Lessons Learned from Designing a 65 nm ASIC for Evaluating Third Round SHA-3 Candidates

Frank K. Gürkaynak[1], Kris Gaj[2], Beat Muheim[1], Ekawat Homsirikamol[2], Christoph Keller[3], Marcin Rogawski[2], Hubert Kaeslin[1], and Jens-Peter Kaps[2]

[1] Microelectronics Designs Center, ETH Zurich, Switzerland
{kgf,muheim,kaeslin}@ee.ethz.ch
[2] ECE Department, George Mason University, Virginia USA
{kgaj,mrogawsk,ehomsiri,jpkaps}@gmu.edu
[3] Integrated Systems Laboratory, ETH Zurich, Switzerland
chrikell@iis.ee.ethz.ch

Abstract. In this paper we present the implementation results for all five SHA-3 third round candidate algorithms, BLAKE, Grøstl, JH, Keccak, and Skein in a standard-cell-based ASIC realized using 65nm CMOS technology. The ASIC includes two sets of implementations developed independently and with different optimization targets, and includes a reference SHA-2 implementation as well. We believe that having the results of two separate sets of implementations allows us to better identify the valid design space for all candidate algorithms. We present data showing the evolution of the solution space for each algorithm from simple synthesis estimations to the final implementation on the ASIC and show that post-layout results can differ by as much as 50% from synthesis results.

1 Introduction

Five candidates have been selected by NIST for the final round of the SHA-3 selection process: BLAKE, Grøstl, JH, Keccak, and Skein. In the NIST announcement [7], section 6.C.ii, it is stated that "*NIST welcomes comments regarding the efficiency of the candidate algorithms when implemented in hardware*.". This paper includes our results from implementing all candidate algorithms and a reference implementation of SHA-2 in standard-cell-based 65 nm CMOS technology. At time of the writing (October 2011) the ASIC containing all implementations has already been taped out, and the fabricated chip is expected back from manufacturing in February 2012, before the 3rd SHA-3 Candidate Conference, where we hope to be able to present measurement results from the actual ASIC.

Two groups from Swiss Federal Institute of Technology Zurich (ETHZ) and George Mason University, Virginia USA (GMU) each contributed one set of implementations with different implementation goals. In total twelve cores were combined on one ASIC, with a common input/output (I/O) interface designed to reduce the required I/O bandwidth, while still allowing adequate control for comprehensive verification.

The paper is organized as follows: The implementation goals, description of the core interface and the input/output block are described in Section 2. The performance metrics used throughout the paper are explained in Section 3 and the design flow employed for the ASIC is presented in Section 4. The following Section 5 summarizes overall comparisons for area, throughput and power (more detailed discussion for each algorithm can be found in Appendix A). Section 6 discusses error sources in our results and finally Section 7 offers our conclusions.

2 Implementation

In this work, two separate sets of SHA-3 candidate algorithms have been implemented using a standard-cell-based ASIC design flow. We have used the Throughput-per-Area-optimized implementations of the algorithms developed by GMU within their ATHENa framework [2]. Originally these implementations were optimized for FPGA structures. Since they do not make use of FPGA specific resources (block RAMs, DSP blocks etc.) they could be directly used within the ASIC design flow for this project. Only minor adjustments were made to adapt the interface of the cores.

At ETHZ a new set of implementations was developed independently. For these cores, a fixed throughput constraint of 2.488 Gbit/s was chosen (data rate for a OC48 channel). The only consideration while determining this data rate was to choose a rate which could readily be achieved by all candidate algorithms. For these cores, rather than trying to achieve optimal implementations, the targeted constraint was used to drive architectural decisions, and (later) the synthesis process. Since the GMU implementations were available from the start, different architectural choices were made whenever possible to add more variation to the results.

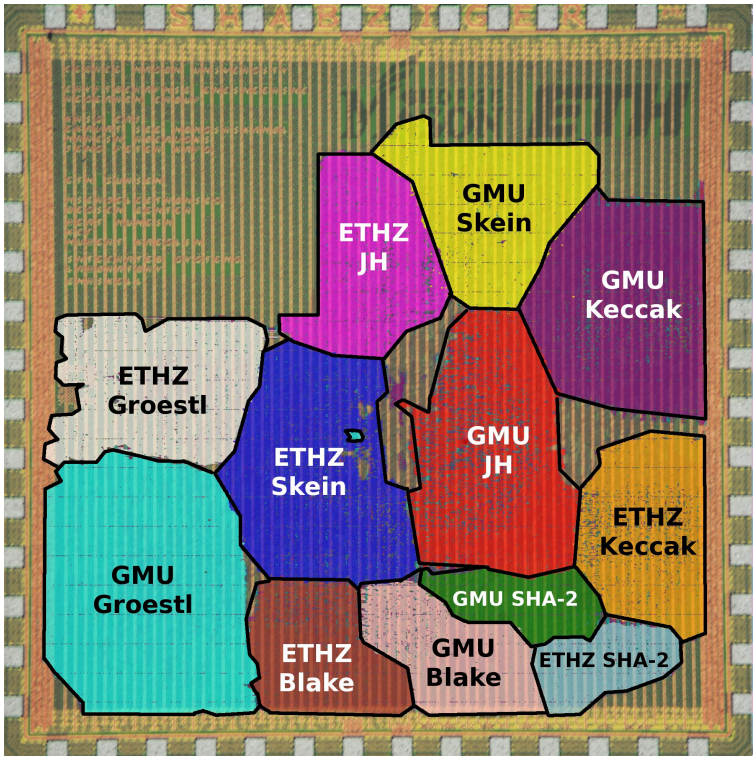

Fig. 1. The chip micrograph of the fabricated ASIC with the location of all twelve cores highlighted. One side of the chip is 1.875 mm long. Area highlight is approximate.

An ASIC using a 65 nm CMOS technology was designed to include all twelve cores at the same time. Figure 1 shows the physical layout of the chip, with individual cores highlighted.

2.1 Assumptions

Each SHA-3 candidate comes with several parameters that can be adjusted to tune the algorithm for different requirements. In this paper we have chosen the authors' recommendations for the parameters providing a security equivalent to that of SHA-2-256. Specifically this means using: BLAKE-256 with 32-bit words, Grøstl-256, JH-256, Keccak[r=1088,c=512], Skein-512-256. The implementations do not contain any additions to support different output widths, special modes of operation (i.e. MAC), or salt values. All algorithms were implemented with round 3 tweaks.

All implementations have 256-bit wide outputs, and the input matches the message block size of each algorithm: 1088 bits for Keccak, 512 bits for all others. The input and output registers were not counted as part of the core area[1]. All implementations accept message lengths of arbitrary size, however they contain no padding unit. It is assumed that the last message block has been padded accordingly.

It is further assumed that the clock frequency of each core could be determined independently and the clock frequency was not limited due to external factors such as limitations of the input/output pads needed for the ASIC.

2.2 Core interface

All cores were designed to use a common interface. A simplified block diagram of this interface can be seen in Fig. 2. The message block is presented to all cores in parallel (1088 bits for Keccak, 512 bits for all others). The `InWrEnxSI` signal is used to notify the core that a new message block is ready. If the message block is the last block of the message, `FinBlockxSI` signal is driven high at the same time. The core is expected to store the data immediately. The core then processes the data, and one cycle before it can accept new data it asserts a `PenUltCyclexSO` signal (Penultimate Cycle). Since the core gives advance notice of being ready, the environment can prepare to dispatch a new block for processing in the next cycle. If the core has finished processing the last message block, it will assert `OutWrEnxSO` as soon as the 256-bit output is ready. The environment is expected to sample this output as soon as the `OutWrEnxSO` is active.

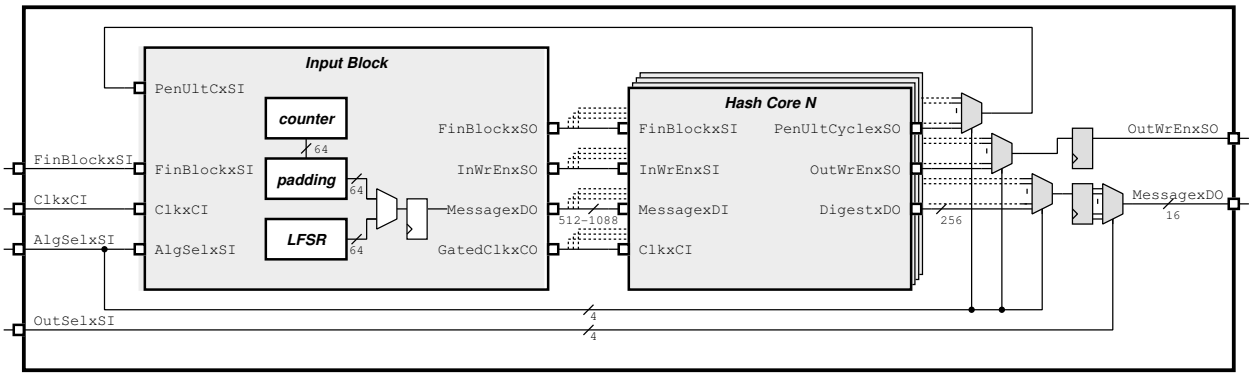

Fig. 2. Simplified block diagram of the ASIC showing the core interface and the I/O block containing the LFSR.

The main challenge in implementing twelve hashing cores in one small ASIC is to control the I/O bandwidth. To be able to use existing test infrastructure, it was decided to use a 56 pin package for the ASIC[2]. The solution that we have adopted, uses an LFSR implementation (using the primitive polynomial $x^{73}+x^{25}+1$) that calculates 64-bits per clock cycle as a pseudo-random number generator[3]. The output of this PRNG is used to generate a new input message block.

A very simple padding scheme is used in the I/O block. This unit supports message sizes of the form $n \cdot message\_block\_size + k$, where k is a constant specific for each algorithm and n is an arbitrary non-negative integer. For example for the SHA-2 algorithm, the padding is fixed to 80 bits. In this case, k is 432 bits. The messages could then be $432, 944, 1456, \ldots, n \cdot 512 + 432$ bits long. Note that this has no influence on the design of the cores, all cores support arbitrary message

[1] If an implementation requires that the input stay stable for more than 3 clock cycles an input register is needed (i.e. ETH-BLAKE). In such a case, this input register has been included within the core area

[2] The ASIC is 1.875mm per side, which does not allow for a large number of I/O pads anyway.

[3] The LFSR is designed to generate 64 bits in parallel. In this configuration, the chosen primitive polynomial has a small area overhead.

lengths. The exact choice of the padding length depends on the specific algorithm, and has been selected to reduce the area of the Input block.

A chip wide `FinBlockxSI` signal is used to tell the I/O block to generate the next message block as a padded final message block. As soon as the core is able to accept a new message block, a core level `FinBlockxSI` signal is generated and the core finishes processing the message. This enables the user to control different message lengths externally. If the `FinBlockxSI` signal is continuously held high, short message behavior of the core can be observed (each message will be fixed to one block and all finalization steps will be performed for this block), and if it is held low, long message behavior can be observed.

The ASIC allows only one core to be active at any given time. This is directly controlled by a 4-bit `AlgSelxSI` signal that controls the chip wide clock gating and the core selection circuitry. Since there are only 12 cores, there are 4 additional modes available. One of these modes is assigned to be a dummy mode, which does not add any active circuitry and will be used to measure the power overhead of the Input block[4].

All cores produce 256-bit outputs. The output of the cores is sampled as soon as the cores drive `OutWrEnxSO` signal. However, only 16 bits are propagated to the chip outputs due to I/O limitations at a time. A 4-bit `OutSelxSI` signal is used to select the 16 bit words, allowing a method to examine all outputs in a time-interleaved fashion.

3 Performance Metrics

3.1 Area

Throughout this paper we have used Gate Equivalent (GE) as our primary measure for area. We define 1 GE as the area of a simple 2-input NAND gate in our target technology (ND2M1) which is equal to 1.44 μm^2. Using the GE as an area unit allows the area figures to be interpreted independently from the target technology.

Our design flow (explained in Section 4) consists of four distinct steps. All the results from **step A** through **step C** report the area as the total cell area used for the circuit. Only for the runs in **step D** the total occupied area is used. This area is the rectangular area in which all the circuit could be placed and routed. Technically, it could be argued that this is a better measure for the area. However, it is practically impossible to extract this value on real ASICs without adding artificial constraints. In our case, the results in **step D** do not contain any overhead for power distribution since it is very difficult to come up with a fair way to account for the additional area, especially with designs using different clock frequencies.

3.2 Throughput

We define the throughput of a hash function as the maximum number of message bits for which a digest can be computed per unit time. For these calculations only very long messages have been considered, the effect of the finalization procedures have been neglected. The values (unless otherwise stated) are given in Giga (10^9) bits per second (Gbit/s).

3.3 Throughput per Area

In this paper we use the Throughput per Area (TpA) as an efficiency metric. The throughput and area numbers are obtained as described above and the ratio is calculated. The unit of this measure in this paper is kilo (10^3) bits per second per gate equivalent (kbit/s·GE). The rather strange unit is a result of scaling (the cores have values between 20 and 500 kbit/s·GE in this project). The goal is to have a high TpA figure, as this means that a longer message can be processed in the same time using the same number of gates.

[4] The three remaining modes are used to drive test circuitry that is unrelated to this project.

3.4 Power and Energy

The power consumption of all cores will be reported for the targeted throughput value of of 2.488 Gbit/s. At a constant throughput, the power figures for the cores will correspond to their energy consumption per bit as well. The ASIC includes a *dummy* mode that activates only the I/O block (including the LFSR). This mode has been measured separately and its value has been subtracted from measurements of individual cores. Thus, both the overall static power consumption (same in both measurements), and the dynamic power consumption of the I/O interface have been subtracted. The reported power values correspond[5] to the dynamic power consumption values for individual cores only.

4 Design Flow

For this study we have used a typical standard-cell-based ASIC design flow with two phases. The *front-end* phase involves developing an RTL model in a Hardware Description Language (HDL), in our case we have used VHDL. The RTL model is verified against a golden model using an HDL simulator. The reference implementations in C within the NIST submission packages were used as golden models[6] and *Mentor Modelsim-SE 10.0b* was used for RTL simulations. Once the RTL description passed functional verification, it was synthesized (converted to a netlist of standard cell components from the target technology) using *Synopsys Design Compiler D-2010.03-SP1-1*. At the end of the *front-end* phase, a gate-level netlist is obtained. This netlist is already targeted for a particular technology, but it does not yet have any physical information. Including this information is done in the second phase, the *back-end* design. In this work, *Cadence Design Systems Encounter Digital Implementation v10.12-s181_1* has been used throughout the *back-end* design phase. Once the final netlist is produced, Modelsim is used to verify the correct functionality, this time taking into account delays incurred through individual gates and their interconnection.

4.1 Technology

For this project we were able to use a run on the UMC65LL technology offered through Europractice MPW services. Most contemporary technologies offer various options which may have a large impact on circuit performance. It is therefore slightly misleading to talk about a generic 65 nm technology, as the specific options between two implementations may be completely different. In general, the 65 nm technology offered by UMC has two families a Low-Leakage (LL) family and a Standard Performance (SP) family. In our case we have used the LL option since the specific MPW run that we have used offered only this option. In addition, each option supports three transistors with different threshold voltages (VT). Generally speaking a transistor with a low threshold voltage will switch faster, but will also have a higher leakage current. The three transistor options have been labeled as Regular (RVT), High (HVT) and Low (LVT). We had access to standard cell libraries with all three VT options, and we have allowed cells from all libraries to be used for synthesis. This enables the synthesizer to choose gates using fast (LVT) transistors for timing-critical paths, and cells using slower but less leaky (HVT) transistors for non-timing-critical paths.

All libraries have been characterized using three corners (typical, best, worst) for process variation, supply voltage, and operating temperature. Throughout this paper we report results using the worst case libraries for the timing. For comparative analysis the choice of the corner does not make a large difference. However, when certain performance numbers need to be achieved (as for the 2.488 Gbit/s throughput target for the ETHZ implementations), it must be remembered that the worst case corners carry significant margin. Table 1 is given as a reference to be able to compare the three characterizations that are commonly available (worst, typical, best) for one implementation of the SHA-2-256.

[5] to the best of our measurement ability

[6] In order to obtain the inputs, message padding had to be performed by customized Perl scripts. The reference implementations accept messages of any length, and perform padding if necessary, but they do not make the padded message available so that it could be directly used.

Table 1. Comparison of different characterizations, post-layout results for the SHA-2-256 implementation of GMU.

	Unit	Worst Case	Typical Case	Best Case
Supply Voltage	[V]	1.08	1.2	1.32
Temperature	[°C]	125	27	0
Critical Path	[ns]	1.860	1.149	0.846
Throughput	[Gbit/s]	4.235	6.855	9.311
Relative Performance		1.00	1.62	2.23

4.2 Estimating Parasitic Interconnection Effects

The delay of a standard CMOS gate is directly proportional to the capacitive loading at its output. Ideally this output capacitance is only due to the sum of the input capacitances that the gate drives. However, the interconnection between the gate output and the inputs it drives adds a parasitic capacitance[7] that in modern processes may be even larger than that of a typical capacitance. The exact effect of the interconnect capacitance can only be determined once the exact location of all the gates are known and a valid routing solution has been found in the back-end design phase.

During the front-end design phase this information is not directly available. Therefore synthesis tools rely on *wireload models* to estimate the load as a result of parasitic wiring capacitances. The wireload model is a statistical model. In the most widely used form, the wireload model includes the additional wiring capacitance and resistance as a function of the number of output connections of a net. Practically all standard cell libraries include a *default* wireload model.

In the first phase of the design, this default wire-load model is used. After the initial design is completed and a placement and routing solution has been established, the corresponding wireload is extracted to be used as a customized model in the subsequent front-end design phases.

We report four different types of performance results from different stages of the design flow. To ease the reporting we have used single letter abbreviations

4.3 Initial Synthesis (step A)

As a first step, all cores in the project were synthesized individually driven by a single parameter that sets a target for the clock period. At this stage this default wireload was used for the synthesis of all cores. A set of netlists were then created by performing a synthesis run each time with a different target clock period parameter. Out of this set of netlists, one netlist was chosen based on its Throughput and Area figures for each hash core. For the GMU implementations, a netlist that exhibited the highest throughput per area figure (see sec.3.3) was selected, while for the ETHZ implementations the goal was to select an implementation that would be capable of delivering 2.488 Gbit/s throughput with the lowest possible area.

4.4 Synthesis using an Extracted Wireload Models (step B)

In a second step, we have placed and routed the entire ASIC with individual cores from **step A** described above. We have deliberately used a relaxed timing constraint (twice the amount used for the original synthesis) to allow a back-end solution that did not involve much optimization (which results in a netlist that is - comparatively - more similar to the one obtained during the synthesis). We have extracted a wireload model for each of the cores individually. All subsequent synthesis runs were produced using this customized wireload model for each core. As can be seen in the results for individual cores, the results with an extracted wireload model are roughly 20% larger in area and have about 15% longer clock periods with at times large variations for individual implementations.

[7] Interconnections parasitic also have, resistive, inductive and conductive parameters. We will use parasitic capacitance in the descriptions as it is the main contributor.

It must be noted that the extracted wireload is not universally representative for a given core. As the synthesis constraints change, the characteristic of a circuit will also change. Depending on the constraints, the new circuit may be much smaller, use different implementations for key components (i.e. the synthesizer may decide to use a simple Ripple Carry Adder, instead of a more costly Brent-Kung adder if the constraints are relaxed sufficiently), which in turn would also alter the corresponding wireload. However, we believe that rather than using unrelated default wireloads, using an extracted wireload will produce more representative results.

Once again, a set of netlists were generated at this stage by multiple synthesis runs in which the target clock period was changed. For each core one netlist was selected out of this set. For GMU implementations, circuits with high throughput per area ratio were favored, while for ETHZ implementations circuits that were closer to the targeted 2.488 Gbit/s throughput were favored.

4.5 Final results from the ASIC sent to manufacturing (step C)

The final ASIC implementation contains a total of twelve cores (six each from GMU and ETHZ) implemented together using a common input/output block. For this phase one netlist for each core was selected from the previous **step B**. A rather complex multi-mode multi-corner (MMMC) constraint file was supplied to the back-end design allowing each hash core to be optimized at its own target clock period. Multiple back-end runs were performed and the MMMC constraints were adjusted to find a solution that would satisfy all constraints. Since multiple designs are optimized at the same time, it is more difficult to achieve a solution where the performance of all individual cores is maximized.

4.6 Individual post-layout runs (step D)

The results from **step C** include many different effects. First of all, the overall timing of the entire chip was a compromise. It is highly likely that, for an individual core, a slightly different constraint would have resulted in a better overall solution. The back-end results tell only how well the current constraints could be met, they do not necessarily reveal how much certain constraints could be changed and what effect such a change would have had on the overall result[8].

Calculating the total circuit area occupied by an individual core is not trivial from **step C**. The cells comprising a given hash core are not placed next to each other without leaving space in between them. The placement includes necessary provisions for power/signal routing and allows gates from other cores to be interspersed to improve overall performance. As a result, the real circuit area is somewhat larger than the total area occupied by the cells.

In order to analyze the overhead of back-end for individual cores independently, we have made individual back-end runs that contain only a single core. The netlist chosen from **step B** was used as a starting point. A series of back-end runs were made with two independent parameters. The first one was the maximum clock period as used throughout all the earlier steps. The second parameter was the initial placement density, which is the ratio of total cell area of the core from **step B** and the total area which is available for the back-end design. An initial placement density of 0.8 means that the back-end run will take the initial netlist, calculate the total cell area of all the cells within this netlist and reserve a square area that is 1.0/0.8=1.25 times larger for placement and routing.

It must be noted that, the results from **step D** are not *practically realizable* circuits as they do not contain any provision for power routing. They are intended to show the limits of achievable placement and routing solutions, and are included to provide a comparison to the actual implemented results from **step C**.

Also note that all the analyzes in **step D** were made with the netlist from **step B**. Differences in this netlist would undoubtedly also cause changes in the back-end results. All in all, the main problem is that the current breed of EDA tools are not based on *finding the best performance* but are geared towards *finding a netlist that satisfies the given constraints*. This makes more academic endeavors such as finding the *optimal* netlist for a given RTL description more difficult.

[8] i.e. the ETHZ implementation of SHA-2 was constrained for 3.50 ns maximum clock period. The end result achieved a maximum clock period of 3.395 ns. At this point it is not possible to say what the result would have been if we were to constrain the ETHZ SHA-2 with 3.2 ns maximum clock period, since this would not only change the ETHZ SHA-2 implementation, but it would also affect all other cores to some extent.

4.7 Measurement Results (Step E)

At the time of writing, we have measurement results from only one ASIC. Measurements from this chip has shown all cores to be fully functional. Furthermore all cores were functionally tested above the indicated post-layout figures (between 15%-92% faster), and the power consumption figure was (10%-70%) higher than the *worst case* estimations. However, the measurements were made using nominal VDD (1.2V) whereas all the estimations were made with *worst case* parameters and VDD (1.08V), which explains most of the discrepancy. However, with only one chip tested, the results are not very reliable. In order not to interfere with the rest of the paper, these measurement results are given in Appendix B separately.

5 Results

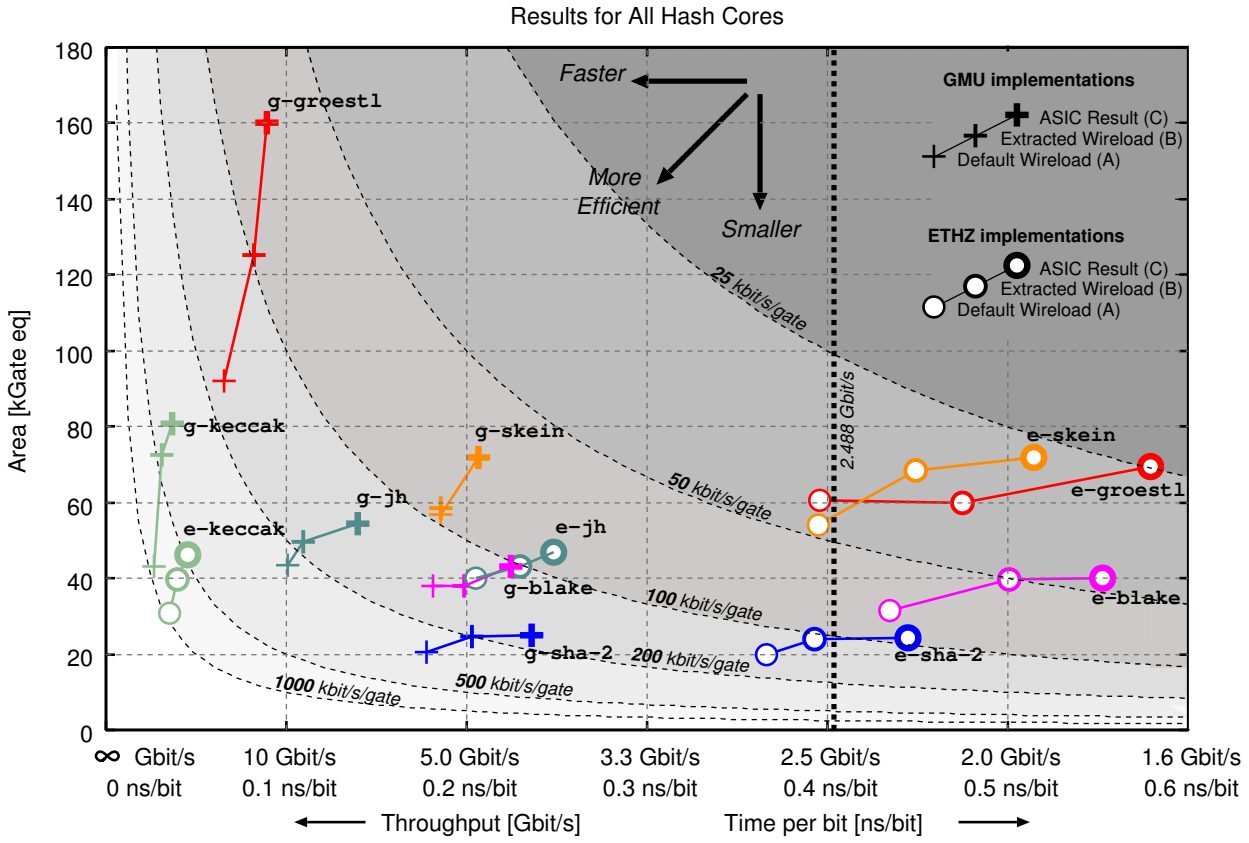

Fig. 3. AT plot showing all implemented cores for worst case conditions. For each core, three points have been plotted. These correspond to the results from **step A**, **step B**, and **step C** respectively. Note that for all implementations, **step A** is more efficient than **step B** which in turn is more efficient than **step C**. The ETHZ implementation target of 2.488 Gbit/s is highlighted on the X axis.

The results in this section are obtained from the actual ASIC implementation that we have sent to fabrication (**step C**). More detailed analysis including results from intermediate stages are given in Appendix A as reference. This work concentrates only on the hardware implementation parameters of the SHA-3 candidates, cryptographic security

Table 2. Area, Throughput. and Throughput per Area figures for all implemented cores on ASIC. Post-layout results using worst case process corner, 1.08 V, 125°C. See Table 6 for preliminary measurement results.

Algorithm	**Block Size** [bits]	**Implementation**	**Area** (FFs) [kGE]	**Max. Clk Freq.** [MHz]	**Throughput** [Gbit/s]	**Throughput per Area** [kbit/s·GE]
SHA-2	512	ETHZ	**24.30** (29%)	294.55	2.251	92.622
		GMU	25.14 (35%)	537.63	4.235	168.453
BLAKE	512	ETHZ	39.96 (26%)	201.41	1.809	45.271
		GMU	43.02 (34%)	252.40	4.456	103.573
Grøstl	512	ETHZ	69.39 (17%)	273.15	*1.727*	*24.881*
		GMU	*160.28* (9%)	459.14	11.194	69.842
JH	512	ETHZ	46.79 (27%)	330.58	4.030	86.124
		GMU	54.35 (31%)	602.77	7.177	132.053
Keccak	1088	ETHZ	46.31 (25%)	485.44	22.006	**475.179**
		GMU	80.65 (19%)	599.16	**27.162**	336.788
Skein	512	ETHZ	71.87 (19%)	349.41	1.945	27.055
		GMU	71.90 (22%)	179.86	4.847	67.409

has not been addressed: for the purposes of this analysis it is assumed that all hash cores provide equivalent levels of security.

In Figure 3 all twelve implementations (ETH implementations denoted using circles, GMU implementations with plus signs) are shown on the same AT graph[9]. In this AT graph the vertical axis represents the *circuit area* in kGE, and the horizontal axis represents the time. The corresponding throughput values (increasing towards left) are also given in Gbit/s for clarity. The resulting graph has its optimum point at the origin (infinitely fast, using no area), and constant AT products (which correspond to constant TpA values given in kbit/s·GE) are plotted to help visualization. It can be said that solutions that fall on the same AT curve are equally efficient, they represent just a different trade-off between operation speed versus the circuit area.

The ETH implementations were designed to achieve 2.488 Gbit/s throughput, which corresponds to a little more than 0.4 nanoseconds per bit operation speed. This value is marked in the graph with a dashed line. To aid visibility, only three data points corresponding to the results from **step A**, **step B**, and **step C** have been plotted for each implementation.

The final area and speed parameters for all implementations from **step C** are also listed in Table 2. The worst result for each category is given in italics, and the best result in boldface. The area results also include the percentage of area occupied by flip-flops (FFs).

For the power results in Table 3, first post layout simulations with extracted parasitics have been performed for 1,000 clock cycles and the circuit activity has been extracted to a Value Change Dump (VCD) file. Using these activity factors, the power consumption of the circuit is calculated using SoC Encounter power analysis feature. During this

[9] In the AT graph, A stands for Area, and T stands for Time, it should not be confused with Throughput per Area (TpA) used extensively in the paper. Obviously throughput and time are related, so constant AT product (shown as curves on the graph) also equals to a constant TpA value (annotated on the graph) as well.

Table 3. Power consumption of all implementations on the ASIC, post layout simulations, worst case process corner, 1.08 V, 125°C, clocked for 2.488 Gbit/s throughput. See Table 7 for preliminary measurement results.

Algorithm	**Block Size** [bits]	**Implementation**	**Latency** [cycles]	**Clk Freq.** [MHz]	**Power** [mW]
SHA-2	512	ETHZ	67	324	11.86
		GMU	65	316	9.16
BLAKE	512	ETHZ	57	276	34.80
		GMU	29	140	16.47
Grøstl	512	ETHZ	81	392	*50.50*
		GMU	21	102	46.01
JH	512	ETHZ	42	204	16.54
		GMU	43	209	17.80
Keccak	1088	ETHZ	24	54	**8.16**
		GMU	24	54	9.98
Skein	512	ETHZ	92	446	50.00
		GMU	19	92	26.19

analysis, for all implementations a clock rate has been chosen so that a throughput of 2.488 Gbit/s could be achieved[10]. Using a fixed throughput value equalizes the work done by each algorithm and allows a more direct comparison (At constant throughput, the energy consumption is directly proportional to the power value)[11]. The latency given in the table refers to the number of clock cycles required to process an input-block (without finalization).

In theory, the ETHZ implementations [12] should consume slightly more power than their GMU counterparts, since the GMU implementations were optimized for throughput per area, whereas the ETHZ implementations only try to achieve a given throughput. For most of the implementations this is indeed the case. As explained in Appendix A.5, for Keccak the ETHZ implementation turned out to be slightly more efficient, therefore it should not be surprising that the ETHZ implementation of Keccak also consumes less power than the GMU implementation. However, for JH there is no direct explanation why the ETHZ implementation consumes less power than the GMU version, even though the difference is less than 10%. For both BLAKE and Skein, the ETHZ implementations consume much more power than their GMU counterparts. In both cases, the problem can be traced back to implementation issues, and/or architectural decisions. It is important to note that without a second set of implementations, such differences could not be reliably detected.

5.1 Ranking of Implementations

There have been numerous studies on hardware performance evaluation of SHA-3 candidates. We present the relative ranking of SHA-3 candidate algorithms obtained in different studies in Table 4 and compare these with our own results.

[10] The names (worst, best, typical) refer to their timing performance. For power analysis, worst case conditions actually deliver the best results, since lower supply voltages are assumed. The results are given for the worst case conditions to remain consistent with the other results.

[11] At the moment measurement results from only one ASIC is available. These are given in Appendix B

[12] Note that on the actual chip, all ETHZ implementations reach clock rates required to achieve 2.488 Gbps throughput as can be seen in Table 6.

The studies in Table 4 differ in the optimization targets. We have listed the optimization target for each study. As explained in Section 2 the ETHZ cores in this study are aimed at achieving a specific throughput with the lowest area, which is not exactly the same as outright small area target.

The measurement results obtained from a recent ASIC designed by VT [4] have also been included. These are comparable to the GMU implementations in this study. Although a different technology has been used, both studies focus on TpA performance and use the third round specifications[13].

We have also included the result of two earlier studies [8] [6] which evaluated all cores with their second round specifications. For some cores (BLAKE, JH) the number of rounds have increased, for some others (Grøstl and Skein) some constants and functions have slight modifications between the second and the third round specifications, all of which would have an effect on their performance.

Some of the results in Table 4 are very close to each other. We have added a bracket '[' around the results which are within 10% of each other. Note that, some studies have concentrated only on one performance metric, and their performance for other factors may not reflect the true ordering of the algorithm. For example, the ETHZ implementations in this study tried to achieve results at a specific throughput. Therefore it is not surprising that many results are close to each other. In fact ideally all results would have been within 10% of each other.

Still the table shows some clear results. Keccak is fast, efficient, and uses the least amount of energy per bit in all studies listed in the table. Of all the SHA-3 candidates BLAKE delivers the smallest circuit size. Grøstl comes second in all throughput centric comparisons, however ranks near the end in energy per bit comparisons. More often than not, Skein has found itself in last position for efficiency and joins Grøstl in the energy per bit comparisons. JH is consistently ranked in the middle (positions 3-4) for almost all comparisons.

6 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 explain what we have done to address them.

6.1 Conflict of interest

The group of people working on hardware implementations of cryptographic functions is rather small, as such the authors of this paper have had ample contact with several of the authors of candidate algorithms. A former member of the Integrated Systems Laboratory (Luca Henzen) is the co-author of BLAKE, and there is an on-going collaboration and student exchange program between ETH Zurich and TU-Graz (involved in Grøstl) in the cryptography field. We do not believe that this has colored our results in any way.

6.2 Designer experience

In this work we used two different sets of implementations. The GMU implementations have been optimized with regard to TpA using the ATHENa [2] framework for FPGAs. The source code for these implementations were made available at the ATHENa website [3]. The ETHZ implementations were derived from our earlier work for SHA-3 candidates [6] and/or custom developed for this evaluation. The two groups (ETHZ and GMU) had no interaction during the development of the cores and pursued different goals for their implementation. This adds another level of confidence to the results, as the likelihood of introducing an obvious error has been reduced.

To enable a fair comparison, 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 [5] as well. In this way, other groups can replicate our results, and can find and correct any mistakes we might have made in the process.

[13] At the moment, the tables in the main text reflect, post layout numbers, initial measurement results are given in AppendixB

Table 4. A comparison of the relative ranking of the performance results for this work and other published accounts. Note that the studies use different optimization targets. Brackets '[' denote that the results are within 10% of each other.

Reference	**This Work**		**[4]**	**[8]**	**[6]**
Institute	GMU	ETHZ	VT	TU-Graz	ETHZ
Implemented as	ASIC	ASIC	ASIC	ASIC	ASIC
Technology	65 nm	65 nm	130 nm	180 nm	90 nm
SHA-3 specifications	3^{rd} round	3^{rd} round	3^{rd} round	2^{nd} round	2^{nd} round
Optimization target	TpA	Constant Throughput†	TpA	Throughput	Throughput
Throughput per Area (TpA)					
1 (best)	Keccak	Keccak	Keccak	Keccak	Keccak
2	SHA-2	SHA-2	⌈ Grøstl	SHA-2	BLAKE
3	JH	JH	⌊ SHA-2	Grøstl	JH
4	BLAKE	BLAKE	⌈ BLAKE	⌈ JH	Grøstl
5	⌈ Grøstl	⌈ Skein	⌊ JH	⌊ BLAKE	Skein
6 (worst)	⌊ Skein	⌊ Grøstl	Skein	Skein	—
Throughput					
1 (best)	Keccak	Keccak	Keccak	Keccak	Keccak
2	Grøstl	JH	Grøstl	Grøstl	Grøstl
3	JH	SHA-2	⌈ JH	JH	⌈ JH
4	⌈ Skein	⌈ Skein	⌊ Skein	BLAKE	⌊ BLAKE
5	⌊ BLAKE	⌊ BLAKE	BLAKE	SHA-2	Skein
6 (worst)	⌊ SHA-2	⌊ Grøstl	SHA-2	Skein	—
Area					
1 (best)	SHA-2	SHA-2	SHA-2	SHA-2	BLAKE
2	BLAKE	BLAKE	BLAKE	BLAKE	⌈ Keccak
3	JH	⌈ Keccak	Keccak	⌈ Keccak	⌊ Skein
4	Skein	⌊ JH	JH	⌊ Grøstl	JH
5	Keccak	Grøstl	Skein	⌊ Skein	Grøstl
6 (worst)	Grøstl	Skein	Grøstl	⌊ JH	—
Energy per bit					
1 (best)	⌈ SHA-2	Keccak	Keccak	—	Keccak
2	⌊ Keccak	SHA-2	SHA-2	—	BLAKE
3	⌈ BLAKE	JH	JH	—	Grøstl
4	⌊ JH	BLAKE	⌈ BLAKE	—	JH
5	Skein	⌈ Skein	⌊ Skein	—	Skein
6 (worst)	Grøstl	⌊ Grøstl	Grøstl	—	—

† The designs were optimized for 2.488 Gbit/s throughput, and small area.

6.3 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 them. 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.

6.4 Bias through assumptions

One of the contentious issues in this work is the choice of the throughput constraint used for ETHZ implementations at 2.488 Gbit/s. The goal of choosing this number was to create a more practical scenario and offer a counter balance for TpA optimized results. The number was chosen after an initial analysis of the performance of TpA optimized GMU implementations, so that a viable solution for all five candidate functions could be obtained[14]. However, the specific choice is not equally advantageous for all algorithms. Some of the algorithms would fare slightly better with a different value. We believe that it would not have been possible to find a *fairer* value for the throughput constraint.

A second problem is the choice of the interface, specifically that the input (and to a lesser degree the output) be available in parallel only during a short initial phase[15]. Implementations that could start with partial (i.e. 64-bit) message block (i.e SHA-2) do not gain any latency advantage with this arrangement. On the other hand, the current interface benefits algorithms that can transfer the entire message block to the internal state, and do not require the message block for further processing. These algorithms can spare the additional register (for 512 or 1088 bits) to store the input.

In this work, we have chosen all data inputs and outputs to be parallel. Different input and output width would also have some effect on the results. In addition, all cores assumed that the messages were padded to the correct length, and do not include any padding circuitry.

6.5 Only SHA-2 alternatives with 256-bit output length considered

For comparison purposes we have chosen to implement variants of the SHA-3 candidates with 256-bit outputs. However, there are in total four SHA-3 variants with output sizes equal to 224, 256, 384, and 512 bits respectively. Some functions (BLAKE, Grøstl, Skein) use a larger internal state register, some (BLAKE and Grøstl) have slightly modified functions while some others (BLAKE, Skein) require more rounds for block lengths of 384 and 512 bits. Designing a core that implements all four required block lengths would result in a larger overall circuit for these architectures. In contrast, JH would be virtually identical, as the exact same architecture is used for all block lengths. By using only a single block length, these differences in algorithms have not been considered.

7 Conclusions

For the purposes of the SHA-3 competition, we believe that our duty is to report the results of the implementations and explain how they were obtained clearly. We have presented the numerical results in Section 5 and have included detailed comments about individual algorithms in Appendix A. We have also compared the ranking of algorithms for four different performance metrics with other published work. In this section we want to highlight some general final observations that we believe is important in hardware evaluations:

- Synthesis results that do not include the effect of placement and routing could be more than 50% off when compared to actual post-layout results as they are unable to properly account for the impact of parasitics.
- There can be substantial differences between the performance of implementations due to seemingly inconsequential differences. At times the differences between GMU and ETHZ cores exhibit larger variation than what their chosen architecture might at first suggest. Even in an controlled environment (same technology, same tool chain, same libraries, same conditions) it is not easy to find conclusive justifications for the differences. It will be even harder to compare results between different groups.
- The current breed of EDA tools are not based on *finding the best performance* but are geared towards *finding a netlist that satisfies the given constraints*. This makes more academic endeavors such as finding the *optimal* netlist for a given RTL description more difficult.

[14] If we were to implement the algorithms for example in a 180nm process, this throughput would be too high, and several algorithms would have no implementation that would reach this throughput.

[15] Only the GMU Grøstl implementation requires that the input remains stable for more than one cycle, all other implementations sample the input with the `InWrEnxSI` signal.

– Even if implementations are clearly not optimized towards efficiency (such as the ETHZ implementations in this work), the results can reveal the *range* of the possible implementations of algorithms in hardware when used in comparison with more systematic implementations.
– Results from an actual ASIC will always be the ultimate proof of implementation. However any such implementation will have various compromises that will result in some performance loss which can not easily be quantified.

Acknowledgments

This project is evaluated by the Swiss National Science Foundation and funded by Nano-Tera.ch with Swiss Confederation financing. The ETHZ implementations make use of previous work by Luca Henzen, Pietro Gendotti, Patrice Guillet, Martin Zoller and Enrico Pargaetzi.

References

1. Ricardo Chaves, Georgi Kuzmanov, Leonel Sousa, and Stamatis Vassiliadis. Improving sha-2 hardware implementations. In *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop*, pages 298–310, 2006.
2. Kris Gaj, Jens-Peter Kaps, Venkata Amirineni, Marcin Rogawski, Ekawat Homsirikamol, and Benjamin Y. Brewster. ATHENa – Automated Tool for Hardware EvaluatioN: Toward fair and comprehensive benchmarking of cryptographic hardware using FPGAs. In *20th International Conference on Field Programmable Logic and Applications - FPL 2010*, pages 414–421. IEEE, 2010.
3. Kris Gaj, Jens-Peter Kaps, Venkata Amirineni, Marcin Rogawski, Ekawat Homsirikamol, and Benjamin Y. Brewster. Athena: Automated tool for hardware evaluation, 2011. `http://cryptography.gmu.edu/athena/`.
4. Xu Guo, Meeta Srivistav, Sinan Huang, Dinesh Ganta, Michael Henry, Leyla Nazhandali, and Patrick Schaumont. Performance Evaluation of Cryptographic Hardware and Software – Performance Evaluation of SHA-3 Candidates in ASIC and FPGA, November 2010. `http://rijndael.ece.vt.edu/sha3/`.
5. Frank K. Gürkaynak, Luca Henzen, Pietro Gendotti, Patrice Guillet, Enrico Pargaetzi, and Martin Zoller. Hardware evaluation of the second-round SHA-3 candidate algorithms, 2010. `http://www.iis.ee.ethz.ch/~sha3/`.
6. Luca Henzen, Pietro Gendotti, Patrice Guillet, Enrico Pargaetzi, Martin Zoller, and Frank Gürkaynak. Developing a hardware evaluation method for sha-3 candidates. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 248–263, 2010.
7. NIST. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. Federal Register, Vol.72, No.212, 2007. `http://www.nist.gov/hash-competition`.
8. Stefan Tillich, Martin Feldhofer, Mario Kirschbaum, Thomas Plos, Jörn-Marc Schmidt, and Alexander Szekely. 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. `http://eprint.iacr.org/`.

A Individual Results

The design process that we have employed in this project consists of several distinct steps as explained in section 4. Algorithms were first optimized individually (**steps A** and **B**). The results therefore *evolved* throughout this process and in this section we would like to present the progression of the individual results. This is relevant because any single performance result is likely to have a rather substantial error associated with it. The solution space with results from multiple design steps as we have presented here could give a slightly more refined idea about where the actual limits of individual algorithms may lie.

For each algorithm we show two graphs, one for GMU and one for ETHZ implementations. These graphs have been constructed the same way as Figure 3. To obtain these graphs we followed these steps:

– In **step A** we synthesized the circuits with different maximum clock period constraints, and examined the performance for each resulting netlist. For GMU implementations we selected the implementation with the highest TpA figure, while for ETHZ implementations the main goal was to choose an implementation with a Throughput of at least 2.488 Gbit/s. The point *Default Wireload (A)*, in all graphs represents this chosen point.

– Using the netlists generated from **step A**, a chip-level netlist containing all cores was generated. From this chip-level netlist, the wireload model was extracted for each core separately. A new synthesis step was performed using these wireload models by using different maximum clock period constraints. The results obtained through this synthesis run have been marked with red cross signs and are labeled as *Synthesis Exploration* in the graphs.
– The best fitting (high TpA for GMU, 2.488 Gbit/s for ETHZ) netlist was selected and is labeled as *Extracted Wireload (B)* in the graph. Note that for several algorithms, the ETHZ implementations were unable to meet the given timing constraint.
– Using the netlists generated in **step B** the final chip-level netlist was generated and for each core the actual post-layout performance was determined. This point is denoted as C on the graph.
– After the ASIC was sent to fabrication, a number of individual placement and routing runs have been performed in **step D**. The result of these runs have been plotted on the graph using blue + signs and labeled as *P&R Exploration*

A.1 SHA-2

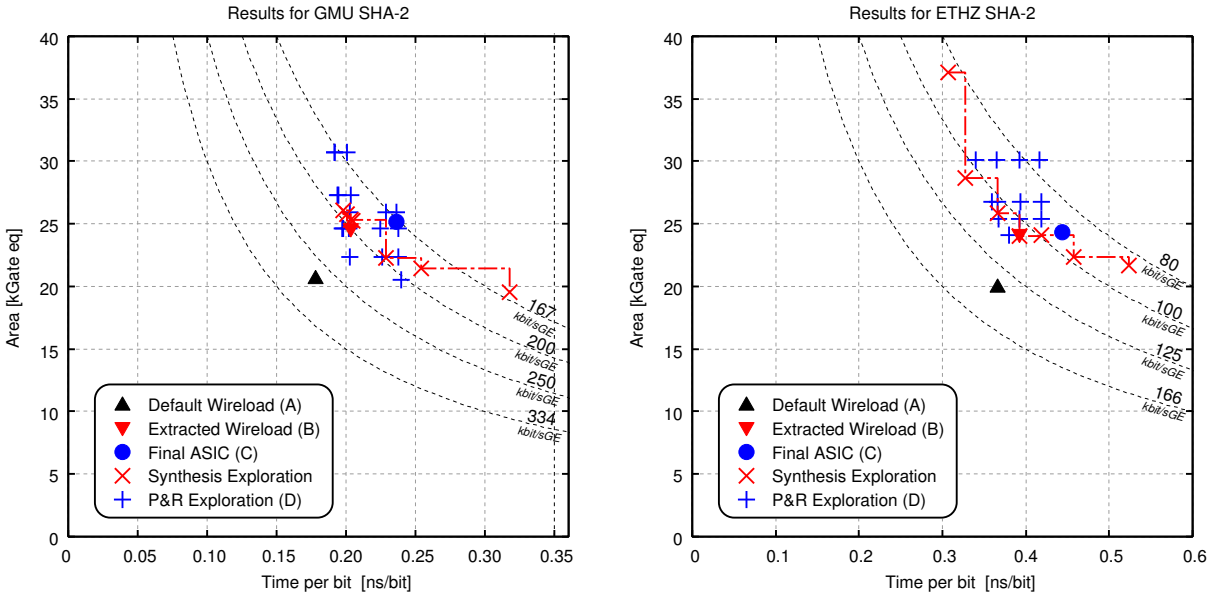

Fig. 4. AT plot for SHA-2 implementations: GMU (left) and ETHZ (right). Note that the X axis is scaled differently.

The SHA-3 selection process is designed to *augment and revise* the current FIPS-180-2 standard that defines the Secure Hash Standard. It is therefore normal that all candidate algorithms are compared to SHA-2. Unfortunately for all other candidates, SHA-2 can be implemented efficiently in hardware. Part of it is due to the fact that the algorithm has been known for quite some time, and more time was available to develop more optimized implementations. In fact, the GMU implementation is based on the implementation by Chavez et al. [1].

The ETHZ implementation is a more straightforward implementation of the algorithm, and the initial hope was that the lower throughput requirement would result in a smaller area. It can be seen that both implementations are equally large, with the ETHZ implementation being significantly slower.

The AT plot for the two implementations in Fig. 4 shows that the default wireload used in **step A** was slightly optimistic. However, there is good agreement between the back-end results (plus signs) and the front-end results

(crosses). For both implementations, the circuit that was used in the ASIC (**step C** denoted by the circle) is about the same size as the netlist obtained through the synthesis **step B**, it is only slightly slower.

Around one third of the area in SHA-2 is spent for the registers holding the state (see Table 2. This is a pretty large proportion and limits the options for possible optimizations in hardware, as the number of FFs (barring implementation mistakes) can not be reduced further.

A.2 BLAKE

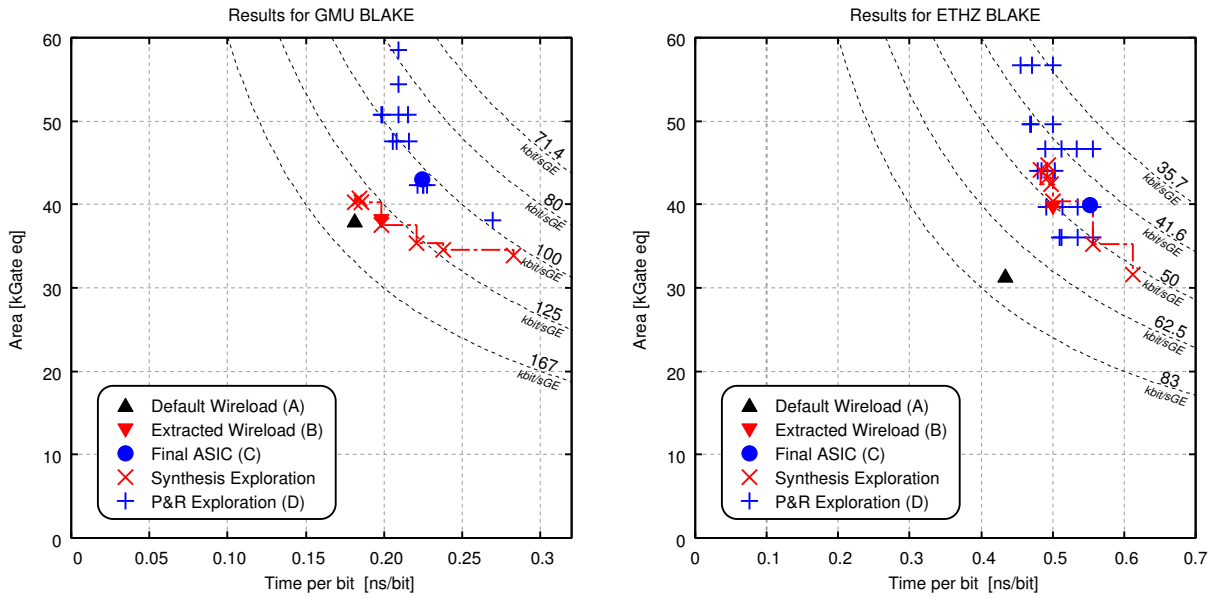

Fig. 5. AT plot for BLAKE implementations: GMU (left) and ETHZ (right). Note that the X axis is scaled differently.

The core of the BLAKE algorithm is the G function, a 128-bit combinational function that is evaluated 8 times per round. The basic architectural choice for BLAKE is the number of G-functions that will be evaluated in parallel. Practical choices are 1, 2, 4 and 8 parallel G-functions. The GMU implementation uses 4 G-functions in parallel, whereas the ETHZ implementation chose only 2 parallel G-functions.

Results in Fig. 5 prove that the algorithm scales pretty well in throughput but not so much in area. At first, this might be surprising. After all, the ETHZ implementation has half the amount of G-functions. There are two main reasons for this discrepancy. First of all, similar to SHA-2, comparatively a large portion of the area (25-35%) is used for the state registers, less area is used for the function itself, resulting in less saving. Secondly, in order to achieve the timing constraints, ETHZ implementations had to be constrained harder, resulting in hardware that is less efficient[16].

The most interesting part about these results is that for the GMU implementation, the results from **step B** seem to be further away from the real back-end solutions (in contrast to the ETHZ implementations). It can also be seen that the final circuit is very close to the maximum that was obtained in the later back-end experiments (**step D**). Since the constraints for the final stage were derived from the results of **step B** GMU BLAKE was over constrained in the process. The difference in the results for these two implementations is actually alarming, as the descriptions of the

[16] This is a common phenomenon. If you apply stricter timing constraints, these are more difficult to attain, the tools will sacrifice more area to achieve the timing constraint. In the end, perhaps the timing will be met, however the result will be less efficient.

two implementations are very similar to each other, yet the extracted wireload for **step B** produced two very different results.

BLAKE is one of the functions with the longest critical path mainly because several 32-bit additions are used in one G-function. This is actually a mixed blessing. On the positive side, there are several well-known implementations for a binary adder with different area-speed trade-offs, allowing the algorithm a wider range of implementations in general. On the other hand, long critical paths translate to slower clock rates, and limit the maximum throughput in iteratively decomposed implementations.

A.3 Grøstl

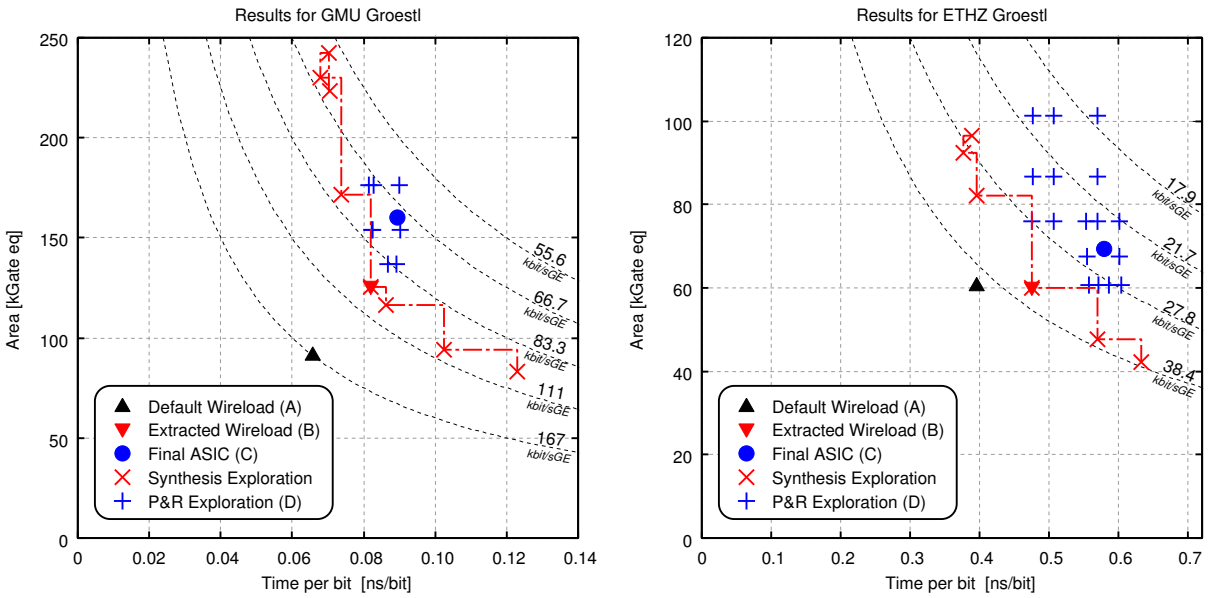

Fig. 6. AT plot for Grøstl implementations: GMU (left) and ETHZ (right). Note that both axes are scaled differently.

Throughout the results it can be seen that, Grøstl consistently demands the largest circuits. Furthermore, at least in the GMU implementation, the differences between results in **step A** to the rest are very high (see Fig 6), suggesting that the implementations incur significant overhead in the back-end flow. Part of this is due to the many AES look-up tables needed for the core functions of Grøstl called P and Q. Both functions require in total 64 AES *SubBytes* functions per round. The GMU implementation uses a combined block that can be configured to run as both P and Q, with 64 look-up tables.

Although P and Q functions are both very similar, the differences add non-negligible hardware. As an example, the *AddRoundConstant* function in essence requires sixty-four, 2-input XOR gates for both P and Q. However, since different bits of the state are involved for each function, a straightforward implementation requires 128 XOR gates and 128 2-input multiplexers. The ETHZ implementation uses two specialized units P and Q each with 8 look-up tables. This allows simpler units, but has a significant drawback when implementing the finalization step, which uses only the P function, as only half of the look-up tables are available, doubling the time needed for the finalization step.

In theory, Grøstl offers a wide range of implementations, when the number of look-up tables per round is considered. A fully parallel implementation would use up to 128 look-up tables, and an implementation could be designed

using only a single look-up table. However, our results show a different story. Although the ETHZ implementation has only one fourth of the look-up tables it is only around half the size. The increased critical path in the ETHZ implementation is due to the logic required to share the look-up tables.

One other problem with Grøstl is the relatively long run times required for synthesis. Table 5 shows that Grøstl for both implementations is among the most demanding algorithms in terms of CPU time. It must be noted that some of this extra effort is clearly due to the size of the circuits.

A.4 JH

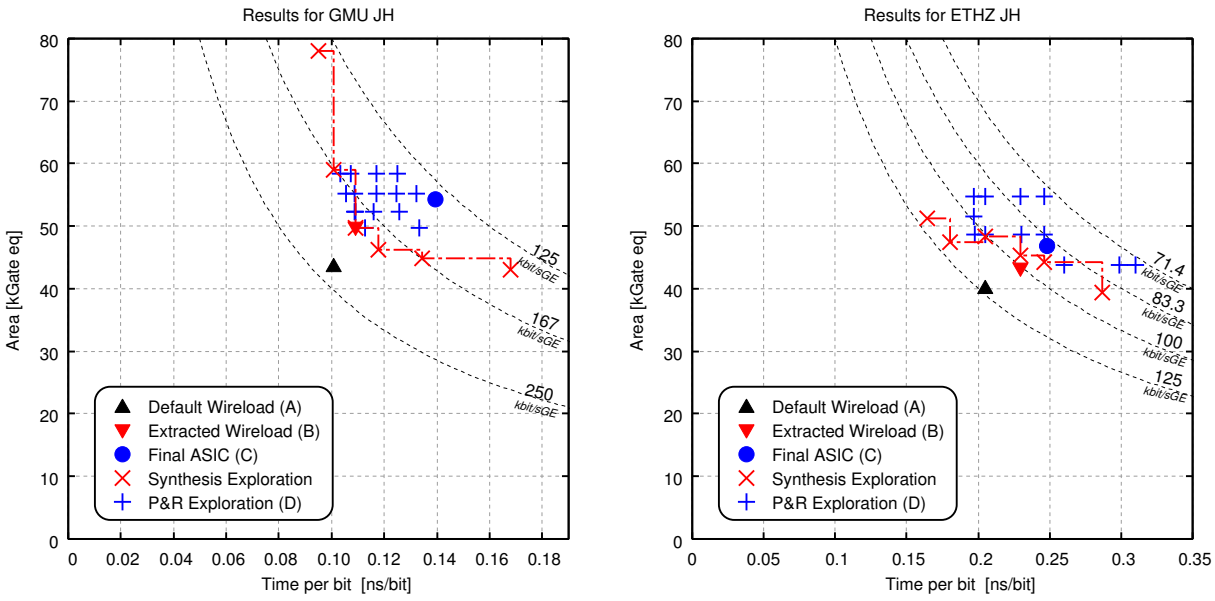

Fig. 7. AT plot for JH implementations: GMU (left) and ETHZ (right). Note that the X axis is scaled differently.

JH is generally a very fast algorithm when implemented in hardware. Similar to Keccak we were unable to find a version that would trade off speed with smaller area. This is why the ETHZ and GMU implementations are very similar in style, each implementing one full round in hardware. The main difference between the two implementations

Table 5. Synthesis times for all candidate algorithms on an Intel Core2 Duo CPU running at 3.16 GHz with 8 Gbytes of memory, using a single core

Algorithm	GMU implementation	ETHZ implementation
SHA-2	262 s	179 s
BLAKE	240 s	942 s
Grøstl	2'076 s	632 s
JH	565 s	343 s
Keccak	671 s	171 s
Skein	348 s	1'182 s

is that the GMU version uses an additional cycle for the initial round reducing the critical path. Since the throughput at this configuration is much higher than the targeted 2.488 Gbit/s, the ETHZ implementation has more relaxed timing constraints.

The interesting property of JH is that, comparatively it is less affected by placement and routing phase. Figure 7 shows that for both implementations the results from all steps are closer together, suggesting that the implementations are less susceptible to changes due to interconnection parasitics.

A.5 Keccak

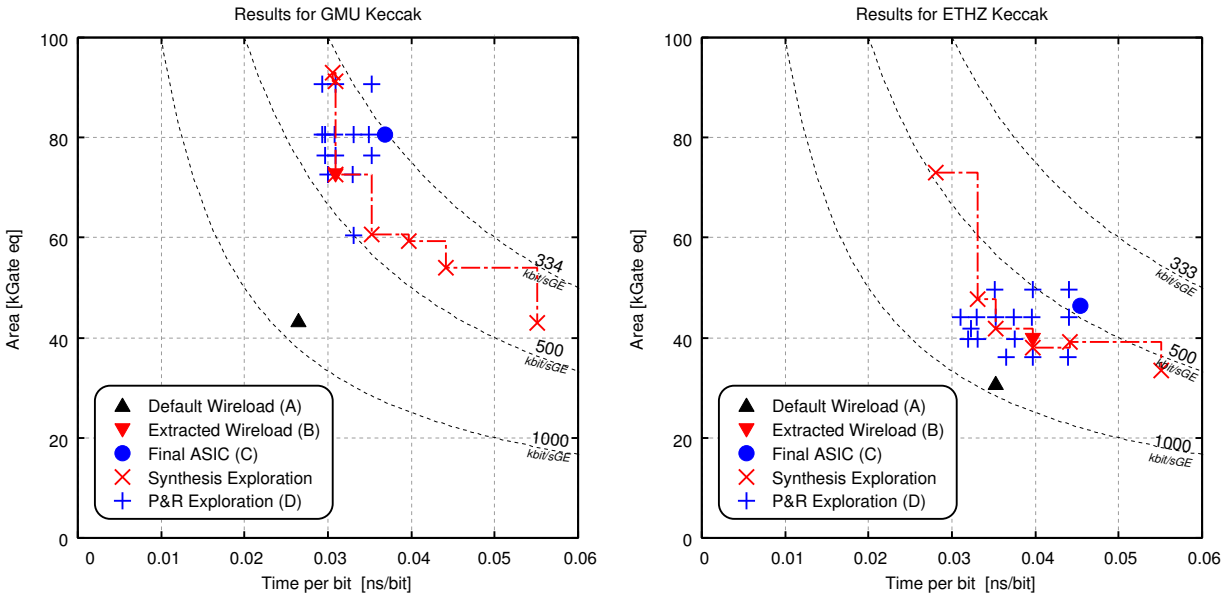

Fig. 8. AT plot for Keccak implementations: GMU (left) and ETHZ (right).

When it comes to throughput or throughput per area, no SHA-3 candidate can come close to Keccak. It is more than twice as fast as its nearest rival (Grøstl), and twice as efficient its nearest rival (SHA-2) when the throughput per area numbers are examined. However, in terms of speed-area trade-offs it is remarkably inflexible. What Keccak excels in is the flexibility in the size of the width of the f permutation. Virtually the same architecture could be used for f-1600 and for example f-400, by simply changing a single parameter. However, for all SHA-3 message digest sizes, the same f-1600 function will be used, and this flexibility will go largely unused.

The GMU implementation uses a standard one-round-per-clock-cycle implementation. The very high throughput is mainly the result of a (relatively) small number of rounds (24) and a large number of message bits processed at a time (1088). For the ETHZ implementation various configurations were tried, it was possible to reduce the throughput, but nothing could be gained out of area, instead often times a (slightly) larger circuit was produced. The ETHZ implementation is virtually the same as the GMU implementation, however it has been synthesized using more relaxed timing constraints.

When compared to other candidates Keccak implementations suffered the most in the back-end. The ETHZ implementation ended up being 50 % larger, and the GMU implementation even 85 % larger, the most in this study.

However, even with these penalties, Keccak implementations are not particularly large. Among the ETHZ implementations, Keccak is the second smallest (with a very small margin) of the SHA-3 candidates. The GMU implementation is considerably larger, and ranks as the second largest, trailing only Grøstl.

Although both implementations are very similar, results in Fig. 8 show very different characteristics. The GMU implementation, for reasons unexplained, seems to suffer much more from parasitic interconnections in the back-end design than the ETHZ implementation. As a result, the ETHZ implementation reaches a throughput per area figure that is 40% higher than that of GMU implementation, which was totally unexpected. Although we have control over the entire design flow, and can guarantee that both implementations were processed the same way, we are unable to account for the difference in both implementations sufficiently[17]. It is not unreasonable to assume that there could be much larger discrepancies between results from two unrelated studies.

A.6 Skein

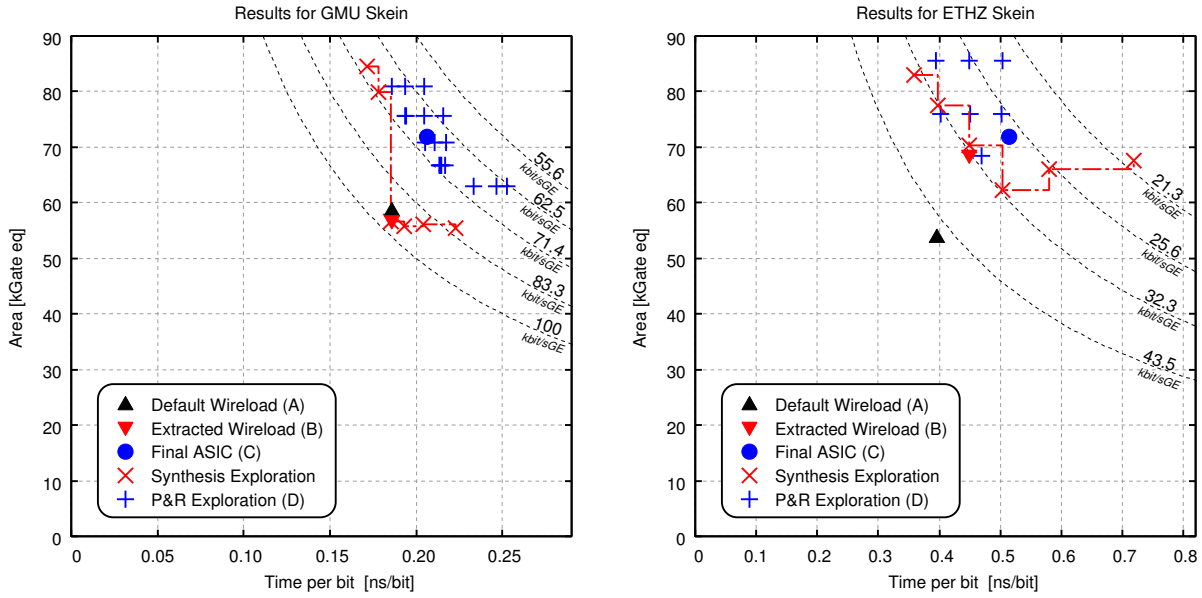

Fig. 9. AT plot for Skein implementations: GMU (left) and ETHZ (right). Note that the X axis is scaled differently.

Skein is based on the Threefish block cipher. In total 72 rounds of the Threefish function are needed for the implementation. Each Threefish round consists of a *Mix* function followed by a *Permute* function. After four such Threefish rounds a subkey addition is performed. The GMU implementation uses a block that implements four Threefish rounds, while the ETHZ implementation calculates each step separately.

The circuit complexity in the Skein implementation is to a large extent due to the large 64-bit binary adder in each *Mix* function. Similar to BLAKE the presence of an arithmetic building block with well-known area/speed trade-offs gives more choice for implementations. On the other hand, there are sometimes unexpected consequences. When the final results in Table 2 are examined, it can be seen that the ETHZ implementation is almost exactly the same size,

[17] Selecting slightly less strict constraints, as done with the ETHZ implementation, seem to bring more efficient results. Between **step A** and **step B** both implementations incurred a 400 ps addition to their critical path, which affected the GMU implementation more as it was targeted for 1.2 ns clock speed, as opposed to 1.6 ns for ETHZ.

but has less than half the throughput, making this implementation particularly inefficient. The initial reasoning behind the ETHZ implementation was to reduce the critical path and the total area by a factor of four by implementing a single Threefish round. Obviously the latency would also increase by a factor of four[18], but this would be in part compensated by a higher clock rate. The problem with this approach is that the critical path through four consecutive 64-bit binary adders is not really four times the delay through a single adder. It is true that due to carry propagation that the MSB of the adder output has a significant delay, but the LSBs will be available in shorter time, at which point the subsequent adders would already have the necessary inputs and could start computing the result. Such multi-operand adders can therefore be realized using smaller adders for the first three operands, thereby also reducing the overall area. In contrast, for the implementation with only a single adder, the propagation delay of the single adder becomes critical, and in the end an extremely large adder has to be implemented.

The results in Figure 9 are also very interesting. For the GMU implementation, **step A** and **step B** seem to overlap, in other words, using a realistic wireload model has no discernible difference to the results. However, in the back-end, obtainable results differ greatly from **step B**. In contrast, for the ETHZ implementation there is a noticeable difference between **step A** and **step B**, however the back-end results seem to be much closer to the ones obtained from **step B**. This is a particularly nice example showing how much results could differ even if the same design flow, and the same design tools are used for functionally equivalent implementations of the same algorithm. If only the results from the GMU implementation were to be considered, it can be said that **step B** was unnecessary, as the results were far away from **step C**, while when the ETHZ implementation is considered, it could be argued that synthesis results without a representative wireload (**step A**) are not able to capture the performance of the circuit.

B Measurement Results

In this section, we have updated the tables 2 and 3 with preliminary measurement results from one fabricated ASIC. It can be seen that all cores have much higher performance, and the power consumption is higher. Both are expected, as for the measurements nominal VDD (1.2 V) has been used, whereas for the estimations the worst case corner VDD (1.08 V) has been used. Also note that all ETHZ cores achieve their targeted throughput of 2.488 Gbps.

Not all cores have changed at the same rate. Particularly both SHA-2 implementations have incurred only 10% power penalty, whereas all the rest are at least 40% to 62% higher. Similarly, the maximum clock frequency of GMU implemenations of SHA-2, Gr/ostl, JH, and Keccak (14%-28%) has not increased at the same rate as the other cores (62%-92%).

As mentioned earlier, results from a single ASIC are far from conclusive, and therefore these results have not been included in the main text. As soon as more results are available the results will be adjusted accordingly.

[18] The ETHZ implementation uses an additional cycle for the subkey, so the latency increases by a factor of five

Table 6. Area, Throughput. and Throughput per Area figures for all implemented cores on ASIC. Measurement results from one ASIC, VDD=1.2 V, room temperature.

Algorithm	**Block Size** [bits]	**Implementation**	**Area** (FFs) [kGE]	**Max. Clk Freq.** [MHz]	**Throughput** [Gbit/s]	**Throughput per Area** [kbit/s·GE]
SHA-2	512	ETHZ	**24.30** (29%)	563.38	4.305	177.156
		GMU	25.14 (35%)	687.29	5.414	215.342
BLAKE	512	ETHZ	39.96 (26%)	378.07	3.396	84.979
		GMU	43.02 (34%)	409.84	7.236	168.179
Grøstl	512	ETHZ	69.39 (17%)	449.44	*2.841*	*40.939*
		GMU	*160.28* (9%)	586.51	14.300	89.217
JH	512	ETHZ	46.79 (27%)	537.63	6.554	140.067
		GMU	54.35 (31%)	692.04	8.240	151.609
Keccak	1088	ETHZ	46.31 (25%)	711.74	**32.266**	**696.703**
		GMU	80.65 (19%)	696.86	31.591	391.707
Skein	512	ETHZ	71.87 (19%)	591.72	3.293	45.817
		GMU	71.90 (22%)	325.73	8.778	122.083

Table 7. Power consumption of all implementations on the ASIC. Measurement results from one ASIC, VDD=1.2 V, room temperature, clocked for 2.488 Gbit/s throughput.

Algorithm	**Block Size** [bits]	**Implementation**	**Latency** [cycles]	**Clk Freq.** [MHz]	**Power** [mW]
SHA-2	512	ETHZ	67	324	13.13
		GMU	65	316	**10.23**
BLAKE	512	ETHZ	57	276	53.78
		GMU	29	140	26.25
Grøstl	512	ETHZ	81	392	71.09
		GMU	21	102	60.54
JH	512	ETHZ	42	204	25.40
		GMU	43	209	28.89
Keccak	1088	ETHZ	24	54	12.83
		GMU	24	54	16.23
Skein	512	ETHZ	92	446	*73.39*
		GMU	19	92	41.63