Comprehensive Evaluation of High-Speed and Medium-Speed Implementations of Five SHA-3 Finalists Using Xilinx and Altera FPGAs

Kris Gaj, Ekawat Homsirikamol, Marcin Rogawski, Rabia Shahid, and Malik Umar Sharif
George Mason University

{kgaj, ehomsiri, mrogawsk, rshahid, msharif2}@gmu.edu

Abstract. In this paper we present a comprehensive comparison of all Round 3 SHA-3 candidates and the current standard SHA-2 from the point of view of hardware performance in modern FPGAs. Each algorithm is implemented using multiple architectures based on the concepts of iteration, folding, unrolling, pipelining, and circuit replication. Trade-offs between speed and area are investigated, and the best architecture from the point of view of the throughput to area ratio is identified. Finally, all algorithms are ranked based on their overall performance in FPGAs. The characteristic features of each algorithm important from the point of view of its implementation in hardware are identified.

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

1. Introduction

Performance in hardware is one of the major criteria used in the SHA-3 competition [1]. Typically, this performance is evaluated using two major technologies: Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). Comparison using FPGAs offers several important advantages, such as short development time, accurate post place & route results, existence of tools for optimum choice of program options and automated collection of a large number of results [2,16], and relatively small number of vendors and device families that dominate the market. As a result, our FPGA performance evaluation covers significantly broader design space than any ASIC comparison we are aware of. In particular, in this paper, each of the SHA-3 finalists is implemented in both basic *variants, with a 256-bit and a 512-bit output*, and each variant is implemented using from *5 to 10 different hardware architectures* based on the concepts of iteration, folding, unrolling, pipelining, and circuit replication. Each architecture is equipped with a *realistic FIFO-based interface* with a modest pin requirement, and the capability for simultaneous processing of the current message block, reading the next message block, and writing the previously computed hash value to the output FIFO [15,23,24]. Unlike any ASIC implementations, and majority of earlier reported FPGA implementations, our SHA-3 candidate cores are equipped with full *padding units*, capable of processing any messages ending on a boundary of a byte. All VHDL source codes have been developed by *two primary designers*, closely collaborating with each other, which substantially minimizes the potential influence of different designer skills. Majority of *source codes and the corresponding block diagrams* have been published on the web and made available for public scrutiny [5]. All cores have been implemented and characterized using *four modern high-performance FPGA families* from two major vendors, Xilinx and Altera. All implementation results have been optimized and generated using ATHENa (Automated Tool for Hardware Evaluation) [2,16]. The details of all 600+ results are available in the ATHENa database [3], where they can be interactively accessed, reviewed, ranked, searched for, and compared to one another. For each set of results, ATHENa database holds also a set of replication scripts and configuration files that can be used by a third party to efficiently reproduce all results without using ATHENa. Finally, we also demonstrate in this paper that selected FPGA results show very good correlation with the corresponding ASIC results [20] obtained using a typical standard-cell library based on the similar 65nm CMOS technology.

2. Previous Work

Previous results on comparison of Round 2 SHA-3 candidates in hardware are summarized in [7,3,4]. These results are classified into four major categories, based on the technology (FPGA vs. ASIC), and the optimization target (High-Speed vs. Low-Area). The previous results most relevant to the subject of this paper belong to the category of High-Speed Implementations in FPGAs. The most comprehensive results belonging to this category were reported by Baldwin et al. [10,11], Gaj et al. [15], Homsirikamol et al. [23], Matsuo et al. [34], and Knežević et al. [32]. All these groups have published results for all 14 Round 2 candidates. Majority of published results concern 256-bit variants of the candidates, implemented using Xilinx Virtex 5 FPGAs. In [23], results for 256-bit and 512-bit variants of all algorithms, implemented using 10 FPGA families from Xilinx and Altera are discussed. Additionally, pipelined implementations of three Round 2 SHA-3 candidates have been investigated in [8]. In our earlier paper, published at CHES 2011 [24], we investigated the throughput vs. area trade-offs in implementations of SHA-2 and five SHA-3 finalists. In this paper, we present results obtained by extending each architecture with a padding unit, and optimizing selected pipelined implementations of the SHA-3 candidates. A similar study, limited to most efficient high-speed single-message architectures has been reported in [33]. Additionally, two unified high-speed hardware architectures of AES and Groestl have been reported in [25,36]. The influence of system parameters on the performance of selected SHA-3 candidates have been investigated in [14].

Several comprehensive comparisons of low-area implementations of Round 3 SHA-3 candidates in FPGAs have been presented in [26,27,29,30,31]. Additional results for BLAKE and Skein have been discussed in [13,9]. The most comprehensive studies of ASIC implementations of the Round 3 SHA-3 candidates have been described in [18,19,20] and documented in [4]. These studies follow previous investigation of Round 2 SHA-3 candidates described in [17,21,22,38].

All results obtained based on the Round 2 specifications of SHA-3 candidates carry without any changes for Keccak and Skein. The specifications of BLAKE, Groestl, and JH have been tweaked at the start of Round 3, in January 2011. The throughput of the Round 3 BLAKE and JH can be calculated based on the results from Round 2 by decreasing it by a factor proportional to the increase in the number of rounds. The area of these implementations will remain practically the same. The change in the throughput and area of Groestl is much more difficult to approximate, as demonstrated in [35].

3. Performance Metrics

Three major performance metrics used in our study are throughput, area, and throughput to area ratio. Throughput presented in all tables and graphs in this paper (except Section 11) is calculated as the throughput for long messages. The formulas describing the adjustments to the values of the throughput required when processing short messages are presented in Section 11.
The resource utilization in FPGAs is a vector, with coordinates specific to the given FPGA family, e.g.

$$\textit{Resource Utilization}_{\text{Virtex 5}} = (\#\textit{CLB slices}, \#\textit{BRAMs}, \#\textit{DSPs}) \quad (1)$$

$$\textit{Resource Utilization}_{\text{Stratix III}} = (\#\textit{ALUTs}, \#\textit{memory\_bits}, \#\textit{DSPs}). \quad (2)$$

In these formulas: *#CLB_slices* is the number of Configurable Logic Block slices, BRAM stands for Block RAM, DSP is a Digital Signal Processing unit, #ALUTs represents the number of Adaptive Look-Up Tables, and *#mem-bits* is the number of bits stored in dedicated Altera FPGA memories.

Taking into account that vectors cannot be easily compared to one another, 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 *#CLB_slices* for Xilinx Virtex 5 and Virtex 6, *#ALUTs* in Stratix III and Stratix IV.

4. Investigated Hardware Architectures

Investigated architectures are described in more detail in our earlier paper presented at CHES 2011 [24]. Additionally, full VHDL source codes and corresponding hierarchical block diagrams of majority of these architectures have been published at [5]. Below, we present only a short summary of major features of the known-to-date high-speed and medium-speed hardware architectures of SHA-3 finalists.

A starting point for our exploration is the basic iterative architecture, shown in Fig. 1a. This architecture is the most efficient (in terms of the throughput to area ratio) non-pipelined architecture of SHA-2, Groestl, JH, and Keccak.

In order to reduce area necessary to implement a given hash algorithm, at the cost of decreasing its throughput, folded architectures can be used. These architectures can be employed only if a round of a hash function has a symmetric structure with respect to either horizontal or vertical axis (with input to a round shown at the top and output shown at the bottom of the round block), as illustrated in Fig. 1.

In Fig. 1b, horizontal folding by a factor of two is demonstrated. We will denote this architecture by /2(h). In this architecture, a half of a round is implemented as combinational logic, and the entire round is executed using two clock cycles. As a result, the block processing time (and thus also throughput) stays approximately the same, and area decreases. These dependencies lead to the overall increase of the Throughput to Area ratio. In general, folding by a factor of k might be possible, and the corresponding architecture will be denoted by /k(h). Among the five finalists, the only candidate that can benefit substantially from horizontal folding is BLAKE. The round of BLAKE consists of two horizontal layers of identical G functions, separated only by a permutation. By implementing only one layer in combinational logic, horizontal folding by a factor of two can be easily achieved. Additionally, each G function has a very symmetric structure along the horizontal axis, and can be easily folded horizontally by a factor of 2. As a result, a folding factor of 4 can be achieved for the entire round. Other SHA-3 finalists do not demonstrate any similar symmetry.

In Fig. 1c, we demonstrate vertical folding by a factor of 2. We will denote this folding by /2(v). In this architecture, the datapath width is reduced by a factor of two. As a result two clock cycles are required to complete a round. In the first clock cycle, only bits of the internal state affecting the first half of the round output are provided to the input of R/2. In the second clock cycle, the remaining bits of the internal state are processed. The first output is stored in an auxiliary register of the size of s/2 bits. This output is concatenated with the output from the second iteration to form a new internal state. The clock period of this architecture is approximately equal to the clock period of the basic iterative architecture. As a result, the block processing time increases approximately by a factor of two compared to the basic iterative architecture. The area reduction is also smaller than in case of horizontal folding, because of the need for an extra s/2-bit register and a multiplexer. As a result the throughput to area ratio is likely to go

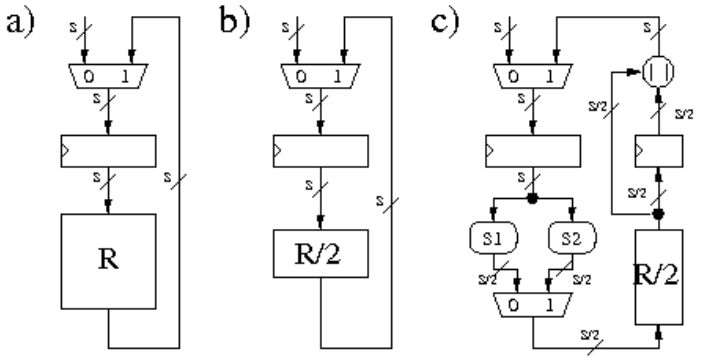


Fig. 1. Three hardware architectures of a hash function: a) basic iterative, x1, b) folded horizontally by a factor of 2, /2(h), c) folded vertically by a factor of 2, /2(v).
Notation: R – round, R/2 – half-round, S1, S2 – selection functions, s – state size in bits.

down. In general, vertical folding by a factor of k might be possible, and the corresponding architecture will be denoted by /k(v). Out of five final SHA-3 candidates, BLAKE and Groestl are most suitable for vertical folding. JH can be folded, but the gain in area is not expected to be substantial, because the round of JH is very simple, and does not dominate the total area of the circuit. For Skein and Keccak, the internal round symmetry, necessary for implementation of vertical folding, is limited.

For vertical folding with the factor $k \geq 4$ it is beneficial to store the internal state in memory, rather than in registers. The obtained throughput to area ratio can be substantially increased as a result of this change in the storage element. We will denote the obtained architectures as /k(v)-m.

In order to increase circuit throughput for processing of a single message, unrolling can be used. In Fig. 2a, architecture with unrolling by a factor of two is demonstrated. The combinational logic of a round is replicated, so now two rounds are performed per clock cycle. Since the total number of clock cycles is reduced approximately by a factor of two, and the clock period increases by a factor less than two (due to optimizations on the boundaries of two rounds, and the smaller relative contributions of the multiplexer delay, the register delay, and the register setup time), the total throughput increases. Unfortunately, at the same time, the area of the circuit is likely to increase by a factor close to the unrolling factor. As a result, in most cases, the throughput to area ratio decreases substantially compared to the basic iterative architecture. As such, architectures with unrolling are typically used only when throughput for a single long message is of the utmost concern, and area is abundant.

Nevertheless, there are exceptions to this rule. Unrolling can improve the throughput to area ratio when rounds used by an algorithm in subsequent iterations are not the same, or there is a potential for substantial delay reductions on the boundary between consecutive rounds. Among the five final SHA-3 finalists, this situation happens only for Skein. As a result, the throughput to area ratio of Skein becomes optimum for one of the unrolled architectures.

Further increase in the throughput and the throughput to area ratio of SHA-3 candidates is possible by using pipelined architectures. In order to take full advantage of the pipelined architectures multiple messages must be processed at the same time. Luckily, this is exactly the situation that appears most often in practical applications of hash functions. For example, in the most widespread Internet security protocols, such as IPSec, SSL, and WLAN (802.11), the inputs to a hash unit are packets. The maximum size of a packet for Internet is limited by so called Maximum Transmission Unit (MTU). The typical size of MTU for Ethernet based networks is 1500 bytes. The Maximum Transmission Unit for the Internet IPv4 path is even smaller, and set to 576 bytes. As a result, in a typical internet node, up to 80% of packets processed have the size of 576 bytes or less, and 100% of packets have sizes equal or smaller than 1500 bytes. Such small sizes of packets mean that hundreds of packets could be easily buffered in the processing nodes, in the form of packet queues, without introducing any significant latency to the total packet travel time from the source to destination. Therefore, the capabilities for parallel processing (including pipelining) seem to be primarily limited by the total area of the hash unit, and not by the number of messages available in parallel. In this paper, we will assume that the number of messages available in parallel is large (at least 10), and we will look at the combined throughput for all available streams of data.

The easiest way to implement pipelining in hash functions is to first unroll, and then introduce pipeline registers between adjacent rounds. The simplest case is the architecture that is two times unrolled, and has two pipeline stages, as shown in Fig. 2b. We will denote this architecture as x2-PPL2. The throughput to area ratio remains roughly the same, and may be either larger or smaller than in the basic iterative architecture, depending on a particular algorithm. The more challenging way of using pipelining is to introduce pipeline registers inside of a hash function round, as shown in Fig. 2c. The improvement in throughput compared to the basic iterative architecture is then equal (either exactly or at least approximately) to the ratio of the new clock frequency to the original clock frequency. Since the critical path is reduced, the increase in throughput is guaranteed, but its level depends on how well the critical path has been divided by pipeline registers into shorter paths with approximately equal delays. At the same time, the area of the circuit increases by the area of pipeline registers, plus any logic required for

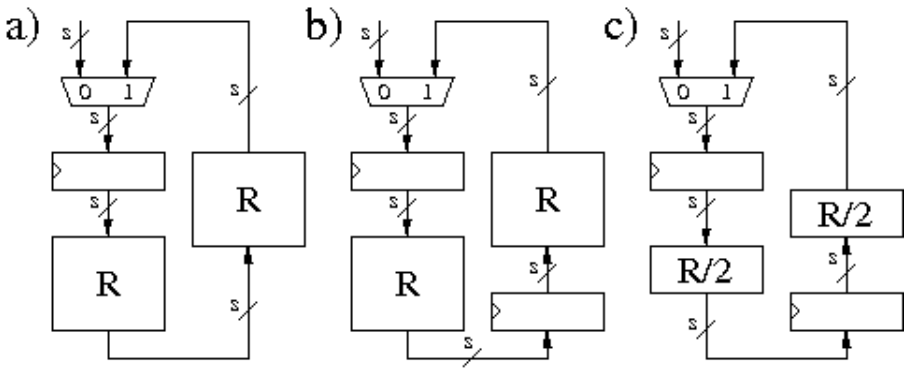


Fig. 2. Three hardware architectures of a hash function a) unrolled by a factor of 2, x2, b) unrolled by a factor of 2 with 2 pipeline stages, x2-PPL2, c) basic iterative with 2 pipeline stages, x1-PPL2. Notation: R – round, R/2 – half-round, s – state size in bits.

simultaneous processing of multiple streams of data. The throughput to area ratio may increase, but the improvement is not guaranteed for all algorithms, and all FPGA families, and may be small or negative in case the basic iterative architecture operates already at the clock frequency close to the maximum clock frequency supported by a given FPGA family.

The final alternative is architecture obtained by replicating the entire circuit multiple number of times. We call this architecture a multi-unit architecture, and we denote it by MUn, where n denotes the number of repetitions of the hash core. Obviously, in this architecture, throughput and area increase proportionally, and n messages are required to be present concurrently in order to take full advantage of the potential increase in throughput. A typical design approach would be to first find an architecture with the best throughput to area ratio, and then replicate it as many times as necessary in order to reach the desired throughput.

The formulas for the throughput of all aforementioned architectures, assuming processing of long messages, are summarized in Table 1.

Table 1: Formulas for the Throughput, Tp, of all investigated architectures. Notation: b – block size, r – number of rounds, f – number of clock cycles required to finalize computations for a block (f = 0 for Keccak and Groestl (P+Q), f=1 for all remaining algorithms), k – folding factor or unrolling factor, n – number of pipeline stages, T – clock period.

Notation	Architecture	Throughput
x1	Basic iterative	$Tp = b/((r + f) \cdot T)$
/k	Folded by a factor of k	$Tp = b/((k \cdot r + f) \cdot T)$
xk	Unrolled by a factor of k	$Tp = b/((r/k + f) \cdot T)$
x1-PPLn	Basic iterative with n pipeline stages	$Tp = n \cdot b/((n \cdot r + f) \cdot T)$
/k-PPLn	Folded by a factor of k with n pipeline stages	$Tp = n \cdot b/((n \cdot k \cdot r + f) \cdot T)$
xk-PPLn	Unrolled by a factor of k with n pipeline stages	$Tp^* = n \cdot b/((n \cdot r/k + f) \cdot T)$
MUn	Multi-unit architecture based on n repetitions of the basic iterative architecture	$Tp = n \cdot b/((r + f) \cdot T)$

- for Skein a modified formula, $Tp = n \cdot b/n \cdot (r/k + f) \cdot T$, applies

5. Design Methodology and Design Environment

Our designs for the basic iterative, folded, and unrolled architectures use the interface and the communication protocol proposed in [15,23]. Our designs for the pipelined architectures, use the interface and surrounding logic shown in Fig. 3. Input FIFOs serve as packet queues. Each FIFO communicates with the corresponding Padding Unit and the associated Finite State Machine 1 (FSM1). FSM1 is responsible for loading the next block of data and padding the last block of a message, if needed (possibly in parallel with the Datapath processing the previous block under the control of FSM2). Outputs corresponding to four independent packets are first stored in the corresponding Parallel-In Serial-Out Units, and then multiplexed to the output FIFO.

All architectures have been modeled in VHDL-93. All VHDL codes have been thoroughly verified using a universal testbench, capable of testing an arbitrary hash function core. 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 candidate's submission package.

For synthesis and implementation, we have used tools developed by FPGA vendors themselves:

- for Xilinx: Xilinx ISE Design Suite v. 13.1, including Xilinx XST,
- for Altera: Quartus II v. 11.1 Subscription Edition Software.

The generation of a large number of results and optimization of tool options was facilitated by an open source benchmarking environment, called ATHENa (Automated Tool for Hardware EvaluatioN) [2,16].

All result graphs included in this paper use color codes introduced by Bernstein and Lange in [6,12].

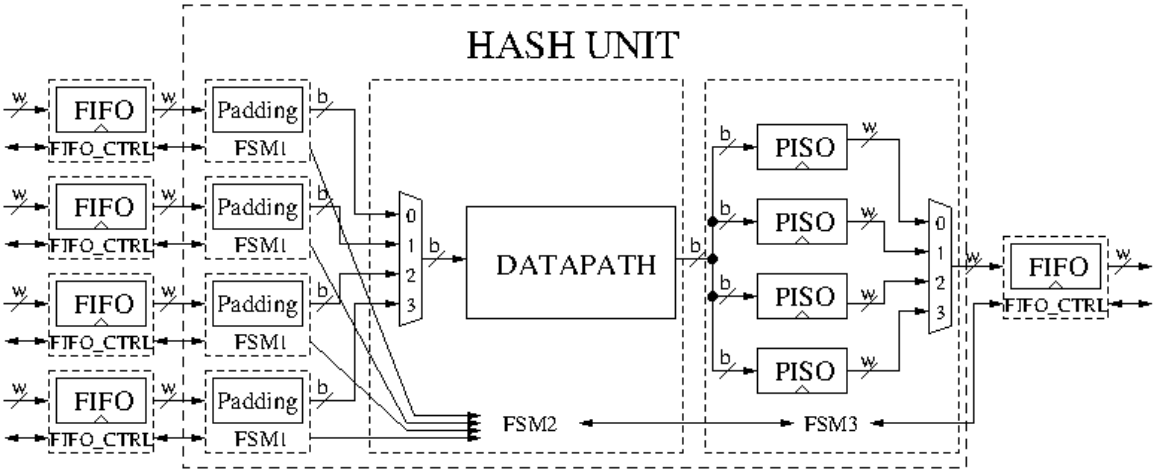


Fig. 3: The interface, high-level block diagram, and surrounding logic of the Hash Unit for the pipelined architecture with four pipeline stages. Notation: Padding – Padding Unit, including SIPO (Serial-In Parallel-Out unit), PISO – Parallel-In Serial-Out unit, w – input/output bus width, w=64 for all investigated algorithms, except SHA-2-256, where w=32.

6. Results with Padding

The results of our implementations with padding units are summarized in Figs. 4-9, and in Tables 2 and 3. In Figs. 4 and 5, we present the detailed throughput vs. area graphs for all implemented architectures of the 256-bit variants of six investigated algorithms in Xilinx Virtex 5 and Altera Stratix III, respectively.

For BLAKE (see Figs. 4a and 5a), the best architecture overall is x1-PPL4, i.e., basic architecture with four pipeline stages. The good performance of this architecture is associated with the symmetric structures of the basic round and each individual G function, which make it easy to divide the datapath into four well-balanced pipeline stages (see Figs. 14 and 15). The best non-pipelined architectures are:

- for Virtex 5: /4(h)/4(v)-m, i.e., architecture with horizontal folding by a factor of 4, combined with vertical folding by a factor of 4, and internal state stored in memory;
- for Stratix III: /2(h), i.e., architecture with horizontal folding by a factor of 2.

The good performance of the former of these two architectures is associated with the significant reduction of the complexity of the input permutation module as a result of vertical folding by a factor of 4. The two less successful architectures include x1 and /2(h)-PPL4 for Virtex 5, and x1 and x1-PPL2 for Stratix III. These architectures are not included in our combined graphs shown in Figs. 6-9.

For Groestl (see Figs. 4b and 5b), we consider two major architecture types: a) parallel architectures, denoted (P+Q), in which Groestl permutations P and Q are implemented using two independent units, working in parallel, and b) quasi-pipeline architectures, denoted (P/Q), in which, the same unit is used to implement both P and Q, and the computations belonging to these two permutations are interleaved [38]. The details of the basic quasi-pipelined architecture of Groestl are described in [38, Section 9] and [23, Section 3.8]. In this study, we apply vertical folding and pipelining to both architectures. The best architectures overall appear to be: x1-PPL2 (P+Q) for Virtex 5, and x1-PPL4 (P+Q) for Stratix III. The best non-pipelined architectures are: x1 (P/Q) for Virtex 5, and /2(v) (P/Q) for Stratix III. Folded parallel architectures, /k(v) (P+Q), are slower than the quasi-pipelined architectures (P/Q) using comparable area. The same is true for the basic iterative parallel architecture, x1 (P+Q). An attempt to pipeline Groestl using 7 pipeline stages (x1-PPL7), using logic-only implementation of S-boxes, appeared to be rather unsuccessful.

For JH (see Figs. 4c and 5c), we consider two major types of architectures: a) with round constants stored in memory, JH (MEM), and b) with round constants calculated on the fly, JH (OTF). Both approaches seem to result in a very similar performance for the basic iterative architectures, x1. Neither vertical folding nor pipelining seem to be efficient when applied directly to the basic architecture. Vertical folding by two, somewhat unexpectedly, increases area, and the basic architecture with two pipeline stages does not improve throughput. Both undesired effects can be tracked back to the simplicity of the main round. Folding does not reduce area, because of extra registers and multiplexers introduced to a very simple round. Pipelining does not increase throughput, because a simple basic round has already very short delay, and is hard to divide into two well balanced pipeline stages. Overall, the best architectures are: x1 (MEM) for Virtex 5 and x2-PPL2 (MEM) for Stratix III.

For Keccak (see Figs. 4d and 5d), the best architecture overall is the basic iterative architecture. Pipelining appears to be quite unsuccessful in Virtex 5, and somewhat more successful in Stratix III, where three different pipelined architectures (x1-PPL2, x2-PPL2, and x2-PPL4) give similar throughput to area ratio as the basic iterative architecture, x1. Vertical folding has been attempted only for a version without padding, and therefore the corresponding results are not shown in Figs. 4-9. As shown in Table 5, vertical folding by a factor of 8, with internal state stored in memory, leads to the reduction in area by a factor of about 4 for Virtex 5 and about 1.5 for Stratix III, at the cost of a significant reduction in throughput, by a factor of about 16 in Virtex 5 and by a factor of about 18 in Stratix III. Thus, the throughput to area ratio decreases by a factor of 4 for Virtex 5, and a factor of 12 for Stratix III.

For Skein (see Figs. 4e and 5e), the unrolled by 4 architecture, x4, is significantly more efficient than the basic architecture, x1. At the same time, unrolling by 8 does not give any additional improvement in the throughput to area ratio. The best pipelined architectures are obtained by first unrolling basic architecture by a factor of four, and then pipelining the obtained circuit using two stages in Virtex 5, and five stages in Stratix III. Five pipeline stages are efficient in Stratix III because of an extra addition executed every fourth round, but they do not improve the overall throughput to area ratio in Virtex 5.

For SHA-2 (see Figs. 4f and 5f), none of the discussed techniques applies. The implementation of this function is already small, so reducing area is not necessary. The best way to speed up this function is by using multiple independent units of SHA-2 working in parallel. We denote this architecture by MUn, where n denotes the number of hash units.

The combined graphs for the 256-bit variants and the 512-bit variants of all algorithms, implemented using Xilinx Virtex 5 FPGAs, are presented in Figs. 6 and 7. Individual dots placed in regular intervals on the dashed lines represent multi-unit architectures. Algorithms can be ranked first in terms of the

throughput to area ratio of their best architecture, as identified above. This is because this architecture can be easily replicated, allowing for processing n streams of data in parallel. Both throughput and area will increase by a factor of n. The secondary criterion is the area of the best architecture. The smaller the area, the denser is the graph representing possible locations of a given function on the throughput vs. area graph.

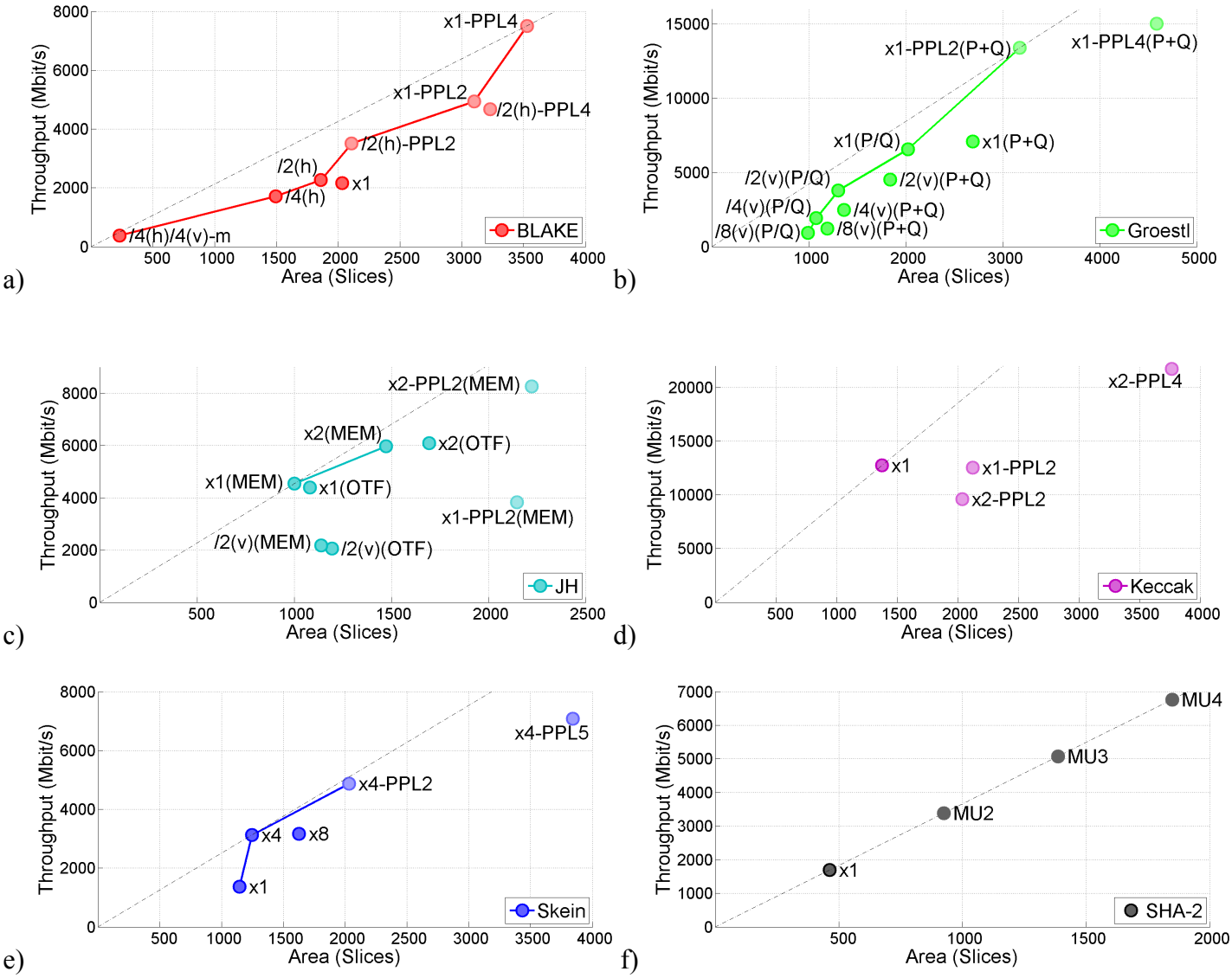


Fig. 4. Throughput vs. Area graphs for multiple architectures of a) BLAKE-256, b) Groestl-256, c) JH-256, d) Keccak-256, e) Skein-256, and f) SHA-256, implemented in ***Xilinx Virtex 5*** FPGAs. Notation: x1 – basic iterative architecture, /k(h) – horizontally folded by a factor of k, /k(v) - vertically folded by a factor of k, /k(v)-m - vertically folded by a factor of k with internal state stored in memory, xk – unrolled by a factor of k, PPLn – pipelined with n pipeline stages, (P+Q) – parallel architecture of Groestl, P/Q – quasi-pipelined architecture of Groestl, MEM – architecture of JH with round constants stored in memory, OTF – architecture of JH with round constants calculated on the fly.

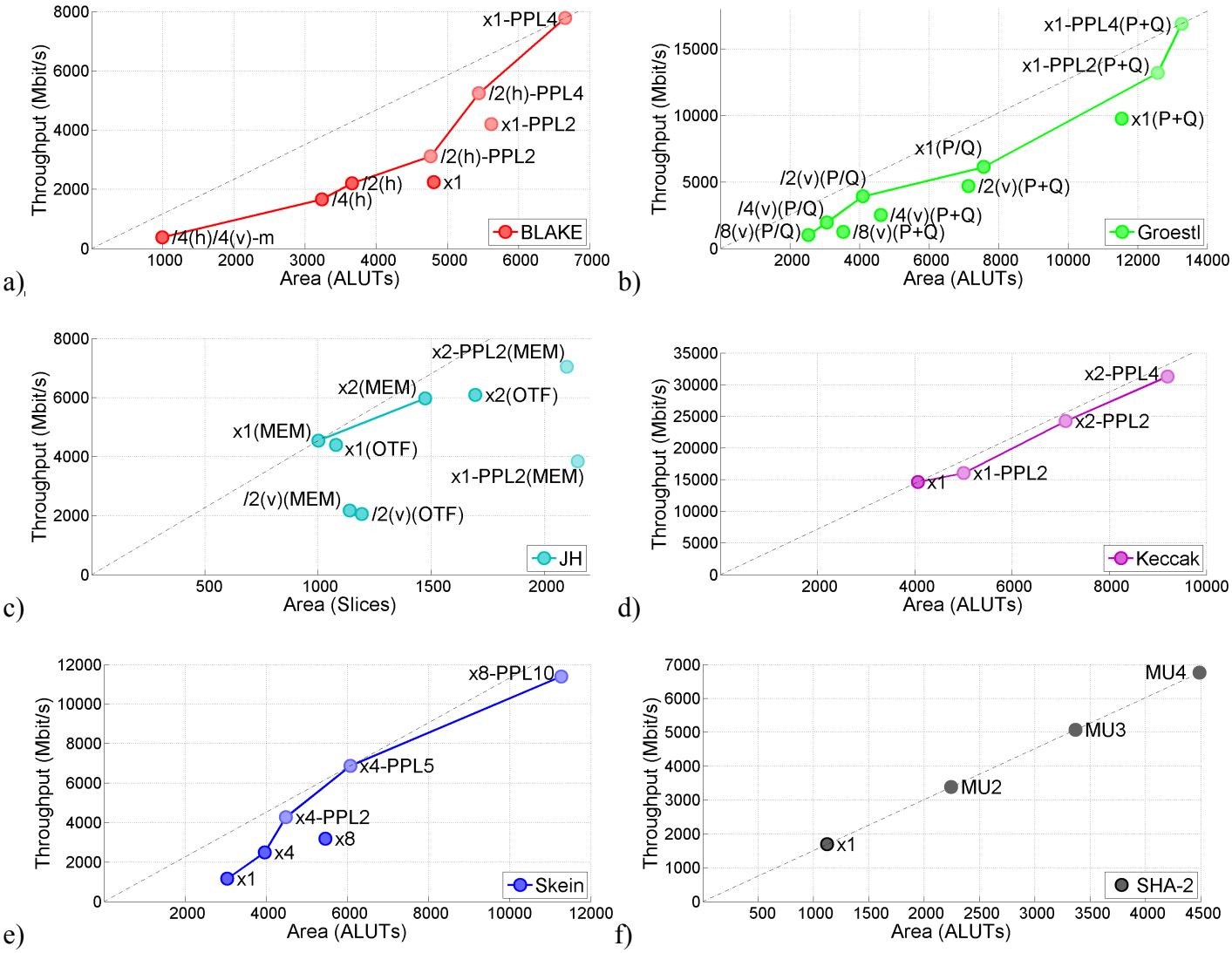


Fig. 5. Throughput vs. Area graphs for multiple architectures of a) BLAKE-256, b) Groestl-256, c) JH-256, d) Keccak-256, e) Skein-256, and f) SHA-256, implemented in ***Altera Stratix III*** FPGAs. Notation: x1 – basic iterative architecture, /k(h) – horizontally folded by a factor of k, /k(v) - vertically folded by a factor of k, /k(v)-m - vertically folded by a factor of k with internal state stored in memory, xk – unrolled by a factor of k, PPLn – pipelined with n pipeline stages, (P+Q) – parallel architecture of Groestl, P/Q – quasi-pipelined architecture of Groestl, MEM – architecture of JH with round constants stored in memory, OTF – architecture of JH with round constants calculated on the fly.

The results for the 256-bit variants of hash functions are shown in Fig. 6. Keccak is the only function that significantly outperforms SHA-2 in terms of the throughput to area ratio. Keccak is also significantly faster than SHA-2, for any area greater than 1400 CLB slices. Groestl and JH demonstrate performance similar to SHA-2, with JH implementations slightly faster than SHA-2 consistently starting from around 1000 CLB slices, and Groestl implementations exceeding the speed of SHA-2 (and JH) only for one implementation shown in the diagram, taking around 3000 CLB slices. Skein and BLAKE trail significantly behind SHA-2, independently of the area. The results for the 512-bit variants of hash functions, shown in Fig. 7, are quite similar, with the exception that, JH performs almost equally well as Keccak (because of the decrease in the Keccak message block size from 1088 to 576 bits), Groestl outperforms SHA-2 and Skein only for area around 6000 CLB slices, Skein has a performance close to SHA-2, and BLAKE is a distant fifth.

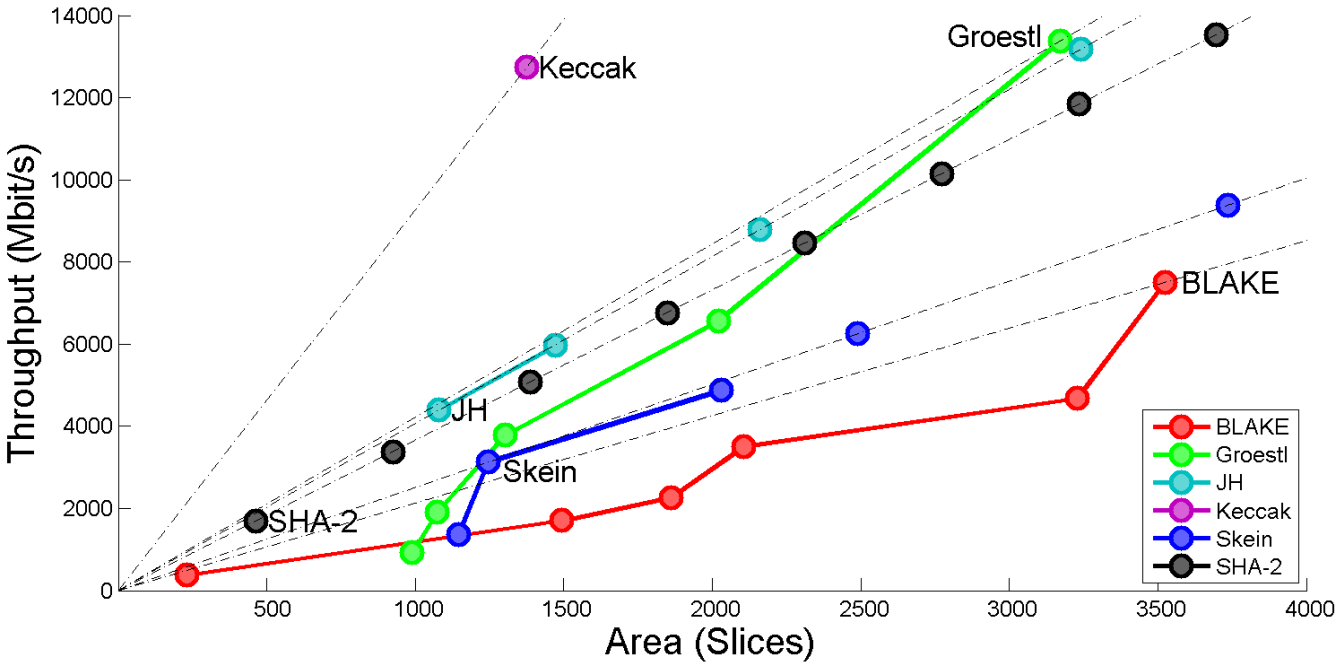


Fig. 6. Combined Throughput vs. Area graph for multiple hardware architectures of the 256-bit variants of BLAKE, Groestl, JH, Keccak, Skein, and SHA-2 implemented in Xilinx Virtex 5 FPGAs.

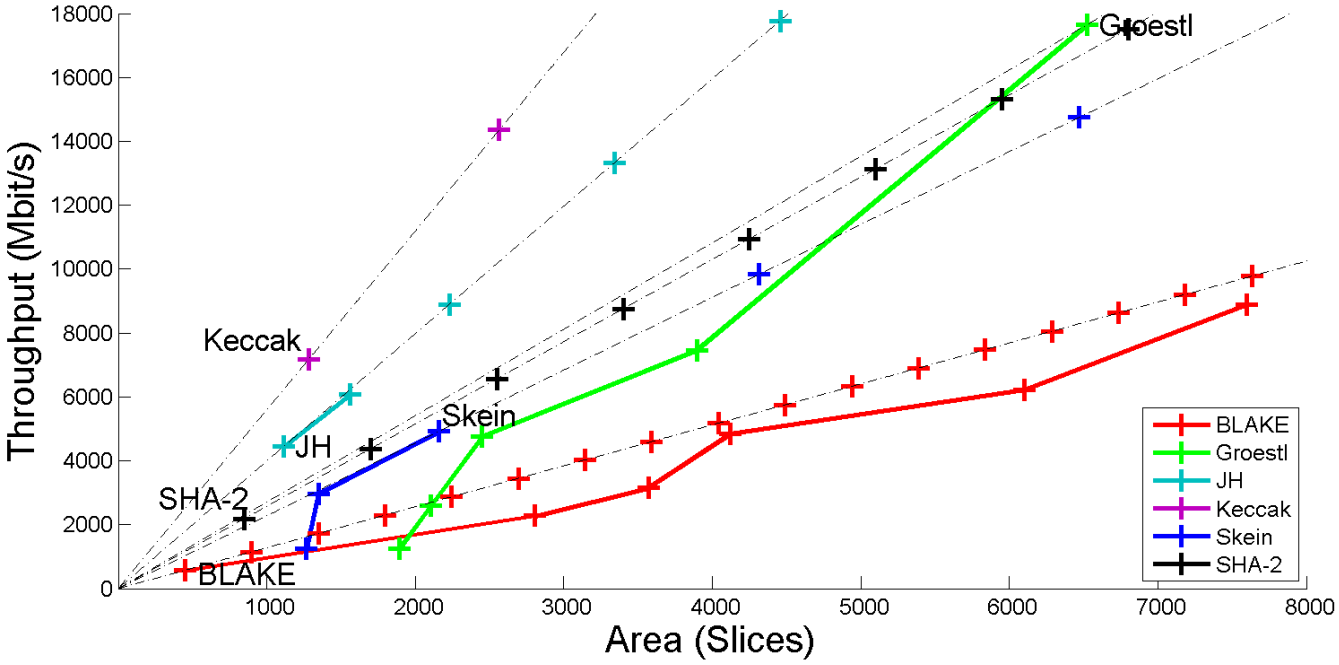


Fig. 7. Combined Throughput vs. Area graph for multiple hardware architectures of the 512-bit variants of BLAKE, Groestl, JH, Keccak, Skein, and SHA-2 implemented in Xilinx Virtex 5 FPGAs.

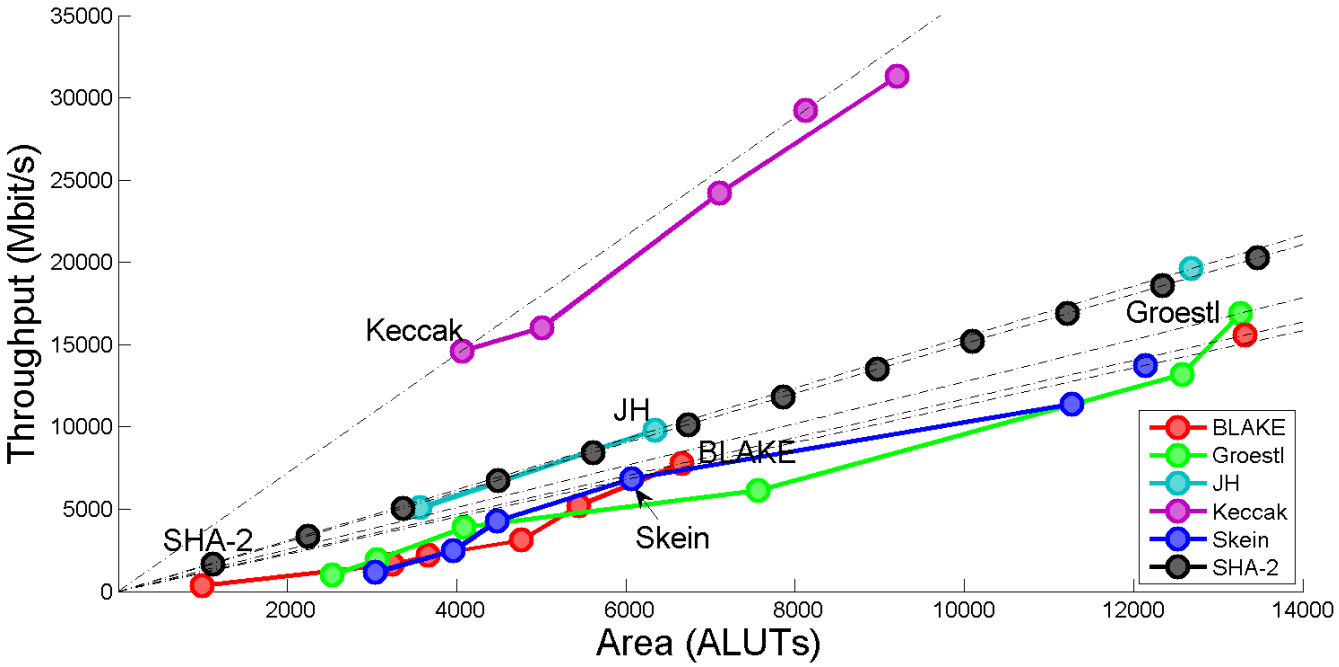


Fig. 8. Combined Throughput vs. Area graph for multiple hardware architectures of the 256-bit variants of BLAKE, Groestl, JH, Keccak, Skein, and SHA-2 implemented in Altera Stratix III FPGAs.

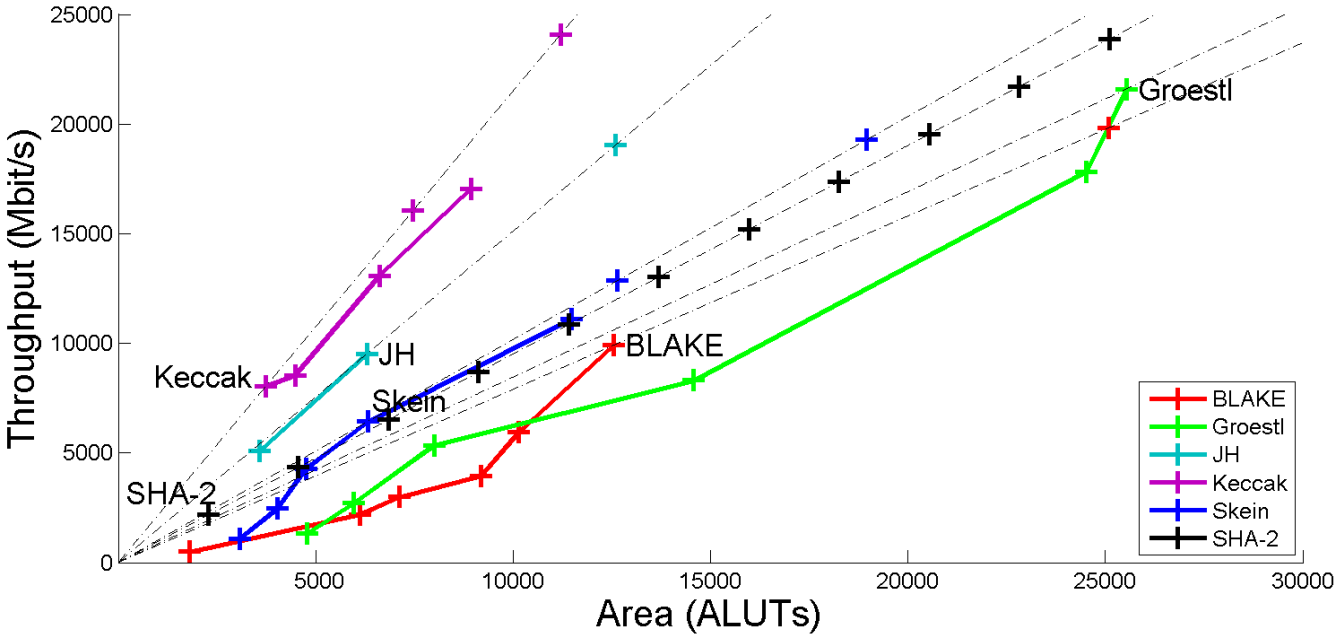


Fig. 9. Combined Throughput vs. Area graph for multiple hardware architectures of the 512-bit variants of BLAKE, Groestl, JH, Keccak, Skein, and SHA-2 implemented in Altera Stratix III FPGAs.

The performance for Altera devices, shown in Figs. 8 and 9, is somewhat different. For the 256-bit versions of the algorithms, Keccak by far outperforms all remaining candidates and SHA-2. Best implementations of JH work as fast as the similar-size implementations of SHA-2, but SHA-2 offers finer granularity, as its size can be increased in much smaller increments, and the throughput changes proportionally. Groestl, BLAKE, and Skein are in tie with each, with performance consistently worse than SHA-2. For the 512-bit versions of the algorithms (see Fig. 9), Keccak and JH outperform SHA-2, Skein is in tie with SHA-2, Groestl and BLAKE fall behind the current standard. The numerical results for all our implementations are summarized in Tables 2 and 3. The best values of the throughput to area ratios and the best architectures for each hash function are listed in **bold** in these tables.

Table 2: Results for 256-bit variants ***with*** padding unit of the Round 3 SHA-3 candidates and SHA-2, implemented using all investigated architectures and four FPGA families, Virtex 5 and Virtex 6 from Xilinx, and Stratix III and Stratix IV from Altera. Notation: Tp – throughput, A – area, Tp/A – Throughput to Area Ratio. The best values of the throughput to area ratios and the best architectures for each hash function are listed in **bold**.

Arch	Virtex 5			Virtex 6			Stratix III			Stratix IV		
	Tp	A	Tp/A	Tp	A	Tp/A	Tp	A	Tp/A	Tp	A	Tp/A
BLAKE-256												
/4(h)/4(v)-m	379	230	1.65	428	200	2.14	379	992	0.38	395	1022	0.39
/4(h)	1713	1493	1.15	1957	891	2.20	1665	3238	0.51	1691	3234	0.52
/2(h)	2266	1860	1.22	2363	1391	1.70	2206	3660	0.60	2316	3680	0.63
x1	2156	2032	1.06	2174	1505	1.44	2244	4807	0.47	2322	4802	0.48
/2(h)-PPL2	3510	2107	1.67	3260	1690	1.93	3118	4765	0.65	3421	4765	0.72
/2(h)-PPL4	4675	3228	1.45	4332	2268	1.91	5238	5442	0.96	5443	5449	1.00
x1-PPL2	4943	3099	1.59	4542	2040	2.23	4193	5619	0.75	4483	5642	0.79
x1-PPL4	7510	3526	**2.13**	8246	2609	**3.16**	7787	6657	**1.17**	7898	6657	**1.19**
Groestl-256												
/8(v) (P+Q)	1211	1191	1.02	1353	958	1.41	1248	3526	0.35	1172	3472	0.34
/4(v) (P+Q)	2486	1362	1.83	2901	1200	2.42	2533	4608	0.55	2391	4585	0.52
/2(v) (P+Q)	4508	1836	2.46	4915	1565	3.14	4685	7128	0.66	4597	7343	0.63
x1 (P+Q)	7081	2689	2.63	9187	2441	3.76	9760	11538	0.85	9181	11294	0.81
x1-PPL2 (P+Q)	13382	3172	**4.22**	11746	2968	3.96	13213	12572	1.05	12586	12570	1.00
x1-PPL4 (P+Q)	15015	4587	3.27	15624	4172	3.74	16903	13261	**1.27**	16126	13258	**1.22**
/8(v) (P/Q)	918	990	0.93	1105	750	1.47	1005	2526	0.40	966	2504	0.39
/4(v) (P/Q)	1920	1074	1.79	2099	811	2.59	1964	3061	0.64	1829	3059	0.60
/2(v) (P/Q)	3784	1302	2.91	4407	944	**4.67**	3925	4086	0.96	3644	4069	0.90
x1 (P/Q)	6572	2020	3.25	7071	1884	3.75	6140	7564	0.81	5640	7464	0.76
JH-256												
/2(v) (MEM)	2176	1139	1.91	1978	879	2.25	2124	3636	0.58	2086	3629	0.57
x1 (MEM)	4543	1001	**4.54**	5086	918	5.54	5024	3383	1.49	4815	3415	1.41
x2 (MEM)	5972	1473	4.05	6883	1381	4.98	6532	5830	1.12	6099	5793	1.05
x1-PPL2 (MEM)	3838	2147	1.79	4840	1486	3.26	5177	4280	1.21	5233	4278	1.22
x2-PPL2 (MEM)	7041	2099	3.35	8409	1781	4.72	9804	6339	**1.55**	9526	6331	**1.50**
x2-PPL4 (MEM)	8526	3085	2.76	7928	2424	3.27	9342	7962	1.17	9207	7960	1.16
/2(v) (OTF)	2054	1194	1.72	2206	1094	2.02	2114	3766	0.56	1981	3798	0.52
x1 (OTF)	4392	1080	4.07	5177	920	**5.63**	5091	3569	1.43	4807	3656	1.31
x2 (OTF)	6094	1695	3.60	7198	1604	4.49	6401	6372	1.00	5878	6360	0.92
Keccak-256												
x1	12745	1375	**9.27**	12451	1147	**10.86**	14624	4060	**3.60**	14009	4052	**3.46**
x1-PPL2	12523	2123	5.90	14942	1456	10.26	16047	5003	3.21	16878	5004	3.37
x2-PPL2	9610	2036	4.72	14444	2338	6.18	24242	7103	3.41	24942	7869	3.17
x2-PPL4	21717	3764	5.77	24644	2900	8.50	31296	9201	3.40	31691	9203	3.44
Skein-256												
x1	1372	1145	1.20	1343	889	1.51	1152	3032	0.38	1269	3031	0.42
x4	3127	1245	**2.51**	2957	1026	2.88	2494	3960	0.63	2647	3970	0.67
x8	3168	1627	1.95	3548	1326	2.68	3193	5455	0.59	3336	5451	0.61
x4-PPL2	4873	2030	2.40	5679	1485	**3.82**	4280	4482	0.95	4688	4496	1.04
x4-PPL5	7077	3840	1.84	7325	2720	2.69	6869	6068	**1.13**	7824	6070	**1.29**
x8-PPL10	N/A	N/A	N/A	11118	5928	1.88	11390	11267	1.01	11485	11267	1.02
SHA-256												
x1	1692	462	**3.66**	1665	305	**5.46**	1690	1122	**1.51**	1764	1114	**1.58**

Table 3: Results for 512-bit variants ***with*** padding unit of the Round 3 SHA-3 candidates and SHA-2, implemented using all investigated architectures and four FPGA families, Virtex 5 and Virtex 6 from Xilinx, and Stratix III and Stratix IV from Altera. Notation: Tp – throughput, A – area, Tp/A – Throughput to Area Ratio. The best values of the throughput to area ratios and the best architectures for each hash function are listed in **bold**.

Arch	Virtex 5			Virtex 6			Stratix III			Stratix IV		
	Tp	A	Tp/A	Tp	A	Tp/A	Tp	A	Tp/A	Tp	A	Tp/A
BLAKE-512												
/4(h)/4(v)-m	575	449	**1.28**	625	350	1.79	495	1797	0.28	537	1791	0.30
/4(h)	2278	2805	0.81	2675	1639	1.63	2206	6124	0.36	2492	6399	0.39
/2(h)	3156	3569	0.88	3333	2608	1.28	3003	7115	0.42	3320	7114	0.47
x1	3046	3384	0.90	N/A	N/A	N/A	3010	9343	0.32	3387	9334	0.36
/2(h)-PPL2	4853	4123	1.18	4452	2882	1.54	3963	9188	0.43	4799	9177	0.52
/2(h)-PPL4	6225	6104	1.02	5412	3951	1.37	5938	10137	0.59	7668	10144	0.76
x1-PPL2	6296	5534	1.14	6430	4171	1.54	5705	10828	0.53	6356	10831	0.59
x1-PPL4	8882	7600	1.17	11030	5080	**2.17**	9908	12537	**0.79**	11048	12530	**0.88**
Groestl-512												
/8(v) (P+Q)	1602	2233	0.72	1683	1726	0.98	1697	6768	0.25	1589	6723	0.24
/4(v) (P+Q)	3147	2570	1.22	3078	1952	1.58	3562	9089	0.39	3323	8916	0.37
/2(v) (P+Q)	5413	3364	1.61	6051	3078	1.97	6642	14597	0.46	6175	14263	0.43
x1 (P+Q)	8781	5448	1.61	11608	5112	2.27	12976	22239	0.58	11903	21807	0.55
x1-PPL2 (P+Q)	17655	6525	**2.71**	16218	5740	**2.83**	17826	24518	0.73	16363	24612	0.66
x1-PPL4 (P+Q)	17918	8453	2.12	16033	7391	2.17	21595	25529	**0.85**	20357	25530	**0.80**
/8(v) (P/Q)	1238	1890	0.65	1271	1358	0.94	1317	4780	0.28	1296	4774	0.27
/4(v) (P/Q)	2602	2107	1.24	2684	1459	1.84	2710	5961	0.45	2586	5887	0.44
/2(v) (P/Q)	4750	2449	1.94	5040	1797	2.80	5352	7995	0.67	5023	7896	0.64
x1 (P/Q)	7462	3895	1.92	6843	3285	2.08	8310	14578	0.57	7882	14542	0.54
JH-512												
/2(v) (MEM)	2044	1145	1.79	2052	925	2.22	2124	3636	0.58	2086	3629	0.57
x1 (MEM)	4533	1125	**4.03**	4834	901	**5.37**	4309	3930	1.10	4201	3919	1.07
x2 (MEM)	6067	1561	3.89	6701	1417	4.73	6532	5830	1.12	6099	5793	1.05
x1-PPL2 (MEM)	3897	2244	1.74	4541	1589	2.86	5157	4537	1.14	5319	4536	1.17
x2-PPL2 (MEM)	8266	2223	3.72	8514	1977	4.31	9514	6297	**1.51**	9484	6303	**1.50**
x2-PPL4 (MEM)	6186	3645	1.70	8047	2744	2.93	9328	8084	1.15	9512	8079	1.18
/2(v) (OTF)	1985	1228	1.62	2131	1077	1.98	2114	3766	0.56	1981	3798	0.52
x1 (OTF)	4443	1114	3.99	4914	965	5.09	5091	3569	1.43	4807	3656	1.31
x2 (OTF)	5940	1664	3.57	6732	1546	4.35	6401	6372	1.00	5878	6360	0.92
Keccak-512												
x1	7179	1283	**5.60**	7465	1052	**7.10**	8029	3734	**2.15**	7607	3723	**2.04**
x1-PPL2	7380	1774	4.16	8114	1263	6.42	8550	4484	1.91	8962	4481	2.00
x2-PPL2	7126	1996	3.57	N/A	N/A	N/A	13090	6617	1.98	12490	6580	1.90
x2-PPL4	13552	3428	3.95	13640	2550	5.35	17058	8934	1.91	17335	8934	1.94
Skein-512												
x1	1258	1267	0.99	1446	987	1.47	1103	3086	0.36	1216	3088	0.39
x4	2972	1348	2.20	3141	1186	2.65	2493	4035	0.62	2597	4026	0.65
x8	2870	1556	1.84	3690	1454	2.54	3137	5527	0.57	3357	5536	0.61
x4-PPL2	4916	2157	**2.28**	5713	1567	**3.65**	4292	4756	0.90	4758	4767	1.00
x4-PPL5	5829	4377	1.33	7337	3160	2.32	6428	6319	**1.02**	7123	6324	**1.13**
x8-PPL10	5946	9032	0.66	9130	6544	1.40	11111	11485	0.97	10542	11486	0.92
SHA-512												
x1	2189	850	**2.58**	2357	548	**4.30**	2171	2282	**0.95**	2409	2313	**1.04**

7. Results Without Padding

In Table 4, we investigate the effect of padding unit on the performance of selected non-pipelined architectures. Based on this table, the largest decrease in the throughput to area ratio is equal to 18%. This decrease depends on the FPGA family, and reaches the maximum of about 10% for Virtex 5, 14% for Virtex 6, 18% for Stratix III, and 15% for Stratix IV. This level variations in the throughput to area ratio do not affect the ranking of candidates as determined in Section 6.

In Tables 5 and 6, the complete set of results for implementations without padding is given. These results are important for comparison with results from other groups, as majority of results reported by other groups during the competition (and submitted to the ATHENa database [3]) concern designs with no padding unit.

Additionally, as shown in the same tables, a folded architecture with internal state stored in memory, /8(v)-m, has been implemented for JH-256 and Keccak-256, in the version without padding. For JH-256, implemented in Virtex 5 and Virtex 6, area decreases by a factor greater than 3 and throughput by a factor greater than 30, compared to the basic iterative architecture. In Stratix III and Stratix IV, area decreases by a factor of about 1.7 and throughput by a factor of about 40. Thus, this architecture is quite inefficient in both Xilinx and Altera FPGAs, and should be considered only when no other architecture fits within the area budget. For Keccak-256, implemented in Virtex 5 and Virtex 6 using the same architecture, area decreases by a factor greater than 3.5 and throughput by a factor greater than 10, compared to the basic iterative architecture. In Stratix III and Stratix IV, area decreases by a factor of about 1.5 and throughput by a factor of about 20. Thus, this architecture might be acceptable in case of Xilinx FPGAs, but is quite inefficient in case of Altera FPGAs.

Table 4: The effect of the padding unit on the performance of 5 Round 3 SHA-3 candidates in four FPGA families, Virtex 5 and Virtex 6 from Xilinx, and Stratix III and Stratix IV from Altera. Notation: Tp – throughput, A – area, Tp/A – Throughput to Area Ratio, Δ [%] – relative change in the Throughput, Area, and Throughput to Area ratio as a result of adding padding unit to the hash unit. The relative change in the throughput to area ratio has been marked in **bold**.

Arch	Virtex 5			Virtex 6			Stratix III			Stratix IV		
	Tp	A	Tp/A	Tp	A	Tp/A	Tp	A	Tp/A	Tp	A	Tp/A
BLAKE-256												
/2(h)	2308	1771	1.30	2226	1257	1.77	2157	3553	0.61	2337	3543	0.66
/2(h)-PAD	2266	1860	1.22	2363	1391	1.70	2206	3660	0.60	2316	3680	0.63
Δ [%]	-1.83	5.03	**-6.53**	6.18	10.66	**-4.04**	2.25	3.01	**-0.74**	-0.90	3.87	**-4.59**
Groestl-256 (P/Q)												
x1	6117	1795	3.41	7220	1870	3.86	6008	7386	0.81	5776	7404	0.78
x1-PAD	6572	2020	3.25	7071	1884	3.75	6140	7564	0.81	5640	7464	0.76
Δ [%]	7.44	12.53	**-4.53**	-2.06	0.75	**-2.79**	2.19	2.41	**-0.21**	-2.36	0.81	**-3.14**
JH-256 (MEM)												
x1	4955	982	5.05	5412	849	6.37	5276	3221	1.64	4759	3210	1.48
x1-PAD	4543	1001	4.54	5086	918	5.54	5024	3383	1.49	4815	3415	1.41
Δ [%]	-8.32	1.93	**-10.06**	-6.02	8.13	**-13.09**	-4.77	5.03	**-9.33**	1.17	6.39	**-4.90**
Keccak-256												
x1	13337	1369	9.74	11839	1086	10.90	15493	3531	4.39	14401	3541	4.07
x1-PAD	12745	1375	9.27	12451	1147	10.86	14624	4060	3.60	14009	4052	3.46
Δ [%]	-4.44	0.44	**-4.86**	5.16	5.62	**-0.43**	-5.61	14.98	**-17.91**	-2.72	14.43	**-14.99**
Skein-256												
x4	3023	1218	2.48	3373	1005	3.36	2475	3943	0.63	2592	3936	0.66
x4-PAD	3127	1245	2.51	2957	1026	2.88	2495	3960	0.63	2647	3970	0.67
Δ [%]	3.43	2.22	**1.19**	-12.33	2.09	**-14.13**	0.77	0.43	**0.34**	2.10	0.86	**1.23**

Table 5: Results for 256-bit variants ***without*** padding unit of the Round 3 SHA-3 candidates and SHA-2, implemented using all investigated architectures and four FPGA families, Virtex 5 and Virtex 6 from Xilinx, and Stratix III and Stratix IV from Altera. Notation: Tp – throughput, A – area, Tp/A – Throughput to Area Ratio. The best values of the throughput to area ratios and the best architectures for each hash function are listed in **bold**.

Arch	Virtex 5			Virtex 6			Stratix III			Stratix IV		
	Tp	A	Tp/A	Tp	A	Tp/A	Tp	A	Tp/A	Tp	A	Tp/A
BLAKE-256												
/4(h)/4(v)-m	389	231	1.69	432	182	2.38	370	924	0.40	399	935	0.43
/4(h)	1735	1437	1.21	1882	886	2.12	1695	3093	0.55	1735	3085	0.56
/2(h)	2308	1771	1.30	2226	1257	1.77	2157	3553	0.61	2337	3543	0.66
x1	2533	2279	1.11	2416	1711	1.41	2181	4620	0.47	2312	4618	0.50
/2(h)-PPL2	3506	2136	1.64	3178	1630	1.95	3131	4570	0.69	3409	4567	0.75
/2(h)-PPL4	4633	3226	1.44	4807	2407	2.00	5205	5039	1.03	5467	5042	1.08
x1-PPL2	4761	2976	1.60	4768	2111	2.26	4162	5436	0.77	4429	5423	0.82
x1-PPL4	7547	3495	**2.16**	8056	2530	**3.18**	7583	6267	**1.21**	8063	6271	**1.29**
Groestl-256												
/8(v) (P+Q)	1237	1124	1.10	1371	936	1.46	1240	3306	0.38	1173	3288	0.36
/4(v) (P+Q)	2215	1208	1.83	2850	1072	2.66	2576	4528	0.57	2366	4402	0.54
/2(v) (P+Q)	4254	1734	2.45	4850	1548	3.13	5028	7444	0.68	4387	6895	0.64
x1 (P+Q)	7214	2906	2.48	8754	2395	3.65	9572	11193	0.86	8962	10961	0.82
x1-PPL2 (P+Q)	12479	2971	4.20	13410	2873	4.67	13166	12531	1.05	12290	12203	1.01
x1-PPL4 (P+Q)	16353	4177	**3.91**	16213	3597	4.51	16198	12885	**1.26**	16141	12933	**1.25**
/8(v) (P/Q)	951	981	0.97	1057	705	1.50	1009	2346	0.43	976	2342	0.43
/4(v) (P/Q)	1907	993	1.92	2381	859	2.77	1998	2919	0.68	1837	2902	0.63
/2(v) (P/Q)	3721	1195	3.11	4201	898	**4.68**	3818	3914	0.98	3701	3906	0.95
x1 (P/Q)	6117	1795	3.41	7220	1870	3.86	6008	7386	0.81	5776	7404	0.78
JH-256												
/8(v)-m (MEM)	138	306	0.45	157	226	0.69	133	1865	0.07	118	1849	0.06
/2(v) (MEM)	2094	1009	2.08	2327	944	2.46	2131	3379	0.63	2138	3368	0.63
x1 (MEM)	4955	982	**5.05**	5412	849	**6.37**	5276	3221	**1.64**	4759	3210	1.48
x2 (MEM)	6149	1489	4.13	6904	1335	5.17	6418	5584	1.15	6128	5542	1.11
x1-PPL2 (MEM)	4711	1842	2.56	5202	1320	3.94	5463	4263	1.28	5439	4259	1.28
x2-PPL2 (MEM)	8289	2312	3.59	9284	2050	4.53	10116	6294	1.61	9772	6259	**1.56**
x2-PPL4 (MEM)	8526	3085	2.76	8839	2162	4.09	9927	6892	1.44	9994	6883	1.45
/2(v) (OTF)	2181	1120	1.95	1993	845	2.36	2084	3473	0.60	2035	3538	0.58
x1 (OTF)	4840	971	4.98	5255	917	5.73	5071	3388	1.50	4912	3385	1.45
x2 (OTF)	6196	1640	3.78	7046	1493	4.72	6359	6121	1.04	5817	5993	0.97
Keccak-256												
/8(v)-m	855	354	2.41	1078	306	3.52	874	2397	0.36	813	2387	0.34
x1	13337	1369	**9.74**	11839	1086	10.90	15493	3531	**4.39**	14401	3541	4.07
x1-PPL2	16121	1950	8.27	18803	1474	**12.76**	19971	4810	4.15	19415	4807	**4.04**
x2-PPL2	17677	2390	7.40	N/A	N/A	N/A	25283	7107	3.56	23660	7018	3.37
x2-PPL4	26690	3714	7.19	29825	2748	10.85	35780	8806	4.06	35006	8803	3.98
Skein-256												
x1	1179	1025	1.15	1330	858	1.55	1115	3005	0.37	1226	3003	0.41
x4	3023	1218	2.48	3373	1005	3.36	2475	3943	0.63	2592	3936	0.66
x8	2890	1492	1.94	3459	1333	2.60	3161	5432	0.58	3345	5432	0.62
x4-PPL2	5338	1858	**2.87**	6212	1628	**3.82**	4273	4423	0.97	4709	4446	1.06
x4-PPL5	6819	4130	1.65	7669	3126	2.45	6974	5941	**1.17**	7675	5925	**1.30**
x8-PPL10	N/A	N/A	N/A	12403	5447	2.28	11741	11163	1.05	11792	10992	1.07
SHA-256												
x1	1401	396	**3.54**	1634	239	**6.83**	1656	959	**1.73**	1798	959	**1.87**

Table 6: Results for 512-bit variants ***without*** padding unit of the Round 3 SHA-3 candidates and SHA-2, implemented using all investigated architectures and four FPGA families, Virtex 5 and Virtex 6 from Xilinx, and Stratix III and Stratix IV from Altera. Notation: Tp – throughput, A – area, Tp/A – Throughput to Area Ratio. The best values of the throughput to area ratios and the best architectures for each hash function are listed in **bold**.

Arch	Virtex 5			Virtex 6			Stratix III			Stratix IV		
	Tp	A	Tp/A	Tp	A	Tp/A	Tp	A	Tp/A	Tp	A	Tp/A
BLAKE-512												
/4(h)/4(v)-m	560	386	**1.45**	613	309	1.98	491	1680	0.29	543	1676	0.32
/4(h)	2300	2840	0.81	2646	1584	1.67	2186	5891	0.37	2442	5885	0.42
/2(h)	3264	3435	0.95	3478	2610	1.33	2928	6977	0.42	3318	6971	0.48
x1	N/A	N/A	N/A	N/A	N/A	N/A	2965	9033	0.33	3323	9024	0.37
/2(h)-PPL2	4841	4515	1.07	4478	2879	1.56	3954	8969	0.44	4742	8959	0.53
/2(h)-PPL4	6171	5794	1.07	6915	4575	1.51	5991	9684	0.62	7859	9694	0.81
x1-PPL2	6364	5674	1.12	6606	4616	1.43	5660	10625	0.53	6351	10615	0.60
x1-PPL4	9567	7497	1.28	10706	5267	**2.03**	9980	12074	**0.83**	11075	12082	**0.92**
Groestl-512												
/8(v) (P+Q)	1557	2251	0.69	1726	1773	0.97	1677	6549	0.26	1614	6510	0.25
/4(v) (P+Q)	3112	2393	1.30	3230	2113	1.53	3447	8727	0.39	3277	8750	0.37
/2(v) (P+Q)	5119	3289	1.56	5793	2971	1.95	6595	14318	0.46	6265	14207	0.44
x1 (P+Q)	10020	5588	1.79	12262	5203	2.36	13061	22062	0.59	11936	21902	0.54
x1-PPL2 (P+Q)	17591	6568	**2.68**	16114	6290	2.56	16900	24292	0.70	16114	24241	0.66
x1-PPL4 (P+Q)	N/A	N/A	N/A	N/A	N/A	N/A	21158	25515	**0.83**	20580	25407	**0.81**
/8(v) (P/Q)	1211	1722	0.70	1326	1358	0.98	1335	4598	0.29	1307	4592	0.28
/4(v) (P/Q)	2573	2036	1.26	2772	1529	1.81	2700	5786	0.47	2596	5770	0.45
/2(v) (P/Q)	4816	2336	2.06	5319	1761	**3.02**	5262	7763	0.68	4989	7724	0.65
x1 (P/Q)	7686	3853	1.99	8375	3630	2.31	8214	14291	0.57	8379	14620	0.57
JH-512												
/8(v)-m (MEM)	138	307	0.45	154	228	0.68	128	1817	0.07	119	1851	0.06
/2(v) (MEM)	2052	1055	1.95	2491	996	2.50	2224	3664	0.61	2175	3660	0.59
x1 (MEM)	4882	1037	4.71	5825	931	**6.26**	5011	3288	1.52	5139	3294	**1.56**
x2 (MEM)	6203	1587	3.91	6859	1377	4.98	6630	5768	1.15	6305	5786	1.09
x1-PPL2 (MEM)	4635	1990	2.33	5060	1534	3.30	5361	4521	1.19	5319	4521	1.18
x2-PPL2 (MEM)	8183	2494	3.28	9439	2128	4.44	9881	6339	**1.56**	9665	6309	1.53
x2-PPL4 (MEM)	8107	3408	2.38	9011	2568	3.51	9456	7427	1.27	8806	7392	1.19
/2(v) (OTF)	2027	1127	1.80	2104	954	2.21	2107	3680	0.57	1982	3669	0.54
x1 (OTF)	4686	992	**4.72**	5181	939	5.52	5181	3557	1.46	5043	3605	1.40
x2 (OTF)	6413	1870	3.43	7128	1501	4.75	6268	6276	1.00	6032	6314	0.96
Keccak-512												
/8(v)-m	512	355	1.44	631	316	2.00	498	2310	0.22	471	2293	0.21
x1	7612	1320	**5.77**	7220	1061	**6.81**	8526	3471	2.46	7825	3467	2.26
x1-PPL2	9306	1720	5.41	9619	1468	6.55	11215	4294	**2.61**	10816	4295	**2.52**
x2-PPL2	9915	2297	4.32	N/A	N/A	N/A	13389	6523	2.05	12984	6519	1.99
x2-PPL4	12935	3387	3.82	15661	2539	6.17	20356	8553	2.38	19300	8549	2.26
Skein-512												
x1	1201	1069	1.12	1441	987	1.46	1135	3072	0.37	1229	3073	0.40
x4	3084	1418	2.17	3462	1114	3.11	2438	4006	0.61	2736	4015	0.68
x8	2832	1577	1.80	3573	1373	2.60	3121	5589	0.56	3322	5507	0.60
x4-PPL2	5378	2026	**2.65**	5943	1702	**3.49**	4271	4705	0.91	4682	4683	1.00
x4-PPL5	N/A	N/A	N/A	7071	3486	2.03	6670	6199	**1.08**	6972	6185	**1.13**
x8-PPL10	N/A	N/A	N/A	12176	6145	1.98	11063	11205	0.99	10802	11204	0.96
SHA-512												
x1	2013	798	**2.52**	2384	513	**4.65**	2128	1995	**1.07**	2378	1996	**1.19**

8. Rankings Based on the Throughput to Area Ratio

In order to explicitly compare SHA-3 finalists against each other across four different FPGA families, we use Tables 7-10, and Figs 10 and 11.

All results presented in these comparisons concern versions of designs ***without padding units***. This choice has been made in order to take into account results submitted by other groups, as majority of these results assume that padding is done outside of the implemented circuits.

The rankings for each FPGA family are divided along the two criteria:

a) 256-bit vs. 512-bit variant, and
b) all architectures vs. single-message architectures.

As a result, *four* rankings are listed for each family, as shown in Tables 7-10.

The category "all architectures" means that we allow both single-message and multi-message architectures. In case two architectures offer the same Throughput to Area ratio, the priority is given to architecture with the smaller area. As a result, the multi-unit architectures, MUn, do not appear in our rankings as they offer at best the same throughput to area ratio as the basic architectures they are based on. Thus, the primary way of improving throughput to area ratio for multi-message architectures is pipelining.

Table 7: Ranking of SHA-3 candidates for 256-bit variant in Xilinx Virtex 5 FPGA. Designs **without padding**, implemented using Xilinx ISE 13.1 (unless explicitly stated otherwise using *).

Rank	Algorithm	Architecture	Throughput/Area ((Mbit/s)/Slices)	Normalized Throughput/Area	Throughput (Mbit/s)	Area (Slices)	Source**
256-bit variant, all architectures							
1	Keccak	x1	10.01	2.83	13,859	1,384	GMU*
2	JH	x1 (OTF)	5.17	1.46	4,725	914	GMU*
3	Groestl	/2(v) (P/Q)	4.90	1.39	6,260	1,277	GMU*
	SHA-2	x1	3.54	1.00	1,401	396	GMU
4	Skein	x4-PPL2	2.87	0.81	5,338	1,858	GMU
5	BLAKE	x1-PPL4	2.16	0.61	7,547	3,495	GMU
256-bit variant, single-message architectures							
1	Keccak	x1	10.01	2.83	13,859	1,384	GMU*
2	JH	x1 (OTF)	5.17	1.46	4,725	914	GMU*
3	Groestl	/2(v) (P/Q)	4.90	1.39	6,260	1,277	GMU*
	SHA-2	x1	3.54	1.00	1,401	396	GMU
4	Skein	x4	2.48	0.70	3,023	1,218	GMU
5	BLAKE	/4(h)/4(v)-m	1.71	0.48	382	223	GMU*
512-bit variant, all architectures							
1	Keccak	x1	5.77	2.29	7,612	1,320	GMU
2	JH	x1 (OTF)	5.17	2.05	4,725	914	GMU*
3	Groestl	x1	2.93	1.16	7,400	2,523	NUST
4	Skein	x4-PPL2	2.65	1.05	5,378	2,026	GMU
	SHA-2	x1	2.52	1.00	2,013	798	GMU
5	BLAKE	/4(h)/4(v)	1.45	0.58	560	386	GMU
512-bit variant, single-message architectures							
1	Keccak	x1	5.77	2.29	7,612	1,320	GMU
2	JH	x1 (OTF)	5.17	2.05	4,725	914	GMU*
3	Groestl	x1	2.93	1.16	7,400	2,523	NUST
	SHA-2	x1	2.52	1.00	2,013	798	GMU
4	Skein	x4	2.17	0.86	3,084	1,418	GMU
5	BLAKE	/4(h)/4(v)	1.45	0.58	560	386	GMU

* Results obtained using Xilinx ISE v.12.4

** GMU – George Mason University (the authors of this paper), NUST – National University of Sciences and Technology, Islamabad, Pakistan [33,3].

As shown in Table 7, for Virtex 5, the ranking of SHA-3 finalists is identical in all categories. Three candidates, Keccak, JH, and Groestl consistently outperform SHA-2. The difference between Keccak and JH is very significant for the 256-bit variants of both functions (almost a factor of 2), and substantially decreases for the 512-bit variants (because of the smaller message block size in Keccak-512 vs. Keccak-256). The advantage of Groestl over SHA-2 also decreases for the 512-bit variants of both functions. For the 256-bit variants of the algorithms, only Skein and BLAKE can benefit from pipelining (in terms of the increase in the throughput to area ratio). For the 512-bit variants of the finalists, none algorithm was shown to benefit from pipelining.

As shown in Table 8, for Virtex 6, the ranking of the five SHA-3 candidates is the same as in Virtex 5, and identical in all categories. Keccak is the only candidate that outperforms SHA-2 for the 256-bit variants of the compared functions. For the 512-bit variants, the performance of Keccak decreases, and becomes comparable to that of JH. Both Keccak and JH outperform SHA-2 in terms of the throughput to area ratio. Groestl, Skein, and BLAKE consistently lag behind SHA-2.

Table 8: Ranking of SHA-3 candidates for 256-bit variant in Xilinx Virtex 6 FPGA. Designs **without padding**, implemented using Xilinx ISE v.13.1 (unless explicitly stated otherwise using *).

Rank	Algorithm	Architecture	Throughput/Area ((Mbit/s)/Slices)	Normalized Throughput/Area	Throughput (Mbit/s)	Area (Slices)	Source**
256-bit variant, all architectures							
1	Keccak	x1	14.94	2.19	13,670	915	NUST
	SHA-2	x1	6.84	1.00	1,634	239	GMU
2	JH	x1 (MEM)	6.73	0.98	5,700	847	GMU*
3	Groestl	x1	6.56	0.96	9,620	1,467	NUST
4	Skein	x4-PPL2	3.82	0.56	6,212	1,628	GMU
5	BLAKE	x1-PPL4	3.18	0.47	8,056	2,530	GMU
256-bit variant, single-message architectures							
1	Keccak	x1	14.94	2.19	13,670	915	NUST
	SHA-2	x1	6.84	1.00	1,634	239	GMU*
2	JH	x1 (MEM)	6.73	0.98	5,700	847	GMU*
3	Groestl	x1	6.56	0.96	9,620	1,467	NUST
4	Skein	x4	3.36	0.49	3,373	1,005	GMU
5	BLAKE	Lightweight	2.86	0.42	475	166	GMU
512-bit variant, all architectures							
1	Keccak	x1	6.89	1.48	6,990	1,015	NUST
2	JH	x1 (MEM)	6.26	1.35	5,825	931	GMU
	SHA-2	x1	4.64	1.00	2,381	513	GMU*
3	Groestl	x1	3.89	0.84	9,170	2,359	NUST
4	Skein	x4-PPL2	3.49	0.75	5,943	1,702	GMU
5	BLAKE	x1-PPL4	2.03	0.44	10,706	5,267	GMU
512-bit variant, single-message architectures							
1	Keccak	x1	6.89	1.48	6,990	1,015	NUST
2	JH	x1 (MEM)	6.26	1.35	5,825	931	GMU
	SHA-2	x1	4.64	1.00	2,381	513	GMU*
3	Groestl	x1	3.89	0.84	9,170	2,359	NUST
4	Skein	x4	3.11	0.67	3,462	1,114	GMU
5	BLAKE	/4(h)/4(v)-m	1.98	0.43	613	309	GMU

* Results obtained using Xilinx ISE v.12.4

** GMU – George Mason University (the authors of this paper), NUST – National University of Sciences and Technology, Islamabad, Pakistan [33,3].

Table 9: Ranking of SHA-3 candidates for 256-bit variant in Altera Stratix III FPGA. Designs **without padding**, implemented using Altera Quartus II v.11.1.

Rank	Algorithm	Architecture	Throughput/Area ((Mbit/s)/Slices)	Normalized Throughput/Area	Throughput (Mbit/s)	Area (ALUTs)	Source
256-bit variant, all architectures							
1	Keccak	x1	4.39	2.54	15,493	3,531	GMU
	SHA-2	x1	1.73	1.00	1,656	959	GMU
2	JH	x1 (MEM)	1.64	0.95	5,276	3,221	GMU
3	Groestl	x1-PPL4 (P+Q)	1.26	0.73	16,197	12,885	GMU
4	BLAKE	x1-PPL4	1.21	0.70	7,583	6,267	GMU
5	Skein	x4-PPL5	1.17	0.68	6,974	5,941	GMU
256-bit variant, single-message architectures							
1	Keccak	x1	4.39	2.54	15,493	3,531	GMU
	SHA-2	x1	1.73	1.00	1,656	959	GMU
2	JH	x1 (MEM)	1.64	0.95	5,276	3,221	GMU
3	Groestl	/2(v) (P/Q)	0.98	0.56	3,818	3,914	GMU
4	Skein	x4	0.63	0.36	2,475	3,943	GMU
5	BLAKE	/2(h)	0.61	0.35	2,158	3,553	GMU
512-bit variant, all architectures							
1	Keccak	x1-PPL2	2.61	2.45	11,215	4,294	GMU
2	JH	x2-PPL2	1.56	1.46	9,881	6,339	GMU
3	Skein	x4-PPL5	1.08	1.01	6,670	6,199	GMU
	SHA-2	x1	1.07	1.00	2,128	1,995	GMU
4	Groestl	x1-PPL4 (P+Q)	0.83	0.78	21,158	25,515	GMU
5	BLAKE	x1-PPL4	0.83	0.77	9,980	12,074	GMU
512-bit variant, single-message architectures							
1	Keccak	x1	2.46	2.30	8,526	3,471	GMU
2	JH	x1 (MEM)	1.52	1.43	5,011	3,288	GMU
	SHA-2	x1	1.07	1.00	2,128	1,995	GMU
3	Groestl	/2(v) (P/Q)	0.68	0.64	5,262	7,763	GMU
4	Skein	x4	0.61	0.57	2,438	4,006	GMU
5	BLAKE	/2(h)	0.42	0.39	2,928	6,977	GMU

As shown in Table 9, for Stratix III and 256-bit variants o the algorithms, the ranking remains almost the same as in Virtex 5 and Virtex 6. Keccak is the only candidate that outperforms SHA-2. Groestl, BLAKE, Skein have efficient pipelined architectures, with almost identical throughput to area ratio. For the 512-bit variants, Keccak and JH are the only candidates that outperform SHA-2 in both categories. In the category of all architectures, Skein has almost identical performance as SHA-2, and Groestl almost identical performance to BLAKE. For the single-message architectures, the differences between the algorithms increase, and the ranking of the candidates becomes the same as in case of Virtex 5 and Virtex 6.

The results for Stratix IV, shown in Table 10, are very similar to those for Stratix III. The primary difference is that for the 512-bit variants of the algorithms, and single-message architectures, Skein jumps ahead of Groestl. For the 512-bit variants and all architectures, Groestl lags behind BLAKE, and becomes least efficient algorithm.

In Figs. 10 and 11, *normalized* throughput to area ratios (i.e., ratios for each algorithm, divided by the respective ratios for SHA-2) are summarized for all four investigated FPGA families. A logarithmic scale is used on the X-axis. These graphs demonstrate quite good agreement between the results for two FPGA families from the same vendor, and quite substantial differences between FPGA families from two different vendors, Xilinx and Altera. In particular, normalized results for Virtex 5 and Virtex 6 seem to be

quite well correlated, with normalized results for Virtex 5 consistently higher than the corresponding normalized values for Virtex 6. These smaller normalized values for Virtex 6 come primarily from the fact that SHA-2 is implemented more efficiently in Virtex 6. On the other hand, in terms of absolute values of the throughput to area ratios, these ratios are consistently higher, as expected, in Virtex 6 for all investigated algorithms (as demonstrated by Tables 7 and 8).

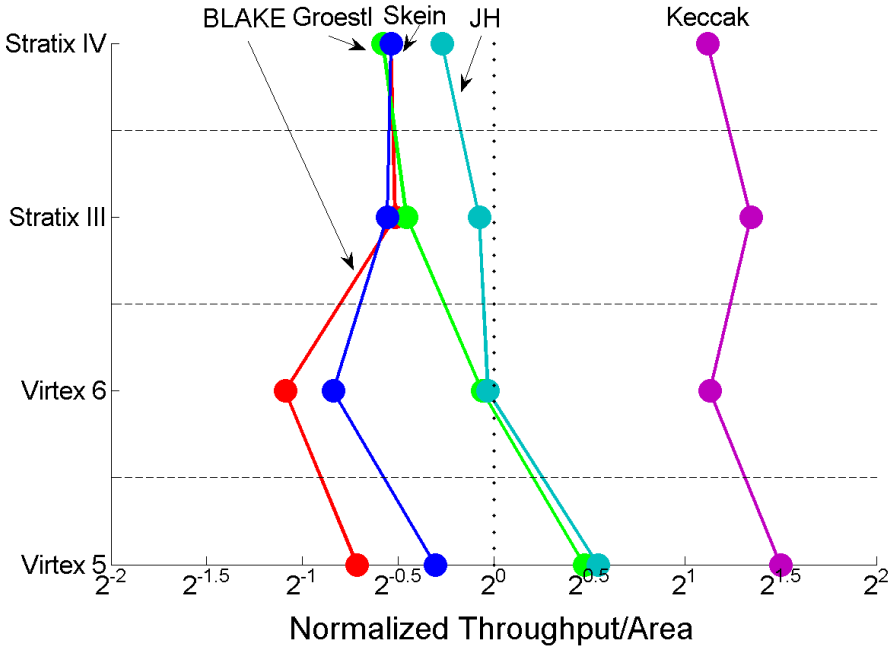
The results for Altera families seem to differ from the results for Xilinx families in several important aspects. In particular, for the 256-bit variants and single-message architectures (see Fig. 10b), the performance gap between JH and Groestl widens in Altera families, while on the other hand BLAKE catches up with Skein. BLAKE, Skein, and Groestl, all benefit from pipelining in Altera families (see Fig. 10a). Additionally, these three algorithms have almost the same normalized throughput to area ratio for both Altera families. For the 512-bit variants of the algorithms and all architectures, Stratix IV seem to be particularly unsuitable for Groestl, which drops from the 3rd position in Xilinx families, to the last 5th position in Stratix IV (see Fig. 11a).

Table 10: Ranking of SHA-3 candidates for 256-bit variant in Altera Stratix IV FPGA. Designs **without padding**, implemented using Altera Quartus II v.11.1 (unless explicitly stated otherwise using *).

Rank	Algorithm	Architecture	Throughput/Area ((Mbit/s)/Slices)	Normalized Throughput/Area	Throughput (Mbit/s)	Area (ALUTs)	Source
256-bit variant, all architectures							
1	Keccak	x1	4.07	2.17	14,401	3,541	GMU
	SHA-2	x1	1.87	1.00	1,798	959	GMU
2	JH	x2-PPL2 (MEM)	1.56	0.83	9,772	6,259	GMU
3	Skein	x4-PPL5	1.30	0.69	7,676	5,925	GMU
4	BLAKE	x1-PPL4	1.29	0.69	8,063	6,271	GMU
5	Groestl	x1-PPL4 (P+Q)	1.25	0.67	16,141	12,933	GMU
256-bit variant, single-message architectures							
1	Keccak	x1	4.07	2.17	14,401	3,541	GMU
	SHA-2	x1	1.87	1.00	1,798	959	GMU
2	JH	x1 (MEM)	1.52	0.81	4,876	3,218	GMU*
3	Groestl	/2(v) (P/Q)	0.95	0.51	3,701	3,906	GMU
4	Skein	x4	0.66	0.35	2,621	3,968	GMU*
5	BLAKE	/2(h)	0.66	0.35	2,337	3,543	GMU
512-bit variant, all architectures							
1	Keccak	x1-PPL2	2.52	2.11	10,816	4,295	GMU
2	JH	x1 (MEM)	1.56	1.31	5,139	3,294	GMU
	SHA-2	x1	1.19	1.00	2,378	1,996	GMU
3	Skein	x4-PPL5	1.13	0.95	6,972	6,185	GMU
4	BLAKE	x1-PPL4	0.92	0.77	11,075	12,082	GMU
5	Groestl	x1-PPL4 (P+Q)	0.81	0.68	20,580	25,407	GMU
512-bit variant, single-message architectures							
1	Keccak	x1	2.26	1.89	7,825	3,467	GMU
2	JH	x1 (MEM)	1.56	1.31	5,139	3,294	GMU
	SHA-2	x1	1.19	1.00	2,378	1,996	GMU
3	Skein	x4	0.68	0.57	2,736	4,015	GMU
4	Groestl	/2(v) (P/Q)	0.65	0.54	4,989	7,724	GMU
5	BLAKE	/2(h)	0.48	0.40	3,318	6,971	GMU

- Results obtained using Altera Quartus II v.10.1

a) 256-bit variant, all architectures



b) 256-bit variant, single-message architectures

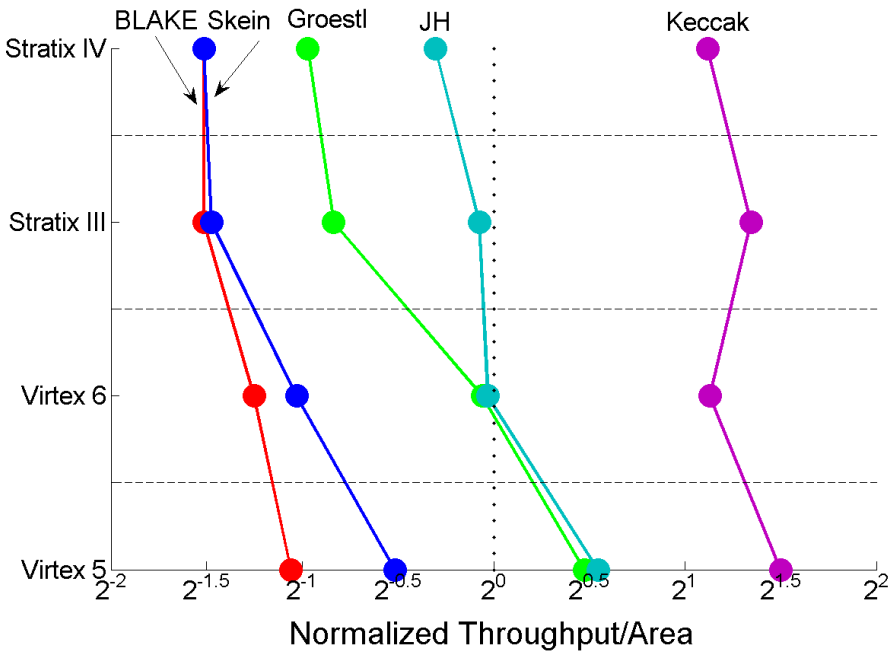
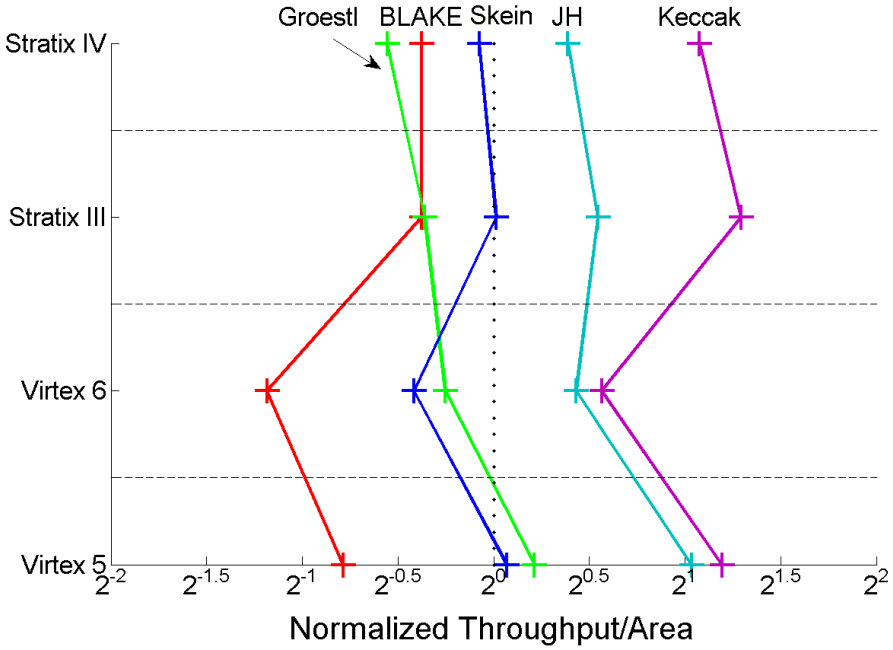


Fig. 10. Rankings of SHA-3 finalists in terms of the Normalized Throughput/Area Ratio for
a) all architectures (including single-message and multi-message architecture),
b) single-message architectures.
256-bit variants of algorithms. Designs **without padding**. Based on the **best results** submitted to the ATHENa Database of Results [3] as of June 1, 2012.

a) 512-bit variant, all architectures



b) 512-bit variant, single-message architectures

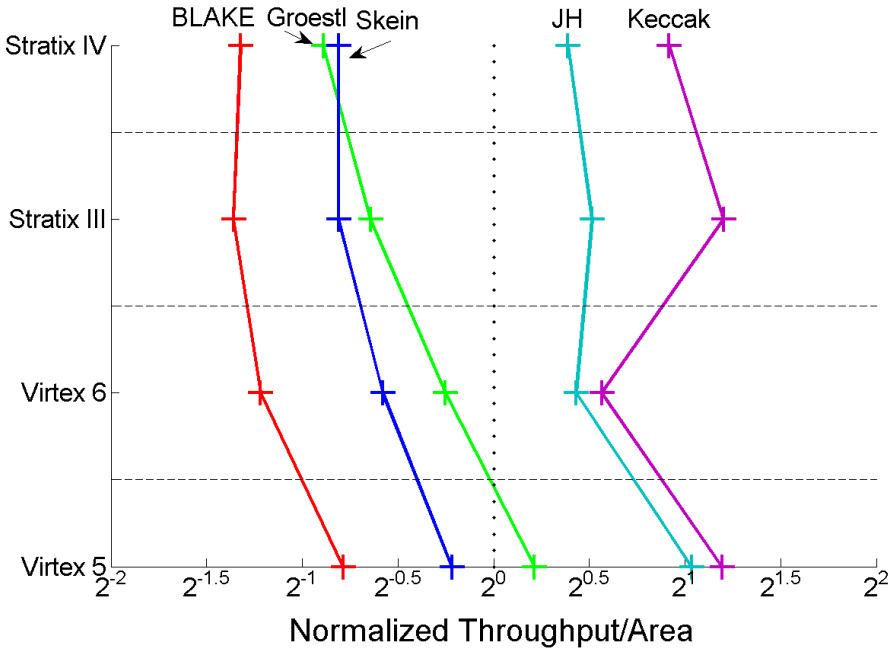


Fig. 11. Rankings of SHA-3 finalists in terms of the Normalized Throughput/Area Ratio for
a) all architectures (including single-message and multi-message architecture),
b) single-message architectures.
512-bit variants of algorithms. Designs **without padding**. Based on the **best results** submitted to the ATHENa Database of Results [3] as of June 1, 2012.

9. Correlation between FPGA results and ASIC results

The number of hardware architectures of SHA-3 candidates implemented in ASICs to date is very small compared to the number of architectures implemented in FPGAs. Typically, only the best non-pipelined architectures for the 256-bit variants of the SHA-3 finalists are reported [18,19,20]. At the same time, multiple applications use ASICs as a primary way of implementing cryptographic transformations, and this trend is likely to continue in the future. Therefore, it is very interesting to see, whether there exist any strong correlation between results obtained for the ASIC and FPGA implementations of the same architectures. In our experiment, performed in collaboration with the group from ETH Zurich, we have implemented selected architectures for all SHA-3 candidates and SHA-2 using standard-cell CMOS 65nm UMC ASIC technology (UMC65LL) offered through Europractice MPW services, and using a 65 nm high-performance Altera FPGA family Stratix III. The selected architectures included the following non-pipelined architectures: basic iterative architectures, x1, for Keccak and SHA-2, basic iterative architecture with round constants computed on the fly, x1 (OTF), for JH, basic iterative parallel architecture of Groestl (P+Q), horizontally folded two times architecture of BLAKE, /2(h), and the unrolled 4 times architecture of Skein, x4.

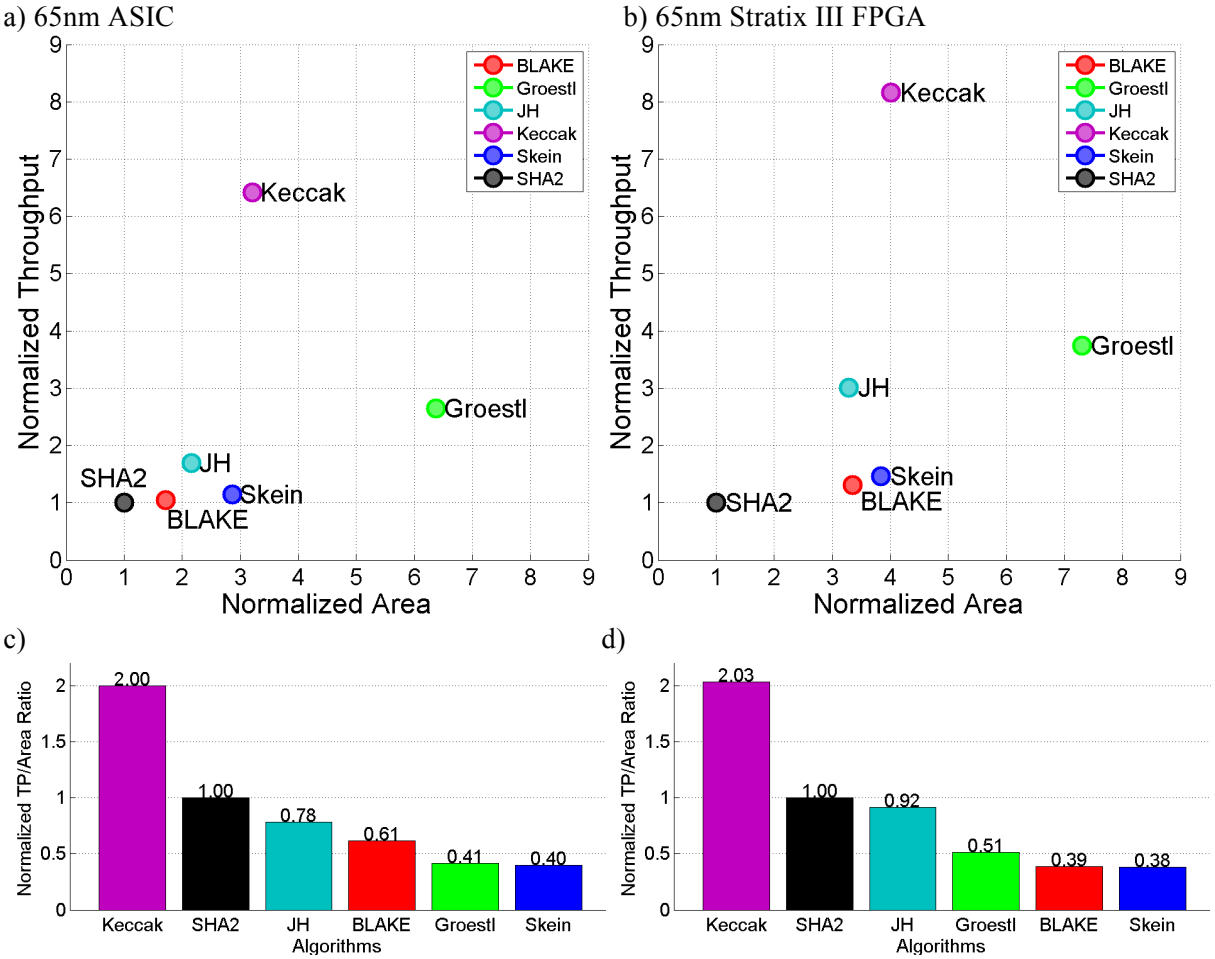


Fig. 12. Correlation between results for 65nm ASIC and 65nm Altera Stratix III FPGA. Normalized throughput vs. normalized area for a) ASIC, b) Stratix III FPGA. Normalized throughput to area ratio for c) ASIC, d) Stratix III FPGA.

All architectures have been designed for the 256-bit variants of the functions, without padding units, and with wide input and output interface (512 or 1088 bits at the input and 256 bits at the output). Exactly the same VHDL source codes have been synthesized, mapped, placed and routed using both technologies. The results, normalized to the results for SHA-2, are presented in Fig. 12. A very good correlation between normalized throughput and normalized area in both technologies have been observed. Ranking in terms of throughput is identical in both technologies. In terms of area, the biggest difference is a relatively smaller area of BLAKE in ASIC technology. In Stratix III FPGAs, JH and BLAKE have almost the same area, in ASIC BLAKE is about 20% smaller than JH.

The biggest difference appears in terms of the throughput to area ratio, where BLAKE moves from the 4th position in tie with Skein in Stratix III to the 3rd position, ahead of Groestl for ASIC. Overall correlation is however very good and indicates that evaluations using Altera FPGAs are likely to give similar results to the evaluations using ASICs fabricated using equivalent technology. Interestingly, similar comparison using Xilinx Virtex 5 FPGAs results in much worse correlation.

10. Results for Implementations using Embedded Resources of FPGAs

The study of the impact of embedded resources on hardware implementations of the SHA-3 candidates has been performed for both non-pipelined (single-message) and pipelined (multi-message) architectures, with padding unit.

The influence of embedded resources on the performance of non-pipelined architectures of all 14 Round 2 candidates has been reported first in [37]. Here, we narrow down this study to five SHA-3 finalists, and we implement all of them using the most efficient non-pipelined and pipelined architectures with padding unit.

The numerical effect of using embedded resources in Xilinx Virtex 5 and Altera Stratix III in *non-pipelined* architectures is summarized in Tables 11-14. In Tables 11 and 12, we have tabulated results for all logic-only implementations, and the corresponding implementations with embedded resources. The relative improvement in terms of Throughput, Area (in CLB slices for Virtex 5, and ALUTs for Stratix III), and Throughput to Area ratio is reported.

None of the final candidates can take advantage of DSP units, because none of them uses multiplication. An attempt to use DSP units for the implementation of a 64-bit addition in Skein, resulted in very inefficient implementations in both Virtex 5 and Stratix III. Similarly, an attempt to use DSP units for the implementation of addition in BLAKE resulted in quite inefficient implementations (not reported in Tables 11 and 12).

Block memories can be used in implementations of BLAKE, Groestl, JH, and Keccak. Based on the results presented in Tables 11 and 12, BLAKE and Groestl benefit most from using embedded memories, with the improvement in the Throughput to Area ratio higher for BLAKE in Virtex 5 FPGAs, and higher for Groestl in Altera Stratix III FPGAs. In Fig. 13, we show the implementation of the Permute transformation using block memories in BLAKE. In Fig. 16, we demonstrate the use of block memories to implement Groestl using a T-box based architecture [36,37].

A relatively small drop (< 15%) in throughput was observed for majority of the candidates. The three exceptions were JH on Virtex 5, with the drop of about 33%, and Skein on Virtex 5 and Stratix III, with the drop of approximately 40%. Skein specifically suffered from the lack of support for a 64-bit addition in DSP units of Virtex 5 and Stratix III.

Area, measured in the number of reconfigurable logic resources (CLB slices for Virtex 5 and ALUTs for Stratix III), decreased the most for BLAKE and Groestl, with BLAKE benefiting more on Virtex 5, and Groestl benefiting more on Stratix III. In all remaining cases, the decrease was smaller than 20%, and in case of Keccak and Skein implemented on Stratix III FPGAs, the amount of reconfigurable logic even increased.

It should be stressed that any reduction in the amount of reconfigurable logic has been accomplished at the expense of substantial usage of embedded memories, which was the largest in Groestl, followed by

BLAKE, JH, and Keccak. In particular, compared to BLAKE, Groestl used about 4 times more BRAMs in Virtex 5, and about 50 times more memory bits in Stratix III.

In Table 13, only relative improvements in terms of the Throughput, Area, and Throughput to Area ratio are presented.

In Table 14, we summarize the effect of the use of embedded resources on the ranking of the SHA-3 finalists in terms of the Throughput to Area ratio for non-pipelined (single-message) architectures. In Virtex 5 and Stratix III, the initial ranking based on the logic-only implementations is as follows: 1) Keccak, 2) JH, 3) Groestl, 4) Skein, 5) BLAKE. As a result of using embedded resources, Groestl jumps ahead of JH to the second position, and BLAKE jumps ahead of Skein to the fourth position. It should be stressed however, that this improvement can be taken advantage of only if in the given system-on-chip including hash functions, none of the other components of the system relies heavily on block memories.

The results for the best pipelined (multi-message) architectures of all algorithms are summarized in Tables 15-18. The pipelined implementation of BLAKE, x1-PPL4, is presented in Figs. 14 and 15. Two rows of G functions are used, and pipelined registers are placed before the first row, before the second row, and in the middle of each G function. Embedded resources can be used only as a part of the function Permute, which now needs to include four sets of BRAMs, one per each stream of data. For Groestl, two pipelined architectures are used. The x1-PPL2(P+Q) architecture has appeared to be more efficient for Virtex 5, while the x1-PPL4(P+Q) architecture more efficient for Stratix III. The latter of these two architectures, x1-PPL4(P+Q), is shown in Fig. 17. Pipelined registers are introduced before the AddP and AddQ operations, after the T-boxes, after the modified network of XORs, and in the feedback loop. Similarly, for Skein, two different architectures have been used. The x4-PPL2 architecture is most efficient for Virtex-5, whereas the x4-PPL5 architecture is most efficient for Stratix III.

As shown in Table 15, for Virtex 5, only Groestl and JH benefit significantly from using embedded resources, and they both reach an improvement of 17.5% in terms of the throughput to area ratio. JH benefits primarily from the increase in the circuit throughput, while Groestl from the decrease in the number of CLB slices. The improvement for BLAKE is negligible (less than 3%), and Keccak and Skein do not benefit at all.

As shown in Table 16, for Stratix III, the improvement for Groestl is much higher (in the range of 94%), and improvement for JH much smaller (less than 6%). BLAKE and Skein have the worse throughput to area ratio after introducing embedded resources, and Keccak shows a negligible improvement. Thus, only Groestl, when implemented using a pipelined architecture on Stratix III, can benefit from using embedded resources.

In Table 17, only relative improvements in terms of the Throughput, Area, and Throughput to Area ratio are presented.

In Table 18, we summarize the effect of the use of embedded resources on the ranking of the SHA-3 finalists in terms of the Throughput to Area ratio for pipelined (multi-message) architectures. In Virtex 5, the initial ranking based on the logic-only implementations is as follows: 1) Keccak, 2) Groestl, 3) JH, 4) Skein, 5) BLAKE. The use of embedded resources does not change this ranking. In Stratix III, the initial ranking based on the logic-only implementations is as follows: 1) Keccak, 2) JH, 3) Groestl, 4) BLAKE, 5) Skein. As a result of using embedded resources, Groestl jumps ahead of JH to the second position, and the remaining order remains the same.

Table 11: The effect of the embedded resources on the performance of 5 Round 3 SHA-3 candidates on Virtex 5 FPGA for **non-pipelined** architectures. Notation: Tp – throughput, Tp/Area – Throughput to Area Ratio, Δ [%] – relative *improvement* in the Throughput, Area, and Throughput to Area ratio as a result of using embedded resources in the hash unit. The relative change in the throughput to area ratio has been marked in **bold**. Xilinx PlanAhead 13.1 and ATHENa were used to generate optimized results after place and route for the embedded architectures. The better of the two results (in terms of the Throughput to Area ratio) was reported below.

Arch	Tp	Area			Tp/Area (Slices)
		Slices	BRAMs	DSPs	
BLAKE-256					
/2(h)-PAD	2266	1860	0	0	1.22
/2(h)-PAD-Emb	2088	802	12	0	2.60
Δ [%]	-7.9%	56.9%	N/A	N/A	**113.1%**
Groestl-256 (P/Q)					
x1-PAD	6572	2020	0	0	3.25
x1-PAD-Emb	5735	1343	49	0	4.27
Δ [%]	-12.7%	33.5%	N/A	N/A	**31.4%**
JH-256 (MEM)					
x1-PAD	4543	1001	0	0	4.53
x1-PAD-Emb	3099	842	5	0	3.68
Δ [%]	-32.8%	7.6%	N/A	N/A	**-18.8%**
Keccak-256					
x1-PAD	12745	1375	0	0	9.27
x1-PAD-Emb	11663	1254	1	0	9.30
Δ [%]	-0.6%	1.7%	N/A	N/A	**0.3%**
Skein-256					
x4-PAD	3127	1245	0	0	2.51
x4-PAD-Emb	1907	1040	0	32	1.83
Δ [%]	-39.0%	16.5%	N/A	N/A	**-27.1%**

Table 12: The effect of the embedded resources on the performance of 5 Round 3 SHA-3 in Stratix III FPGA for **non-pipelined** architectures. Notation: Tp – throughput, Tp/Area – Throughput to Area Ratio, Δ [%] – relative *improvement* in the Throughput, Area, and Throughput to Area ratio as a result of using embedded resources in the hash unit. The relative change in the throughput to area ratio has been marked in **bold.**

Arch	TP	Area			TP/Area (ALUTs)
		ALUTs	membits	DSPs	
BLAKE-256					
/2(h)-PAD	2206	3660	0	0	0.60
/2(h)-PAD-Emb	1880	2008	12288	0	0.94
Δ [%]	-14.8%	45.1%	N/A	N/A	**56.7%**
Groestl-256 (P/Q)					
x1(P/Q)-PAD	6140	7564	0	0	0.81
x1(P/Q)-PAD-Emb	6082	2862	655,360	0	2.13
Δ [%]	-0.9%	62.2%	N/A	N/A	**163.0%**
JH-256 (MEM)					
x1-PAD	5024	3383	0	0	1.49
x1-PAD-Emb	4801	3108	15,744	0	1.55
Δ [%]	-4.4%	8.1%	N/A	N/A	**4.0%**
Keccak-256					
x1-PAD	14624	4060	0	0	3.60
x1-PAD-Emb	13902	4111	2048	0	3.38
Δ [%]	-4.9%	-1.3%	N/A	N/A	**-6.1%**
Skein-256					
x4-PAD	2494	3960	0	0	0.63
x4-PAD-Emb	1463	5203	0	128	0.28
Δ [%]	-41.3%	-31.4%	N/A	N/A	**-55.6%**

Table 13: Change in the results between the logic-only implementation (without DSP units and Block RAMs) and the implementation using these embedded resources for **non-pipelined** architectures. The respective columns represent: ΔThroughput [%] - Relative Improvement in Throughput, ΔReconfigurable Logic [%] - Relative Reduction in the amount of Reconfigurable Logic, ΔTp/Reconfigurable Logic [%] - Relative Improvement in Throughput/Reconfigurable Logic Ratio.

Algorithm	Δ Throughput [%]		Δ Reconfigurable logic [%]		ΔTp/Reconfigurable logic [%]	
	Virtex 5	Stratix III	Virtex 5	Stratix III	Virtex 5	Stratix III
BLAKE	-7.9%	-14.8%	56.9%	45.1%	113.1%	56.7%
Groestl	-12.7%	-0.9%	33.5%	62.2%	31.4%	163.0%
JH	-32.8%	-4.4%	7.6%	8.1%	-18.8%	4.0%
Keccak	-0.6%	-4.9%	1.7%	-1.3%	0.3%	-6.1%
Skein	-39.0%	-41.3%	16.5%	-31.4%	-27.1%	-55.6%

Table 14: Change in the ranking based on the throughput to area ratio between the logic-only implementations (without DSP units and Block RAMs) and the implementations with embedded resources for **non-pipelined** architectures. Values given in bold represent the best results for a given algorithm and FPGA family, and the improvement column represents a relative improvement compared to the logic-only implementation achieved using the best of the two implementations.

Algorithm & Architecture	Virtex 5			Stratix III		
	Logic-only	With embedded resources	Improvement	Logic-only	With embedded resources	Improvement
Keccak: x1	9.27	**9.30**	**0.3%**	**3.60**	3.38	0.0%
JH: x1 (MEM)	**4.53**	3.68	0.0%	1.49	**1.55**	**4.0%**
Groestl: x1 (P/Q)	3.25	**4.27**	**31.4%**	0.81	**2.13**	**163.0%**
Skein: x4	**2.51**	1.83	0.0%	**0.63**	0.28	0.0%
BLAKE: /2(h)	1.22	**2.60**	**113.1%**	0.60	**0.94**	**56.7%**

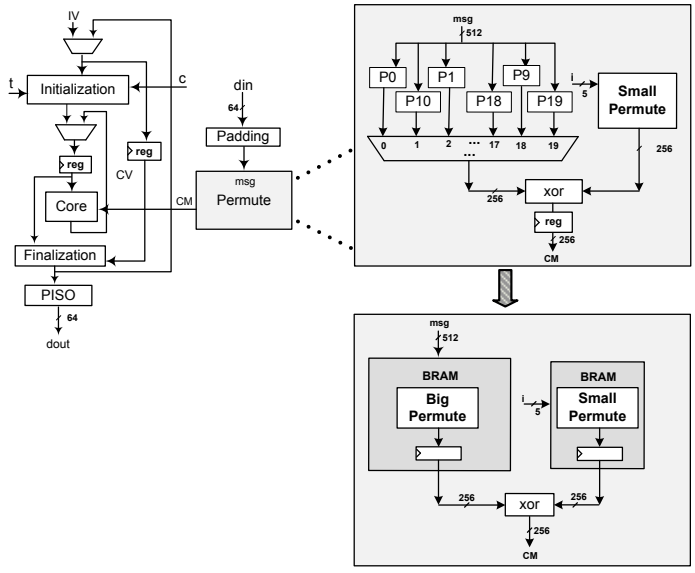


Fig. 13. BLAKE. Transformation of the datapath from the logic-only implementation to the implementation using embedded resources for **non-pipelined architecture**.

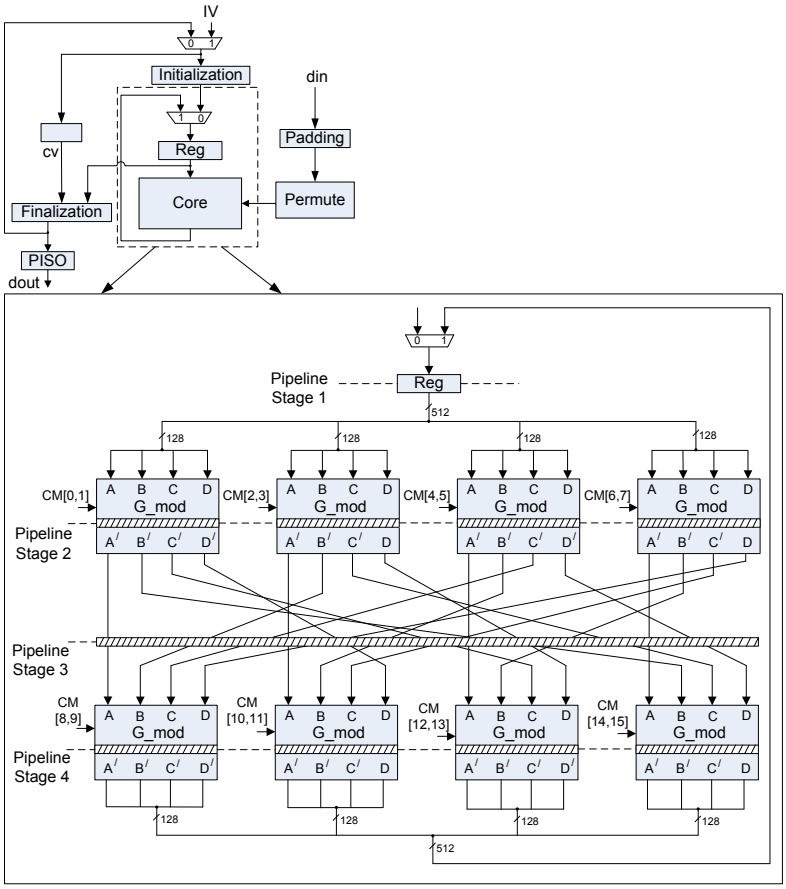


Fig. 14. BLAKE. Block diagram of the pipelined implementation of BLAKE-256, x1-PPL4. The Permute unit can be implemented using either logic-only approach, or using block memories.

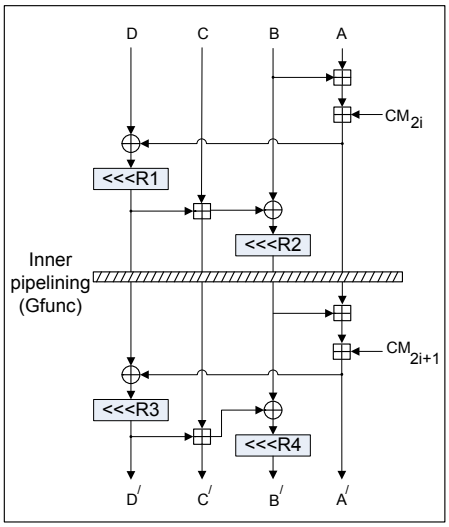


Fig. 15. BLAKE. G-function, represented as G_mod in Fig. 9b, with one inner-pipelining register.

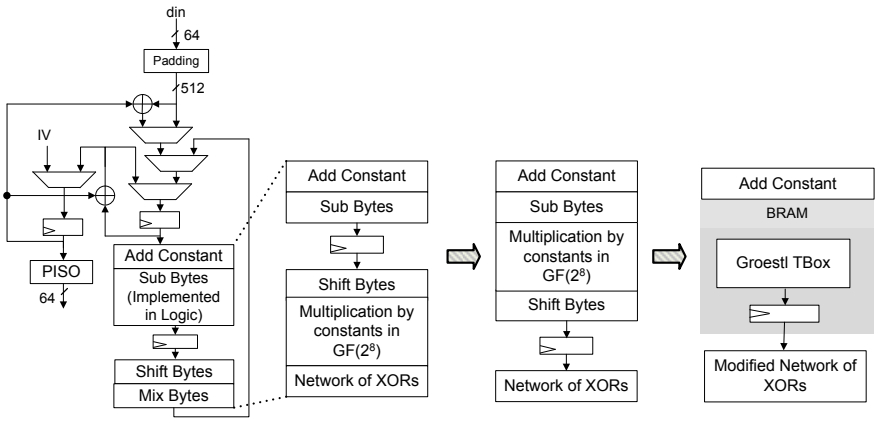


Fig. 16. Groestl. Transformation of the datapath from the S-Box based logic-only implementation to the T-Box based implementation using embedded resources for **non-pipelined** architecture.

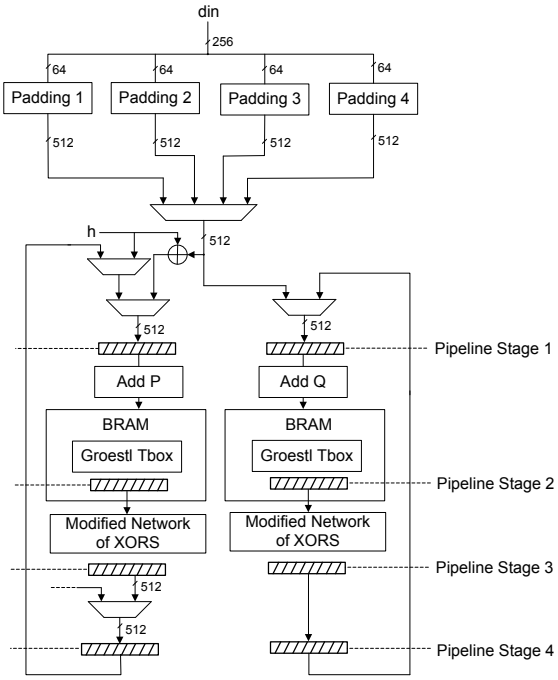


Fig. 17. Groestl. Block diagram of the pipelined implementation of Groestl, x1-PPL4, based on the T-box architecture. T-box is implemented using block memories, as shown in Fig. 16.

Table 15: The effect of the embedded resources on the performance of 5 Round 3 SHA-3 in Virtex 5 FPGA for **pipelined** architectures. Notation: Tp – throughput, Tp/Area – Throughput to Area Ratio, Δ [%] – relative *improvement* in the Throughput, Area, and Throughput to Area ratio as a result of using embedded resources in the hash unit. The relative change in the throughput to area ratio has been marked in **bold**. Xilinx PlanAhead 13.1 and ATHENa were used to generate optimized results after place and route for the embedded architectures. The better of the two results (in terms of the Throughput to Area ratio) was reported below.

Arch	TP	Area			TP/Area (Slices)
		Slices	BRAMs	DSPs	
BLAKE-256					
x1-PPL4-PAD	7510	3526	0	0	2.13
x1-PPL4-PAD-Emb	5314	2427	50	0	2.19
Δ [%]	-29.2%	31.2%	N/A	N/A	**2.8%**
Groestl-256 (P+Q)					
x1-PPL2-PAD	13382	3172	0	0	4.22
x1-PPL2-PAD-Emb	11051	2223	99	0	4.97
Δ [%]	-17.4%	29.9%	N/A	N/A	**17.8%**
JH-256 (MEM)					
x2-PPL2-PAD	5879	2056	0	0	2.86
x2-PPL2-PAD-Emb	7041	2099	10	0	3.35
Δ [%]	19.8%	-2.1%	N/A	N/A	**17.1%**
Keccak-256					
x1-PPL2-PAD	12523	2123	0	0	5.90
x1-PPL2-PAD-Emb	11562	2035	1	0	5.68
Δ [%]	-7.7%	-4.1%	N/A	N/A	**-3.7%**
Skein-256					
x4-PPL2-PAD	4873	2030	0	0	2.40
x4-PPL2-PAD-Emb	2725	1910	0	48	1.43
Δ [%]	-44.1%	5.9%	N/A	N/A	**-40.4%**

Table 16: The effect of the embedded resources on the performance of 5 Round 3 SHA-3 in Stratix III FPGA for **pipelined** architectures. Notation: Tp – throughput, Tp/Area – Throughput to Area Ratio, Δ [%] – relative *improvement* in the Throughput, Area, and Throughput to Area ratio as a result of using embedded resources in the hash unit. The relative change in the throughput to area ratio has been marked in **bold.**

Arch	TP	Area			TP/Area (ALUTs)
		ALUTs	membits	DSPs	
BLAKE-256					
x1-PPL4-PAD	7787	6657	0	0	1.17
x1-PPL4-PAD-Emb	5101	5185	49152	0	0.98
Δ [%]	-34.5%	22.1%	N/A	N/A	**-15.9%**
Groestl-256 (P+Q)					
x1-PPL4-PAD	16903	13261	0	0	1.28
x1-PPL4-PAD-Emb	15591	6297	1310720	0	2.48
Δ [%]	-7.8%	52.5%	N/A	N/A	**94.2%**
JH-256 (MEM)					
x2-PPL2-PAD	9804	6339	0	0	1.55
x2-PPL2-PAD-Emb	9270	5672	16384	0	1.63
Δ [%]	-5.4%	10.5%	N/A	N/A	**5.6%**
Keccak-256					
x1-PPL2-PAD	16047	5003	0	0	3.21
x1-PPL2-PAD-Emb	16563	4984	2048	0	3.32
Δ [%]	3.2%	0.4%	N/A	N/A	**3.6%**
Skein-256					
x4-PPL5-PAD	6869	6068	0	0	1.13
x4-PPL5-PAD-Emb	1263	9912	0	384	0.13
Δ [%]	-81.6%	-63.3%	N/A	N/A	**-88.8%**

Table 17: Change in the results between the logic-only implementation (without DSP units and Block RAMs) and the implementation using embedded resources for **pipelined** architectures. The respective columns represent: ΔThroughput [%] - Relative Improvement in Throughput, Δ Reconfigurable Logic [%] - Relative Reduction in the amount of Reconfigurable Logic, ΔTp/Reconfigurable Logic [%] - Relative Improvement in Throughput/Reconfigurable Logic Ratio.

Algorithm	Δ Throughput [%]		Δ Reconfigurable logic [%]		Δ Tp/Reconfigurable logic [%]	
	Virtex 5	Stratix III	Virtex 5	Stratix III	Virtex 5	Stratix III
BLAKE	-29.2%	-34.5%	31.2%	22.1%	2.8%	-15.9%
Groestl	-17.4%	-7.8%	29.9%	52.5%	17.8%	94.2%
JH	19.8%	-5.4%	-2.1%	10.5%	17.1%	5.6%
Keccak	-7.7%	3.2%	-4.1%	0.4%	-3.7%	3.6%
Skein	-44.1%	-81.6%	5.9%	-63.3%	-40.4%	-88.8%

Table 18: Change in the ranking based on the throughput to area ratio between the logic-only implementations (without DSP units and Block RAMs) and the implementations with embedded resources for **pipelined** architectures. Values given in bold represent the best results for a given algorithm and FPGA family, and the improvement column represents a relative improvement compared to the logic-only implementation achieved using the best of the two implementations.

Algorithm & Architecture	Virtex 5			Stratix III		
	Logic-only	With embedded resources	Improvement	Logic-only	With embedded resources	Improvement
Keccak: x1-PPL2-PAD	**5.90**	5.68	0.0%	3.21	**3.32**	**3.6%**
Groestl: x1-PPL2-PAD/ x1-PPL4-PAD*	4.22	**4.97**	**17.8%**	1.28	**2.48**	**94.2%**
JH: x2-PPL2-PAD	2.86	**3.35**	**17.1%**	1.55	**1.63**	**5.6%**
Skein: x4-PPL2-PAD/ x4-PPL5-PAD **	**2.40**	1.43	0.0%	**1.13**	0.13	0.0%
BLAKE: x1-PPL4-PAD	2.13	**2.19**	**2.8%**	**1.17**	0.98	0.0%

* Groestl x1-PPL2-PAD is implemented on Virtex 5 whereas Groestl x1-PPL4-PAD is implemented on Stratix III.
** Skein x4-PPL2-PAD is implemented on Virtex 5 whereas Skein x4-PPL5-PAD is implemented on Stratix III.

11. Results for Short Messages

In all previous sections, the Throughput is understood as the throughput for long messages, and does not take into account the time taken for reading the very first block of the message, initialization, finalization, and writing a hash value to the output memory. To be exact, we define Throughput for all *single-message architectures* using the following formula:

$$Thr = Thr_{long} = \frac{b \cdot N}{T \bullet (Htime(N+1) - HTime(N))} \quad (3)$$

where b is a message block size, characteristic for each hash function (as defined in the function specification, and shown in Table 20), $HTime(N)$ is a total number of clock cycles necessary to hash an N-block message, and T is a clock period, different and characteristic for each hardware implementation of a specific hash function.

All our designs follow the same interface, described in detail in []. This interface has the parameter:

- w = the width of the input data bus, *din*, and the output data bus, *dout*. These buses are independent of each other, and both have the width w. In our implementations, w=64 for all algorithms and algorithm variants, except SHA-2-256, where w=32.

The general formula for the time necessary to hash N blocks of the message can be written in the following form:

$$HTime(N) = C_{INIT} + C_{IN} + C_{BLOCK} \cdot N + C_{FINAL} + C_{OUT} \quad (4)$$

In this formula:

- C_{INIT} is the number of clock cycles necessary to establish communication with the source of data (typically, Input FIFO) and read the length of the message (in our formulas we assume that the length of the message is smaller than 2^{w-1}, and $C_{INIT}=2$).
- C_{IN} is the number of clock cycles required to load the very first block of the message. $C_{IN} = b/w$.
- C_{BLOCK} is the number of clock cycles required to process one block of the message.
- C_{FINAL} is the number of clock cycles required for the finalization. We assume that only one finalization is required per entire message (if the finalization needs to be repeated for every block of the message, its number of clock cycles is included in C_{BLOCK}).
- C_{OUT} is the number of clock cycles required to write hash value to the destination circuit (typically Output FIFO). $c_{OUT}=output\_size/w$.

Values of the constants C_{BLOCK} and C_{FINAL} are specific to each algorithm and the algorithm variant. Values of these constants for various single-message architectures are summarized in Table 19.

The combined throughput for the pipelined architecture with n pipeline stages, derived from an arbitrary single-message architecture, *ARCH*, and processing n long messages in parallel, is given by the following equations:

For Keccak, Groestl (P+Q), and Skein:

$$Thr_{ARCH-PPLn-long} = \frac{T_{ARCH}}{T_{ARCH-PPLn}} \cdot Thr_{ARCH-long} = \frac{f_{ARCH-PPLn}}{f_{ARCH}} \cdot Thr_{ARCH-long} \quad (5)$$

For BLAKE, Groestl (P/Q), and JH:

$$Thr_{ARCH-PPLn-long} = \frac{T_{ARCH}}{T_{ARCH-PPLn}} \cdot (1 + \frac{(n-1)}{n \cdot cycles\_per\_round + 1}) \cdot Thr_{ARCH-long} \quad (6)$$

where T_{ARCH} and f_{ARCH} are the clock period and the clock frequency of the single-message architecture *ARCH*, $T_{ARCH-PPLn}$ and $f_{ARCH-PPLn}$ are the clock period and the clock frequency of the pipelined architecture, *ARCH-PPLn*, derived from the architecture ARCH, $Thr_{ARCH-long}$ is the throughput of the architecture ARCH for long messages, and *cycles_per_round* is the number of clock cycles per round in the architecture *ARCH*.

Thus, for Keccak, Groestl (P+Q), and Skein, the throughput increases proportionally to the increase in the clock frequency. For BLAKE, Groestl (P/Q), and JH, the dependence is somewhat more complicated, as given by Eq. (6), and as a result, the throughput increases by a slightly larger factor.

The formulas for short messages, as a function of the message length, m, are as follows:

$$Thr_{short}(m) = \frac{m}{T \bullet HTime(N(m))} \quad (7)$$

$$N(m) = \left\lceil \frac{m + min\_pad}{b} \right\rceil \quad (8)$$

$$Relative\_Effective\_Throughput(m) = RET(m) = \frac{Thr_{short}(m)}{Thr_{long}} \quad (9)$$

Table 19: The number of clock cycles required to process one block of data (C_{BLOCK}), and the number of clock cycles required for the finalization (C_{FINAL}) for various single-message architectures.

	256-bit variant		512-bit variant	
Algorithm	C_{BLOCK}	C_{FINAL}	C_{BLOCK}	C_{FINAL}
BLAKE				
x1	15	0	17	0
/2(h)	29	0	33	0
/4(h)	57	0	65	0
/4(h)/4(v)-m	240	0	272	0
Groestl				
x1 (P+Q)	10	10	14	14
x1 (P/Q)	21	21	29	29
/2(v) (P+Q)	20	20	28	28
/2(v) (P/Q)	42	42	58	58
/4(v) (P+Q)	40	40	56	56
/4(v) (P/Q)	84	84	116	116
/8(v) (P+Q)	80	80	112	112
/8(v) (P/Q)	168	168	232	232
JH				
x1 (MEM, OTF)	43	0	43	0
/2(v) (MEM, OTF)	85	0	85	0
/8(v)-m (MEM)	688	0	688	0
x2 (MEM, OTF)	22	0	22	0
Keccak				
x1	24	0	24	0
/8(v)-m	233	0	233	0
Skein				
x1	73	73	73	73
x4	19	19	19	19
x8	10	10	10	10
SHA-2				
x1	65	0	81	0

Values of the constants *b* and *min_pad* are specific to each algorithm and the algorithm variant, and are summarized in Table 20.

The Relative Effective Throughput, RET(m), can be further transformed to the formula (10):

$$RET(m) = \frac{m}{\left\lceil \frac{m + min\_pad}{b} \right\rceil \cdot b} \cdot \frac{C_{BLOCK} \cdot N(m)}{C_{OVR} + C_{BLOCK} \cdot N(m)} = \frac{m}{m_{after-padding}} \cdot \frac{C_{BLOCK} \cdot N(m)}{C_{OVR} + C_{BLOCK} \cdot N(m)} \quad (10)$$

In this formula, $C_{OVR} = C_{INIT} + C_{IN} + C_{FINAL} + C_{OUT}$, where values of these constants are defined above.

As can be seen from these formulas, the slow down compared to the throughput for long messages is caused by two factors:

1. The difference between the size of the message before and after padding.
2. Overhead associated with the initialization, loading the first message block, the one-time hash function finalization, and writing a hash value to the output.

The first of these two effects is inherent for processing of small messages. The latter effect can be either minimized or even eliminated completely in our architectures by overlapping processing of two subsequent messages.

The graphs shown in Figs. 18-20, represent values of the Relative Effective Throughput, as a function of the message size *m*, under the worst case condition, where either only a single message is processed, or there is no overlap between processing of multiple messages of size *m*.

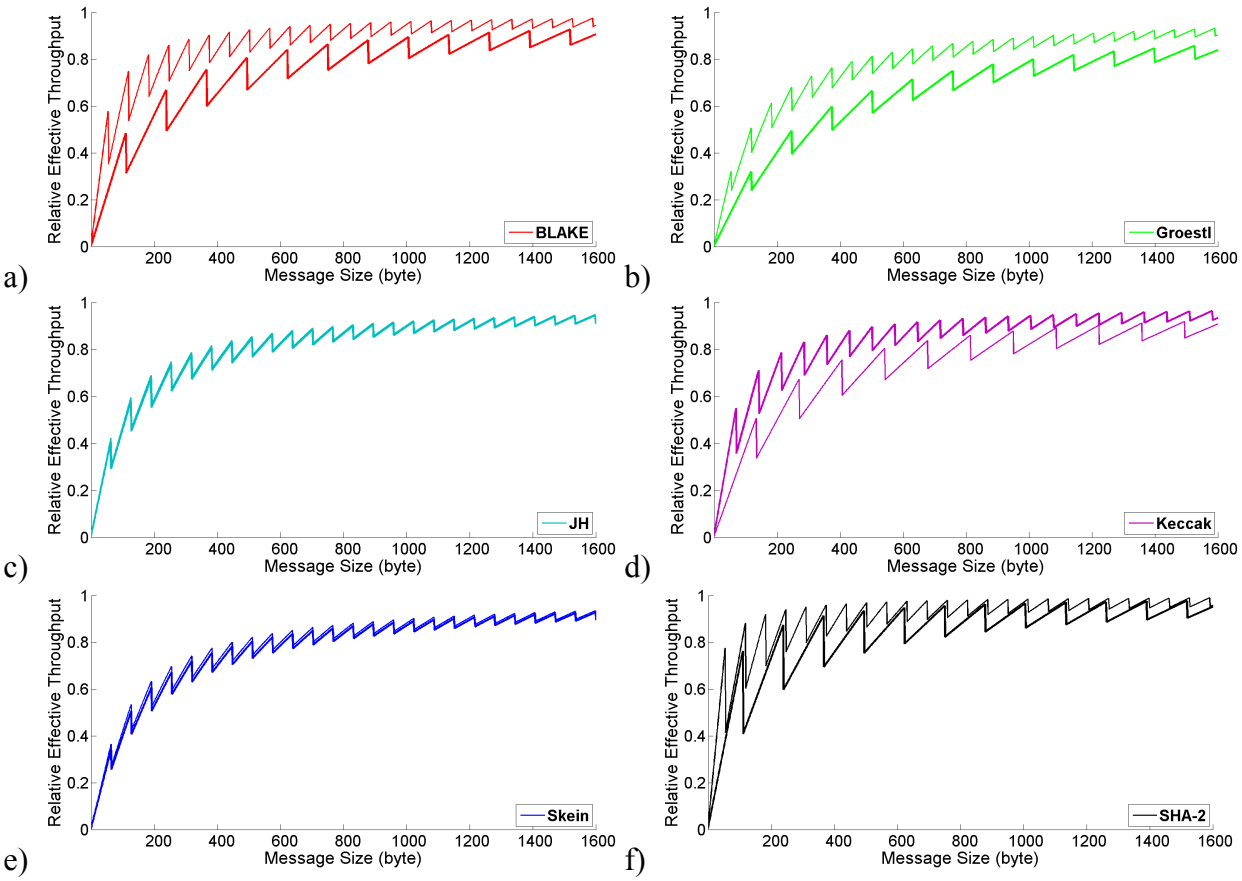


Fig. 18. Relative effective throughput vs. message size for short messages up to 1,600 bytes for 256 and 512-bit variants of all SHA-3 Candidates and SHA-256 in Virtex 5. Darker lines denote 512-bit variants.

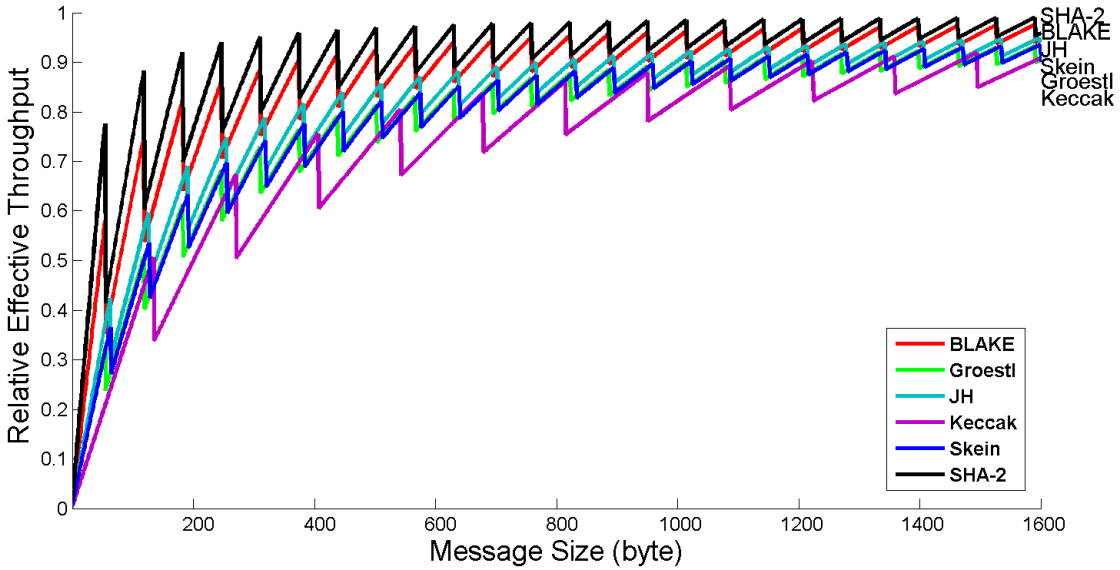


Fig. 19. Relative effective throughput vs. message size for short messages up to 1,600 bytes. 256-bit variants of all SHA-3 Candidates and SHA-256.

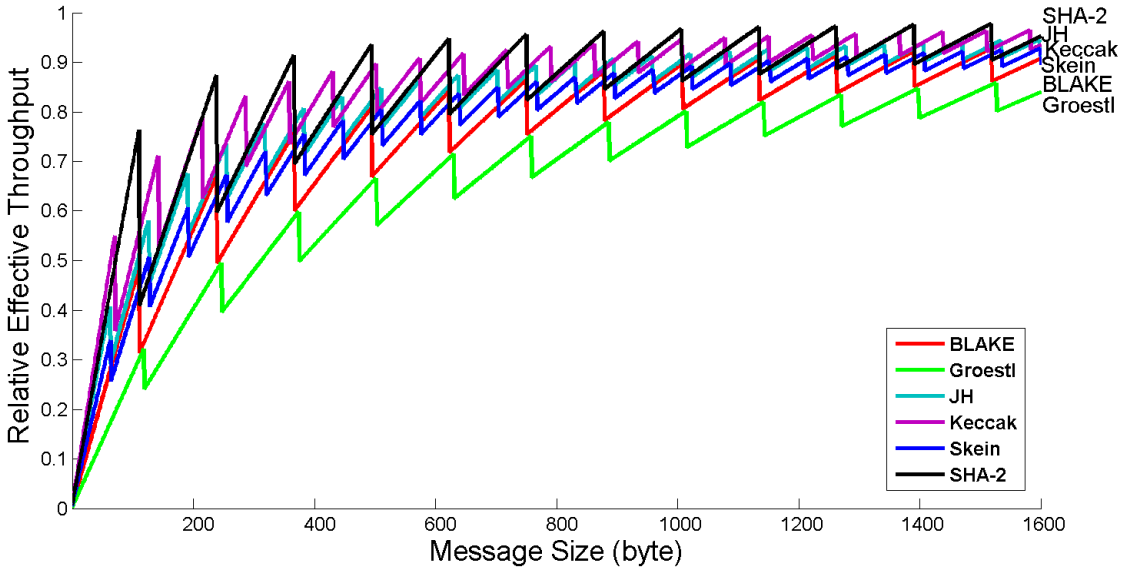


Fig. 20. Relative effective throughput vs. message size for short messages up to 1,600 bytes. 512-bit variants of all SHA-3 Candidates and SHA-256.

Table 20: Block size and minimum padding length (*min_pad*) for all SHA-3 finalists and SHA-2.

	256-bit variant		**512-bit variant**	
Algorithm	Block size, *b*	*min_pad*	Block size, *b*	*min_pad*
BLAKE	512	66	1024	130
Groestl	512	65	1024	65
JH	512	512	512	512
Keccak	1088	2	576	2
Skein	512	0	512	0
SHA-2	512	65	1024	129

In Fig. 18, Relative effective throughputs, are shown individually, for each of the 5 SHA-3 finalists and SHA-2, for message sizes up to 1600 bytes. The interval between consecutive local maximums is equal to the message block size, *b* (in bytes). The message size corresponding to the first local maximum is equal to: *b-min_pad* (in bytes) for all functions, except JH, where it is equal to *b*.

For JH and Skein, the dependence does not depend on the hash output size. For BLAKE, Groestl, and SHA-2, the relative effective throughput is smaller for the 512-bit variant compared to the 256-bit variant. In Keccak, the reverse relation is true. This is because the message block size for the 512-bit variant (576 bits) is smaller than the message block size for the 256-bit variant (1088 bits).

In Fig. 19, the diagrams for the 256-bit variants of all investigated hash functions are combined together. Based on this figure, the effect of the message size on the relative effective throughput is the smallest in case of SHA-2 and the largest in case of Keccak. This relation holds because Keccak has a significantly larger value of the block size, *b* (1088 bits = 136 bytes), and the difference *b-min_pad* (1086 bits) than any other candidate. Additionally, SHA-2 has the smallest effective overhead, as it can start processing a block of message without loading it first to the hash unit.

In Fig. 20, the equivalent combined diagram is shown for the 512-bit variants of all investigated functions. This time, the worst performers are BLAKE and Groestl, the algorithms with the largest values of the block size, *b* (1024 bits = 128 bytes), and the largest values of *b-min_pad*. SHA-2 has also a large block size (1024 bits), but it benefits from a small overhead associated with reading data, and a large number of clock cycles, which makes the influence of the overhead clock cycles negligible.

12. Conclusions

In this paper, we have performed a systematic investigation of *high-speed* hardware architectures for the five final SHA-3 candidates. The investigated architectures were based on the concepts of the basic iterative architecture, horizontal folding, vertical folding, unrolling, pipelining, and parallel processing using multiple independent units. Each architecture was implemented using four high-performance FPGA families: Virtex 5 and Virtex 6 from Xilinx, and Stratix III and Stratix IV from Altera. Based on the obtained results, we have identified the most efficient hardware architecture for each of the investigated algorithm, based on the best throughput to area ratio.

For majority of algorithms and algorithm variants, the best architecture is specific to particular FPGA family, or at least to FPGAs of a particular vendor (Xilinx and Altera). Inner-round pipelining is the most efficient method of increasing throughput to area ratio compared to the basic iterative architecture, assuming the availability of multiple streams of data. In case of Skein, pipelining must be preceded by unrolling by a factor of 4. In case of JH, pipelining is efficient only on Altera FPGAs, and only if preceded by unrolling by a factor of 2. In case of Keccak, pipelining seems to improve only throughput, but not the throughput to area ratio. For majority of algorithms, an optimum number of pipeline stages is specific to both an algorithm variant and an FPGA family.

The results for all investigated functions, and the most successful architectures have been summarized using the comprehensive throughput vs. area graphs. Then, the results for the best architectures in terms of the throughput to area ratio, have been used to determine two rankings per each FPGA family:

a) ranking for single-message (non-pipelined) architectures, and
b) ranking for all architectures (non-pipelined and pipelined).

The latter case (ranking for all architectures) seems to be more realistic, as in majority of practical applications, the number of data streams available in parallel is substantial.

The effect of padding units on the results of ranking was investigated, and was found to be negligible. The results for circuits with padding units are provided in the paper as these circuits represent the most complete, ready to use solution. The results for circuits without padding units are provided as well in order to allow comparison with results from other groups, as the majority of designs reported in the literature and in the database of results [3] do not include padding unit.

The effect of the typical message size on the ranking of candidates was also studied, and was shown to be highly dependent on the specific message size, and relatively small for a typical distribution of packet sizes used in the Internet security protocols.

The most efficient single-message architectures[1], identified using results for Altera FPGAs, were ported to the 65nm standard-cell UMC ASIC technology. An integrated circuit containing these architectures for the 5 SHA-3 finalists and SHA-2 has been fabricated. The obtained results have been compared with FPGA results for Xilinx and Altera families. Results for the Altera family, Stratix III, have been demonstrated to show a good correlation with the aforementioned ASIC results. Thus, it is quite likely, that even an extended study, involving a larger number of architectures, conducted using ASICs, would give similar results to the evaluations using Altera FPGAs fabricated using an equivalent fabrication process.

Our study has revealed that Keccak is the only candidate that *consistently* outperforms SHA-2 for all considered FPGA families and two hash function variants (with 256-bit and 512-bit output). The typical improvement factor in terms of the throughput to area ratio is at least 2. The only drawback of this function appears to be its limited suitability for folding. Additionally, no pipelined architecture, improving the throughput to area ratio has been demonstrated in our study.

JH performs better than SHA-2 consistently, across all investigated FPGA families, for 512-bit variants of both functions. For the 256-bit variants, JH shows an improvement for only one out of four FPGA families. Interestingly, for majority of FPGA families, JH is the most efficient in its basic iterative architecture, and is not very suitable for either folding or inner-round pipelining.

Groestl outperforms SHA-2 in terms of the throughput to area ratio for only one out of four FPGA families, Virtex 5. Even then, this advantage is reached only for the relatively large area of about 3000 CLB slices for Groestl-256 and about 6000 CLB slices for Groestl-512. Although Groestl appears to be very suitable for vertical folding, the very nature of this technique causes that the decrease in area is typically accompanied by an even greater decrease in speed. The only exception to this rule is the quasi-pipelined Groestl architecture (P/Q) folded vertically by a factor of /2(v)(P/Q). If embedded resources are abundant, Groestl can take advantage of block memories of modern FPGAs, in order to decrease its usage of reconfigurable logic, and increase the throughput to area ratio (with area representing reconfigurable logic only). In such cases, Groestl jumps ahead of JH to the second position in ranking, at least for Altera FPGAs. Additionally, Groestl is the only finalist that can efficiently share resources with AES, when both algorithms are implemented on the same FPGA, thus reducing the cost of the respective system-on-chip.

Skein is the only finalist that can substantially benefit from unrolling. It is also the fastest for the pipelined versions of the 4x unrolled architecture, and is the only algorithm that can be pipelined efficiently up to 10 times. Skein performs particularly well compared to other algorithms for the 512-bit variants of hash functions, in the category of all architectures, on Altera FPGAs. In this ranking, Skein is exceptionally third, and achieves performance comparable to SHA-2-512. For the 256-bit variants of hash functions, and for the 512-bit variant on Virtex 6, Skein lags behind SHA-2.

BLAKE is the algorithm with the highest flexibility, and the largest number of potential architectures. It can be easily folded horizontally and vertically by factors of two and four. It can also be easily pipelined even in the folded architectures. It is also the only algorithm that has a relatively efficient architecture that is smaller than the basic iterative architecture of SHA-2. Finally, BLAKE, similarly to Groestl, can benefit substantially from using embedded block memories of both Xilinx and Altera FPGAs. At the same time, BLAKE ranks last or second to last in almost all FPGA rankings (concerning high-speed architectures) based on the throughput to area ratio.

Our future work will include experimental testing of selected high-speed architectures of the SHA-3 finalists, using high-performance FPGA boards based on Xilinx and Altera FPGAs, equipped with high-speed communication interface, such as PCI Express.

---

[1] the only exception was Groestl where the basic iterative architecture, x1(P+Q), was used instead of slightly more efficient vertically folded architecture /2(v)(P/Q).

Acknowledgments

The authors would like to thank Frank Gürkaynak and other members of the Microelectronics Designs Center at ETH Zurich for providing us with the post-layout results of ASIC implementations of GMU cores. We also thank Ambarish Vyas for preliminary results regarding hash cores with padding units [39], and Rajesh Velegalati for extensive help with multiple ATHENa runs.

References:

[1] SHA-3 Contest, http://csrc.nist.gov/groups/ST/hash/sha-3/index.html
[2] ATHENa Project Website, http://cryptography.gmu.edu/athena/.
[3] ATHENa Database of FPGA Results, available at http://cryptography.gmu.edu/athenadb/fpga_hash
[4] ATHENa Database of ASIC Results, available at http://cryptography.gmu.edu/athenadb/asic_hash
[5] GMU Source Codes of SHA-3 Candidates, http://cryptography.gmu.edu/athena/index.php?id=source_codes
[6] eBASH Website, http://bench.cr.yp.to/results-sha3.html
[7] SHA-3 Hardware Implementations, http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations
[8] A. Akin, A. Aysu, O.C. Ulusel, and E. Savas, "Efficient Hardware Implementation of High Throughput SHA-3 Candidates Keccak, Luffa and Blue Midnight Wish for Single- and Multi-Message Hashing," The Second SHA-3 Candidate Conference, Santa Barbara, CA, Aug. 23-24, 2010.
[9] N. At, J.-L. Beuchat, and I. San, "Compact Implementation of Threefish and Skein on FPGA," 5th IFIP International Conference on New Technologies, Mobility and Security, IEEE Press, 2012, available at http://www.cipher.risk.tsukuba.ac.jp/~beuchat/Publications/
[10] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. O'Neill, and W.P. Marnane, "FPGA Implementations of the Round Two SHA-3 Candidates," The Second SHA-3 Candidate Conference, Santa Barbara, CA, Aug. 23-24, 2010.
[11] B. Baldwin, A. Byrne, L. Lu, M. Hamilton, N. Hanley, M. O'Neill, and W.P. Marnane, "FPGA Implementations of the Round Two SHA-3 Candidates," 20th International Conference on Field Programmable Logic and Applications, Milano, Italy, Aug. 31st - Sep. 2nd, 2010, available at http://www.ucc.ie/en/crypto/SHA-3Hardware/NISTSHA-3/BaldwinHashFPL2010paperUpdate.pdf
[12] D.J. Bernstein and T. Lange, "The New SHA-3 Software Shootout," The Third SHA-3 Candidate Conference, Washington, D.C., March 22-23, 2012.
[13] J.-L. Beuchat, E. Okamoto, T. Yamazaki, "Compact Implementations of BLAKE-32 and BLAKE-64 on FPGA," International Conference on Field-Programmable Technology, FPT 2010, Beijing, China, Dec. 2010, pp. 170-177, available at http://www.cipher.risk.tsukuba.ac.jp/~beuchat/Publications/
[14] Z. Chen, X. Guo, A. Sinha, and P. Schaumont, "Data-Oriented Performance Analysis of SHA-3 Candidates on FPGA Accelerated Computers," Design, Automation and Test in Europe, DATE 2011, Grenoble, France, March 2011, pp. 1650-1655.
[15] K. Gaj, E. Homsirikamol, and M. Rogawski, "Fair and Comprehensive Methodology for Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs," Proc. Cryptographic Hardware and Embedded Systems workshop, CHES 2010, Santa Barbara, Aug. 2010, pp. 264-278.
[16] K. Gaj, J.P. Kaps, V. Amirineni, M. Rogawski, E. Homsirikamol, B.Y. Brewster, "ATHENa – Automated Tool for Hardware EvaluatioN: Toward Fair and Comprehensive Benchmarking of Cryptographic Hardware Using FPGAs," 20th International Conference on Field Programmable Logic and Applications, Milano, Italy, Aug. 31st - Sep. 2nd, 2010.
[17] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont, "Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations," The Second SHA-3 Candidate Conference, Santa Barbara, CA, Aug. 23-24, 2010.
[18] X. Guo, M. Srivistav, S. Huang, D. Ganta, M. Henry, L. Nazhandali, and P. Schaumont, "Pre-silicon Characterization of NIST SHA-3 Final Round Candidates", 14th Euromicro Conference on Digital System Design Architectures, Methods and Tools, DSD 2011, Oulu, Finland, Aug. 31-Sep. 2, 2011.
[19] X. Guo, M. Srivistav, S. Huang, D. Ganta, M.B. Henry, L. Nazhandali, and P. Schaumont, "ASIC Implementations of Five SHA-3 Finalists," Proc. Design, Automation and Test in Europe, DATE 2012, Dresden, Germany, March 2012, pp. 1006-1011.
[20] F. K. Gürkaynak, K. Gaj, B. Muheim, E. Homsirikamol, C. Keller, M. Rogawski, H. Kaeslin, and J.-P. Kaps, "Lessons Learned from Designing a 65nm ASIC for Evaluating Third Round SHA-3 Candidates," The Third SHA-3 Candidate Conference, Washington D.C., March 2012.

[21] L. Henzen, P. Gendotti, P. Guillet, E. Pargaetzi, M. Zoller and F.K. Gurkaynak, "Developing a Hardware Evaluation Method for SHA-3 Candidates," Proc. Cryptographic Hardware and Embedded Systems workshop, CHES 2010, Santa Barbara, Aug. 2010, pp. 248-263.
[22] L. Henzen, J.-P. Aumasson, W. Meier, and R.C.-W. Phan, "VLSI Characterization of the Cryptographic Hash Function BLAKE," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19 , issue 10, Oct. 2011, pp. 1746 - 1754, available at http://131002.net/data/papers/HAMP10.pdf
[23] E. Homsirikamol, M. Rogawski, and K. Gaj, "Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs," Cryptology ePrint Archive: Report 2010/445.
[24] E. Homsirikamol, M. Rogawski, and K. Gaj, "Throughput vs. Area Trade-offs in High-Speed Architectures of Five Round 3 SHA-3 Candidates Implemented Using Xilinx and Altera FPGAs," LNCS 6917, Cryptographic Hardware and Embedded Systems workshop, CHES 2011, Nara, Japan, Sep. 28-Oct. 1, pp. 491-506.
[25] K. Järvinen, "Sharing Resources Between AES and the SHA-3 Second Round Candidates Fugue and Grøstl," The Second SHA-3 Candidate Conference, Aug. 23-24, 2010.
[26] B. Jungk and J. Apfelbeck, "Area-Efficient FPGA Implementations of the SHA-3 Finalists," 2011 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2011, Cancun, Mexico, Nov. 30-Dec. 2, 2011.
[27] B. Jungk, "Evaluation Of Compact FPGA Implementations For All SHA-3 Finalists," The Third SHA-3 Candidate Conference, Washington, D.C., March 22-23, 2012.
[28] E. B. Kavun and T. Yalcin, "On the Suitability of SHA-3 Finalists for Lightweight Applications," The Third SHA-3 Candidate Conference, Washington, D.C., March 22-23, 2012.
[29] J.-P. Kaps, P. Yalla, K.K. Surapathi, B. Habib, S. Vadlamudi, S. Gurung, and J. Pham, "Lightweight Implementations of SHA-3 Candidates on FPGAs," 12th International Conference on Cryptology, Indocrypt 2011, Chennai, Dec. 11-14, 2011.
[30] J.-P. Kaps, P. Yalla, K.K. Surapathi, B. Habib, S. Vadlamudi, S. Gurung, "Lightweight Implementations of SHA-3 Finalists on FPGAs," The Third SHA-3 Candidate Conference, Washington, D.C., March 22-23, 2012.
[31] S. Kerckhof, F. Durvaux, N. Veyrat-Charvillon, F. Regazzoni, G. Meurice de Dormale, F.-X. Standaert, "Compact FPGA Implementations of the Five SHA-3 Finalists," 10th Smart Card Research and Advanced Application Conference, CARDIS 2011, Leuven, Belgium, Sep. 14-16, 2011.
[32] M. Knežević, K. Kobayashi, J. Ikegami, S. Matsuo, A. Satoh, U. Kocabaş, J. Fan, T. Katashita, T. Sugawara, K. Sakiyama, I. Verbauwhede, K. Ohta, N. Homma and T. Aoki, "Fair and Consistent Hardware Evaluation of Fourteen Round Two SHA-3 Candidates," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 20, issue 5, May 2012, pp. 827–840.
[33] K. Latif, M.M. Rao, A. Aziz, and A. Mahboob, "Efficient Hardware Implementations and Hardware Performance Evaluation of SHA-3 Finalists," The Third SHA-3 Candidate Conference, Washington, D.C., March 22-23, 2012.
[34] S. Matsuo, M. Knezevic, P. Schaumont, I. Verbauwhede, A. Satoh, K. Sakiyama, and K. Ota, "How Can We Conduct "Fair and Consistent" Hardware Evaluation for SHA-3 Candidate?" The Second SHA-3 Candidate Conference, 2010, Santa Barbara, CA, Aug. 23-24, 2010.
[35] M. Rogawski and K. Gaj, "Groestl Tweaks and their Effect on FPGA Results," Cryptology ePrint Archive: Report 2011/635.
[36] M. Rogawski and K. Gaj, "A High-Speed Unified Hardware Architecture for the AES and SHA-3 Candidate Grøstl," Proc. 15th Euromicro Conference on Digital System Design, Sep. 5-8, 2012, Cesme, Izmir, Turkey.
[37] R. Shahid, M.U. Sharif, M. Rogawski, and K. Gaj, "Use of Embedded FPGA Resources in Implementations of SHA-3 Candidates," The 2011 International Conference on Field-Programmable Technology, FPT 2011, New Delhi, India, Dec. 12-14, 2011.
[38] S. Tillich, et al. "High-Speed Hardware Implementations of Blake, Blue Midnight Wish, Cubehash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, Shavite-3, SIMD, and Skein. Cryptology ePrint Archive, Report 2009/510.
[39] A. Vyas, "Implementing and Benchmarking of Padding Units and HMAC for SHA-3 Candidates in FPGAs and ASICs," Master's Thesis, George Mason University, Fall 2011.