Open access • Journal Article • DOI:10.1109/TVLSI.2011.2128353

Fair and Consistent Hardware Evaluation of Fourteen Round Two SHA-3 Candidates

Miroslav Knezevic, K. Kobayashi, Jun Ikegami, Shin'ichiro Matsuo ...+10 more authors

Institutions: Katholieke Universiteit Leuven, Tohoku University

Published on: 01 May 2012 - IEEE Transactions on Very Large Scale Integration Systems (IEEE)

Topics: Field-programmable gate array, Application-specific integrated circuit and Gate array

Fair and Consistent Hardware Evaluation of Fourteen Round Two SHA-3 Candidates

Miroslav Knežević*, Kazuyuki Kobayashi†, Jun Ikegami†, Shin'ichiro Matsuo‡, Akashi Satoh¶, Ünal Kocabaş*, Junfeng Fan*, Toshihiro Katashita¶, Takeshi Sugawara§, Kazuo Sakiyama†, Ingrid Verbauwhede*, Kazuo Ohta†, Naofumi Homma§, Takafumi Aoki§

Abstract—The first contribution of our paper is that we propose a platform, a design strategy and evaluation criteria for a fair and consistent hardware evaluation of the second-round SHA-3 candidates. Using a SASEBO-GII FPGA board as a common platform, combined with well defined hardware and software interfaces, we compare all 256-bit version candidates with respect to area, throughput, latency, power and energy consumption.

Our approach defines a standard testing harness for SHA-3 candidates, including the interface specification for the SHA-3 module on our testing platform. The second contribution is that we provide both FPGA and 90 nm CMOS ASIC synthesis results and thereby are able to compare the results. Our third contribution is that we release the source code of all the candidates and by using a common, fixed, publicly available platform, our claimed results become reproducible and open for a public verification.

Index Terms—Hash Function, SHA-3 Competition, Hardware Evaluation, FPGA, ASIC, SASEBO-GII.

1 Introduction

Since collisions on standard hash functions were reported in 2004 [1], [2], improvements to hash attack methods and improvements to hash algorithms have been investigated at a similar, rapid pace [3]. For this reason, NIST decided to initiate the development of a new hash standard. Similar to the development of the present block cipher standard – AES, NIST uses a competition model that has been proved to assure a fair selection among various candidates [4].

The competition is organized in three phases, with the second phase scheduled to complete by the end of summer 2010. Out of the original 64 submissions to the first phase, fourteen candidates have been selected for detailed analysis in the second phase (BLAKE, BMW, CubeHash, ECHO, Fugue, Grøstl, Hamsi, Keccak, JH, Luffa, Shabal, SHAvite-3, SIMD, Skein). NIST will then reduce this set to an even smaller number during the third, final phase.

The selection of winning candidates is driven by considering security properties as well as implementation efficiency of the proposed hash algorithms both in hardware and software. However, a systematic cryptanalysis of hash functions is not well established, and it is hard to measure the cryptographic strength of a hash function beyond obvious metrics such as digest length. For this reason, the implementation efficiency of hardware and software plays a vital role in the selection of the finalist.

*Katholieke Universiteit Leuven, ESAT/SCD-COSIC and IBBT, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium, Email: {mknezevi, ukocabas, jfan, iverbauw}@esat.kuleuven.be

†The University of Electro-Communications, 1-5-1, Chofugaoka, Chofu, Tokyo 182-8585, Japan, Email: {k-kazu, jike, saki, ota}@ice.uec.ac.jp

‡National Institute of Information and Communications Technology, 4-2-1 Nukui-Kitamachi, Koganei, Tokyo 184-8795, Japan Email: smatsuo@nict.go.jp

§Graduate School of Information Sciences, Tohoku University Aoba 6-6-05, Aramaki, Aoba-ku, Sendai, 980-8579, Japan

¶Research Center for Information Security, National Institute of Advanced Industrial Science and Technology, 1-18-13, Sotokanda, Chiyoda, Tokyo 101-0021, Japan, Email: {akashi.satoh, toshiro.katashita}@aist.go.jp

There are several projects that have evaluated the hardware efficiency of the SHA-3 candidates [5], [6], [7], [8], [9]. However, the validity and consistency of the evaluation criteria and methods of such research are not well discussed yet. In order to evaluate the hardware efficiency over a set of SHA-3 candidates, we need to fix an evaluation environment (i.e., platform), an implementation method (i.e., design strategy), and a performance comparison method (i.e., evaluation criteria). A consensus on such points is required for a fair and consistent comparison.

The performance evaluation of hardware, including the measurement of power consumption, execution time, and hardware resources, is a rather complex problem. There are several reasons for this. Most importantly, the design space for hardware performance evaluation is larger than that of software. Additional design constraints (such as low-area, max-throughput, and min-energy) are required to define an optimal implementation. Second, accurate and generic performance evaluation metrics are hard to obtain. A throughput can be characterized provided that the hardware design can be accurately timed. The area metrics depend strongly on the target technology (ASIC/FPGA). A measurement of the power consumption is the most difficult, and it is almost never mentioned in publications.

In this paper we try to address most of these issues and therefore, we summarize our contributions as follows.

- First, we propose a platform, a design strategy, and evaluation criteria for a fair and consistent hardware evaluation of the SHA-3 candidates.
- Second, we use a prototyping approach by mapping each of the 256-bit version hash candidates onto a SASEBO-GII FPGA board [10]. The hash candidates are then evaluated with respect to throughput, latency, hardware cost, and power and energy consumption.
- Third, we provide synthesis results in 90 nm CMOS technology with respect to throughput and circuit size. In addition, we provide power and energy consump-

tion estimates.
- Finally, by releasing the source code of all the candidates and by using a common, fixed, publicly available platform, our claimed results become reproducible and open for public verification.

1.1 Related Work

Recently, several research groups have proposed comprehensive performance evaluation methods, which evaluate a set of hash algorithms on a common platform.

- Tillich et al. [11] developed RTL VHDL/Verilog code for all SHA-3 candidates. They present synthesis results in 180 nm CMOS technology. In order to reach the highest degree of accuracy, they further perform the place & route for the best versions of all fourteen candidates [5].
- Gaj et al. [6] developed a scripting system called ATHENa, targeted towards FPGA. A fair comparison is achieved by defining a standard interface and by automatic design space exploration. Furthermore, in [12] they report a comparison of all 512-bit version SHA-3 candidates using the same methodology.
- Baldwin et al. [13] propose a standard interface to achieve a fair comparison and illustrate their approach by providing the hardware figures for all fourteen SHA-3 candidates. They evaluate hardware designs and test for all message digest sizes (224, 256, 384, and 512 bits) and also include the padding as part of the hardware for the SHA-3 hash functions.
- Henzen et al. [8] evaluated all fourteen second-round SHA-3 candidates using 90 nm CMOS technology. All designs were placed & routed and the post-layout figures were reported.
- Guo et al. [9] presented post place & route figures for all fourteen candidates in 130 nm CMOS technology.

2 General Requirements for Hardware Evaluation

In this section, we reconsider the main requirements for conducting a fair and consistent hardware evaluation of the fourteen SHA-3 candidates.

First, we comment on the feasibility of compact implementations. Second, we discuss the speed performance metrics and power/energy consumption. Then, we open a question concerning fair comparison and consistent hardware evaluation of the remaining SHA-3 candidates. Finally, we present an attempt to classify the candidates with respect to their design properties. This classification will be useful, later on, for drawing some conclusions and comparing different candidates.

2.1 Area: Lower Bound on Compact Implementations

Depending on the application scenarios, one of the decision points, prior to starting with the hardware evaluation, is a choice of the actual architecture. Therefore, we provide a lower bound estimation on each of the fourteen candidates and argue that, given the required security margins, there are no candidates suitable for a lightweight implementation. Our estimation is simply based on the minimum amount of total memory needed for a certain algorithm. We define the state size to be the size of the chaining variable (see Table 1). We also refer to the work of Ideguchi et al. [14], that studies the RAM requirements of various SHA-3 candidates for the low-cost 8-bit CPUs. Furthermore, we estimate the size of the required memory with respect to the number of gate equivalences (GE), which represents the lower bound size. Finally, we provide figures for current, compact implementations of some of the second-round candidates.

TABLE 1
Memory Requirements for the SHA-3 Candidates.

Candidate	State Size [bit]	Total Memory [14] [bit]	Total Memory† [GE]	Total Area [GE]
BLAKE	512	768	4,608	13,560 [15]
BMW	512	1,536	9,216	N/A‡
CubeHash	1,024	1,024	6,144	7,630 [16]
ECHO	2,048	2,560	15,360	82,800 [17]
Fugue	960	960	5,760	59,220 [18]
Grøstl	512	1,024	6,144	14,620 [19]
Hamsi	512	768	4,608	N/A‡
JH	1,024	1,024	6,144	N/A‡
Keccak	1,600	1,600	9,600	N/A‡
Luffa	768	768	4,608	10,340 [20]
Shabal	1,408	1,408	8,448	23,320 [16]
SHAvite-3	896	1,024	6,144	N/A‡
SIMD	512	3,072	18,432	N/A‡
Skein	512	768	4,608	N/A‡

Estimates for versions with 256-bit digest size are given.
† We estimate the size of a single flip-flop to be 6 GE.
‡ To the best of our knowledge, as of November 2010, these candidates had no published figures for low-cost hardware implementations.

Comparing the lower bound size of all fourteen candidates with the size of state of the art lightweight block ciphers, e.g., PRESENT [21] and KATAN & KTANTAN [22], we conclude that all candidates are rather suited for a so-called *welterweight* category. Therefore, in this work, we focus only on the high-throughput variants of all second-round candidates.

2.2 Speed: Latency versus Throughput

Regarding the speed of a hash candidate, we distinguish two performance figures. Depending whether the input message is a long (we consider very long messages in this case) or a short one (e.g., 256 bits or less), we evaluate the throughput and the latency, respectively. The throughput is defined as the amount of information processed per unit of time (bits/s), while the latency represents the time delay necessary for processing a certain amount of information from start to end (s).

This approach provides a fair comparison and an accurate evaluation for each of the candidates. In both cases, the speed performance is a function of several factors: maximum frequency, number of clock cycles necessary for a hash operation, number of cycles necessary for input and output, and the input block size. Furthermore, the latency also depends on the message size and the presence of

the finalization function. Later, in Section 3.3, we provide formulae that support the previous discussion.

2.3 Power versus Energy

The power consumption of a hash design is measured during a complete hash operation. The total power consumption can be seen as the sum of the static and the dynamic power dissipation. The energy cost is therefore the integral of the power consumption over the period of a hash operation. In order to obtain a standardized nJ/bit metric, the energy cost is normalized to the input block size and to the message length for long and short messages, respectively.

2.4 Fair Comparison

An important requirement for an open competition such as the SHA-3 competition is a fair comparison. To achieve this goal, we need to consider the following two aspects. First, the evaluation environment needs to be open and available to all designers and evaluators. It also needs to be unified and common for all the candidates. Second, the claimed results need to be reproducible and open for public verification. By using a common, fixed platform and making our code publicly available, we achieve the desired goal.

2.5 Classification of Candidates

Another interesting issue to consider is the great diversity of all the second-round candidates. Therefore, we first classify all the algorithms with respect to their design properties. Figure 1 represents such a classification.

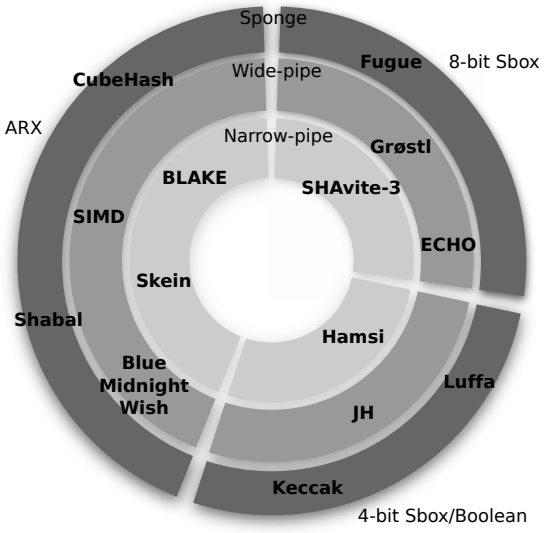


Fig. 1. Round 2 SHA-3 Candidates Classified with Respect to Their Design Properties (courtesy of Dai Watanabe from Hitachi Ltd, the designer of Luffa hash function).

With respect to the main source of non-linearity used in a design, all fourteen candidates can be classified into three main groups, as indicated by the three parts of the pie.

- 8-bit Sbox based: ECHO, Fugue, Grøstl, SHAvite-3.
- 4-bit Sbox/Boolean based: Hamsi, JH, Keccak, Luffa.
- Addition Rotation XOR (ARX) based: Blake, BMW, CubeHash, Shabal, SIMD, Skein.

Another classification by comparing the size of the compression function to the digest size and the input block size is possible, as indicated by the concentric circuits on the pie. If the output length of the intermediate compression function is equal to the digest size, the structure is called a *narrow-pipe*. The candidates with the output length of the compression function larger than the final hash length are classified as *wide-pipe*. Finally, the candidates whose compression function size and digest size are fixed, and whose input block size is determined by considering a trade-off between security and efficiency are called the *sponge* constructions. Therefore, depending on the size of the compression function, the candidates can again be classified into three subgroups.

- Narrow-pipe: Blake, Hamsi, SHAvite-3, Skein.
- Wide-pipe: BMW, ECHO, Grøstl, JH, SIMD.
- Sponge: CubeHash, Fugue, Keccak, Luffa, Shabal.

Finally, we classify the candidates with respect to their input block size.

- 32-bit: Fugue, Hamsi.
- 256-bit: CubeHash, Luffa.
- 512-bit: Blake, BMW, Grøstl, JH, Shabal, SHAvite-3, SIMD, Skein.
- 1024-bit: Keccak.
- 1536-bit: ECHO.

Another classification, with respect to the number of cycles necessary for performing the hash operation, is also possible but would highly depend on the implementation strategy. Therefore we do not consider it at this time. However, this observation becomes interesting later, in Section 4, where the implementation results are discussed in detail. Next, we discuss our proposed evaluation scheme. We describe the evaluation environment, hardware/software interface, design strategy, evaluation metrics and finally, we provide the experimental results.

3 Hardware Evaluation Platform for SHA-3 Candidates

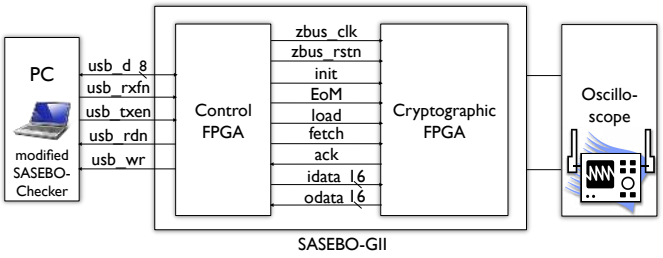


Fig. 2. Evaluation Environment Using SASEBO-GII.

Figure 2 illustrates the target platform for our evaluation, which includes a SASEBO-GII board, a PC and an oscilloscope. The SASEBO board includes two FPGAs: a control FPGA and a cryptographic FPGA. On the PC, a test

program enables a user to enter a sample message, which is transmitted to the control FPGA through a USB interface. The control FPGA controls the data flow to send this message to the cryptographic FPGA, where hash operations are performed. After the hash operation is done, the digest is returned to the PC through the control FPGA. As illustrated in Fig. 2, the interface between the control FPGA and the cryptographic FPGA is fixed and common among all SHA-3 candidates.

The control FPGA checks the latency of a single hash operation that is performed on the cryptographic FPGA and reports the number of clock cycles to the PC. The PC then reports two different performance metrics. One is the number of clock cycles including the interface overhead while the other one is excluding the cycles for the data input and output.

During message hashing, we also measure the power consumption of the hashing operation. This trace, in combination with the performance data, enables a precise characterization of the power dissipation and energy consumption of the SHA-3 candidate on the cryptographic FPGA.

3.1 Hardware and Software Interface

A key concept in our approach is the use of a standard interface to integrate the hash algorithms inside the cryptographic FPGA. In this section, we describe the major principles of this interface. We also compare our ideas with those of several other proposals, including the interfaces defined by Chen et al. [23], by Gaj et al. [24], and by Baldwin et al. [25].

In the following observations, it is useful to refer to the method used to interface SHA-3 candidates in software. For that purpose, the software implementations use an Application Program Interface (API) defined by NIST [26]. Three function calls are used:

- `void init(hashstate *d)` initializes the algorithm state of the hash, which is typically stored in a separate structure in order to make the hash implementation re-entrant.
- `void update(hashstate *d, message *s)` hashes a message of a given length and updates the hash state. The message is chopped into pieces of a standard length called a *block*. In case the message length is not an integral number of blocks, the API will use a *padding* procedure which extends the message until it reaches an integral number of blocks in length.
- `void finalize(hashstate *d, digest *t)` extracts the actual digest from the hash state.

A hardware interface for a SHA-3 module emulates a similar functionality as the software API interface. The hardware interface therefore needs to address the following issues.

Handshake protocol: The hash interface needs to synchronize data transfer between the SHA-3 module and the environment. This is done by using a handshake protocol and one can distinguish a *master* and a *slave* protocol, depending on which party takes the initiative to establish the synchronization. The interface by Chen [23] uses a slave protocol for the input and output of the algorithm. The interfaces by Baldwin [25] and Gaj [24] define a slave protocol for the input and a master protocol for the output. The former type of interface is suited for a co-processor in an embedded platform, while the latter one is suited for high-throughput applications that would integrate the SHA-3 module using First Input First Output (FIFO) buffers. The interface in our proposal uses a slave protocol.

Wordlength: Typical block and digest lengths are wider (e.g., 512 bits) than the word length that can be provided by the standard platforms (e.g., 32 bits). Therefore, each hash operation will result in several data transfers. While this overhead is typically ignored by hardware designers, it is inherently part of the integration effort of the SHA-3 module. In our proposal, we use a 16-bit interface, which size is driven by the size of the data-bus shared among the control FPGA and the cryptographic FPGA.

Control: The functions of the software API need to be translated to the equivalent control signals in hardware. One approach, followed by Gaj, is to integrate this control as in-band data in the data stream. A second approach is to define additional control signals on the interface, for example to indicate the message start and end. This is the approach taken by Chen and Baldwin. We follow the same approach in our proposal as well.

Padding: Finally, *padding* may or may not be included in the SHA-3 hardware module. In the latter case, the hardware module implicitly assumes that an integer number of blocks will be provided for each digest. Common padding schemes are defined by in-band data formatting, and this makes it possible to implement the padding outside of the hardware module. The interface proposal by Baldwin explicitly places the padding hardware into the interface. The other interface proposals leave the padding to be done outside of the hardware module. However, Chen assumes that the hardware padding will only be implemented at the word-level, while Gaj supports bit-level padding as well. We follow the approach of Chen.

Note that there are many solutions to the interface issue, and that we present only one approach. We also observe that the key issue for a fair comparison is to use a *common* interface for all the candidates. In addition, and that is very important, we show that our performance evaluation mechanism allows to factor out the overhead of the interface communication.

3.2 Design Strategy

Besides a standard platform, our approach also defines a design strategy. As classified by Schaumont et al. [27] there are three types of cores that can be distinguished with respect to their implementation scope (register mapped, memory mapped and network mapped). Similar to this approach, Tillich [28] proposes the following classification:

- *Fully Autonomous Implementation* (Fig. 3a): Equivalent to a register mapped implementation proposed by Schaumont et al. [27]. In this architecture, one transfers the message data to a hash function over multiple clock cycles, until a complete message block is provided. The hash module buffers a complete message block locally,

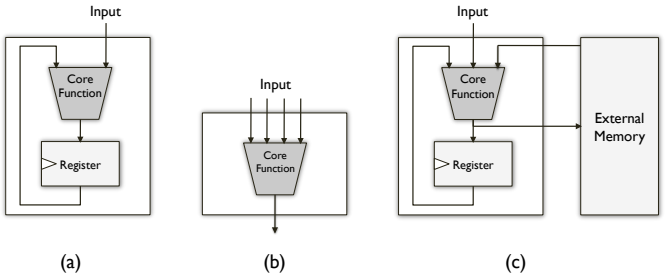


Fig. 3. Three Types of Architectures: (a) Fully Autonomous. (b) with External Memory. (c) Core Functionality.

before initializing the hash operation. Therefore, this architecture can work autonomously, and the resulting hash module is well suited for the integration into other architectures (e.g., System-on-Chip).

- *Implementation of the Core Functionality* (Fig. 3b): This architecture has only the core part of a hash function, and ignores the storage of a full message block. In other words, this architecture ignores the influence of a fixed interface on the total hardware performance.
- *Implementation with External Memory* (Fig. 3c): Equivalent to a memory mapped implementation proposed by Schaumont et al. [27]. In this architecture, only data necessary for executing the hashing calculation is stored in registers. Other data (e.g., intermediate values) is stored in the external memory. In general, the external memory is less expensive than the register based memory. Therefore, the architecture becomes a low-cost implementation. However, this architecture requires additional clock cycles for accessing the external memory, and therefore it is not suitable for high-throughput implementations.

In this work, we choose the Fully Autonomous architecture. Additionally, we estimate influence of the standard hardware interface on each of the fourteen candidates. Our choice of a 16-bit data width is driven by the specification of the common evaluation platform, i.e., SASEBO-GII board. In addition, we provide evaluation metrics that allow us to estimate the hardware performance for an arbitrary data width as well. One can easily obtain the figures by taking into account the highest achievable frequency and the input block size of each of the candidates. Furthermore, we provide the hardware figures by factoring out the overhead introduced by the standard interface.

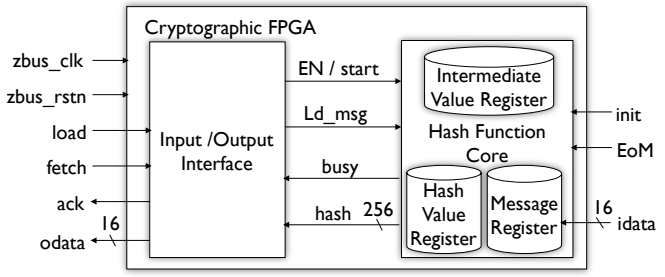


Fig. 4. Architecture of Cryptographic FPGA.

Figure 4 shows the detailed architecture of the cryptographic FPGA which we use for evaluating hardware performance. The cryptographic FPGA consists 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, we use a message register file for that purpose.

3.3 Platform Specific Evaluation Topics

We implement fourteen SHA-3 candidates on the cryptographic FPGA, Xilinx Virtex-5 (xc5vlx30-3ff324) placed on the SASEBO-GII evaluation board. We check the hardware performance in terms of speed and hardware cost. The speed performance is evaluated by calculating latency or throughput, depending on the message length. It is calculated using the input block size, the maximum clock frequency, and the total number of clock cycles with or without the communication overhead. The cost performance is evaluated with the number of slices, registers, and LUTs for FPGA and the number of gate equivalences for ASIC. A design that has a high throughput with a low hardware cost is regarded as efficient. The power consumption of a hash design is measured during a complete hash operation. The energy cost is therefore the integral of the power consumption over the period of a hash operation. In order to obtain a standardized nJ/bit metric, the energy cost is normalized with respect to the input block size and to the message length for long and short messages respectively.

In order to make the following discussion easier we introduce notations that are used further in the paper.

B :	Input block size,
w :	Word size (interface data width),
I :	Total number of clock cycles,
I_{in} :	Number of clock cycles for loading one message block,
I_{out} :	Number of clock cycles for outputting the message digest,
I_{core} :	Number of clock cycles for completing the hash process,
I_{final} :	Number of clock cycles for the finalization,
I_w :	Number of clock cycles for transmitting one word of data,
f_{max} :	Maximum clock frequency,
T :	Throughput,
L :	Latency,
M :	Size of the message without padding,
M_p :	Size of the message with padding,
H :	Size of the message digest (hash output).

A hash function executes a hashing process for each data block of input block size, and uses the result as a chaining value for the next input data block to perform the whole hashing process. The number of clock cycles needed for hashing M bits of data can be expressed as

$$I = \frac{M_p}{B}(I_{in} + I_{core}) + I_{final} + I_{out} \ . \quad (1)$$

Here, $\frac{M_p}{B}$ is the number of hash core invocations where the hash core processes a B-bit data block per single invocation. Note that the coefficients of I_{final} and I_{out} are both equal to one, since these processes are only executed when outputting the final message digest. The number of clock cycles needed for the input of the message block and

the output of the hash result can be evaluated as

$$I_{in} = \frac{B}{w} I_w \ ,$$
$$I_{out} = \frac{H}{w} I_w \ . \quad (2)$$

In our specific protocol, we use $w = 16$ bits and $I_w = 3$ cycles. The former is driven by the evaluation platform specification, while the latter is a result of a simple acknowledgement-based protocol. As a result, the final throughput can be expressed as

$$T = \frac{M_p f_{max}}{\frac{M_p}{B}\left(I_{in} + I_{core}\right) + I_{final} + I_{out}} \ , \quad (3)$$

It is also useful to estimate the throughput of the core function only, by factoring out the interface part. Therefore, we write

$$T_{Core} = \frac{M_p f_{max}}{\frac{M_p}{B} I_{core} + I_{final}} \ . \quad (4)$$

When M_p is sufficiently large, for example in the case of hashing a long message, I_{final} and I_{out} are negligible in Eq. 3 and Eq. 4. In this case, the throughput is approximated as

$$T_{LongMessage} = \frac{B f_{max}}{I_{in} + I_{core}} \ ,$$
$$T_{LongMessageCore} = \frac{B f_{max}}{I_{core}} \ . \quad (5)$$

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 metric for a short message (rather than the throughput), we use Eq. 6 to compare the speed performance of the SHA-3 candidates.

$$L = \frac{M_p}{T} \ ,$$
$$L_{Core} = \frac{M_p}{T_{Core}} \ . \quad (6)$$

Finally, we calculate power and normalized energy per bit consumption for both short and long messages. By P_U and P_F we denote the power consumption during the update and the final phase, respectively, and by f we denote the operating frequency.

$$P_{ShortMessage} = \frac{\frac{M_p}{B} I_{core} P_U + I_{final} P_F}{\frac{M_p}{B} I_{core} + I_{final}} \ ,$$
$$E_{ShortMessage} = \frac{\frac{M_p}{B} I_{core} P_U + I_{final} P_F}{Mf} \ ,$$
$$P_{LongMessage} = P_U \ ,$$
$$E_{LongMessage} = \frac{P_U I_{core}}{Bf} \ . \quad (7)$$

4 FPGA Evaluation Results

In this work, we implement SHA-256 and all fourteen SHA-3 candidates aiming at high-throughput hardware implementations[1]. Although it is not possible to completely factor out the designer's influence in our comparison, all fifteen algorithms were prototyped and tested using the same evaluation platform. Each of them was evaluated according to the metrics indicated above, comparing speed performance, area, power consumption and energy consumption.

Table 2 shows a comprehensive summary of the measurement results. **Bold** and gray data represent the best and the worst result in its class, respectively. As with all measurement data, it is important to understand the assumptions used when collecting these numbers. The table includes the following quantities for each candidate.

- The input message block size in bits;
- The highest clock frequency achievable on the Virtex-5 FPGA (xc5vlx30-3ff324) in MHz.
- The latency in terms of clock cycles. Several cases are shown: the cycle count of the input interface overhead (I_{in}); the cycle count of the output interface overhead (I_{out}); the cycle count of the core function (I_{core}); and the cycle count of the final processing (I_{final}). All mentioned measures are defined in Section 3.3.
- The throughput of the design in Mbps. This value is calculated assuming that the FPGA is operating at the maximum achievable clock frequency for the given design. Both the throughput with (T) and without (T_{Core}) interface overhead is shown.
- The latency of the design for short messages in μs. This value is calculated assuming that the FPGA is operating at the maximum achievable clock frequency for the given design. Both the latency with (L) and without (L_{Core}) interface overhead is shown. We choose the size of a short message to be 256 bits prior to padding.
- The area cost of the design, in terms of occupied Virtex-5 slices, number of slice registers, and number of slice LUTs. The number of occupied slices provides the primary area measure in this case, while the numbers of slice registers and slice LUTs illustrate the actual utilization of the occupied slices.
- The power consumption of the design for long and short messages. For long messages, the average power consumption includes only the core functionality. For short messages, the average power consumption includes the core functionality and the finalization. The power consumption is measured directly on the core power supply of the FPGA. The power consumption is measured with the FPGA operating at 24 MHz which is the default operating frequency of the board.
- The energy consumption of the design for long and short messages. The energy consumption is normalized with the input block size and the message length for long and short messages, respectively (expressed in nJ/bit). Also in this case, the difference between long-message energy and short-message energy relates

1. We release the Verilog/VHDL source code for these 15 algorithms at http://www.rcis.aist.go.jp/special/SASEBO/SHA3-en.html.

to the inclusion of the finalization processing in the measurement.

As can be seen from the amount of reported data in Table 2, there are many different dimensions where the comparison is possible. Since our main goal is a high-throughput implementation of all the candidates, we provide Fig. 5 where the candidates are compared with respect to the highest achievable throughput. We also offer the throughput estimates assuming different interfaces. The throughput is first estimated for the core function. Next, we provide the throughput figures assuming the ideal interface, meaning that we use only I_w clock cycles for the input and another I_w clock cycles for the output. Finally, we measure the throughput assuming a realistic interface width (from 16 bits to 128 bits).

Here, we draw an interesting, somewhat natural conclusion. The influence of the interface width is more noticeable for the candidates that have a small number of rounds and a larger size of the input block. Therefore, one may notice that the influence of the fixed interface is especially noticeable for BMW, Grøstl, Keccak, and Luffa.

In order to have a complete picture regarding the hardware cost that one needs to pay for implementing a high-throughput version of each candidate, we provide Fig. 6. The left-hand side of the figure represents a throughput versus area graph, ignoring the influence of the fixed interface, while the right-hand part shows the same graph by taking the interface into account. The candidates within the dashed ellipse are the ones with the largest Throughput/Area ratio.

Due to the very small number of rounds of the core function, the hash candidate BMW provides the highest core throughput among all candidates. The hardware price, however, due to the heavy unrolled architecture, is large (BMW also consumes most of the hardware resources). Other candidates that have noticeably high core throughput are Keccak, Grøstl and Luffa. Furthermore, Luffa and Keccak achieve a high core throughput with a relatively small hardware cost.

Assuming a fixed interface with parameters $w = 16$ bits and $I_w = 3$, which indeed complies with our evaluation platform, Luffa achieves the highest throughput. Luffa also has the highest hardware efficiency since it achieves the highest throughput with a relatively small hardware cost. Other candidates that have noticeably high throughput in this case are Keccak and SHAvite-3.

To have a complete picture regarding the latency of all candidates with respect to different sizes of the unpadded message, we provide Fig. 7. The left-hand side represents the core latency of all candidates versus message size, while the right-hand side represents the latency by taking the 16-bit interface into account. It is interesting to observe that for short messages, with less than 512 bits, CubeHash, Shabal, and Fugue show rather high core latency. This is due to the fact that these candidates have a large number of rounds in the final stage of the hashing process. The stair-steps on the graph appear due to the fact that an additional message block for padding is needed whenever we hash an unpadded message with size equal to the input block size of the algorithm. Since the input block size of Fugue and

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

SHA-3 Candidate	Input Block Size [bits]	Max. Clock Freq [MHz]	Total Number of Clock Cycles [cycles]				Long Message Throughput [Mbps]		Short Message Latency [μs] M=256 bits		Number of Occupied Slices	Number of Slice Registers	Number of Slice LUTs	Power [W]		Energy [nJ/bit]	
			I_{in}	I_{out}	I_{core}	I_{final}	T	T_{Core}	L	L_{Core}				Long Msg	Short Msg	Long Msg	Short Msg
SHA-256	512	260	96	48	68	0	812	1,958	0.815	0.262	609	1,224	2,045	0.21	0.21	0.65	1.30
BLAKE-32	512	115	99	48	22	0	487	2,676	1.443	0.191	1,660	1,393	5,154	0.27	0.27	0.49	0.98
BMW-256	512	34	96	48	2	2	178	8,704	4.353	0.118	4,350	1,317	15,012	0.41	0.41	**0.07**	**0.27**
CubeHash16/32-256	256	185	48	48	16	160	740	2,960	1.816	1.038	**590**	1,316	**2,182**	**0.23**	**0.23**	0.61	7.27
ECHO-256	1,536	149	315	48	99	0	553	2,312	3.101	0.664	2,827	4,198	9,885	0.28	0.28	0.75	4.49
Fugue-256	32	78	6	48	2	37	312	1,248	2.013	0.705	4,013	1,043	13,255	0.36	0.37	0.95	3.28
Grøstl-256	512	154	96	48	10	10	744	7,885	1.065	0.130	2,616	1,570	10,088	0.31	0.31	0.25	1.00
Hamsi-256	32	210	6	48	4	5	672	1,680	0.681	0.195	718	**841**	2,499	**0.23**	**0.23**	1.19	1.52
JH-256	512	201	96	48	39	0	762	2,639	0.910	0.194	2,661	1,612	8,392	0.25	0.25	0.80	1.60
Keccak(-256)	1,024	205	192	48	24	0	972	**8,747**	1.288	0.117	1,433	2,666	4,806	0.29	0.29	0.29	1.16
Luffa-256	256	**261**	48	48	9	9	**1,172**	7,424	**0.655**	**0.103**	1,048	1,446	3,754	0.24	0.24	0.36	1.07
Shabal-256	512	228	96	48	50	150	800	2,335	1.509	0.877	1,251	2,061	4,219	**0.23**	**0.23**	0.94	7.62
SHAvite-3_{256}	512	251	108	48	38	0	880	3,382	0.773	0.151	1,063	1,363	3,564	0.24	0.24	0.73	1.45
SIMD-256	512	75	96	48	46	0	270	835	2.533	0.613	3,987	6,693	13,908	0.29	0.29	1.09	2.17
Skein-512-256	512	91	102	48	19	19	385	2,452	2.066	0.418	1,370	1,956	4,979	0.30	0.30	0.47	1.86

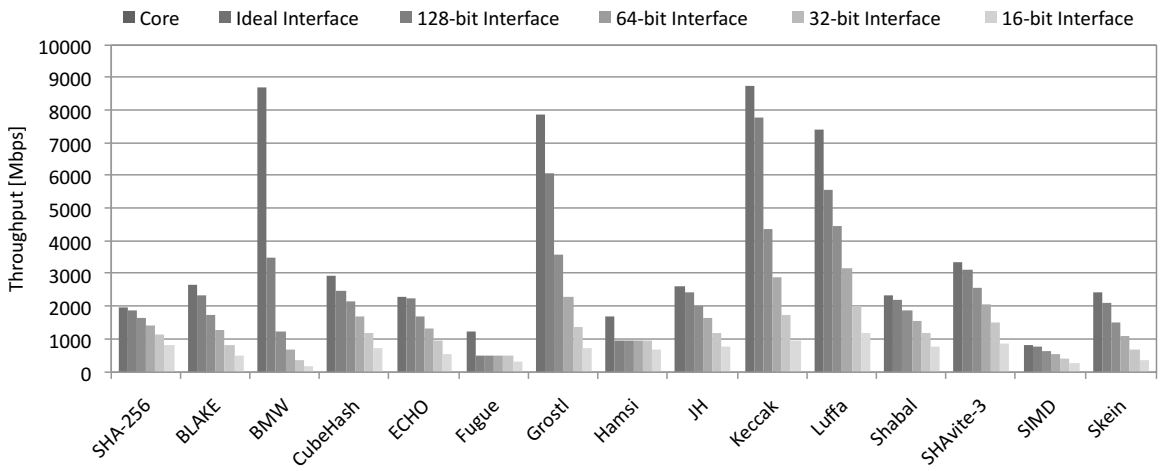


Fig. 5. Maximum Throughput for Various Types of Interface with $I_w = 3$. Target Platform: Virtex-5 (xc5vlx30-3ff324) FPGA Board.

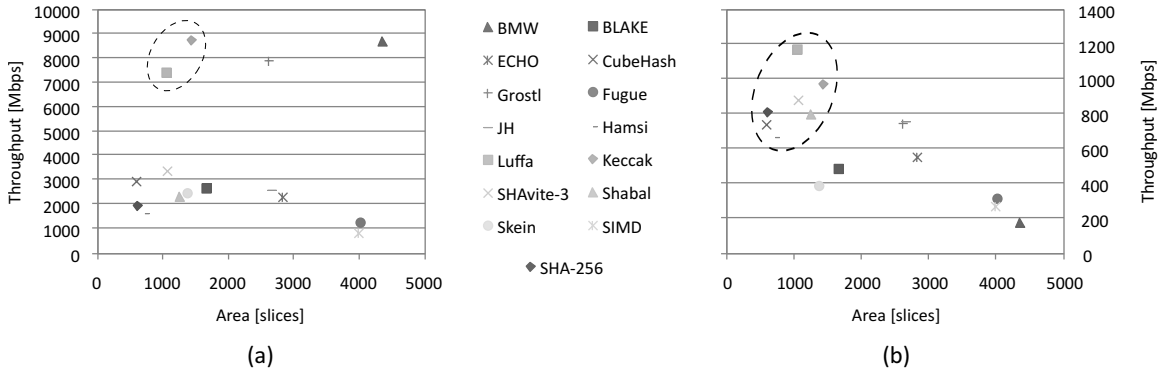


Fig. 6. Throughput versus Area graph: (a) Core Function only. (b) Fixed Interface with $w = 16$ bits and $I_w = 3$. Target Platform: Virtex-5 (xc5vlx30-3ff324) FPGA Board.

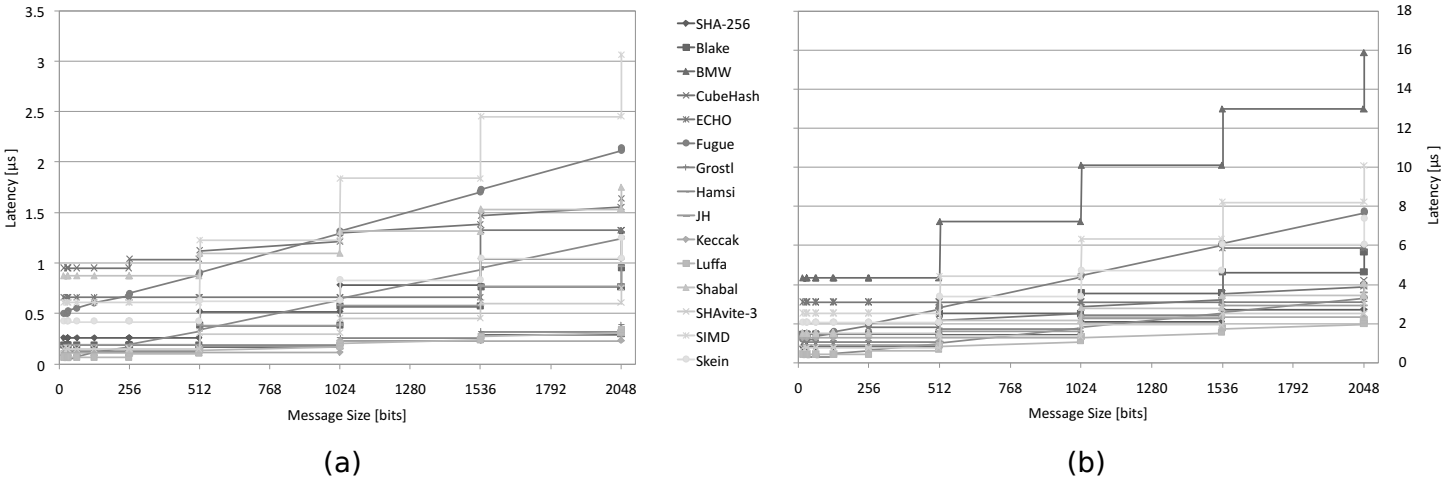


Fig. 7. Latency versus Message Size graph: (a) Core Function only. (b) Fixed Interface with $w = 16$ bits and $I_w = 3$. Target Platform: Virtex-5 (xc5vlx30-3ff324) FPGA Board.

Hamsi is only 32 bits and in order to have a clear graphical representation, we approximate their latency performance with the linear segments.

In order to explore the influence of a fixed interface on the minimum latency, we additionally provide Fig. 8. Here, we assume the length of the short unpadded message to be 256 bits. It can be noticed that Luffa has the shortest core latency among all candidates. Even when including the interface overhead, Luffa shows the best performance. The candidates with a larger number of cycles needed for the finalizing stage, such as CubeHash, Fugue, and Shabal, have noticeably high core latency. The biggest influence of a fixed standard interface is again demonstrated by BMW.

Finally, in Fig. 9 we show a latency versus area graph. Regarding the core latency versus area, we can select the set of candidates which show somewhat better performance compared to others, and those are: Luffa, Keccak, SHAvite-3, Hamsi, Blake, and Skein. With respect to the total latency (including the interface overhead) versus area, the set containing Hamsi, Luffa, and SHAvite-3 shows the best performance. These candidates show the smallest Latency-Area product.

4.1 Power and Energy Consumption

As mentioned in Section 2.3, we distinguish between a platform-dependent power (static power) and an algorithm-dependent power consumption (dynamic power). We measured the static power dissipation of the Virtex-5 FPGA on SASEBO-GII to be around 200 mW. Hence, the power numbers listed in Table 2 are dominated by the static power. To have an accurate comparison, we simply compare the candidates with respect to their algorithmic properties by measuring the dynamic power only, as depicted in Fig. 10a (the dynamic power is simply obtained by subtracting the static power from the total power consumption).

Due to the similar behavior during the update and the final phase, the difference between the power consumption for long and short messages is negligible. On the other hand, the dynamic energy consumption (see Fig. 10b) differs for long and short messages and is especially noticeable for candidates which require additional cycles for the finalizing stage (CubeHash, Fugue, Grøstl, Shabal, and Skein). ECHO and Keccak also have the same discrepancy, and this is due to the large input block while hashing a short message of only 256 bits. Since BMW is the largest design among all candidates, its power consumption is thereby the largest as well. However, due to the very small number of cycles needed for a hashing operation, BMW on the other hand consumes the least amount of energy.

4.2 Algorithmic Features versus Implementation Results

Recalling the classification from Fig. 1 we conclude that no obvious connection can be made between the hardware performance and the design properties of the fourteen candidates. As an illustration we provide the fact that the top 5 designs with respect to the core throughput are Keccak (4-bit Sbox/Boolean, Sponge, 1024-bit), BMW (ARX, wide-pipe, 512-bit), Grøstl (8-bit Sbox, wide-pipe, 512-bit), Luffa (4-bit Sbox/Boolean, Sponge, 256-bit) and SHAvite-3 (8-bit Sbox, narrow-pipe, 512-bit). They, all together, basically cover the complete design space as defined in Section 2.5.

However, several interesting conclusions can still be made by observing some of the algorithmic features versus the implementation results. Therefore, we observe that the narrow-pipe designs (BLAKE, Hamsi, SHAvite-3, and Skein) offer relatively low core throughput. Grøstl, Keccak, and Luffa, on the other hand, provide high throughput regardless of the interface type (none of them is a narrow-pipe design). Designs with very small input block size of only 32 bits (Fugue and Hamsi) offer a relatively small core throughput. ECHO, which is the candidate with the largest input block size also offers a small throughput, but this is more because ECHO has the largest number of rounds for hashing a block of the message.

As a conclusion of this section we argue that the Sponge based candidates with the light non-linear part (4-bit Sbox/Boolean based) and large "input block size/number of rounds" ratio (Keccak and Luffa) show somewhat better overall performance in comparison to the other candidates. Due to the simplicity of the design, they have the shortest critical path, which in combination with the large "input block size/number of rounds" ratio results in high throughput and low latency.

5 ASIC Evaluation Results

In order to have a complete picture regarding the possible hardware platforms, we synthesized the code of SHA-256 and all fourteen SHA-3 candidates using the STM 90 nm CMOS technology. We used Synopsys Design Compiler version A-2007.12-SP3. The tool automatically estimated power consumptions by using its own signal switching model for the datapaths, and thus we did not control test vectors for the power estimation.

We synthesized several circuits from one design by changing speed constraints (maximum frequency), and chose the three circuits, which showed the smallest size, the highest throughput, and the highest efficiency (throughput/gate). The result are presented in Table 3.

Our results are based on synthesis and we only provide the core throughput and the core latency as measures of speed. However, as we further plan to tape out the candidates which will be chosen in the third, and final round of the competition, and to use a very similar evaluation platform (SASEBO-R), we provide estimates of the interface influence on the ASIC performance as well.

Similar to the previous section, we provide the following figures:

- Fig. 11 – Maximum throughput of all fourteen candidates assuming various types of interface.
- Fig. 12 – Throughput versus area graph.
- Fig. 13 – Latency versus message size graph.
- Fig. 14 – Minimum latency of all fourteen candidates assuming various types of interface.
- Fig. 15 – Latency versus area graph.
- Fig. 16 – Power and energy consumption.

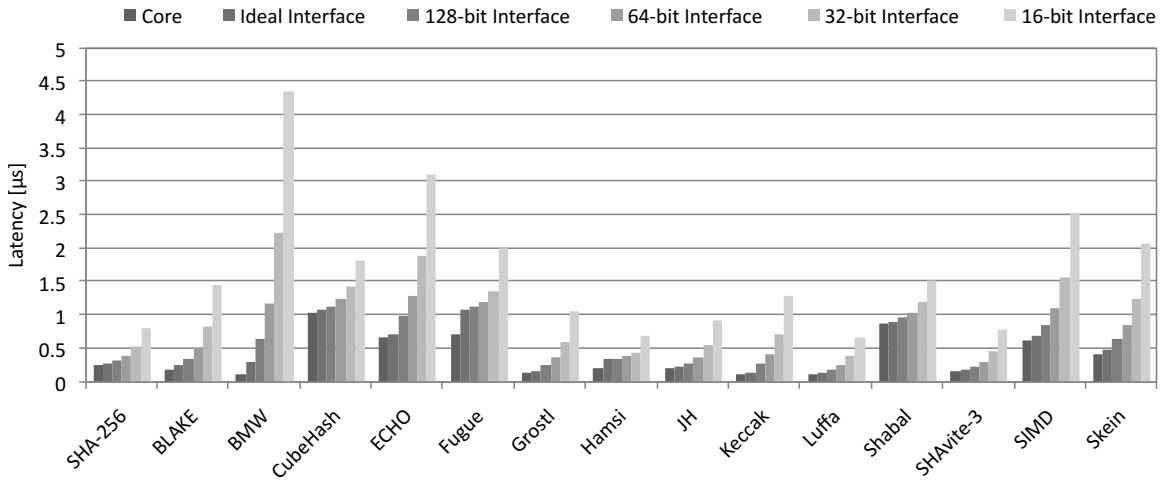


Fig. 8. Minimum Latency for Various Types of Interface with $I_w = 3$. Target Platform: Virtex-5 (xc5vlx30-3ff324) FPGA Board.

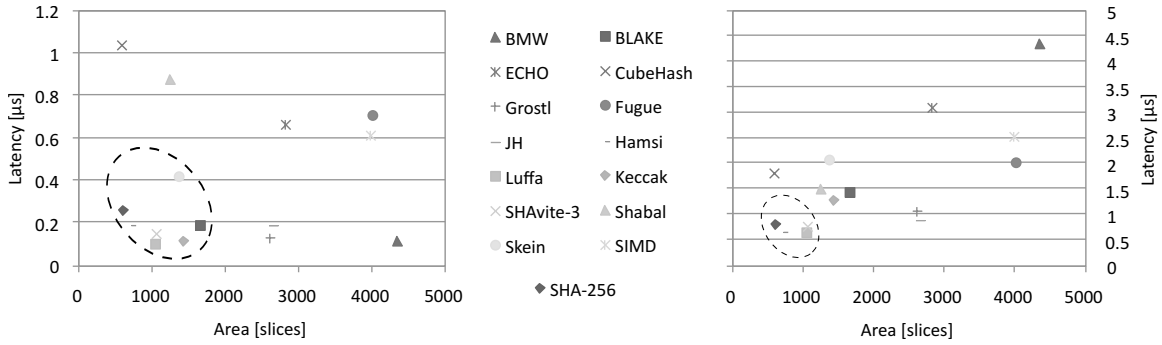


Fig. 9. Latency versus Area graph: (a) Core Function only. (b) Fixed Interface with $w = 16$ bits and $I_w = 3$. Target Platform: Virtex-5 (xc5vlx30-3ff324) FPGA Board.

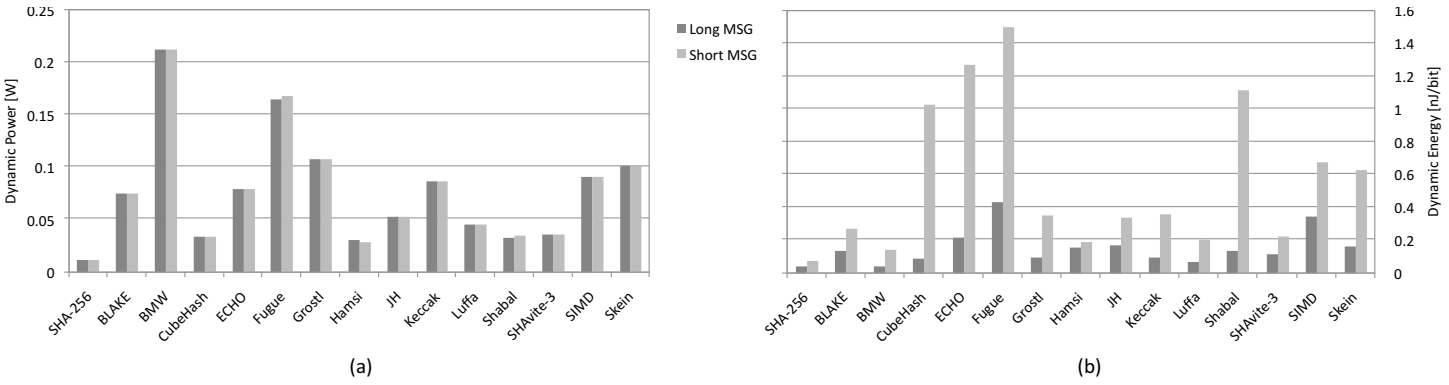


Fig. 10. (a) Dynamic Power Consumption. (b) Dynamic Energy Consumption. Target Platform: Virtex-5 (xc5vlx30-3ff324) FPGA Board.

TABLE 3
Synthesis Results of the SHA-3 Candidates using 90 nm CMOS Technology.

SHA-3 Candidate	Max. Freq.† [MHz]	Max. Core Throughput† [Mbps]	Min. Core Latency† [μs]	Total Area [GE]	Dynamic Power‡ [mW]	Dynamic Energy [pJ/bit]		Hardware Efficiency [kbps/GE]
						Long Msg	Short Msg	
SHA-256	735	5,536	0.09	18,677	3.11	2.31	4.62	290.6
	356	2,680	0.19	13,199	2.09	1.55	3.09	203.0
	117	878	0.58	11,332	1.77	1.32	2.63	77.4
BLAKE-32	286	6,668	0.08	36,944	10.84	4.66	9.31	180.5
	260	6,061	0.08	30,292	4.94	2.12	4.25	200.1
	147	3,412	0.15	23,214	3.77	1.62	3.24	147.0
BMW-256	101	25,937	0.04	128,655	9.25	0.36	1.44	201.6
	84	21,603	0.05	115,001	8.46	0.33	1.32	187.9
	67	17,262	0.06	105,566	7.47	**0.29**	**1.16**	163.5
CubeHash16/32-256	515	8,247	0.37	35,548	7.07	4.42	53.00	232.0
	352	5,834	0.55	21,336	4.07	2.54	30.53	264.1
	172	2,749	1.12	**16,320**	3.60	2.25	26.98	168.5
ECHO-256	362	5,621	0.27	101,068	17.24	11.11	11.11	55.6
	260	4,040	0.38	97,803	8.88	5.73	34.36	59.6
	147	2,278	0.67	57,834	8.32	5.36	32.16	39.4
Fugue-256	170	2,721	0.32	56,734	3.57	2.23	7.66	48.0
	113	1,808	0.49	45,553	3.01	1.88	6.46	37.9
	78	1,245	0.71	46,683	2.92	1.82	6.27	26.7
Grøstl-256	338	17,297	0.06	139,113	22.52	4.40	17.59	124.3
	258	13,196	0.08	86,191	12.74	2.49	9.95	153.1
	128	6,547	0.16	56,665	7.85	1.53	6.13	115.5
Hamsi-256	971	7,767	0.04	67,582	6.94	8.67	11.11	114.9
	544	4,348	0.08	36,981	3.44	4.31	5.51	117.6
	352	2,817	0.12	32,116	2.80	3.50	4.48	87.7
JH-256	763	10,022	0.05	54,594	2.94	2.24	4.48	183.6
	694	9,117	0.06	42,775	**2.07**	1.57	3.14	213.1
	353	4,639	0.11	31,864	2.13	1.63	3.25	145.6
Keccak(-256)	781	**33,333**	**0.03**	50,675	6.36	1.55	6.21	657.8
	541	23,063	0.04	33,664	3.62	0.88	3.54	685.1
	355	15,130	0.07	29,548	3.52	0.86	3.44	512.0
Luffa-256	**1010**	28,732	**0.03**	39,642	5.14	1.81	5.42	724.8
	538	15,293	0.05	19,797	2.85	1.00	3.01	**772.5**
	263	7,466	0.10	19,359	2.91	1.02	3.07	385.6
Shabal-256	592	6,059	0.34	34,642	5.80	5.66	45.30	174.9
	544	5,565	0.37	30,328	3.13	3.05	24.42	183.5
	351	3,593	0.57	27,752	3.16	3.08	24.65	129.5
SHAvite-3_{256}	625	8,421	0.06	59,390	3.61	2.68	5.36	141.8
	493	6,637	0.08	42,036	2.46	1.83	3.66	157.9
	207	2,784	0.18	33,875	2.41	1.79	3.57	82.2
SIMD-256	285	3,171	0.16	138,980	13.56	12.18	24.37	22.8
	261	2,906	0.18	122,118	10.77	9.67	19.35	23.8
	113	1,259	0.41	88,947	10.74	9.64	19.29	14.2
Skein-512-256	251	6,734	0.15	43,132	17.17	6.37	25.48	76.4
	206	5,551	0.18	28,782	4.42	4.68	18.73	87.7
	50	1,347	0.76	22,562	3.25	3.25	13.01	79.0

†Only the first subrow in each row is relevant for comparison of Max. Frequency, Max. Core Throughput, and Min. Core Latency.
‡The power consumption is estimated for the frequency of 100 MHz.

Since the designs were implemented to achieve the highest throughput, only the first subrow in each row is relevant for comparison of maximum frequency, maximum core throughput, and minimum core latency. Therefore, we mark (in **bold** and gray) fastest and slowest designs by observing the first subrows only. For other columns, we mark the extreme results by observing every subrow in each row.

5.1 Correlation between ASIC and FPGA results

By observing the provided graphs we argue that there is a good level of correlation between the ASIC and the FPGA results, with a few considerable differences. For example, when observing Fig. 6a and Fig. 12a, we notice that Fugue and Grøstl and differ considerably, while Blake and Hamsi differ noticeably. Further comparing Fig. 6b versus Fig. 12b, Fig. 9a versus Fig. 9a, and Fig. 9b versus Fig. 15b we notice that Fugue, Grøstl, and JH differ considerably. Another considerable difference is in the power/energy consumption for BMW, ECHO, and Grøstl. These three candidates are the largest in area among all, and since the power is estimated and measured using different platforms (ASIC and FPGA), this difference is acceptable. Therefore, we conclude that the obtained FPGA results represent rather reliable way of estimating the ASIC performance, especially with respect to speed and area.

6 Conclusion

For a complete hardware evaluation, there are plenty of evaluation platforms to be considered. Therefore, fixing one is crucial for conducting a fair and a consistent comparison. In this paper, we propose an evaluation platform and a consistent evaluation method to conduct a fair

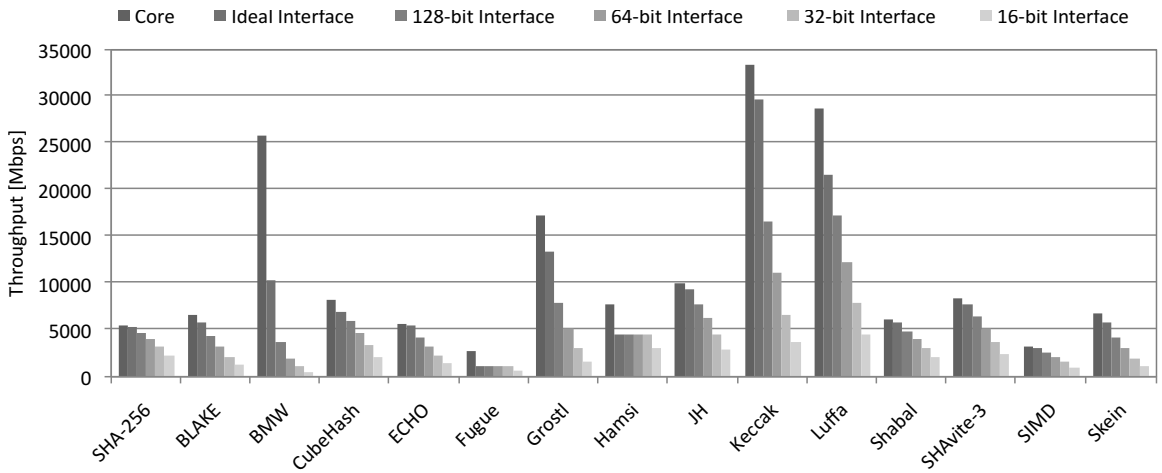


Fig. 11. Maximum Throughput for Various Types of Interface with $I_w = 3$. Target Platform: STM 90 nm CMOS Technology, Synthesis Results.

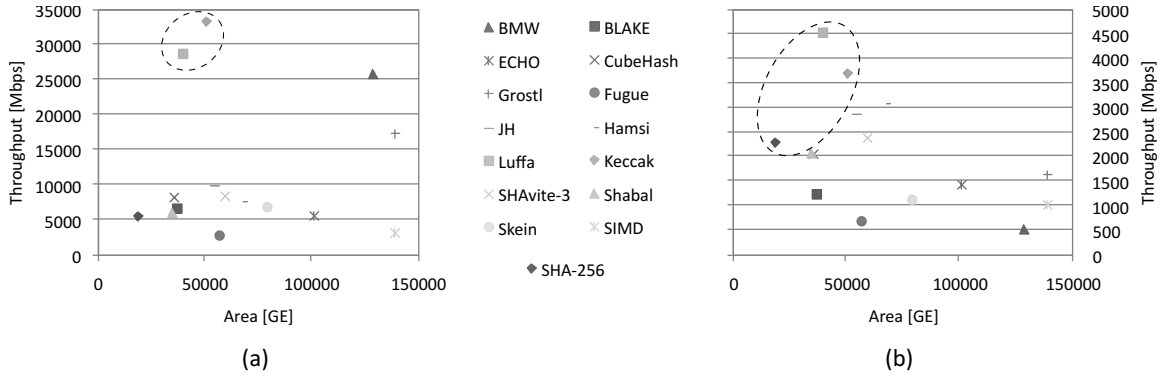


Fig. 12. Throughput versus Area graph: (a) Core Function only. (b) Fixed Interface with $w = 16$ bits and $I_w = 3$. Target Platform: STM 90 nm CMOS Technology, Synthesis Results.

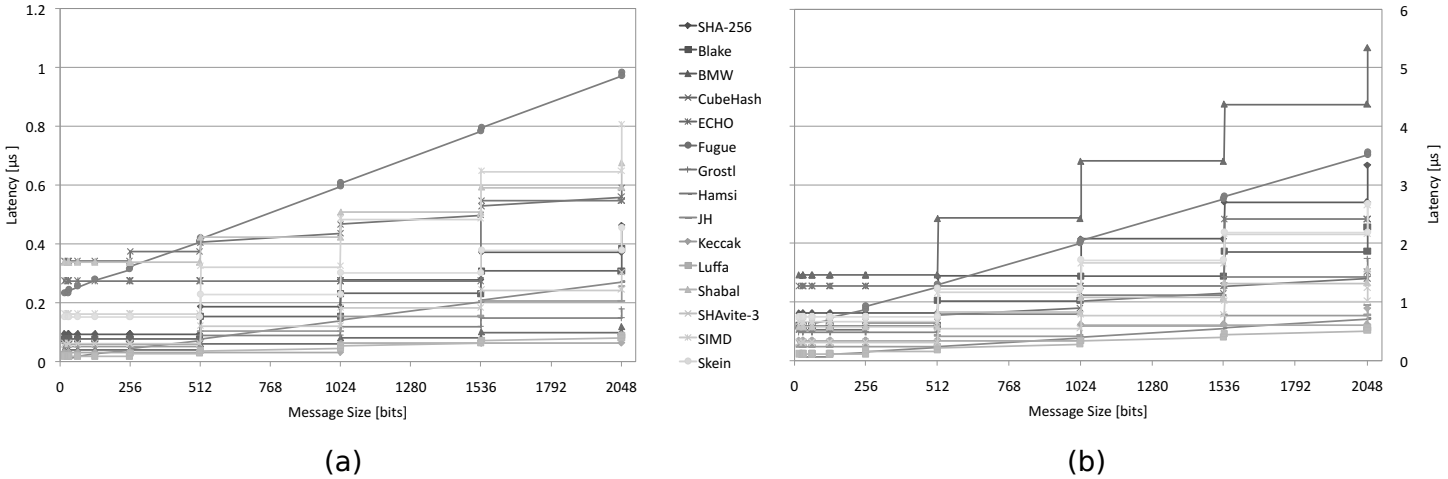


Fig. 13. Latency versus Message Size graph: (a) Core Function only. (b) Fixed Interface with $w = 16$ bits and $I_w = 3$. Target Platform: STM 90 nm CMOS Technology, Synthesis Results.

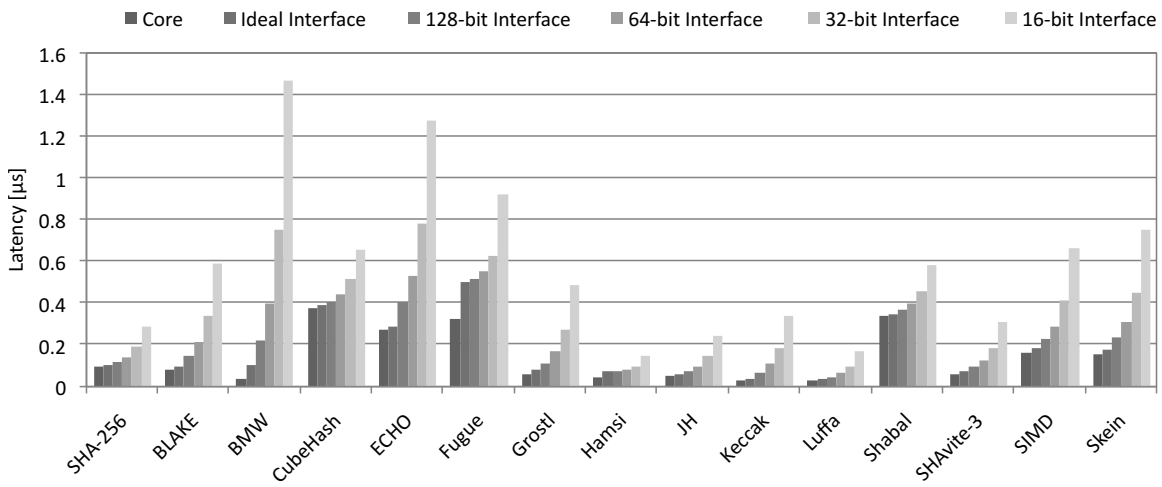
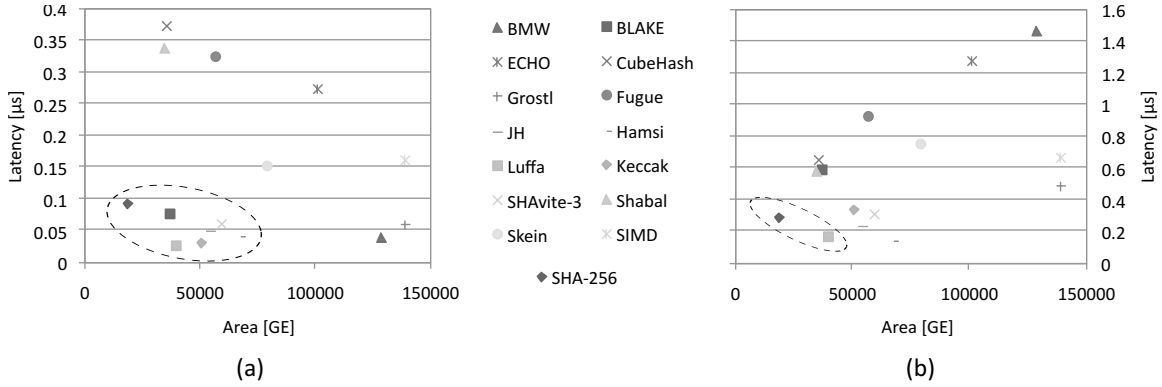
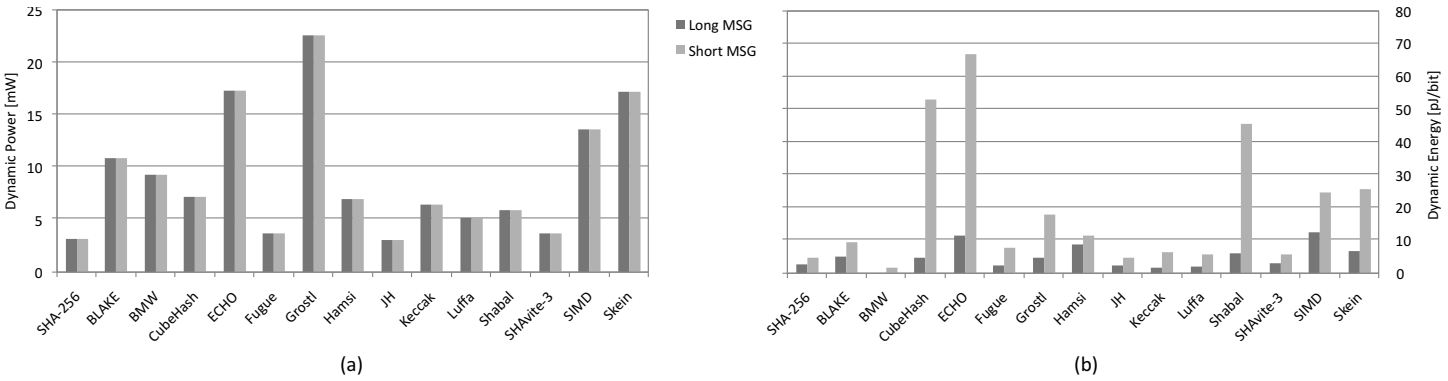


Fig. 14. Minimum Latency of all 14 Candidates assuming Various Types of Interface with $I_w = 3$. Target Platform: STM 90 nm CMOS Technology, Synthesis Results.



(a) (b)

Fig. 15. Latency versus Area graph: (a) Core Function only. (b) Fixed Interface with $w = 16$ bits and $I_w = 3$. Target Platform: STM 90 nm CMOS Technology, Synthesis Results.



(a) (b)

Fig. 16. (a) Dynamic Power Consumption. (b) Dynamic Energy Consumption. Target Platform: STM 90 nm CMOS Technology, Synthesis Results.

hardware evaluation of the remaining SHA-3 candidates. This proposal meets the requirements analyzed from actual hash applications and conditions of standard selection. The platform includes a SASEBO-GII evaluation board, evaluation software, and appropriate interface definition. Using this method, we implement all the second-round SHA-3 candidates and obtain the resulting cost and performance factors. This technical study provides a fair and a consistent evaluation scheme. At the end, we hope that by sharing our experience we contribute to the SHA-3 competition and by providing the proposed methodology we influence other similar future selections of the standard cryptographic algorithms.

ACKNOWLEDGMENT

The authors would like to thank Dai Watanabe from Hitachi Ltd, the designer of Luffa hash function, for providing valuable inputs regarding the classification of the SHA-3 candidates.

This work is supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State, by FWO project G.0300.07, by the European Commission under contract number ICT-2007-216676 ECRYPT NoE phase II, by K.U.Leuven-BOF (OT/06/40), and by the Research Council K.U.Leuven: GOA TENSE. This research was supported by Strategic International Cooperative Program (Joint Research Type), Japan Science and Technology Agency.

REFERENCES

[1] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *Advances in Cryptology — EUROCRYPT 2005*, vol. 3494 of *Lecture Notes in Computer Science*, Springer, 2005.

[2] X. Wang, Y. L. Yin, and H. Yu, "Finding Collisions in the Full SHA-1," in *Advances in Cryptology — CRYPTO 2005*, vol. 3621 of *Lecture Notes in Computer Science*, Springer, 2005.

[3] W. E. Burr, "Cryptographic Hash Standards: Where Do We Go from Here?," *IEEE Security and Privacy*, vol. 4, no. 2, pp. 88–91, 2006.

[4] National Institute of Standards and Technology (NIST), "Cryptographic Hash Algorithm Competition."

[5] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J. Schmidt, and A. Szekely, "Uniform Evaluation of Hardware Implementations of the Round-Two SHA-3 Candidates." The Second SHA-3 Candidate Conference, 2010.

[6] K. Gaj, E. Homsirikamol, and M. Rogawski, "Fair and Comprehensive Methodology for Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates using FPGAs," in *Cryptographic Hardware and Embedded Systems — CHES 2010* [29], pp. 264–278.

[7] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. O'Neill, and W. P. Marnane, "FPGA Implementations of the Round Two SHA-3 Candidates," in *Proceedings of 20th International Conference on Field Programmable Logic and Applications — FPL 2010*, 2010.

[8] L. Henzen, P. Gendotti, P. Guillet, E. Pargaetzi, M. Zoller, and F. K. Gürkaynak, "Developing a Hardware Evaluation Method for SHA-3 Candidates," in *Cryptographic Hardware and Embedded Systems — CHES 2010* [29], pp. 248–263.

[9] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont, "Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations." The Second SHA-3 Candidate Conference, 2010.

[10] National Institute of Advanced Industrial Science and Technology (AIST), Research Center for Information Security (RCIS), "Side-channel Attack Standard Evaluation Board (SASEBO)."

[11] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J. 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. http://eprint.iacr.org/.

[12] K. Gaj, E. Homsirikamol, and M. Rogawski, "Comprehensive Comparison of Hardware Performance of Fourteen Round 2 SHA-3 Candidates with 512-bit Outputs Using Field Programmable Gate Arrays." The Second SHA-3 Candidate Conference, 2010.

[13] B. Baldwin, A. Byrne, L. Lu, M. Hamilton, N. Hanley, M. O'Neill, and W. P. Marnane, "A Hardware Wrapper for the SHA-3 Hash Algorithms," in *Signals and Systems Conference — ISSC 2010, IET Irish*, pp. 1–6, 2010.

[14] K. Ideguchi, T. Owada, and H. Yoshida, "A Study on RAM Requirements of Various SHA-3 Candidates on Low-cost 8-bit CPUs." Cryptology ePrint Archive, Report 2009/260, 2009. http://eprint.iacr.org/.

[15] L. Henzen, J.-P. Aumasson, W. Meier, and R. C.-W. Phan., "VLSI Characterization of the Cryptographic Hash Function BLAKE," 2010. Available at http://131002.net/data/papers/HAMP10.pdf.

[16] M. Bernet, L. Henzen, H. Kaeslin, N. Felber, and W. Fichtner, "Hardware implementations of the SHA-3 candidates Shabal and CubeHash," *Midwest Symposium on Circuits and Systems*, vol. 0, pp. 515–518, 2009.

[17] L. Lu, M. O'Neil, and E. Swartzlander, "Hardware Evaluation of SHA-3 Hash Function Candidate ECHO." Presentation at the Clauce Shannon Institute Workshop on Coding and Cryptography 2009, 2009.

[18] S. Halevi, W. E. Hall, and C. S. Jutla, "The Hash Function Fugue." Submission Document, 2008.

[19] S. Tillich, M. Feldhofer, W. Issovits, T. Kern, H. Kureck, M. Mühlberghuber, G. Neubauer, A. Reiter, A. Köfler, and M. Mayrhofer, "Compact Hardware Implementations of the SHA-3 Candidates ARIRANG, BLAKE, Grøstl, and Skein." Cryptology ePrint Archive, Report 2009/349, 2009. http://eprint.iacr.org/.

[20] S. Mikami, N. Mizushima, S. Nakamura, and D. Watanabe, "A Compact Hardware Implementation of SHA-3 Candidate Luffa," 2010. Available at http://www.sdl.hitachi.co.jp/crypto/luffa/ACompactHardwareImplementationOfSHA-3CandidateLuffa_20101105.pdf.

[21] A. Bogdanov, L. R. Knudsen, G. Le, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," in *Cryptographic Hardware and Embedded Systems — CHES 2007*, vol. 4727 of *Lecture Notes in Computer Science*, pp. 450–466, Springer, 2007.

[22] C. D. Cannière, O. Dunkelman, and M. Knežević, "KATAN and KTANTAN – A Family of Small and Efficient Hardware-Oriented Block Ciphers," in *Cryptographic Hardware and Embedded Systems — CHES 2009*, vol. 5747 of *Lecture Notes in Computer Science*, pp. 272–288, Springer, 2009.

[23] P. S. Z. Chen, S. Morozov, "A Hardware Interface for Hashing Algorithms." Cryptology ePrint Archive, Report 2008/529, 2008. http://eprint.iacr.org/.

[24] CERG at George Mason University, "Hardware Interface of a Secure Hash Algorithm (SHA). Functional Specification," October 2009. http://cryptography.gmu.edu/athena/.

[25] B. Baldwin, A. Byrne, L. Lu, M. Hamilton, N. Hanley, M. O'Neill, and W. P. Marnane, "A Hardware Wrapper for the SHA-3 Hash Algorithms." Cryptology ePrint Archive, Report 2010/124, 2010. http://eprint.iacr.org/.

[26] National Institute of Standards and Technology (NIST), "ANSI C Cryptographic API Profile for SHA-3 Candidate Algorithm Submissions," 2008.

[27] P. Schaumont, K. Sakiyama, A. Hodjat, and I. Verbauwhede, "Embedded Software Integration for Coarse-Grain Reconfigurable Systems," *Parallel and Distributed Processing Symposium, International*, vol. 4, p. 137, 2004.

[28] The SHA-3 Zoo, "SHA-3 Hardware Implementations." http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations.

[29] *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, vol. 6225 of *Lecture Notes in Computer Science*, Springer, 2010.