Microelectronics, Tsinghua University, Beijing, China. (bai-l07@mails.tsinghua.edu.cn)

Shuguo Li is now with the Institute of Microelectronics, Tsinghua University, Beijing, China. (lisg@tsinghua.edu.cn).

978-1-4244-3870-9/09/$25.00 ©2009 IEEE

This paper is organized as follows. The typical SHA-256 algorithm is introduced in section 2. The critical path improvement is given in section 3, which contains amendatory scheme analysis. In section 4, the performance comparison result of the proposed scheme is shown based on Synopsys Design Compiler and FPGA. Meanwhile, the simplified FPGA verification flow will be given. Section 5 makes a conclusion of the whole work.

2. Normative SHA-256 Algorithm Flow

SHA-256 transforms an input message into the 256 bits digest. The process is non-reversible theoretically. According to Secure Hash Signature Standard (Refer to [1], SHS FIPS PUB 180-2), the input message whose length is shorter than 2^{64} bits, should be operated by 512 bits in group and becomes a 256-bit message digest. The algorithm is summarized in 5 steps as follows.

Step 1: Padding. Before the hash computation begins, padding the original message with '1000...000' sequence and the length of the message. And the length is expressed by 64-bit. The padded message's length is a multiple of *512* bits, which decides how many '0' to be padded.

Step 2: Parsing. The padded message is parsed into N 512-bit blocks: $M^{(1)}, M^{(2)}, \cdots M^{(N)}$. Each block can be divided into *16* 32-bit words: $M_0^{(i)}, M_1^{(i)}, \cdots M_{15}^{(i)}$, which are then expanded into 64 words labeled $W_0, W_1, \cdots W_{63}$ under the certain rule prescribed by SHA-2 standard. Refer to the detail in Appendix.

Step 3: Initialize the 256-bit buffer *H0*, *H1*$\cdots$*H7* (32-bit each) to store the intermediate and final results. If the algorithm core is processing blocks of the same message, the buffer is initialized by the intermediate hash value. (Refer to Appendix)

Step 4: The algorithm is implemented by 64-cycle iterative computation each block. The eight working variables are labeled *A, B, C...H*, which are updating the value during the 64-cycle as follows.

$$T1_i = H_i + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + K_i + W_i$$

$$T2_i = \Sigma_0(A_i) + Maj(A_i, B_i, C_i)$$

$$H_{i+1} = G_i,\quad G_{i+1} = F_i,\quad F_{i+1} = E_i$$

$$E_{i+1} = D_i + T1_i,\quad D_{i+1} = C_i,\quad C_{i+1} = B_i,\quad B_{i+1} = A_i$$

$$A_{i+1} = T1_i + T2_i$$

Note: K_i is the constant. The functions including $\sum_0, \sum_1$, *Ch, Maj*, are given by Appendix, as well as the constant K_i.

Step 5: Compute the 256-bit buffer and the intermediate results by the 8 working variables above. The *i*th intermediate Hash value is computed as follows.

$$H_0^i = A + H_0^{i-1} \quad H_1^i = B + H_1^{i-1} \quad H_2^i = C + H_2^{i-1} \quad H_3^i = D + H_3^{i-1}$$
$$H_4^i = E + H_4^{i-1} \quad H_5^i = F + H_5^{i-1} \quad H_6^i = G + H_6^{i-1} \quad H_7^i = H + H_7^{i-1}$$

After processing the whole N blocks, the final 256-bit result is the message digest as below.

$$H_0^N \| H_1^N \| H_2^N \| H_3^N \| H_4^N \| H_5^N \| H_6^N \| H_7^N$$

The updating function process is illustrated in Fig. 1.

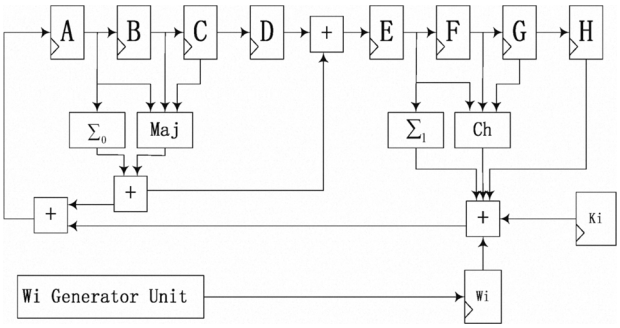

Fig. 1. Canonical SHA-256 Algorithm Scheme

Computation of Wi is revealed in Fig. 2. M_{i+1} represents the next 32-bit input message.

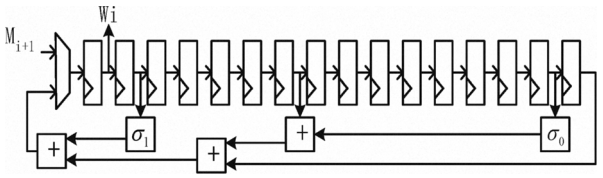

Fig. 2. Wi Generator Unit

5 steps discussed above can be summed up to be 2 parts, which contain message preprocessing and Hash function computation. The previous is done by software which is not discussed explicitly here. The paper will focus on analyzing the process that calculates the Hash sequence and the hardware improvement on speed.

3. VLSI Implementation and Improved Scheme

3.1 Architecture of SHA-256 IP core

Considering about the convenience of SoC application, define the width of input data to be 32, as well as the output data. Since the padded message is the multiple of 512, SHA256 IP core will receive the input data in multi-cycle. Set a control signal register to decide when to receive the message block, and when to export the Hash value. Setting the feedback signal active represents the core is ready to receive the next block, for the previous block has been completed.

Fig. 3 shows the main modules of SHA-256 architecture and data paths.

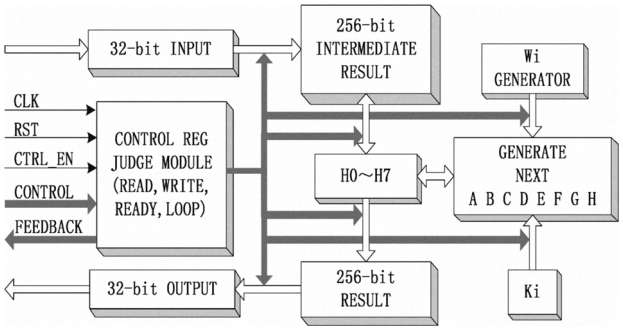

Fig. 3. Architecture of SHA-2 IP Core

Control register in Fig. 3 involves 4 signals that are READ, WRITE, READY and LOOP. When the signal CTRL_EN is active, external control signals are written to the internal control register. CTRL_EN is determined by FEEDBACK. FEEDBACK expresses whether the internal operation is done. Table **I** shows the function of each signal of CONTROL REG.

Table I
Control register function definition

	Active	Inactive
WRITE	Write message block	No operation
READ	Read Hash value	No operation
LOOP	Internal 64 operation loops	New message
READY	Ready to receive new block	No operation

Both of the typical scheme and the improved scheme are based on the architecture above, which is expedient to verify the IP and transplant to other SHA-2 family algorithms, such as SHA-384 and SHA-512. Furthermore, with regular data width of 32 bits and control signal register, this architecture can be applied to SoC and embedded systems.

3.2 Improved scheme by 7-3-2 compressor

Analyzing the process of Hash sequence generation in Fig. 1, it is obvious that the critical data path is the computation of next A. In Fig. 4, the blocks in shadow indicate the logic units on the critical data path.

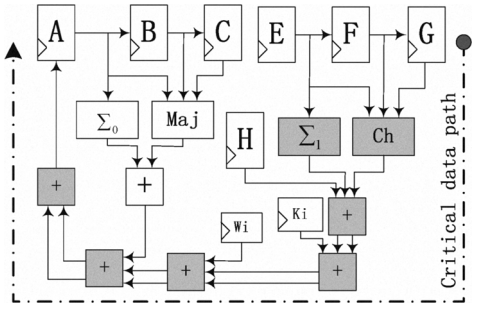

Fig. 4. Critical Data Path

There are 5 FAA (Full Adder Array) and some other functions (*Ch*, $\sum_1$) on the critical path. Logic functions *Ch* and $\sum_1$ are the combination of XOR and shift operations, whose delay can be ignored comparing with 32-bit adder. So

it's just need to think about to improve adders to achieve the higher clock frequency. Obtain the expression of next A from Section 2:

$$T1_i = H_i + \Sigma_1(E_i) + Ch(E_i, F_i, G_i), + K_i + \mathrm{W_i}$$

$$T2_i = \Sigma_0(A_i) + Maj(A_i, B_i, C_i)$$

$$A_{i+1} = T1_i + T2_i$$

Expand equations above, A_{i+1} is the sum of 7 numerical values. That is:

$$A_{i+1} = \Sigma_0(A_i) + Maj(A_i, B_i, C_i) + H_i + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + K_i + W_i \quad (1)$$

Consider to improve the structure of adder with 7 operands to reach a higher speed. The keystone is reducing the delay of carry as much as possible. Using pipeline structure and introduce new variables to reduce the critical path delay are normal schemes to raise frequency, which consumes too much area. Compromising between the delay and area, this paper adopts a new structure of adder called 7-3-2 compressor. The structure is depicted in Fig. 5.

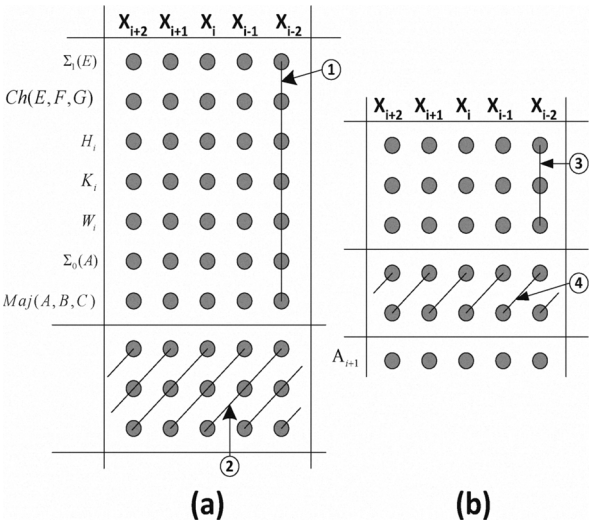

Fig. 5. 7-3-2 Compressor

Take the middle 5 of 32 bits for example. The leftmost column in Fig. 5(a) represents the 7 operands mentioned in (1). Bias labeled② contains 3 bits, which are full-adder results of Column X_{i-2} labeled①. Deducing the rest 31 columns by analogy, 7-row operands are compressed to 3-row. Fig. 5(b) denotes the rest work to get the final result. Construe column X_{i-2}: 2-bit of Bias④ is the sum of 3-bit in Column X_{i-2} labeled ③. The 3 rows are compressed to be 2 rows sequentially. A_{i+1} will be attained by adding the last 2 operands. The longest carry propagation occurs at the last operation, which need to add 2 32-bit operands. The advantage of 7-3-2 compressor is that the computation of 32 columns is parallel. The operation in 7-3 and 3-2 compressor is just to add several single bits, whose delay is much smaller than full adder with the 32-bit operands. The new critical path is shown in Fig. 6.

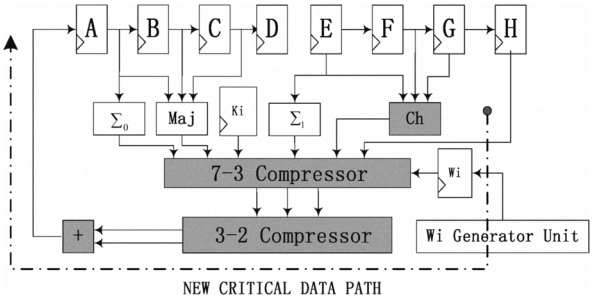

Fig. 6. New Critical Path of Improved Scheme

Analyzing the critical data path of Fig. 6, the modules in shadow is the main block of the critical path. Omitting the logic function delay, the delay is equal to: delay(FAA) + delay(7-3 adder) + delay(3-2 adder), which is almost half of the delay in Fig 4.

4. Results of ASIC Synthesis and FPGA Verification

4.1 ASIC Synthesis Result

The described scheme has been implemented in Verilog using Cadence NC-sim and synthesized using the slow library of Synopsys Design Complier based on SMIC 0.18um process. The performance is better than other ASIC implementation under the same process and area constraint without pipeline. The maximum frequency, area, throughput are listed in Table **II**.

Table II
Synthesis Comparison

	Frequency (MHz)	Area (μm^2)	Throughput (Mbps)
SHA-256[3]	120	178710.84	472
This work	208	211954.88	819

Due to every 256-bit Hash value is the result of 65 cycles iterative calculation (64 loops + 1 initializing cycle):

$$Throughput = \frac{256}{65cycles} = \frac{256}{65 \times T_{clock}} = \frac{256}{65} \times f_{clock} = 3.94 f_{clock} bps$$

Obviously, synthesized by the same process library, the scheme proposed in the paper can operate at higher speed than [3], as well as throughput. 7-3-2 compressor scheme raises the clock frequency by 73.3%.

4.2 FPGA Verification

FPGA is reconfigurable and flexible, which is a natural choice for implementation of Hash algorithm in a broad range of applications with different area-performance requirements. The ASIC design is advantageous to be verified on FPGA. The proposed scheme is validated on the target device of Altera Cyclone III(EP3C25U240C8).

Before performing the FPGA verification, we generate more test vectors by C++ code besides the message given by SHA-2 standard in order to heighten the test integrity. Design a test module for the sha-256 core and download the module with the IP core to the FPGA target device by Quartus II 7.2.

Table **III** gives the FPGA verification results compared with other works.

Table III
FPGA Verification Comparison

	Frequency (MHz)	LE	Throughput (Mbps)
SHA-256[2]	42	7879	336
SHA-256 [3]	50	2227	196
SHA-256 [4]	74	2207	291
This work	83	2633	327

7-3-2 compressor scheme of this work has the advantage in speed on FPGA. It increases clock frequency by about 13% than [4] and 48% than [3]. In addition, with the similar throughput, this work consumes only *1/3* logic elements of [2].

4.3 Comparison Summary

This work operates a comparable throughput just with 1/3 LE comparing with [2]. Normative scheme proposed in [3] and [4] is the most common implementation scheme of SHA-256, which takes small area and can be designed easily. But it's obvious that proposed design in this paper, 7-3-2 compressor scheme without much more consumed area, can achieve the highest clock frequency comparatively.

5. Conclusion

Using 7-3-2 compressor to speed up the calculation of SHA-256 is presented in the paper. The delay of the critical data path is only 4.8ns. Furthermore, the new versatile architecture proposed can be totally applied for SHA-2 family (SHA-384, SHA-512), which just need to increase the data width and change the logical functions. This advanced SHA-256 algorithm IP core has been verified sufficiently on FPGA with test vectors generated by Microsoft Visual C++, besides the message examples given by FIPS PUB 180-2 Standard.

Appendix

1. Initialized Hash value.
   In Step 3 of section 2, initialize the 256-bit buffer as follows.
   A=0x6A09E667, B=0xBB67AE85, C=0x3C6EF372,
   D=0xA54FF53A, E=0x510E527F, F=0x9B05688C,
   G=0x1F83D9AB, H=0x5BE0CD19
2. The rule of expanding the padded message.
   Record the padded message as M_i, and the expanding message as W_i. Expanding operation is listed in (2)

$$\begin{cases} W_i = M_i, & 0 \le i \le 15; \\ W_i = \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16}, & 16 \le i \le 79; \end{cases} \quad (2)$$

There into:

$$\sigma_1(x) = RS^{17}(x) \oplus RS^{19}(x) \oplus SHR^{10}(x)$$

$$\sigma_0(x) = RS^{7}(x) \oplus RS^{18}(x) \oplus SHR^{3}(x)$$

RS is the rotate right (circular right shift) operation, and *SHR* represents the right shift operation.

3. Logic function description.

Summarize all the function used in SHA-256 algorithm in Table **IV**.

Table IV
Logic Function

Designation	Function
Maj(x,y,z)	$(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
Ch(x,y,z)	$(x \wedge y) \oplus (\bar{x} \wedge z)$
RS	Circular right shift
SHR	Logical right shift
$\Sigma_0(x)$	$RS^2(x) \oplus RS^{13}(x) \oplus RS^{22}(x)$
$\Sigma_1(x)$	$RS^6(x) \oplus RS^{11}(x) \oplus RS^{25}(x)$
$\sigma_1(x)$	$RS^{17}(x) \oplus RS^{19}(x) \oplus SHR^{10}(x)$
$\sigma_0(x)$	$RS^7(x) \oplus RS^{18}(x) \oplus SHR^3(x)$

4. Constant K_i is given by [1].

References

[1] SHA-2 Standard Specification. National Institute of Standards and Technology (NIST), Secure Hash Standard, FIPS PUB 180-2. www.itl.nist.gov/fipspubs/fip180-2.htm, (2002).

[2] Imtiaz Ahmad. Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGA. Computers and Electrical Engineering 31, p.345(2005).

[3] Marsgod. Secure Hash Algorithm. www.opencores.org, 2004.

[4] S.Dominikus. A Hardware Implementation of MD4 Family Hash Algorithms. Proc of IEEE International Conference on Electronics Circuits and Systems (ICECS'02), Vol.III, p.1143(2002).

[5] Wanzhong Sun, Hongpeng Guo. Design and Optimized Implementation of the SHA-2(256, 384, 512) Hash Algorithms. International Conference On ASIC Electronic Version Proceedings, 2007.

[6] Xiaohui Yang, Zibin Dai. Implementation of SHA-256 Based on FPGA. Microcomputer Information, 2006, 22(4-2):146-148.

[7] Zhun Huang. Efficient hardware architecture for secure hash algorithm SHA-1. Journal of Tsinghua University(Science and Technology), 2005, Vol 45, No.1.

[8] Chao Li, Yuechuan Wei, Bin Sun. Structure Security of Compress Function of SHA-256. Journal of Applied Sciences-Electronics and Information Engineering, 2008, 26(1): 1-5.

[9] Michail H, Kakarountas A.P. A low-power and high-throughput implementation of the SHA-1 hash function. Circuits and Systems, ISCAS 2005. IEEE International Symposium on, Page(s): 4086 - 4089 Vol. 4(2005).

[10] Luigi Dadda, Marco Macchetti, Jeff Owen. An ASIC Design for a High Speed Implementation of the Hash Function SHA-256(384, 512). Proceedings of the 14th ACM Great Lakes symposium on VLSI, p.421(2004).