Lightweight Implementations of SHA-3 Candidates on FPGAs* **

Jens-Peter Kaps, Panasayya Yalla, Kishore Kumar Surapathi, Bilal Habib, Susheel Vadlamudi, Smriti Gurung, and John Pham

ECE Department, George Mason University, Fairfax, VA 22030, U.S.A.
{jkaps, pyalla, ksurapat, bhabib, svadlamu, sgurung, jpham4}@gmu.edu
http://cryptography.gmu.edu

Abstract. The NIST competition for developing the new cryptographic hash algorithm SHA-3 has entered its third round. One evaluation criterion is the ability of the candidate algorithm to be implemented on resource-constrained platforms. This includes FPGAs for embedded and hand-held devices. However, there has not been a comprehensive set of lightweight implementations for FPGAs reported to date. We hope to fill this gap with this paper in which we present lightweight implementations of all SHA-3 finalists and all round-2 candidates with the exception of SIMD. All implementations were designed to achieve maximum throughput while adhering to an area constraint of 400-600 slices and one Block RAM on Xilinx Spartan-3 devices. We also synthesized them for Virtex-V, Altera Cyclone-II, and the new Xilinx Spartan-6 devices.

Keywords: SHA-3, FPGA, lightweight implementation, benchmarking

1 Introduction and Motivation

The National Institute of Standards and Technology (NIST) started a public competition to develop a new cryptographic hash algorithm in November 2007. From the submitted 64 entries only 14 were selected for the second round of the competition and in December 2010, the 5 Secure Hash Algorithm-3 (SHA-3) finalists were announced. NIST is expected to announce the winner in 2012. In its decision which candidate algorithms should advance to the next round, NIST

* Jens-Peter Kaps, Panasayya Yalla, Kishore Kumar Surapathi, Bilal Habib, Susheel Vadlamudi, Smriti Gurung and John Pham. Lightweight Implementations of SHA-3 Candidates on FPGAs. In Daniel J. Bernstein and Sanjit Chatterjee editors, *Progress in Cryptology INDOCRYPT 2011*, Lecture Notes in Computer Science vol. 7107, pages 270–289. Springer Berlin Heidelberg, Dec, 2011. The original publication is available at http://www.springerlink.com. http://dx.doi.org/10.1007/978-3-642-25578-6_20

** This work has been supported in part by NIST through the Recovery Act Measurement Science and Engineering Research Grant Program, under contract no. 60NANB10D004.

used the following criteria [39][3]: security, cost, and algorithm and implementation characteristics. The cost criterion describes the computational efficiency (speed) and memory requirements (gate counts for hardware implementations). One important implementation characteristic is the ability of the hash function to be "[...] implemented securely and efficiently on a wide variety of platforms, including constrained environments, such as smart cards"[3]. During the second phase of the SHA-3 competition many hardware implementations of the candidates have been published. The first comprehensive analysis on FPGAs that included I/O overhead was done by Kobayashi et al. [31] on high throughput implementations of 8 round-2 candidates. The authors adapted an interface from [14] to the SASEBO [37]. Matsuo et al. [33] implemented all 14 round-2 candidates on FPGAs on SASEBO. Gaj et al.,[19] also implemented all 14 round-2 candidates on FPGAs optimized for throughput over area ratio with a different interface [15]. All implementations mentioned above only consider hash sizes of 256 bits. Homsirikamol et al. [26] and Baldwin et al [5] implemented all 14 round-2 candidates on FPGAs considering other hash sizes. Neither of these comprehensive implementations were done for resource-constrained applications. In a system-on-chip (SOC) on FPGAs, cryptographic functions such as encryption algorithms or hash algorithms are not necessarily the main purpose of the application but a part of it. Many other components such as soft-core processors, optimized signal processing algorithms, etc. are integrated in one chip. Furthermore, a space-constrained implementation could allow for using a smaller FPGA which in turn leads to cost and power savings. Recent developments in low-cost and low-power FPGAs [40] will increase their usage in battery powered devices which makes small implementations even more important.

Unfortunately, designing low-area implementations is not as straightforward as optimizing a design for best throughput over area. One has to go beyond merely reducing the datapath width and carefully evaluate the trade-off speed vs. area at every step of the design process. The control unit is an additional hurdle. Extensive component re-use in the datapath can lead to a very complex control logic which might negate the area savings in the datapath. There have been several publications that show low-area implementations of single SHA-3 candidates on FPGAs such as BLAKE [11], Grøstl [29], Keccak [9], and Skein [35][**?**]. Unfortunately, they are implemented on different FPGAs from different vendors and with different target sizes. This makes a fair comparison amongst these implementations impossible. Most recently Jungk [28] presented compact implementations of Grøstl, JH and Skein and Kerckhof et al. [30] of all five SHA-3 finalists at the "ECRYPT II Hash Workshop 2011". [30] shows results for 256-bit digest versions only on Spartan-6 devices. This device choice makes comparisons with previously reported results impossible. Furthermore, the authors did not formulate a clear design criterion other than "compact". Neither design is the smallest possible, yet the implementation area varies from 117 slices to 304 slices, the throughput from 105 Mbit/s to 960 Mbit/s, with the largest design (Grøstl) being the fastest. Only if one criterion is fixed (e.g. area or throughput) a meaningful comparison can be made.

Standardized interfaces have been proposed [14][15] for implementations of SHA-3 candidates and used by several comprehensive implementations in order to facilitate a fair comparison. Depending on the design of the hash function the interface can become a bottleneck. Furthermore, the interface protocol causes overhead and increases the size of the data path and control logic. Low area implementations will be particularly affected by the protocol overhead. Only the most recent publications, [28] and [30] use standardized interfaces.

In this paper we present low-area implementations of all five finalists and all round-2 candidates with the exception of SIMD[1], designed using the same criterion (space constraint), device, interface and optimization methods. This work is the most comprehensive analysis of lightweight implementations reported to date. In Sect. 2 we present the design methodology we used including clear assumptions and goals, interface description and performance metrics. Due to space constraints we describe only the datapaths of the five SHA-3 finalists in detail in Sect. 3. Our designs of the other algorithms are summarized in Table 1. Section 4 shows the results of our implementations and compares the 13 candidates with each other and other reported implementations.

2 Methodology

The primary target for our lightweight implementations are the low-cost Xilinx Spartan-3 FPGAs. We choose VHDL to describe our lightweight architectures. All implementations were designed at a low level for our main target FPGA family such that we can already obtain a rather precise estimate of the required area from detailed datapath diagrams. This approach allowed us to enforce a similar coding style across several designers and algorithms. Furthermore, we built a small VHDL library of elementary functions that was used by all designers.

2.1 Assumptions and Goals

Only SHA-3 variants with 256-bit digest have been implemented as these are the most likely variants to be used in area-constrained designs. Furthermore, we assume that padding is done in software. This assumption goes hand-in-hand with the application of hash functions to SOC designs. The salt values of all SHA-3 candidates who support them are set to zero. Typical optimization goals for hardware implementations are: maximum throughput, maximum throughput to area ratio, and minimum area. In order to compare lightweight implementations the minimum area target seems logical. However, optimizing the implementations for minimum area would yield a ranking of algorithms solely based on area, i.e. we would know which is the smallest and which is the largest irrespective of the throughput that is achieved by these implementations. This information is of not much use in practice. A different approach is to optimize for throughput

[1] Our initial investigation has shown that it is unlikely that SIMD could be implemented within our area constraints, due to its complex underlying functions.

given an area constraint. We believe that this is a much more realistic scenario. Additionally this optimization goal lets us determine how efficient an algorithm is in a constrained environment which is a factor of an algorithm's flexibility. This is a clearly stated evaluation criterion by NIST [3]. We choose to use an area range of 400 to 600 slices and 1 Block RAM on Xilinx Spartan-3 FPGAs as our constraint. The size of the range was chosen based on low-area implementation results published on the SHA-3 Zoo [2] website and our own analysis. Within this area constraint we try to achieve maximum throughput. Therefore, our final comparisons will be in terms of the ratio of throughput to area. The Block RAM was chosen due to the large storage requirements of some hash functions.

2.2 Tools and Result Generation

Even though all designs were targeted for Spartan-3 devices it is interesting to see how our implementations perform on low-cost devices from another vendor such as Altera Cyclone-II, newer devices such as Spartan-6 and on high speed devices such as Xilinx Virtex-V. Complete results are published in the ATHENa results database [1]. All designs were implemented using the vendor tools: Xilinx ISE 12.3 Web Pack and Altera Quartus II v. 10.0 Web Edition, and verified after place-and-route against known answer test files provided by the submissions packet of each hash function. All results were generated using the open source benchmarking tool ATHENa (Automated Tool for Hardware EvaluatioN) [20]. Other than simplifying the result generation, ATHENa also varies the vendor tool parameters to achieve optimal results.

2.3 Interface and Protocol

We based our hardware interface and I/O protocol (Fig. 1) on the one presented in [15] and updated in [19]. The SHA Core assumes that its inputs and outputs are connected to FIFOs. We believe that the FIFO interface model proposed in [15] is very suitable for lightweight implementations. In its simplest form a FIFO is a single w-bit wide register with minimal logic to support the handshake of read/write and ready. This can easily be interfaced to a microcontroller or other circuitry in an embedded system. Lightweight applications usually have smaller databus sizes than the 32 or 64 bits proposed in [15]. Therefore, we use a databus width w of 16 bits. The protocol supports two scenarios: 1) when the message length is known and 2) when the message length is not known. In case 1) the message is sent as a single segment starting with the message length after padding "msg_len_ap" in 32-bit words concatenated with a '1' followed by the message length before padding "msg_len_bp" in bits followed by the message. The "msg_len_bp" is needed by several algorithms even when the message is already padded. In case 2) the message can be processed in segments $seg_0, seg_1, \cdots, seg_{n-1}$. Each segment $seg_0, \cdots, seg_{n-2}$ is headed by the segment length after padding "seg_len_ap" concatenated with a '0' followed by the segment of the message. The last segment seg_{n-1} follows the format of case 1). It

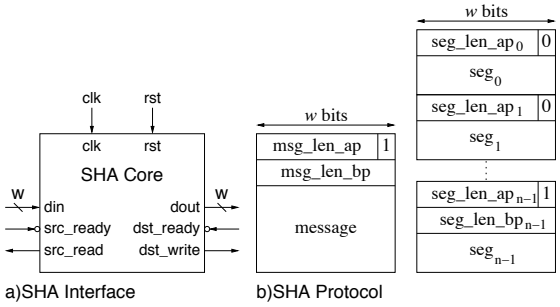


Fig. 1: Interface and protocol for our SHA cores

contain a block of the message and must contain all padding. The formulae to compute the total number of bits before padding and after padding are:

$$msg\_len\_ap = \sum_{i=0}^{n-1} seg\_len\_ap_i \cdot 32$$

$$msg\_len\_bp = \sum_{i=0}^{n-2} seg\_len\_ap_i \cdot 32 + seg\_len\_bp_{n-1}$$

Furthermore in order to conserve logic resources needed for message counters we limit the total amount of data in a single message to 2^{32} bits i.e. 4 Gbits which we believe is sufficient for lightweight applications.

2.4 Area Minimization Techniques

Datapath: The most straightforward approach to reducing the area of the datapath is folding. Vertical folding reduces the datapath width while horizontal folding reduces the size of processing elements while maintaining the datapath width. How many times and in which direction a design can be folded depends on the algorithm. The extent to which folding can be applied to the SHA-3 candidates and how much it affects their throughput and throughput over area ratio has been examined by Homsirikamol et al. [27]. They show that only BLAKE can reach our area constraints through folding alone, Grøstl remains too large, JH area increases when folded, and Keccak as well as Skein cannot be folded at all and hence far exceed our area constraints. Another technique is reusing of processing elements. We heavily use this technique and additionally, we apply vertical folding at multiple levels down to single processing elements, not just the datapath as a whole as done in [27]. For example the Skein algorithm uses 4 Mix functions each using a 64-bit adder and a 64-bit XOR. We fold the 4 Mix functions into 1 and within the Mix function we reuse a 32-bit adder to perform 64-bit additions. The same adder is also reused for the key injections. Both folding and reuse of processing elements minimize the area consumption at the cost of an increased number of clock cycles. We reduced this increase to some extent by interleaving operations through pipelining.

Block RAM: Block RAMs (BRAMs) offer a large amount of memory space for storage but have a limited number of ports and I/O lines. Xilinx Spartan-3 BRAMs can be configured as single or dual port memories with a maximum data width of 64 bits or 32 bits per port, respectively. Each port is associated with a single address input. This limits the number of independent values and the number of bits that can be accessed in a single clock cycle. Our Grøstl design processes four 8-bit values in each clock cycle. Even though these are only 32 bits, a dual port BRAM does not allow reading of four independent values in one clock cycle. Hence, we store that data in 4 Distributed RAMs. The Spartan-3 BRAM data sheet specifies that data is written to the address applied in the current clock cycle, but read from the address of the previous clock cycle. Hence, computing $Mem[i] = Mem[i] + k$, where each element is a 64-bit word, requires 2 clock cycles per address location i, i.e. dedicated write cycles. These are not needed when computing $Mem[i + i] = Mem[i] + k$, i.e. when an address shift is acceptable. In our early Keccak design, this address shift increased the complexity of the control logic and with it the area consumption beyond our constraint. Hence it now uses dedicated write cycles. The new Xilinx Spartan-6 and Virtex-6 devices allow for independent read and write addresses for 64-bit data width.

Control Logic: The control logic of our implementations consists of a main finite state machine (FSM) with up-to 8 states, a single counter to count the clock cycles per state, and ROM-based FSMs for each state of the main FSM. ROM-based FSMs are more efficient in terms of area consumption and speed compared to conventional FSM [36], [38], [21], and their maximum frequency is independent of the complexity. However they are more complex to design. The area required to implement ROM-based FSMs is determined by the number of control signals and states. In order to reduce the number of control signals we try to use bits from the counter output, the main finite state machine, and simple boolean logic combinations thereof wherever possible. Furthermore, short sequences of control signals are placed in sub-controllers. The complexity of address generation for BRAMs can be reduced by placing datasets in memory locations starting at addresses which are a power of 2.

2.5 Performance Metrics

The number of clock cycles needed to hash N message blocks using our implementations can be computed from the number of clock cycles required to perform the following functions:

i Initialization (if not precomputed)	p Processing one block
h Loading protocol header of message	z Finalization
$l1$ Loading first block	o Output of the hash value
l Loading each subsequent block	

This results in the following formula for the number of clock cycles clk for hashing N blocks of data.

$$clk = i + h + l1 + l \cdot (N - 1) + p \cdot N + z + o$$

This formula can now be simplified to reflect the number of clock cycles needed for the initial steps before processing can begin $st = i + h + (l1 - l)$, loading and processing one block of data $l + p$, and finalization and output of the hash value $end = z + o$ resulting in (1).

$$clk = st + (l + p) \cdot N + end \tag{1}$$

Throughput is defined as the number of input bits processed per unit of time. The precise formula for throughput of a hash function is dependent on the number of message blocks N to be hashed, the block size b of the algorithm, the number of clock cycles needed to hash the message clk and the clock period T. We can derive the formula to compute the throughput from (1).

$$throughput(N) = \frac{b \cdot N}{clk \cdot T} = \frac{b \cdot N}{(st + (l + p) \cdot N + end) \cdot T} \tag{2}$$

Especially in embedded applications, messages can be very short. It is therefore important to also calculate the throughput for short messages. We use the empty message which after padding is one block long and therefore set $N = 1$ in (2) to compute the throughput.

When computing the throughput for very long messages, we can neglect st and end as their influence on the result goes to zero. This leads to the simplified equation (3).

$$throughput_{long} = \frac{b}{(l + p) \cdot T} \tag{3}$$

Resource Utilization of FPGAs is very difficult to define. All FPGAs contain configurable logic elements which contain flip-flops (Xilinx: slices, Altera: LE), BRAMs, multipliers and other resources. These resources have different features not only depending on the vendor but even on the FPGA family. Hence, we can compare implementations using the metric of throughput over area ratio only within a specific FPGA family and provided they use the same number of dedicated resources. As area in this formula we use solely *slices* for Xilinx and *LEs* for Altera devices as there is no direct mapping from BRAM utilization to slice or LE.

3 Implementations

Due to space constraints we only briefly describe our implementations of the SHA-3 finalists. A short list of implementation details of all 13 SHA-3 candidates evaluated in this paper is shown in Table 1. The throughput formulae for all implementations is shown in Table 2.

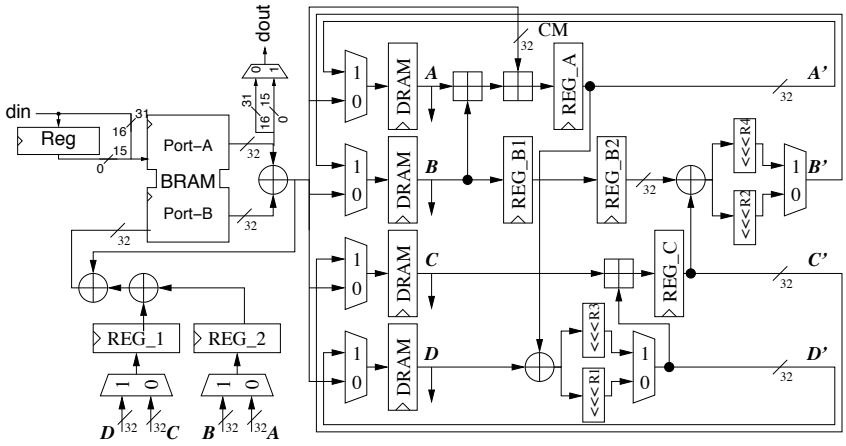


Fig. 2: Blockdiagram of BLAKE

3.1 BLAKE

Our implementation of BLAKE-32 and BLAKE-256 (Fig. 2) uses the BRAM to store the message, constants, initial hash values, chaining hash values, salt, and a counter. It takes 16 clock cycles to initialize the internal state. The internal V-state is stored in four Distributed RAMs which can be accessed easily for each G-Function. We implemented 1/2 G-Function with interleaved pipeline stages such that it takes 20 clock cycles for computing 8 G-Functions. Additionally we need one extra clock cycle to store the registered value back into Distributed RAM leading to a total of 21 clock cycles for each round. The G-Function requires permuted values of constants and messages which are stored in BRAM. This permutation doesn't have a repeatable pattern, therefore the BRAM addressing alone consumes 70% of the size of the controller. For round-3 of the SHA-3 competition a tweak was introduced for BLAKE which increases the number of rounds by four resulting in an increase in area consumption as the permutation function needs data values for 4 more rounds from BRAM. This version of BLAKE is called BLAKE-256.

3.2 Grøstl

Grøstl [22] is based on the AES round with the following sequence of operations: AddRoundConstant, SubBytes, ShiftBytes, and MixBytes. In our implementation (Fig. 3) the state, consisting of two 512-bit matrices P & Q, is stored in 16 4x8 Distributed RAMs. Each row is stored in one Distributed RAM. In order to get the first 64-bit column we access byte0 from RAM0, byte1 from RAM1... etc. This access scheme performs the ShiftBytes operation with which we start each round. SubBytes is implemented using 4 pipelined S-Boxes which are described as logic functions [**?**]. The multiplier takes a column from SubBytes and produces 32 bits of the new column in one clock cycle, the remaining 32 bits in the

second clock cycle. It takes a total 3 clock cycles to produce a new column. Each round of P and Q computes 16 new columns which takes 48 clock cycles. We interleave the computations of P and Q through the pipeline. The XOR operation ($P \oplus Q \oplus h$) takes 32 clock cycles. So a block of message is processed in 515 clock cycles ($48 \cdot 10 + 32 + 3$ clock cycles to fill the pipeline). BRAM is used in dual port mode and stores the initialization vector and the intermediate hash (h). For round-3 of the SHA-3 competition a tweak was introduced which changes the shifts in the ShiftBytes operation and introduces a different AddRoundConstant function. This has minimal effect on the area consumption and does not change the overall architecture. Grøstl from round-2 is now called Grøstl-0.

3.3 JH

Our implementation of JH (Fig. 4) stores the state and constants in BRAM. Two independent 32x8 Distributed RAMs store the state of the round constant generator. The BRAM is used as two independent memories to simplify the control logic and ease synchronization with the round constant generator. During initialization, which takes 35 cycles, the location of the state in BRAM is initialized with the precomputed starting value of $H^{(0)}$ from another address in the same BRAM. Grouping and de-grouping take advantage of the dual port memory and read two addresses from the BRAM simultaneously, retaining 4 bits from each address in registers and discarding the rest. This is repeated 4 times to write a full 32-bit value back into the BRAM and takes 160 cycles. The core round function is 32 times vertically folded and contains a pipelined permutation function. It needs a total of 34 cycles per round. Difficulties in creating this implementation were the memory access delays and the nonconsecutive read and write addresses. The tweak for round-3 of the SHA-3 competition increases the number of rounds to 42. This version of JH is called JH42.

3.4 Keccak

One round of Keccak [9] applies five functions, θ, ρ, π, χ, and ι to its state. In our implementation (Fig. 5), we store the state and the round constant in BRAM. The basic operations of Keccak use 64-bit data values which is also the maximum that we can read or write to BRAM in a single clock cycle. Therefore, in order to make the design more efficient we decided to quasi pipeline our functions. We have merged the θ and ρ functions. The later function uses a variable rotator. A barrel shifter consumes 192 slices on Spartan-3, hence we build a shifter that can only shift the 25 offsets Keccak needs. It uses on average 1.5 clock cycles per rotation and consumes only 128 slices. We use dedicated write cycles to accommodate the data rearrangement of the π function. These three functions take a total of 91 clock cycles. The χ function takes its operands from BRAM, applies a series of simple logical operations, and stores the result into BRAM. The ι operation combines a round constant with one 64-bit value of the new state. These operations take an additional 63 clock cycles. A single round operation thus takes a total of 154 clock cycles. Reducing the number of

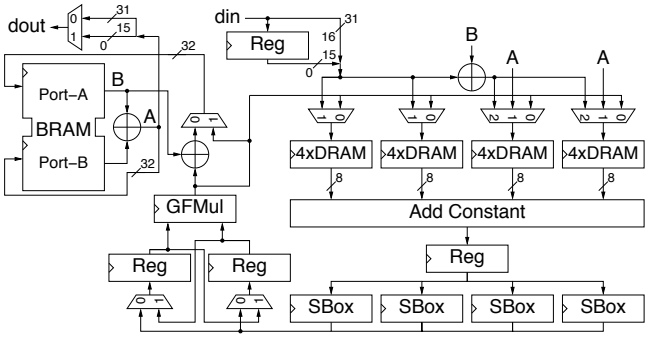


Fig. 3: Blockdiagram of Grøstl-0

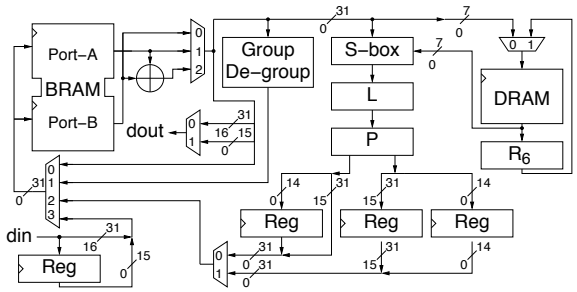


Fig. 4: Blockdiagram of JH

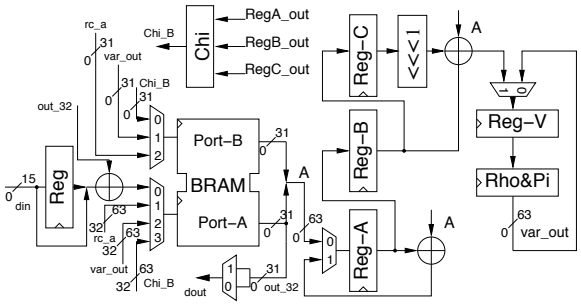


Fig. 5: Blockdiagram of Keccak

clock cycles would require more BRAM accesses which is not possible or more registers or Distributed RAMs, both would increase area consumption.

3.5 Skein

The basic building block of Skein [18] is a Mix function which consists of 64-bit ADD, XOR and rotate operations. Even though all main operations are of 64-bit size we chose a 32-bit datapath for our Skein-512-256 implementation (Fig. 6). The BRAM limits us to read two 32-bit values per clock cycle. Hence, we read the 32 LSB of two operands in one clock cycle and perform an addition, followed by the 32 MSB in the next clock cycle. The big advantage of this strategy is that we can use a 32-bit adder which has a much shorter critical path than a 64-bit adder. The variable rotator is realized as a 64-bit barrel shifter with a single pipeline stage. It is the single largest block in our design. We use the BRAM to store the state and the processed IV. This allows us to skip the initialization. Hashing a single block of data takes 72 rounds and 19 key injections. Within each round the Mix function is used 4-times which takes 20 clock cycles. The first key injection takes 48 clock cycles and all following 45, resulting in 858 clock cycles per block. After the round function completes a new chaining value has to be generated which is used to generate the new key for the next message block. Permutations take an additional 109 clock cycles. When all message blocks are processed the message finalization starts. This finalization is equivalent to processing a message block, except no new key has to be generated. For round-3 of the SHA-3 competition only a single constant got changed. This neither affects the size of our implementation nor its speed.

4 Results and Conclusions

4.1 Implementation Results

The results of our implementations are summarized by the graph shown in Fig. 7. It shows the area consumption of each implementation on the x-axis and the throughput on the y-axis. Hash functions where the implementations did not change between round 2 and round 3 of the SHA competition are marked as "Round 2 & 3". Otherwise they are grouped by competition rounds. It can be seen that all implementations fall within our target range of 400 to 600 slices. Each algorithm was optimized for maximum throughput without violating the area constraint. The throughput over area ratio of each implementation on a Xilinx Spartan-3 and, due to page limitations, only of the finalists on other FPGA devices is shown in Fig. 8. Each graph is sorted by throughput over area for long messages according to (3) in red. The results for short messages of one block only are computed according to (2) and shown in light-blue. The order of the algorithms differs slightly depending on the implementation platform. Shabal outperforms all other hash functions for long messages. Of the five finalists, BLAKE-256 performs better on Xilinx devices, Grøstl on the Altera device. It

Table 1: Implementation details of SHA-3 candidate implementations

Algorithm	Datapath Size (bits)	Rounds	Clock cycles per Round	Additional Clock cycles	Clock cycles per block p	Clock cycles per byte of message	Implementation Details
BLAKE-32	32	10	21	24	234	3.7	See description in Sect. 3.1
BLAKE-256	32	14	21	24	318	5.0	.
BMW	32	1	730	0	730	11.4	BRAM stores IV and state, shifter and rotator implemented separately to perform in parallel, the outputs of rotator, shifter and the adders are registered to reduce delay.
CubeHash	32	16	58	0	928	29.0	Processed IV and state is stored in BRAM. Finalization is equivalent to 10*Rounds. Distributed RAMs store immediate values to overcome swapping.
ECHO	64	8	290	129	2449	12.8	Message and state stored in BRAM. S-box implemented as logic. Round is generated using 32-bit adder and 32-bit register and stored in Distributed RAM.
Fugue	32	1	61	0	61	15.3	State is stored in BRAM. Super-Mix is created from 4 AES S-Boxes and fixed rotations.
Grøstl(-0)	32	10	48	35	515	7.8	See description in Sect. 3.2.
Hamsi	64	3	16	26	74	18.5	Message stored in register. Expansion function constants 32*128 bits, initialization constants, and state in BRAM. p/pf function constants in Distributed Ram. 64-bit p/pf function.
JH	32	36	34	384	1608	25.1	See description in Sect. 3.3
JH42	32	42	34	385	1813	28.3	.
Keccak	64	24	154	0	3696	27.2	See description in Sect. 3.4.
Luffa	32	8	66	78	606	18.9	Message injection through serialized XOR. Tweak uses shift register. The SubCrumb is implemented as ROM. Constants, IVs and state stored in BRAM
Shabal	32	48	1	16	64	1.0	Design from [17] with IVs and C-state located in BRAM.
SHAvite-3	32	12	38	288	744	11.6	BRAM stores state and IV. 4 S-Boxes implemented as ROM. 4x32-bit shift register is tapped at 32 bit positions to provide data for MixColumns multiplication. Key generation uses same datapath and takes 288 clock cycles.
Skein	32	72	20	967	2407	37.6	See description in Sect. 3.5.

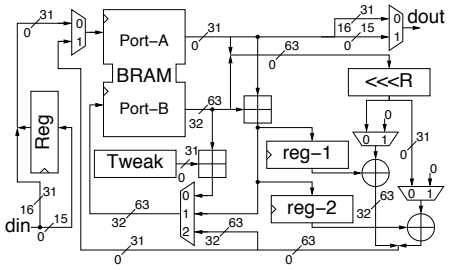


Fig. 6: Blockdiagram of Skein

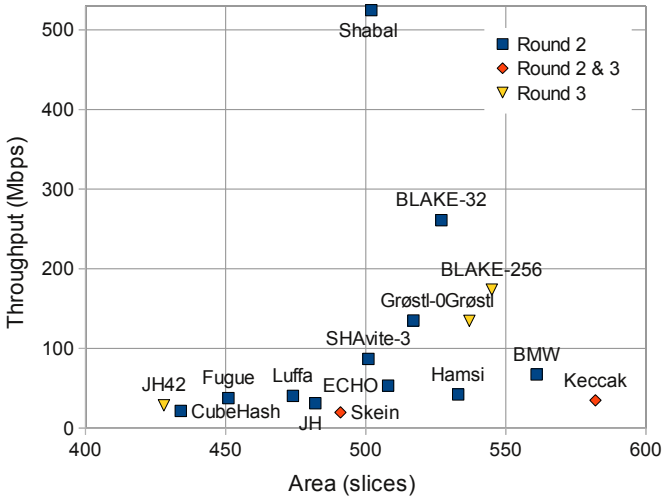


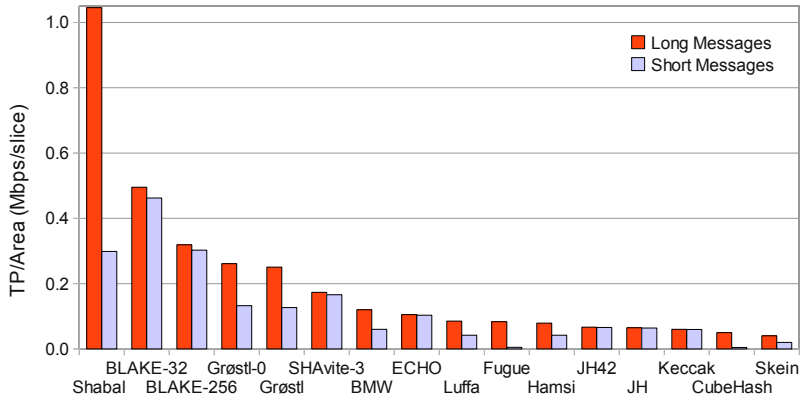
Fig. 7: Throughput over area of our SHA-3 implementations on Xilinx Spartan-3

Table 2: Throughput formulae for our implementations of SHA-3 candidates

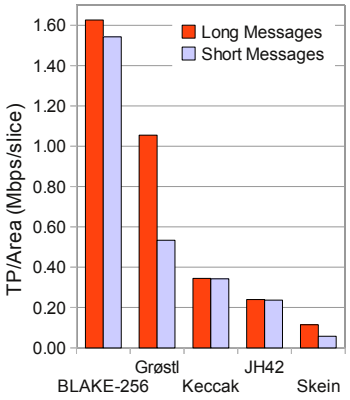
Algorithm	Specification	Block Size (bits) b	Clock Cycles to hash N blocks $clk =$ $st + (\ l + \ p) \cdot N + \ end$	Throughput $\frac{b}{(l+p) \cdot T}$
BLAKE-32	[4]	512	$2 + (\ 32 + \ 234) \cdot N + \ 17$	$512/(\ 266 \cdot T)$
BLAKE-256	[4]	512	$2 + (\ 32 + \ 318) \cdot N + \ 17$	$512/(\ 350 \cdot T)$
BMW	[24]	512	$2 + (\ 32 + \ 730) \cdot N + \ 757$	$512/(\ 762 \cdot T)$
CubeHash	[7]	256	$2 + (\ 16 + \ 928) \cdot N + 9312$	$256/(\ 944 \cdot T)$
ECHO	[6]	1536	$18 + (\ 96 + 2449) \cdot N + \ 17$	$1536/(2545 \cdot T)$
Fugue	[25]	32	$33 + (\ 2 + \ 61) \cdot N + \ 990$	$32/(\ 63 \cdot T)$
Grøstl-0, Grøstl	[22], [23]	512	$2 + (\ 32 + \ 515) \cdot N + \ 532$	$512/(\ 547 \cdot T)$
Hamsi	[32]	32	$2 + (\ 2 + \ 74) \cdot N + \ 65$	$32/(\ 76 \cdot T)$
JH	[41]	512	$35 + (\ 32 + 1608) \cdot N - \ 15$	$512/(1640 \cdot T)$
JH42	[42]	512	$35 + (\ 32 + 1813) \cdot N - \ 15$	$512/(1845 \cdot T)$
Keccak	[9], [10]	1088	$2 + (\ 68 + 3696) \cdot N + \ 17$	$1088/(3764 \cdot T)$
Luffa	[16]	256	$2 + (\ 16 + \ 606) \cdot N + \ 647$	$256/(\ 622 \cdot T)$
Shabal	[13]	512	$32 + (\ 32 + \ 64) \cdot N + \ 208$	$512/(\ 96 \cdot T)$
SHAvite-3	[12]	512	$18 + (\ 32 + \ 744) \cdot N + \ 17$	$512/(\ 776 \cdot T)$
Skein	[18]	512	$5 + (\ 32 + 2407) \cdot N + 2423$	$512/(2439 \cdot T)$

can clearly be seen, that algorithms that have a lengthy finalization step do not perform well for short messages. It is interesting to note, that all round-3 candidates which introduced a tweak after round-2 that requires a change in the datapath perform slightly worse after the tweak with the exception of JH42. Its tweak leads to an increase in the number of rounds, however, this penalty is compensated for by the simpler datapath resulting from dropping the half round. The detailed results of our implementations on Xilinx Spartan-3, Spartan-6, Virtex-V and Altera Cyclone-II devices are summarized in Tables 3 and 4. Both tables show first the results of the SHA-3 finalists followed by the remaining round-2 candidates.

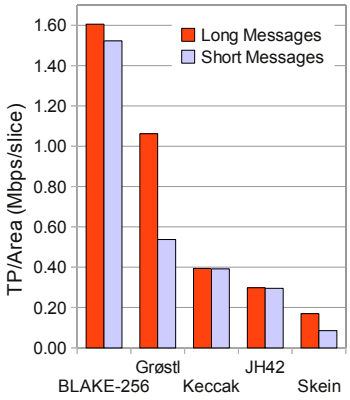
One challenge when implementing the hash functions for low-area, is the trade-off between simplicity of the datapath versus complexity of the control logic. Reuse of components in the datapath for several different functions, clock cycle optimized usage of the BRAM, pipelining complex functions to achieve low critical path delay, and interleaving elementary functions of a hash algorithm all lead to a better throughput over area ratio of the datapath. However, all of them also lead to a complex control logic. Figure 8e shows how many percent of the total area consumption of each hash function was used for the datapath and for the control unit. A small control unit indicates that the control signals needed for an algorithm are very regular, i.e. the algorithm is very regular and can be easily scaled down for lightweight implementations. A large control unit might indicate the opposite. On the other hand, BLAKE-256 has a very good throughput over area ratio, yet, due to its permutation schedule with 210 entries it requires a relatively large control unit.



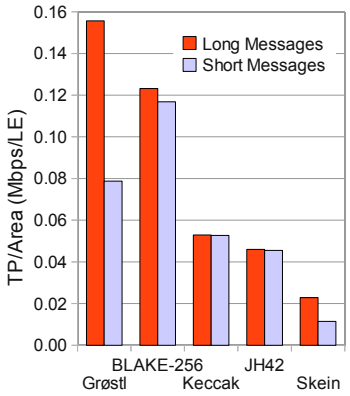
(a) Throughput over area ratio on Xilinx Spartan-3



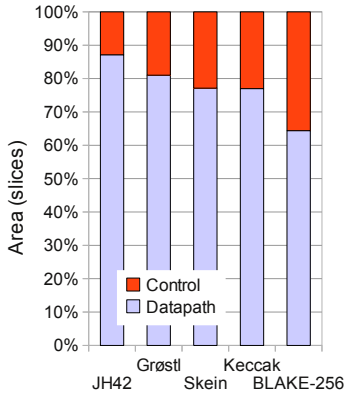
(b) on Spartan-6



(c) on Xilinx Virtex-V



(d) on Altera Cyclone-II



(e) Ratio of Datapath vs. Control Unit area consumption on Xilinx Spartan-3

Fig. 8: Lightweight SHA-3 implementations results

Table 3: Implementation results of our implementations of SHA-3 candidates

Algorithm	Xilinx xc3s50-5			Xilinx xc6slx4csg-3			Xilinx xc5vlx20-2			Altera ep2c5f256c6		
	Area (slices)	Block RAMs	Maximum Delay (ns) T	Area (slices)	Block RAMs	Maximum Delay (ns) T	Area (slices)	Block RAMs	Maximum Delay (ns) T	Area (LEs)	Memory Bits	Maximum Delay (ns) T
BLAKE-256	545	1	8.42	139	1	6.47	212	1	4.30	1,365	2,048	8.70
Grøstl	537	1	6.95	163	1	5.44	234	1	3.77	1,026	2,560	5.86
JH42	428	1	9.74	142	1	8.16	176	1	5.28	702	8,704	8.59
Keccak	582	1	8.30	142	1	5.91	196	1	3.74	996	8,192	5.48
Skein	491	1	10.68	227	1	8.07	215	1	5.74	930	4,096	9.89
BLAKE-32	527	1	7.38	138	1	7.23	238	1	4.37	1,262	2,048	8.71
BMW	561	1	9.99	183	1	7.15	233	1	5.16	1,104	8,192	9.45
CubeHash	434	1	12.58	131	1	8.11	231	1	6.05	2,761	16,384	9.18
ECHO	508	1	11.33	155	1	9.72	232	1	5.34	1,069	16,512	10.14
Fugue	451	1	13.48	269	1	12.84	209	1	6.43	940	16,384	7.81
Grøstl-0	517	1	6.93	163	1	5.23	232	1	3.61	1,020	2,560	5.93
Hamsi	533	1	9.97	162	1	12.83	208	1	5.28	687	10,240	8.87
JH	482	1	9.99	180	1	8.35	161	1	4.97	702	8,704	8.59
Luffa	474	1	10.17	107	1	6.86	176	1	5.08	946	8,192	7.66
Shabal	502	1	10.17	165	1	7.66	231	1	4.86	2,093	1,760	9.16
Shavite-3	501	1	7.60	120	1	5.24	136	1	3.37	471	16,384	7.01

4.2 Comparison with Other Reported Results

We compare our results with previously reported ones in Table 5. Due to space limitations we concentrate only on the SHA-3 finalists. Even though our primary design target is Xilinx Spartan-3, we synthesized our implementations for other devices to match the devices of reported results. This puts our designs at a disadvantage as we could not take full advantage of their features. Most notably, pipeline stages might become unbalanced when synthesizing a design for a device with 4-input LUTs on a 6-input LUT device. The compact BLAKE-32 design reported in [11] uses two BRAMs, one for the controller and one to store the message, constant, internal states, hash values and counters. In order to have a fair comparison we moved the control unit of our BLAKE-32 design to a second BRAM. This enabled us to reduce the number of clock cycles. The designs are quite comparable in terms of throughput to area ratio. Furthermore, the designers of [11] did not include the clock cycles needed for loading a message block which would bring our designs even closer. Our BLAKE-256 design performs significantly better than the design by Kerckhof [30]. Our result for Keccak on Virtex-V compares favorably with the design reported in [8]. For comparison we are assuming that we can equate their system memory with one BRAM. The reason for the difference in clock cycles is that we chose to use a quasi pipelined design whereas [8] has implemented each of the functions separately. However, Kerckhof's result for Keccak [30] is better than ours even though their area is

Table 4: Throughput results of our lightweight implementations of SHA-3 candidates. First are round-3 results followed by round-2 results.

Message	Xilinx xc3s50-5				Xilinx xc6slx4csg225-3			
	Long		Short		Long		Short	
Algorithm	Throughput (Mbps)	TP/Area (Mbps/slice)	Throughput (Mbps)	TP/Area (Mbps/slice)	Throughput (Mbps)	TP/Area (Mbps/slice)	Throughput (Mbps)	TP/Area (Mbps/slice)
BLAKE-256	173.8	0.32	164.8	0.302	226.0	1.63	214.4	1.542
Grøstl	134.6	0.25	68.1	0.127	171.9	1.05	87.0	0.534
JH-42	28.5	0.07	28.2	0.066	34.0	0.24	33.6	0.237
Keccak	34.8	0.06	34.7	0.060	48.9	0.34	48.6	0.343
Skein	19.7	0.04	9.9	0.020	26.0	0.11	13.0	0.057
BLAKE-32	261.0	0.50	243.6	0.462	266.4	1.93	248.6	1.802
BMW	67.3	0.12	33.7	0.060	93.9	0.51	47.1	0.257
CubeHash	21.6	0.05	2.0	0.005	33.5	0.26	3.1	0.024
ECHO	53.3	0.10	52.5	0.103	62.1	0.40	61.2	0.395
Fugue	37.7	0.08	2.2	0.005	39.6	0.15	2.3	0.009
Grøstl-0	135.1	0.26	68.4	0.132	179.0	1.10	90.6	0.556
Hamsi	42.2	0.08	22.4	0.042	32.8	0.20	17.4	0.108
JH	31.2	0.06	30.9	0.064	37.4	0.21	37.0	0.205
Luffa	40.5	0.09	19.8	0.042	60.0	0.56	29.4	0.275
Shabal	524.7	1.05	149.9	0.299	687.5	4.17	196.4	1.190
Shavite-3	86.8	0.17	83.1	0.166	126.0	1.05	120.6	1.005

Message	Xilinx xc5vlx20-2				Altera ep2c5f256c6			
	Long		Short		Long		Short	
Algorithm	TP (Mbps)	(Mbps /slice)	TP (Mbps)	(Mbps /slice)	TP (Mbps)	(Mbps /LE)	TP (Mbps)	(Mbps /LE)
BLAKE-256	340.4	1.61	322.8	1.523	168.1	0.12	159.4	0.117
Grøstl	248.5	1.06	125.7	0.537	159.7	0.16	80.8	0.079
JH-42	52.5	0.30	52.0	0.295	32.3	0.05	31.4	0.046
Keccak	77.2	0.39	76.8	0.392	52.7	0.05	52.5	0.053
Skein	36.6	0.17	18.3	0.085	21.2	0.02	10.6	0.011
BLAKE-32	440.8	1.85	411.4	1.728	220.9	0.18	206.2	0.163
BMW	130.3	0.56	65.3	0.280	71.1	0.06	35.6	0.032
CubeHash	44.8	0.19	4.1	0.018	29.6	0.01	2.7	0.001
ECHO	113.0	0.49	111.5	0.481	59.5	0.06	58.7	0.055
Fugue	79.0	0.38	4.6	0.022	65.0	0.07	3.8	0.004
Grøstl-0	259.6	1.12	131.4	0.566	157.9	0.15	79.9	0.078
Hamsi	79.7	0.38	42.4	0.204	47.5	0.07	25.2	0.037
JH	62.8	0.39	62.0	0.385	36.3	0.05	35.9	0.051
Luffa	81.1	0.46	39.7	0.225	53.8	0.06	26.3	0.028
Shabal	1097.8	4.75	313.7	1.358	582.5	0.28	166.4	0.080
Shavite-3	196.1	1.44	187.6	1.380	94.1	0.20	90.1	0.191

Table 5: Comparison of lightweight implementations of SHA-3 finalists on Xilinx FPGAs ([TW] – this work)

Algorithm	Reference	Device	I/O Width	Datapath Width	Clock Cycles $(l+p)$	Area (slices)	Block RAMs	Maximum Delay (ns)	Throughput (Mbps)	TP/Area (Mbps/slice)
BLAKE-32	[11]	xc3s50-5	32	32	846	124	2	5.26	115.0	0.927
BLAKE-32	[TW]	xc3s50-5	16	32	220	360	2	7.38	315.6	0.877
BLAKE-256	[30]	xc6vlx75t-1	64	64	1,336	117	0	3.65	105.0	0.897
BLAKE-256	[TW]	xc6vlx75t-1	16	32	350	146	1	5.27	277.7	1.902
Grøstl-0	[29]	xc3s200	64	64	160	1,276	0	16.67	192.0	0.150
Grøstl-0	[TW]	xc3s200-5	16	32	547	529	1	7.15	131.0	0.248
Grøstl	[30]	xc6vlx75t-1	64	64	176	285	0	3.57	815.0	2.860
Grøstl	[TW]	xc6vlx75t-1	16	32	547	179	1	4.13	226.7	1.266
Grøstl	[28]	xc5v	32	64	160	470	0	2.82	1,132.0	2.409
Grøstl	[TW]	xc5v	16	32	547	234	1	3.77	248.5	1.062
JH42	[30]	xc6vlx75t-1	64	64	689	240	0	3.47	214.0	0.892
JH42	[TW]	xc6vlx75t-1	16	32	1,845	164	1	5.39	51.5	0.314
JH42	[28]	xc5v	32	8	6,466	205	0	2.93	27.0	0.132
JH42	[TW]	xc5v	16	32	1,845	176	1	5.28	52.5	0.299
Keccak	[8]	xc5vlx50-3	64	64	5,492	448	1	3.77	52.5	0.117
Keccak	[TW]	xc5vlx50-3	16	64	3,764	192	1	3.54	81.8	0.426
Keccak	[30]	xc6vlx75t-1	64	64	2,125	144	0	4.00	128.0	0.889
Keccak	[TW]	xc6vlx75t-1	16	64	3,764	154	1	3.69	78.3	0.509
Skein	[30]	xc6vlx75t-1	64	64	458	240	0	6.25	179.0	0.746
Skein	[TW]	xc6vlx75t-1	16	32	2,439	162	1	6.02	34.9	0.215
Skein	[28]	xc5v	32	64	585	555	0	3.69	237.0	0.427
Skein	[TW]	xc5v	16	32	2,439	215	1	5.74	36.6	0.170

slightly smaller. One reason is, that we have to use dedicated write cycles for the BRAM and that our rotator requires 1.5 clock cycles on average. The Grøstl-0 implementation reported in [29] is more than twice as large as our design, yet our throughput to area ratio is 1.5 times better. Their design processes the data in fewer clock cycles leading to a higher throughput. However, as they are using 8 S-boxes with no pipeline register in between, their delay is high and the area is comparatively large. The Grøstl implementations by Kerckhof [30] and Jungk [28] outperform our implementation. However, both are significantly larger, have a wider I/O, and a 64-bit datapath. We can make the same observation also for the JH implementation in [30] and Skein in [30] and [28]. Our area constraint and the BRAM restricts us to a 32-bit datapath. It is interesting to note though, that the throughput over area ratio is highly non-linear with the area of implementations of the same algorithm, i.e. the more area is available, the disproportional more the throughput improves. Jungk's implementation of JH [28] has a worse performance than ours due to its 8-bit datapath.

4.3 Conclusions

In this paper we presented the first comprehensive comparison of lightweight FPGA implementations of all SHA-3 finalists and all round-2 candidates with the exception of SIMD. All algorithms were implemented using the same assumptions, goals, tools, interface, and the same area optimization techniques. The lightweight implementations were evaluated with regards to their throughput over area ratio. The resulting ranking of algorithms is very different from implementations for best throughput over area reported in the literature [33], [19], [5]. The finalists with the best throughput over area ratio on Xilinx devices is BLAKE-256 followed by Grøstl. On the Altera Cyclone-II Grøstl is followed by BLAKE-256. JH42 and Keccak are very close to each other. The finalist with the lowest ratio is Skein. However, this ranking might change, when we change the available area or the BRAM requirement. As future work we would like to explore how much more area is needed for each of these algorithms in order to achieve a significant increase in throughput over area. This will give us an even better understanding of the scalability of the algorithms.

References

1. ATHENa results database. http://cryptography.gmu.edu/athenadb/, automated Tool for Hardware EvaluatioN project
2. The sha-3 zoo. http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo, eCRYPT, Information Societies Technology (IST) Programme of the European Commission
3. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. Federal Register/ Vol. 72, No. 212 (Nov 2007), notices 62212
4. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 proposal BLAKE. Submission to NIST (Round 3) (2010), http://131002.net/blake/blake.pdf
5. Baldwin, B., Hanley, N., Hamilton, M., Lu, L., Byrne, A., O'Neill, M., Marnane, W.P.: FPGA implementations of the round two SHA-3 candidates. Tech. rep., Second SHA-3 Candidate Conference (2010)
6. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 proposal: ECHO. Submission to NIST (updated) (Feb 2009), http://crypto.rd.francetelecom.com/echo/
7. Bernstein, D.J.: CubeHash specification (2.b.1). Submission to NIST (Round 2) (2009), http://cubehash.cr.yp.to/
8. Bertoni, G., Daemen, J., Peeters, M., Gilles, V.A.: Keccak function version 2.0 (Sep 2009)
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak sponge function family main document. http://keccak.noekeon.org (Apr 2009), version 1.2
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak SHA-3 submission. Submission to NIST (Round 3) (2011), http://keccak.noekeon.org/Keccak-submission-3.pdf
11. Beuchat, J.L., Okamoto, E., Yamazaki, T.: Compact implementations of BLAKE-32 and BLAKE-64 on FPGA. Cryptology ePrint Archive, Report 2010/173 (2010)
12. Biham, E., Dunkelman, O.: The SHAvite-3 hash function. Submission to NIST (Round 2) (2009), http://www.cs.technion.ac.il/~orrd/SHAvite-3/Spec.15.09.09.pdf

13. Bresson, E., et al.: Shabal, a submission to NISTs cryptographic hash algorithm competition. Submission to NIST (October 2008), http://ehash.iaik.tugraz.at/uploads/6/6c/Shabal.pdf
14. Chen, Z., Morozov, S., Schaumont, P.: A hardware interface for hashing algorithms. Cryptology ePrint Archive, Report 2008/529 (2008), http://eprint.iacr.org/
15. Cryptographic Engineering Research Group, George Mason University: Hardware Interface of a Secure Hash Algorithm (SHA), v. 1.4 edn. (Jan 2010)
16. De Cannière, C., Sato, H., Watanabe, D.: Hash function Luffa: Specification. Submission to NIST (Round 2) (Oct 2009), http://www.sdl.hitachi.co.jp/crypto/luffa/Luffa_v2_Specification_20091002.pdf
17. Detrey, J., Gaudry, P., Khalfallah, K.: A low-area yet performant FPGA implementation of Shabal. Cryptology ePrint Archive, Report 2010/292 (2010)
18. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein hash function family. Submission to NIST (Round 3) (2010), http://www.skein-hash.info/sites/default/files/skein1.3.pdf
19. Gaj, K., Homsirikamol, E., Rogawski, M.: Fair and comprehensive methodology for comparing hardware performance of fourteen round two SHA-3 candidates using FPGA. In: Cryptographic Hardware and Embedded Systems, CHES 2010. LNCS, Springer (2010)
20. Gaj, K., Kaps, J.P., Amirineni, V., Rogawski, M., Homsirikamol, E., Brewster, B.Y.: ATHENa – Automated Tool for Hardware EvaluatioN: Toward fair and comprehensive benchmarking of cryptographic hardware using FPGAs. In: 20th International Conference on Field Programmable Logic and Applications - FPL 2010. pp. 414–421. IEEE (2010)
21. García-Vargas, I., Senhadji-Navarro, R., Jiménez-Moreno, G., Civit-Balcells, A., Guerra-Gutiérrez, P.: Rom-based finite state machine implementation in low cost FPGAs. In: International Symposium on Industrial Electronics, ISIE 2007. pp. 2342–2347. IEEE (June 2007)
22. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schäffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate. Submission to NIST (Oct 2008), http://www.groestl.info/
23. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schäffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate. Submission to NIST (Round 3) (2011), http://www.groestl.info/Groestl.pdf
24. Gligoroski, D., Klima, V., Knapskog, S.J., El-Hadedy, M., Amundsen, J., Mjølsnes, S.F.: Cryptographic hash function blue midnight wish. Submission to NIST (Round 2) (Sep 2009), http://people.item.ntnu.no/~danilog/Hash/BMW-SecondRound/Supporting_Documentation/BlueMidnightWishDocumentation.pdf
25. Halevi, S., Hall, W.E., Jutla, C.S.: The hash function Fugue. Submission to NIST (updated) (Sep 2009), http://domino.research.ibm.com/comm/research_projects.nsf/pages/fugue.index.html
26. Homsirikamol, E., Rogawski, M., Gaj, K.: Comparing hardware performance of fourteen round two SHA-3 candidates using FPGAs. Cryptology ePrint Archive, Report 2010/445 (2010), http://eprint.iacr.org/
27. Homsirikamol, E., Rogawski, M., Gaj, K.: Throughput vs. area trade-offs architectures of five round 3 SHA-3 candidates implemented using Xilinx and Altera FPGAs. In: Workshop on Cryptographic Hardware and Embedded Systems CHES. LNCS, Springer (Sep 2011)
28. Jungk, B.: Compact implementations of Grøstl, JH and Skein for FPGAs (May 2011), eCRYPT II Hash Workshop 2011

29. Jungk, B., Reith, S.: On FPGA-based implementations of Grøstl. Cryptology ePrint Archive, Report 2010/260 (2010)
30. Kerckhof, S., Durvaux, F., Veyrat-Charvillon, N., Regazzoni, F., de Dormale, G.M., Standaert, F.X.: Compact FPGA implementations of the five SHA-3 finalists (May 2011), eCRYPT II Hash Workshop 2011
31. Kobayashi, K., Ikegami, J., Matsuo, S., Sakiyama, K., Ohta, K.: Evaluation of hardware performance for the SHA-3 candidates using SASEBO-GII. `http://eprint.iacr.org/2010/010` (Jan 2010)
32. Özgül Küçük: The hash function Hamsi. Submission to NIST (updated) (2009), http://www.cosic.esat.kuleuven.be/publications/article-1203.pdf
33. Matsuo, S., Knežević, M., Schaumont, P., Verbauwhede, I., Satoh, A., Sakiyama, K., Ota, K.: How can we conduct "fair and consistent" hardware evaluation for SHA-3 candidate? Tech. rep., Second SHA-3 Candidate Conference (2010)
34. Namin, A., Hasan, M.: Hardware implementation of the compression function for selected SHA-3 candidates. In: CACR 2009-28. p. 29 (July 2009)
35. Namin, A., Hasan, M.: Implementation of the compression function for selected SHA-3 candidates on FPGA. In: Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW). pp. 1–4. IEEE (2010)
36. Rawski, M., Selvaraj, H., Luba, T.: An application of functional decomposition in ROM-based FSM implementation in FPGA devices. J. Syst. Archit. 51(6-7), 424–434 (2005)
37. Research centre for information security (RCIS), National institute of advanced industrial science and technology (AIST): Side-channel attack standard evaluation board SASEBOGII specification, version 1.01 edn. (Nov 2009)
38. Skylarov, V.: Synthesis and implementation of RAM-based finite state machines in FPGAs. In: Hartenstein, R.W., Grünbacher, H. (eds.) Field-Programmable Logic and Applications – FPL'00. LNCS, vol. 1896, pp. 718–728. Springer-Verlag (2000)
39. Sönmez Turan, M., Perlner, R., Bassham, L.E., Burr, W., Chang, D., jen Chang, S., Dworkin, M.J., Kelsey, J.M., Paul, S., Peralta, R.: Status report on the second round of the SHA-3 cryptographic hash algorithm competition. NIST Interagency Report 7764, NIST, Gaithersburg, MD, USA (Feb 2011)
40. Tuan, T., Kao, S., Rahman, A., Das, S., Trimberger, S.: A 90nm low-power FPGA for battery-powered applications. In: International symposium on Field programmable gate arrays - FPGA '06. pp. 3–11. ACM/SIGDA, ACM, New York, NY, USA (2006)
41. Wu, H.: The hash function JH. Submission to NIST (updated) (Sep 2009), `http://icsd.i2r.a-star.edu.sg/staff/hongjun/jh/`
42. Wu, H.: The hash function JH. Submission to NIST (round 3) (2011), `http://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf`