A Hardware Implementation of SHA3 Hash Processor using Cortex-M0

Dong-Seong Kim, Sang-Hyun Lee, Kyung-Wook Shin
Kumoh National Institute of Technology
E-mail; kdsung322@kumoh.ac.kr

Abstract

This paper describes a SHA3 hash processor that is interfaced with Cortex-M0 to be used as an IP. The SHA3 hash processor was designed with a round-iterative structure of 1600-bit data-path, and it supports four different message digest sizes of 512, 384, 256, and 224 bits depending on the hash function used. The SHA3 processor interfaced with Cortex-M0 was verified by FPGA implementation. It was estimated to have a throughput of 5 Gbps at the maximum clock frequency of 289 MHz, and it occupied 1,692 slices of Virtex5 FPGA device.

Keywords: Hash, SHA3, Keccak, integrity, security

1. Introduction

A Cryptographic hash function transforms the message of arbitrary size into a string of fixed size, called the 'message digest'. Hash algorithms play a key role in information security applications such as message authentication code (MAC), digital signature schemes, and pseudo random number generations. Until recently, many hash algorithms have been proposed including SHA-1 (Secure Hash Algorithm-1), SHA-2 [1], HAS-160 [2], and Whirlpool [3]. Keccak function was selected by the National Institute of Standards and Technology (NIST) as a new Secure Hash Algorithm-3 (SHA3) in 2015 [4]. SHA3 is known to be the most secure hashing algorithm so far, since it is less vulnerable to attacks such as collision resistance and pre-image resistance.

The hash functions may be implemented in software or hardware, but it is generally known that the hardware implementation is more secure than software realizations. Hardware implementations also can provide the flexibility to be optimized to meet the application-specific requirements such as small area, high performance, and low power consumption.

This paper describes a hardware implementation of SHA3 Keccak. Section 2 briefly introduces the SHA3 hash function, and the SHA3 processor is described in section 3. Section 4 describes AHB interface with Cortex-M0, then FPGA verification is discussed in section 5, and conclusion is followed.

2. SHA3 Hash Function

SHA3 family [4] consists of four cryptographic hash functions: SHA3-224, SHA3-256, SHA3-384, and SHA3-512. In each case, the suffix after the dash indicates the fixed length of digest, e.g., SHA3-256 produces message digest of 256-bit. The SHA3 has two extendable-output functions (XOFs), SHAKE128 and SHAKE256, which allow the output to be extended to any desired length. The suffixes 128 and 256 indicate security strength. Each of the six SHA3 functions employs the same underlying permutation as the main component in sponge construction.

Keccak is based on the so-called sponge function family. Unlike existing Merkle-Damgard based hash functions, the Keccak permutation is performed on a *state* with a fixed size of b bits. In the Keccak-f permutation denoted by Keccak-*f[b]*, the state size b can be {25, 50, 100, 200, 400, 800, 1600}. Of seven different permutation functions, Keccak-*f[1600]* was chosen for the SHA3 standard. The state matrix of Keccak-*f[1600]* is represented as a 3-D array of 5x5x64 bits, as depicted in Figure 1. The *state* matrix is composed of *slices* and *lanes*. A *slice* is a 2-D matrix composed 5x5 bits, and a lane is a 1-D array of 64 bits.

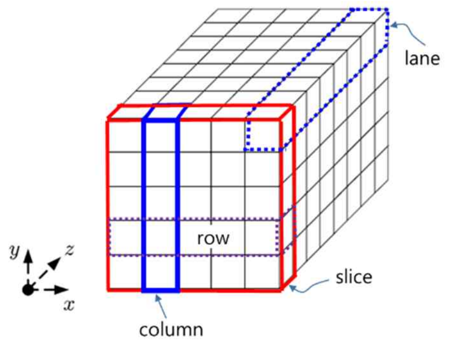


Figure 1: 3-D array of 5 × 5 × 64 bits for Keccak-*f[1600]* **state matrix [4]**

The Keccak permutation is composed of three phases that are initializing, absorbing, and squeezing, as illustrated in Figure 2. In initializing phase, all the bits of the state are set to zero, and input data is padded and is divided into blocks of r bits. During absorbing phase, the r-bit *rate* of the initialized state is XORed with the first *r*-bit of the state. The new *rate* and together with the *capacity c* of the initialized state forms a new state used in *f*-permutation. The resulting state serves as the new initial state for the next permutation. This process repeats depending on the length of the input message. In squeezing phase, desired length of output is obtained by truncation. In the sponge construction, the *rate* denoted by *r* is the number of input bits to be processed in one block permutation. The *capacity*, denoted by *c*, is the remaining bits of the state, i.e., *c=b-r*, where *b* is the width of *f*-permutation. For Keccak-*f[1600]*, *r=1088*, and *c=2n=512*, where *n* is the length of the output.

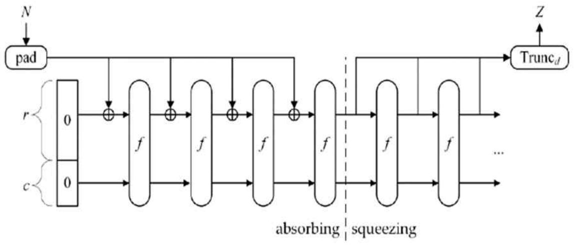


Figure 2: SHA3 sponge function [4]

The Keccak-*f[1600]* permutation consists of 24 rounds, and each round is divided into five steps: Theta (θ), Rho (ρ), Pi (π), Chi (χ), and Iota (ι) which are defined in equations (1) ~ (5).

Theta (θ) step:

$$A'[x,y,z] = A[x,y,z] + \sum_{y'=0}^{4} A[(x-1)\bmod 5, y', z] + \sum_{y'}^{4} A[(x+1)\bmod 5,\ y',\ (z-1)\bmod w] \quad (1)$$

Rho (ρ) step:

$$A'[x,y,z] = A[x,y,(z + offset(x,y))\bmod w] \quad (2)$$

Pi (π) step:

$$A'[y,(2x+3y)\bmod 5] = A[x,y] \quad (3)$$

Chi (χ) step:

$$A'[x,y,z] = A[x,y,z] + \big((A[(x+1)\bmod 5,y,z]+1)\cdot A[(x+2)\bmod 5,y,z]\big) \quad (4)$$

Iota (ι) step:

$$A'[0,0] = A[0,0] + RC \quad (5)$$

3. SHA3 Hash Processor

Figure 3 shows the internal structure of the SHA3 processor that is composed of Reg_map block for interfacing with Cortex-M0 through bus matrix, Padder block for message padding, and Round block. 64-bit data-path between Padder block and Round block was used to minimize data buffer size.

To achieve either high-performance or lightweight hardware implementation, we can consider various hardware structures for the round block. For high

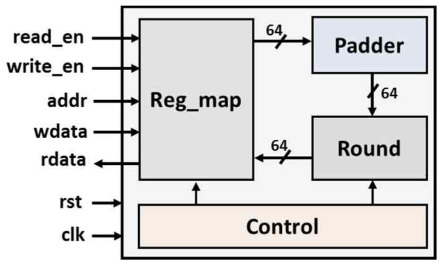


Figure 3: Architecture of SHA3 Processor

performance, the approaches such as loop unrolling, pipelining, and parallel processing can be used, which require large hardware cost. Another consideration in the round block design is to determine data-path width that has trade-off between performance and hardware cost. Recently, an analysis of throughput per area performance of SHA3 hash processor was reported in [5], which evaluated five types of data-path structure with 64-b, 320-b, 640-b, 960-b, and 1600-b. The analysis showed that 1600-b data-path has the best throughput per area performance.

In our design, the basic round-iterative architecture was adopted, and Figure 4 depicts the internal structure of the round block designed with 1600-b data-path. The 1600-b register stores the intermediate results of each iteration, and the Keccak-*p* permutation block computes five steps defined in Eqs. (1) ~ (5). It uses one clock cycle per round, so it takes 24 clock cycles for a permutation. The five steps in a round were implemented in three parts: theta, rho & pi, and chi & iota. Instead of storing 24 pre-computed round constants (RCs) of 64-bit, a simplified RC generator was designed by storing only non-zero 8-bit in each of RC in LUT [6], thus resulting in reduced hardware.

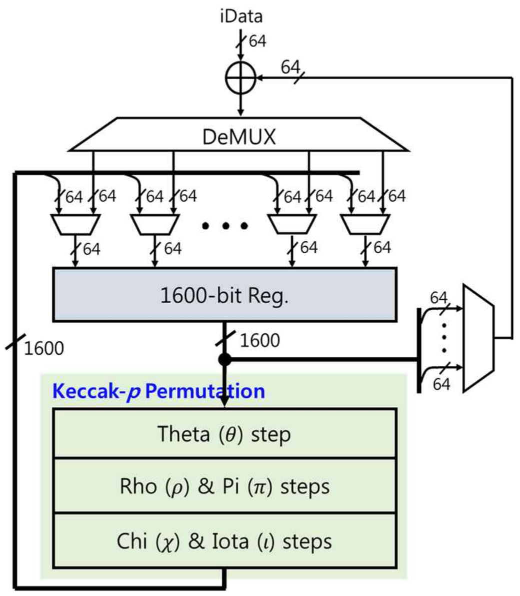


Figure 4: Round block

4. AHB Interface with Cortex-M0

To integrate the SHA3 hash processor described in previous section into a security system-on-chip (SoC) as an IP, the AHB interface with Cortex-M0 was designed as shown in Figure 5. The AHB_slave_IF block implements the AHB protocol used to send and receive data between the Cortex-M0 and the SHA3 hash processor. The Cortex-M0 controls the operation of the SHA3 hash processor by setting parameters in the control register of Reg_map block.

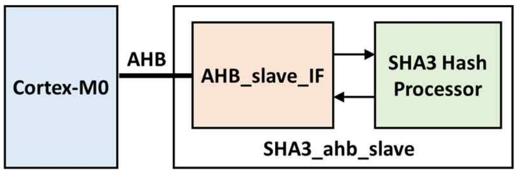


Figure 5: AHB Interface with Cortex-M0

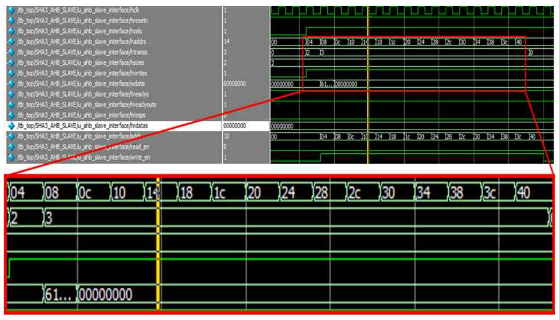
5. Verification

Figure 6 shows the results of bus function model (BFM) simulations for the AHB slave interface of the SHA3 hash processor with Cortex-M0. For the BFM simulation, the AHB Master block was modeled on a test-bench, and the results were confirmed by interfacing SHA3_ahb_slave to AHB Master Block. Figure 6-(a) shows that data sent from AHB Master are stored in the internal register of SHA3_ahb_slave. Figure 6-(b) shows the message digest stored in the register map, which shows the correct operation of AHB interface. From the BFM simulation results, we have confirmed that the SHA3 hash processor interfaced with Cortex-M0 works correctly.

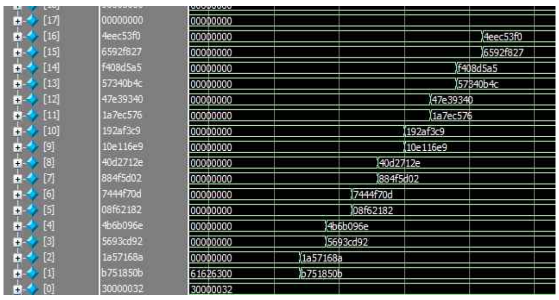
The SHA3 hash processor was also verified by FPGA implementation. FPGA verification setup consists of Xilinx Virtex5 FPGA (XC5VSX-95T) board, UART interface, and GUI software as shown in Figure 7-(a). After input message is loaded from PC to FPGA via UART interface, SHA3 hash processor implemented in the FPGA device calculates message digest, and then the message digest is sent back to the PC via UART interface and displayed on GUI screen. Figure 7-(b) is a screenshot of the FPGA verification results showing that the message digest of " b751 850b 1a57 168a 5693 cd92 4b6b 096e 08f6 2182 7444 f70d 884f 5d02 40d2 712e 10e1 16e9 192a f3c9 1a7e c576 47e3 9340 5734 0b4c f408 d5a5 6592 f827 4eec 53f0" is obtained by the input message "abc".

6. Conclusion

A SHA3 hash processor, which supports four hash functions of SHA3-512, SHA3-384, SHA3-256, SHA3-224 defined in NIST FIPS PUB-202, was designed and interfaced with Cortex-M0, and verified by FPGA implementation. It occupies 1,692 slices in Virtex5 FPGA device, and provides 5 Gbps throughput at a maximum clock frequency of 289 MHz. The SHA3 hash processor will be integrated into a security SoC as an IP core.

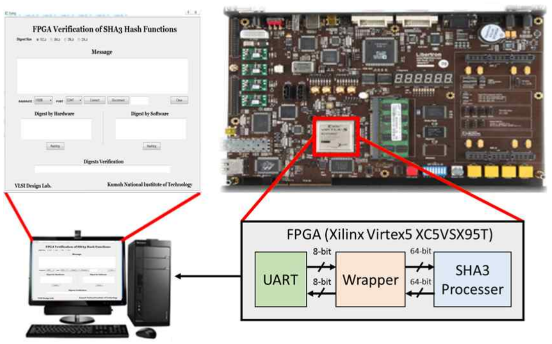


(a) AHB slave operation

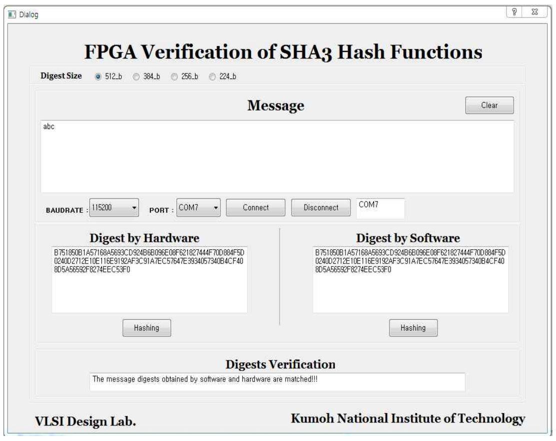


[17]	00000000	00000000
[16]	4eec53f0	00000000 4eec53f0
[15]	6592f827	00000000 6592f827
[14]	f408d5a5	00000000 f408d5a5
[13]	57340b4c	00000000 57340b4c
[12]	47e39340	00000000 47e39340
[11]	1a7ec576	00000000 1a7ec576
[10]	192af3c9	00000000 192af3c9
[9]	10e116e9	00000000 10e116e9
[8]	40d2712e	00000000 40d2712e
[7]	884f5d02	00000000 884f5d02
[6]	7444f70d	00000000 7444f70d
[5]	08f62182	00000000 08f62182
[4]	4b6b096e	00000000 4b6b096e
[3]	5693cd92	00000000 5693cd92
[2]	1a57168a	00000000 1a57168a
[1]	b751850b	61626300 b751850b
[0]	30000032	30000032

(b) Message digest stored in register map
Figure 6: BFM simulation results



(a) FPGA verification setup



Dialog

FPGA Verification of SHA3 Hash Functions

Digest Size [✓ 512_b] 384_b 256_b 224_b

Message

abc

BAUDRATE : 115200 PORT : COM7 COM7

Digest by Hardware

B751850B1A57168A5693CD924B6B096E08F621827444F70D884F5D
0240D2712E10E116E9192AF3C91A7EC57647E3934057340B4CF40
8D5A56592F8274EEC53F0

Digest by Software

B751850B1A57168A5693CD924B6B096E08F621827444F70D884F5D
0240D2712E10E116E9192AF3C91A7EC57647E3934057340B4CF40
8D5A56592F8274EEC53F0

Digests Verification

The message digests obtained by software and hardware are matched!!!

VLSI Design Lab. Kumoh National Institute of Technology

(b) Screenshot of FPGA verification result
Figure 7: FPGA verification of SHA3 hash core

Acknowledgement

• This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2017R1D1A3B03031677).
• This work was supported by Korea Institute for Advancement of Technology (KIAT) grant funded by the Korean government (Ministry of Trade, Industry & Energy, HRD Program for Software-SoC convergence) (No. N0001883).
• The authors also are thankful to IDEC for EDA software support.

References

[1] NIST std. FIPS 180-2, Secure Hash Standard (SHS), National Institute of Standard and Technology (NIST), Oct. 2001.
[2] TTA std. TTAK.KO-12.0011/R1, Hash Function Standard - Part 2: Hash Function Algorithm Standard (HAS-160), Telecommunications Technology Association (TTA), Dec., 2000.
[3] P. Barreto and V. Rijmen, "The Whirlpool Hashing Function," May 2003.
[4] National Institute of Standards and Technology. FIPS PUB 202 - SHA3 Standard: Permutation-Based Hash and Extendable-Output Functions, Aug. 2015.
[5] D. S. Kim and K. W. Shin, "Analysis of Optimal Design Conditions for SHA3-512 Hash Function," *Korea Institute of Information and Communication Engineering*, Oct. 2018.
[6] M. M. Wong, J. H. Yahya, S. Sau, "A New High Throughput, and Area Efficient SHA-3 Implementation," *IEEE International Symposium on Circuits and System*, May 2018