Evaluation of Hardware Performance for the SHA-3 Candidates Using SASEBO-GII

Kazuyuki Kobayashi[1], Jun Ikegami[1], Shin'ichiro Matsuo[2], Kazuo Sakiyama[1] and Kazuo Ohta[1]

[1] The University of Electro-Communications,
1-5-1, Chofugaoka, Chofu, Tokyo 182-8585, Japan
{k-kazu, jike, saki, ota}@ice.uec.ac.jp
[2] National Institute of Information and Communications Technology,
4-2-1 Nukui-Kitamachi, Koganei, Tokyo 184-8795, Japan
smatsuo@nict.go.jp

Abstract. As a result of extensive analyses on cryptographic hash functions, NIST started an open competition for selecting a new standard hash function SHA-3. One important aspect of this competition is in evaluating hardware implementations and in collecting much attention of researchers in this area. For a fair comparison of the hardware performance, we propose an evaluation platform, a hardware design strategy, and evaluation criteria that must be consistent for all SHA-3 candidates. First, we define specifications of interface for the SASEBO-GII platform that are suitable for evaluating the performance in real-life hash applications, while one can also evaluate the performance of the SHA-3 core function that has an ideal interface. Second, we discuss the design strategy for high-throughput hardware implementations. Lastly, we explain the evaluation criteria to compare the cost and speed performance of eight SHA-3 candidates out of fourteen.

1 Introduction

1.1 Background

Since collisions for standard hash functions were reported in 2004, much progress in researches on hash functions has been made until now. As a result of such progress, NIST decided to conduct a competition of a new standard hash function SHA-3 to select more secure and efficient hash function [1].

The selection procedure consists of a security analysis and an efficiency evaluation like the AES project held from 1997 to 2000. Currently, fourteen candidates out of sixty four submissions are selected in the second round, and the smaller number of algorithms will be selected as the finalist of the competition in 2010. After that, NIST will select a SHA-3 algorithm from the finalist according to the evaluation results on security and efficiency with considering a target application and a platform on which the hash function is executed.

In the first round of the competition, the evaluation was mainly taken place on design criteria, algorithm security and implementation. This evaluation on

implementation was basically on software using a reference platform. In the first SHA-3 candidate workshop held in 2009, the evaluation for many kinds of hardware applications, such as server, PC, smart phone, smart card and RFID, were extensively discussed. Though there are some consensus on a fair evaluation for software implementations, discussion for hardware implementations is still immature. There are much research on hardware evaluation of SHA-3 candidates[2],[3],[4],[5], however, validity and consistency of the evaluation criteria and methods of such research are not well discussed yet. In order to conduct a hardware evaluation, we need to fix an evaluation environment (*i.e.* platform), implementation method (*i.e.* design strategy), and a performance comparison method (*i.e.* evaluation criteria). The consensus on such points is required for a fair hardware evaluation.

Although it is preferable to have a common and solid evaluation criterion, it seems difficult to make it realized since there are many aspects on hash usage. Hash functions are used in many cryptographic protocols such as SSL and SSH, and are implemented into a wide range of application platforms. There may be many combinations of such usages in real-life systems, and thus there may be several different criteria which are matched with each aspect. In this situation, it is a good solution that evaluators prepare evaluation criteria and evaluation reports for major usages, and then NIST can prioritize these evaluation criteria from usage of hash functions. As a result, NIST can select a SHA-3 algorithm based on the evaluation in a real-life hash usage and a platform by taking all of the reports into account. In this sense, the prime requirement should be "the criteria are consistent and fair in some aspect."

1.2 Our Contribution

In this paper, we propose a platform, a design strategy, and evaluation criteria for a fair hardware evaluation on the SHA-3 candidates. They include an interface specification, an architecture of SHA-3 hardware, and methods for performance evaluation. The platform used in our experiments is able to support different types of hash algorithms.

We use SASEBO-GII which is publicly available and is a promising standard platform for hardware evaluations [6]. According to our proposal, we can evaluate the hardware performance independent of interface specifications, and therefore our proposal covers a various types of hardware architectures. The hardware designs we implemented in Verilog can be reused for a future ASIC development on the SASEBO platform.

We report the results of the FPGA hardware evaluation on eight hash functions out of the fourteen second-round candidates.

2 Overview of Hardware Evaluation Platform for SHA-3 Candidates

In this section, we introduce our proposed evaluation platform using SASEBO-GII. Figure 1 shows the overview of the platform. SASEBO-GII (Side-channel

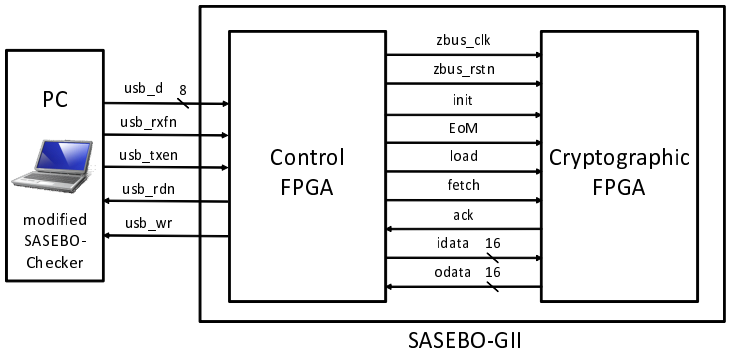

Fig. 1. Evaluation Environment Using SASEBO-GII.

Attack Standard Evaluation Board) [6] has a communication interface with a PC and two FPGAs. They are a control FPGA and a cryptographic FPGA. Message data to be hashed are transmitted to the control FPGA by a software called SASEBO-Checker on a PC through USB. We used a modified SASEBO-Checker based on "Quick Start Guide" developed by AIST [6]. The control FPGA controls the data flow to send a message to the cryptographic FPGA, where hash operations are performed. After the hash operations, the hash value is sent back to SASEBO-Checker through the control FPGA. As illustrated in Fig. 1, we define a specification of the interface between the control FPGA and the cryptographic FPGA.

The control FPGA checks the latency of a single hash operation for an input data that is performed in the cryptographic FPGA and reports the number of clock cycles to SASEBO-Checker. More precisely, SASEBO-Checker reports two kinds of performance metrics for evaluating the performance of a hash calculation. One is the number of clock cycles including the cycles for receiving input data and the other is one excluding the cycles for the data input.

2.1 Interface between Control and Cryptographic FPGAs

In this section, we explain an interface and a communication protocol between two FPGAs.

The performance of SHA-3 hardware implementations heavily depend on the communication overhead, *i.e.* the time used for receiving and transferring data. On the one hand, the communication overhead is likely to be a bottleneck of a system in practice. Therefore, one would say that the interface specification should be realistic, *e.g.* 32-bit data are sent in two cycles. On the other hand, one tends to assume an ideal interface because one would like to maximize the performance of a hash core. In this case, one ignores the communication overhead and the hardware design would be impractical in general.

For a fair comparison, we employ a compromise plan. Namely, we use a practical interface that can support a 16-bit data communication in three cycles. However, if needed, we ignore or reduce the communication overhead when calculating the speed performance. In this way, we can evaluate the SHA-3 hardware performance depending on how the hash core is interfaced. In the followings, we explain our interface in detail.

Communication Protocol between Control and Cryptographic FPGAs The interface between the control and cryptographic FPGAs is based on work by Chen *et al.* proposed in [7]. In order to guarantee a stable communication between two FPGAs, we invert the clock signal of the control FPGA and supply it to the cryptographic FPGA. The init signal shown in Fig. 1 is a signal to initialize a hash function in the cryptographic FPGA. The load and fetch signals are used for transmitting and receiving the message data and the hash value between the control and cryptographic FPGAs, and the ack signal is a response signal for the load and fetch signals. The input and output data for the cryptographic FPGA are sent in a unit of 16 bits via idata and odata signals, respectively.

With this protocol, it takes three cycles for transmitting and receiving one 16-bit data between the FPGAs. Therefore, we can calculate the precise number of clock cycles for data communications by using the total length of input or output data, *e.g.* it takes $3 \cdot \frac{256}{16}$ or 48 cycles for 256-bit input data. When a hash core can support a two-cycle protocol, we can simply reduce 16 from the obtained communication overhead for the 256-bit input data to derive a potential performance.

3 Design Strategy for Hardware Implementation of SHA-3 Candidates

3.1 Specification of Data Input to Cryptographic FPGA

The detailed specification for data input sent to the cryptographic FPGA is described in this section.

Message Padding A hash function executes the hashing process for each input data with a constant block size. Then, the hash function uses the intermediate result as the next input data to proceed the next hashing process. Therefore, the size of the input data must be a multiple of the block size. If they are not, the hash function fixes the input data to be the multiple of the block size by padding. In this work, we assume that padding is performed before sending a message from SASEBO-Checker.

EoM (End of Message) There are several SHA-3 candidates which need an extra output calculation process in addition to a normal hashing process.

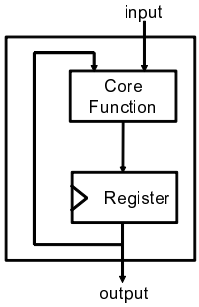

Fig. 2. Fully Autonomous

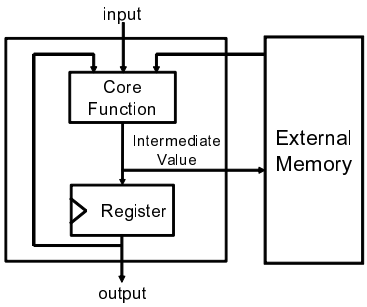

Fig. 3. External Memory

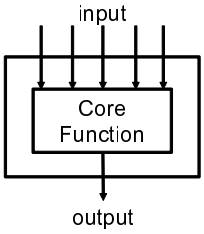

Fig. 4. Core Functionality

Therefore, the control FPGA sends the end of message to the cryptographic FPGA by raising the EoM signal.

Bit Length of Message The bit length of the message is sent via idata. Therefore, the data communication overhead happens for the SHA-3 candidates which need to send information about the bit length, so that it could require more time than the candidates which do not need the bit length of a message.

3.2 Architectures of Cryptographic FPGA

Hardware architectures to implement a hash function are described in this section. There are mainly three types of architectures which, in our case, are implemented on the cryptographic FPGA [8].

Fully Autonomous (Fig. 2) In this architecture, one transfers message data to a hash function in multiple cycles in a unit of a fixed data size (*e.g.* 16 bits). The hash module stores all of the intermediate values in registers during the hashing process. Therefore, after all message data are input, the hash function can start executing the process. It can be said that this method is regarded as an implementation assuming to be used in a real system.

External Memory (Fig. 3) In this architecture, only the data necessary for executing the hashing calculation are stored in registers. Other data that are not related to the part of the hash calculation are stored in an external memory that is less expensive than registers in general. Therefore, the hash function hardware becomes a low-cost implementation. However, the architecture requires overhead cycles for accessing an external memory, and hence it is not suitable for high-speed implementations.

Core Functionality (Fig. 4) This architecture has only the core part of a hash function. The hashing process is executed supposing that data are provided from an external module in some way. Therefore, one can estimate a rough performance of the hash function hardware. In other words, it can be said that this architecture is used for estimating a hardware performance under an ideal interface where the overhead of the data access is ignored in the hashing process.

The previous work for evaluating hardware performance has been executed without using a standardized architecture, *i.e.* different architectures are used. For example, the implementation method by Namin *et al.* [3] and Baldwin *et al.* [4] is based on the core functionality type and they a evaluate rough estimate of the performance of hash function hardware. On the other hand, the implementation method by Tillich *et al.* [2] and Jungk *et al.* [5] is based on the type of the fully autonomous. They assumed that the input data for the hash function hardware is sent in one cycle, so that the length of the input data is assumed long (*e.g.* 256 or 512 bits). Consequently, their evaluation results cannot be used straightforwardly for a performance evaluation of an accelerator of CPU where only a limited size of data access is available in one.

In this paper, we evaluate the performance of SHA-3 candidates when they are used in a real system. In addition, we prepare an evaluation environment that one could evaluate hardware performance even if using a different interface. In the case of evaluating hardware performance using SASEBO-GII, it is natural to employ the architecture of Fig. 2 or 3 because the architecture of Fig. 4 needs long bits length for input and output the data. In addition, as our proposal is to evaluate the performance for high-speed hardware implementations, we decide to employ the architecture of Fig. 2, because the architecture of Fig. 3 takes extra cycles for memory accesses, and hence it is not suitable for high-speed implementation.

The input and output data between the control and cryptographic FPGAs are assumed to be 16-bit length. This is reasonable when we assume the application such as an accelerator of CPU. One can measure the number of clock cycles when assuming an ideal interface that does not cause communication overheads for an input message by using SASEBO-Checker. Furthermore, one can evaluate hardware cost when using the architecture of the core functionality by synthesizing only the core function part of a hash function hardware.

Figure 5 shows the detailed architecture of the cryptographic FPGA which we use for evaluating hardware performance. The cryptographic FPGA consists

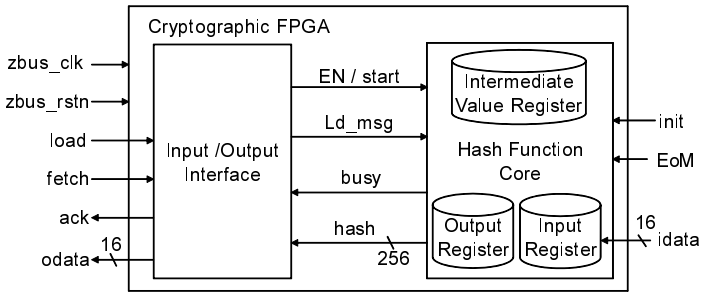

Fig. 5. Architecture of Cryptographic FPGA.

of an interface block which controls input and output, and a core function block which executes a hashing process. There are several SHA-3 candidates which need to keep an input message during the hashing process. In our environment, it is able to prepare a message register in the core function block. Also, we may be able to prepare an output register in the core function block which keeps hashed values.

3.3 Hardware Performance

We mainly focus on improving the throughput of hardware implementation in this paper. As every SHA-3 candidate has a different input block size, the throughput can be expressed as

$$Throughput = Input\ Block\ Size \cdot \frac{Max\ Clock\ Frequency}{Number\ of\ Clock\ Cycles}. \quad (1)$$

The *Input Block Size* expresses the size of an input data and *Number of Clock Cycles* expresses the number of clock cycles which is necessary to hash the input data. *Max Clock Frequency* can be defined as a reciprocal of a critical path delay of a combinational logic. As can be seen from Eq. (1), it is necessary to increase *Max Clock Frequency* and/or decrease *Number of Clock Cycles* to improve *Throughput*. In general, improving *Throughput* tends to be more costly in terms of hardware resource.

3.4 Techniques to Improve Throughput

Our proposal covers the following techniques for implementing an optimized hash function.

Retiming Transformation The retiming transformation is a technique that Lee *et al.* used for hash hardware implementations in [9]. This technique holds down the critical path delay to improve the throughput by averaging a processing

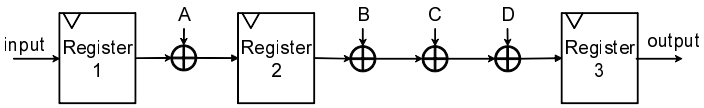

Fig. 6. An Example DFG.

time between two registers. We use a DFG (Data Flow Graph) shown in Fig. 6 as an example case to explain the technique. The critical path of the example DFG consists of three adders between the registers 2 and 3. Because the maximum clock frequency depends on the critical path delay, it should be shorten in order to improve the throughput.

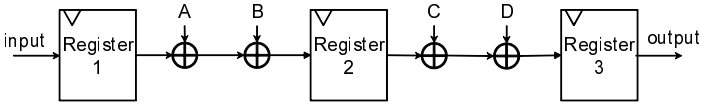

Fig. 7. Applying Retiming Transformation to Fig. 6.

Figure 7 shows an optimized DFG where the retiming transformation is applied to the DFG in Fig. 6. In this case, the critical path consists of two adders, and therefore the maximum clock frequency improves. This technique is applied to SHA-256 which is used as a reference design in our experiments.

Unfolding Transformation The unfolding transformation is a technique that Lee *et al.* also used in [9]. This technique decreases the total number of clock cycles. Therefore we can expect the improvement of the throughput. Figure 8 shows a DFG after applying the unfolding transformation to the DFG in Fig. 6.

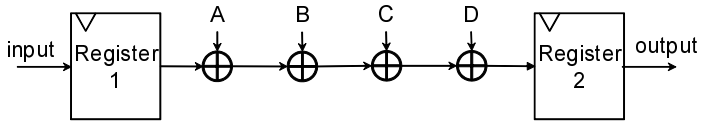

Fig. 8. Applying Unfolding Transformation to Fig. 6.

The DFG shown in Fig. 8 performs four operations addition with A, B, C and D in one cycle. The throughput of this DFG might improve although the

maximum clock frequency becomes lower. Namely, suppose that the maximum clock frequency becomes about $\frac{4}{3}$ times lower and the number of clock cycles becomes 2 times smaller, the throughput could improve by a factor of $\frac{3}{4} \cdot 2$ or 1.5. However, the improvement depends on the type of the combinational logics. Therefore, this technique has to be applied for architectures for which we can expect a throughput improvement even if the critical path delay becomes longer. In this work, we apply this technique only to Skein.

3.5 Hardware Design Strategy

We show how to deal with the optimization techniques discussed in Sect. 3.4 as follows.

Initial Value There are several SHA-3 candidates which have a procedure to generate an initial value. We assume that the initial value is calculated before hash operations.

Criteria of Applying the Unfolding Transformation The criteria of applying the unfolding transformation to a hash function are described in this section. We use the following notations to explain the criteria.

B : Input block size,
I : Total number of clock cycles,
D : Critical path delay,
Th : Throughput,
f_{max} : Maximum clock frequency,
M : Message without padding,
M_p : Message with padding,

Figures 9 and 10 show a representative example of the DFG where the unfolding transformation could be applied. S, S_0, and S_1 denote the hardware costs of a combinational logic and T_d, T_{d_0}, and T_{d_1} denote the path delays of a combinational logic. These combinational circuits output the result in I_0 cycles.

Figures 11 and 12 show the DFGs after the unfolding transformation against Fig. 9. The throughput of the case of Fig. 9 is

$$Th_1 = B_0 \cdot \frac{1/T_d}{I_0}. \tag{2}$$

On the contrary, the throughput of the case of Fig. 11 is

$$Th_a = B_0 \cdot \frac{1/T_d}{I_0/a} = a \cdot Th_1. \tag{3}$$

In this case, the hardware cost of the combinational logic becomes a times expensive, but the throughput improves a times higher.

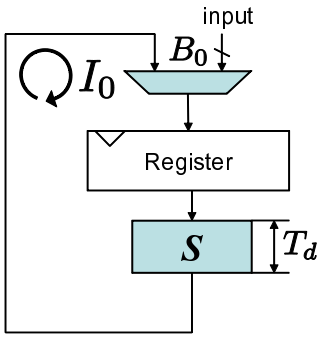

Fig. 9. Example DFG before Applying the Unfolding Transformation (1).

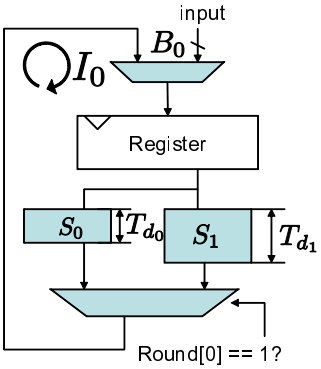

Fig. 10. Example of DFG before Applying the Unfolding Transformation (2).

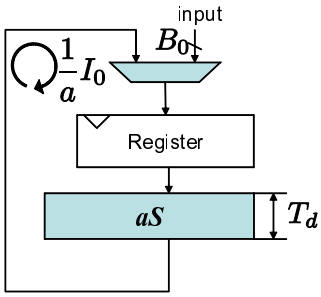

Fig. 11. After Applying the Unfolding Transformation against Fig. 9 (a).

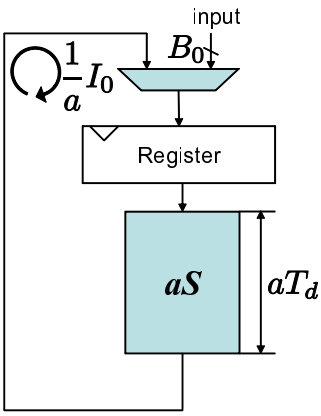

Fig. 12. After Applying the Unfolding Transformation against Fig. 9 (b).

As for Fig. 12, the throughput is

$$Th_b = B_0 \cdot \frac{1/(a \cdot T_d)}{I_0/a} = Th_1. \tag{4}$$

Therefore, the throughput is the same as Fig. 9, although the hardware cost of the combinational logic becomes a times expensive.

Figure 10 is the case where two different operations are performed in every alternative cycle. The critical path of the DFG is $T_{d_1}(T_{d_1} > T_{d_0})$. The throughput of this case is

$$Th_1 = B_0 \cdot \frac{1/T_{d_1}}{I_0}. \tag{5}$$

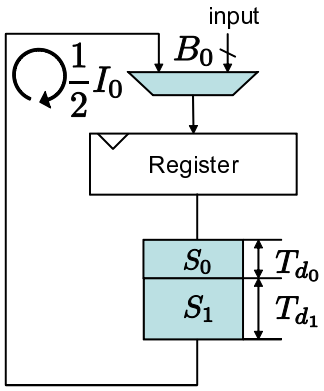

Fig. 13. After Applying the Unfolding Transformation against Fig. 10 (c).

Figure 13 shows the DFG after the unfolding transformation against Fig. 10, and the throughput is

$$Th_a = B_0 \cdot \frac{1/(T_{d_0} + T_{d_1})}{I_0/2} = \frac{2 \cdot T_{d_1}}{T_{d_0} + T_{d_1}} \cdot Th_1 > Th_1. \tag{6}$$

As a result, the throughput improves, while the hardware cost of the combinational logic is almost the same as Fig. 10.

To give a high priority to throughput improvements, we perform the unfolding transformation to the SHA-3 candidates which can raise the throughput such as Figs. 11 and 13.

4 Evaluation Criteria for Hardware SHA-3 candidates

4.1 Evaluation Items

We implement eight SHA-3 hash candidates on the cryptographic FPGA, Xilinx Virtex-5 (xc5vlx30-3ff324) on SASEBO-GII. We check the hardware performance in terms of speed and hardware cost. The speed performance is evaluated with its latency or throughput that is calculated with the input block size, the maximum clock frequency, and the total number of clock cycles with or without the communication overheads. The cost performance is evaluated with the number of slices, registers, LUTs and the size of a single-port RAM used as ROM. A hash function which has a high throughput with a low hardware cost is regarded as efficient.

4.2 Evaluation Metrics

A hash function executes a hashing process for each data with a input block size, and uses the result as the next input data to proceed the whole hashing process. The clock cycles necessary for hashing $|M|$-bit data can expresses as

$$I = \frac{|M_p|}{B}(I_{in} + I_{core}) + I_{final} + I_{out} \ . \tag{7}$$

Here, $\frac{|M_p|}{B}$ is the number of hash core operations when the hash core can perform B-bit data in one operation. I_{in}, I_{core}, I_{final} and I_{out} denote the number of clock cycles used to input data, to execute hashing process in the core function block, to perform the final calculation process and to output the hash results, respectively. Note that the coefficients of I_{final} and I_{out} are both ones, because these processes are only executed when outputting the resultant data.

As a result, the throughput and the latency can be expressed as

$$Th = |M_p| \times \frac{f_{max}}{\frac{|M_p|}{B}(I_{in} + I_{core}) + I_{final} + I_{out}}, \tag{8}$$

$$L = \frac{|M_p|}{Th} \ . \tag{9}$$

When $|M_p|$ is sufficiently large, for example in the case of hashing a long message, I_{final} and I_{out} can be negligible from Eq. (8). In this case, the throughput $Th_{LongMessage}$ is approximated as

$$Th_{LongMessage} = \frac{Bf_{max}}{I_{in} + I_{core}}. \tag{10}$$

On the other hand, when $|M_p|$ is small, for example in the case of hashing a short message for authentication, we cannot ignore I_{final} and I_{out}. Moreover, as the latency is an important metrics for a short message rather than the throughput, we use Eq. (9) to compare the speed performance of the SHA-3 candidates.

Table 1 shows the evaluation metrics. Here, the throughput of the core function block Th_{core} is

$$Th_{core} = |M_p| \times \frac{f_{max}}{\frac{|M_p|}{B} I_{core} + I_{final}}. \tag{11}$$

4.3 Implementation Results for Eight SHA-3 Candidates

In this work, we implement SHA-256 and eight SHA-3 candidates aiming at a high-speed hardware implementation.

Tables 2 and 3 summarize the results of the implementation for the SHA-3 candidates. Skein, CubeHash, Grøstl, Hamsi, Shabal and Luffa have the output calculation processes, so that these candidates need the EoM signal as explained

Table 1. Evaluation Metrics.

	Long Message (Throughput)	Short Message (Latency)
Interface + Core	$\frac{B \cdot f_{max}}{I_{in}+I_{core}}$	$\frac{\lvert M_p \rvert}{Th}$
Core Function Block	$\frac{B \cdot f_{max}}{I_{core}}$	$\frac{\lvert M_p \rvert}{Th_{core}}$

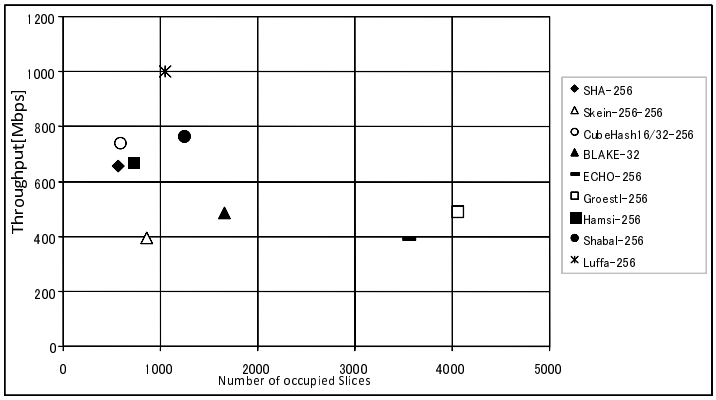

Fig. 14. Throughput of Hash Function Including Interface Overheads for Long Message.

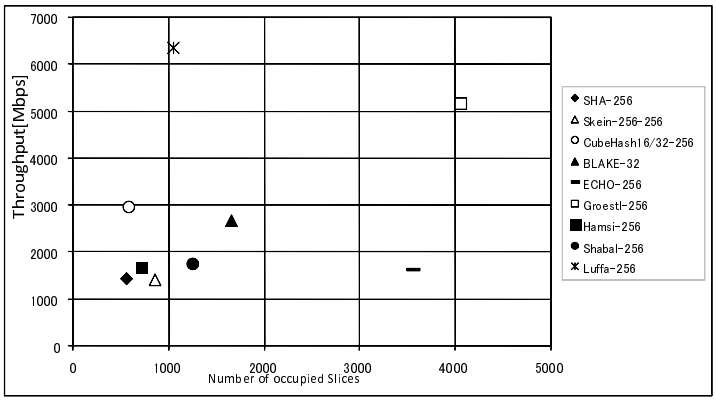

Fig. 15. Throughput of Hash Core Function Block for Long Message.

in Sect. 3.1. Skein, BLAKE and ECHO use the bit length of a message, and therefore these candidates require extra clock cycles when sending the message.

Figures 14 and 15 show the cost performance in the case for a long message. From these results, we consider Luffa is the most efficient hash function

Table 2. Performance Results of the SHA-3 Candidates on Virtex-5 (xc5vlx30-3ff324).

	Input Block Size [bits]	Max. Clock Frequency [MHz]	Total Number of Clock Cycles [cycles]		Long Message *3	Short Message *3
			Interface + Core *1	Core Function *2	Throughput [Mbps]	$\|M\| = 1024$ Latency [μs]
SHA-256	512	190	148 (199)	68 (68)	657 (1431)	2.61 (1.074)
Skein	256	115	75 (146)	21 (41)	393 (1482)	3.23 (0.904)
CubeHash	256	185	64 (275)	16 (176)	740 (2960)	2.87 (1.297)
BLAKE	512	115	121 (172)	22 (22)	487 (2676)	3.60 (0.574)
ECHO	1536	104	407 (458)	99 (99)	392 (1614)	4.40 (0.952)
Grøstl	512	101	106 (167)	10 (20)	488 (5171)	3.75 (0.396)
Hamsi	32	210	12 (63)	4 (7)	560 (1680)	1.92 (0.681)
Shabal	512	214	143 (345)	63 (214)	766 (1739)	2.95 (1.589)
Luffa	256	223	57 (117)	9 (18)	1002 (6343)	1.55 (0.242)

*1 : $I_{in} + I_{core}$ (I). *2 : I_{core} ($I_{core} + I_{final}$). *3 : including communication overheads by interface. Values in parenthesis are the case excluding the overheads, *e.g.* Throughput of the core function block.

Table 3. Hardware Costs of the SHA-3 Candidates on Virtex-5 (xc5vlx30-3ff324).

	Number of Occupied Slices	Number of Slice Registers	Number of Slice LUTs	Block RAM [bits]
SHA-256	561	1177	1969	64×32
Skein	854	929	2864	0
CubeHash	590	1316	2182	0
BLAKE	1660	1393	5154	0
ECHO	3556	4198	11668	0
Grøstl	4057	1570	13222	0
Hamsi	718	841	2499	0
Shabal	1251	2061	4219	0
Luffa	1048	1446	3754	0

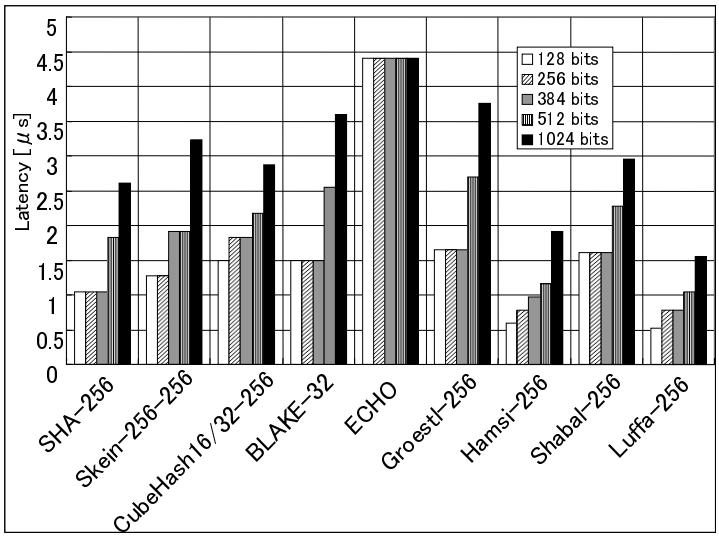

Fig. 16. Latency of Hash Function Including Interface Overheads for Short Message.

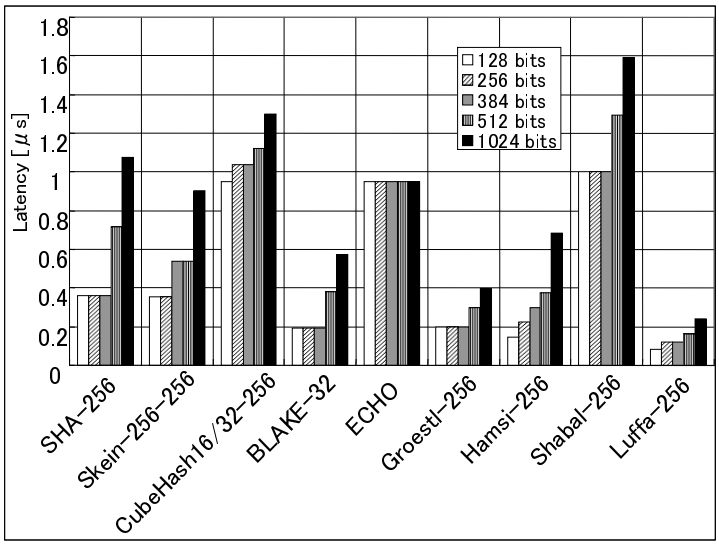

Fig. 17. Latency of Core Function Block for Short Message (128, 256, 384, 512, 1024 bits).

for a long message. Figures 16 and 17 show the latency of the hash function with interface overheads and the core function block in the case of $|M| =$

128, 256, 384, 512, 1024. We evaluate Luffa is the most efficient hash function also for short message in our experiments.

In addition, as the input block size of ECHO is 1536 bits, the total processing time is the same when hashing data whose size is less than 1536 bits. Therefore, ECHO might have a disadvantage of hashing a short message such as authentication.

From our result, the hash core function has a less effect for the throughput differences because the input and output processes are likely to be a bottleneck. Therefore, the performance of an interface could be an important part for a fair hardware comparison.

5 Conclusions

In this paper, we firstly propose a consistent evaluation criteria which realizes a fair hardware evaluation on cryptographic hash function. This criteria consist of the basic design of an evaluation environment using SASEBO-GII, the interface specification between two FPGAs, and architecture of hardware implementations of hash function. We also propose evaluation items and the way to evaluation.

We implement eight out of fourteen SHA-3 candidates using the above environment and show the results. Our future work includes evaluation of the rest of the SHA-3 candidates, evaluation for low-power devices such as RFID tags and security evaluation of HMAC using SHA-3 candidates against side-channel attacks.

Acknowledgments

The authors would like to thank Akashi Satoh and Research Center for Information Security, National Institute of Advanced Industrial Science and Technology (AIST) for developing the SASEBO-GII.

References

1. National Institute of Standards and Technology (NIST), "Cryptographic Hash Algorithm Competition," `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`.
2. S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J. -M. Schmidt, and A. 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.
3. A. H. Namin and M. A. Hasan, "Hardware Implementation of the Compression Function for Selected SHA-3 Candidates," CACR 2009-28 (2009).
4. B. Baldwin, A. Byrne, M. Hamilton, N. Hanley, R. P. McEvoy, W. Pan, and W. P. Marnane, "FPGA Implementations of SHA-3 Candidates:CubeHash, Grøstl, Lane, Shabal and Spectral Hash," Cryptology ePrint Archive, Report 2009/342, 2009.

5. B. Jungk, S. Reith, and J. Apfelbeck, "On Optimized FPGA Implementations of the SHA-3 Candidate Grøstl," Cryptology ePrint Archive, Report 2009/206, 2009.
6. "National Institute of Advanced Industrial Science and Technology (AIST), Research Center for Information Security (RCIS) : "Side-channel Attack Standard Evaluation Board (SASEBO)"," http://www.rcis.aist.go.jp/special/SASEBO/SASEBO-GII-ja.html.
7. Z. Chen, S. Morozov, and P. Schaumont, "A Hardware Interface for Hashing Algorithms," Cryptology ePrint Archive, Report 2008/529, 2008.
8. ECRYPT II, "SHA-3 Hardware Implementations," http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations.
9. Y. K. Lee, H. Chan, and I. Verbauwhede, "Iteration Bound Analysis and Throughput Optimum Architecture of SHA-256 (384, 512) for Hardware Implementations," in *In Information Security Appliciations, 8th International Workshop, WISA 2007*, vol. 4867 of *LNCS*, pp. 102-114, Springer, 2007.