Fair and Comprehensive Methodology for Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs*

Kris Gaj, Ekawat Homsirikamol, and Marcin Rogawski

ECE Department, George Mason University, Fairfax, VA 22030, U.S.A.
{kgaj,ehomsiri,mrogawsk}@gmu.edu
http://cryptography.gmu.edu

Abstract. Performance in hardware has been demonstrated to be an important factor in the evaluation of candidates for cryptographic standards. Up to now, no consensus exists on how such an evaluation should be performed in order to make it fair, transparent, practical, and acceptable for the majority of the cryptographic community. In this paper, we formulate a proposal for a fair and comprehensive evaluation methodology, and apply it to the comparison of hardware performance of 14 Round 2 SHA-3 candidates. The most important aspects of our methodology include the definition of clear performance metrics, the development of a uniform and practical interface, generation of multiple sets of results for several representative FPGA families from two major vendors, and the application of a simple procedure to convert multiple sets of results into a single ranking.

Keywords: benchmarking, hash functions, SHA-3, FPGA.

1 Introduction and Motivation

Starting from the Advanced Encryption Standard (AES) contest organized by NIST in 1997-2000 [1], open contests have become a method of choice for selecting cryptographic standards in the U.S. and over the world. The AES contest in the U.S. was followed by the NESSIE competition in Europe [2], CRYPTREC in Japan, and eSTREAM in Europe [3].

Four typical criteria taken into account in the evaluation of candidates are: security, performance in software, performance in hardware, and flexibility. While security is commonly recognized as the most important evaluation criterion, it is also a measure that is most difficult to evaluate and quantify, especially during a relatively short period of time reserved for the majority of contests. A typical outcome is that, after eliminating a fraction of candidates based on security flaws, a significant number of remaining candidates fail to demonstrate any easy

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

S. Mangard and F.-X. Standaert (Eds.): CHES 2010, LNCS 6225, pp. 264–278, 2010.

to identify security weaknesses, and as a result are judged to have adequate security.

Performance in software and hardware are next in line to clearly differentiate among the candidates for a cryptographic standard. Interestingly, the differences among the cryptographic algorithms in terms of hardware performance seem to be particularly large, and often serve as a tiebreaker when other criteria fail to identify a clear winner. For example, in the AES contest, the difference in hardware speed between the two fastest final candidates (Serpent and Rijndael) and the slowest one (Mars) was by a factor of seven [1][4]; in the eSTREAM competition the spread of results among the eight top candidates qualified to the final round was by a factor of 500 in terms of speed (Trivium x64 vs. Pomaranch), and by a factor of 30 in terms of area (Grain v1 vs. Edon80) [5][6].

At this point, the focus of the attention of the entire cryptographic community is on the SHA-3 contest for a new hash function standard, organized by NIST [7][8]. The contest is now in its second round, with 14 candidates remaining in the competition. The evaluation is scheduled to continue until the second quarter of 2012.

In spite of the progress made during previous competitions, no clear and commonly accepted methodology exists for comparing hardware performance of cryptographic algorithms [9]. The majority of the reported evaluations have been performed on an ad-hoc basis, and focused on one particular technology and one particular family of hardware devices. Other pitfalls included the lack of a uniform interface, performance metrics, and optimization criteria. These pitfalls are compounded by different skills of designers, using two different hardware description languages, and no clear way of compressing multiple results to a single ranking. In this paper, we address all the aforementioned issues, and propose a clear, fair, and comprehensive methodology for comparing hardware performance of SHA-3 candidates and any future algorithms competing to become a new cryptographic standard.

The hardware evaluation of SHA-3 candidates started shortly after announcing the specifications and reference software implementations of 51 algorithms submitted to the contest [7][8][10]. The majority of initial comparisons were limited to less than five candidates, and their results have been published at [10]. The more comprehensive efforts became feasible only after NISTs announcement of 14 candidates qualified to the second round of the competition in July 2009. Since then, two comprehensive studies have been reported in the Cryptology ePrint Archive [11][12]. The first, from the University of Graz, has focused on ASIC technology, the second from two institutions in Japan, has focused on the use of the FPGA-based SASEBO-GII board from AIST, Japan. Although both studies generated quite comprehensive results for their respective technologies, they did not quite address the issues of the uniform methodology, which could be accepted and used by a larger number of research teams. Our study is intended to fill this gap, and put forward the proposal that could be evaluated and commented on by a larger cryptographic community.

2 Choice of a Language, FPGA Devices, and Tools

Out of two major hardware description languages used in industry, VHDL and Verilog HDL, we choose VHDL. We believe that either of the two languages is perfectly suited for the implementation and comparison of SHA-3 candidates, as long as all candidates are described in the same language. Using two different languages to describe different candidates may introduce an undesired bias to the evaluation.

FPGA devices from two major vendors, Xilinx and Altera, dominate the market with about 90% of the market share. We therefore feel that it is appropriate to focus on FPGA devices from these two companies. In this study, we have chosen to use seven families of FPGA devices from Xilinx and Altera. These families include two major groups, those optimized for minimum cost (Spartan 3 from Xilinx, and Cyclone II and III from Altera) and those optimized for high performance (Virtex 4 and 5 from Xilinx, and Stratix II and III from Altera). Within each family, we use devices with the highest speed grade, and the largest number of pins.

As CAD tools, we have selected tools developed by FPGA vendors themselves: Xilinx ISE Design Suite v. 11.1 (including Xilinx XST, used for synthesis) and Altera Quartus II v. 9.1 Subscription Edition Software.

3 Performance Metrics for FPGAs

Speed. In order to characterize the speed of the hardware implementation of a hash function, we suggest using Throughput, understood as a throughput (number of input bits processed per unit of time) for long messages. To be exact, we define Throughput using the following formula:

$$Throughput = \frac{block\_size}{T \cdot (HTime(N+1) - HTime(N))} \tag{1}$$

where $block\_size$ is a message block size, characteristic for each hash function (as defined in the function specification, and shown in Table 3), $HTime(N)$ is a total number of clock cycles necessary to hash an N-block message, T is a clock period, different and characteristic for each hardware implementation of a specific hash function.

In this paper, we provide the exact formulas for $HTime(N)$ for each SHA-3 candidate, and values of $f = 1/T$ for each algorithm–FPGA device pair (see Tables 3 and 6).

For short messages, it is more important to evaluate the total time required to process a message of a given size (rather than throughput). The size of the message can be chosen depending on the requirements of an application. For example, in the eBASH study of software implementations of hash functions, execution times for all sizes of messages, from 0-bytes (empty message) to 4096 bytes, are reported, and five specific sizes 8, 64, 576, 1536, and 4096 are featured in the tables [13]. The generic formulas we include in this paper (see Table 3) allow the calculation of the execution times for any message size.

In order to characterize the capability of a given hash function implementation for processing short messages, we present in this study the comparison of execution times for an empty message (one block of data after padding) and a 100-byte (800-bits) message before padding (which becomes equivalent for majority, but not all, of the investigated functions to 1024 bits after padding).

Resource Utilization/Area. Resource utilization is particularly difficult to compare fairly in FPGAs, and is often a source of various evaluation pitfalls. First, the basic programmable block (such as CLB slice in Xilinx FPGAs) has a different structure and different capabilities for various FPGA families from different vendors. Taking this issue into account, we suggest avoiding any comparisons across family lines. Secondly, all modern FPGAs include multiple dedicated resources, which can be used to implement specific functionality. These resources include Block RAMs (BRAMs), multipliers (MULs), and DSP units in Xilinx FPGAs, and memory blocks, multipliers, and DSP units in Altera FPGAs. In order to implement a specific operation, some of these resources may be interchangable, but there is no clear conversion factor to express one resource in terms of the other.

Therefore, we suggest in the general case, treating resource utilization as a vector, with coordinates specific to a given FPGA family. For example,

$$Resource\_Utilization_{Spartan3} = (\#CLBslices, \#BRAMs, \#MULs) \quad (2)$$

Taking into account that vectors cannot be easily compared to each other, we have decided to opt out of using any dedicated resources in the hash function implementations used for our comparison. Thus, all coordinates of our vectors, other than the first one have been forced (by choosing appropriate options of the synthesis and implementation tools) to be zero. This way, our resource utilization (further referred to as Area) is characterized using a single number, specific to the given family of FPGAs, namely the number of CLB slices ($\#CLBslices$) for Xilinx FPGAs, the number of Logic Elements ($\#LE$) for Cyclone II and Cyclone III, and the number of Adaptive Look-Up Tables ($\#ALUT$) in Stratix II and Stratix III.

4 Uniform Interface

In order to remove any ambiguity in the definition of our hardware cores for SHA-3 candidates, and in order to make our implementations as practical as possible, we have developed an interface shown in Fig. 1a, and described below. In a typical scenario, the SHA core is assumed to be surrounded by two standard FIFO modules: Input FIFO and Output FIFO, as shown in Fig. 1b. In this configuration, SHA core is an active module, while a surrounding logic (FIFOs) is passive. Passive logic is much easier to implement, and in our case is composed of standard logic components, FIFOs, available in any major library of IP cores.

Each FIFO module generates signals empty and full, which indicate that the FIFO is empty and/or full, respectively. Each FIFO accepts control signals write and read, indicating that the FIFO is being written to and/or read from, respectively.

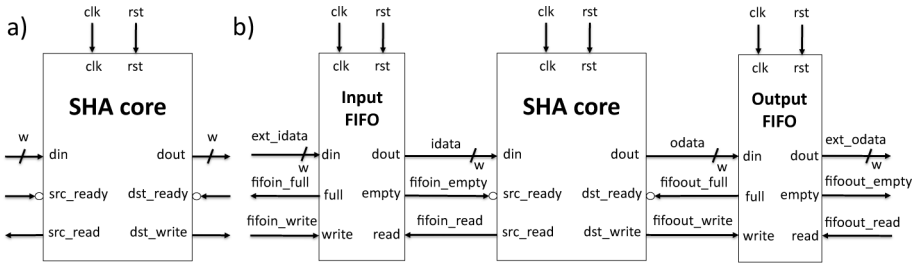

Fig. 1. a) Input/output interface of a SHA core. b) A typical configuration of a SHA core connected to two surrounding FIFOs.

The aforementioned assumptions about the use of FIFOs as surrounding modules are very natural and easy to meet. For example, if a SHA core implemented on an FPGA communicates with an outside world using PCI, PCI-X, or PCIe interface, the implementations of these interfaces most likely already include Input and Output FIFOs, which can be directly connected to the SHA core. If a SHA core communicates with another core implemented on the same FPGA, then FIFOs are often used on the boundary between the two cores in order to accommodate for any differences between the rate of generating data by one core and the rate of accepting data by another core.

Additionally, the inputs and outputs of our proposed SHA core interface do not need to be necessarily generated/consumed by FIFOs. Any circuit that can support control signals src_ready and src_read can be used as a source of data. Any circuit that can support control signals dst_ready and dst_write can be used as a destination for data.

The exact format of an input to the SHA core, for the case of pre-padded messages, is shown in Fig. 2. Two scenarios of operation are supported. In the first scenario, the message bitlength after padding is known in advance and is smaller than 2^w. In this scenario, shown in Fig. 2a, the first word of input represents message length after padding, expressed in bits. This word has the least significant bit, representing a flag called last, set to one. This word is followed by the message length before padding. This value is required by several SHA-3 algorithms using internal counters (such as BLAKE, ECHO, Shavite-3, and Skein), even if padding is done outside of the SHA core. These two control words are followed by all words of the message.

The second format, shown in Fig. 2b, is used when either message length is not known in advance, or it is greater than 2^w. In this case, the message is processed in segments of data denoted as seg_0, seg_1,. . . ,seg_n-1. For the ease of processing data by the hash core, the size of the segments, from seg_0 to seg_n-2 is required to be always an integer multiple of the block size b, and thus also of the word size w. The least significant bit of the segment length expressed in bits is thus naturally zero, and this bit, treated as a flag called last, can be used to differentiate between the last segment and all previous segments of the message.

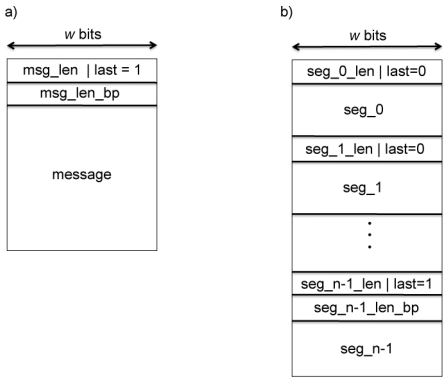

Fig. 2. Format of input data for two different operation scenarios: a) with message bitlength known in advance, and b) with message bitlength unknown in advance. Notation: msg_len – message length after padding, msg_len_bp – message length before padding, seg_i_len – segment i length after padding, seg_i_len_bp – segment i length before padding, last – a one-bit flag denoting the last segment of the message (or one-segment message), "|" – bitwise OR.

The last segment before padding can be of arbitrary length $< 2^w$. Scenario a) is a special case of scenario b). In case the SHA core supports padding, the protocol can be even simpler, as explained in [14].

5 Optimization Target and Design Methodology

Our study is performed using the following assumptions. Only the SHA-3 candidate variants with a 256-bit output are compared in this paper. Padding is assumed to be done outside of the hash cores (e.g., in software). All investigated hash functions have very similar padding schemes, which would lead to similar absolute area overhead if implemented as a part of the hardware core.

Only the primary mode of operation is supported for all functions. Special modes, such as tree hashing or MAC mode are not implemented. The salt values are fixed to all zeros in all SHA-3 candidates supporting this special input (namely BLAKE, ECHO, SHAvite-3, and Skein).

We believe that the choice of the primary optimization target is one of the most important decisions that needs to be made before the start of the comparison. The optimization target should drive the design process of every SHA-3 candidate, and it should also be used as a primary factor in ranking the obtained SHA-3 cores. The most common choices are: Maximum Throughput, Minimum Latency, Minimum Area, Throughput to Area Ratio, etc.

Our choice is the Throughput to Area Ratio, where Throughput is defined as Throughput for long messages, and Area is expressed in terms of the number of basic programmable logic blocks specific to a given FPGA family. This choice has multiple advantages. First, it is practical, as hardware cores are typically applied

in situations, where the size of the processed data is significant and the speed of processing is essential. Otherwise, the input/output latency overhead associated with using a hardware accelerator dominates the total processing time, and the cost of using dedicated hardware (FPGA) is not justified. Optimizing for the best ratio provides a good balance between the speed and the cost of the solution.

Secondly, this optimization criterion is a very reliable guide throughout the entire design process. At every junction where the decisions must be made, starting from the choice of a high-level hardware architecture down to the choice of the particular FPGA tool options, this criterion facilitates the decision process, leaving very few possible paths for further investigation.

On the contrary, optimizing for Throughput alone, leads to highly unrolled hash function architectures, in which a relatively minor improvement in speed is associated with a major increase in the circuit area. In hash function cores, latency, defined as a delay between providing an input and obtaining the corresponding output, is a function of the input size. Since various sizes may be most common in specific applications, this parameter is not a well-defined optimization target. Finally, optimizing for area leads to highly sequential designs, resembling small general-purpose microprocessors, and the final product depends highly on the maximum amount of area (e.g., a particular FPGA device) assumed to be available.

Our design of all 14 SHA-3 candidates followed an identical design methodology. Each SHA core is composed of the Datapath and the Controller. The Controller is implemented using three main Finite State Machines, working in parallel, and responsible for the Input, Main Processing, and the Output, respectively. As a result, each circuit can simultaneously perform the following three tasks: output hash value for the previous message, process a current block of data, and read the next block of data. The parameters of the interface are selected in such a way that the time necessary to process one block of data is always larger or equal to the time necessary to read the next block of data. This way, the processing of long streams of data can happen at full speed, without any visible input interface overhead. The finite state machines responsible for input and output are almost identical for all hash function candidates; the third state machine, responsible for main data processing, is based on a similar template. The similarity of all designs and reuse of common building blocks assures a high fairness of the comparison.

The design of the Datapath starts from the high level architecture. At this point, the most complex task that can be executed in an iterative fashion, with the minimum overhead associated with multiplexing inputs specific to a given iteration round, is identified. The proper choice of such a task is very important, as it determines both the number of clock cycles per block of the message and the circuit critical path (minimum clock period).

It should be stressed that the choice of the most complex task that can be executed in an iterative fashion should not follow blindly the specification of a function. In particular, quite often one round (or one step) from the description of the algorithm is not the most suitable component to be iterated in hardware.

Table 1. Main iterative tasks of the hardware architectures of SHA-3 candidates optimized for the maximum Throughput to Area ratio

Function	Main Iterative Task	Function	Main Iterative Task
BLAKE	$G_i..G_{i+3}$	**JH**	Round function R_8
BMW	entire function	**Keccak**	Round R
CubeHash	one round	**Luffa**	The Step Function, Step
ECHO	AES round/AES round/ BIG.SHIFTROWS, BIG.MIXCOLUMNS	**Shabal**	Two iterations of the main loop
Fugue	2 subrounds (ROR3, CMIX, SMIX)	**SHAvite-3**	AES round
Groestl	Modified AES round	**SIMD**	4 steps of the compression function
Hamsi	Truncated Non-Linear Permutation P	**Skein**	8 rounds of Threefish-256

Table 2. Major operations of SHA-3 candidates (other than permutations, fixed shifts and fixed rotations). mADDn denotes a multioperand addition with n operands.

Function	NTT	Linear code	S-box	GF MUL	MUL	mADD	ADD /SUB	Boolean
BLAKE						mADD3	ADD	XOR
BMW						mADD17	ADD,SUB	XOR
CubeHash							ADD	XOR
ECHO			AES 8x8	x02, x03				XOR
Fugue			AES 8x8	x04..x07				XOR
Groestl			AES 8x8	x02..x07				XOR
Hamsi		LC[128, 16,70]	Serpent 4x4					XOR
JH			Serpent 4x4	x2, x5				XOR
Keccak								NOT,AND,XOR
Luffa			4x4	x2				XOR
Shabal					x3, x5		ADD,SUB	NOT,AND,XOR
SHAvite-3			AES 8x8	x02, x03				NOT,XOR
SIMD	NTT_{128}				x185, x233	mADD3	ADD	NOT,AND,OR
Skein							ADD	XOR
SHA-256						mADD5		NOT,AND,XOR

Either multiple rounds (steps) or fractions thereof may be more appropriate. In Table 1 we summarize our choices of the main iterative tasks of SHA-3 candidates. Each such task is implemented as combinational logic, surrounded by registers.

The next step is an efficient implementation of each combinational block within the DataPath. In Table 2, we summarize major operations of all SHA-3 candidates that require logic resources in hardware implementations. Fixed shifts, fixed rotations, and other more complex permutations are omitted because they appear in all candidates and require only routing resources (programmable interconnects). The most complex out of logic operations are the Number Theoretic Transform (NTT) [15] in SIMD, linear code (LC) [16] in Hamsi, and basic operations of AES (8x8 AES S-box and multiplication by a

constant in the Galois Field GF(2^8)) in ECHO, Fugue, Groestl, and SHAvite-3; and multioperand additions in BLAKE, BMW, SIMD, and SHA-256.

For each of these operations we have implemented at least two alternative architectures. NTT was optimized by using a 7-stage Fast Fourier Transform (FFT) [15]. In Hamsi, the linear code was implemented using both logic (matrix by vector multiplications in GF(4)), and using look-up tables. AES 8x8 S-boxes (SubBytes) were implemented using both look-up tables (stored in distributed memories), and using logic only (following method described in [17], Section 10.6.1.3). Multi-operand additions were implemented using the following four methods: carry save adders (CSA), tree of two operand adders, parallel counter, and a "+" in VHDL. Finally, integer multiplications by 3 and 5 in Shabal have been replaced by a fixed shift and addition.

All optimized implementations of basic operations have been applied uniformly to all SHA-3 candidates. In case the initial testing did not provide a strong indication of superiority of one of the alternative methods, the entire hash function unit was implemented using two alternative versions of the basic operation code, and the results for a version with the better throughput to area ratio have been listed in the result tables.

All VHDL codes have been thoroughly verified using a universal testbench, capable of testing an arbitrary hash function core that follows interface described in Section 4 [18]. A special padding script was developed in Perl in order to pad messages included in the Known Answer Test (KAT) files distributed as a part of each candidates submission package. An output from the script follows a similar format as its input, but includes apart from padding bits also the lengths of the message segments, defined in Section 4, and shown schematically in Fig. 2b. The generation of a large number of results was facilitated by an open source tool ATHENa (Automated Tool for Hardware EvaluatioN) [18]. This benchmarking environment was also used to optimize requested synthesis and implementation frequencies and other tool options.

6 Results

In Table 3, we summarize the major parameters of our hardware architectures for all 14 SHA-3 candidates, as well as the current standard SHA-256. Block size, b, is a feature of the algorithm, and is described in the specification of each SHA-3 candidate. The I/O Data Bus Width, w, is a feature of our interface described in Section 4. It is the size of the data buses, din and dout, used to connect the SHA core with external logic (such as Input and Output FIFOs). The parameter w has been chosen to be equal to 64, unless there was a compelling reason to make it smaller. The value of 64 was considered to be small enough so that the SHA cores fit in all investigated FPGAs (even the smallest ones) without exceeding the maximum number of user pins. At the same time, setting this value to any smaller power of two (e.g., 32) would increase the time necessary to load input data from the input FIFO and store the hash value to the output FIFO. In some cases, it would also mean that the time necessary for processing a single block

Table 3. Timing characteristics of our hardware architectures of SHA-3 candidates. Notation: T – minimum clock period in *ns* (specific for each algorithm and each FPGA device, see Table 6), N - Number of blocks of an input message after padding.

Function	**Block size,** b **[bits]**	**I/O Data Bus Width,** w **[bits]**	**Time to hash N message blocks [clock cycles]**	**Throughput [Mbit/s]**
BLAKE	512	64	2+8+20·N+4	512/(20·T)
BMW	512	64	2+⌈8/8⌉+N+⌈4/8⌉	512/T
CubeHash	256	64	2+4+16·N+160+4	256/(16·T)
ECHO	1536	64	3+24+27·N+4	1536/(27·T)
Fugue	32	32	2+N+18+8	32/T
Groestl	512	64	3+8+21·N+4	512/(21·T)
Hamsi	32	32	3+1+3·(N-1)+6+8	32/(3·T)
JH	512	64	3+8+36·N+4	512/(36·T)
Keccak	1088	64	3+17+24·N+4	1088/(24·T)
Luffa	256	64	3+4+9·N+9+4	256/(9·T)
Shabal	512	64	3+8+1+25·N+3·25+4	512/(25·T)
Shavite-3	512	64	3+8+37·N+4	512/(37·T)
SIMD	512	64	3+8+8+9·N+4	512/(9·T)
Skein	256	64	2+4+9·N+4	256/(9·T)
SHA-256	512	32	2+1+65·N+8	512/(65·T)

of data would be smaller than the time of loading the next block of data, which would decrease the overall throughput. The only exceptions are made in case of Fugue and Hamsi, which have a block size b equal to 32 bits. Additionally, in the old standard SHA-256, the input/output data bus is set naturally to 32-bits, as the message scheduling unit accepts only one word of data per clock cycle.

In case of BMW, an additional faster i/o clock was used on top of the main clock shown in Fig. 1a. This faster clock is driving input/output interfaces of the SHA core, as well as surrounding FIFOs. The ratio of the i/o clock frequency to the main clock frequency was selected to be 8, so the entire block of message (512 bits) can be loaded in a single clock cycle of the main clock (8 cycles of the fast i/o clock).

The forth column of Table 3 contains the detailed formulas for the number of clock cycles necessary to hash N blocks of the message after padding. The formulas include the time necessary to load the message length, load input data from the FIFO, perform all necessary initializations, perform main processing, perform all required finalizations, and then send the result to the output FIFO. Finally, the last column contains the formula for the circuit throughput for long messages as defined by equation (1).

In Table 4, we list absolute values of the major parameters describing our implementations for one particular FPGA family, Xilinx Virtex 5. According to this table the highest throughput to area ratio is achieved by Keccak, Luffa, Groestl, and CubeHash. The highest absolute throughput is accomplished by

Table 4. Major performance measures of SHA-3 candidates when implemented in Xilinx Virtex 5 FPGAs. Notation: T_{empty} – Time to hash an empty message (after this message is padded in software), T_{100B} – Time to hash a 100-byte message (after this message is padded in software).

Function	Clk Freq [MHz]	Area [CLB slices]	Throughput [Mbits/s]	Throughput to Area Ratio	T_{empty} [ns]	T_{100B} [ns]
BLAKE	102.0	1851	2610.6	1.4	333.4	529.5
BMW	10.9	4400	5576.7	1.3	459.1	550.9
CubeHash	199.4	730	3189.8	4.4	922.9	1163.7
ECHO	178.1	6453	10133.4	1.6	308.8	308.8
Fugue	98.5	956	3151.2	3.3	304.6	558.5
Groestl	355.9	1884	8676.5	4.6	101.2	160.2
Hamsi	248.1	946	2646.2	2.8	96.7	399.1
JH	282.2	1275	4013.5	3.1	180.7	308.3
Keccak	238.4	1229	10806.5	8.8	201.4	201.4
Luffa	281.5	1154	8008.0	6.9	103.0	198.9
Shabal	128.1	1266	2624.0	2.1	905.4	1100.5
SHAvite-3	208.6	1130	2885.9	2.6	249.3	426.8
SIMD	40.9	9288	2325.9	0.3	635.9	1076.2
Skein	49.8	1312	1416.1	1.1	381.6	924.0
SHA-256	207.0	433	1630.5	3.8	352.7	1294.7

Table 5. Results for the reference design of SHA-256

	Spartan 3	Virtex 4	Virtex 5	Cyclone II	Cyclone III	Stratix II	Stratix III
Max. Clk Freq. [MHz]	90.8	183.0	207.0	111.0	126.9	158.1	212.8
Throughput [Mbit/s]	715.6	1441.6	1630.5	874.7	999.3	1245.2	1676.3
Area	838	838	433	1655	1653	973	963
Throughput to Area Ratio	0.85	1.72	3.77	0.53	0.60	1.28	1.74

Table 6. Clock frequencies of all SHA-3 candidates and SHA-256 expressed in MHz (post placing and routing)

Function	Spartan 3	Virtex 4	Virtex 5	Cyclone II	Cyclone III	Stratix II	Stratix III
BLAKE	41.87	79.82	101.98	52.40	52.37	85.77	109.21
BMW	4.19	12.37	10.89	7.69	8.41	13.45	16.45
CubeHash	84.70	187.58	199.36	115.67	133.83	179.40	237.64
ECHO	52.10	131.90	176.24	N/A	105.70	109.50	164.20
Fugue	39.67	72.86	98.47	53.25	60.71	83.75	123.64
Groestl	105.72	234.74	355.87	132.00	148.46	216.73	270.27
Hamsi	90.37	200.88	248.08	148.83	183.52	193.87	294.81
JH	119.36	221.58	282.20	173.43	215.89	267.45	364.30
Keccak	96.32	202.47	238.38	165.07	174.28	198.65	296.30
Luffa	129.84	260.28	281.53	171.64	173.43	219.88	307.31
Shabal	30.99	114.03	128.12	69.57	68.76	105.40	126.87
SHAvite-3	84.60	152.23	208.55	95.40	114.40	170.00	255.00
SIMD	17.20	29.25	40.89	21.66	23.97	37.07	47.40
Skein	18.22	38.16	49.79	22.30	25.14	38.89	52.29
SHA-512	90.84	183.02	207.00	111.04	126.86	158.08	212.81

Table 7. Throughput of all SHA-3 candidates normalized to the throughput of SHA-256. N/A means that the design did not fit within any device of a given family.

Function	Spartan 3	Virtex 4	Virtex 5	Cyclone II	Cyclone III	Stratix II	Stratix III	Overall
Keccak	6.10	6.37	6.63	8.56	7.91	7.23	8.01	7.21
ECHO	4.14	5.21	6.15	N/A	6.02	5.00	5.57	5.30
Luffa	5.16	5.14	4.91	5.58	4.94	5.02	5.21	5.13
Groestl	3.60	3.97	5.32	3.68	3.62	4.24	3.93	4.02
BMW	3.00	4.39	3.42	4.50	4.31	5.53	5.02	4.48
JH	2.37	2.19	2.46	2.82	3.07	3.05	3.09	2.70
CubeHash	1.89	2.08	1.96	2.12	2.14	2.31	2.27	2.10
Fugue	1.77	1.62	1.93	1.95	1.94	2.15	2.36	1.95
SHAvite-3	1.64	1.46	1.77	1.51	1.58	1.89	2.11	1.70
Hamsi	1.35	1.49	1.62	1.82	1.96	1.66	1.88	1.67
BLAKE	1.50	1.42	1.60	1.53	1.34	1.76	1.67	1.54
Shabal	0.89	1.62	1.61	1.63	1.41	1.73	1.55	1.46
SIMD	1.37	1.15	1.43	1.41	1.36	1.69	1.61	1.38
Skein	0.72	0.75	0.87	0.73	0.72	0.89	0.89	0.79

Table 8. Area (utilization of programmable logic blocks) of all SHA-3 candidates normalized to the area of SHA-256

Function	Spartan 3	Virtex 4	Virtex 5	Cyclone II	Cyclone III	Stratix II	Stratix III	Overall
CubeHash	1.81	1.81	1.69	1.87	1.88	1.99	2.01	1.86
Hamsi	2.17	2.16	2.18	1.92	1.94	2.40	2.41	2.16
BLAKE	4.96	4.87	4.27	2.17	2.16	2.00	2.04	2.96
Luffa	3.28	3.29	2.67	2.74	2.77	3.40	3.43	3.07
Skein	3.41	3.45	3.03	3.28	3.34	3.68	3.74	3.41
Shabal	3.75	3.84	2.92	3.67	3.68	3.90	3.74	3.63
Keccak	3.97	3.99	2.84	3.77	3.62	4.20	4.63	3.82
JH	4.84	4.78	2.94	4.37	4.31	3.18	3.24	3.88
SHAvite-3	4.91	4.91	2.61	5.68	5.64	2.57	2.59	3.89
Fugue	4.26	4.44	2.21	5.85	5.87	3.70	3.73	4.11
Groestl	15.96	16.01	4.35	4.60	4.50	3.21	3.22	5.86
BMW	12.07	13.45	10.16	12.00	12.02	12.99	13.12	12.24
SIMD	20.97	19.99	21.45	18.53	18.57	23.03	23.24	20.39
ECHO	30.87	28.48	14.90	N/A	39.77	22.29	22.52	25.29

Table 9. Throughput to Area Ratio of all SHA-3 candidates normalized to the throughput to area ratio of SHA-256

Function	Spartan 3	Virtex 4	Virtex 5	Cyclone II	Cyclone III	Stratix II	Stratix III	Overall
Keccak	1.54	1.60	2.34	2.27	2.18	1.72	1.73	1.89
Luffa	1.57	1.56	1.84	2.04	1.78	1.48	1.52	1.67
CubeHash	1.04	1.15	1.16	1.13	1.14	1.16	1.13	1.13
Hamsi	0.62	0.69	0.74	0.94	1.01	0.69	0.78	0.77
JH	0.49	0.46	0.84	0.65	0.71	0.96	0.95	0.70
Groestl	0.23	0.25	1.22	0.80	0.81	1.32	1.22	0.69
BLAKE	0.30	0.29	0.37	0.71	0.62	0.88	0.82	0.52
Fugue	0.42	0.36	0.88	0.33	0.33	0.58	0.63	0.47
SHAvite-3	0.33	0.30	0.68	0.27	0.28	0.74	0.81	0.44
Shabal	0.24	0.42	0.55	0.44	0.38	0.44	0.41	0.40
BMW	0.25	0.33	0.34	0.38	0.36	0.43	0.38	0.37
Skein	0.21	0.22	0.29	0.22	0.21	0.24	0.24	0.23
ECHO	0.13	0.18	0.41	N/A	0.15	0.22	0.25	0.21
SIMD	0.07	0.06	0.07	0.08	0.07	0.07	0.07	0.07

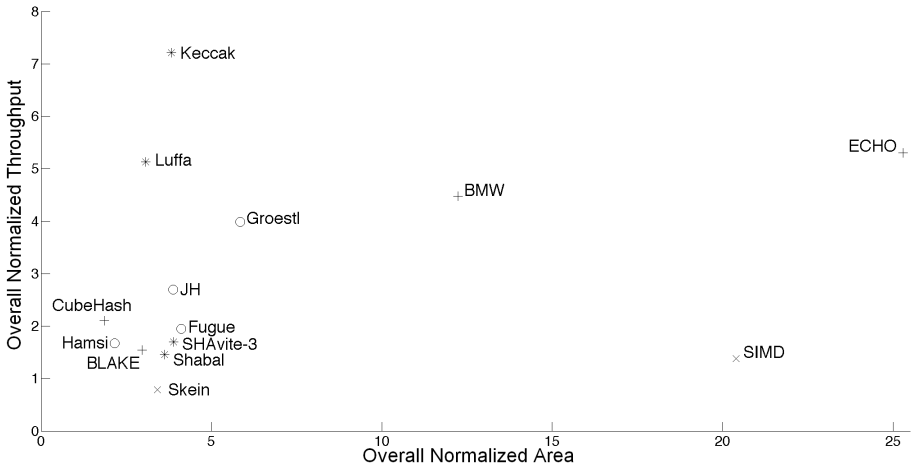

Fig. 3. Relative performance of all Round 2 SHA-3 Candidates in terms of the overall normalized throughput and the overall normalized area (with SHA-256 used as a reference point)

Keccak, ECHO, Groestl, Luffa, and BMW. The smallest hashing time for an empty message is achieved by Hamsi, Groestl, Luffa, and JH. For a 100 byte message, the list of the first four candidates changes to Groestl, Luffa, Keccak, and JH. As one can see, the execution time for small messages is not strongly correlated with the throughput for long messages, and therefore it must be treated as a separate evaluation criterion (as discussed in Section 3).

In Table 5, we summarize the absolute results obtained for our implementation of the current standard SHA-256. The results are repeated for all seven FPGA families used in our study. As hardware architecture, we have selected the architecture by Chaves et al., presented at CHES 2006 [19]. This architecture has been specifically optimized for the maximum throughput to area ratio [19][20], and is considered one of the best known SHA-2 architectures of this type.

In Table 6, the maximum clock frequencies are listed for each pair: hash algorithm–FPGA family. These frequencies can be used together with the formulas provided in Table 3, in order to compute the exact execution times of each algorithm (depending on the number of the message blocks, N) and the values of the throughputs for long messages.

In the following analysis, the absolute values of the three major performance measures: throughput, area, and the throughput to area ratio, for all SHA-3 candidates, have been normalized by dividing them by the corresponding values for the reference implementation of SHA-256. The corresponding ratios, referred to as normalized throughput, normalized area, and normalized throughput to area ratios are summarized in Tables 7, 8, and 9. The Overall column represents the geometric mean of all normalized results available for a given algorithm. The candidate algorithms are ranked based on the value of this Overall metric, representing the performance for a wide range of different FPGA families.

Interestingly, based on Table 9, only three candidates, Keccak, Luffa, and CubeHash outperform SHA-256 in terms of the throughput to area ratio. The additional four candidates, Hamsi, JH, Groestl, and BLAKE, have the overall normalized ratio higher than 0.5.

In Fig. 3, we present a two dimensional diagram, with Normalized Area on the X-axis and Normalized Throughput on the Y-axis. The algorithms seem to fall into several major groups. Group with the high normalized throughput (>5), medium normalized area (<4), and the high normalized throughput to area ratio (>1.5), include Keccak and Luffa. Groestl, BMW, and ECHO, have all high normalized throughput (>4), but their normalized area varies significantly from about 6 in case of Groestl, through 12 for BMW, up to over 25 in case of ECHO. SIMD is both relatively slow (less then 2 times faster than SHA-256) and big (more than 20 times bigger than SHA-256). The last group includes 8 candidates covering the range of the normalized throughputs from 0.8 to 2.7, and the normalized areas from 1.9 to 4.1.

7 Conclusions and Future Work

Our evaluation methodology, applied to 14 Round 2 SHA-3 candidates, has demonstrated large differences among competing candidates. The ratio of the best result to the worst result was equal to about 9 in terms of the throughput (Keccak vs. Skein), over 13 times in terms of area (CubeHash vs. ECHO), and about 27 in terms of our primary optimization target, the throughput to area ratio (Keccak vs. SIMD). Only three candidates, Keccak, Luffa, and CubeHash, have demonstrated the throughput to area ratio better than the current standard SHA-256. Out of these three algorithms, Keccak and Luffa have also demonstrated very high throughputs, while CubeHash outperformed other candidates in terms of minimum area. All candidates except Skein outperform SHA-256 in terms of the throughput, but at the same time none of them matches SHA-256 in terms of the area.

Future work will include the evaluation of the remaining variants of SHA-3 candidates (such as variants with 224, 384, and 512 bit outputs, and an all-in-one architecture). The uniform padding units will be added to each SHA core, and their cost estimated. We will also investigate the influence of synthesis tools from different vendors (e.g., Synplify Pro from Synopsys). The evaluation may be also extended to the cases of hardware architectures optimized for the minimum area (cost), maximum throughput (speed), or minimum power consumption. Each algorithm will be also evaluated in terms of its suitability for implementation using dedicated FPGA resources, such embedded memories, dedicated multipliers, and DSP units. Finally, an extension of our methodology to the standard-cell ASIC technology will be investigated.

Acknowledgments. The authors would like to acknowledge all students from the Fall 2009 edition of the George Mason University course entitled "Digital System Design with VHDL," for conducting initial exploration of the design space of all SHA-3 candidates.

References

1. Nechvatal, J., et al.: Report on the Development of the Advanced Encryption Standard (AES), http://csrc.nist.gov/archive/aes/round2/r2report.pdf
2. NESSIE, https://www.cosic.esat.kuleuven.be/nessie/
3. eSTREAM, http://www.ecrypt.eu.org/stream/
4. Gaj, K., Chodowiec, P.: Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard Using Field Programmable Gate Arrays. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 84–99. Springer, Heidelberg (2001)
5. Hwang, D., Chaney, M., Karanam, S., Ton, N., Gaj, K.: Comparison of FPGA-targeted Hardware Implementations of eSTREAM Stream Cipher Candidates. In: State of the Art of Stream Ciphers Workshop, SASC 2008, February, pp. 151–162 (2008)
6. Good, T., Benaissa, M.: Hardware Performance of eStream Phase-III Stream Cipher Candidates. In: State of the Art of Stream Ciphers Workshop, SASC 2008, February 2008, pp. 163–173 (2008)
7. SHA-3 Contest, http://csrc.nist.gov/groups/ST/hash/sha-3/index.html
8. SHA-3 Zoo, http://ehash.iaik.tugraz.at/wiki/TheSHA-3Zoo
9. Drimer, S.: Security for Volatile FPGAs. ch. 5: The Meaning and Reproducibility of FPGA Results. Ph.D. Dissertation, University of Cambridge, Computer Laboratory, uCAM-CL-TR-763 (Nov 2009)
10. SHA-3 Hardware Implementations, http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations
11. Tilich, S., et al.: High-speed Hardware Implementations of Blake, Blue Midnight Wish, Cubehash, ECHO, Fugue, Groestl, Hamsi, JH, Keccak, Luffa, Shabal, Shavite-3, SIMD, and Skein. Cryptology, ePrint Archive, Report 2009/510 (2009)
12. Kobayashi, K., et al.: Evaluation of Hardware Performance for the SHA-3 Candidates Using SASEBO-GII. Cryptology, ePrint Archive, Report 2010/010 (2010)
13. ECRYPT Benchmarking of Cryptographic Systems, http://bench.cr.yp.to
14. CERG GMU Group: Hardware Interface of a Secure Hash Algorithm (SHA), http://cryptography.gmu.edu/athena/index.php?id=interfaces
15. Meyer-Baese, U.: Digital Signal Processing with Field Programmable Gate Arrays, ch. 6, 7, 3rd edn., pp. 343–475. Springer, Heidelberg (2007)
16. van Lint, J.H.: Introduction to Coding Theory, 2nd edn. Springer, Heidelberg (1992)
17. Gaj, K., Chodowiec, P.: FPGA and ASIC Implementations of AES. In: Cryptographic Engineering, ch. 10, pp. 235–294. Springer, Heidelberg (2009)
18. ATHENa Project Website, http://cryptography.gmu.edu/athena
19. Chaves, R., Kuzmanov, G., Sousa, L., Vassiliadis, S.: Improving SHA-2 Hardware Implementations. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 298–310. Springer, Heidelberg (2006)
20. Chaves, R., Kuzmanov, G., Sousa, L., Vassiliadis, S.: Cost Efficient SHA Hardware Accelerators. IEEE Trans. Very Large Scale Integration Systems 16, 999–1008 (2008)