electronics

Article

An Efficient and Low-Power Design of the SM3 Hash Algorithm for IoT

Xin Zheng [1], Xianghong Hu [1], Jinglong Zhang [1], Jian Yang [1,*] and Shuting Cai [1] and Xiaoming Xiong [1,2]

1 School of Automation, Guangdong University of Technology, Guangzhou 510006, China; xinzheng9209@gmail.com (X.Z.); xianghong_hu@163.com (X.H.); zjl_jesse@163.com (J.Z.); shutingcai@gdut.edu.cn (S.C.); xmxiong@gdut.edu.cn (X.X.)
2 Company of Chipeye Microelectronics Foshan Ltd., Foshan 528200, China
* Correspondence: yj@gdut.edu.cn; Tel.: +86-135-6022-7290

Received: 5 August 2019; Accepted: 11 September 2019; Published: 14 September 2019

Abstract: The Internet-of-Things (IoT) has a security problem that has become increasingly significant. New architecture of SM3 which can be implemented in loT devices is proposed in this paper. The software/hardware co-design approach is put forward to implement the new architecture to achieve high performance and low costs. To facilitate software/hardware co-design, an AHB-SM3 interface controller (AHB-SIC) is designed as an AHB slave interface IP to exchange data with the embedded CPU. Task scheduling and hardware resource optimization techniques are adopted in the design of expansion modules. The task scheduling and critical path optimization techniques are utilized in the compression module design. The proposed architecture is implemented with ASIC using SMIC 130 nm technology. For the purpose of comparison, the proposed architecture is also implemented on Virtex 7 FPGA with a 36 MHz system clock. Compared with the standard implementation of SM3, the proposed architecture saves the number of registers for approximately 3.11 times, and 263 Mbps throughput is achieved under the 36 MHz clock. This design signifies an excellent trade-off between performance and the hardware area. Thus, the design accommodates the resource-limited IoT security devices very well. The proposed architecture is applied to an intelligent security gateway device.

Keywords: the cryptographic algorithm; SM3; AHB-SIC; optimization

1. Introduction

Information security plays a very important role [1,2]. Although IoT technology provides many benefits, it also brings in various security threats such as the attack vulnerability of the hard-coded security key and the user privacy information leaking problem [3,4]. A lot of previous work has been done to mitigate and address potential security threats. For example, the static or dynamic analysis of the firmware and the source code running on IoT devices is used to discover and cope with the potential vulnerabilities for IoT devices [5]. An Interference Mitigation Risk Aware (IMRA) framework is proposed to settle the problem of mitigating the interference imposed by the intruders so as to protect the proper operation of passive RFID networks [6]. There are also studies focusing on designing new lightweight algorithms or optimizing the original cryptography algorithms [7,8]. A novel tiny symmetric encryption algorithm (NTSA) providing enhanced security for the transfer of text files through the IoT network by introducing additional key confusions dynamically for each round of encryption is proposed in [9]. In [10], a Function-based Access Control scheme in IoT (IoT-FBAC) is presented, and it uses an Identity-based Encryption (IBE) scheme. The data masking and the encryption algorithm are utilized in many proposed solutions to protect sensitive information, but these solutions reduce the availability of original data and increase the time delay [4].

In all the above-mentioned work, the hash function, which belongs to the one-way encryption algorithms that compress the message with arbitrary length to the digest with fixed-length [11], is usually applied. The typical hash algorithms include MD5, SHA-0, SHA-1, SHA-2, SHA-3 and SM3 [12–16]. Since the SHA-0, SHA-1 and MD5 have not been secure enough, we will not discuss them here. Table 1 shows a comparison between SHA-2, SHA-3, and SM3 [17]. In order to ensure the security of the hash algorithm, the digest size should not be too short, as shown in Table 1.

Table 1. Comparison of Hash Algorithms (sizes and security are specified in bits).

Algorithms	**SHA-224**	**SHA-256**	**SHA-384**	**SHA-512**	**SHA3-224**	**SHA3-256**	**SHA3-384**	**SHA3-512**	**SM3**
Message size	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$	–	–	–	–	$<2^{64}$
Block size	512	512	1024	1024	1152	1088	832	576	512
Digest size	224	256	384	512	224	256	384	512	256
Security	112	128	192	256	112	128	192	256	128

SM3 is a hash algorithm published by the Security Commercial Code Administration Office of China in 2010 and recognized by the ISO/IEC international standard [11,18]. SM3 can be used for a digital signature and identity authentication in different applications, and for the IoT application, SM3 should be designed to possess the characteristics of high performance, low power and low costs. The structure of SM3 is similar to that of SHA256. Recently, low power and efficient implementation of the hash algorithm have become an active topic. A software implementation of SHA-3 on the Intel Core-i5 and Cavium Networks Octeon embedded platform is presented in the work of [19]. It has more flexibility and uses less resources, but runs with a lower rate. instead of commonly used shift registers, a compact hardware implementation of SM3 using SRAM to do message expansion is proposed in article [12]. This method has higher security and speed, but uses more hardware resources and has low flexibility. In article [16], four new different hardware architectures are proposed to improve the performance of SHA-256, reducing the critical path by reordering some operations required at each iteration of the algorithm and computing some values in advance. This work has a significant improvement in performance, but it is not suitable for IoT devices because huge hardware resources are required, and recently, the software/hardware co-design methods are adopted to implement the hash function, which can achieve an excellent trade-off between performance and costs. For instance, in article [13], authors adopt the software/hardware co-design method to implement SM3, which is applied to the financial IC card. Moreover, the different optimized SM3 architecture and methods, such as the 3-stage pipeline approach, the parallel implementation strategy, the optimized architecture adopted the CSA adder, the implementation with the embedded ARM core, etc., are proposed [20–25]. Previous work promotes the development of SM3 VLSI architecture which has the characteristics of small hardware area and low power consumption.

To balance the performance and hardware resources of a cryptographic system for IoT devices, we adopt the software/hardware co-design implementation method to accommodate both the high complexity of computation and flexibility of the algorithm. The optimized algorithm implementation and the AHB-SIC interface IP are also proposed to make the design more efficient and easier-to-use in IoT applications.

In this paper, the followings are made:

- New overall implementation architecture is proposed, which includes the embedded CPU, AHB-SIC, the SM3 circuit module and other modules. The AHB-SIC is designed to easily convert SM3 modules with the non-standard interface into the standard AHB slave interface. The SM3 circuit module is implemented by the software/hardware co-design method to enhance flexibility and reduce the hardware resource.

- Task scheduling and hardware resource optimization methods are applied in the expansion process to reduce the hardware area and power consumption so as to improve overall performance of SM3 implementation. The task scheduling and critical path optimization techniques are also applied in compression module design to reduce time delay and improve efficiency. The identical controller is shared in both expansion and compression modules to simplify the control circuit and reduce the hardware overheads.
- The proposed architecture is implemented on FPGA and ASIC, and also applied to an intelligent gateway. Combined with the other cryptographic modules, our proposed SM3 module can realize the digital signature and identity authentication to protect the security of user data. This framework can also be integrated into other IoT devices.

The rest of this paper is organized as follows: The mathematical background of SM3 is described in Section 2. The system architecture of the design is presented in Section 3. The theoretical analysis and experiment results are shown in Section 4. An example of IoT application of proposed design is given in Section 5. There is a conclusion about this paper in Section 6.

2. SM3 Background

Given the message with length of l ($l < 2^{64}$) bits, the SM3 hash algorithm maps it to produce a message digest of 256 bits. The procedure consists of message padding and parsing, expansion and compression [11].

2.1. Padding and Parsing

Message padding and parsing extend the length of the original message to an integer multiple of 512, and then parse the message into 512-bit blocks.

Suppose the length of message m is l bits. First, add the bit $'1'$ to the end of the message, and then add k bits $'0'$, where k is the smallest non-negative integer that satisfies $l + 1 + k = 448 \ mod\ 512$. Then, add a 64-bit bit string which is binary representation of length l. If $k = 0$, we simply need to pad a bit $'1'$ and the binary length l. The length of the padding message m' is a multiple of 512. After that, the padding message is then parsed into n 512-bit blocks which are denoted as $B^{(0)}, B^{(1)}, ..., B^{(n-1)}$. The processes of padding and parsing on disparate lengths of message are shown in Figure 1.

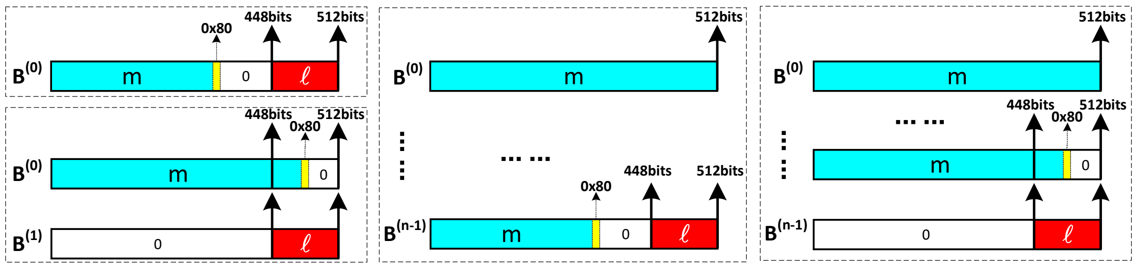


Figure 1. Padding and parsing on different lengths of messages.

2.2. Expansion

This operation expands each block message $B^{(i)}$ to generate 132 words W_0, W_1, ..., W_{67}, W'_0, W'_1, ..., W'_{63} for the compression function (CF) as follows:

- Divide the message $B^{(i)}$ into 16 words $W_0, W_1, ..., W_{15}$. The size of each word is 32-bit.
- FOR i = 16 to 67

$$W_i = P_1(W_{i-16} \oplus W_{i-9} \oplus (W_{i-3} \lll 15)) \oplus (W_{i-13} \lll 7) \oplus W_{i-6} \quad (1)$$

ENDFOR

In the Equation (1), $\oplus$ and $\lll$ represent the bitwise XOR and ROL operations, respectively; $P_1(X) = X \oplus (X \lll 15) \oplus (X \lll 23)$ is the permutation function.

- FOR $i = 0$ to 63

$$W'_i = W_i \oplus W_{i+4} \tag{2}$$

ENDFOR

2.3. Compression

The compression is the core operation of SM3. The grouped message $m' = B^{(0)}, B^{(1)}, ..., B^{(n-1)}$, where $n = (l + k + 65)/512$. Iterate m' as follows.

FOR $i = 0$ to $n - 1$

$$V^{(i+1)} = CF(V^{(i)}, B^{(i)}) \tag{3}$$

ENDFOR

In the Equation (3), $V^{(0)}$ is the 256-bit initial value IV, and the result of iterative compression is $V^{(n)}$. Eight registers denoted as $\boldsymbol{A, B, C, D, E, F, G}$, and $\boldsymbol{H}$ are used in the compression function CF. The size of each register is a word. Let $SS1, SS2, TT1$, and $TT2$ be the middle variables, and the iteration of the CF process is described as follows.

$$\begin{aligned}
&ABCDEFGH = V^{(i)} \\
&FOR\ j = 0\ to\ 63 \\
&\qquad SS1 = ((A \lll 12) + E + (T_j \lll j)) \lll 7,\ SS2 = SS1 \oplus (A \lll 12) \\
&\qquad TT1 = FF_j(A, B, C) + D + SS2 + W'_j,\ TT2 = GG_j(E, F, G) + H + SS1 + W_j \\
&\qquad D = C,\ C = B \lll 9,\ B = A,\ A = TT1,\ H = G,\ G = F \lll 19,\ F = E,\ E = P_0(TT2) \\
&ENDFOR \\
&V^{(i+1)} = ABCDEFGH \oplus V^{(i)}
\end{aligned} \tag{4}$$

The registers $\boldsymbol{A}$ to $\boldsymbol{H}$ are initialized with IV for the first message block, where IV = 7380166f, 4914b2b9, 172442d7, da8a0600, a96f30bc, 163138aa, e38dee4d, b0fb0e4e. $\vee$, $\wedge$ and $\neg$ represent the bitwise OR, AND and NOT operations, respectively. The permutation function is represented as $P_0(X) = X \oplus (X \lll 9) \oplus (X \lll 17)$. The constant T_j and the Boolean functions FF_j and GG_j are given in Table 2.

Table 2. The value of the constant and functions.

Value Range	T_j	FF_j	GG_j
$0 \leqslant j \leqslant 15$	79cc4519	$X \oplus Y \oplus Z$	$X \oplus Y \oplus Z$
$16 \leqslant j \leqslant 63$	7a879d8a	$(X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$	$(X \wedge Y) \vee (\neg X \wedge Z)$

After computing the result of the last group, we obtain $ABCDEFGH = V^{(n)}$, and the final 256-bit hash value is $y = ABCDEFGH$.

3. Proposed Architecture

3.1. Overall Implementation Architecture

In this section, overall implementation architecture, which includes the embedded CPU, AHB-SIC, the SM3 circuit module and other modules as shown in Figure 2, is proposed. The embedded CPU is the master of this IoT SoC in which the software design is implemented. The structure of AHB-SIC is simple and easy to use. The non-standard interface of the cryptographic module can be quickly converted to the AHB slave interface, so as to realize the SoC design effectively. As the slave module of proposed architecture, the SM3 circuit module consists of the expansion module, the compression module, as well as the controller and the result module. In our design, the controller and the data path are separated. The former is chiefly used to control the execution process of the circuit and provide control signals. Composed of several sub-circuit modules, the latter mainly implements SM3 encryption functions, and the result module reads and outputs the result in the form of 32-bit. Besides, the other modules cover RAM, ROM, and other cryptographic modules or interface modules.

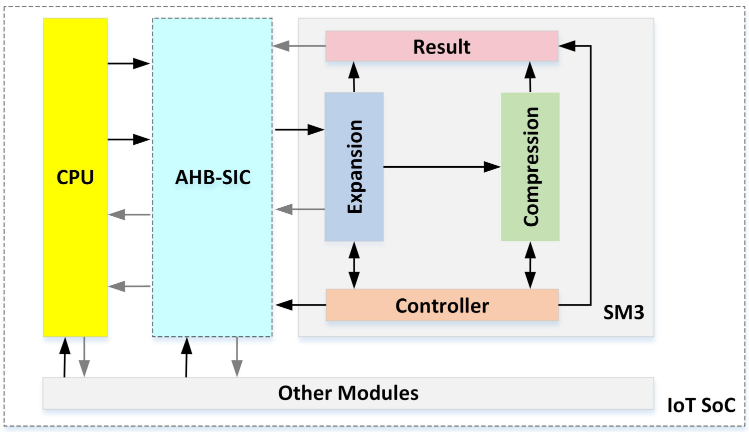


Figure 2. The overall architecture of IoT SoC.

3.2. The AHB-SM3 Interface Controller

The data interaction between software and hardware is controlled by the AHB-SIC we proposed. The AHB-SIC is composed of AHB Bus Interface Control Logic (BICL) and four function registers. In our design, the master is the embedded CPU, and the slave is the SM3 circuit modules. Data transfer between the master and the slave is through the AHB-SIC. The BICL is made up of control logic, the address decoder, the data distributor and multiplexer. According to the different types of signals, we design four kinds of function registers (the input register, the output register, the control register and the status register) to realize the data interaction with SM3. The input signals of SM3 consist of write control signal W, read control signal R and 512-bit input data DIN. The output signals of SM3 incorporate status signal $STATE$, finish signal $FINISH$, and 256-bit hash value $DOUT$. The block diagram of AHB-SIC structure is shown in Figure 3.

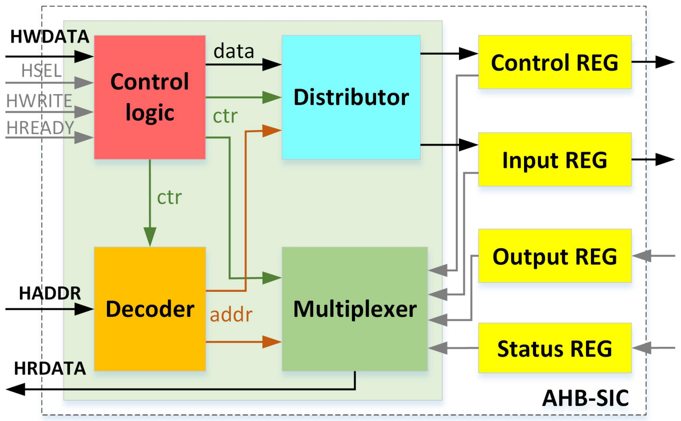


Figure 3. The block diagram of AHB-SIC structure.

The signals of the AHB bus include HADDR, HWDATA, HRDATA, HSEL and so on [26]. The AHB protocol sequence under the basic transmission mode is shown in Figure 4. In the read mode, once the master drives the address and HWRITE signals after the first rising edge, the slave will send data on the second rising edge, and the master will sample the data on the third rising edge. While in the write mode, the master drives the address and HWRITE signals after the first rising edge and outputs the corresponding data on the second rising edge; the slave will sample the address and data on the second and third rising edge, respectively.

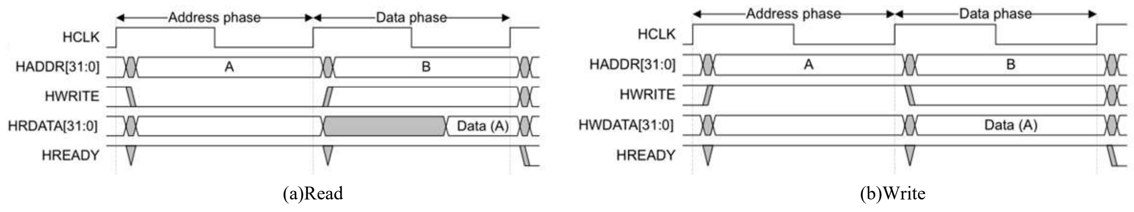


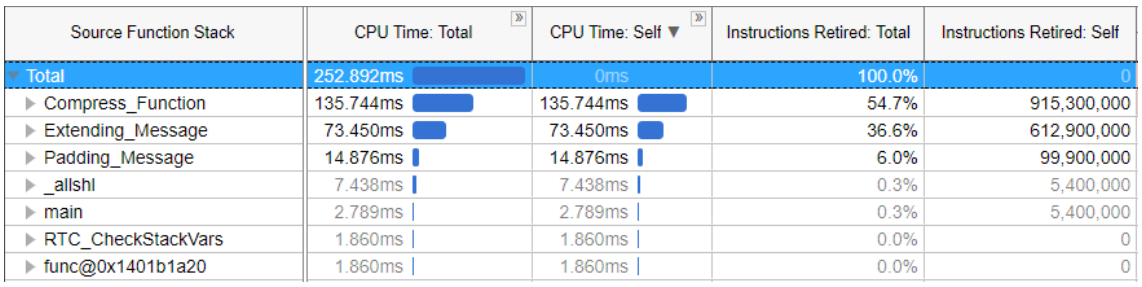
Figure 4. The AHB protocol sequence under the basic transmission mode. (a) Read; (b) Write.

BICL is designed based on AHB bus timing, which mainly implements the data transfer between the master and four sorts of function registers. As seen from the Figure 3, the control logic reads disparate signals on the AHB bus and generates control signals through internal logic to control the address decoder, the data distributor and multiplexer. The four function registers are primarily used for calculation, control, data interaction and the status flag, which can flexibly control the cryptographic hardware modules and obtain their current status for software debugging. Finally, combining a 32-bit low-power embedded CPU, we adopt the AHB-SIC to integrate the SM3 module into an SoC.

3.3. SM3 Implementation

3.3.1. Software/Hardware Co-Design

Before the design of the SM3 circuit, we firstly use C/C++ to implement SM3 on the platform of CPU I7-8700k@3.2 GHz. Figure 5 shows the CPU time consuming of each part of SM3, and it is the basic hotspot analysis result which is obtained via using the VTune Amplifier XE tool to list the most active (most time-consuming) functions in our application [27]. As shown in Figure 5, the expansion and compression operations occupy over 90% of the total run time. If we implement the expansion and compression module in software, the encryption efficiency will be very low. Besides, considering the variable length of message, we find it difficult to store whole message, and it will also consume a lot of hardware resources if padding and parsing are designed in hardware. Thus, in our design, the padding and parsing operations are implemented in software, and expansion and compression operations are implemented in hardware. After the software/hardware partitioning, the resulting architecture of the SM3 circuit can be determined.



Source Function Stack	CPU Time: Total	CPU Time: Self ▼	Instructions Retired: Total	Instructions Retired: Self
Total	252.892ms	0ms	100.0%	0
Compress_Function	135.744ms	135.744ms	54.7%	915,300,000
Extending_Message	73.450ms	73.450ms	36.6%	612,900,000
Padding_Message	14.876ms	14.876ms	6.0%	99,900,000
_allshl	7.438ms	7.438ms	0.3%	5,400,000
main	2.789ms	2.789ms	0.3%	5,400,000
RTC_CheckStackVars	1.860ms	1.860ms	0.0%	0
func@0x1401b1a20	1.860ms	1.860ms	0.0%	0

Figure 5. The CPU time consuming of each part of SM3.

3.3.2. Controller Design

A standard three-stage Finite State Machine (FSM) structure is adopted in the controller module to generate and provide control signals ($ctrl$, $finish$ and $countout$ signals) [28]. The three-stage FSM structure can remove the glitch, which is more conducive to optimize the code and facilitate the user to set appropriate timing constraints. The $ctrl$ is the process state signal which contains four states: $idle$, $write$, $encryption$ and $read$.

In the $idle$ state, if the input signal W is set to 1, SM3 enters the $write$ state and writes data from DIN to the data register. When the data encryption is completed or the last 32-bit data is read, $finish$ will be set to 1. In this case, if R equals to 1, the module enters the $read$ state and outputs the result of the current operation in a 32-bit format. It should be noted that each time 512-bit data has been encrypted or once the 256-bit result has been read, the state returns to be $idle$, and the initialization operation is performed to ensure the precision of the data in the next encryption process. If R is equal to 0, the module retains the current operation result and enters the $idle$ state to wait for the next block of data. Besides, $countout$ is the output control signal of the counter which is used to control the process of other modules, such as expansion, compression and the entire encryption iteration process. The FSM state transition diagram is shown in Figure 6.

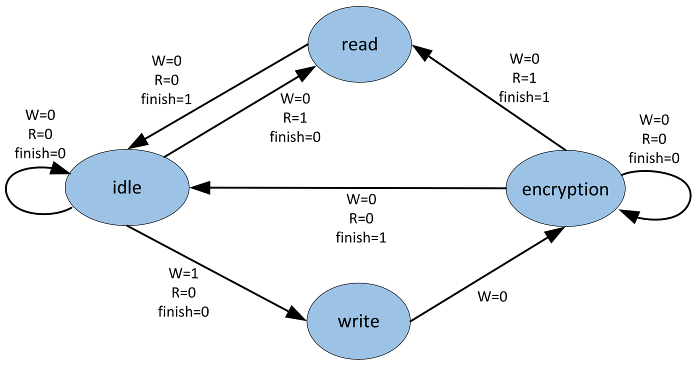


Figure 6. The FSM state transition diagram.

3.3.3. Expansion Module Design

For the part of expansion, the length of padding message m' is assumed to be 512 bits. We analyze the expansion module in the standard method first. Conventionally, the 32 bits divided message $W_0, W_1, ..., W_{15}$ are sequentially stored in 16 registers $\boldsymbol{W_0}, \boldsymbol{W_1}, ..., \boldsymbol{W_{15}}$ in 16 clock cycles. When the rising edge of each cycle comes, a set of data is written into the corresponding register. Starting with the 17th clock, we calculated Equation (1) given in Section 2 when the rising edge of the clock comes, and a new register $\boldsymbol{W_{16}}$ would be set to store the calculated value. That is, the new register $\boldsymbol{W_{16}}$ is set at the 17th clock and its value is calculated. Until the 68th clock, the W_{67} is updated. After completing the calculation of Equation (1) given in Section 2, we obtain the 68 words $W_0, W_1, ..., W_{67}$ are obtained, which takes 68 clock cycles. If we calculate W'_j after calculating all the W_i $(i = 0, 1, ..., 67)$ in order, 64 new registers are used to store $\boldsymbol{W'_0}, \boldsymbol{W'_1}, ..., \boldsymbol{W'_{63}}$ and 64 clock cycles are consumed. At this point, a total of 68 + 64 = 132 clock cycles and registers are adopted.

In order to reduce the hardware resources and enhance the performance of the SM3, task scheduling and register optimization are adopted in this work. The proposed design of expansion is shown in Figure 7. In line with the principle analysis of the SM3 algorithm, it can be known that we can get the data required for the second 64-round iterations directly instead of waiting until the first 68-round iterations are completed. It only needs to start computing after the 4th round iteration of the first 68-round iterations. That is to say, the second 64-round iterations can be nested into the first 68-round iterations and begin with its fifth iteration. In this way, merely 68 rounds of iterations or 68 cycles are required. Since the register will bring a lot of areas and power consumption overheads, in order to further reduce the power consumption, the register multiplexing method is adopted to optimize and reduce the number of registers.

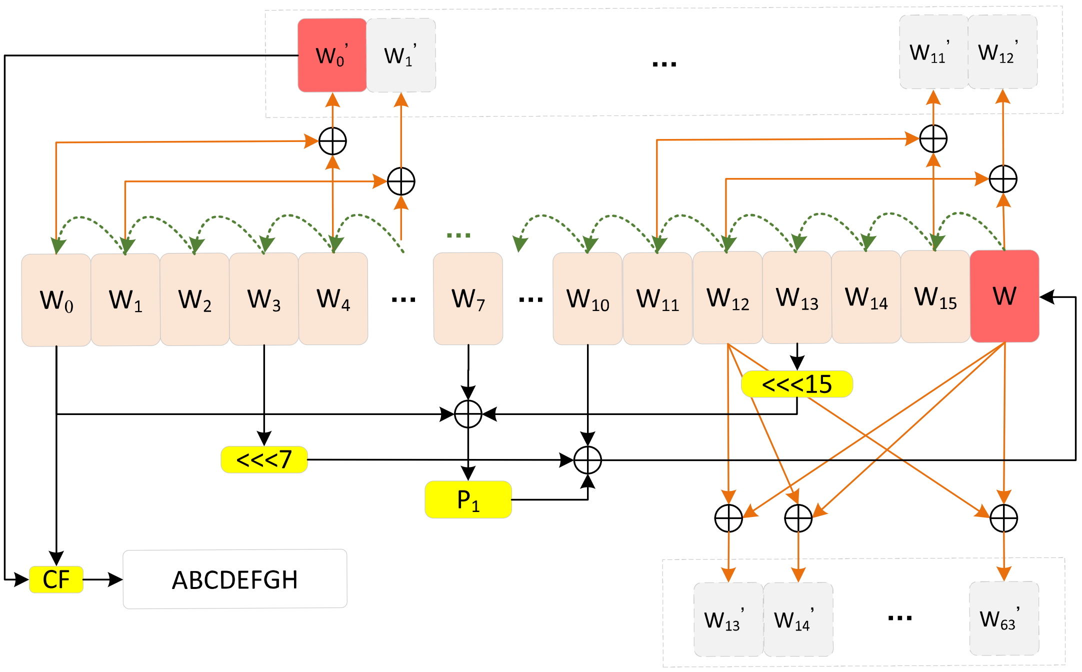


Figure 7. The proposed design of expansion.

For updating W_i, we still use the registers $\boldsymbol{W_0}, \boldsymbol{W_1}, ..., \boldsymbol{W_{15}}$ to store the expanding message m', and then set a register $\boldsymbol{W}$ to store the current value W_i. The value of W_i is then assigned to W_{i-1} after each round of calculation. For example, after calculating W_{16}, the result is directly assigned to $\boldsymbol{W}$. Then, the values of all the registers are assigned to the previous register when W_{17} is computed. The value of $\boldsymbol{W}$ is assigned to $\boldsymbol{W}_{15}$; the value of $\boldsymbol{W}_{15}$ is assigned to $\boldsymbol{W}_{14}$; the value of $\boldsymbol{W}_0$ is discarded. This method can reduce the original 68 registers to merely 17 registers ($\boldsymbol{W_0}, \boldsymbol{W_1}, ..., \boldsymbol{W_{15}}, \boldsymbol{W}$) without affecting the calculation results. Since not all the W_i is necessary to be retained, some intermediate variables are not saved. Y and $P1$ are directly involved in the calculation as intermediate variables, and do not need to be saved in the register. For updating W'_j, it can be seen from the Equation (2) given in Section 2 that the calculation result of W'_j is only related to W_i. After the registers are simplified in the previous step, W_i simply keeps the current calculated value and value of 16 adjacent words. Therefore, the calculation of W'_j must be performed when the corresponding W_i is reserved. For example, when the value of W'_0 is calculated, W_0 and W_4 are required. The value of W_4 is assigned at the 4th cycle, and in the next cycle, the value of W_5 is read, and W'_1 can be calculated. At the 15th cycle, the values of $W_0, W_1, ..., W_{15}$ are all read, and the value of W'_{11} is calculated. Starting from the 16th cycle, we find that W_i begins to store the result in the register of $\boldsymbol{W}$. The calculation of W'_j only needs to use the value of W_j and W_{j+4} calculated by the current clock cycle. In this case, the calculated value (W_{j+4}) of the current clock cycle is always stored in the register $\boldsymbol{W}$. Thus, we adopt the register reduction method to obtain W_j. Starting from the 16th clock cycle, we find that the values of $\boldsymbol{W_0}, \boldsymbol{W_1}, ..., \boldsymbol{W_{15}}$ will shift to the left every cycle. From the beginning of W'_{12}, each subsequent calculation of W'_j is $W'_j = W_{12} \oplus W$ ($j \geqslant 12$).

3.3.4. Compression Module Design

The compression module is the core of SM3, and its main functions are the implementation of the iteration and compression which produce most of the critical path delay. The iterative process of the SM3 covers logical operations such as XOR, ROL, and addition operations (ADD). The delay of XOR or ROL is almost negligible, while the addition operation has a great delay effect [29]. Hence, during each iteration, the maximum number of serial additions determines the size of delay. We define the calculation path with the largest number of serial addition operations as the critical path. In order to reduce the critical path delay and the number of registers, we adopt the task scheduling and critical path optimization. The proposed design of compression is shown in Figure 8.

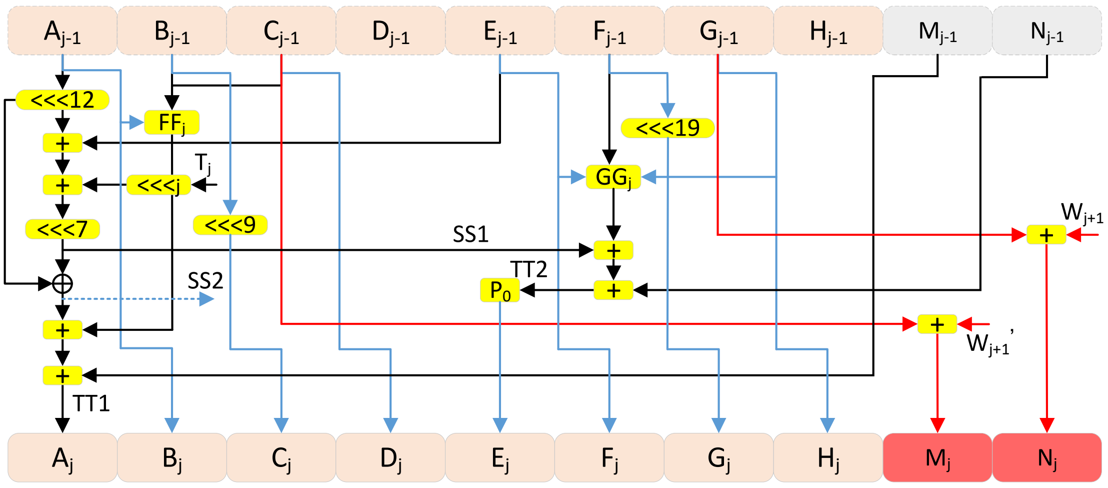


Figure 8. The proposed design of compression.

As seen from Figure 8, the calculation paths of A and E require five additions to complete. Therefore, the calculation paths of A and E are the critical paths. The total delay of one iteration is the sum of the delays of five additions. Obviously, the compression process requires a 64-round loop iteration, which consumes another 64 cycles and the total delay will be large. In our design, two novel registers $\boldsymbol{M}$ and $\boldsymbol{N}$ are added to reduce one addition operation of A and E, and we let $M_j = D_j + W'_{j+1}$, $N_j = H_j + W_{j+1}$. Set the initial value of $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{E}, \boldsymbol{F}, \boldsymbol{G}, \boldsymbol{H}, \boldsymbol{M}$, and $\boldsymbol{N}$ to be $A_{-1}, B_{-1}, C_{-1}, D_{-1}, E_{-1}, F_{-1}, G_{-1}, H_{-1}, M_{-1}$, and N_{-1}, respectively. Since the intermediate variable M and N are introduced, the initial addition operations of $M_{-1} = D_{-1} + W'_0$ and $N_{-1} = H_{-1} + W_0$ need to be implemented before the start of the iterative process. Thus, one clock cycle is consumed before the compression iteration. It is noteworthy that the calculation of M_j and N_j is parallel with that of A_j and E_j in the critical path. This parallel computing will expedite the compression in SM3. The compression optimization algorithm is shown in Figure 9.

Through analyzing Equation (4) given in Section 2, we figure out that the calculation of TT_1 only needs the value of W'_j in the current clock cycle. Thus, W'_j and TT_1 can be simultaneously calculated. In other words, compression can be nested within the expansion process in advance and it can start with the sixth iteration of expansion, which finally adds just two more clock cycles. Besides, by task scheduling optimization, the expansion and compression modules can share the control signals generated by the same controller, which reduces the complexity and overheads of the control circuit. The result module mainly realizes the parallel conversion output from the 256-bit to 32-bit format and controls the conversion process by the counter, which needs 8 cycles; after completing the conversion, we find that finish = 1, indicating that the reading is completed and the next message can be encrypted. In the end, it takes only 70 cycles to complete an expansion and compression operation.

```
The compression optimization algorithm
Input:   IV :The 256-bit initial value of $V^{(0)}$;
         $W_0, W_1, \ldots, W_{67}$: The 68-round generated words;
         $W_0', W_1', \ldots, W_{63}'$: The 64-round generated words;
         $D_{-1}, H_{-1}$: The initial value of register D, H.
Output: The final hash value y.
Procedure:
Initialize the register ABCDEFGH to IV;
Calculate the initial value of M and N;
For i = 0 to n-1
    Update the value of $V^{(i)}$ to ABCDEFGH;
    For j = 0 to 63
        Calculate the middle variables through ROL, XOR and Boolean functions;
        Calculate and update the value of ABCDEFGHMN.
    Endfor
    Update the value of ABCDEFGH to $V^{(i+1)}$;
Endfor
The final hash value y = ABCDEFGH.
```

Figure 9. The compression optimization algorithm.

4. Analysis and the Experiment Result

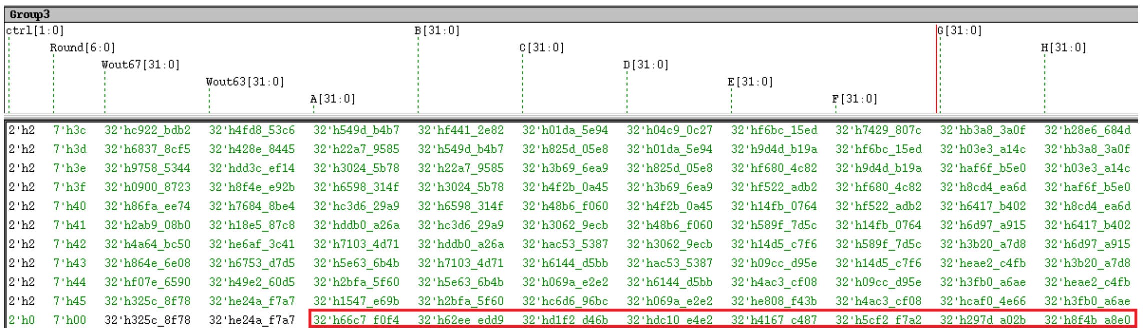
In this section, we conduct the experiments and analyze our proposed implementation including time and resource consumption analysis, computation amount analysis, analysis of comparison between pure software and other existing work.

4.1. The Setting and Implementation of the Experiment

In the experiment, the proposed architecture of SM3 described above is implemented with the C and Verilog-HDL language. The environment of Vivado 2016 and the platform of Xilinx Virtex 7 xc7vh580 are adopted to implement the architecture of SM3. Besides, the hardware module of SM3 is also synthesized by DC with the SMIC 130 nm technology process. The performance evaluation of SM3 is based on the area, power and throughput.

Firstly, we verify the function of proposed architecture on FPGA. The input message vector is set to be 'abc' which is the published example test case in [11]. The performance, power and the area of this design on FPGA are achieved in Section 4.2.

Secondly, through using the Verilog Compiled Simulator, we can obtain the result data and the waveform to check the validity of this design. The simulation results are shown in Figure 10 and Figure 11, showing the correctness of this design.



```
Group3
ctrl[1:0] Round[6:0] Wout67[31:0] Wout63[31:0] A[31:0]       B[31:0]       C[31:0]       D[31:0]       E[31:0]       F[31:0]       G[31:0]       H[31:0]
2'h2  7'h3c  32'hc922_bdb2  32'h4fd8_53c6  32'h549d_b4b7  32'hf441_2e82  32'h01da_5e94  32'h04c9_0c27  32'hf6bc_15ed  32'h7429_807c  32'hb3a8_3a0f  32'h28e6_684d
2'h2  7'h3d  32'h6837_8cf5  32'h428e_8445  32'h22a7_9585  32'h549d_b4b7  32'h825d_05e8  32'h01da_5e94  32'h9d4d_b19a  32'hf6bc_15ed  32'h03e3_a14c  32'hb3a8_3a0f
2'h2  7'h3e  32'h9758_5344  32'hdd3c_ef14  32'h3024_5b78  32'h22a7_9585  32'h3b69_6ea9  32'h825d_05e8  32'hf680_4c82  32'h9d4d_b19a  32'haf6f_b5e0  32'h03e3_a14c
2'h2  7'h3f  32'h0900_8723  32'h8f4e_e92b  32'h6598_314f  32'h3024_5b78  32'h4f2b_0a45  32'h3b69_6ea9  32'hf522_adb2  32'hf680_4c82  32'h8cd4_ea6d  32'haf6f_b5e0
2'h2  7'h40  32'h86fa_ee74  32'h7684_8be4  32'hc3d6_29a9  32'h6598_314f  32'h48b6_f060  32'h4f2b_0a45  32'h14fb_0764  32'hf522_adb2  32'h6417_b402  32'h8cd4_ea6d
2'h2  7'h41  32'h2ab9_08b0  32'h18e5_87c8  32'hddb0_a26a  32'hc3d6_29a9  32'h3062_9ecb  32'h48b6_f060  32'h589f_7d5c  32'h14fb_0764  32'h6d97_a915  32'h6417_b402
2'h2  7'h42  32'h4a64_bc50  32'he6af_3c41  32'h7103_4d71  32'hddb0_a26a  32'hac53_5387  32'h3062_9ecb  32'h14d5_c7f6  32'h589f_7d5c  32'h3b20_a7d8  32'h6d97_a915
2'h2  7'h43  32'h864e_6e08  32'h6753_d7d5  32'h5e63_6b4b  32'h7103_4d71  32'h6144_d5bb  32'hac53_5387  32'h09cc_d95e  32'h14d5_c7f6  32'heae2_c4fb  32'h3b20_a7d8
2'h2  7'h44  32'hf07e_6590  32'h49e2_60d5  32'h2bfa_5f60  32'h5e63_6b4b  32'h069a_e2e2  32'h6144_d5bb  32'h4ac3_cf08  32'h09cc_d95e  32'h3fb0_a6ae  32'heae2_c4fb
2'h2  7'h45  32'h325c_8f78  32'he24a_f7a7  32'h1547_e69b  32'h2bfa_5f60  32'hc6d6_96bc  32'h069a_e2e2  32'he808_f43b  32'h4ac3_cf08  32'hcaf0_4e66  32'h3fb0_a6ae
2'h0  7'h00  32'h325c_8f78  32'he24a_f7a7  32'h66c7_f0f4  32'h62ee_edd9  32'hd1f2_d46b  32'hdc10_e4e2  32'h4167_c487  32'h5cf2_f7a2  32'h297d_a02b  32'h8f4b_a8e0
```

Figure 10. The simulation result of this design.

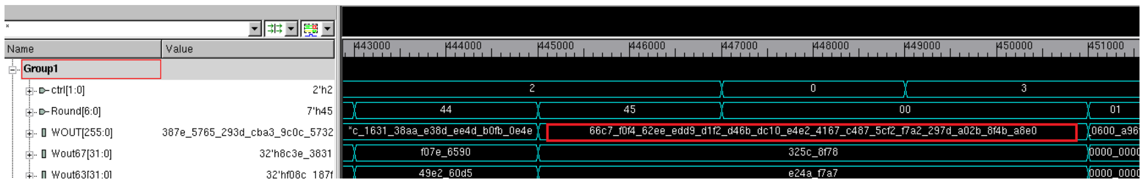

Name
Value
Group1
ctrl[1:0] 2'h2
Round[6:0] 7'h45
WOUT[255:0] 387e_5765_293d_cba3_9c0c_5732
Wout67[31:0] 32'h8c3e_3831
Wout63[31:0] 32'hf08c_187f
443000
444000
445000
446000
447000
448000
449000
450000
451000
2
0
3
44
45
00
01
*c_1631_38aa_e38d_ee4d_b0fb_0e4e
66c7_f0f4_62ee_edd9_d1f2_d46b_dc10_e4e2_4167_c487_5cf2_f7a2_297d_a02b_8f4b_a8e0
0600_a96
f07e_6590
325c_8f78
0000_0000
49e2_60d5
e24a_f7a7
0000_0000

Figure 11. The waveform result of this design.

Thirdly, we process the synthesis on DC with SMIC 130 nm technology. The slack, performance, power and the area of this design are acquired and shown in Section 4.2.

In order to further indicate the efficiency of our proposed design, we implement this design by pure software and the software/hardware co-design method on FPGA. Besides, the system clock is normalized to be 36 MHz, and the time consuming is shown in Figure 12.

After conducting the experiment and obtaining the result, we analyze the time and resource consumption, computation amount, comparison with pure software and other existing work to support our design.

4.2. Time and Resource Consumption Analysis

We analyze the time consumption and hardware resource consumption of our proposed design in theory first. Before task scheduling and register optimization, the expansion module requires 132 clock cycles to complete two iterations. The compression module requires a 64-round loop iteration and consumes another 64 cycles so that 132 + 64 = 196 cycles are consumed. If the strategy of register multiplexing has not be adopted, the expansion module requires 132 32-bit registers to store the 132 intermediate variables, and the compression module requires a total of 16 32-bit registers. Eight registers are applied to store the initial value of IV or the result of the previous 512-bit data block $B^{(i)}$, and the other 8 registers are used to store the currently obtained 256-bit data. Thus, 148 32-bit registers in total are required. After task scheduling and register optimization, only 70 cycles are needed to accomplish the expansion and compression. Besides, the expansion module needs 18 32-bit registers to store 16 32-bit intermediate data and the current two 32-bit data W_j, W'_j. The number of registers of the compression module is 18. The 16 32-bit registers are still needed, and the other 2 registers $\boldsymbol{M}, \boldsymbol{N}$ are added to reduce the critical path delay. Therefore, 36 32-bit registers in total are required.

Hence, if the time overhead of other circuits is not considered, the theoretical performance will be improved by nearly 2.8 times, and the number of registers will fall by around 3.11 times. The theoretical area overhead will decrease and eventually the power consumption will also be significantly reduced. In addition, after critical path optimization, from the DC simulation, we set the clock period to 7 ns and the input/output delay to 2 ns. The data required time is 6.90 ns, and the data arrival time is 5.59 ns. The slack is improved by 1.31 ns. The consumption comparison between the optimized and non-optimized (standard) method is shown in Table 3.

Table 3. The consumption comparison between the optimized and non-optimized method.

Operation	Clock Cycles	Register	Additions of the Critical Path	Slack
Non-optimized	196	148	5	0.0
Optimized	70	36	4	1.31

4.3. Computation Amount Analysis

Since the main operations of SM3 are concentrated on the compression function, we provide some theoretical analysis of the computational complexity. In this work, we set LOAD and STORE to represent the operations of the data load and the store, respectively. Before optimization, we need to load and store W_0 to W_{15} one time severally. From W_{16} to W_{67}, each W_i consumes 5 LOAD, 1 STORE, 6 XOR and 4 ROL; then, we calculate each $W'_i (i = 0, 1, ..., 63)$, and find that 2 LOAD, 1 STORE and

1 XOR are needed. Eventually, each iteration of compression needs 3 LOAD, 12 STORE, 8 ADD, 3 XOR, 8 ROL, 1 FF_i and 1 GG_i function. Here the FF_i function executes 2 XOR ($i = 0, 1, ..., 15$) or 3 AND and 2 OR ($i = 16, 17, ..., 63$), and the GG_i function executes 2 XOR ($i = 0, 1, ..., 15$) or 2 AND, 1 OR and 1 NOT ($i = 16, 17, ..., 63$). Based on the above theoretical analysis, the computation amount of a complete compression function is listed in Table 4.

After optimization, since the calculation of W'_j and W_{j+4} is simultaneous, only 12 LOAD and 12 STORE are consumed in the iteration of W_0 to W_{15}. From W_{16} to W_{67}, each W_i consumes 5 LOAD, 1 STORE, 6 XOR and 4 ROL; W'_j is also simultaneously calculated with $TT1$, and there is no extra LOAD and STORE consumption of W'_j. T_j is a constant so that $(T_j \lll j)) \lll 7$ can be calculated and stored in advance. Thus, an optimized compression of each iteration needs 2 LOAD, 2 STORE, 8 ADD, 3 XOR, 5 ROL, 1 FF_j and 1 GG_j. Table 4 shows that the optimized compression function reduces the times of LOAD, STORE and ROL. In theory, the optimized algorithm can improve the run time by 28.9%.

Table 4. The computation amount of the compression function.

Operation	LOAD	STORE	XOR	LOAD	ADD	AND	OR	NOT	Total
Non-optimized	596	900	632	720	512	240	144	48	3792
Optimized	400	192	632	528	512	240	144	48	2696

4.4. Comparison with Pure Software

In order to test the performance of the SM3 and AHB-SIP, we compare the time consumption between the pure software design of SM3 and our software/hardware co-design of SM3. Let input messages have different lengths, and we conduct six comparative experiments. The time consumption on FPGA under the 36 MHz clock is shown in Figure 12, where the blue line indicates the time required by the pure software method, and the red line indicates the time required by the software/hardware co-design method.

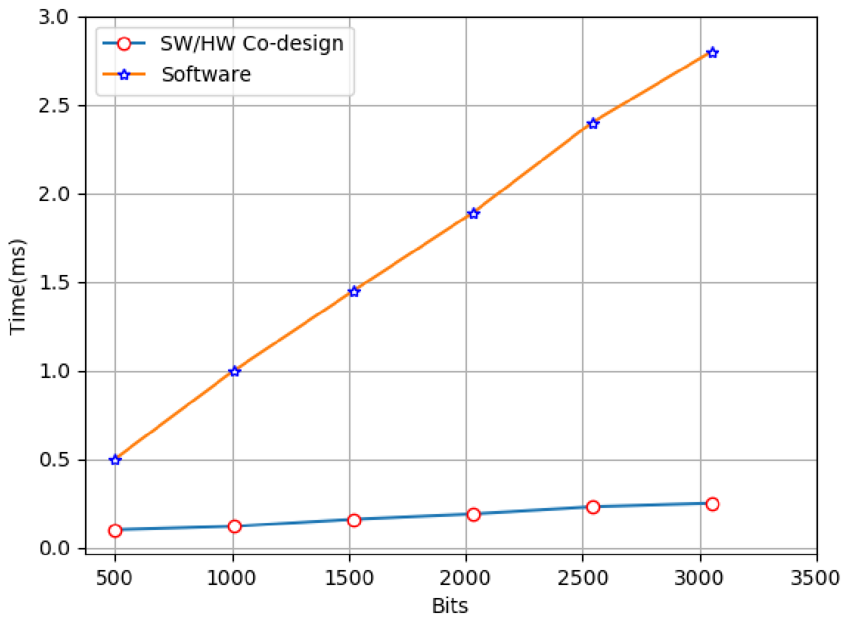


Figure 12. Time consumption on FPGA.

We can see from Figure 12 that when the message is less than 512 bits, the speed of our method is 6.8 times faster than that of the pure software method. Since padding and parsing operations are implemented by the CPU, and these two steps only need to be done once, the computing efficiency will be further increased with the length of message increasing. In theory, the computing speed can be 18.7 times faster than that of pure software.

In our experiment, the system clock frequency F = 36 MHz, which meets the requirement of low power applications. The throughput of SM3 is calculated by Equation (5):

$$V_{SM3} = \frac{F \times L}{T} \tag{5}$$

where L is the length of each block of the message and T is the clock period for one round of SM3 encryption. The ASIC implementation results are shown in Table 5. The DC synthesis results reflect that the area of SM3 hardware is 6036 gates and its throughput can be up to 263 Mbps.

4.5. Comparison with Other Related Work

We then compare our proposed architecture with other state-of-the-art hash architecture. We firstly discuss some state-of-the-art techniques. In [20], a SM3 IP design method with the high throughput rate is proposed. The 3-stage pipeline which shortens the critical path is adopted in this method, and the 64 round calculation is expanded to improve the performance. For the implementations reported in [21], the authors propose a parallel implementation strategy to reduce the delay. This method is easy to control, but throughput and efficiency are not very high. In [22], the Keyed-Hash Message Authentication Code and Secure Hash Algorithm 256 (HMAC-SHA256) is implemented, and a Trust-Based system that identifies the malicious nodes and differentiates them from trusted nodes is proposed to detect untrusted nodes. This solution is implemented in software and its throughput is not reported. In the work of [23], two different attacks on SM3 are introduced. The authors mainly studied the improvement of the resistance to attacks, but the internal details of the architecture were not reported, and thus we only discuss and compare the related implementation approaches. In [24], a Carry Save Adder (CSA) was adopted to optimize the critical path, and a dual-channel parallel adder was proposed to achieve higher throughput. In [25], authors combine the implementation with the ARM processor to enhance the throughput. Because of lack of detailed data in [21–23], no direct comparison can be made with the work.

As seen from Table 5, several different implement approaches of the hash algorithm are provided, and the results indicate that the SHA-3 design in [30] only requires 886 gates while the power and throughput are worse than ours. In [20], the proposed implementation has extremely high throughput while the area is also very large. The result shows that the throughput of our design is 1.14 times than that of design in [24] under the normalized frequency. Compared with the compact architecture implementation in [12], the saved areas with our proposed architecture are approximately 37%. In [31], the shift registers instead of the SRL are used for message expansion since the SRL is not suitable to ASIC. The compact 8-bit SHA-256 architecture in [32] has a smaller area, but the power is higher and throughput is lower compared with our architecture. It is worth noting that, the area of AHB-SIC is only 0.072 mm^2. The results indicate that our proposed implementation achieves an excellent trade-off between performance, the area and power under the normalized frequency.

Table 5. SM3 hardware performance comparison on the ASIC platform.

Design	Platforms	Frequency (MHz)	Clock Cycles	Areas (Gates)	Power (mW)	Throughput (Mbps)
This work	0.13-µm	36	70	6036	1.24	263
SHA-3 [30]	0.04-µm	28.8	3329	886	4.87	14
SM3 [20]	0.18-µm	200	1	8800k	-	105k
SM3 [24]	0.065-µm	526.3	80	5370	-	3368
SM3 [12]	0.13-µm	250	464	8277	-	276
	0.13-µm	200	68	12,956	-	1506
SM3 [31]	0.13-µm	216	68	9458	-	1619
SHA-256 [32]	0.13-µm	102	1120	9036	3.06	47

The proposed SM3 circuit structure is also implemented on the Vertex 7 FPGA platform, which shows a low power and efficient cryptographic processing performance. As shown in Table 6, there are seven different baselines compared with our proposed architecture. The SHA-256 architecture in [33] uses a three-stage pipeline implementation. The design without masking and with masking consumed 7219 and 10,918 logic cells respectively. The throughput is close to our method under the normalized frequency, but the area is larger than that of our method. In [25], the throughput of this design is only 1.09 times than that of our design while the area is 2.16 times than that of our proposed implementation. The architectures proposed in [31] are divided into the forms of compact and high throughput including the architecture of C-SM3, T-SM3 and Standard-SM3. The result shows that three architectures have a high performance, but 7.53 additional cycles are consumed. There are only 2 additional cycles consumed in our proposed method. In [34], a new SHA-3 architecture is proposed, and it simply consumes 56 slices of the area, but two extra BRAM are adopted, and the throughput of SHA-3 is not very efficient. Thus, compared with the above baseline architectures on FPGA, our proposed architecture also shows an excellent trade-off between performance and the area.

Table 6. SM3 hardware performance comparison on the FPGA platform.

Design	Platforms	Frequency (MHz)	Clock Cycles	Areas	Throughput (Mbps)
This work	Virtex-7	36	70	808 Slices	263
	Virtex-7	50	70	808 Slices	366
	Virtex-7	100	70	808 Slices	731
SHA-256 [33]	Cyclone II	116.24	68	7219 Logic Cells	875.22
	Cyclone II	87.08	68	10,918 Logic Cells	655.66
SM3 [25]	ZYNQ-7020	167.8	64	1743 Slices	1342
Standard-SM3	Virtex-5	214	7.53	384 Slices	1611
C-SM3	Virtex-5	215	7.53	234 Slices	1619
T-SM3 [31]	Virtex-5	362	7.53	328 Slices	2726
SHA-3 [34]	Virtex-5	372	846	56 Slices + 2 BRAM	225

5. An Example of IoT Application of SM3

Recently, with the rapid growth of IoT devices, the related applications of IoT have been widely researched [35–37]. For the SM3 algorithm, few IoT applications have been proposed. In [13], the design of SM3 reportedly can be applied to the IC card, but it did not provide the implementation in given application. In this section, we give an example of our proposed architecture implementation in the intelligent gateway. Since the costs of different IoT devices and applications are disparate, it is difficult to set a cost evaluation criterion for various scenarios. The cost considered here is that of proposed design of SM3 only. Using the integrated SM3 module, we implemented the digital signature and identity authentication to avoid the attacks and to protect the user data from being tampered with. The IP network topology of the given intelligent security gateway is shown in Figure 13.

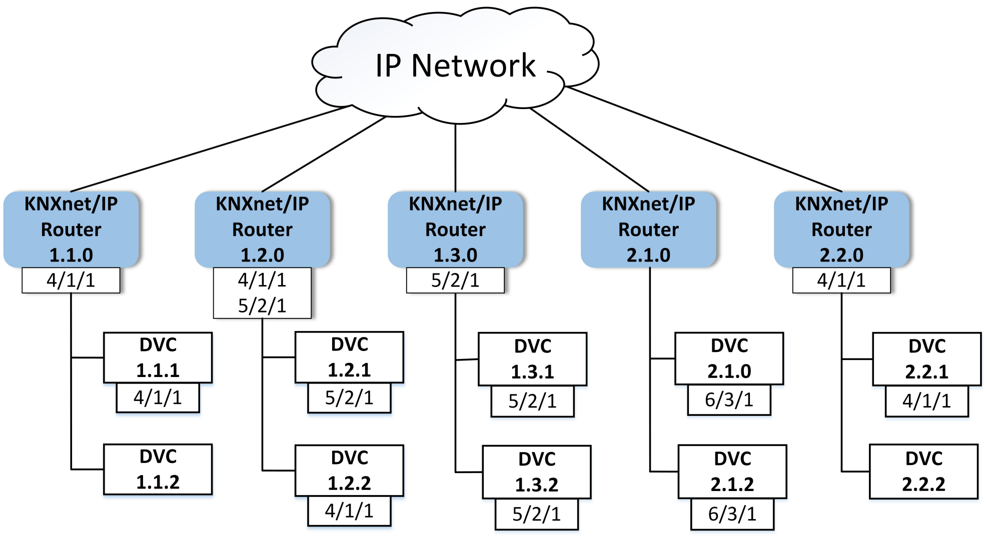


Figure 13. The IP network topology of the intelligent security gateway.

In Figure 13, the intelligent security gateway is a KNXnet/IP router. Generally, the security of IoT communication is guaranteed by a special physical medium and the modulation mode, while the IP is the conventional communication medium. When IoT devices use the IP to conduct large-scale networking, security issues are particularly important. The data transferring between gateways and devices is often the plaintext. It is easy to obtain the control commands of IoT devices through network capture, so as to conduct intrusion control. The function of the intelligent security gateway is to connect all IoT devices through the IP and provide secure communication between gateways. A crucial measure to ensure the security in this gateway is embeding the SM3 module. The AHB-SIC is a very convenient data transfer medium of master and slave devices.

In this application, the security chip embedded in the gateway also integrated with the asymmetric cryptographic module, which can be combined with our proposed module to realize the digital signature and verification. Our proposed design mainly achieves the message digest during the process of the signature. As shown in Figure 14, if the device connected with the gateway needs to authenticate the identity, we can launch the SM3 module to generate the message digest as the ciphertext and input the signature module. If there is no need for authentication, we only use the SM3 module to encrypt the plaintext and transfer the data to ensure the communication channel security.

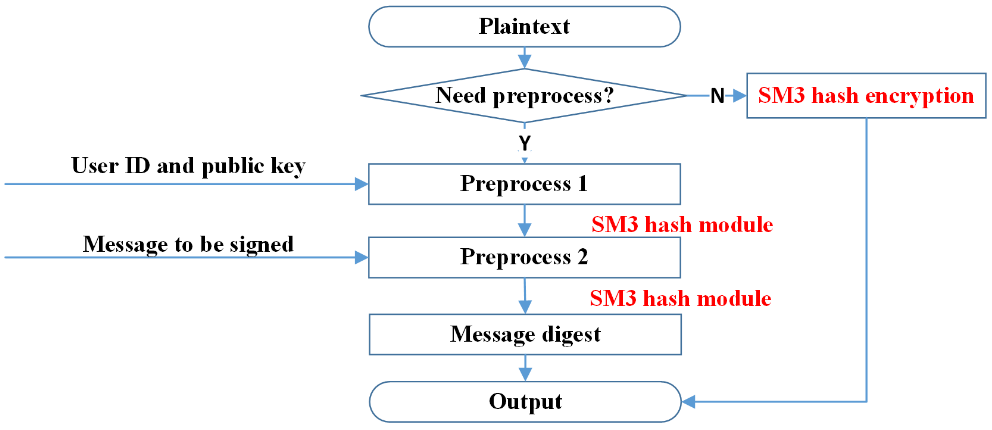


Figure 14. The preprocess flow of the digital signature in the security chip.

Figure 15 shows the application scenario of this intelligent security gateway. All devices need to be authenticated when we access or communicate with the gateway, and all data is encrypted to guarantee non-repudiation and integrity.

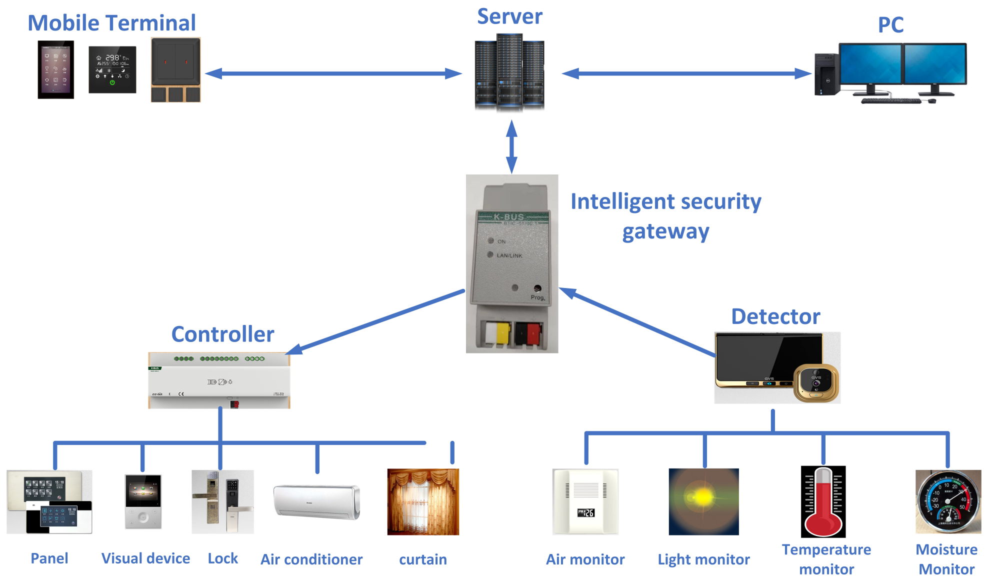


Figure 15. The application scenario of the intelligent security gateway.

6. Conclusions

The SM3 with efficient hardware and low-power architecture is proposed in this paper. The AHB-SIC is designed to quickly convert the non-standard interface of the cryptographic module into the AHB slave interface. The software/hardware co-design implementation approach is adopted to improve the overall performance and to reduce the hardware consumption. Specifically, the padding and parsing operations are implemented in software, while the expansion and compression operations are implemented in hardware. In the expansion module, task scheduling hardware resource optimization techniques are applied to cut down hardware consumption and to increase overall performance of SM3. In the compression module, the task scheduling and critical path optimization techniques are adopted to reduce delay time and enhance overall performance. Compared with the standard implementation of SM3, the proposed architecture reduces approximately 3.11 times of the total registers with 263 Mbps throughput achieved. This design shows an excellent trade-off between performance and the area when compared with previous work. The proposed architecture can be readily applied to the IoT devices.

Author Contributions: All authors contributed to this work. Investigation, X.Z, J.Z.; Methodology, X.Z, X.H.; Supervision, J.Y., S.C.; Writing-original draft, X.Z.; Writing-review and editing, J.Y., S.C. and X.X.

Funding: This research was supported by Science and Technology Planning Project of Guangdong Province of China (2017B010124003, 2017B090909001 and 2017B090909004.).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zhou, J.; Cao, Z.; Dong, X.; Vasilakos, A.V. Security and privacy for cloud-based IoT: Challenges. *IEEE Commun. Mag.* **2017**, *55*, 26–33. [CrossRef]
2. Surendran, S.; Nassef, A.; Beheshti, B.D. A survey of cryptographic algorithms for IoT devices. In Proceedings of the 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT), Bahawalpur, Pakistan, 23–25 October 2018; pp. 1–8.
3. Hwang, Y.H. IoT Security and Privacy: Threats and Challenges. In Proceedings of the Acm Workshop on Iot Privac, Singapore, 14–17 April 2015.
4. Zhou, W.; Jia, Y.; Peng, A.; Zhang, Y.; Liu, P. The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions, and Challenges Yet to Be Solved. *IEEE Internet Things J.* **2018**, *6*, 1606–1616. [CrossRef]

5. Davidson, D.; Moench, B.; Ristenpart, T.; Jha, S. FIE on firmware: finding vulnerabilities in embedded systems using symbolic execution. In Proceedings of the 22nd USENIX conference on Security, Washington, DC, USA, 14–16 August 2013.
6. Tsiropoulou, E.E.; Baras, J.S.; Papavassiliou, S.; Qu, G. On the Mitigation of Interference Imposed by Intruders in Passive RFID Networks. In *Decision and Game Theory for Security, Proceedings of the International Conference, New York, NY, USA, 2–4 November 2016*; Springer: Berlin/Heidelberg, Germany, 2016.
7. Shi, Y.; Wei, W.; He, Z.; Fan, H. An ultra-lightweight white-box encryption scheme for securing resource-constrained IoT devices. In Proceedings of the the 32nd Annual Conference, Los Angeles, CA, USA, 5–9 December 2016.
8. Buchmann, J.; Göpfert, F.; Güneysu, T.; Oder, T.; Pöppelmann, T. High-performance and lightweight lattice-based public-key encryption. In Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security, Xi'an, China 30 May–3 June 2016; pp. 2–9.
9. Rajesh, S.; Paul, V.; Menon, V.G.; Khosravi, M.R. A secure and efficient lightweight symmetric encryption scheme for transfer of text files between embedded IoT devices. *Symmetry* **2019**, *11*, 293. [CrossRef]
10. Yan, H.; Wang, Y.; Jia, C.; Li, J.; Xiang, Y.; Pedrycz, W. IoT-FBAC: Function-based access control scheme using identity-based encryption in IoT. *Future Gener. Comput. Syst.* **2019**, *95*, 344–353. [CrossRef]
11. State Cryptography Administration of China. *Specification of SM3 Cryptographic Hash Function*; State Cryptography Administration of China: Beijing, China, 2010.
12. Ao, T.; He, Z.; Rao, J.; Dai, K.; Zou, X. A compact hardware implementation of SM3 hash function. In Proceedings of the 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, Beijing, China, 24–26 September 2014; pp. 846–850.
13. Hu, Y.; Wu, L.; Wang, A.; Wang, B. Hardware design and implementation of SM3 hash algorithm for financial IC card. In Proceedings of the 2014 Tenth International Conference on Computational Intelligence and Security, Yunnan, China, 15–16 November 2014; pp. 514–518.
14. Khan, A.; Ganesh, G.; Dhodapkar, S.D.; Biswas, B.B.; Patil, R.K. A cryptographic primitive based authentication scheme for run-time software of embedded systems. In Proceedings of the 2010 2nd International Conference on Reliability, Safety and Hazard-Risk-Based Technologies and Physics-of-Failure Methods (ICRESH), Mumbai, India, 14–16 December 2010; pp. 500–504.
15. Juliato M , Gebotys C H . Tailoring a Reconfigurable Platform to SHA-256 and HMAC through Custom Instructions and Peripherals. In Proceedings of the International Conference on Reconfigurable Computing and Fpgas, Cancun, Mexico, 9–11 December 2009.
16. Algredo-Badillo, I.; Feregrino-Uribe, C.; Cumplido, R.; Morales-S, Oval, M. FPGA-based implementation alternatives for the inner loop of the Secure Hash Algorithm SHA-256. *Microprocess. Microsyst.* **2013**, *37*, 750–757. [CrossRef]
17. Federal Information Processing Standards Publication 180-2. *Announcing the Secure Hash Standard*; US DoC/NIST: Gaithersburg, MD, USA, 2002.
18. "ISO/IEC 10118-3", IT Security Techniques-Hash Functions-Part 3: Dedicated Hash Functions. Available online: https://www.iso.org/standard/67116.html (accessed on 7 July 2019).
19. Preneel, B. Software performance of encryption algorithms and hash functions. In Proceedings of the Selected Areas in Cryptography Annual International Workshop Sac, Ottawa, Canada, 18-19 May 1995; pp. 89–98.
20. Chen, D.; Li, Y.J. A High Throughput Rate SM3 IP Design and Implementation. *Electron Devices* **2017**, *03*, 622-625.
21. Chen, P. Optimization Implementation of SM3 Algorithm Based on 64 Rounds Grading Calculation. In Proceedings of the IOP Conference Series: Earth and Environmental Science, Yuzhno-Sakhalinsk, Russia, 27–31 May 2019.
22. Azeez, N.A.; Chinazo, O.J. Achieving Data Authentication with Hmac-Sha256 Algorithm. *Comput. Sci. Telecommun.* **2018**, *54*, 135–142.
23. Shen, Y.; Bai, D.; Yu, H. Improved cryptanalysis of step-reduced SM3. *Sci. China Inf. Sci.* **2018**, *61*, 038105:1–038105:2. [CrossRef]
24. Yu, Y.; Yan, Y.; Li, W. High Speed ASIC Design and Implementation of SM3 Algorithm. *Microelectron. Comput.* **2016**, *33*, 21–26.
25. Zhou, W.; Wang, B.; Zhang, W. Research and application of SM3 hardware implementation. *Electron. Meas. Technol.* **2015**, *38*, 67-71.

26. *AMBA 5 AHB Protocol Specification*; ARM Inc.: Cambridge, UK, 2015.
27. Pandey, A.; Tesfay, D.; Jarso, E. Performance analysis of Intel Ivy Bridge and Intel Broadwell microarchitectures using Intel VTune amplifier software. In Proceedings of the 2018 2nd International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 19–20 January 2018; pp. 423–426.
28. Mavridou, A.; Laszka, A. Designing secure ethereum smart contracts: A finite state machine based approach. *arXiv* **2017**, arXiv:1711.09327.
29. 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 IEEE Computer Society Symposium on Emerging Vlsi Technologies and Architectures, Karlsruhe, Germany, 2–3 March 2006.
30. Zhang, Y.; Xu, L.; Dong, Q.; Wang, J.; Blaauw, D.; Sylvester, D. ReCryptor: A Reconfigurable Cryptographic Cortex-M0 Processor With In-Memory and Near-Memory Computing for IoT Security. *IEEE J. Solid-State Circuits* **2018**, *53*, 995–1005. [CrossRef]
31. Ma, Y.; Xia, L.; Lin, J.; Jing, J.; Liu, Z.; Yu, X. Hardware performance optimization and evaluation of SM3 hash algorithm on FPGA. In Proceedings of the International Conference on Information and Communications Security, Hong Kong, China, 29–31 October 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 105–118.
32. Cao, X.; Lu, L.; O'Neill, M. A compact SHA-256 architecture for RFID tags. In Proceedings of the 22nd IET Irish Signals and Systems Conference, Dublin, Ireland, 23–24 June 2011.
33. He, Z.; Wu, L.; Zhang, X. High-speed Pipeline Design for HMAC of SHA-256 with Masking Scheme. In Proceedings of the 2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID), Xiamen, China, 19–21 October 2018.
34. Beuchat, J.L.; Okamoto, E.; Yamazaki, T. Compact implementations of BLAKE-32 and BLAKE-64 on FPGA. In Proceedings of the 2010 International Conference on Field-Programmable Technology, Beijing, China, 8–10 December 2010; pp. 170–177.
35. Yin, Y.; Jiang, D. Research and Application on Intelligent Parking Solution Based on Internet of Things. In Proceedings of the 2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics, Hangzhou, China, 26–27 August 2013.
36. Tsiropoulou, E.E.; Baras, J.S.; Papavassiliou, S.; Sinha, S. RFID-based smart parking management system. *Cyber-Phys. Syst.* **2017**, *3*, 22–41. [CrossRef]
37. Pinto, A.; Costa, R. Hash-chain-based authentication for IoT. *Adv. Distrib. Comput. Artif. Intell. J.* **2016**, *5*, 43–57. [CrossRef]