

Throughput vs. Area Trade-offs in High-Speed Architectures of Five Round 3 SHA-3 Candidates Implemented Using Xilinx and Altera FPGAs*

Ekawat Homsirikamol, Marcin Rogawski, and Kris Gaj

ECE Department, George Mason University, Fairfax, VA 22030, U.S.A.
{ehomsiri, mrogawsk, kgaj}@gmu.edu
<http://cryptography.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 folding, unrolling, and pipelining. 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, and 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 has proven to be an important tie-breaker in the contests for new cryptographic standards. For example, in the AES contest [14], performance in FPGAs and ASICs has played a major role, because all five finalists have been judged to have adequate security, and their performance in hardware varied substantially.

In this paper, we focus on comparing hardware performance of the remaining five final candidates in the SHA-3 contest organized by NIST in the period from 2007 to 2012 [1]. The unique and novel feature of our approach is the investigation of multiple hardware architectures of each algorithm. Our goal is to analyze the entire performance space in terms of the throughput to area trade-offs, for all Round 3 SHA-3 candidates, as well as the current standard, SHA-2. This investigation is very important because the exact requirements on the speed and area of a hash function core depend on a particular application and vary in a

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

wide range. Knowledge of alternative architectures may allow the developer to substantially reduce the relative area of a hash core in a system-on-chip, or move to a substantially less expensive part in case of a stand-alone implementation of a hash core in an FPGA.

We perform our investigation using four high-performance FPGA families from two major vendors: Virtex 5 and Virtex 6 from Xilinx and Stratix III and Stratix IV from Altera. All algorithms have been implemented based on their updated Round 3 specifications, published in January 2011.

2 Previous work

Previous results on comparison of Round 2 SHA-3 candidates in hardware are summarized in [2]. 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 have been reported in [5][8][12][13]. All these papers include 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 [12], 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 [4].

Some of the most interesting low-area implementations of the SHA-3 candidates have been described in [6][7][15]. The most comprehensive studies of the ASIC implementations of the Round 2 SHA-3 candidates are presented in [10][11][16].

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, and at the time of writing, we are not aware of any published reports on the high-speed FPGA implementations of the Round 3 variants of these algorithms.

3 Performance Metrics

Three major performance metrics used in our study are throughput, area, and throughput to area ratio. Throughput is understood as the throughput for long messages, or cumulative throughput for a large number of small messages (where processing and input/output functions overlap in time). The resource utilization in FPGAs is a vector, with coordinates specific to the given FPGA family, e.g.

$$Resource\ Utilization_{Virtex\ 5} = (\#CLB_slices, \#BRAMs, \#DSPs) \quad (1)$$

$$Resource\ Utilization_{Stratix\ III} = (\#ALUTs, \#memory_bits, \#DSPs). \quad (2)$$

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 $\#memory_bits$ is the number of bits placed in dedicated Altera FPGA memories. Taking into account that vectors cannot be easily compared to each other, we have decided to opt out of using any dedicated resources in the hash function implementations used for our comparison. Thus, all coordinates of our vectors, other than the first one have been forced (by choosing appropriate options of the synthesis and implementation tools) to be zero. This way, our resource utilization (further referred to as Area) is characterized using a single number, specific to the given family of FPGAs, namely $\#CLB_slices$ for Xilinx Virtex 5 and Virtex 6, and $\#ALUTs$ in Stratix III and Stratix IV. We believe that the capability of using embedded resources should be treated as a measure of the algorithm flexibility, and should be investigated independently from this study.

4 Investigated Hardware Architectures

A starting point for our exploration of various architectures of hash functions is the basic iterative architecture, shown in Fig. 1a. The characteristic features of this architecture are as follows: a) datapath width = state size (denoted by s), b) one round is performed in a single clock cycle, c) only one message is processed at a time. The minimum block processing time is typically given by (3),

$$T_{block} = (r + f) \cdot T \quad (3)$$

where r is the number of rounds, f is the number of clock cycles required to finalize computations for a block (typically 0 or 1), and T is the minimum clock period. The corresponding throughput is given by (4),

$$Tp = b/T_{block} \quad (4)$$

where b is the size of a message block in bits. We denote the area of this architecture by $Area$. The basic iterative architecture is typically an architecture of choice for high-speed hardware implementations of SHA-1, SHA-2, and SHA-3 candidates.

If a round of a hash function has a symmetric structure, with two or more similar operations performed one after another, horizontal folding is possible. 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. The datapath width stays the same as in the basic iterative architecture, and is equal to the state size, s . The block processing time is given by (5),

$$T_{block-2(h)} = (2 \cdot r + f) \cdot T_{/2(h)} \quad (5)$$

where $T/2 < T_{/2(h)} < T$, ideally $T_{/2(h)} \approx T/2$, and $Area/2 < Area_{/2(h)} < Area$. As a result, the block processing time (and thus also throughput) stays

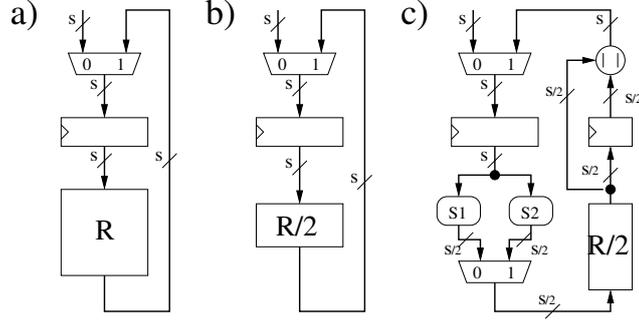


Fig. 1: Three hardware architectures of a hash function: a) basic iterative: $\times 1$, b) folded horizontally by a factor of 2: $/2(h)$, c) folded vertically by a factor of 2: $/2(v)$. R – round, S1, S2 – selection functions.

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, is achieved for the entire round. Other SHA-3 finalists do not demonstrate any similar symmetry.

In case horizontal folding is either not possible or does not achieve the required reduction in area, vertical folding may be attempted. In Fig. 1c, we demonstrate vertical folding by a factor of 2, which we denote 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, $T_{/2(v)} \approx T$. As a result, the block processing time, increases approximately by a factor of two compared to the basic architecture, as shown in the equation below:

$$T_{block- /2(v)} = (2 \cdot r + f) \cdot T_{/2(v)} \approx (2r + f) \cdot T. \quad (6)$$

The area reduction is also smaller than in case of horizontal folding, because of the need for an extra $s/2$ -bit register and multiplexer. As a result the throughput

to area ratio is likely to go 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 missing.

In order to increase the throughput of a hash function, different architectures must be applied. The three common approaches are unrolling, pipelining, and parallel processing. Unrolling is suitable for increasing throughput of a single long message. Pipelining and parallel processing increase the combined data throughput in case of processing multiple messages (e.g., multiple packets) at the same time.

In Fig. 2a, architecture with unrolling by a factor of two is demonstrated. We will denote this architecture by $x2$. The datapath width stays the same as in the basic iterative architecture. 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 single long messages 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. Among the five final SHA-3 finalists, this situation happens only for Skein.

In majority of practical applications of hash functions, the messages that are processed are relatively short (typically smaller than 1500 bytes), and multiple messages (packets) are available for processing by a hashing unit at the same time. 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 at 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. 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.

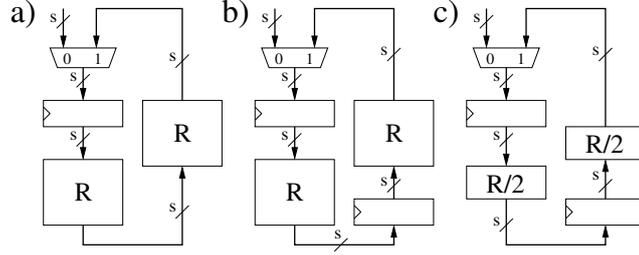


Fig. 2: Three hardware architectures of a hash function a) unrolled by a factor of 2: x2, b) unrolled by a factor of 2 with 2 pipeline stages: x2-PPL2, c) basic iterative with 2 pipeline stages: x1-PPL2.

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 clock period of this architecture is approximately equal to the clock period of the basic iterative architecture, T . Processing a single block takes the same number of clock cycles as in the basic iterative architecture. However, since two blocks belonging to two different messages are processed simultaneously, the combined throughput increases by a factor of two. 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. 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 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 the given FPGA family.

The formulas for the block processing time and the throughput of all aforementioned architectures are summarized in Table 1.

5 Design Methodology and Design Environment

Our designs for the basic, folded, and unrolled architectures use the interface and the communication protocol proposed in [8]. Our designs for the pipelined architectures, use the interface and surrounding logic shown in Fig. 3.

Table 1: Formulas for the time required to process a single message block, T_{block} , and the Throughput, Tp , for 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.

Architecture	Time required to process a single message block	Throughput
Basic iterative, x1	$T_{block} = (r + f) \cdot T$	$Tp = b/T_{block}$
Folded by a factor of k , /k	$T_{block} = (k \cdot r + f) \cdot T$	$Tp = b/T_{block}$
Unrolled by a factor of k , xk	$T_{block} = (r/k + f) \cdot T$	$Tp = b/T_{block}$
Basic iterative with n pipeline stages, x1-PPLn	$T_{block} = (n \cdot r + f) \cdot T$	$Tp = n \cdot b/T_{block}$
Folded by a factor of k with n pipeline stages, /k-PPLn	$T_{block} = (n \cdot k \cdot r + f) \cdot T$	$Tp = n \cdot b/T_{block}$
Unrolled by a factor of k with n pipeline stages, xk-PPLn	$T_{block} = (n \cdot r/k + f) \cdot T$	$Tp = n \cdot b/T_{block}$

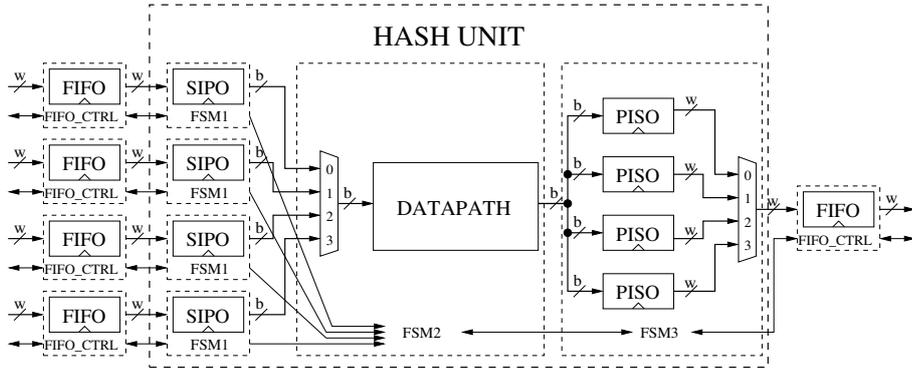


Fig. 3: Interface, high-level block diagram, and surrounding logic of the Hash Unit for the pipelined architecture with four pipeline stages. Notation: SIPO – Serial-In Parallel-Out unit, PISO – Parallel-In Serial-Out unit, w – input/output bus width, $w = 64$ for all investigated algorithms, except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

Input FIFOs serve as packet queues. Each FIFO communicates with the corresponding Serial-In Parallel-Out (SIPO) unit and the associated Finite State Machine 1 (FSM1). FSM 1 is responsible for reading in the next block of data, using b/w clock cycles, 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 candidates submission package.

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

- for Xilinx: Xilinx ISE Design Suite v. 12.4, including Xilinx XST,
- for Altera: Quartus II v. 10.1 Subscription Edition Software.

The generation of a large number of results was facilitated by an open source benchmarking environment, ATHENa (Automated Tool for Hardware EvaluationN) [3][9]. The details of results and selected source codes are available at [3].

6 Results

The results of our implementations are summarized in Figs. 4-9, and in Tables 2 and 3. In Fig. 4, 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 FPGAs.

For BLAKE (see Fig. 4a), the two best architectures in terms of the throughput to area ratio are: $/4(h)/4(v)$, i.e., architecture with horizontal folding by a factor of 4, combined with vertical folding by a factor of 4; and x1-PPL2, i.e., basic architecture with two pipeline stages. The good performance of the former of these two architectures is associated with the significant reduction of the complexity of the BLAKE PERMUTE function as a result of vertical folding by 4. The good performance of the latter is associated with the perfectly symmetric structure of the round, which makes it easy to divide the datapath into two well-balanced pipeline stages. The two less successful architectures include x1 and $/2(h)$ -PPL4. These architectures are not included in our combined graphs shown in Figs. 5-8.

For Groestl (see Fig. 4b), we consider two major architectures: a) parallel architecture, denoted (P+Q), in which Groestl permutations P and Q are implemented using two independent units, working in parallel, and b) quasi-pipeline architecture, 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 [16]. The best architecture overall appears to be the parallel architecture (P+Q) in the basic version, with two pipeline stages, x1-PPL2. Vertical folding by 2 provides quite substantial reduction in area, but at the price of an even greater reduction in throughput. 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 Fig. 4c), 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

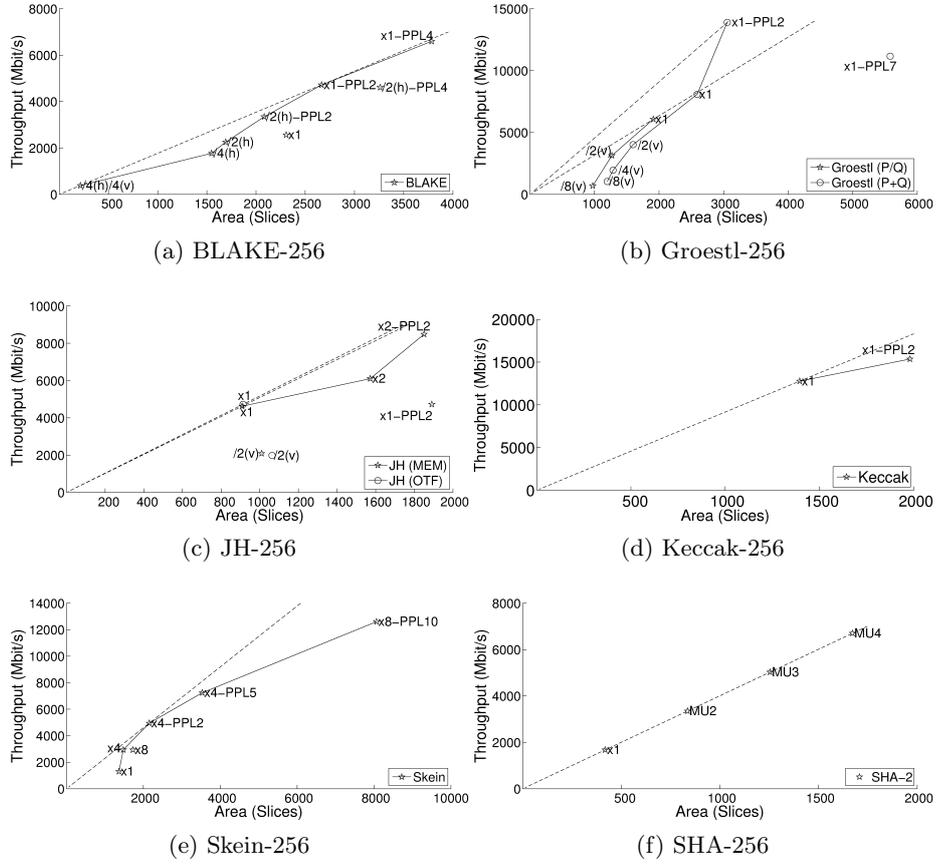


Fig. 4: Throughput vs. Area graphs for multiple architectures of a) BLAKE-256, b) Groestl-256, c) JH-256, d) Keccak-256, e) Skein-256, and f) SHA-256, implemented in Xilinx Virtex 5 FPGAs. Notation: x1 – basic iterative architecture, /k(h) – horizontally folded by a factor of k , /k(v) – folded vertically by a factor of k , 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, MU – multi-unit architecture.

architecture. Vertical folding, 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 is hard to divide into two well balanced pipeline stages. As a result,

the basic iterative architecture remains most efficient in terms of the throughput to area ratio.

For Keccak (see Fig. 4d), neither horizontal nor vertical folding applies. Two pipeline stages increase throughput, but by a factor smaller than the increase in the circuit area.

For Skein (see Fig. 4e), the unrolled by 4 architecture, x4, appears to be significantly more efficient than the basic architecture, x1. At the same time, unrolling by 8 does not give any additional improvement. The best results are obtained by first unrolling basic architecture by a factor of four, and then pipelining the obtained circuit using two pipeline stages. Five pipeline stages have been attempted as well because of an extra addition executed every fourth round, but did not improve the overall throughput to area ratio.

For SHA-2 (see Fig. 4f), 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. 5 and 6. 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.

The results for the 256-bit variants of hash functions, shown in Fig. 5, indicate that the order of the SHA-3 candidates in terms of throughput, for implementations using 1500 or more CLB slices is: 1) Keccak, 2) JH, 3) Groestl, 4) Skein, and 5) BLAKE. Keccak and JH clearly outperform SHA-2, while Groestl becomes faster only with more than 3000 CLB slices. At the same time, only BLAKE and SHA-2 have implementations based on basic iterative architecture and/or folding, with area below 500 CLB slices.

The results for the 512-bit variants of hash functions, shown in Fig. 6, 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), SHA-2 is ranked third, Skein slightly outperforms Groestl (because of the increase in the number of rounds of Groestl from 10 to 14), and BLAKE is a distant sixth.

The performance for Altera devices, shown in Figs. 7 and 8 is somewhat different. For the 256-bit versions of the algorithms, Keccak is the only function that outperforms SHA-2 in terms of the throughput to area ratio. JH is the third in ranking, with two architectures offering the similar ratio as SHA-2. BLAKE, Groestl, and Skein are in tie with each, with Groestl being somewhat disadvan-

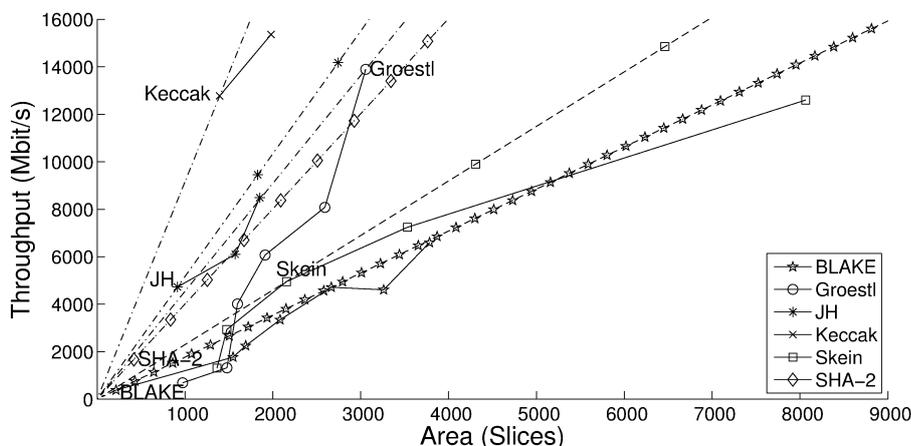


Fig. 5: Combined Throughput vs. Area graph for multiple hardware architectures of the 256-bit variants of BLAKE, Groestl, JH, Keccak, Skein, and SHA-2, implemented in Xilinx Virtex 5 FPGAs.

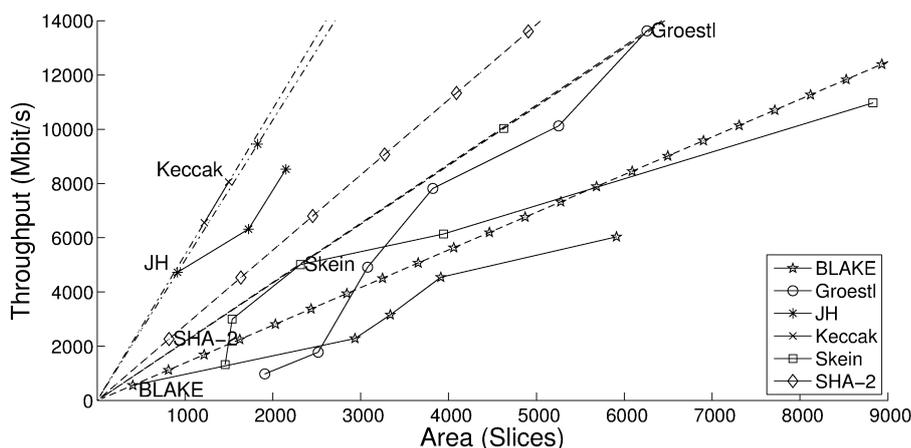


Fig. 6: Combined Throughput vs. Area graph for multiple hardware architectures of the 512-bit variants of BLAKE, Groestl, JH, Keccak, Skein, and SHA-2, implemented in Xilinx Virtex 5 FPGAs.

tagged by approximately twice as large area of its most efficient architecture. For the 512-bit versions of the algorithms (see Fig. 8), Keccak and JH outperform SHA-2, Skein is in tie with SHA-2, Groestl and BLAKE fall significantly 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 the 256-bit variants 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.

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)	381	215	1.77	412	181	2.28	370	915	0.40	378	915	0.41
/4(h)	1770	1547	1.14	1784	888	2.01	1708	3153	0.54	1747	3157	0.55
/2(h)	2253	1691	1.33	1956	1247	1.57	2151	3603	0.60	2302	3605	0.64
/2(h)-PPL2	3346	2083	1.61	3069	1792	1.71	3149	4571	0.69	3471	4570	0.76
x1	2561	2306	1.11	2388	1721	1.39	2195	4745	0.46	2305	4742	0.49
/2(h)-PPL4	4609	3261	1.41	5002	2516	1.99	4894	5080	0.96	5312	5049	1.05
x1-PPL2	4714	2666	1.77	5156	2206	2.34	4487	5420	0.83	4704	5431	0.87
x1-PPL4	6596	3784	1.74	7937	2616	3.03	7524	6273	1.20	8186	6278	1.30
Groestl-256 (P+Q)												
/8(v)	1042	1197	0.87	1161	980	1.18	1103	2716	0.41	1094	2736	0.40
/4(v)	1948	1287	1.51	2289	1134	2.02	2129	4079	0.52	2058	4093	0.50
/2(v)	4014	1598	2.51	4890	1560	3.13	4623	6130	0.75	4349	6073	0.72
x1	8081	2591	3.12	9340	2630	3.55	9608	11122	0.86	9122	11154	0.82
x1-PPL2	13894	3057	4.55	17084	3034	5.63	13793	11727	1.18	13749	11727	1.17
x1-PPL7	11167	5582	2.00	N/A	N/A	N/A	13964	14487	0.96	14392	14470	0.99
Groestl-256 (P/Q)												
/8(v)	691	973	0.71	808	813	0.99	812	2141	0.38	791	2141	0.37
/4(v)	1322	1477	0.89	1687	996	1.69	1401	3660	0.38	1378	3658	0.38
/2(v)	3136	1270	2.47	3301	1074	3.07	3198	4208	0.76	3209	4216	0.76
x1	6072	1912	3.18	4621	1737	2.66	6041	7498	0.81	5586	7287	0.77
JH-256 (MEM)												
/2(v)	2088	1010	2.07	2202	861	2.56	2104	3365	0.63	2066	3377	0.61
x1	4624	909	5.09	5700	847	6.73	5146	3207	1.60	4868	3209	1.52
x1-PPL2	6104	1572	3.88	7176	1382	5.19	6225	5607	1.11	6001	5574	1.08
x2	4728	1891	2.50	4986	1613	3.09	5314	4254	1.25	5378	4262	1.26
x2-PPL2	8487	1851	4.58	8846	1934	4.57	9816	6303	1.56	9522	6259	1.52
JH-256 (OTF)												
/2(v)	1981	1064	1.86	2219	915	2.42	2039	3464	0.59	2010	3469	0.58
x1	4725	914	5.17	5306	1039	5.11	5028	3380	1.49	4965	3383	1.47
Keccak-256												
x1	12777	1395	9.16	11843	1165	10.17	12971	3909	3.32	13159	4129	3.19
x1-PPL2	15362	1980	7.76	16236	1446	11.23	19193	4955	3.87	18610	4953	3.76
x1-PPL4	12652	3849	3.29	13201	2785	4.74	16019	5391	2.97	17913	5402	3.32
Skein-256												
x1	1307	1364	0.96	1382	1127	1.23	1108	3538	0.31	1247	3539	0.35
x4	2937	1476	1.99	3523	1216	2.90	2455	3965	0.62	2621	3968	0.66
x8	2931	1728	1.70	3275	1510	2.17	3178	5586	0.57	3372	5493	0.61
x4-PPL2	4950	2154	2.30	5858	1860	3.15	4273	4421	0.97	4596	4423	1.04
x4-PPL5	7240	3532	2.05	7465	2839	2.63	6772	5920	1.14	7693	5935	1.30
x8-PPL10	12602	8065	1.56	N/A	N/A	N/A	12283	10994	1.12	11378	10996	1.03
SHA-256												
x1	1675	418	4.01	2273	286	7.95	1654	988	1.67	1744	988	1.77

Table 3: Results for the 512-bit variants 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.

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)	563	406	1.39	612	324	1.89	485	1664	0.29	546	1675	0.33
/4(h)	2287	2935	0.78	2709	1936	1.40	2230	6137	0.36	2477	6161	0.40
/2(h)	3159	3337	0.95	3187	2628	1.21	2905	7127	0.41	3288	7128	0.46
/2(h)-PPL2	4544	3912	1.16	4821	3642	1.32	4033	8960	0.45	4780	8962	0.53
x1	3401	3984	0.85	3273	3823	0.86	2947	9251	0.32	3310	9268	0.36
/2(h)-PPL4	6035	5911	1.02	6948	4922	1.41	5535	9698	0.57	7521	9703	0.78
x1-PPL2	6405	5730	1.12	6426	4922	1.31	5549	10616	0.52	6222	10627	0.59
x1-PPL4	3825	7497	0.51	3607	6234	0.58	4952	12100	0.41	5594	12100	0.46
Groestl-512 (P+Q)												
/8(v)	1351	2249	0.60	1484	1837	0.81	1496	5312	0.28	1367	5303	0.26
/4(v)	2533	2263	1.12	2933	2237	1.31	2902	8031	0.36	2692	7945	0.34
/2(v)	4914	3079	1.60	6257	3154	1.98	5985	12295	0.49	5851	12216	0.48
x1	10124	5254	1.93	11566	5106	2.27	12393	21854	0.57	12164	21847	0.56
x1-PPL2	13628	6258	2.18	N/A	N/A	N/A	17050	22570	0.76	17196	22412	0.77
x1-PPL7	12669	11194	1.13	N/A	N/A	N/A	17635	29320	0.60	18395	28976	0.63
Groestl-512 (P/Q)												
/8(v)	984	1908	0.52	1037	1406	0.74	1052	4749	0.22	1010	4744	0.21
/4(v)	1783	2516	0.71	2145	1787	1.20	1855	6000	0.31	1945	6301	0.31
/2(v)	4139	2161	1.92	4442	2172	2.04	4550	7417	0.61	4352	7426	0.59
x1	7819	3821	2.05	8468	3658	2.31	8029	14445	0.56	7944	14461	0.55
JH-512 (MEM)												
/2(v)	2187	1102	1.98	2392	1002	2.39	2199	3621	0.61	2076	3620	0.57
x1	4624	909	5.09	5700	847	6.73	5146	3207	1.60	4868	3209	1.52
x1-PPL2	4610	1973	2.34	5007	1745	2.87	5208	4514	1.15	5261	4527	1.16
x2	6321	1723	3.67	6935	1425	4.87	6309	5769	1.09	6176	5806	1.06
x2-PPL2	8523	2148	3.97	8321	1895	4.39	9704	6335	1.53	9328	6307	1.48
JH-512 (OTF)												
/2(v)	2068	1041	1.99	2287	955	2.39	2099	3701	0.57	2134	3816	0.56
x1	4725	914	5.17	5306	1039	5.11	4912	3548	1.38	4860	3549	1.37
Keccak-512												
x1	6556	1220	5.37	7225	1231	5.87	6859	3477	1.97	6805	3470	1.96
x1-PPL2	8056	1498	5.38	8853	1470	6.02	9836	4431	2.22	9490	4418	2.15
x1-PPL4	7095	3756	1.89	7202	2650	2.72	9440	5175	1.82	9328	5201	1.79
Skein-512												
x1	1325	1457	0.91	1356	1155	1.17	1082	3632	0.30	1229	3631	0.34
x4	2999	1537	1.95	3321	1258	2.64	2398	4074	0.59	2619	4093	0.64
x8	2810	1658	1.70	3113	1591	1.96	3111	5571	0.56	3296	5573	0.59
x4-PPL2	5013	2314	2.17	5649	1818	3.11	4236	4677	0.91	4607	4680	0.98
x4-PPL5	6141	3942	1.56	7664	3209	2.39	6378	6189	1.03	6977	6179	1.13
x8-PPL10	10973	8831	1.24	11982	7323	1.64	10839	11204	0.97	10945	11203	0.98
SHA-512												
x1	2267	818	2.77	3081	592	5.20	2146	2072	1.04	2399	2073	1.16

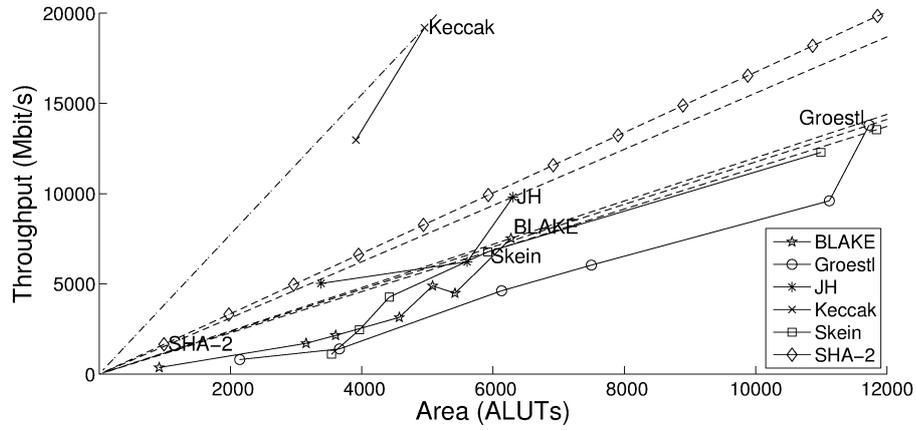


Fig. 7: Combined Throughput vs. Area graph for multiple hardware architectures of the 256-bit variants of BLAKE, Groestl, JH, Keccak, Skein, and SHA-2, implemented in Altera Stratix III FPGAs.

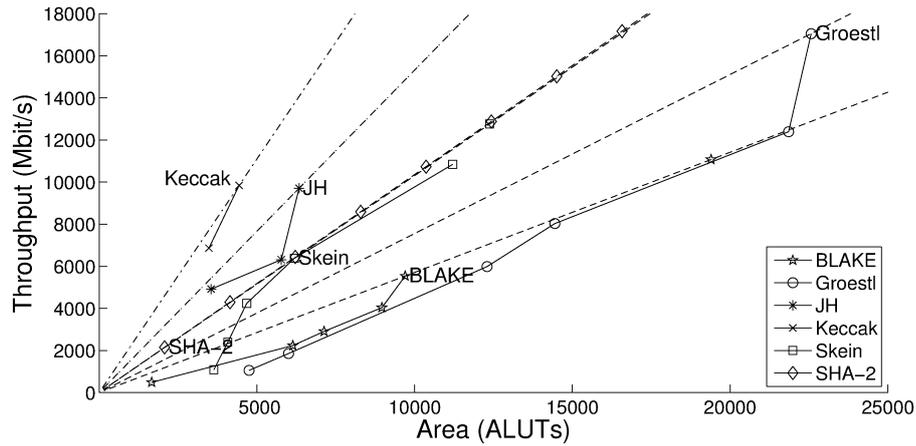


Fig. 8: Combined Throughput vs. Area graph for multiple hardware architectures of the 512-bit variants of BLAKE, Groestl, JH, Keccak, Skein, and SHA-2, implemented in Altera Stratix III FPGAs.

Additionally, we have also performed an initial study on the influence of padding units on the ranking of the candidates. Based on this study, the largest decrease in the throughput to area ratio caused by adding a padding unit to the basic architecture of a SHA-3 candidate has not exceeded 16%. So small variations in this ratio are not likely to affect the overall ranking of the candidates.

7 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.

In case of four out of five candidates (all except JH), the most efficient architecture appeared to be a pipelined architecture. The optimum number of pipeline stages was specific to the algorithm, and was equal to two for Keccak and Groestl, and four for BLAKE. The optimum pipelined architecture for Skein was the architecture with four rounds unrolled, and n pipeline stages, where the optimum value of n was equal to two for Xilinx high-performance FPGAs, and five for Altera high-performance FPGAs.

The results for all investigated functions, and the most successful architectures have been then summarized on the comprehensive throughput vs. area graphs. These graphs have 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 only drawback of this function appears to be that it is not suitable for any kind of folding, and thus requires a quite substantial minimum area (in the range of 1400 CLB slices in Virtex 5) to be implemented in its basic iterative version.

JH performed better than SHA-2 in three out of four scenarios. It was outperformed by SHA-2 only for the 256-bit function variants implemented using Altera FPGAs. Interestingly, JH is most efficient in its basic iterative architecture, and is not suitable for either folding or inner-round pipelining.

Groestl was the only other candidate outperforming SHA-2 in at least one scenario, for the 256-bit variants implemented using Virtex 5. However this advantage was reached only for the relatively large area of about 3000 CLB slices. Although Groestl appeared to be very suitable for vertical folding, the very nature of this technique caused that the decrease in area was accompanied by the very significant decrease in speed.

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 up to 10 times. It performs particularly well compared to other algorithms for the 512-bit variants of hash functions implemented using Altera.

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.

Our future work will include experimental testing of all developed 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.

Acknowledgments. The authors would like to thank Ambarish Vyas for preliminary results regarding hash cores with padding units, and Rajesh Velegalati for extensive help with multiple ATHENA runs.

References

1. Cryptographic Hash Algorithm Competition, <http://csrc.nist.gov/groups/ST/hash/sha-3>.
2. SHA-3 Hardware Implementations, http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations.
3. ATHENA Project Website, <http://cryptography.gmu.edu/athena>.
4. Akin, A., Aysu, A., Ulusel, O.C., and Savas, E.: Efficient Hardware Implementation of High Throughput SHA-3 Candidates Keccak, Luffa and Blue Midnight Wish for Single- and Multi-Message Hashing. 2nd SHA-3 Candidate Conf. (2010)
5. Baldwin, B., et al.: FPGA Implementations of the Round Two SHA-3 Candidates. 2nd SHA-3 Candidate Conf. (2010)
6. Beuchat, J.-L., Okamoto, E., and Yamazaki, T.: A Compact FPGA Implementation of the SHA-3 Candidate ECHO. Cryptology ePrint Archive, Report 2010/364 (2010)
7. Detrey, J., Gaudry, P., and Khalfallah, K.: A Low-Area Yet Performant FPGA Implementation of Shabal. Proc. SAC'10 (2010) 99–113
8. Gaj, K., Homsirikamol, E., and Rogawski, M.: Fair and Comprehensive Methodology for Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs. Proc. CHES'10 (2010) 264-278.
9. Gaj, K., Kaps, et al.: ATHENA – Automated Tool for Hardware Evaluation: Toward fair and comprehensive benchmarking of cryptographic hardware using FPGAs. Proc. FPL'10 (2010)
10. Guo, X., Huang, S., Nazhandali, L., and Schaumont, P.: Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations. 2nd SHA-3 Candidate Conf. (2010)
11. Henzen, L., et al.: Developing a hardware evaluation method for SHA-3 candidates. Proc. CHES'10 (2010) 248-263
12. Homsirikamol, E., Rogawski, M., and Gaj, K.: Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs. Cryptology ePrint Archive, Report 2010/445 (2010)
13. Matsuo, S., et al.: How Can We Conduct “Fair and Consistent” Hardware Evaluation for SHA-3 Candidate? 2nd SHA-3 Candidate Conf. (2010)
14. Nechvatal, J., et al.: Report on the Development of the Advanced Encryption Standard (AES), <http://csrc.nist.gov/archive/aes/round2/r2report.pdf>.
15. Sklavos, N., and Kitsos, P.: BLAKE HASH Function Family on FPGA: From the Fastest to the Smallest. Proc. ISVLSI'10 (2010)
16. Tillich, S., et al.: High-speed Hardware Implementations of Blake, Blue Midnight Wish, Cubehash, ECHO, Fugue, Groestl, Hamsi, JH, Keccak, Luffa, Shabal, Shavite-3, SIMD, and Skein, Cryptology ePrint Archive, Report 2009/510 (2009).