# A Compact SHA-256 Architecture for RFID Tags

**Xiaolin Cao, Liang Lu, Maire O'Neill**

*Centre for Secure Information Technologies (CSIT)*

*Institute of Electronics, Communications and Information Technology (ECIT)*

*Queen's University Belfast*

*email: xcao03@qub.ac.uk, l.lu@ecit.qub.ac.uk, m.oneill@ecit.qub.ac.uk*

*Abstract* — **Radio Frequency Identification (RFID) will be a leading technology in the future of ubiquitous computing. Authentication protocols are important in providing security and privacy for RFID tags, and cryptographic hash functions are commonly used as a building block in authentication protocols for RFID applications. In this paper, a compact 8-bit SHA-256 hardware architecture is proposed that can be employed to provide data integrity and authentication in RFID tags. The low-cost techniques used in the proposed architecture include hardware reuse and operation rescheduling. The design has been implemented on both UMC 0.13 µm CMOS standard cell technology and Xilinx Virtex FPGAs. The proposed SHA-256 design utilises approximate 9,036 gate equivalents when implemented in CMOS and consumes 615 slices on a Virtex-4 FPGA device. To the best of the authors' knowledge, the proposed architecture achieves the smallest area in comparison to other SHA-256 designs and SHA-3 finalist candidates reported in the literature to date.**

***Keywords* – SHA-256; RFID; reuse; rescheduling; optimization**.

## I INTRODUCTION

RFID technology is increasingly being deployed in daily life application such as access control, product tracking and logistics automation thanks to its powerful data collection and distribution ability. However, its wireless communication behaviour, which is not line-of-sight, raises significant security and privacy concerns, such as product forgery and consumer tracking. In order to compete with barcode technology, it is believed that the cost of an RFID tag has to be lower than 10 cents [1]. The most often quoted area constraint for the addition of security features on a passive RFID tag is 200 ~ 3,000 gate equivalents (GEs) for security module out of a total budget of 1,000 ~ 10,000GEs. Meanwhile, in relation to power, it is expected that the average current of a tag must be lower than 15µA at an operating frequency of 100 kHz [2]. Therefore, there is considerable interest in developing compact security hardware for RFID systems.

Cryptographic hash functions are popular security components in RFID authentication protocols [1 — 3] due to their desirable security properties: one-way and collision-free. The one-way property can be used to provide forward privacy and the collision-free property can be used to construct constant-time look-up tables. EPC Class-1 Generation-2 [4] defines the de-facto standard for low-cost RFID tags. According to the incoming specification of new generation EPC tags [5], the minimum security strength should be 128 bits. Therefore, a secure hash function with an output greater than or equal to 256 bits is required in order to provide a 128-bit security level [6]. When weaknesses were discovered in SHA-1 in 2005 [7], the US National Institute of Standards and Technology (NIST) recommended that security applications should use the stronger hash function family, SHA-2 [6], since it is believed that practical SHA-2 attacks are unlikely to appear in the next decade [8]. The SHA-256 hash function (which is part of SHA-2 standard) provides a 256-bit hash value. Therefore, SHA-256 is acceptable for use in terms of low-cost RFID tags.

In 2007, the NIST launched the SHA-3 competition [9], with the aim of issuing a new cryptographic hash standard by the end of 2012. There are currently 5 candidates in the final round of the competition: Blake, Grøstl, JH, Keccak and Skein [10]. In the interim period, SHA-2 will play a key role in secure hash applications, and as illustrated in this paper, it may well be the preferred choice over SHA-3 for low-cost RFID applications.

This paper is organized as follows. Section II gives an introduction of the SHA-256 hash function. The proposed low-cost SHA-256 hardware architecture is described in section III. Section IV summarizes the related work. Comparison results are provided in section V, and section VI concludes the paper.

## II SHA-256 HASH FUNCTION

The function of the SHA-256 algorithm [6] is to convert a message with length between 0-$2^{64}$ bits into a 256-bit message digest. The conversion process can be divided into 4 stages as illustrated in Figure 1.

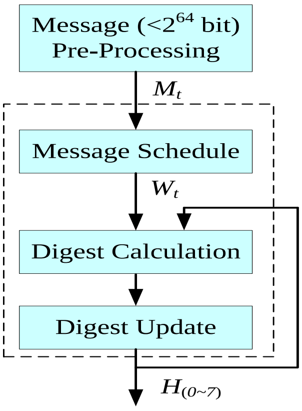

Figure 1: SHA-256 algorithm flow diagram

The first stage is Pre-Processing where the message is segmented and padded into data blocks of length 512 bits. These data blocks ($M_t$) are then sequentially fed into the second stage, namely the Message Schedule where they are expanded into sixty-four 32-bit words ($W_t$) based on the generation formula:

$$W_t = \begin{cases} M_t, 0 \leq t \leq 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}, 16 \leq t \leq 64 \end{cases} \quad (1)$$

where $\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$, and$\sigma_0(x) = ROTR^{7}(x) \oplus ROTR^{18}(x) \oplus SHR^{3}(x)$.

Next, each 32-bit word ($W_t$) is put into the third stage called the Digest Calculation being processed in an iterative loop. There are eight 32-bit working variables ($a \sim h$) with their initialization value $H_{(0\sim 7)}$, and two 32-bit intermediate values ($T_1$, $T_2$) in every loop to be calculated and updated in this stage, based on a series of dedicated functions as follows:

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

where $\sum_1(e) = ROTR^{6}(e) \oplus ROTR^{11}(e) \oplus ROTR^{25}(e)$, $\sum_0(a) = ROTR^{2}(a) \oplus ROTR^{13}(a) \oplus ROTR^{22}(a)$, $Ch(e,f,g) = (e\&f) \wedge ((-e)\&g)$ and $Maj(a,b,c) = (a\&b) \wedge (a\&c) \wedge (b\&c)$.

This stage is the most computationally complex of the hash process. After 64 iterations in this stage, values of the 8 working variables are sent to the final stage, namely the Digest Update. In this stage, the digest variables ($H_{(0\sim 7)}$) are then updated by adding them to the working variables as follows:

$$H_{(0\sim 7)} = H_{(0\sim 7)} + (a \sim h) \quad (3)$$

Normally, one data block processing operation is composed of Message Schedule, Digest Calculation and Digest Update. The iterative block process is completed when all data blocks are processed.

## III PROPOSED SHA-256 ARCHITECTURE

The basic architecture of an RFID tag comprises a digital controller unit and a crypto processing unit [11, 12]. The function of the Pre-Processing stage in the SHA-256 algorithm can be easily computed by the digital controller. Since the majority of ASIC and FPGA SHA (SHA-256 and SHA-3) designs reported in the literature do not include this part, for comparison purposes, an 8-bit SHA-256 is designed without the Pre-Processing stage in this paper. The proposed architecture also falls within the category of "fully autonomous implementation" based on the classification method used in the SHA-3 Zoo [10].

### *a) Architecture Overview*

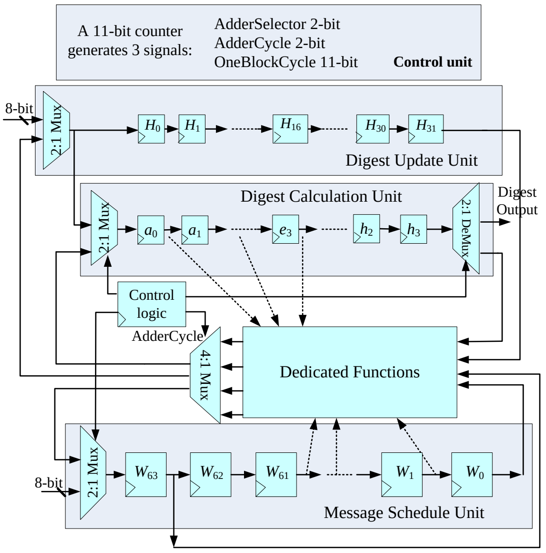

Figure 2: The control unit and data-path diagram

The proposed SHA-256 architecture consists of a control unit and an 8-bit data-path. The novel control unit, based on an 11-bit counter, generates three control signals as illustrated in Figure 2. The AdderCycle (2-bit) signal indicates the corresponding 8-bit word in the 32-bit word which is going to be used in addition operation; the AddSelector (2-bit) signal indicates the adder chain operation that is going to be executed; the OneBlockCycle (11-bit) signal indicates when one data block has been processed.

As illustrated in Figure 2, the 8-bit data-path is composed of the Message Schedule, Digest Calculation and Digest Update units. In the Message Schedule unit, there are sixty-four 8-bit circular shift registers ($W_t$) where $W_{63}$ is used as an input register for the Dedicated functions. Converting Equation (1) into an 8-bit equivalent expression, the indices of the four feedback registers are 0, 4, 36 and 56.

The Digest Calculation unit comprises thirty-two 8-bit circular shift registers, denoted as $(a \sim h)_{[3:0]}$ as shown in Figure 2, while the Digest Update unit also utilises thirty-two 8-bit circular shift registers $H_{[31:0]}$, which are loaded with the initial hash values at system reset. Only at the end of every block message processing stage are the $H_{[31:0]}$ registers updated using a circular shift method and the final message digest output produced. At the beginning of every data block processing stage, the working variable registers are loaded with the $H_{[31:0]}$ values in a circular shift manner. Then the working variable registers are updated every 4 clock cycles.

*b) Rescheduling and Reuse*

It is possible to share a number of logic blocks between the 3 units outlined above, such as the adder-tree and dedicated functions. These are explained in detail below. In order to reduce area, addition and XOR operations are rescheduled and reused as often as possible as depicted in Figure 3, 4 and 5.

Firstly, the 32-bit addition is divided into four 8-bit additions with the assistance of a 4:1 multiplexer, which is controlled by the control signal AdderCycle, illustrated in Figure 2. The intermediate carry bits must be taken into account. In Figure 3, the $C_A$, $C_E$, $C_H$, $C_W$, $C_{T1}$, and $C_{T2}$ signals represent the carry bits of the corresponding additions. Secondly, in the Digest Update stage, the $H_{[31:0]}$ registers are updated using a circular shift method, so that the 8 additions in Equation (3) can be reduced to 1 addition (called an adder chain $H$). From Equation (1) and (2), five other adder chain operations ($W$, $T_1$, $T_2$, $e$, $a$) can be identified. The number of additions in each of these adder chains is different. For instance, the number of adders used to compute the intermediate register $T_1$ , the message schedule register $W_t$ and others are 5, 4 and 2, respectively. Therefore, to make full use of one adder tree, the following data-flow is proposed.

As displayed in Figure 3, the 6 different adder chain operations are implemented using one 8-bit adder tree, whose depth is equal to the longest adder chain, namely $H$. Six identical 4:1 multiplexers are controlled by the control signal AddSelector. When AddSelector equals 1, the first row ($\sigma_1$, $W_{36}$, $\sigma_0$, $W_0$, 0, $C_W$) is selected as inputs to the adder tree to calculate the adder chain $W$. When AddSelector equals 2, the second row ($h_3$, $\sum_1$, $Ch$, $K$, $W_{63}$, $C_{T1}$) is selected as inputs of the adder tree to calculate adder chain $T_1$. When AddSelector equals 3, the third row ($d_3$, $T_1$, $C_E$, $h_3$, $H_0$, $C_H$) is selected as inputs of the adder tree to calculate the adder chains $e$ and $H$. When AddSelector equals 4, the fourth row ($\sum_0$, $Maj$, $C_{T2}$, $T_1$, $T_2$, $C_A$) is selected as inputs to the adder tree to calculate the adder chains $T_2$ and $a$.

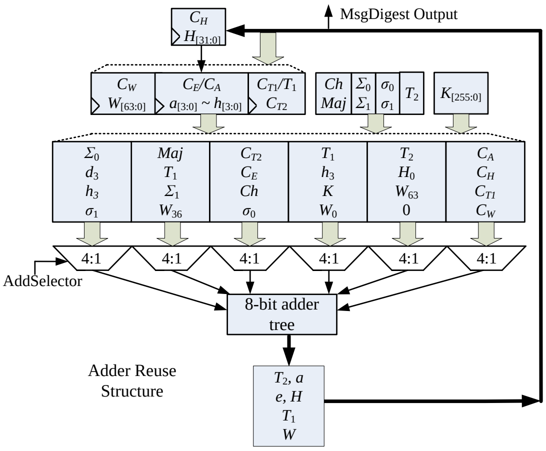

Figure 3: Addition reuse diagram

As a result, the message schedule register $W_t$ and the intermediate register $T_1$ are added in the 1st and 2nd cycles respectively; the working variable register $e$ and digest register $H_{[31:0]}$ are calculated simultaneously on the 3rd cycle. On the 4th cycle, the intermediate variable $T_1$ and the working variable register, $a$, are also calculated simultaneously. With this dataflow, the 8-bit adder is fully utilized and an 8-bit register for $T_2$ can be saved to reduce the area. The update of the working variables can be carried out in 2 clock cycles.

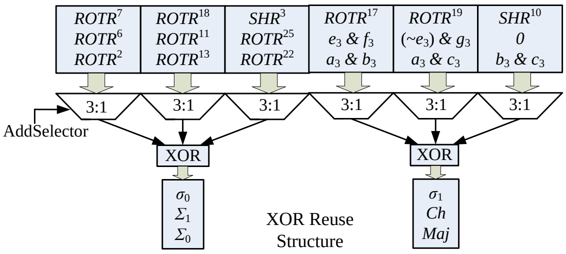
$ROTR^7$ $ROTR^6$ $ROTR^2$
$ROTR^{18}$ $ROTR^{11}$ $ROTR^{13}$
$SHR^3$ $ROTR^{25}$ $ROTR^{22}$
$ROTR^{17}$ $e_3$ & $f_3$ $a_3$ & $b_3$
$ROTR^{19}$ $(\sim e_3)$ & $g_3$ $a_3$ & $c_3$
$SHR^{10}$ 0 $b_3$ & $c_3$
3:1
AddSelector
XOR
XOR
$\sigma_0$ $\Sigma_1$ $\Sigma_0$
XOR Reuse Structure
$\sigma_1$ $Ch$ $Maj$

Figure 4: XOR reuse diagram

The XOR operation is used in 6 different dedicated functions ($\sigma_0$, $\sigma_1$, $\sum_0$, $\sum_1$, $Maj$, $Ch$). Due to the fact that $W_t$, $T_1$ and $T_2$ are calculated in three different clock cycles respectively and in each cycle, two XOR operations are applied, six 3:1 multiplexers are needed to reuse the two XOR operators, illustrated in Figure 4. These six multiplexers are also controlled by AddSelector. On the 1st cycle, the first rows ($ROTR^7$, $ROTR^{18}$, $SHR^3$) and ($ROTR^{17}$,

$ROTR^{19}$ , $SHR^{10}$ ) are selected for XORing to generate the values of $\sigma_0$ and $\sigma_1$. On the 2nd cycle, the second rows ($ROTR^6$, $ROTR^{11}$, $ROTR^{25}$) and ($e_3$&$f_3$, (~$e_3$)& $g_3$, 0) are selected to generate the values of $\sum_1$ and $Ch$. Finally the third rows ($ROTR^2$, $ROTR^{13}$, $ROTR^{22}$) and ($a_3$&$b_3$, $a_3$&$c_3$, $b_3$&$c_3$) are used to generate the values of $\sum_0$ and $Maj$.

The design method for the $n$-bit $SHR^n$ ($n$ = 3, 10) and $ROTR^n$ operations ($n$ = 2, 6, 7, 11, 13, 17, 18, 19, 22, 25) is the same as that used in [13] to reduce the logic area in the 32-bit rotation instructions. For example, the wiring connections of $SHR^3$, $ROTR^7$ and $ROTR^{18}$ are illustrated in Figure 5, in which $W_{(2,3,4,5,6,7)}$ is part of the message schedule registers.

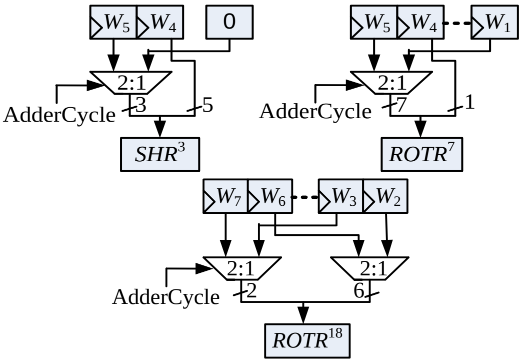

Figure 5: *SHR* and *ROTR* logic diagram

### c) Timing Results

According to the specification in ISO-18000-3 [13], the tag processing time should be within 320 µs. But the interleaved challenge-response protocol proposed by Feldhofer *et al*. [12] already extends this to 18 ms, which is equivalent to a total latency of 1800 clock cycles at an operating frequency of 100 kHz.

Because only one 8-bit adder tree is used in the proposed design, 4 clock cycles are required for each 8-bit message processing. At the beginning, 64 (16x4) clock cycles are used to load the padded data block. Then 64 messages are processed in 1,024 (64x4x4) cycles. At the end, to update the thirty-two 8-bit registers $H_{(0\sim31)}$, 32 clock cycles are required. Thus, the processing of one data block takes 1,120 clock cycles in total, not exceeding the 1,800 limitation. Therefore, the proposed architecture meets the timing requirement of RFID tags.

Due to the timing limitation, it is not possible to further reduce the area by using only one 8-bit adder in this system. If only one 8-bit adder is adopted, the minimum latency will be 3,648 clock cycles. Therefore, the clock frequency would need to be increased in order to meet the response time demand. However, this would result in a significant increase in the power consumption.

## IV RELATED WORK

Previous research for low-cost SHA-256 designs includes work by Satoh and Inoue [14], which requires 11,484GEs and does not use any addition and XOR rescheduling. Feldhofer and Rechberger in 2006 [15] presented a 32-bit compact hardware architecture for SHA-256, requiring 10,868GEs in area, 52.37µW in power consumption and 1128 clock cycles in block latency. In 2007, they reported that an identical design with a different voltage of 1.2V consumed 8.79µW [2]. In FPGA technology, the most compact implementations are provided by Helion Technology [16] and Chaves *et al*. [17], respectively requiring 758 slices and 755 slices. Both of these utilise a Block RAM (BRAM) to store constants and other variables.

At present, there are 5 candidates in the 3rd round of the NIST SHA-3 competition: Blake, Grøstl, Keccak, Skein and JH. A hardware implementation comparison of these candidates is provided on the SHA-3 Zoo website [18]. The smallest ASIC design is achieved by Keccak-256. If no external memory is used, 9,300GEs are required; otherwise, 5,000GEs are needed. Meanwhile the smallest FPGA designs are of Skein-256 and Blake-256. The Skein-256 consumes 937 slices on a Virtex-5 device, and Blake-256 utilises 124 slices and 2 BRAMs [19]. There are no area optimized hardware performance results available for JH-256 [18].

In the existing literature, some designs only implement the “core functionality” of the hash functions, but the proposed architecture is a “fully autonomous” design. Many designs [16 – 19] use external memory (ROM or RAM chips) or BRAMs to replace register arrays in order to store constants and intermediate digest values. However, Feldhofer *et al.* [10] points out that this method is not practical, since external memory is too expensive and not available in low-cost RFID tag applications. The proposed design outlined in this paper does not use any custom memory to replace the register array. Therefore, in Table 2, only “fully autonomous” ASIC designs that do not utilise external memory are compared.

## V IMPLEMENTATION AND COMPARISON

The proposed architecture was implemented in the Faraday UMC 130µm ASIC standard cell library and Xilinx Virtex-2 and Virtex-4 technology for comparison purposes. The ASIC implementation was synthesized using Synopsys Physical Compiler 2006-06, and power simulation undertaken with Synopsys Primetime-PX 2007-06. Modelsim 6.4a was used as the functional and post-synthesis timing simulation tool. For the FPGA implementation, the design was synthesized using Synplify Pro 9.4 and implemented using the Xilinx ISE 9.1 tools. The test vectors for SHA-256 from the official NIST website [6] were used for the proposed design.

### *a) ASIC Implementation Comparison*

Table 1: ASIC implementation component ratio

| Module/ Component | Area (GEs) | Area (%) | Power (μW) @100kHz | Power (%) |
|---|---|---|---|---|
| Control/ counter | 397 | 4.4 | 1.24 | 40.5 |
| Datapath/ $W$ | 3,328 | 73.7 | | |
| Datapath/ $a\sim h$ | 1,664 | | | |
| Datapath/ $H$ | 1,664 | | | |
| Control/ others | 97 | 1.1 | 1.82 | 59.5 |
| Datapath/ addertree | 184 | 2.0 | | |
| Datapath/ others | 1,692 | 18.8 | | |
| Total | 9,036 | 100 | 3.06 | 100 |

Table 2: ASIC implementation comparison

| Design | Tech. (μm/V) | Area (GEs) | Power (μW) @100kHz | Timing (cycle) |
|---|---|---|---|---|
| SHA-256 | 0.13/1.2 | 9,036 | 3.06 | 1,120 |
| SHA-256 [2] | 0.35/1.5 | 10,868 | 8.79 | 1,128 |
| SHA-256 [14] | 0.13/NA | 11,484 | NA | 72 |
| Blake-256 [18] | 0.18/NA | 13,575 | NA | 816 |
| Blake-256 [18] | 0.35/ NA | 25,569 | NA | 1,038 |
| Grøstl-256 [18] | 0.35/ NA | 14,622 | NA | 196 |
| Keccak-256 [18] | 0.13/ NA | 9,300 | NA | 5,160 |
| Skein-256 [18] | 0.35/ NA | 12,890 | NA | 1,034 |

Table 1 lists the component area and power ratios in the proposed SHA-256 ASIC implementation. The average power consumption at 100 kHz is 3.06 μW with a core voltage of 1.2V. According to the specifications in [12], this result meets the demands of RFID tag applications. From Table 1, it can be observed that the area ratios for control logic, adders, and dedicated functions are reduced compared to the work in [2, 15].

A comparison of the proposed ASIC implementation with previous low-cost SHA-256 work and SHA-3 designs is given in Table 2. The proposed design is 16.8% smaller than the previous smallest SHA-256 design in [2]. Estimating a maximum 10% difference in the GE number on different ASIC technologies [13], this equates to an 8.5% area reduction. Compared to the smallest SHA-3 candidate, Keccak-256 [18], the area reduction of the proposed architecture is about 250GEs, and it is also 4.5 times faster. Overall, this proposed design achieves the smallest area and the lowest power consumption of SHA-256 ASIC designs reported to date, and interestingly, is also smaller in comparison to all the SHA-3 candidates in the final round.

### *b) FPGA Implementation Comparison*

Table 3: FPGA implementation comparison

| Design | Xilinx Tech. | Area (slices) | Frequency (MHz) |
|---|---|---|---|
| SHA-256 | Virtex-4 | 615 | 102 |
| SHA-256 | Virtex-2 | 639 | 85 |
| SHA-256 [16] | Virtex-4 | 758 + 1 BRAM | 162 |
| SHA-256 [17] | Virtex-2 | 755 + 1 BRAM | 174 |
| Blake-256 [18] | Virtex-2 | 958 | 59 |
| Blake-256 [18] | Virtex-4 | 960 | 68 |
| Blake-256 [19] | Virtex-4 | 124 + 2 BRAM | 357 |
| Grøstl-256 [18] | Virtex-2 | 2,754 | 81.5 |
| Keccak-256 [18] | Virtex-5 | 444 + External Memory | 265 |
| Skein-256 [18] | Virtex-5 | 937 | 68.4 |

A comparison of the proposed FPGA implementation with previous area optimized SHA-256 work and SHA-3 design is given in Table 3. Although slightly slower than the work in [16, 17], the proposed design is at least 15% smaller if BRAM is not considered. According to the Xilinx FPGA datasheet, on a Virtex-2 device, 4 slices can be used to store up to 128 bits of data [20], and on a Virtex-4 device, 4 slices can store a maximum data of 64 bits [21]. Therefore, the equivalent slice area of a BRAM for used to store 256 8-bit constants as in [17] is at least 64 (256x8/128x4) slices, which gives a further 8.5% reduction in area.

Compared to the SHA-3 designs that do not utilise BRAMs or external memory, the proposed architecture is the smallest since they require more than 900 slices. For the Blake-256 design in [19], it not only uses BRAMs to store the register file and control unit, but also utilises a pipeline structure to accelerate process speed. The register file comprises 16 32-bit constants and 64 32-bit registers, and the control unit comprises 844 18-bit instructions. Therefore the equivalent slice area for this usage of BRAMs is at least 1,073 ((80x32+844x18)/64x4) slices, which is bigger than the Blake-256 design that does not utilise BRAMs [18].

From a hardware implementation perspective, there are different considerations for area optimization in ASIC and FPGA technology. However, from the performance comparison results, it can be observed that a feature of the proposed architecture is that it is not only suitable for ASIC, but also for FPGA.

## VI CONCLUSIONS

Cryptographic hash functions currently play an important role in providing data security in next-generation wireless and ubiquitous devices. In this paper, a new compact 8-bit SHA-256 architecture for RFID tag security has been proposed. Hardware logic reuse and rescheduling techniques are explored. A single 8-bit adder tree is rescheduled and reused to reduce area, and XOR is also reused. This achieves significant area reduction over previous work. The proposed architecture has been implemented and verified using both ASIC and FPGA technology, showing that it meets the requirements of RFID interleaved communication protocols. Finally, the performance result of the new architecture show that it achieves the smallest area in comparison to other work. Future research will focus on integrating this low-cost design with security protocols for RFID systems.

## REFERENCES

[1] A. Juels, "RFID Security and Privacy: A research Survey", *IEEE Journal on Selected Areas in Communications*, February 2006.

[2] M. Feldhofer, and J. Wolkerstorfer, "Strong Crypto for RFID Tags - A Comparison of Low-Power Hardware Implementations", *IEEE Int'l Symp. on Circuits and Systems*, 2007, pp. 1839-1842.

[3] B. Song, "RFID Authentication Protocols using Symmetric Cryptography", *PhD thesis*, December 2009.

[4] EPCglobal Standard: UHF Class 1 Gen 2 Standard v1.2.0, *EPCglobal Inc.* Available from http://www.epcglobalinc.org/standards/.

[5] Tag Classification Definitions. *EPCglobal Inc.* Available from http://www.epcglobalinc.org/standards/.

[6] Secure Hash Standard (SHS): FIPS PUB 180-3. , *NIST*. Available from http://csrc.nist.gov/publications/PubsFIPS.html.

[7] X. Wang, Y.L. Yin, and H. Yu, "Finding collisions in the full SHA-1", *Advances in Cryptology CRYPTO 2005*, Springer-Verlag, 2005, pp. 1736.

[8] NIST Special Publication 800-107, Recommendation for Applications Using Approved Hash Algorithms, *NIST*, February 2009.

[9] Cryptographic hash Algorithm Competition, *NIST*. Available from http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[10] The SHA-3 Zoo. Available from http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo.

[11] K. Finkenzeller, RFID Handbook, *John Wiley & Sons*, 2003.

[12] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong Authentication for RFID Systems Using the AES Algorithm", in M. Joye and J.-J. Quisquater, editors, *Pro-ceedings of CHES 2004*, Lecture notes in computer science 3156, Springer Verlag, 2004, pp. 357-370.

[13] M. O'Neill (Nee McLoone), "Low-Cost SHA-1 Hash Function Architecture for RFID Tags", *RFID security workshop (RFIDSec'08)*, July 2008.

[14] A. Satoh, and T. Inoue, "ASIC-Hardware-Focused Comparison for Hash Functions MD5, RIPEMD-160 and SHS", *Proc. of the Int'l Conf. on Information Technology: Coding and Computing (ITCC'05)*, April 2005, pp. 532-537.

[15] M. Feldhofer, and C. Rechberger, "A Case Against Currently Used Hash Functions in RFID Protocols", *1st Int'l W/shop on Inf. Security*, Vol. 4277, Springer, Montpellier, France, 2006, pp. 372-381.

[16] SHA-256 hash core IP on Xilinx Virtex 4-11, *Helion Technology*. Available from http://www.heliontech.com/sha256.htm.

[17] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, "Cost-Efficient SHA Hardware Accelerators", *IEEE Trans. on VLSI Systems*, Vol. 16, No. 8, August 2008.

[18] SHA-3 Hardware Implementations. Available from http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations#Implementation_of_Core_Functionality.

[19] Jean-Luc Beuchat, Eiji Okamoto, and Teppei Yamazaki. Compact Implementations of BLAKE-32 and BLAKE-64 on FPGA. *IACR Eprint report* 2010/173.

[20] Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet. Available from http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf.

[21] Virtex-4 Family Overview. Available from http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf.