© 2011 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. Digital Object Identifier : 10.1109/TENCON.2011.6129082

Two Hardware Designs of BLAKE-256 Based on Final Round Tweak

Muh Syafiq Irsyadi*† and Shuichi Ichikawa*‡
*Dept. Knowledge-based Information Engineering
Toyohashi University of Technology, Hibarigaoka, Tempaku, Toyohashi 441-8580 Japan
†Institut Teknologi Bandung, Email: irsyadi@students.itb.ac.id
‡Numazu National College of Technology, Email: ichikawa@ieee.org

***Abstract*—This paper presents two implementations of BLAKE hash family algorithm that has been selected as one of the SHA-3 competition finalist. The first implementation is a modification from the implementation of Beuchat et al. which significantly reduces the required ROM size up to 36% from the original requirement with small trade-off in additional logic circuit. The second implementation is an extension from Half-G structure that was designed to be flexible for different kinds of application. The highly compact BLAKE-256 design uses 356 LE and 9776 bits of memory when implemented in Cyclone III FPGA. Regular data-path design requires 369 slices and 1 memory block in Virtex 5 FPGA. Both designs are fully autonomous, which means that these designs do not require any additional memory or logic outside its system.**

***Index Terms*—SHA-3 competition, BLAKE, hardware implementation, FPGA**

I. Introduction

Mobile communication and computing had created new requirements for a high-speed, reliable but secure method of delivering data. Network of high performance computing system also requires ways to secure their information. In another field, smart card and RFID had replaced common method of payment that needs a system that is secure but small and cheap for exchanging financial information. Cryptographic hardware is utilized ranging from high speed high performance system to low cost small system.

BLAKE has been selected as one of the finalists of SHA-3 Competition [1]. Hash function is one form of cryptographic algorithms that are frequently utilized in digital signature and authentication. Hash algorithm should be able to produce fixed length output regardless of message input length. Commonly used hash functions, MD5 and SHA-1 have been proven as breakable. SHA-2 which has been designed to be more robust than them is still considered secure. However, SHA-2 shared the same structure with SHA-1, which raised some concern about its security. Therefore NIST announced a competition for a new hash function that will augment SHA-2 [2]. The competition is held in three rounds. Aside from the security criteria, these new algorithm should be implementable in a wide range of hardware and software platforms. NIST will evaluate hardware cost which includes computational efficiency or algorithm speed and memory requirement which includes gate/logic counts and memory blocks.

BLAKE is a combination of three proven cryptographic method, they are LAKE, HAIFA, and ChaCha. BLAKE iteration mode is derived from LAKE, while its internal structure is derived from HAIFA structure, and its compression algorithm is a modified version of ChaCha algorithm. LAKE hash function uses local wide-pipe internal structure which make local collision impossible. Hash Iterative FrAmework (HAIFA) is an update from Merkle-Damgard construction. HAIFA allows one pass computation with a fixed amount of memory regardless of the size of the input message. ChaCha compression function proposed by Bernstein [3] is a Salsa20 stream cipher family in which its security property has been deeply analyzed.

Hash cryptographic hardware is utilized in wide range of application starting from low-power cost-constrained smart card to high-speed network switch. Each of these applications requires different kinds of hardware optimization. This study aimed to design BLAKE hash cryptographic hardware that can be easily modified to achieve certain throughput value based on its application requirements with a slight modification in its code design. A modification of a BLAKE hardware design, that was first proposed by Beuchat et al. [4] which significantly reduces the microcode ROM size, is also presented especially to correspond with final round tweak. In the next section, short introduction to BLAKE hash function algorithm is examined. Section III shows two hardware designs of BLAKE, and in the following section we examine their performances in FPGA. Comparison with other hardware designs of BLAKE is also presented.

II. BLAKE Hash Family Algorithm

BLAKE hash algorithm is a hash functions family comprises of four members, BLAKE-224, BLAKE-256, BLAKE-384, and BLAKE-512. The first two are intended for 32 bit application while the latter two are for 64 bit application. Each version has similar algorithm but differ in its initial value, message padding, and constant values. After the announcement of the SHA-3 competition finalist, Aumasson et al. [5] proposed design tweaks for BLAKE. The first tweak is only in the naming to avoid confusion and made it easier to recognize. The other tweaks increase round iterations in the compression function. For 32-bit versions of BLAKE (BLAKE-224, BLAKE-256) round iteration is increased from 10 to 14 rounds. For 64-bit versions (BLAKE-384, BLAKE-512) round iteration is increased from 14 to 16 rounds. These

modifications were made to increase its security margin, which was possible because BLAKE has been recognized as a fast algorithm.

All BLAKE family shares the same three steps algorithm. It started with initialization process where an inner state of 4×4 matrix is computed from initial value (or previous chain value), salt, and counter. Round process iterates this inner state and transforms it using compression functions $G_i(a,b,c,d)$. Finalization step generates digest message that also used for the next chain value.

A. BLAKE-256

A brief explanation about BLAKE hash algorithm especially for BLAKE-256 variant will be shown in the next subsection and modification for the other variants will follow.

1) Initialization: A 4×4 state matrix consist of 16 words v_0-v_{15} is initialized with initial chain value (h_0-h_7), salt (s_0-s_3), constant (c_0-c_7), and counter (t_0,t_1) such that different input will generate different state matrix.

$$\begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix}$$

2) Round: After the state matrix is initialized, the compression function takes place that consist of series of 14 iteration rounds. Each round execute eight transformation defined by G_i functions in equation 1.

$$\begin{aligned} & G_i(a,b,c,d), i \in \{0,...,7\}, r \in \{0,...,14\} \\ & a \longleftarrow a + b + \left(m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}\right) \\ & d \longleftarrow (d \oplus a) \ggg 16 \\ & c \longleftarrow c + d \\ & b \longleftarrow (b \oplus c) \ggg 12 \\ & a \longleftarrow a + b + \left(m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}\right) \\ & d \longleftarrow (d \oplus a) \ggg 8 \\ & c \longleftarrow c + d \\ & b \longleftarrow (b \oplus c) \ggg 7 \end{aligned} \tag{1}$$

As shown in equation 1, each G_i functions transformation besides operating v_0 through v_{15}, it also takes message and constant value as an input where σ_r denotes a permutation value defined by predetermined permutation table.

Four G_i functions operate on column wise and the four operate on diagonal wise. Each function G_0, G_1, G_2, G_3 operates on different column and effectively updates different values of v. In the same manner, each diagonal function G_4, G_5, G_6, G_7 operates on different diagonal column to calculate and update different values of v.

3) Finalization: Finalization step computes a new chain value $h^{'} = \left\{h_0^{'}, ..., h_7^{'}\right\}$ is computed by XOR-ing previous chain value with salt and two states v as shown in equation 2.

$$\begin{aligned} h'_0 &\longleftarrow h_0 \oplus s_0 \oplus v_0 \oplus v_8 \\ h'_1 &\longleftarrow h_1 \oplus s_1 \oplus v_1 \oplus v_9 \\ h'_2 &\longleftarrow h_2 \oplus s_2 \oplus v_2 \oplus v_{10} \\ h'_3 &\longleftarrow h_3 \oplus s_3 \oplus v_3 \oplus v_{11} \\ h'_4 &\longleftarrow h_4 \oplus s_0 \oplus v_4 \oplus v_{12} \\ h'_5 &\longleftarrow h_5 \oplus s_1 \oplus v_5 \oplus v_{13} \\ h'_6 &\longleftarrow h_6 \oplus s_2 \oplus v_6 \oplus v_{14} \\ h'_7 &\longleftarrow h_7 \oplus s_3 \oplus v_7 \oplus v_{15} \end{aligned} \tag{2}$$

B. Other BLAKE family members

Other variant of 32-bit version of BLAKE, BLAKE-224 share the same algorithm and constant value but it differs in its initial chain value. The 64-bit versions of BLAKE (BLAKE-384 and BLAKE-512) use different initial value and constant because of their word length difference. The compression functions of BLAKE-512 and BLAKE-384 are similar with the 32-bit variant except that they use different values for constant rotation factor. They also perform 16 rounds of iteration instead of 14.

III. Hardware Design

Currently proposed hardware designs of BLAKE did not incorporate final round tweak. The core functionality itself did not change but there are some possible changes in the control sequences that will affect logic utilization and memory bits usage. In this paper, two different designs are presented which conform to the final specification of BLAKE. The **highly compact design** attemps to reduce logic utilization to its minimum by time-sharing most of the arithmetic and logic computation. The second design is **regular data-path design**, which implements Data Flow Graph (DFG) directly. This design can be easily modified to meet certain application requirement such as required operating frequency or data throughput.

A. Highly compact design

1) Reproduction of proposed design by Beuchat et al.: This design extends the ideas proposed in [4]. Compact design is achieved by using minimal arithmetic and logic unit (ALU) circuitry in a time-shared manner. In [4], compact design of BLAKE-32 is proposed by dividing BLAKE-32 round computation into 10 simple equations which contains only one or two operations.

The design consists of control unit which is a simple counter, microcode ROM to store instruction and memory address, dual port RAM to store all data, and Arithmetic Logic Unit (ALU) as shown in figure 1. The ALU is pipelined into four stages and each stage only performs one specific basic arithmetic operation. All operations are governed by four fields in the microcode. The first field is write-enable signal for write back process to the memory. Second and third are addresses for port B and port A of the dual port memory respectively. The last field is control signal to control ALU behavior.

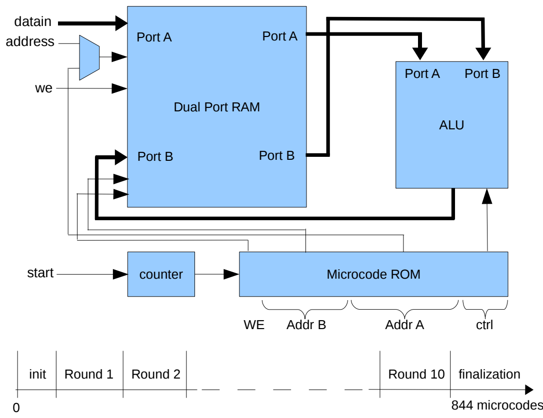

Fig. 1. Reproduction from BLAKE-32 design and its timeline

Each G-function is separated into 10 basic arithmetic operations, there are 8 G-functions in each round, and there are 10 rounds for BLAKE-32. Therefore, to finish all BLAKE-32 round function, it requires 800 clock cycles. Additional cycles are required for initialization (16 clock cycles), finalization (24 clock cycles), and data loading to the pipeline (4 clock cycles). Required clock cycles are 844 cycles in total.

This design excels in its low logic area requirement but it requires considerably large ROM to store all of its microcode. It requires 20×844 (16880) bits of ROM. These ROM would become 23280 bits for BLAKE based on latest design specification because BLAKE designer had decided to increase the number of round iteration.

2) New microcode design for fully autonomous BLAKE-256: In order to reduce these ROM requirements, an instruction compaction technique is utilized. Close examination in each round reveals that equations in each BLAKE round are similar. The difference lies in the permutation variables defined by permutation constants for determining message (m) and constant (c) address. These permutation variables in each G-function in each round are controlled by predefined permutation table presented in BLAKE specification. Moreover, these permutation variables are only called in 2 equations from 10, and they also appeared in regular time interval.

Based on these two observations, by separating microcode for calling permutation table into two different ROMs, microcode ROM can be executed iteratively. Each round will execute exactly the same code which can drastically reduce microcode ROM size as shown in figure 2. The permutation ROM size itself still depends on the number of iteration but it only holds the address for constant and message which are 4 bit each, so it can radically reduce the number of ROM bits required.

Microcode ROM is split into two ROMs. The first ROM (microcode ROM) holds the initialization, iterative control sequences, and finalization instructions, while the second ROM (permutation ROM) holds the non-iterative sequences instruction which governed by permutation value in current round. There are 16 calls of permutation values for message (m) and counter (c) in each round and they are divided into 4 group calls. Each permutation value is a simple address pointer for message and constant. Consequently it only needs 8 bits word length to store address for m and c in a single call (4 bits each).

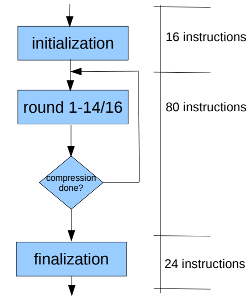

Fig. 2. Microcode iteration

A counter retains the microcode ROM address. Additional counter is required to generate both permutation ROM addresses and keep track of iteration rounds. A decoder is used to determine whether the instruction should come from microcode ROM or permutation round according to the counter value and current state. This simple modification reduce a great deal of ROM bits in exchange for a slight increase of logic area for counters and a decoder. The functionality of the design does not change but maximum operational frequency is expected to be slightly lower due to additional decoder logic.

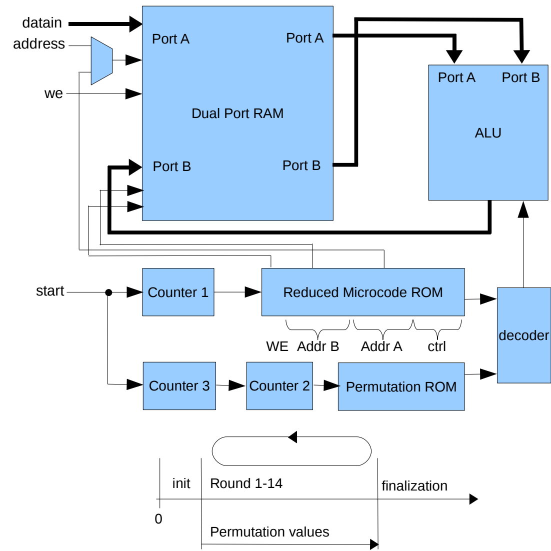

Fig. 3. Modified compact BLAKE-256 design and its timeline

This design shown in figure 3, requires $14 \times 8 \times 10$ (1120) clock cycles to finish the round iterations; it also requires additional 20 cycles for initialization and 22 cycles for finalization and write-back of digest messages to memory. Total required clock cycles are 1162 cycles, measured from the beginning of initialization stage until the digest message becomes ready to be read.

B. Regular data-path design

While highly compact design excels in minimal usage of logic circuit, it commonly occupies fairly large amount of memory bits and or shows rather low throughput because ALU is time-shared. Regular data-path design made some tradeoff between logic area and throughput while it can be easily be optimized for speed or throughput as required because of its regular structure.

Equation 3 shows that each G-function can be divided into eight arithmetic operations. A close examination also reveals that each G variables (a, b, c, d) is updated twice. Therefore, by introducing another variable g, the arithmetic operations can be collapsed into just four. Variable g defines whether the current operation happened in the first half of G-function, or in the second half. This approach resulted in similar design to the half-G design in [5].

$$
\begin{aligned}
&G_i\left(a, b, c, d\right), i \in \{0, ..., 7\}, r \in \{0, ..., 14\}, g \in \{0, 1\} \\
&\quad a \longleftarrow a + b + \left(m_{\sigma_r(2i+g)} \oplus c_{\sigma_r(2i+1-g)}\right) \\
&\quad d \longleftarrow \left(d \oplus a\right) \ggg 8 \ggg 8\left(1-g\right) \\
&\quad c \longleftarrow c + d \\
&\quad b \longleftarrow \left(b \oplus c\right) \ggg 7 \ggg 5(1-g)
\end{aligned}
\tag{3}
$$

Equation 3 can be arranged into Data Flow Graph (DFG) fashion. The DFG shows that the longest path would be from $m \rightarrow b^{'}$ or $c \rightarrow b^{'}$ or $a \rightarrow b^{'}$. We can safely assume that adder circuit generates longer delay compared to XOR circuitry. Therefore, the highest path delay in this circuit would be from $a \rightarrow b^{'}$ that goes through three adders, two XOR, and two constant shifters. We can also omit the constant shifter in the analysis because it can be implemented as re-wiring that do not use any logic circuit at all.

Based on this analysis, a direct implementation of a BLAKE ALU can be generated. The ALU has six inputs and four outputs. The circuit only compute one half of G-function, so it requires another cycle to compute complete G-function. A multiplexer is inserted into each input of $a, b, c,$ and d, to choose whether the input data should come from memory or from its previous computation.

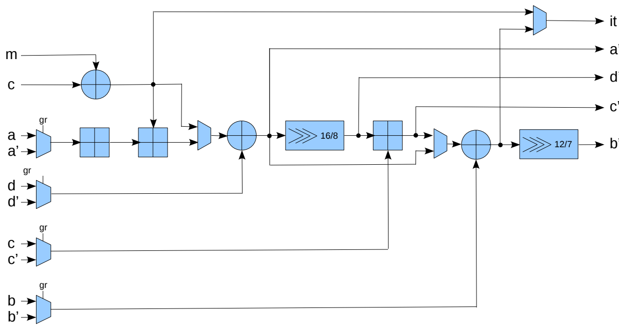

Fig. 4. BLAKE-256 ALU

The same circuitry can also be utilized to computes state initialization, and finalization step. Both steps are done before and after rounds computation respectively, so it is possible to time share the logic. For initialization step, a single XOR block is needed. The best possible candidates would be using XOR block that computes m and c. For the finalization step, all XOR circuits are utilized sumultaneously. Additional multiplexers in the input and in between XOR logic are required to switch the ALU functionality between each step. Complete design of ALU is shown in figure 4.

The ALU receives its inputs from 6 different single port RAMs. Four 32×4 RAMs are used to store v data, while the other two is used to store messages (32×16) and constants (32×16). It also requires two small RAMs to store initial chain value (h), counter (t) and salt (s) with size 32×8 each. Constant and memory address are controlled by PAROM value. State machine controller, control and record current step, round, and G-function state. Complete system of BLAKE-256 is shown in figure 5.

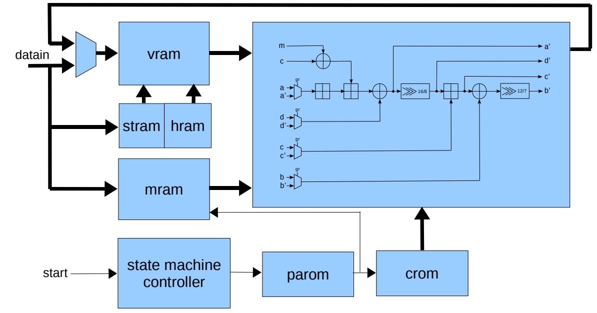

Fig. 5. BLAKE-256 design

IV. FPGA Implementation and Comparisons

FPGA implementation is attempted to evaluate hardware cost which becomes one of the deciding criteria for SHA-3 competition. Hardware cost includes computational speed, logic usage, and memory usage. Computational speed is translated into throughput that is calculated by dividing working frequency with total required clock cycles and multiplied with message length [6]. Logic usage in FPGA is measured differently among different FPGA vendors. In Altera FPGA, logic usage is measured in Logic Element (LE) while for Xilinx FPGA logic usage is measured in slices. Each previous design of BLAKE use different kinds of FPGA which makes direct comparison is difficult [7]–[10] as shown in [11]. Therefore, the BLAKE designs are synthesized in two kinds of FPGA platform for fair comparison with the most similar design.

The original design has been synthesized on numerous FPGA platforms [4]. Altera Cyclone III device is chosen for comparison because it has limited resources which are ideal for a compact hardware design. Aumasson et al. also synthesized their design on several FPGA targets [5]. Xilinx Virtex-5 is chosen as the second platform because it has the largest hardware resources in terms of logic and memory blocks which are ultimately necessary for higher freedom of design optimization.

A. Highly compact design

There are three 32-bit BLAKE designs that were synthesized with Cyclone III FPGA (EP3C5F256C6) as its target. The first BLAKE-32 design is a direct reproduction from the original design (BLAKE-32 A). This design was examined as a reference for comparison purpose. The second BLAKE-32 design is the design that was being optimized in microcode ROM area (BLAKE-32 B). The third design is BLAKE-256 which compatible to the latest BLAKE design specification.

TABLE I
COMPACT DESIGN SYNTHESIS RESULT

Design	Area (LE)	Mem. (bit)	Freq. (MHz)	Thr. (Mbps)
BLAKE-32 [4]	285	-	192	116
BLAKE-32 A	260	24576	214	130
BLAKE-32 B	356	8944	214	130
BLAKE-256	356	9776	214	94
BLAKE-64 [4]	542	-	140	123
BLAKE-64 A	492	32192	205	180
BLAKE-64 B	619	13872	211	186
BLAKE-512	620	14288	190	147

From the synthesis result in table I, it is evident that the required memory bits for the modified versions are reduced considerably to only 36% and 43% from the original requirement of BLAKE-32 and BLAKE-64 respectively, while additional logic elements are required as a trade-off. The BLAKE-256 design adds 4 additional rounds which are translated into additional clock cycles. This tweak, does not add additional logic, but it reduce the design throughput performance. It also requires small increase in memory requirement to store the constant and message address in round 11 through 14, while the other microcodes are shared with the previous rounds. The BLAKE-512 design also do not requires an increase in logic usage. A 1.5% increase in memory bit usage is required to store constant and message for the additional rounds.

B. Regular data-path design

Regular data-path design approach is generally much closer to BLAKE reference design than to highly compact design hardware. Thus, comparison with BLAKE reference design is much more proportional. The design is synthesized on Virtex 5 FPGA. Table II shows that the regular data-path design requires similar slice area compared to BLAKE-256[1G] design but it has higher optimal frequency. It also has relatively low throughput compared to reference design, because it only implements half of the compression functions. The purpose of this design is to create a minimal and regular hardware design that can be easily further optimized. This regular style of design have the potential for customization to increase its performance.

TABLE II
REGULAR DATA-PATH DESIGN SYNTHESIS RESULT

Design	Area (slice)	Mem. (block)	Freq. (MHz)	Thr. (Mbps)
BLAKE-256[8G] [5]	1694	-	67	3103
BLAKE-256[4G] [5]	1217	-	100	2438
BLAKE-256[1G] [5]	390	-	91	575
BLAKE-32 [12]	1660	0	115	487
BLAKE-256 Reg	369	1	145	266

V. CONCLUSION

In this paper, two designs of BLAKE hash function hardware have been presented. Each design is targeted to different application constraint. For a cost and area constrained application, we proposed a compact and small memory BLAKE-256 design modified from [4]. This design made a trade-off by sacrificing throughput to achieve small logic and memory footprint. For a custom high performance system, we also proposed regular data-path design. This design can be further optimized based on target application by using simple technique like pipelining and parallel processing to achieve higher throughput.

REFERENCES

[1] B. W. E. (2010, dec) The SHA-3 Finalists. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html

[2] NIST. (2007, nov) Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. NIST. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf

[3] D. J. Bernstein, "ChaCha, a variant of Salsa20," 2008. [Online]. Available: http://cr.yp.to/chacha/chacha-20080128.pdf

[4] J.-L. Beuchat, E. Okamoto, and T. Yamazaki, "Compact Implementations of BLAKE-32 and BLAKE-64 on FPGA," in *Proc. FPT 2010*, Eds. IEEE Press, 2010, pp. 170–177.

[5] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. (2010, dec) SHA-3 Proposal BLAKE version 1.3, December 16, 2010. [Online]. Available: http://www.131002.net/blake/blake.pdf

[6] Y. K. Lee, M. Knezevic, and I. M. Verbauwhede, "Hardware design for Hash functions," in *Secure Integrated Circuits and Systems*, ser. Series on Integrated Circuits and Systems,Eds. Springer US, 2010, pp. 79–104.[Online]. Available: http://dx.doi.org/10.1007/978-0-387-71829-3_5

[7] N. Sklavos and P. Kitsos, "BLAKE HASH Function Family on FPGA: From the Fastest to the Smallest," *Proc. ISVLSI 2010*, pp. 139–142, 2010.

[8] S. Tillich et al., "Compact Hardware Implementations of the SHA-3 Candidates ARIRANG, BLAKE, Grøstl, and Skein," Cryptology ePrint Archive, Report 2009/349, 2009.

[9] S. Tillich et al., "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.

[10] A. H. Namin and M. A. Hasan, "Hardware Implementation of the Compression Function for Selected SHA-3 Candidates," 2009. [Online]. Available: http://www.vlsi.uwaterloo.ca/~ahasan/web_papers/technical_reports/web_five_SHA_3.pdf

[11] ECRYPT. (2010, dec) The SHA-3 Zoo. [Online]. Available: http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo

[12] K. Kobayashi et al., "Evaluation of Hardware Performance for the SHA-3 Candidates Using SASEBO-GII," Cryptology ePrint Archive, Report 2010/010, 2010.