A High-Throughput Hardware Implementation of SHA-256 Algorithm

Yimeng Chen
Institute of Microelectronics
Tsinghua University, Beijing, China
Email: cym18@mails.tsinghua.edu.cn

Shuguo Li
Institute of Microelectronics
Tsinghua University, Beijing, China
Email: lisg@tsinghua.edu.cn

***Abstract*—The SHA-256 algorithm is widely used in the field of security. In this paper, we propose a rescheduling method for the SHA-256 round computation. Based on the proposed rescheduling, we propose a design for SHA-256, in which the critical path is reduced. Our design is implemented on the Xilinx Virtex-4 FPGA. It achieves the throughput of 1984 Mbps with the area of 979 slices. Compared with other designs on FPGA, our design shows a better performance in terms of the throughput.**

***Index Terms*—SHA-256, Round Computation, Rescheduling, FPGA**

I. Introduction

Hash functions can compress a message into a string of fixed length, which is called the message digest. The SHA-2 hash functions are issued by the National Institute of Standards and Technology (NIST) in 2002. All algorithms of the SHA-2 family are iterative [1]. In this paper, we focus on SHA-256, one representative algorithm of the SHA-2 family.

The SHA-256 algorithm is widely used in the field of security, such as block chain and message authentication code (MAC). The high-throughput hardware implementation of SHA-256 is in high demand. There are several techniques that are widely used in the published work:

- Unrolled architecture [4], [7], [8]. It is widely used in the implementations of cryptographic algorithms that are iterative. This architecture improves the throughput but consumes more area.
- Parallel counters [2]. The round computation of SHA-256 can be regarded as the addition of multiple operands. So the parallel counters can be used to reduce the area cost and the critical path.
- Pipeline [2]–[6]. Although the pipeline structure can improve the throughput, paper [4] emphasizes that registers cannot be added at will in the design for SHA-256, due to the feedback.
- Delay balancing [2]. This technique is used to balance the delays of different paths.

In this paper, we propose a rescheduling method for the SHA-256 round computation. Based on our rescheduling, we propose a design for SHA-256, in which some intermediate values required in the next round can be precalculated in the current round. Then the critical path is reduced. This paper is organized as follows. Section II describes the background of the SHA-256 algorithm. Section III presents our design for SHA-256. Section IV shows the implementation results of our design on FPGA. Finally, Section V concludes this paper.

II. SHA-256 Algorithm

The SHA-256 algorithm consists of two stages: preprocessing and hash computation [1].

A. Preprocessing

The first step of preprocessing is message padding, which ensures the length of the padded message is a multiple of 512 bits. Suppose that the message length is l bits before padding. The bit "1" and k zero bits are appended to the end of the message. According to the rules in [1], the value k is the smallest solution to the equation $l + k + 1 \equiv 448 \bmod 512$. Then the padded message is expressed as N 512-bit blocks, denoted $M^{(1)}$, $M^{(2)}$, ... , $M^{(N)}$.

B. Hash Computation

In the hash computation, message blocks are processed in order. Each message block is processed as follows.

The block $M^{(i)}$ is expressed as sixteen 32-bit words, denoted $M_0^{(i)}$, $M_1^{(i)}$, ... , $M_{15}^{(i)}$. And it is expanded into sixty-four 32-bit words as shown in (1) and (2). The details of the functions σ_0 and σ_1 are given in [1].

For $0 \le t \le 15$,

$$W_t = M_t^{(i)} \tag{1}$$

For $16 \le t \le 63$,

$$\begin{aligned} W_t = {} & \sigma_1(W_{t-2}) + W_{t-7} \\ & + \sigma_0(W_{t-15}) + W_{t-16} \end{aligned} \tag{2}$$

The word W_t is used in the round computation, as shown in Fig. 1. The round computation will be performed 64 times, and the new intermediate hash values can be computed by:

$$\begin{aligned} H_0^{(i)} &= A_{64} + H_0^{(i-1)} \\ H_1^{(i)} &= B_{64} + H_1^{(i-1)} \\ &\cdots \\ H_7^{(i)} &= H_{64} + H_7^{(i-1)} \end{aligned} \tag{3}$$

When the next block $M^{(i+1)}$ is processed, the values $H_0^{(i)} - H_7^{(i)}$ are used as the initial values of $A - H$. The functions σ_0, σ_1, Σ_0, Σ_1, Maj and Ch are given in [1].

After the final block has been processed, the 256-bit message digest is obtained.

978-1-7281-3320-1/20/$31.00 ©2020 IEEE

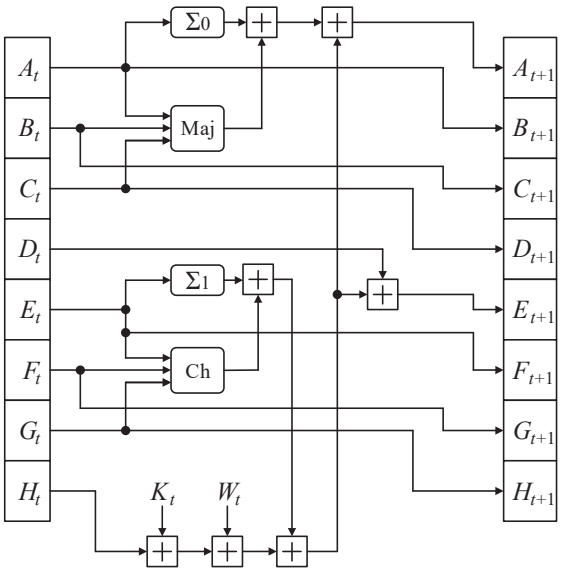

Fig. 1. SHA-256 round computation

III. Proposed Design For SHA-256

In each round of SHA-256, the computation of the variables A and E can be described as:

$$\begin{aligned} A_{t+1} &= \Sigma_0(A_t) + Maj(A_t, B_t, C_t) + \Sigma_1(E_t) \\ &\quad + Ch(E_t, F_t, G_t) + K_t + W_t + H_t \\ E_{t+1} &= \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + K_t + W_t \\ &\quad + H_t + D_t \end{aligned} \tag{4}$$

The remaining variables can be obtained directly as:

$$\begin{aligned} B_{t+1} &= A_t \quad C_{t+1} = B_t \quad D_{t+1} = C_t \\ F_{t+1} &= E_t \quad G_{t+1} = F_t \quad H_{t+1} = G_t \end{aligned} \tag{5}$$

A. Basic Rescheduling

According to the work in [2], [3], [11], (4) can be rewritten as:

$$\begin{aligned} A_{t+1} &= \Sigma_0(A_t) + Maj(A_t, B_t, C_t) + \Sigma_1(E_t) \\ &\quad + Ch(E_t, F_t, G_t) + P_t \\ E_{t+1} &= \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + Q_t \end{aligned} \tag{6}$$

where P_t and Q_t are computed by:

$$\begin{aligned} P_t &= K_t + W_t + H_t = K_t + W_t + G_{t-1} \\ Q_t &= K_t + W_t + H_t + D_t \\ &= K_t + W_t + G_{t-1} + C_{t-1} \end{aligned} \tag{7}$$

As shown in (2), the computation of W_t is independent of the variables $A - H$. And K_t $(0 \leq t \leq 63)$ is the round constant. So P_t and Q_t can be precalculated in the previous clock cycle. As a result, the round computation of SHA-256 is divided into two stages. The values P_t and Q_t can be precalculated in the first stage. The values A_{t+1} and E_{t+1} are computed in the second stage.

B. Proposed Rescheduling

In this paper, we propose a rescheduling method for the SHA-256 round computation. The values S_t and N_t are used to save the results of $\Sigma_1(E_t)$ and $Ch(E_t, F_t, G_t)$, as shown in (8). The values P_t and Q_t continue to be used.

$$\begin{aligned} S_t &= \Sigma_1(E_t) \\ N_t &= Ch(E_t, F_t, G_t) \end{aligned} \tag{8}$$

So the value E_t can be computed by:

$$E_t = S_{t-1} + N_{t-1} + Q_{t-1} \tag{9}$$

Because F_t and G_t can be given directly by E_{t-1} and F_{t-1} respectively, the values S_t and N_t can be computed by:

$$\begin{aligned} S_t &= \Sigma_1(S_{t-1} + N_{t-1} + Q_{t-1}) \\ N_t &= Ch(\ (S_{t-1} + N_{t-1} + Q_{t-1}), E_{t-1}, F_{t-1}\ ) \end{aligned} \tag{10}$$

This means that the values S_t and N_t can also be precalculated in the previous clock cycle. So the round computation in (6) can be rewritten as:

$$\begin{aligned} A_{t+1} &= \Sigma_0(A_t) + Maj(A_t, B_t, C_t) + S_t + N_t + P_t \\ E_{t+1} &= S_t + N_t + Q_t \\ S_{t+1} &= \Sigma_1(S_t + N_t + Q_t) \\ N_{t+1} &= Ch(\ (S_t + N_t + Q_t), E_t, F_t\ ) \end{aligned} \tag{11}$$

Also, the first values of S_t and N_t can be computed by:

$$\begin{aligned} S_0 &= \Sigma_1(E_0) \\ N_0 &= Ch(E_0, F_0, G_0) \end{aligned} \tag{12}$$

Further, the value A_{t+1} can be computed by (13). The values $M1$ and $M2$ are the sum and carry vectors of a carry save addition. This addition can be performed in parallel with the operations $\Sigma_0(A_t)$ and $Maj(A_t, B_t, C_t)$. So the value A_{t+1} can be regarded as the addition of 4 intermediate values.

$$\begin{aligned} A_{t+1} &= \Sigma_0(A_t) + Maj(A_t, B_t, C_t) + M1 + M2 \\ M1 + M2 &= S_t + N_t + P_t \end{aligned} \tag{13}$$

According to (7), (11), (12) and (13), a hardware structure of SHA-256 is proposed, as shown in Fig. 2. The values $M1$ and $M2$ are generated by a CSA (carry save adder). And the addition of 4 operands in (13) are implemented by a 4-2 compressor and an adder. The 4-2 compressor is introduced in [12] and its critical path is equal to three XORs.

The round computation is divided into two pipeline stages by the registers P and Q. Before the registers P and Q get their first values (denoted P_0 and Q_0 respectively), the pipeline is empty. So one clock cycle is needed to make the pipeline full before the first round computation. According to (7), the values D_0 and H_0 are necessary for the computation of P_0 and Q_0. When the pipeline is full, the values C_t and G_t are used to precalculate the values P_t and Q_t. So D_t and H_t are only used in the first clock cycle of processing one block.

In Fig. 2, there are 6 registers which are used to store the variables A, B, C, E, F and G. Because D_t and H_t are only used at the beginning, the registers storing the variables D and

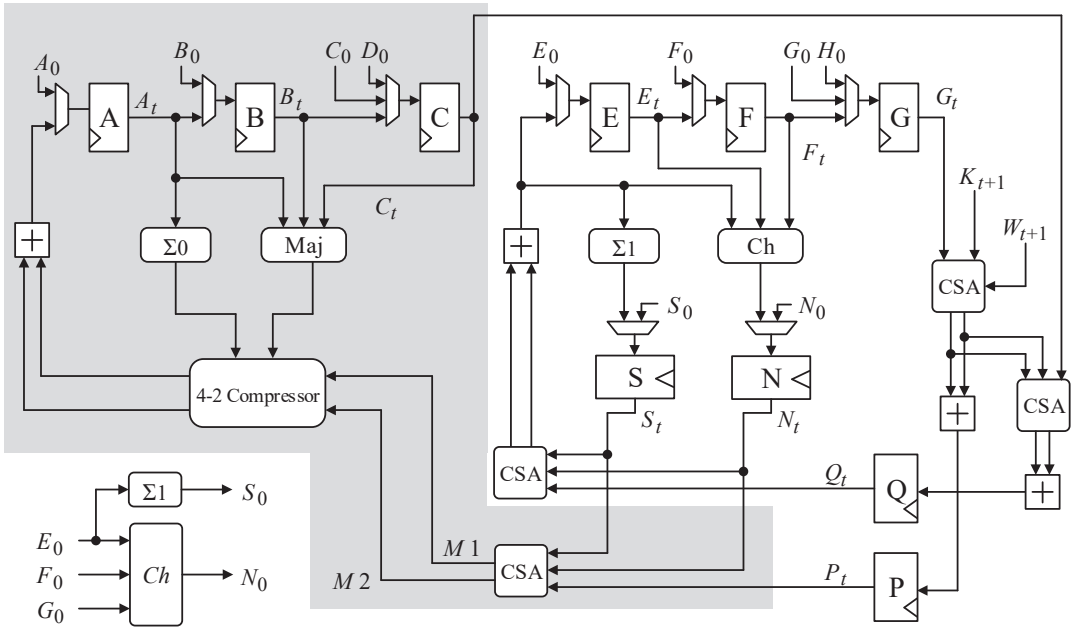

Fig. 2. Structure of SHA-256 with rescheduling

H are removed in order to reduce the area cost. This design works as follows:

- In the first clock cycle, the values D_0 and H_0 are required. The registers C and G are loaded with the values D_0 and H_0 respectively. The values P_0 and Q_0 are computed in this cycle.
- In the second clock cycle, the values of D and H are not required. The registers A, B, C, E, F and G are loaded with their first values: A_0, B_0, C_0, E_0, F_0 and G_0 respectively. Meanwhile, the registers S and N are loaded with their first values S_0 and N_0 respectively. As shown in (12), S_0 and N_0 can be computed by using E_0, F_0 and G_0. Also, the registers P and Q get their first values P_0 and Q_0. The values $A_0 - H_0$ are the standard constants or the previous hash values.
- From the third clock cycle, the registers get their new values from the round computation. Considering that $D_{t+1} = C_t$ and $H_{t+1} = G_t$, the final values of the variables D and H can be obtained in advance, although there are no registers for D and H in Fig. 2.

According to [9], the functions Maj and Ch can be expressed as:

$$\begin{aligned} Maj(a,b,c) &= (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c) \\ &= a \wedge (b \vee c) \vee (b \wedge c) \\ Ch(e,f,g) &= (e \wedge f) \oplus (\neg e \wedge g) \\ &= (e \wedge f) \vee (\neg e \wedge g) \end{aligned} \tag{14}$$

Furthermore the delay of Σ_0 or Σ_1 is comparable to the delay of a CSA. So the critical path is the computation of the next value of A, as shown in the gray region in Fig. 2. The delay of the critical path is equal to: delay(CSA) + delay(4-2 compressor) + delay(adder) + delay(multiplexer).

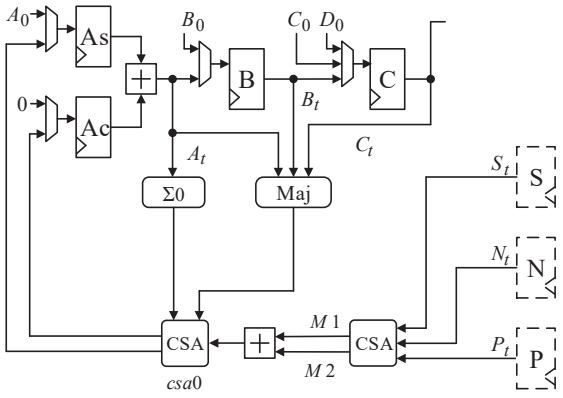

Fig. 3. Optimization of the structure inside the gray region in Fig. 2

C. Further Optimization

The structure inside the gray region in Fig. 2 is further optimized. The remaining region in Fig. 2 keeps unchanged. The details of the optimization are as follows:

- Firstly, an adder is used to merge the vectors $M1$ and $M2$. So the 4-2 compressor can be replaced by a CSA, denoted $csa0$. But there will be two adders in the critical path.
- Then the delay balancing technique introduced in [2] can be used. The sum and carry vectors that $csa0$ outputs are stored in two registers As and Ac respectively, as shown in Fig. 3. So an adder is moved out of the critical path.

So the delay of the critical path is reduced to: delay(CSA) + delay(adder) + delay(CSA) + delay(multiplexer). And several steps of processing one block have changed as follows.

TABLE I
COMPARISON OF DIFFERENT DESIGNS

Design	Device	Clock frequency (MHz)	Clock cycles per block	Throughput (Mbps)	Area (Slices)	Efficiency (Mbps/Slice)
[4] (2x-unrolled)	Virtex-II	73.975	38	996.7	2032	0.491
[6]	Virtex-4	170.75	65	1344.98	610	2.2
[10]	Virtex-4	50.06	280	91.53	422	0.22
[13] (Case I)	Virtex-4	238	321	379.6	382	0.99
[13] (Case II)	Virtex-4	222	129	881.1	485	1.82
Our design	Virtex-4	255.7	66	1984	979	2.02

When the register B is loaded with its first value B_0, the register As is loaded with A_0 and Ac is loaded with zero. After the final round, the values of As and Ac are added to the old A_0. This addition is implemented by a CSA and an adder.

IV. IMPLEMENTATION RESULTS AND COMPARISON

Our design for SHA-256 is implemented on Xilinx Virtex-4 XC4VLX100-12 FPGA, using the Xilinx ISE 14.7 tool. The whole implementation also contains a message expander. The performance comparison of our design with some published designs is shown in Table I. And the throughput and efficiency can be computed by:

$$\begin{aligned} Throughput &= \frac{Block\ size \times Clock\ frequency}{Clock\ cycles\ per\ block} \\ Efficiency &= \frac{Throughput}{Number\ of\ Slices} \end{aligned} \tag{15}$$

The block size for SHA-256 is 512-bit. Our design requires 66 clock cycles to process a 512-bit block. One is for the computation of P_0 and Q_0. And one is for the final addition. Other 64 clock cycles are for the round computation.

The 2x-unrolled design in [4] combines pipeline with unrolling techniques, which can perform two rounds in one clock cycle. The design in [6] is based on the pipeline architecture and carry skip adders are used to improve the performance of the hardware core. The design in [10] is a compact FPGA processor for SHA-256. The designs in [13] use architectural folding technique. Case I is folded by factor 5 and has the lowest area cost. Case II is folded by factor 2 and has a better balance between the area and throughput than Case I.

Among the designs in Table I, our design has the highest throughput. The design in [6] has the higher efficiency than ours. But the throughput of our design is 47% higher than that of the design in [6].

V. CONCLUSION

Our rescheduling method is effective for reducing the critical path in the SHA-256 hardware implementation. The implementation results show that our design based on the proposed rescheduling has a significant improvement in terms of the throughput. Further, the structure in our design for SHA-256 can be extended to the implementations of all the SHA-2 family.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No.61974083 and No.61674086. The corresponding author is Shuguo Li.

REFERENCES

[1] National Institute of Standards and Technology (NIST). Announcing the secure hash standard. Federal Information Processing Satndards Publication 180-2 (FIPS PUB 180-2), August 2002.

[2] L. Dadda, M. Macchetti, and J. Owen, "The design of a high speed ASIC unit for the hash function SHA-256 (384, 512)," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, Feb 2004, pp. 70–75.

[3] L. Dadda, M. Macchetti,and J. Owen, "An ASIC design for a high speed implementation of the hash function SHA-256 (384, 512)," in *ACM Great Lakes symposium on VLSI*, 2004, pp. 421–425.

[4] R. P. McEvoy, F. M. Crowe, C. C. Murphy, and W. P. Marnane, "Optimisation of the SHA-2 Family of Hash Functions on FPGAs," in *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, March 2006, pp. 317–322.

[5] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, "Cost-Efficient SHA Hardware Accelerators," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 8, pp. 999–1008, Aug 2008.

[6] M. Padhi and R. Chaudhari, "An optimized pipelined architecture of SHA-256 hash function," in *2017 7th International Symposium on Embedded Computing and System Design (ISED)*, Dec 2017, pp. 1–4.

[7] H. Michail, A. Kakarountas, A. Milidonis, and C. Goutis, "A Top-Down Design Methodology for Ultrahigh-Performance Hashing Cores," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 4, pp. 255–268, Oct 2009.

[8] H. E. Michail, G. S. Athanasiou, V. Kelefouras, G. Theodoridis, and C. E. Goutis, "On the Exploitation of a High-Throughput SHA-256 FPGA Design for HMAC," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 1, pp. 1–28, 2012.

[9] F. Opritoiu, S. L. Jurj, and M. Vladutiu, "Technological solutions for throughput improvement of a Secure Hash Algorithm-256 engine," in *2017 IEEE 23rd International Symposium for Design and Technology in Electronic Packaging (SIITME)*, Oct 2017, pp. 159–164.

[10] R. García, I. Algredo-Badillo, M. Morales-Sandoval, C. Feregrino-Uribe, and R. Cumplido, "A compact FPGA-based processor for the Secure Hash Algorithm SHA-256," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 194–202, 2014.

[11] I. Algredo-Badillo, C. Feregrino-Uribe, R. Cumplido, and M. Morales-Sandoval, "FPGA-based implementation alternatives for the inner loop of the Secure Hash Algorithm SHA-256," *Microprocessors and Microsystems*, vol. 37, no. 6, pp. 750–757, 2013.

[12] K. Prasad and K. K. Parhi, "Low-power 4-2 and 5-2 compressors," in *Conference Record of Thirty-Fifth Asilomar Conference on Signals, Systems and Computers*, vol. 1, Nov 2001, pp. 129–133.

[13] M. M. Wong, V. Pudi, and A. Chattopadhyay, "Lightweight and High Performance SHA-256 using Architectural Folding and 4-2 Adder Compressor," in *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2018, pp. 95–100.