Efficient Hardware Architecture of SHA-256 Algorithm for Trusted Mobile Computing

Mooseop Kim[1], Jaecheol Ryou[2], and Sungik Jun[1]

[1] Electronics and Telecommunications Research Institute (ETRI)
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-700, South Korea
{gomskim,sijun}@etri.re.kr
[2] Division of Electrical and Computer Engineering,
Chungnam National University
220 Gung-dong, Yuseong-gu, Daejeon, 305-764, South Korea
jcryou@home.cnu.ac.kr

Abstract. We present a compact SHA-256 hardware architecture suitable for the Trusted Mobile Platform (TMP), which requires low-area and low-power characteristics. The built-in hardware engine to compute a hash algorithm in TMP is one of the most important circuit blocks and contributes the performance of the whole platform because it is used as key primitives supporting platform integrity and command authentication. Unlike personal computers, mobile platform have very stringent limitations with respect to available power, physical circuit area, and cost. Therefore, special architecture and design methods for a compact hash hardware module are required. Our SHA-256 hardware can compute 512-bit data block using 8,588 gates on a 0.25μm CMOS process. The highest operation frequency and throughput of the proposed architecture are 136MHz and 142Mbps, which satisfies processing requirement for the mobile application.

1 Background and Motivation

The Trusted Mobile Platform(TMP) [1] guarantees the integrity of the mobile platform and is a preferred requirement in the evolutionary process where the mobile device changes into the open platform and value-based application technology. TMP improves the reliability and security of a device using Trusted Platform Module (TPM), which ensures that the device is running on the authorized software and hardware environment. The TPM is a microcontroller-based security-engine based on an industry standard specification issued by the Trusted Computing Group (TCG). It protects encryption keys and digital signature keys to maintain data confidentiality and integrity. Especially important, TPM chip is specifically designed to protect a platform and user authentication information from software-based attacks.

The built-in hardware engine for hash algorithm in TMP is one of the most important circuit blocks because it is used as a key primitive supporting integrity verification and used in the most of TPM-commands for authentication of the platform. Current version of TMP [1] specification recommends to use the SHA-1 algorithm. However, recently, National Security Agency (NSA) announced Suite B Cryptography that specify encryption, signature and hash algorithms at the 2005 RSA Conference. According to the Suite B Cryptography, SHA-256 is a common mode for widespread interpretability and appropriate for protecting classified information up to the SECRET level. This cryptographic trend to move over Suite B is realized in the trusted mobile computing. Furthermore, TCG announced that TPM must support SHA-256 algorithm in the revised specification, TPM NEXT [2].

Integrating TCG's security features into a mobile phone could be a challenge work. In reality, most mobile devices do not require a very high data processing speed. For example, when cellular wireless network technology migrates from 2.5G to 3G, only the data rate is increased from 144kbps to 2Mbps. Also, data rate of Bluetooth, which is one of the wireless Personal Area Network(PAN), is only 10Mbps maximum. However, when security function is added, considerable computing power is demanded to the microprocessor of a handheld device. For example, the processing requirements for AES and SHA at 10Mbps are 206.3 and 115.4 MIPS respectively [3]. In comparison, a state-of-the art handset processors, such as MPC860 and ARM7 are capable of delivering up to 106MIPS and 130MIPS, respectively [4, 5]. The above data indicates a clear disparity between security processing requirements and available processor capabilities, even when assuming that the handset processor is fully dedicated to security processing. In reality, the handset processor also needs to execute the operating system, and application software, which by themselves generate a significant processing workload.

Another critical problem of security processing on a mobile platform is power consumption and battery capacity. Unlike personal computers, mobile devices have strict environment in power consumption, in battery life and in available circuit area. Among these limitations, the power consumption is the critical metric to be minimized in the design of cryptographic circuits for mobile platforms. For battery-powered systems, the energy drawn from the battery directly influences the systems battery life, and, consequently, the duration and extent of its mobility, and its overall utility. In general, battery-driven systems operate under stringent constraint especially in limited power. The power limitation gets more serious when the mobile device is subject to the demand of security operations. According to the estimation of [6], the execution of security applications on a battery-powered device can decrease battery life at least by half.

Therefore, design methodologies at different abstraction levels, such as systems, architectures, logic design, basic cells, as well as layout, must take into account to design of a low-cost SHA-256 module for trusted mobile platform. In this paper, we introduce an efficient hardware architecture of low-cost SHA-256 algorithm for trusted mobile platforms. As a result, a compact SHA-256 hardware implementation capable of supporting the integrity check and command authentication of trusted mobile platforms could be developed and evaluated so far.

The rest of this paper is constructed as follows. Section 2 describes a brief review of SHA-256 algorithm and a short summary of some previous works for the design of SHA-256 algorithm. Section 3 describes the architecture of our SHA-256 circuits and corresponding implementation options are described. The implementation result is summarized and compared with other SHA-256 implementation in section 4. Finally, in section 5, we conclude this work.

2 SHA-256 Algorithm and Its Previous Works

2.1 Basic Scheme of SHA-256

The SHA-256 takes a message of length less than 2^{64} bits and produces a final message digest of 256 bits. The message digest is dependent of the input message composed by multiple blocks of 512 bits each. The data block is applied to the 64 rounds of the SHA-256 Hash function in words of 32 bits, denoted by W_t.

In the SHA-256 algorithm, six logical functions operating on 32-bit values are used:

$$\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \oplus (\bar{x} \wedge z) \\
Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\
\textstyle\sum_0(x) &= S^2(x) \oplus S^{13}(x) \oplus S^{22}(x) \\
\textstyle\sum_1(x) &= S^6(x) \oplus S^{11}(x) \oplus S^{25}(x) \\
\sigma_0(x) &= S^7(x) \oplus S^{18}(x) \oplus R^3(x) \\
\sigma_1(x) &= S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)
\end{aligned} \tag{1}$$

where $\wedge$ and $\oplus$ are bitwise AND and XOR operations; and S and R are the rotate right and shift right functions respectively.

The first operation of SHA-256 computation is a message padding. In order to hash a message M of l bits, a single 1-bit is appended to the end of the message followed by k 0-bits. The k is the smallest solution to the equation $l+1+k \equiv 448 \mod 512$. The binary representation of l as a 64-bit number is then appended so the length of the padded message is a multiple of 512 bits. The padded message is then divided into N 512-bit blocks $M^{(1)}, M^{(2)} \cdots M^{(N)}$.

After a message padding, eight 32-bit working variables a, b, c, d, e, f, g, h are initialized to $H_0, H_1, \cdots, H_7$ respectively. These initial values specified in [7] are only set for the first data block. The consecutive data blocks use the partial digest message computed for the previous data block. The message compression function is then performed for 64 rounds:

$$\begin{aligned}
T_1 = h + \textstyle\sum_1(e) + Ch(e,f,g) + K_t + W_t;\ T_2 = \textstyle\sum_0(a) + Maj(a,b,c); \\
h = g;\ g = f;\ f = e;\ e = d + T_1;\ d = c;\ c = b;\ b = a;\ a = T_1 + T_2
\end{aligned} \tag{2}$$

where W_t is the 32-bit data computed by message scheduler for the t round and the K_t value represents the 32-bit constant that also depends on the round. Finally, the partial digest message or final digest is computed:

$$\begin{aligned}
H_0 = a + H_0;\ H_1 = b + H_1;\ H_2 = c + H_2;\ H_3 = d + H_3; \\
H_4 = e + H_4;\ H_5 = f + H_5;\ H_6 = g + H_6;\ H_7 = h + H_7;
\end{aligned} \tag{3}$$

2.2 Previous Works

After the ratification of Secure Hash Algorithm (SHA) in 1995, numerous FPGA [8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21] and ASIC [12, 14, 15, 21] implementations of SHA-256 algorithm were previously proposed and evaluated. Major design differences among them lie in the trade-off between area and speed. Most of these implementations feature high speeds and high costs suitable for high-performance usages such as WTLS, IPSec and so on.

Early SHA-256 design were mostly straightforward implementations of various loop rolling architectures with limited number of architectural optimization. S.Dominikus [12] used loop rolling technique in order to reduce area requirement. He proposed an architecture uses only 4 operation blocks, one for each round. Using a temporal register and a counter, each operation block is reused for 20 iterations. This architecture use a feedback structure where the data are iteratively transformed by the round functions.

The most critical path in the SHA-256 circuit is the chain of additions (modulo 2^{32}) for the calculation of working variable A. Several methods have been proposed to speed up calculations in the SHA design.

The most relevant method for SHA implementation is using a Carry-Save Adder (CSA) [10, 11, 17]. CSA separates the sum and carry paths, thus it minimizes the delay caused by traditional carry propagation time. Since CSAs accept 3 input operands, the working variable A can be calculated in the SHA-256 using just 5 CSAs. However, this method requires additional 2-input operand adder to recombine the sum and carry results.

Another remarkable architecture of the SHA-256 implementation is based on the use of four pipeline stages [16]. This method uses registers to break the long critical path within SHA core. The main advantage of this architecture is that it increases the parallelism of SHA-256 algorithm, which should have a positive effect on throughput. But this approach requires large area, since this method duplicates hardware for implementing each round.

Unfortunately, most of these implementations have been designed aiming only at large message and high throughput operation, without considering the power consumption.

3 Proposed Hardware Architecture

For the SHA-256 design proposed in this paper, we assume that one 512-bit data block, that is processed by microprocessor, is stored in memory and available to SHA-256 circuit for reading and writing. The operation of our SHA-256 circuit is broken into three steps. The initial step comprises reading of a message and control signal. Here, the interface block reads message data by 32 bits and saves them to data memory. The next step is round operation which repeats itself by 64 rounds. During this step, message expansion block computes the intermediate value W_t simultaneously. In the final step, the final hash values are computed from the intermediate output values.

3.1 Design of a Compact SHA-256 Architecture

In the proposed design, we make an effort to minimize the complexity of architecture implementing SHA-256 algorithm. Fig.1 shows main circuit blocks and their interactions in our SHA-256 design: interface block, message schedule, message compression and controller.

I/O Interface block is responsible for saving input message into the data memory in the message schedule module. We use 32-bit data bus for an efficient design of the SHA-256 core, since all the operations of SHA-256 algorithm and the corresponding variables need 32-bit data as a basic unit. Although the smaller bus size may require the less registers, it invokes the more data selectors with the restricted resource sharing, resulting in an inefficient implementation.

The controller is used to generate signal sequences necessary to check an input signal sequence and control the datapath parts. After initialize phase, the controller is totally responsible for the system operation. It defines the proper constants and manages the memory block. It also controls all the proper algebraic and digital logic functions for SHA-256 algorithm. The controller structure consists of state register and two logic blocks. The input logic block computes the next state by executing the finite state machine over the current state and the new set of input signals. The output logic block generates the control signals for datapath employing the function signals of the current states. A reasonable amount of power is consumed in the logic blocks and the clock distribution due to the flip-flop operation of the state register.

The efficiency of the SHA-256 hardware in terms of circuit area, power consumption and throughput is mainly determined by the data path structure of

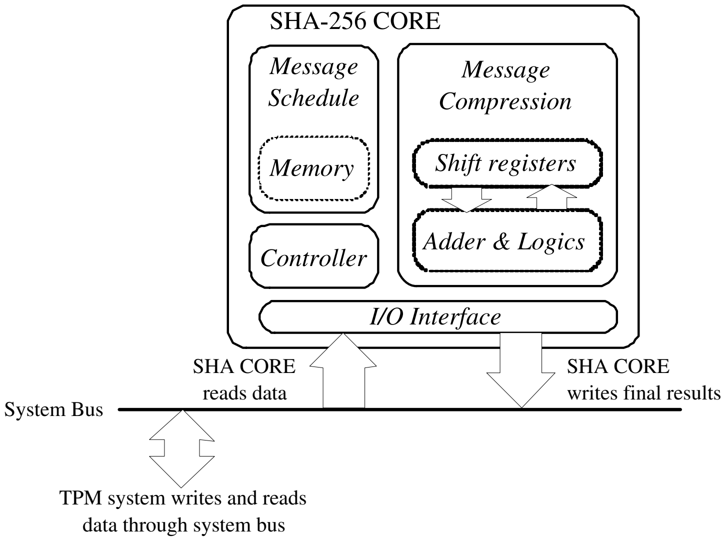

Fig. 1. Outline of SHA-256 circuit block

message schedule and message compression blocks. To devise a compact architecture, we adopt a folded architecture to design message schedule and message compression blocks. The folded architecture, that is based on rolling architecture, executes each round computation over several clocks to reduce hardware resource. The most important advantage of folded architecture over previous works is that it can dramatically reduce both the number of adders in message compression block and the number of registers in message schedule block.

Our architecture needs just one adder to compute the round operation of the SHA-256 algorithm. Adders are the most spacious part in the message compression block. In addition, we use only one 32-bit register for the computation of message schedule block, while previous woks require 16 32-bit registers. Since the number of adders and registers determines the overall size of any SHA-256 hardware circuit, this approach employed for implementing both message compression and message schedule is crucial for an efficient SHA-256 hardware implementation.

In the following paragraph, the method presented by this paper to reduce the circuit area of message schedule and message compression is explained in detail.

3.2 Design of Message Compression

The message compression block performs the actual hashing operation over the message-dependant word(W_t) stream output from the message schedule block. As shown in Fig.2, SHA-256 algorithm uses eight 32-bit variables $a, b, \cdots, h$ to store new values in each round operation.

It can be easily seen from Fig.2 that six out of eight values at a, b, c and e, f, g are shifted by one position down to b, c, d and f, g, h respectively, in each round and a, e should be occupied with a new value that is computed, where the operands depend on all input values, the round-dependant constant K_t, and current message W_t.

Fig.3 shows in detail the proposed architecture of message compression block. The characteristics of this architecture are follows. Firstly, we use the eight-stage 32-bit shift registers because six out of eight values at a, b, c and e, f, g are shifted by one position down in each round.

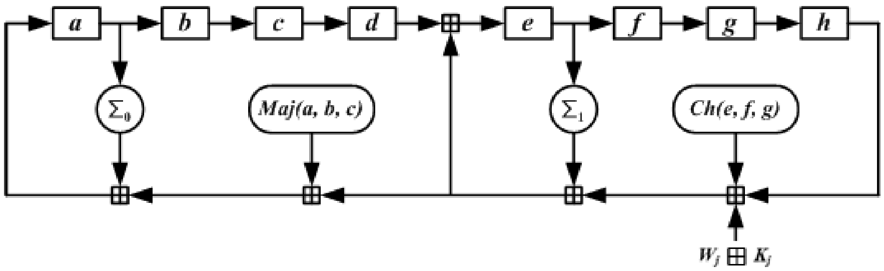

Fig. 2. Message compression functional block diagram of SHA-256 algorithm

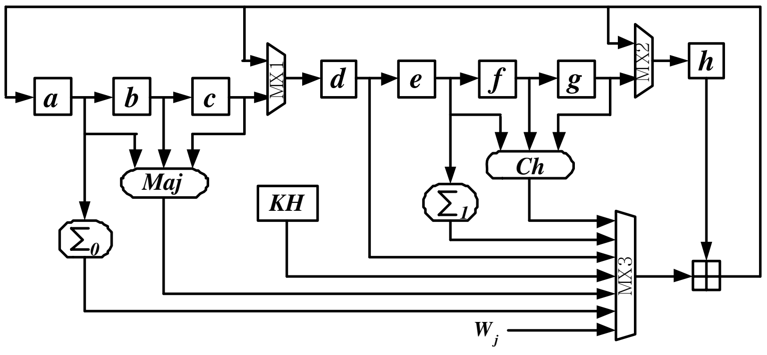

Fig. 3. Folded architecture of message compression block

Table 1. Functional steps for single round operation for message compression block

Steps	Detailed operations	temporary values
step 1	$h = h + Ch(e,f,g)$	$h = h + Ch(e,f,g)$
step 2	$h = h + K_j$	$h = h + Ch(e,f,g) + K_j$
step 3	$h = h + \sum_1(e)$	$h = h + Ch(e,f,g) + K_j + \sum_1(e)$
step 4	$h = h + W_j$	$h = h + Ch(e,f,g) + K_j + \sum_1(e) + W_j$
step 5	$d = d + h$	$h = h + Ch(e,f,g) + K_j + \sum_1(e) + W_j$ $d = d + h + Ch(e,f,g) + K_j + \sum_1(e) + W_j$
step 6	$h = h + Maj(a,b,c)$	$h = Maj(a,b,c) + h + Ch(e,f,g) + K_j + \sum_1(e) + W_j$
step 7	$a = h + \sum_0(a)$; $b = a$; $b = a$ $c = b$; $d = c$; $e = d$; $f = e$; $g = f$; $h = g$;	$a = \sum_0(a) + Maj(a,b,c) + h + Ch(e,f,g) + K_j + \sum_1(e) + W_j$; $b = a$; $c = b$; $d = c$; $e = d + \sum_1(e) + Ch(e,f,g) + h + K_j + W_j$ $f = e$; $g = f$; $h = g$;

Secondly, in the block, only one 32-bit adder is necessary for all round computations. For iterative computation of $T_1 = h + \sum_1(e) + Ch(e,f,g) + K_t + W_t$ and $T_2 = \sum_0(a) + Maj(a,b,c)$, register h and d are used for the storage of the temporary addition values. It is necessary seven times of modulo 2^{32} addition for a single round operation. Therefore, seven clock cycles are required for one round operation as summarized in table1.

In the initialization phase, all registers are initialized and multiplexers choose path zero to load initial values($H_0, H_1 \cdots H_7$) stored in KH module. In this phase, eight 32-bit registers($a, b, \cdots h$) for working variables operate as a shift register. Therefore, eight clock cycles are required to load initial values to each register. For optimized power consumption, the gated clock is applied to each register in the block.

Four logical functions, ($Ch, Maj, \sum_0, \sum_1$) are used for the round operation of message compression. Each of these functions operates on 32-bit words and produces a 32-bit word as output.

KH module in Fig.3 stores the set of specified constants supporting round-dependant value K_t. It also stores the initial values H_i, $0 \leq i \leq 7$, that the SHA-256 specifies. These initial values are used in the initial procedure of the message compression block and in the final computation for message-digest.

The selection of K_t and H_i is determined by the selection signal from the controller.

After 64 round operations for message compression, eight 32-bit intermediate values are stored in the registers $a, b, \cdots h$. The final message-digest computation consists of the modulo 2^{32} additions between these intermediate values and eight initial values. This can be done very efficiently with the help of additional multiplexer and the eight stage shift registers for working variables. Therefore, it takes eight clock cycles to compute the final message-digest. The results of these additions are stored in the KH module as the initial values for the next 512-bit data block computation.

3.3 Design of Message Scheduler

Another important part of SHA-256 core data path is a message schedule block that generates message dependant words, W_t, used as an input to each round operation of the message compression block. The message schedule block is the most expensive part of SHA-256 in terms of hardware resources. Therefore, its space-efficient implementation is a critical part in the compact design of SHA-256 circuit.

As shown in Fig.4, the conventional message schedule block is usually implemented using 16-stage 32-bit shift registers and three 32-bit adders for 512-bit data block processing. However, this method is inefficient to be employed in mobile platforms because it requires a significant amount of circuit area.

The message schedule block performs the cryptographic computation formulated in Eq. 4, where $M_t^{(i)}$ denotes t-th value in the first sixteen 32-bit data of i-th data block.

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases} \tag{4}$$

It can be easily seen from Eq. 4 that the message in the memory is used only to compute the first 16 round operations. In other words, the memory is not used any more after sixteenth round calculations. From the viewpoint of resource efficiency, this method is inefficient and wastes too many hardware resources.

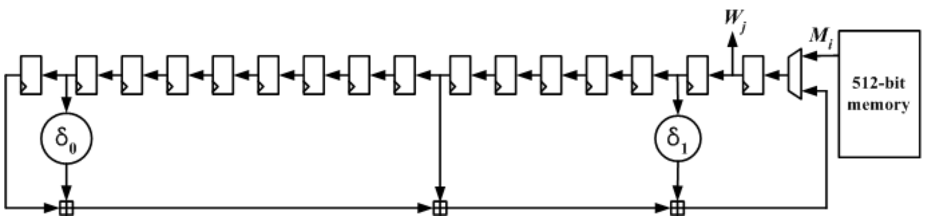

Fig. 4. General method of message schedule for SHA-256

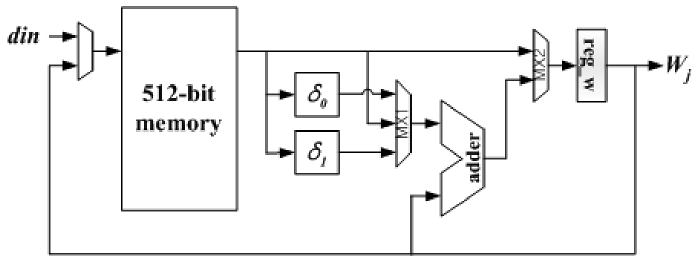

Fig. 5. Proposed architecture of message schedule block

Table 2. Functional steps for message schedule block

		Operation of circuit blocks				
Round(i)	step	mem.out	mx1.out	adder.out	mx2.out	reg_w
$0 \le i \le 15$	s1	M_i	x	x	M_i	$\Rightarrow$
$16 \le i \le 63$	s1	M_{i-16}	x	x	M_{i-16}	$\Rightarrow$
	s2	M_{i-15}	$\sigma_0(M_{i-15})$	$M_{i-16} + \sigma_0(M_{i-15})$	$\Rightarrow$	$\Rightarrow$
	s3	M_{i-7}	$\Rightarrow$	$M_{i-16} + \sigma_0(M_{i-15}) + M_{i-7}$	$\Rightarrow$	$\Rightarrow$
	s4	M_{i-2}	$\sigma_1(M_{i-2})$	$M_{i-16} + \sigma_0(M_{i-15}) + M_{i-7} + \sigma_1(M_{i-2})$	$\Rightarrow$	$\Rightarrow$

As an alternative compact design, we propose an optimized architecture of message schedule block, which enhances the resource sharing of the memory. Our approach uses only one 32-bit register(reg_w) and 32-bit adder to implement message schedule in the SAH-256 circuit. The register reg_w is used to save temporary values during the computation of W_t. The detailed design of the proposed message schedule block is shown in Fig.5.

The iterative functional steps of logic units for our message schedule block are summarized in Table 2, where 'x' means logical function of *Don't care* and '$\Rightarrow$' represents the identical derivation of the left row. Four values (M_{t-2}, M_{t-7}, M_{t-15}, M_{t-16}) of memory data have to be read and applied to compute the result that is written back to memory in each round. As summarized in Table 2,this process takes 4 clock cycles. Message schedule could be calculated simultaneously during message compression, which consumes seven clock cycles. Therefore, no additional clock cycle is required for the computation of W_t in the message schedule block.

The new computation result of W_t, completed on the fourth clock, is fed to the message compression block for round computation and is written back to the memory for the following round calculation of W_{t+1}. Dedicated hard wired logic and counter are used to compute the address necessary for memory access. In this process, the memory address is defined as $t \bmod 2^4$ when the current round is t.

The memory in Fig.5 is 32-bit register-based and single port 512-bit memory that is installed by using standard logic cells. In order to minimize the power consumption, the internal registers of the memory are disabled when they are in idle state, thus reducing the amount of unnecessary switching activity. Additional multiplexer is used to select input data between initial input message and new word data W_t.

4 Implementation Results and Comparison

The described architectures of our design have been implemented in VHDL and their operations are verified through functional simulation using Active HDL, from Aldec Inc. In order to evaluate our design, we used Synopsys synthesis flows on Sun Solaris platform. The target technology is Samsung Electronics STD110 library, featuring $0.25\mu m$ CMOS standard cell and 2.5V core voltage. Synopsys Power Compiler was then used to calculate the overall power dissipation of our design. Although the maximum operating frequency obtained using timing analysis is 136 MHz, we used 25 MHz as the operating frequency to evaluate the consuming power of our circuit because the system clock of most cellular phones is about 20 MHz. We would like to emphasize that our design is on the algorithmic and architectural level. Implementing our designs using an low power ASIC library or a full custom design will enable higher energy and power savings.

Table 3 and table 4 show the circuit area and power estimation quantities of the design in the level of individual logic block. The proposed SHA-256 circuit and its auxiliary blocks like interface and memory are implemented in an area of 8,588 gates. As it is considered that the hardly-optimizable memory block and registers for working variables use about 56% of the total area, the remaining logic blocks are designed effectively. The proposed design needs 490 clock cycles(including data input and output cycles) to compute the hash value of one 512-bit data block.

Table 3. Components and their complexity of SHA-256 core

Logic block	circuit component	gates	percent [%]
IO interface	con_reg	32	0.38
	IO buffer	1,024	12
Message schedule	memory	3,434	40
	σ_0, σ_1	234	2.7
	mux	286	3.3
	adder2	198	2.3
	reg_w	176	2.1
Massage compression	reg_a∼h	1,408	16.4
	$\sum_0, \sum_1$	256	3
	Ch()	86	1.0
	Maj()	96	1.1
	H_table	46	0.5
	K_table	488	5.68
	mux	364	4.24
	adder1	198	2.3
controller		262	3.0
Total		8,588	100 %

Table 4. Components and their power consumption of SHA-256 core

logic block	circuit component	power [mW]	percent [%]
IO interface	con_reg	0.00	0.0
	IO buffer	0.119	4.3
Message schedule	memory	0.895	32
	σ_0, σ_1	0.096	3.5
	mux	0.198	7.0
	adder2	0.164	5.9
	reg_w	0.041	1.5
Message compression	reg_a∼h	0.263	9.4
	$\sum_0, \sum_1$	0.027	1.0
	Ch()	0.015	0.5
	Maj()	0.01	0.3
	H_table	0.005	0.2
	K_table	0.072	2.6
	mux	0.247	8.8
	adder1	0.492	17.6
controller		0.15	5.4
Total		2.794	100 %

Table 5. Hardware performance comparison of SHA-256 function

Reference	Function	Platform	Hardware size	Freq. (MHz)	Clock cycles	Throughput (Mbps)
This work	SHA-256	0.25μm ASIC	8,588 gates	136	490	142.1
		Virtex-2 xc2v2000	779 slices	71.5		74.7
[8]	SHA-256	Virtex xcv300E-8	1,261 slices	88	73	617
[9]	SHA-1	xcv300	2,606 slices	37		233
	SHA-256	Virtex xcv200	1,060 slices	83		326
[10]	SHA-1	Virtex xcv1000-6	2,200 slices	75.76	84	462
[11]	SHA-1	APEX EP20K1000E	10,573 logic elements	18	81	114
[12]	SHA-1	0.6μm ASIC	10,900 gates + RAM	59	255	119
	SHA-256				392	77
	SHA-1	Virtex xcv300E	2,008 slices	42.9	255	86
	SHA-256				392	56
[13]	SHA-1	Virtex-2 xc2v3000	1,550 slices	38.6	22	899.8
[14]	SHA-256	0.18μm ASIC	22,000 gates	200	65	1,575
		Virtex II-6	849 slices+1 BRAM	87	65	685
[15]	SHA-1	0.13μm ASIC	9,859 gates	333.3	85	2,006
	SHA-256		15,329 gates		72	2,370
[16]	SHA-256	Virtex-2 xc2v2000	1,373 slices	133.06	68	1,009
[17]	SHA-256	Virtex-2 xc2v2000	797 slices+1 BRAM	150	65	1,184
[18]	SHA-256	Virtex xcv200	1,306 slices	77	66	597
[19]	SHA-256	Virtex-2 xc2v2000	1,938 slices	81	32	1,296
[20]	SHA-256	Virtex v200pq240	2,120 slices	74	65	582
[21]	SHA-1	0.18μm ASIC	14,500 gates	350	82	2,185
	SHA-256		20,500 gates	280	66	2,172
[22]	SHA-256		50,000 gates	150		1,280

The total power consumption of the proposed design at 25 MHz is about 2.8 mW. This power estimation have been performed by Synopsys Power Compiler. Unfortunately, there exists no prescription nor recommendation concerning to the preferred power consumption of TPM chip integrated into mobile device. Therefore, as an indirect way to suggest a reasonable level of TPM consuming power suitable for mobile devices, in this paper, we refer to the power consumption of the Universal Subscriber Identity Module (USIM) embedded in the mobile phone. According to the USIM specification [23], the maximum allowable current for Class B type is about 50 mA at the 5 MHz. Since both TPM and USIM chips provide a similar quality of value-added security functions for mobile devices, it is reasonable that the recommended maximum current, which the TPM chip can consume in mobile device, is the same 50mA as for USIM chip.

Notice that the proposed SHA-256 circuit consumes only 2.2%(1.1 mA) of the maximum allowable current of the mobile TPM chip. Since SHA-256 is the primary TPM function consuming very small portion of allowable power, it can be safely declared that the proposed architecture is suitable to the mobile platform in viewpoint of energy consumption.

In Table 5, we show the comparison table of the proposed design and the previous works of the SHA-256 implementations. It is clear from the table that the proposed architecture trades speed for compact area and our implementation uses the minimum hardware resources among all kinds of published designs.

5 Conclusions

This paper presents a folded architecture of SHA-256 hardware implementation for trusted mobile computing. The presented architecture is a highly effective to implement the message schedule and message compression in SHA-256 algorithm with a minimum resource usage. The presented architecture has a chip area of 8,588 gates and has a current consumption of $1.1mA$ at the frequency of 25MHz.

Among all of published SHA-256 implementations, the proposed architecture turns out to be implemented in the smallest area in terms of logic gates. In summary, the combined performance results of power consumption, throughput, and functionality strongly indicate that our SHA-256 hardware is suitable for mobile trusted computing devices and other low-end embedded systems that urge for high-performance and small-sized solutions.

Acknowledgements

This work was supported by the IT R&D program of the Ministry of Knowledge Economy, Korea. [2006-S-041-02, "Development of a common security core module for supporting secure and trusted service in the next generation mobile terminals"]

The second author of this research was supported by the Ministry of Knowledge Economy, Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Advancement) (IITA-2008-C1090-0801-0016) .

References

1. Trusted Mobile Platform NTT DoCoMo, IBM, Intel, Trusted Mobile Platform: Hardware Architecture Description Rev1.0, Trusted Computing Group (2004)
2. Trusted Computing Group, Trusted Module Library: Commands and Structures, Specification version 0.7, Level 1 Revision 030 (November 28, 2007)
3. Ravi, S., Raghunathan, A., Porlapally, N.: Securing wireless data: system architecture challenges. In: Proccedings of the 15th International Symposium on System Synthesis, pp. 195–200 (2002)
4. MPC860 Product Summary, `http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MPC860`
5. ARM7 Product Summary, `http://www.arm.com/products/CPUs/families/ARM7family.html`
6. Raghunathan, A., Ravi, S., Hattangady, S., Quisquater, J.: Securing Mobile Appliances: New Challenges for the System Designer. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE 2003) (2003)
7. SHA-2 Standard, NIST. Secure Hash Standard, FIPS PUB 180-2 (2002), `www.itl.nist.gov/fipspubs/fip180-2.htm`

8. Ting, K.K., Yuen, S.C.L., Lee, K.-H., Leong, P.H.W.: An FPGA based SHA-256 processor. In: Glesner, M., Zipf, P., Renovell, M. (eds.) FPL 2002. LNCS, vol. 2438, pp. 577–585. Springer, Heidelberg (2002)
9. Sklavos, N., et al.: On the Hardware Implementations of the SHA-2 (256, 384, 512) Hash Functions. In: Proc. ISCAS 2003, May 2003, vol. V, pp. 153–156 (2003)
10. Grembowski, T., Lien, R., Gaj, K., Nguyen, N., Bellows, P., Flidr, J., Lehman, T., Schott, B.: Comparative analysis of the hardware implementations of hash functions SHA-1 and SHA-512. In: Chan, A.H., Gligor, V.D. (eds.) ISC 2002. LNCS, vol. 2433, pp. 75–89. Springer, Heidelberg (2002)
11. Kang, Y.K., et al.: An Efficient Implementation of Hash Function Processor for IPsec. In: Proc. 3rd Asia-Pacific Conference on ASICs, pp. 93–96 (August 2002)
12. Dominikus, S.: A Hardware Implementation of MD4-Family Hash Algorithms. In: Proc. ICECS 2002, vol. III, pp. 1143–1146 (September 2002)
13. Diez, M.J., et al.: Hash algorithm for cryptographic protocols: FPGA implementation. In: Proc. TELFOR 2002, Belgrade, pp. 26–28 (November 2002)
14. Helion IP Core Products, Helion Technology, `http://www.heliontech.com/core.htm/`
15. Satoh, A., Inoue, T.: ASIC-Hardware-Focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS. In: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC 2005), pp. 532–537 (2005)
16. McEvoy, R.P., Crowe, F.M., Murphy, C.C., Marnane, W.P.: Optimisation of the SHA-2 family of hash functions on FPGAs. In: Proceedings of the 2006 Emerging VLSI Technologies and Architectures (ISVLSI 2006) (2006)
17. Chaves, R., Kuzmanov, G., Sousa, L., Vassiliadis, S.: Improving SHA-2 hardware implementations. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 298–310. Springer, Heidelberg (2006)
18. Glabb, R., et al.: Multi-mode Operator for SHA-2 Hash Functions. Journal of Systems Architecture: the EUROMICRO Journal 53(2-3), 127–138 (2007)
19. Zeghid, M., et al.: A Reconfigurable Implementation of the New Secure Hash Algorithm. In: Second International Conference on Availability, Reliability and Security (ARES 2007), pp. 281–285 (2007)
20. Sklavos, N., Koufopavlou, O.: Implementation of the SHA-2 hash family standard using FPGAs. The Journal of Supercomputing 31(3), 227–248 (2005)
21. CAST Encryption Core Products, CAST Inc., `www.cast-inc.com`
22. SB-SHA2-P2 SHA2 Hardware IP core, Certicom, `www.certicom.com`
23. ETSI TS 102.221: UICC-Terminal Interface; Physical and Logical Characteristics