Developing a Hardware Evaluation Method for SHA-3 Candidates

Luca Henzen[1], Pietro Gendotti[2], Patrice Guillet[2], Enrico Pargaetzi[2], Martin Zoller[2], and Frank K. Gürkaynak[3]

[1] Integrated Systems Laboratory, ETH Zurich
henzen@iis.ee.ethz.ch
[2] Department of Information Technology and Electrical Enginnering, ETH Zurich
{gpietro,pguillet,penrico,mzoller}@ee.ethz.ch
[3] Microelectronics Designs Center, ETH Zurich
kgf@ee.ethz.ch

Abstract. The U.S. National Institute of Standards and Technology encouraged the publication of works that investigate and evaluate the performances of the second round SHA-3 candidates. Besides the hardware characterization of the 14 candidate algorithms, the main goal of this paper is the description of a reliable methodology to efficiently characterize and compare VLSI circuits of cryptographic primitives. We took the opportunity to apply it on the ongoing SHA-3 competition. To this end, we implemented several architectures in a 90 nm CMOS technology, targeting high- and moderate-speed constraints separately. Thanks to this analysis, we were able to present a complete benchmark of the achieved post-layout results of the circuits.

1 Introduction

In 2007, the U.S. National Institute of Standards and Technology (NIST) started a public competition aiming at the selection of a new standard for cryptographic hashing [9]. Hash functions are cryptographic primitives that generate a sort of digital fingerprint of an arbitrary-length file, following some fundamental principles. Due to their flexibility, hash functions are used in a wide range of communication protocols where they provide data integrity, user authentication and many other security features. The motivation behind the NIST competition has been the growing concern of the security of two widely deployed hash functions MD5 and SHA-1 following a series of successful attacks [12, 1, 2]. The structural similarity of MD5 and SHA-1 with the current standard SHA-2 encouraged the NIST to start a new evaluation and selection process similar to the competition which promoted the Rijndael block cipher as new Advanced Encryption Standard (AES) in 2001. The cryptographic community was asked to propose new hash functions and to evaluate the security level of other candidates. In 2008, a total of 51 functions were accepted to the first round, while in July 2009 this number has been reduced to 14 second round candidates. The final decision, i.e., the proclamation of the winner algorithm, has been scheduled for 2012. To this end, the organizers are not only interested in the cryptographic strength of the candidates but also in the evaluation of the performance of the algorithm implemented in different platforms. The new SHA-3

S. Mangard and F.-X. Standaert (Eds.): CHES 2010, LNCS 6225, pp. 248–263, 2010.

standard is indeed expected to provide at least the security of SHA-2 with significantly improved efficiency. Several applications, from multi-gigabit mass storage devices to radio-frequency identification (RFID) tags, are expected to utilize SHA-3. It is therefore crucial that the final SHA-3 function should be flexible enough to be used in both high-performance and resource constrained environments. From a pure hardware point of view, the SHA-3 algorithm should provide good performance in terms of speed, area, and power.

Our interest in the SHA-3 selection process started with our involvement with the development of the candidate algorithm BLAKE. We participated in the algorithm specification, providing relevant information on the hardware performance and possible optimizations in this direction. When the SHA-3 competition entered the second phase, we started a VLSI characterization of several candidates within three separate student projects at our institute. The resulting designs were manufactured in three different ASICs, each containing a dedicated interface for I/O communication and the selected algorithms. At this time, we had implemented twelve out of fourteen candidate algorithms (all apart from ECHO and SIMD). We then decided to extend the analysis to all candidate algorithms.

In this paper we develop and present one methodology to evaluate the ASIC implementation of all SHA-3 second round algorithms. Rather than going for extremes of performance (fastest or smallest implementation) we propose to optimize all algorithms for multiple clearly defined specifications. We have applied our methodology and have evaluated several architectural variations of all candidate algorithms and presented the results.

The organization of the paper is as follows: A discussion of our methodology is the focus of Sect. 2. We present our approach to have a fair comparison, provide details and reasoning for key design decisions. Implementation details are given in Sect. 3. Due to limited space we were unable to provide implementation details for the architectures, an abbreviated summary of all architectures is provided in the Appendix. The results of our evaluation are presented in Sect. 4 together with a subsection that explains the errors in our methodology. We hope that this "open" approach will allow independent researchers to validate our findings. Finally in Sect. 5 we have concluding remarks.

2 Evaluation Methodology

In this work we will attempt to make a fair comparison between VLSI implementations of a set of algorithms all of which realize a similar function, but have very different structures. The main difficulty in this particular evaluation is the lack of concrete hardware specifications for the secure hash function candidates.

In practice, the specifications of the hardware are determined by the application. The hardware designers can then make several well-known trade-offs to come up with a design that offers the best compromise between, the required silicon area, the amount of energy required for the operation and the throughput/latency of the operation. For this study the requirements state *efficient hardware implementation* without being specific[1].

[1] This should not necessarily be understood as criticism for the NIST specifications. However, lack of concrete specifications make a fair comparison more difficult.

In some cases, such as telecommunication algorithms which have to fulfill requirements of certain well-defined standards, the application field alone sets sufficient constraints on the system. However cryptographic functions, like the SHA-3 hash function candidates that is the topic of this paper, are used for a very wide range of applications with different requirements. This makes it difficult to determine which of the performance parameters is more important. A hash function that is part of a battery operated wireless transmitter would probably be optimized for energy consumption, while the same algorithm when implemented in a telecommunication base station would most likely favor a high-throughput realization.

For comparative studies, if concrete specifications are not present, the authors will usually determine one parameter to be more important (i.e. throughput) [11,8,7], or will come up with aggregate performance metrics such as throughput per mm^2 [10, 3, 5]. Both approaches have their problems. Focusing on one parameter will favor algorithms which are strong on one parameter (i.e. throughput), but will not merit algorithms which perform better in other scenarios. Aggregate performance metrics on the other hand, may end up hiding the absolute performance of an implementation, impractical design corners (i.e. very large area, very low throughput) may perturb the results.

In the following subsection we will first define the performance metrics that we will consider in this evaluation. The next step will be to define specifications that will set limits on these performance metrics.

2.1 Performance Metrics

The most common metrics for hardware include the operation speed, the circuit area and the power consumption. For this analysis we have decided to use the following three main metrics for performance:

- **Circuit Area**
  Generally speaking the *cost* of an ASIC implementation of a function for a particular technology directly depends on the area required to realize the function[2]. In this evaluation we will use the net circuit area of a placed and routed design, including the overhead for power routing, clock trees. The area will be reported in kilo gate equivalents (kGE), where a gate equivalent corresponds to the area of a nominal drive strength 2-input NAND (or NOR) gate in the standard cell library used for the design realization. This metric covers the evaluation criteria *4.B.ii Memory requirements* in the NIST specification [9].
- **Throughput**
  We need a measure to determine how *fast* the implementation is. To this end we define the throughput of a hash function as the amount of message (input information) in bits for which a message digest can be computed per second. Furthermore, we assume that the hash function has been properly initialized, and the message sizes are matched to individual candidate functions for best case performance. The

[2] This is only true if the area is within a certain range. Extremely large circuits will have yield penalties, while very small circuits will not be able to justify the overhead associated with manufacturing.

throughput numbers are given in Gigabits per second (Gbps). This metric covers the evaluation criteria *4.B.i Computational Efficiency* in the NIST specification [9].

- **Energy Consumption**
  Power and energy metrics have gained more importance in recent years. On one hand there are power density limits the circuits have to comply for sub 100 nm technologies, and on the other hand for systems with scarce energy resources (handheld devices, smartcards, RFID devices etc.) reduced energy consumption equals to increased functionality or longer operating time. In this evaluation we will consider the energy consumption as our metric and will calculate the energy per bit of input information processed by the hash function. This will be obtained by dividing the total power consumption (in Watts) by the throughput (Gigabits/s) described above. The energy consumption will be given in milli Joules per Gigabit (mJ/Gbit). This metric partly covers the evaluation criteria *4.C.i.b Flexibility* in the NIST specification [9] as the energy efficiency is a deciding factor for implementation in constrained environments.

2.2 SHA-3 Parameters

The SHA-3 Minimum Acceptability Requirements state that all candidates should support message digest sizes of 224,256, 384, and 512 bits, and support a maximum message length of at least $2^{64} - 1$ bits. All algorithms process the message in blocks. The so-called *message block size* differs from algorithm to algorithm. In addition several submissions have included a *salt* input that can be used as a parameter in the hash function.

In our evaluation we have chosen:

- **Message Digest Size of 256**
  Several algorithms use (slightly) different architectures for different output lengths. Additional circuitry is then required to support all possible digest sizes. By selecting a single length, we aim to focus on the core algorithm which also simplifies certain architectural decisions. Out of the four required sizes, we have eliminated 224 and 384 as they are not a power of two (always an advantage in hardware design). We have settled on 256 as it will usually result in smaller hardware and faster implementations.
- **Use the largest message block size available**
  For each algorithm we have used the largest message block size and we have assumed that the message has already been padded (i.e. the length of the padded message is an exact multiple of the message block size). For throughput computation we always give the maximum achievable values, e.g., very long message for algorithms that have an initialization procedure.
- **No *salt* inputs**
  Since not all algorithms provide such an input, we have not included any *salt* inputs. For algorithms that provide a salt, the inputs are set to their default values according to the specification, and these constants have been propagated during synthesis to allow further optimizations whenever possible.

2.3 Defining Specifications

As mentioned earlier, the main difficulty in this evaluation is the lack of precise specifications that the candidate algorithms have to fulfill. Hardware design is based on finding a compromise between competing parameters that determine circuit performance. For example, there are several architectural transformations that allow to increase the throughput at the expense of the circuit area (see [6]). Without guiding specifications, it is difficult to determine which of the circuit metrics is more important for a design.

In summary, the NIST specifications in [9] require that the candidate algorithms to be computationally efficient (4.B.i), have limited memory requirements (4.B.ii), to be flexible (4.C.i) and simple (4.C.ii) [3].

The classical way to perform this analysis would be to concentrate on only the throughput metric and try to find out which algorithms are the fastest. In the last year, several groups presented comparative works and, almost certainly, others will be publishing new results to this effect. However, if only the maximum throughput requirement is investigated the *flexibility* of candidate algorithms may not be visible. Therefore we suggest to use two separate specifications: an aggressive **high-throughput** target and a **moderate-throughput** target.

The high throughput target has been chosen to be beyond the expected performance of most algorithms, and would therefore still be able to rank the algorithms in their maximum throughput capability. Our observation has been that even with older fabrication technologies, such as 180 nm CMOS, several candidate algorithms are able to reach throughputs of multiple Gigabits/s.

There are certainly applications which could make use of such throughputs, however such data rates are way beyond the requirements for many applications. For the moderate throughput requirement we have decided to determine a throughput which is at least two orders of magnitude lower than that used in the first case.

Fixing one of the performance metrics, allows us to make a fairer comparison between the remaining performance metrics (area and energy), and by considering two distinct throughput targets, we hope to uncover the *flexibility* of the candidate algorithms for different operational requirements. In particular, we will be interested in the circuit area for our high-throughput target, while we will be more interested in the energy consumption for our moderate-throughput target.

The maximum achievable throughput by a circuit implementing a cryptographic algorithm depends on the specific technology into which the circuit will be mapped. A throughput value that is easily achieved in 65 nm process, may not be feasible at all when using a 180 nm process. Therefore the specifications for our two scenarios have to be chosen while considering the capabilities of our target process.

We have decided to use the 90 nm CMOS process by UMC with the free libraries from Faraday Technology Corporation, mainly because we already had experience in designing ASICs with this technology and it was readily available within our design environment at the time of this study.

[3] Note that, computational efficiency could be interpreted in different ways, however, in the NIST specification it is stated that the "*computational efficiency essentially refers to the speed of the algorithm*". Similarly the memory requirements refer to the circuit area in hardware implementations.

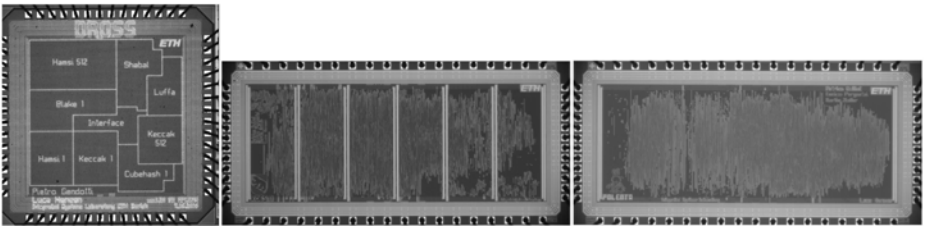

Fig. 1. From left to right: Photograph of the fabricated 90 nm chip implementing BLAKE, Cubehash, Hamsi, Keccak, Luffa and Shabal. Photograph of the 180 nm chips implementing BMW, Fugue, Grøstl, JH, SHAvite and Skein.

Table 1. Post-Layout results of the implemented algorithms

Algorithm	**Area** [kGE]	**Throughput** [Gbps]	**Energy** [mJ/Gbit]	**Technology** [nm]
BLAKE-32	33.55	7.314	15.291	UMC 90
BMW-256	95.00	3.527	31.407	UMC 180
CubeHash16/32-256	39.69	8.000	20.700	UMC 90
Fugue-256	26.00	2.806	122.506	UMC 180
Grøstl-256	65.00	4.064	73.075	UMC 180
Hamsi-256	32.25	7.467	23.624	UMC 90
Hamsi-512	68.66	7.467	46.605	UMC 90
JH-256	44.00	2.371	72.885	UMC 180
Keccak-256†	27.85	39.822	5.726	UMC 90
Keccak-512†	26.94	19.911	11.933	UMC 90
Luffa-256	29.70	22.400	9.482	UMC 90
Shabal-256	35.99	4.923	30.713	UMC 90
SHAvite-3_{256}	48.00	2.452	93.764	UMC 180
Skein-256-256	27.00	1.917	44.329	UMC 180

† First round specification.

Our experiences from designing the three ASICs (one of which was manufactured using this target technology) have given us a good estimation for the expected performance of all algorithms in the 90 nm process. We have decided to use 20 Gigabits/s for our high throughput target and 0.2 Gigabits/s for our moderate performance specifications. In the high-speed mode, almost all designs should be pushed to their speed limit, while with the latter we could evaluate the scalability and therefore the flexibility of each candidate algorithm.

2.4 ASIC Realizations

During this work twelve out of the fourteen second round SHA-3 candidates (some with several architectural variations) were fabricated in three different ASICs as shown in Fig. 1. Table 1 shows a list of algorithms that were implemented and their performances measured on the manufactured chips.

Actually implementing the designs in real silicon is certainly the best way to validate a design and determine its true potential. However, during this work we have realized that several practical factors have affected these results. The maximum available silicon area (that can be afforded for this project), the total number of I/O pins, the capabilities of the test infrastructure that is available for the test of the ASIC have all set limits on the implementations.

Since none of the designs was large enough to merit its own ASIC, each ASIC comprised of several independent modules. All modules shared a common interface which provided the inputs and collected the outputs from individual hash function realizing cores. For practical reasons, cores with similar clock frequencies were grouped together and were optimized using common constraints. In many cases compromises had to be made to allow two or more cores to be optimized at the same time. All of these had non-negligible influence on the outcome.

Practical considerations for testing of the systems has brought even more constraints. The necessity to include test structures (scan chains) adds some overhead, but more importantly, the maximum achievable clock rate greatly depends on the capabilities of the ASIC test infrastructure available. Designs with a high clock frequency (more than 500 MHz for 90 nm designs) put yet other constraints. When compared to designs running at lower frequencies, these designs suffer more from clock and power distribution problems, and are difficult to test at speed.

When designing these three ASICs we were forced to make many design decisions (i.e. blocks running faster than 700 MHz were deemed to be impractical within our environment) based on practical constraints which had its influence on the results. Scheduling constraints have also played a role in the choice of technology used to implement the designs. For the last two ASICs, there were no feasible 90 nm MPW (Multi Project Wafer) runs available. Consequently we had to submit these designs to a 180 nm run, which in turn made direct comparisons more difficult.

For this reason we have taken the design experience from the actual implementation of the individual cores, and have decided to re-implement all cores without considering these practical limitations. In particular we have decided:

- **No limits on the clock frequency**
  In this study we will not set any artificial limits on the clock rate. Obviously designs with high clock rates will still face the penalties for clock distribution, but we will not deal with practical considerations such as test, crosstalk and I/O limitations.
- **No test structures**
  Testing is an essential part of IC design. The exact overhead for testing depends on many factors, such as the desired test quality, and a one-size fits all solution is difficult to find[4]. Since the designs in this study will not be manufactured directly we chose not to include any test specific structures into the designs to have a fair comparison.
- **Assumed an ideal interface**
  The candidate algorithms differ in the number of I/Os they require. We have assumed that these core will eventually be part of a larger system which has an

[4] Simply using a full-scan methodology for example would not ensure that all designs have the same test coverage. Furthermore certain designs could be partially tested using functional vectors, or would be more amenable to BIST structures.

adequate I/O interface matching the requirements of each core. In this way, every function could express its maximum potentiality without suffering from any external limiation. However, we made no assumptions about how long the inputs stayed valid, all required inputs were sampled by the cores at the beginning of the operation. In other words, we implemented an internal message block memory for designs that require the input to be stable for more than one clock cycle.

– **No macro blocks**
We have not used any macro blocks to realize look-up tables or register files for portability reasons. All look-up tables and memory blocks were realized by standard cells.

3 Implementation

3.1 Design Flow

The same design procedure was used for all candidate algorithms. We have first developed a *golden* model based on the *Known Answer Tests* provided by the submission package. This golden model was then used to generate the stimuli vectors and expected responses that we have used to verify the RTL description of the algorithm written in VHDL.

We have then used Synopsys Design Vision-2009.06 to map the RTL description to the UMC 90 nm technology using the fsd0a_a_2009Q2v2.0 RVT standard cell library from Faraday Technology Corporation. All outputs are assumed to have a capacitive loading of 50 fF (equivalent to the input capacitance of about 9 medium strength buffers), and the input drive strength is assumed to be that of a medium strength buffer (BUFX8).

We use the worst case condition (1.08 V, 125 °C) characterization of the standard cell libraries. We have decided to use worst case characterized libraries in order to guarantee that we can meet the specifications. Table 2 is given as a reference to be able to compare the three characterizations that are commonly available (worst, typical, best) for one of the candidate algorithms.

Table 2. Comparison of different characterizations, synthesis results for the ECHO algorithm

	Worst Case	Typical Case	Best Case
Supply Voltage	1.08 V	1.2 V	1.32 V
Temperature	125 °C	27 °C	-40 °C
Critical Path	3.49 ns	2.24 ns	1.59 ns
Throughput	13.75 Gbps	21.42 Gbps	30.19 Gbps
Relative Performance	64.2 %	100 %	140.9 %

Depending on the throughput requirements, we try different architectural transformations such as parallelization, pipelining to come up with an architecture that meets (or comes closest to meeting) the requirements. We then use the Cadence Design Systems Velocity-9.1 tool for the back-end design. The technology used in this evaluation uses 8 metal layers (metallization option 8m026), out of which the top-most two are

double pitch (wider and thicker). A square floorplan is generated, leaving 30 µm space around the core for the power connections. For all designs we have used a 85 % utilization of the core area, in other words we have left 15 % of the area for post-layout optimization and power and ground distribution overhead. For power routing we have used a power grid utilizing Metal-7 and Metal-8.

Then the design is placed, a clock tree is synthesized and subsequently the design is routed. After every step the timing is checked, and if necessary a timing optimization is performed. At the end, if a valid layout without any Design Rule Check (DRC) violations are found, the *total core area* is reported as the area of the system. The total core area excludes the 30 µm space reserved for power rings, but includes all the available area that the placement and routing tool can use for the design. By default, all designs start with a 15 % overhead for post-layout optimizations. Depending on the design some amount of this overhead is used during various optimization phases during the back-end design. However it is difficult to quantify the *minimum required overhead* for every design reliably. We have decided to start all designs with the same initial placement density, and verified that the final design was not overly-congested. In a congested design, the routing solution includes many detours which adversely affect timing. For these designs the initial row utilization would have been reduced by 5 %, increasing the overhead. This was not necessary for any designs in this study[5]. In some designs, the routing resources are sparsely utilized. Such designs could have benefited from a higher initial row utilization, which could have resulted in a slightly smaller circuit without noticeable timing penalties. As mentioned earlier, it is not trivial to make sure that two designs have exactly the same amount of overhead. Therefore, we have not considered changing the default row utilization, unless there was a noticeable problem.

The timing results are taken from the finalized design. First, the Velocity tool is used to extract the post-layout parasitics and an SDF file containing the delays of all interconnections and instances is generated. The final netlist and the SDF files is read by the Mentor Graphics Modelsim-6.5a simulator and the functionality of the design is verified. At the same time, a Value Change Dump (VCD) file that records the switching activity of all the nodes during the simulation is produced. To have more realistic results, the start of the VCD file is chosen after the circuit has been properly initialized. This VCD file is then read back into the Velocity tool and a statistical power analysis is performed. The *Total Power* number is used to determine the energy consumption of the system.

3.2 Algorithms

For a given candidate algorithm, there are several well-known architectural transformations such as parallelization, pipelining, loop-unrolling etc. that will allow different trade-offs between circuit size and throughput. In addition, within the submission document, the authors often suggest different computational methods to perform a specific transformation of their candidate function. A good example is the frequently used

[5] Note that the initial density strongly depends on the technology options such as used metal layers. We have used 85 % as a result of our previous experience with this particular technology.

substitution boxes. They can be implemented as look-up tables, or can be realized as a circuit that computes the underlying function mathematically. To make matters worse, the exact trade-off between alternative realizations may only be visible after placement and routing. All these aspects broaden the spectrum of the possible hardware architectures. For a single candidate, there is often a large set of circuits with different trade-offs between size and speed. To identify the *best* design among many possibilities is not a trivial task. Despite all attempts to formalize architectural exploration, our experience has been that optimizing the circuit still remains a manual task, that relies on the skill and experience of the designer.

In this work, for each candidate algorithm we have selected what we believe was the most appropriate architecture that was able to reach the target throughput (20 and 0.2 Gbps) with minimal resources. For every candidate we designed and implemented two different architectures. The specifications of the single designs used within this work, is given in App. A. We make no claims that any of the architectures we have reported in this paper is the *best* possible architecture for a given candidate algorithm. In our opinion, it is not possible to make such a claim, and the exact implementations should be open to public scrutiny and review. For this purpose we have made all the source code that was used for this evaluation public on our www site [4].

4 Results

In this section we present the performance of the circuits implemented for high and moderate speed environments. The comparison between these two scenarios gives a further overview of the efficiency and flexibility of the candidate algorithms. We will refrain from concluding remarks about the performance of the algorithms, as we do not consider the results complete without public scrutiny.

For each architecture we report two operating frequencies/throughputs. The *Maximum Clock Frequency* is the maximum achievable clock frequency of the given architecture. When operating with this clock frequency the circuit can achieve the given *Maximum Achievable Throughput*. In most cases, this throughput is not exactly the same as the required throughput (either 20 or 0.2 Gbps). The second clock frequency states the clock frequency required to reach the target throughput. The final value in the tables is a relative indicator of how close the architecture is in achieving the target clock frequency. A number lower than one means that the architecture failed to achieve the target throughput. One can take this as a ratio of how closely we were able to optimize the circuit to the given target performance.

4.1 High Throughput Scenario

As expected, not all the circuits optimized for high-speed were able to reach the target throughput. Only two algorithms, Keccak and Luffa, were able to achieve the constraint. Table 3 lists the main performance figures for all architectures. In this scenario both area and energy were sacrificed to achieve high-throughput. The corresponding layouts can be seen in Fig. 2. The scale is given in the lower right corner of the figure. Circuits with a higher congestion rate (i.e. BMW or SIMD) require indeed the entire core for routing,

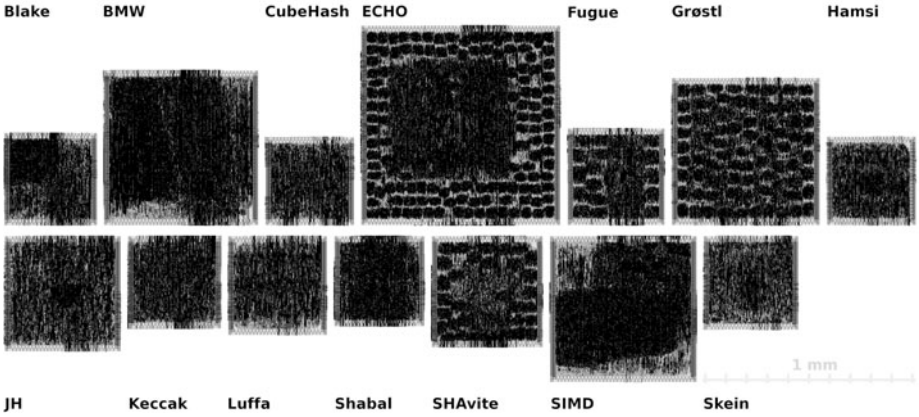

Fig. 2. The final layouts of all candidate algorithms for a target throughput of 20 Gbps

and would probably reach a faster throughput with more core area, i.e., a lower row utilization. Particularly interesting is also the local congestion for the 8-bit LUT-based S-boxes which makes them easily identifiable within ECHO, Grøstl, Fugue, and partly SHAvite.

4.2 Medium Throughput Scenario

The moderate-throughput circuits match the target throughput of 0.2 Gbps without difficulty. As can be seen in Table 4 the maximum achievable clock rate always exceeds the clock frequency required for 0.2 Gbps operation. To some extent, the additional speed can be traded to reduce the overall energy consumption, by lowering the supply voltage. It must be noted that there is a lower limit for the supply voltage (around 0.5 V for this process). Such voltage scaling techniques were not considered in this comparison, all results are listed for 1.2 V supply voltage.

Since in this scenario, timing was quite relaxed, the main figure of merit becomes the area and the energy dissipation. The layouts of all fourteen architectures are compared in Fig. 3, with the scale indicated on the bottom left.

The most interesting result is that a smaller area (or indeed throughput) does not always equal lower energy consumption (see Hamsi or Skein compared to BMW or SIMD). It must be noted that, no special precautions were taken for a low-power design (i.e. proper clock-gating, input-silencing). In addition some architectural decisions resulted in increased number of operations and/or increased circuit activity which affected the energy consumption differently for separate algorithms. We believe that there is much room for improvement in terms of low-power performance of the architectures. We must conclude that the present specifications do not necessarily result in low-power realizations in the medium-throughput corner. In a next step, the design methodology could be extended to provide a low-power scenario.

Table 3. Post-layout performances of all candicate algorithms for a target throughput of 20 Gbps in the UMC 90 nm process

Algorithm	Area [kGE]	Energy [mJ/Gbit]	Maximum Achievable Throughput [Gbps]	Maximum Clock Frequency [MHz]	Clock Freq. for 20 Gbps Throughput [MHz]	Max. / Target Frequency Ratio
BLAKE-32	47.5	11.00	9.752	400	820	0.49
BMW-256	150.0	16.86	8.486	298	703	0.42
CubeHash16/32-256	42.5	13.71	10.667	667	1250	0.53
ECHO-256	260.0	43.41	13.966	291	417	0.70
Fugue-256	55.0	15.60	8.815	551	1250	0.44
Grøstl-256	135.0	14.13	16.254	667	820	0.81
Hamsi-256	45.0	15.90	8.686	814	1876	0.43
JH-256	80.0	17.54	10.807	760	1406	0.54
Keccak-256	50.0	2.42	43.011	949	441	2.15
Luffa-256	55.0	6.92	23.256	727	625	1.16
Shabal-256	45.0	14.83	6.819	693	2033	0.34
SHAvite-3_{256}	75.0	19.21	7.999	562	1406	0.40
SIMD-256	135.0	35.66	5.177	364	1406	0.26
Skein-256-256	50.0	30.47	3.558	264	1484	0.18

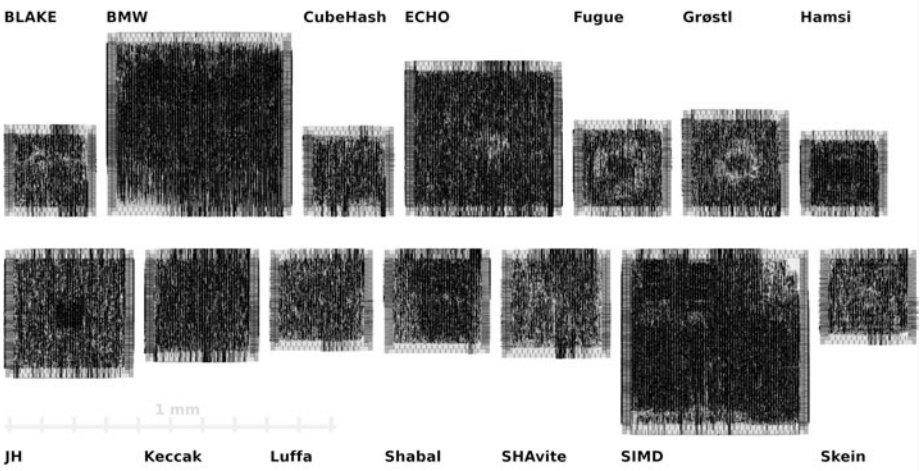

Fig. 3. The final layouts of all candidate algorithms for a target throughput of 0.2 Gbps

4.3 Sources of Error

Although we have tried our best to ensure a fair comparison, there are many factors that could have influenced the results. In this section we try to outline the possible sources of error in our results, and outline what we have done to address them.

- **Conflict of interest**
  One of the authors of this paper, Luca Henzen, is involved with the SHA-3 candidate algorithm BLAKE. Our interest in implementing the SHA-3 candidate algorithms has started by investigating optimal hardware implementations of BLAKE. We have tried to be as impartial as possible when implementing other candidate

Table 4. Post-layout performances of all candidate algorithms for a target throughput of 0.2 Gbps in the UMC 90 nm process

Algorithm	Area [kGE]	Energy [mJ/Gbit]	Maximum Achievable Throughput [Gbps]	Maximum Clock Frequency [MHz]	Clock Freq. for 0.2 Gbps Throughput [MHz]	Max. / Target Frequency Ratio
BLAKE-32	16.0	13.00	0.463	73.282	31.646	2.32
BMW-256	85.0	14.04	1.845	64.876	7.031	9.23
CubeHash16/32-256	16.0	10.50	1.741	217.581	25.000	8.70
ECHO-256	60.0	59.44	0.204	137.061	134.771	1.02
Fugue-256	19.0	9.02	1.828	114.260	12.500	9.14
Grøstl-256	25.0	22.28	0.412	128.750	62.500	2.06
Hamsi-256	15.0	35.12	0.200	150.083	149.925	1.00
JH-256	37.5	13.03	1.909	134.228	14.063	9.54
Keccak-256	27.5	5.50	6.767	149.276	4.412	33.83
Luffa-256	22.0	21.79	1.265	118.624	18.751	6.33
Shabal-256	25.0	26.57	0.399	128.634	64.475	2.00
SHAvite-3_{256}	25.0	11.43	1.871	131.527	14.063	9.35
SIMD-256	90.0	32.49	0.943	66.295	14.063	4.71
Skein-256-256	19.0	32.67	0.200	118.765	118.765	1.00

algorithms. However, it is true that we are more familiar with this algorithm than any other algorithm.

– **Designer experience**
The algorithms have been implemented by a group of students over a period of several months. Different designers may have more or less success in optimizing a given design. We have confidence in our team, but it is possible that for some algorithms we have inadvertently missed a possible optimization while for the others we were more successful. In addition, over time the designers naturally gain more experience and are more successful with the designs.
We believe that the most important aspect of a fair comparison is openness. For this reason we have made the source code and run scripts for the EDA tools used to implement all designs presented in this paper available on our website [4]. In this way, other groups can replicate our results, and can find and correct any mistakes we might have made in the process.

– **Accuracy of numbers**
The numbers delivered by synthesis and analysis tools rely on the library files provided by the manufacturer. The values in the libraries are essentially statistical entities and sometimes have large uncertainties associated with it. In addition most of the design process involves heuristic algorithms which depending on a vast number of parameters can return different results. Our experience with synthesis tools suggest that the results have around $\pm 5\%$ variation. We therefore consider results that are within 10% of each other to be comparable.
In an effort to be more accurate we have chosen to report post-layout area numbers that include clock and power distribution overhead. We have designed all circuits with the same overhead. For some circuits this overhead is adequate, for others it is too much, and for others is insufficient. We made sure that there is an acceptable solution for all cases.

- **Bias through specification**
  We have chosen two design corners in our applications, these specifications have helped us to have a common base for comparing all 14 algorithms. Regardless of how these specifications are chosen, it is possible that they benefit some algorithms more than the others. We hope that, similar studies by other groups which use different specifications will help to give a clearer picture.
- **Simplification due to assumptions**
  All our assumptions, the specific choices we made for SHA-3 parameters and the practical choices we made in the design flow will have some effect on the results. For example, we have decided not take IR-drop or crosstalk effects into account. As a result, the cores that achieve their reported performance by using very high clock frequencies will be more difficult to realize in practice. The assumptions in the design flow are a practical necessity and were designed to create a methodology in which the same solution could be used for all designs.

5 Conclusions

In this paper we have presented a methodology to compare the SHA-3 candidate algorithms. Our previous experiences in designing ASIC implementations of candidate algorithms (Table 1) has been instrumental in developing what we believe is a fair set of specifications. Rather than targeting outright performance, we have set limits for one performance metric (throughput) and re-implemented all algorithms to meet two distinct throughput requirements. This enabled us to compare the *flexibility* of the algorithms (Tables 3 and 4).

A public selection process, such as the SHA-3 invariably attracts a large number of submissions with many different algorithms. In early stages of the selection process, the sheer number of algorithms (51 in the first round) makes it impractical to employ a detailed analysis for hardware suitability. Our experience has shown that even with the 14 second round candidates, it is difficult to present an authoritative and fair evaluation of all candidates. We believe that for the final round of evaluations, a similar approach to what we have demonstrated in this paper should be utilized: Clear constraints should be set for the implementations, preferably more than one performance corner should be targeted, the evaluation process should be well documented and the errors in the evaluation process should be openly discussed. We would also suggest the addition of a low-power corner that also considers voltage scaling for low-power operation to our methodology.

In many parts of this paper, we have extensively commented on limitations of our methodology, and have included a whole subsection on sources of error. We strongly believe that any such comparison must be thorough with its analysis of error sources and clear with its performance metrics.

References

1. De Cannière, C., Rechberger, C.: Finding SHA-1 characteristics: General results and applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)

2. De Cannière, C., Rechberger, C.: Preimages for reduced SHA-0 and SHA-1. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 179–202. Springer, Heidelberg (2008)
3. El-Hadedy, M., Gligoroski, D., Knapskog, S.J., Aas, E.J.: Low area FPGA and ASIC implementations of the hash function "Blue Midnight Wish-256". In: International Conference on Computer Engineering & Systems, ICCES 2009, Cairo, pp. 10–14 (2009)
4. Gürkaynak, F.K., Henzen, L., Gendotti, P., Guillet, P., Pargaetzi, E., Zoller, M.: Hardware evaluation of the second-round SHA-3 candidate algorithms (2010), http://www.iis.ee.ethz.ch/~sha3/
5. Gürkaynak, F.K., Luethi, P., Bernold, N., Blattmann, R., Goode, V., Marghitola, M., Kaeslin, H., Felber, N., Fichtner, W.: Hardware evaluation of eSTREAM candidates: Achterbahn, grain, mickey, mosquito, sfinks, trivium, vest, zk-crypt. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/015 (2006), http://www.ecrypt.eu.org/stream
6. Kaeslin, H.: Digital Integrated Circuit Design, from VLSI Architectures to CMOS Fabrication. Cambridge University Press, Cambridge (2008)
7. Kobayashi, K., Ikegami, J., Matsuo, S., Sakiyama, K., Ohta, K.: Evaluation of hardware performance for the SHA-3 candidates using SASEBO-GII. Cryptology ePrint Archive, Report 2010/010 (2010), http://eprint.iacr.org/
8. Namin, A.H., Hasan, M.A.: Hardware implementation of the compression function for selected SHA-3 candidates. CACR 2009-28 (2009), http://www.vlsi.uwaterloo.ca/~ahasan/hasan_report.html
9. NIST. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. Federal Register 72(212) (2007), http://www.nist.gov/hash-competition
10. Tillich, S., Feldhofer, M., Issovits, W., Kern, T., Kureck, H., Mühlberghuber, M., Neubauer, G., Reiter, A., Köfler, A., Mayrhofer, M.: Compact hardware implementations of the SHA-3 candidates ARIRANG, BLAKE, Grøstl, and Skein. Cryptology ePrint Archive: Report 2009/349 (2009)
11. Tillich, S., Feldhofer, M., Kirschbaum, M., Plos, T., Schmidt, J.-M., Szekely, A.: High-speed hardware implementations of BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, and Skein. Cryptology ePrint Archive, Report 2009/510 (2009)
12. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)

A Hardware Architectures

Table 5 gives an overview of the architectures, used within this work. For some candidates we used the same design for the 20 Gbps (HS) and 0.2 Gbps (MS) analysis. In such cases, different optimization parameters were used. The detailed description of the architectures has been omitted because of the limited article length. Refer to [4] for the complete source code for all the architectures used in this evaluation

Table 5. Design specification of the HS and MS-target architectures. For the latency, the enclosed value refers to the finalization cycles.

Algorithm	Message Block Size [bits]	Arch.	Latency [cycles]	Implementation details
BLAKE	512	HS	21	Four parallel G function modules, anticipation of the first message-constant addition.
		MS	81	One G function module.
BMW	512	HS-MS	18 (+18)	f_0 and f_2 computed in one cycle, while f_1 iteratively decomposed in a single *expand* block.
CubeHash	256	HS	16 (+160)	Single round per cycle, initial state stored.
		MS	32 (+320)	Half round, initial state stored.
ECHO	1536	HS	32	8 AES rounds per clock cycle.
		MS	1034	Single 32-bit AES core, one parallel BigMixColumn unit.
Fugue	32	HS	2 (+37)	S-box as LUT.
		MS	2 (+37)	S-box as composite field logic.
Grøstl	512	HS	21 (+21)	Interleaved P and Q permutation with one pipeline stage, *SubBytes* as LUT.
		MS	160 (+160)	Single-column round (64-bit datapath), *SubBytes* as composite field.
Hamsi	32	HS	3 (+6)	Message expansion in three 256×256 LUTs, single round per cycle, substitution layer as logic.
		MS	24 (+48)	Same as HS, datapath reduced to 128 bits.
JH	512	HS-MS	36	S-boxes S_0 and S_1 stored in LUTs, constants stored.
Keccak	1088	HS-MS	24	Single round per cycle.
Luffa	256	HS	8	Three parallel *Step* function modules, *SubCrumb* function as logic.
		MS	24	One *Step* function modules, *SubCrumb* function as logic.
Shabal	512	HS	52 (+156)	One keyed permutation round per cycle. In total, 30 adders and 16 subtractors.
		MS	165	One adder and one subtractor only.
SHAvite-3	512	HS	36	One AES round for message expansion and one AES round for the F^3 round, *SubBytes* as LUT.
		MS	36	Same as HS, *SubBytes* in composite field.
SIMD	512	HS-MS	36 (+36)[†]	Four parallel Feistel modules, message expansion based on NNT_8 and eight multipliers for tweadle mult.
Skein	256	HS	19 (+19)	Four unrolled Threefish rounds.
		MS	152 (+152)	Half Threefish round.

[†] Further 36 cycles of initialization required for message expansion.