A Compact Hardware Implementation of SHA-3 Candidate *Luffa*

Shugo Mikami[1], Nagamasa Mizushima[1], Setsuko Nakamura[1], and Dai Watanabe[1]

Systems Development Laboratory, Hitachi, Ltd.,
292 Yoshida-cho, Totsuka-ku, Yokohama, 244-0817, Japan
dai.watanabe.td@hitachi.com

Abstract. In this document, the hardware performances of *Luffa*-256[1] are reported. Our implementations mainly target size optimized implementations in ASIC and the smallest architecture can be implemented with only 10.3 KGE while it achieves about 500 Mbps.

Keywords. Hash function, *Luffa*, hardware implementation

1 Introduction

A cryptographic hash function has a lot of application such as a digital signature and a message authentication code. Recently, several important breakthroughs have been made in the cryptanalysis against hash functions and they imply that most of the currently used standard hash functions are vulnerable against new attacks. In these circumstances, National Institute of Standards and Technology (NIST) decided to organize Cryptographic Hash Algorithm Competition (The SHA-3 competition) [6] and started to call for algorithms.

Luffa [2] is a family of hash functions submitted to the SHA-3 competition and was selected as one of the second round candidates. *Luffa* modified its algorithm at the beginning of the second round and the current algorithm is called *Luffa* v2. Throughout this document, we discuss the modified algorithm (*Luffa* v2) and denote it *Luffa*.

In this document, we discuss the hardware implementations of *Luffa* in ASIC. There have been some reports on the hardware implementations of *Luffa* including the self evaluation report [3]; The first implementation of *Luffa* was done by Knežević and Verbauwhede [4], and their architecture followed those of [3]. Namin and Hasan [5] tried to implement a whole round processing. Recently, Satoh *et al.* [7] covered four architecture. Two of them were new, but their trade-off between the throughputs and the size were not so significant.

In this document, we propose two new architectures of *Luffa*-256 which are fully functional architecture and they target size-optimized implementation. One of them can be implemented with 10.3 KGE and achieves about 500 Mbps,

[1] *Luffa* is a registered trademark of Hitachi, Ltd. in Japan.

while the other can be implemented with 14.0 KGE and achieves 3 Gbps. These results indicate that *Luffa* is quite a flexible algorithm in terms of hardware implementation.

The rest of this paper is organized as follows: The three hardware architectures of *Luffa*-256 including two new ones are introduced in Section 3. Then the evaluation results are given in Section 4. We conclude this document in Section 5.

2 The Specification of *Luffa*-256

See [2].

3 Hardware Architectures

In this section, we explain two new architectures. They adopt the shift register based architecture instead of the selector based architecture in order to save the number of selectors. This technique was applied to AES by Shimizu *et al.* [8] and also applied to SHA-3 2nd round candidates Shabal and CubeHash by Bernet *et al.* [1].

3.1 Architecture 1: A Step Function

The architecture, which shares a step function by three 256-bit permutations (See Figure 1), has been examined in [4, 7]. We implemented this architecture just for the comparison to other two new architectures.

3.2 Architecture 2: A Half Step Function

The second architecture implements only a half of the step function.

Figure 2 shows the rough structure of the second architecture. A line represents 128-bit data flow. In this architecture, a half of the message injection function MI and `SubCrumb` are implemented as well as `MixWord` and they process even bits and odd bits of a 256-bit data in turn. When a 256 bits message is input, the half is processed at the cycle, and the other half is stored to the temporary register to be processed at the next cycle. The temporary register is shared with that for the constant generator.

The way to cut a step function into two comes from the structure of `MixWord`. It is a four-round Feistel ladder consisting of four XORs and four rotations. The first three rotations rotate even bits to the left, while the last one rotates 1 bit to the left. This choice of the rotations enables to separate a `MixWord` into two functions (except the last rotation). Namely, one processes the even bits in the words and the other processes the odd bits.

This architecture requires double as many cycles as Architecture 1 for processing a round.

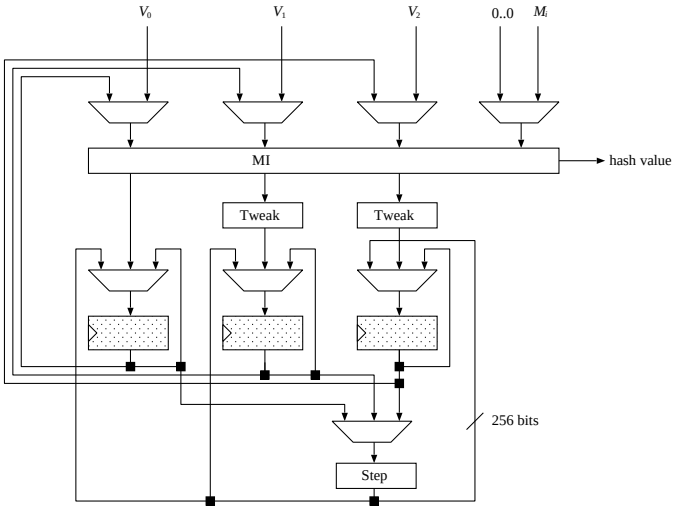

Fig. 1. Architecture 1: a step function

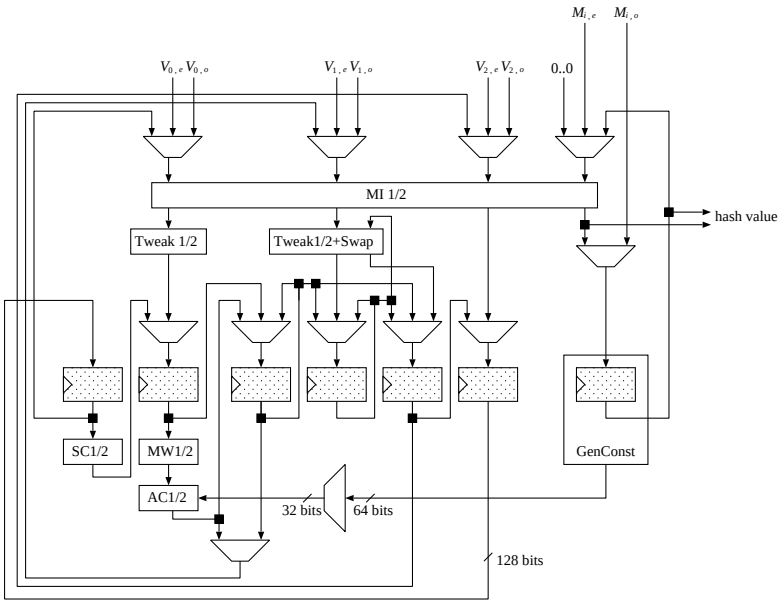

Fig. 2. Architecture 2: a half step function with a feedback shift register

3.3 Architecture 3: Six Sboxes and A MixWord

The third architecture implements only 6 Sboxes and a `MixWord` for the mixing function.

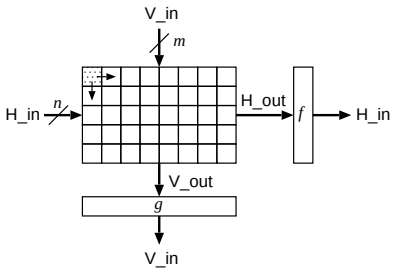

Fig. 3. The basic behavior of the 2 dimensional feedback shift register

A two dimensional array with two feedback function is suitable for the explanation of this architecture. Figure 3 shows the basic behavior of the two dimensional feedback shift register. If the register receives the signal to "shift to right", the data H_{out} pushed out from the most right hand of the register are processed by the function f, then the output of the function are feedback to the most left hand of the register. If the register receives the signal to "shift to bottom", the data V_{out} pushed out from the most lower side of the register are processed by the function g and feedback to the most upper side of the register in the same manner.

In our implementation, two 32×4 bits array is uses to store a 256-bit data H_j and they store two four words a_0, a_1, a_2, a_3 and a_4, a_5, a_6, a_7 respectively. The data pushed out to down (3×8 bits) are input to 6 Sboxes and $MI1/32$. The data pushed out to right (2×32 bits) are input to `MixWord`. Two tweaks are applied on the way from a 256-bit register to the next.

This architecture requires 32 cycles for processing `SubCrumb`, 12 cycles for `MixWord`. In addition, 32 cycles are spent for processing the message injection function MI. Therefore $32 + (32 + 12) \times 8 = 384$ cycles are required for a round processing.

4 Performances in ASIC

We wrote RTL codes in Verilog-HDL for the three architectures mentioned above and synthesized them using Synopsys Design Compiler (Version C-2009.06.SP5)[2].

[2] Synopsys and Design Compiler are registered trademarks of Synopsys, Inc. in the United States and/or other countries.

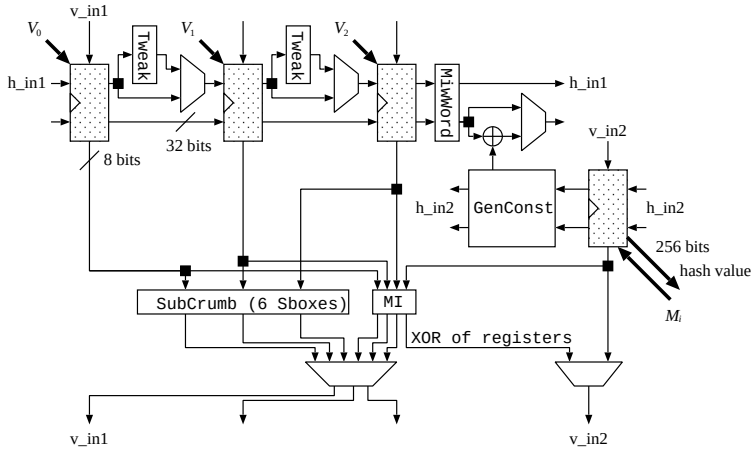

Fig. 4. Architecture 3: 6 Sboxes + 1 `MixWord` with two dimensional feedback shift register

Table 1. ASIC implementations of the *Luffa*-256

Reference	Architecture	# of cycles per round	Frequency (MHz)	Area (GE)	Throughput (Mbps)	Process
This paper	One-step	25	645	18,276	6,606	TSMC90nm
			1,163	25,683	11,907	
	Half-step	50	629	13,981	3,220	
			1,099	17,145	5,626	
	6 Sbox + 1 MixWord	384	806	10,338	538	
			1,754	11,738	1,170	

We used TSMC[3] 90 nm CMOS library for the synthesis. We set that the input and output delays are 0.4 ns. In the estimation of the throughputs, we only considered the very long message and ignored the delay due to the finalization. The throughputs are calculated according to the following equation:

$$\text{Throughput} = \frac{\text{Frequency}}{\#\text{ of Cycles}} \times 256 \text{ bits}.$$

Table 1 summarizes our implementation results and Figure 5 shows the trade-off curves between the throughputs and the sizes. These three architecture covers from 10 to 26 KGE, which achieve from 500 to 12,000 Mbps.

[3] TSMC is a registered trademark of TSMC, Ltd. in Taiwan and other countries.

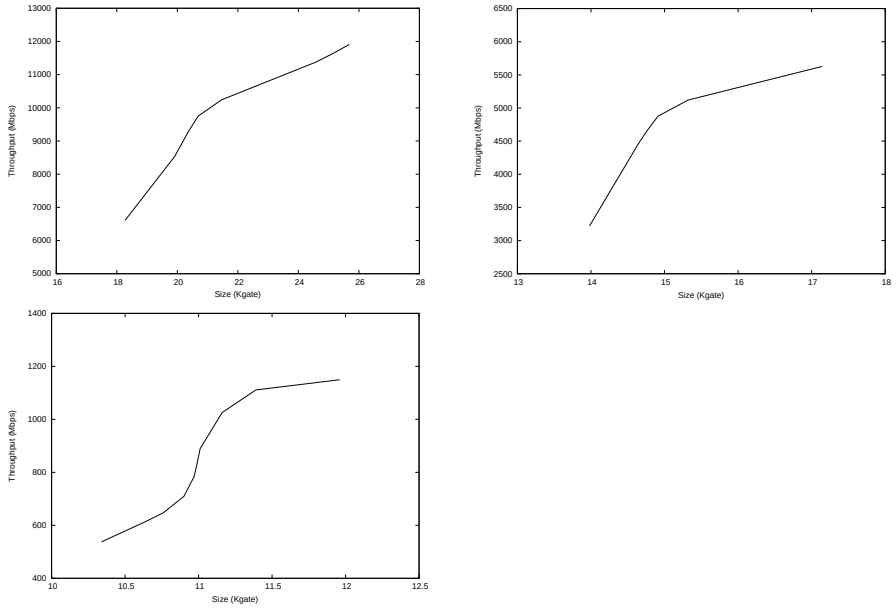

Fig. 5. Trade-off curves for three architectures: One-step (Upper-left), A Half-step (Upper-right), 6 Sboxes + 1 MixWord (Lower-left).

5 Conclusion

In this document, we proposed two new architectures of *Luffa*-256 which target size-optimized implementations in ASIC. We also implemented a known architecture which shares a step function by the three permutations. These three architectures cover from 10 to 25 KGE, which achieve from 500 to 15,000 Mbps. These results indicate that *Luffa* is quite a flexible algorithm in terms of hardware implementation.

References

1. Markus Bernet, Luca Henzen, Hubert Kaeslin, Norbert Felber, and Wolfgang Fichtner, "Hardware Implementations of the SHA-3 Candidates Shabal and CubeHash," 52nd IEEE International Midwest Symposium on Circuits and Systems, 2009. Available on-line at `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5236043`.
2. C. De Cannière, H. Sato, D. Watanabe, "Hash Function Luffa: Specification," Submission to NIST SHA-3 Competition, 2008. Available at `http://www.sdl.hitachi.co.jp/crypto/luffa/`.

3. C. De Cannière, H. Sato, D. Watanabe, "Hash Function Luffa: Supporting Document," Submission to NIST SHA-3 Competition, 2008. Available at `http://www.sdl.hitachi.co.jp/crypto/luffa/`.
4. Miroslav Knezević and Ingrid Verbauwhede, "Hardware Evaluation of the Luffa Hash Family," COSIC internal report, 2009. Also presented at WESS 2009.
5. A. H. Namin and M. A. Hasan, "Hardware implementation of the compression function for selected SHA-3 candidates," CACR 2009-28, July 2009.
6. National Institute of Standards and Technology, cryptographic hash project, `http://csrc.nist.gov/groups/ST/hash/index.html`.
7. A. Satoh, T. Katashita, T. Sugawara, T. Aoki and N. Homma, "Hardware Implementations of Hash Function Luffa, " *Hardware-Oriented Security and Trust, HOST 2010*, June 2010.
8. Hideo Shimizu and Masahiko Motoyama, "Compact hardware implementation of AES," IEICE Technical Report, ISEC 2001-149, 2001.
9. Stefan Tillich, Martin Feldhofer, Mario Kirschbaum, Thomas Plos, Jörn-Marc Schmidt and Alexander Szekely, "High-Speed Hardware Implementations of BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, and Skein," Cryptology ePrint Archive: Report 2009/510, 2009.