Lightweight Implementations of SHA-3 Finalists on FPGAs*

Jens-Peter Kaps Panasayya Yalla Kishore Kumar Surapathi Bilal Habib
Susheel Vadlamudi Smriti Gurung

ECE Department, George Mason University, Fairfax, VA 22030, U.S.A.
http://cryptography.gmu.edu

Abstract

The NIST competition for developing the new cryptographic hash algorithm SHA-3 has entered its third round. One evaluation criterion is the ability of the candidate algorithm to be implemented on resource-constrained platforms. This includes FPGAs for embedded and hand-held devices. In this paper we present two sets of lightweight implementations of all SHA-3 finalists and SHA-2, one using only logic resources (slices) and one which additionally uses one Block RAM. All implementations were designed to achieve maximum throughput while adhering to an area constraint of 800 slices or 400-600 slices and one Block RAM, respectively, on Xilinx Spartan-3 devices. We also synthesized them for Virtex-5, Altera Cyclone-II, and the new Xilinx Spartan-6 and Virtex-6 devices to examine the influence of device choice and to allow for comparison with other reported results. Furthermore, we measured the power consumption of all implementations on Spartan-3 to evaluate the efficiency of the algorithms.

1 Introduction and Motivation

The National Institute of Standards and Technology (NIST) started a public competition to develop a new cryptographic hash algorithm in November 2007. From the submitted 64 entries only 14 were selected for the second round of the competition and in December 2010, the 5 Secure Hash Algorithm-3 (SHA-3) finalists were announced. NIST is expected to announce the winner in 2012. One important criterion that the hash algorithm should fulfill in order to become the next American hash standard SHA-3 is its ability to be "[...] implemented securely and efficiently on a wide variety of platforms, including constrained environments, such as smart cards"[1]. Unfortunately, designing low-area implementations is not as straightforward as optimizing a design for best throughput over area. One has to go beyond merely reducing the datapath width and carefully evaluate the trade-off speed vs. area at every step of the design process. The control unit is an additional hurdle. Extensive component re-use in the datapath can lead to a very complex control logic which might negate the area savings in the datapath.

There have been several publications that show low-area implementations of single SHA-3 candidates on FPGAs such as BLAKE [2], Grøstl [3], Keccak [4], and Skein [5][6]. Unfortunately, they are implemented on different FPGAs from different vendors and with different target sizes. The first publication that covers implementations of all SHA-3 finalists is [7] followed by [8] and [9]. The later publication is from the same research group as this paper and also covers 13 SHA-3 round 2 candidates. Kerckhof et al. [7] implemented 256-bit digest versions of all SHA-3 finalists on Virtex-6 devices. Their optimization goal was to achieve maximum throughput over area ratio within a few hundred slices using a 64-bit datapath. Junkg et al. [8] developed area-optimized implementations of 256-bit digest versions of all SHA-3 finalists on Virtex-5 devices. While both papers contribute greatly to the state-of-the art in low area implementations of hash functions through introducing new and innovative design techniques, they do not lend themselves well to obtaining

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

a fair ranking of algorithms. In [7] the area consumption of the algorithms varies up to 2.2 times and the resulting throughput 7.8 times with the smallest implementation having the worst throughput and the largest the highest throughput. The variations in [8] with 2.7 for area and 42.4 for speed are even larger. We believe that only if one criterion is fixed (e.g. area or throughput) a fair comparison can be made.

In [9] we fixed the area to around 500 slices and 1 Block RAM and asked the question: "Given an area budget, what is the fastest algorithm". In this paper we expanded upon our previous work by introducing an improved datapath for JH and adding new implementations of all SHA-3 finalists designed for a new area requirement that does not use Block RAMs but only logic resources (slices). This allows for an interesting analysis on the impact of using Block RAMs and for fair comparison with reported results of other groups who do not use Block RAMs either. Furthermore, we implemented SHA-2 for both area requirements to examine which SHA-3 candidates can outperform SHA-2. A very important metric for implementations on embedded devices is the power consumption. Westermann et al.[10] measured and compared the power consumption of all round 2 candidates on a personal computer. However, to the authors knowledge there have not been any power/energy consumption results reported of the SHA-3 candidates on FPGAs. We measured the average and maximum power consumption of our implementations, and ranked the algorithms based upon the energy needed to hash a message per bit of message.

The remainder of the paper is organized as follows. In Sect. 2 we present the design methodology we used including clear assumptions and goals, the power measurement technique and performance metrics. We describe the datapaths of the five SHA-3 finalists and SHA-2 in detail in Sect. 3. Section 4 shows the results of our implementations and compares them amongst each other and with implementations of [7] and [8].

2 Methodology

The primary target for our lightweight implementations are the low-cost Xilinx Spartan-3 FPGAs. We choose VHDL to describe our lightweight architectures. All implementations were designed at a low level for our main target FPGA family such that we can already obtain a rather precise estimate of the required area from detailed datapath diagrams. This approach allowed us to enforce a similar coding style across several designers and algorithms. Furthermore, we built a small VHDL library of elementary functions that was used by all designers.

2.1 Assumptions and Goals

Only SHA-3 variants with 256-bit digest have been implemented as these are the most likely variants to be used in area-constrained designs. Furthermore, we assume that padding is done in software. This assumption goes hand-in-hand with the application of hash functions to SOC designs. The salt values of all SHA-3 candidates who support them are set to zero. Typical optimization goals for hardware implementations are: maximum throughput, maximum throughput to area ratio, and minimum area. In order to compare lightweight implementations the minimum area target seems logical. However, optimizing the implementations for minimum area would yield a ranking of algorithms solely based on area, i.e. we would know which is the smallest and which is the largest irrespective of the throughput that is achieved by these implementations. This information is of not much use in practice. A different approach is to optimize for throughput given an area constraint. We believe that this is a much more realistic scenario. Additionally this optimization goal lets us determine how efficient an algorithm is in a constrained environment which is a factor of an algorithm's flexibility. This is a clearly stated evaluation criterion by NIST [1]. Resource consumption of an FPGA is characterized by the number of logic resources (slices for Xilinx, LEs for Altera), Block RAMs, and Multipliers consumed. In this paper we choose to explore two area targets, one using only logic resources of up to 768 slices, which is equivalent to the smallest Xilinx Spartan-3 device (xc3s50pq) and one using between 450 to 650 slices and 1 Block RAM on Xilinx Spartan-3 FPGAs. The size of the ranges was chosen based on our previous low-area implementation results. Within these area constraints we try to achieve maximum throughput. Therefore, our final comparisons will be in terms of the ratio of throughput to area.

2.2 Tools and Result Generation

Even though all designs were targeted for Spartan-3 devices it is interesting to see how our implementations perform on low-cost devices from another vendor such as Altera Cyclone-II, high speed devices such as Xilinx Virtex-5, and on newer devices such as Spartan-6 and Virtex-6. All results will published in the ATHENa results database [11]. All designs were implemented using the vendor tools: Xilinx ISE 13.1 Web Pack and Altera Quartus II v. 10.0 Web Edition, and verified after place-and-route against known answer test files provided by the submissions packet of each hash function. All results were generated using the open source benchmarking tool ATHENa (Automated Tool for Hardware EvaluatioN) [12]. Other than simplifying the result generation, ATHENa also varies the vendor tool parameters to achieve optimal results.

2.3 Interface and Protocol

We based our hardware interface and I/O protocol on the one we used in [9] which has its origins in [13] and was updated in [14]. The SHA Core assumes that its inputs and outputs are connected to FIFOs.

2.4 Power Measurement Methodology

In order to measure the power consumption of a hash function on an FPGA, we have to build a separate circuit that supplies it with test vectors and checks the results. We call this separate circuit the *Wrapper* as it envelopes the hash circuit. The Wrapper and the SHA Core are connected via the FIFO interfaces mentioned in Sect. 2.3 as shown in Fig. 1. We implemented the Wrapper on a Nexys-2 board and the SHA Core on a Spartan-3E Starter Board. Both boards are connected via a a bridge connector which is wired for all FIFO signals, clock, reset and ground. Each board has its own power supply, hence we are able to measure the power consumption of the SHA Core independently of the Wrapper. Observing the signals of the FIFO interfaces allows the Wrapper to identify which phase the SHA Core is(*Idle*, *Loading* a block, *Processing* a block, *Writing* a hash value) and generate a trigger signal accordingly.

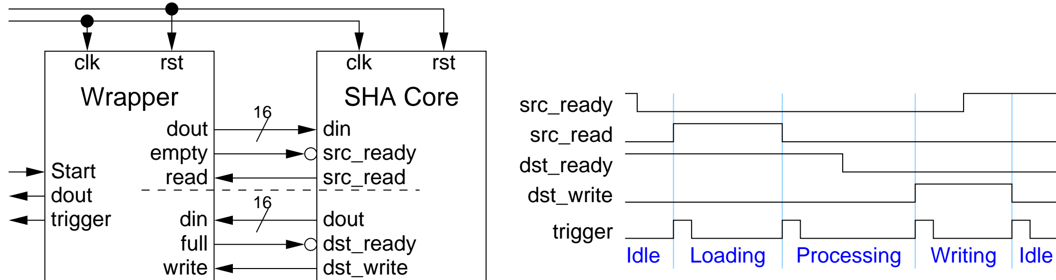


Figure 1: Power Measurement Setup

A Xilinx Spartan-3 FPGA uses three supply voltages: V_{Core}, V_{Aux}, and V_{Out}. The supplies V_{Aux} and V_{Out} are used for I/O, clock managers, JTAG, etc. All internal logic functions of the FPGA are powered by V_{Core}. Even though measuring the power supplied through V_{Aux} and V_{Out} would be needed for a full power profile, we expect that there will not be a significant difference in these power consumptions as neither of our implementations uses any internal feature of the FPGA powered by them other than I/O. As we test all cores with the same messages and receiving equal length digests, the power consumed due to I/O signals should be very similar. Hence, we only measure the power consumed from V_{Core}.

The power consumption consists of a static and a dynamic part: $P_{Total} = P_{Static} + P_{Dynamic}$. P_{Static} is constant while $P_{Dynamic}$ is proportional to the clock speed. In order to estimate the power consumption for a different clock speed than the one we use for measurement, it is important to know both. We obtain P_{Static} by measuring I_{Core} drawn from the V_{Core} supply when neither clock nor any input values are applied. Once we apply a clock of 50 MHz we record I_{Core} during the *Loading*, *Processing*, and *Writing* phases, identify

the maximum and calculate the average value to obtain $P_{Total} = V_{Core} \cdot I_{Core}$ which allows us to calculate $P_{Dynamic}$. The circuit we use to measure I_{Core} is shown in Fig. 2.

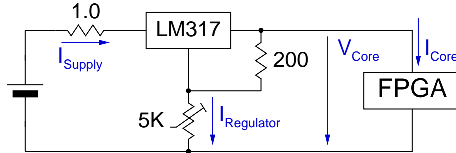


Figure 2: Power Measurement Circuit

2.5 Area Minimization Techniques

Datapath: The most straightforward approach to reducing the area of the datapath is folding. Vertical folding reduces the datapath width while horizontal folding reduces the size of processing elements while maintaining the datapath width. How many times and in which direction a design can be folded depends on the algorithm. The extent to which folding can be applied to the SHA-3 candidates and how much it affects their throughput and throughput over area ratio has been examined by Homsirikamol et al. [15]. They show that only BLAKE can reach our area constraints through folding alone, Grøstl remains too large, JH area increases when folded, and Keccak as well as Skein cannot be folded at all and hence far exceed our area constraints. Another technique is reusing of processing elements. We heavily use this technique and additionally, we apply vertical folding at multiple levels down to single processing elements, not just the datapath as a whole as done in [15]. For example the Skein algorithm uses 4 Mix functions each using a 64-bit adder and a 64-bit XOR. We fold the 4 Mix functions into 1 and within the Mix function we reuse a 32-bit adder to perform 64-bit additions. The same adder is also reused for the key injections. Both folding and reuse of processing elements minimize the area consumption at the cost of an increased number of clock cycles. We reduced this increase to some extent by interleaving operations through pipelining. In some of the algorithms like SHA-2, straight forward application of pipelining techniques is not trivial due to data dependency. So an optimization technique called quasi-pipelining is applied to reduce the critical time. These technique rearranges order of operations along with introduction of pipeline registers which optimizes the critical path along with reduction in latency.

Logic Resources: In Xilinx Spartan-3 FPGA devices, there are two types of slices SLICELs and SLICEMs. SLICELs has LUTs which can configured only for logic implementation. But SLICEM LUTs can be configured either as memory or logic. In memory mode, SLICEM LUTs can be configured either as 1x16 DRAM called RAM16 or 16-bit shift register called SLR16. The RAM16s can be configured as single-port or dual-port memory units. The dual-port DRAM is not a true dual-port as only one port can be used as read/write port while the other as read only port.

Block RAM: Block RAMs (BRAMs) offer a large amount of memory space for storage but have a limited number of ports and I/O lines. Xilinx Spartan-3 BRAMs can be configured as single or dual port memories with a maximum data width of 64 bits or 32 bits per port, respectively. Each port is associated with a single address input. This limits the number of independent values and the number of bits that can be accessed in a single clock cycle. Our Grøstl design processes four 8-bit values in each clock cycle. Even though these are only 32 bits, a dual port BRAM does not allow reading of four independent values in one clock cycle. Hence, we store that data in 4 Distributed RAMs. The Spartan-3 BRAM data sheet specifies that data is written to the address applied in the current clock cycle, but read from the address of the previous clock cycle. Hence, computing $Mem[i] = Mem[i] + k$, where each element is a 64-bit word, requires 2 clock cycles per address location i, i.e. dedicated write cycles. These are not needed when computing $Mem[i + i] = Mem[i] + k$, i.e. when an address shift is acceptable. In our early Keccak design, this address shift increased the complexity of the control logic and with it the area consumption beyond our

constraint. Hence it now uses dedicated write cycles. The new Xilinx Spartan-6 and Virtex-6 devices allow for independent read and write addresses for 64-bit data width.

Control Logic: The control logic of our implementations consists of a main finite state machine (FSM) with up-to 8 states, a single counter to count the clock cycles per state, and ROM-based FSMs for each state of the main FSM. ROM-based FSMs are more efficient in terms of area consumption and speed compared to conventional FSM [16], [17], [18], and their maximum frequency is independent of the complexity. However they are more complex to design. The area required to implement ROM-based FSMs is determined by the number of control signals and states. In order to reduce the number of control signals we try to use bits from the counter output, the main finite state machine, and simple boolean logic combinations thereof wherever possible. Furthermore, short sequences of control signals are placed in sub-controllers. The complexity of address generation for BRAMs can be reduced by placing datasets in memory locations starting at addresses which are a power of 2.

2.6 Performance Metrics

The number of clock cycles needed to hash N message blocks using our implementations can be computed as the sum of the number of clock cycles needed for the initial steps before processing can begin st, loading and processing one block of data $l + p$, and finalization and output of the hash value end.

$$clk = st + (l + p) \cdot N + end \tag{1}$$

Throughput is defined as the number of input bits processed per unit of time. The precise formula for throughput of a hash function is dependent on the number of message blocks N to be hashed, the block size b of the algorithm, the number of clock cycles needed to hash the message clk and the clock period T. We can derive the formula to compute the throughput from (1).

$$throughput(N) = \frac{b \cdot N}{clk \cdot T} = \frac{b \cdot N}{(st + (l + p) \cdot N + end) \cdot T} \tag{2}$$

For short messages we assume a message of 1 byte length which after padding is one block long for for all finalists and SHA-2 with the exception of JH. The minimum length of a padded JH message is 2 blocks.

When computing the throughput for very long messages, we can neglect st and end as their influence on the result goes to zero. This leads to the simplified equation (3).

$$throughput_{long} = \frac{b}{(l + p) \cdot T} \tag{3}$$

Resource Utilization of FPGAs is very difficult to define. All FPGAs contain configurable logic elements which contain flip-flops (Xilinx: slices, Altera: LE), BRAMs, multipliers and other resources. These resources have different features not only depending on the vendor but even on the FPGA family. Hence, we can compare implementations using the metric of throughput over area ratio only within a specific FPGA family and provided they use the same number of dedicated resources. As area in this formula we use solely *slices* for Xilinx and *LEs* for Altera devices as there is no direct mapping from BRAM utilization to slice or LE.

With respect to efficiency we consider two metrics: *power consumption* and *energy per bit*. The first is of interest for applications where only a limited amount of power is available and it is dependent on the clock frequency $1/T$ as shown (4) where P_{Static} and V_{Core} depend on the FPGA, not the implementation, and α (average switching activity) $\cdot C_l$ (load capacitance) depend on the implementation.

$$P_{Total} = P_{Static} + \underbrace{\alpha \cdot C_l \cdot V_{Core}^2 \cdot 1/T}_{P_{Dynamic}} \tag{4}$$

We can compute the energy required to hash a message of N blocks using (5) where $A = \alpha \cdot clk$ is the total switching activity.

$$E_N = \underbrace{P_{Static} \cdot T \cdot clk}_{E_{Static}} + \underbrace{A \cdot C_l \cdot V_{Core}^2}_{E_{Dynamic}} \tag{5}$$

For long messages we can compute *energy per bit* as

$$EnergyPerBit = \frac{E_N}{b \cdot N} = \frac{l+p}{b} \cdot P_{Static} \cdot T + \alpha \cdot C_l \cdot V_{Core}^2 \tag{6}$$

3 Implementations

We implemented two versions of each algorithm, one which utilizes only logic resources (Logic version) and one that additionally utilizes a single Block RAM (BRAM version). The throughput formulae for all implementations are shown in Table 1.

3.1 BLAKE

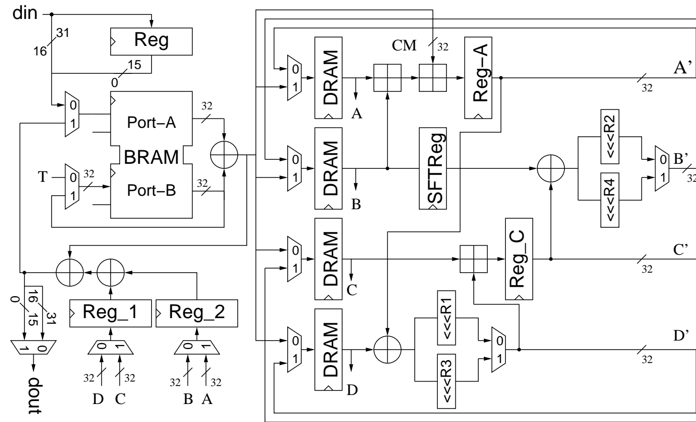


Figure 3: Block Diagram of BLAKE (BRAM)

Our implementations of BLAKE-256 are similar to the ones presented in [2, 8]. Our BRAM version (Fig. 3) stores the message, constants, initial and chaining hash values along with salt and a 64 bit counter value in BRAM. Loading the message through our 16-bit interface into BRAM takes 32 clock cycles. BLAKE-256 has 3 functions namely initialization, round and finalization. The initialization takes 16 clock cycles to produce the internal state (V-state), which we store in four Distributed RAMs. The round function consists of 8 almost identical G-Functions which operate on the internal V-state. We implemented 1/2 G-Function with quasi-pipelined stages and reorganized the G-Functions to avoid data conflicts (pipeline stalls). This results in a total of 16 clock cycles for one round. At the end of 14 rounds we require 2 extra clock cycles to empty the pipeline. In total it takes $14 \cdot 16 + 2 = 226$ clock cycles for the compression function. The G-Function requires permuted values of constants and message which are stored in BRAM. This permutation doesn't have a repeatable pattern, therefore the addressing of the message and constants for 14 rounds consumes 70% of the size of the controller. The finalization is again producing an intermediate hash output using the V-state and the chaining hash value. This requires 16 clock cycles. The total number of clock cycles to process on message block is therefore $32 + 16 + 226 + 16 = 290$ clock cycles.

The Logic version replaces the BRAM with two Distributed RAMs. One Distributed RAM stores message and the initial hash values, the other stores the constants, salt, and counter and chaining hash values. We introduced a register after each DRAM in order to match the BRAM version. This version therefore takes the same number of clock cycles as the BRAM version. The only change to the datapath shown in Fig. 3

is the change from one BRAM to two DRAMs. As the state was already stored in DRAM removing the BRAM did not yield additional optimization possibilities for BLAKE.

3.2 Grøstl

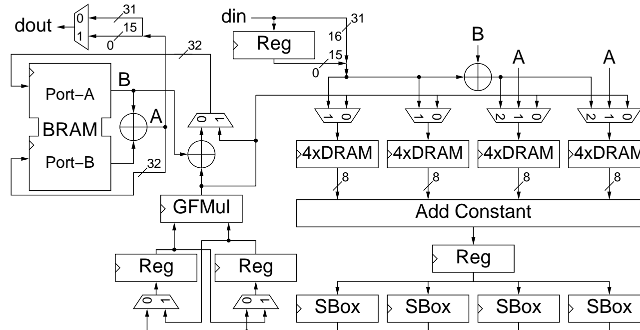


Figure 4: Block Diagram of Grøstl (BRAM)

Grøstl [19] is based on the AES round with the following sequence of operations: AddRoundConstant, SubBytes, ShiftBytes, and MixBytes. The implementation of our BRAM version (Fig. 4) uses one BRAM in dual port mode to store the initialization vector and the intermediate hash (h). It stores the state, consisting of two 512-bit matrices P & Q in 16 4x8 Distributed RAMs. Each row is stored in one Distributed RAM. In order to get the first 64-bit column we access byte0 from RAM0, byte1 from RAM1...etc. This access scheme performs the ShiftBytes operation with which we start each round. SubBytes is implemented using 4 pipelined S-Boxes which are described as logic functions [20]. The multiplier takes a column from SubBytes and produces 32 bits of the new column in one clock cycle, the remaining 32 bits in the second clock cycle. It takes a total 3 clock cycles to produce a new column. Each round of P and Q computes 16 new columns which takes 48 clock cycles. We interleave the computations of P and Q through the pipeline. The XOR operation ($P \oplus Q \oplus h$) takes 32 clock cycles. So a block of message is processed in 515 clock cycles ($48 \cdot 10 + 32 + 3$ clock cycles to fill the pipeline).

Our Logic version is primarily based on the BRAM version described above. Replacing the BRAM by Distributed RAM increases the area only marginally as the BRAM only stored the initialization vector and the intermediate hash. The new area budget allows us to implement a full Galois field multiplier which can produce a 64-bit column in one clock cycle. It now takes only a total of 2 clock cycles to produce a new column. Therefore, computing a round of P and Q takes now only 32 clock cycles. The total number of clock cycles to process one block of message are 357 clock cycles ($32 \cdot 10 + 32 + 5$ clock cycles to fill the pipeline).

3.3 JH

Our implementation of JH in the Logic version (Fig. 5) uses two single port 32x32 bit Distributed RAMs to store the initial and chaining hash values. Two pair of dual port Distributed RAMs are used to store the state and constants generated for each round. The initial round constants are stored in a ROM and the Distributed RAMs are loaded with the constants for each new message. The message is stored in a single port 32x16 bit Distributed RAM. The initial and chaining hash values are always stored in grouped format to avoid the grouping and ungrouping for each message block [7]. The grouped initial/chaining hash value is XORed with the message for each message block and stored in two dual port Distributed RAMs. This takes 64 clock cycles. The E8 function operates on two 32-bit words that are read out from the two Distributed RAMs. It takes 16 clock cycles to process each round. As we are using read and write addresses for dual port Distributed Ram, the addressing is repeated after 4 rounds for one Distributed RAM and after 8 rounds for the other Distributed RAM which minimizes the size of the controller. After 42 rounds the computed state is XORed with the message again and the final/chaining hash value is stored back into two single port

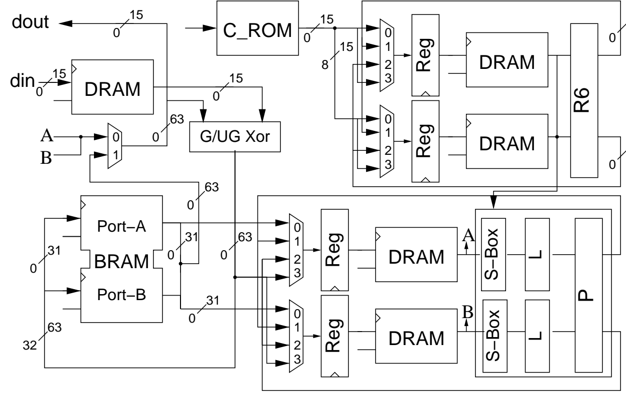


Figure 5: Block Diagram of JH (BRAM)

Distributed RAMs for the next message block. This XORing takes another 64 clock cycles. In total it takes 64+16*42+64 = 800 clock cycles for each message block.

In our BRAM version we replaced the two single port 32x32 bit Distributed RAMs by a dual port BRAM which is the only change to the datapath shown in Fig. 5. This dual port BRAM stores the initial and chaining hash values. As we are replacing the Distributed RAMs with BRAM it takes one extra clock cycle to start the BRAM and load the Distributed RAMs which store the internal state. As the state was already stored in DRAM removing the BRAM did not yield additional optimization possibilities for JH.

3.4 Keccak

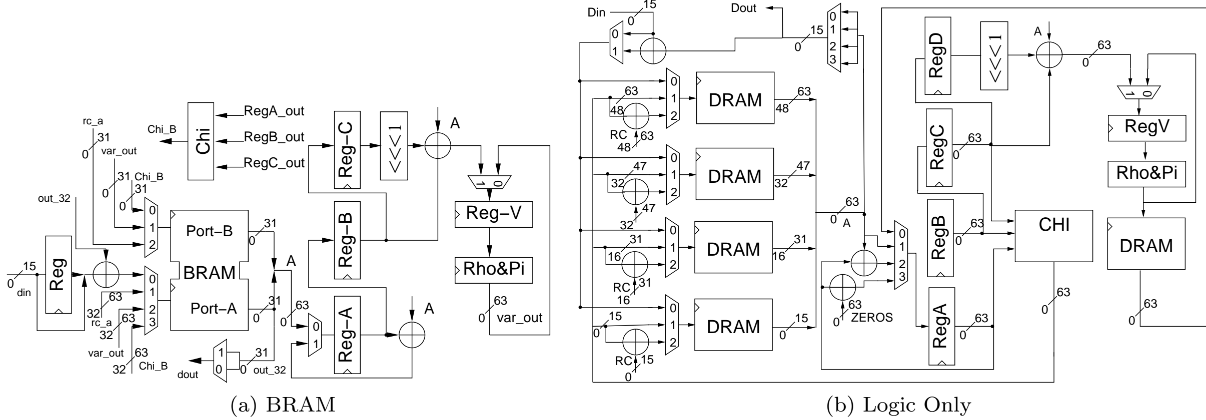


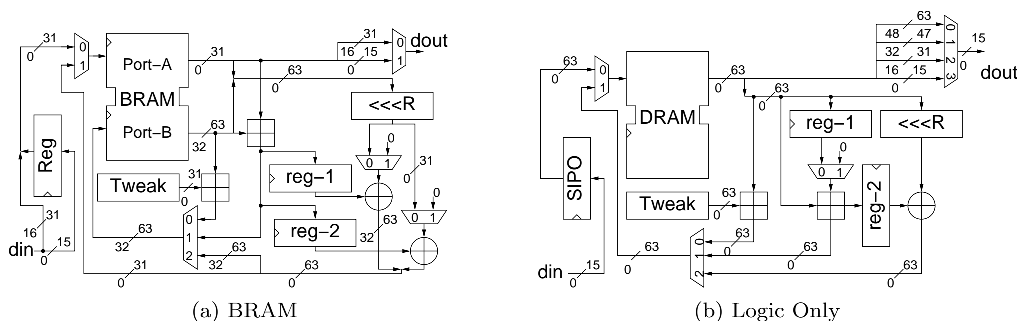
Figure 6: Block Diagram of Keccak

One round of Keccak [21] applies five functions, θ, ρ, π, χ, and ι to its state. In the BRAM version of our implementation of Keccak (Fig. 6), we store the state and the round constant in BRAM. The basic operations of Keccak use 64-bit data values which is also the maximum that we can read or write to BRAM in a single clock cycle. Therefore, in order to make the design more efficient we decided to quasi pipeline our functions. We have merged the θ and ρ functions. The later function uses a variable rotator. A barrel shifter consumes 192 slices on Spartan-3, hence we build a shifter that can only shift the 25 offsets Keccak needs. It uses on average 1.5 clock cycles per rotation and consumes only 128 slices. We use dedicated write cycles to accommodate the data rearrangement of the π function. These three functions take a total of 91 clock cycles. The χ function takes its operands from BRAM, applies a series of simple logical operations, and stores the result into BRAM. The ι operation combines a round constant with one 64-bit value of the new state. These

operations take an additional 63 clock cycles. A single round operation takes a total of $91 + 63 = 154$ clock cycles. Thus 3696 clock cycles are required to process one block of message. The reason for such a large number of clock cycles for this algorithm is that each state of a function has a dependency on the previous states and functions which limits how far the implementation can be quasi pipelined. Furthermore, the restriction on the amount of data that can be written and read from the BRAM during a single clock cycle makes it difficult to increase the throughput.

Our Logic version splits the BRAM into four single port Distributed RAMs of 16-bits each to store the initial state. The round constants can be easily stored in a ROM. The order of scheduling the operations and almost all the functions are implemented in the same manner as in the BRAM version with minor tweaks. We now have four registers (A through D) instead of three at the output of the DRAM‘s to compute the θ stage. The output for the ρ and π is now stored in an additional 64-bit single port Distributed RAM. This de-couples reading from the writing which eliminates address contention. This allows us to remove the dedicated write cycles we used in the BRAM version. The ρ , π and θ stages are now computed in 58 clock cycles, while the χ and   stages take another 39 clock cycles. Overall one round is processed in 97 clock cycles while one block is computed in 2328 clock cycles.

3.5 Skein



(a) BRAM

(b) Logic Only

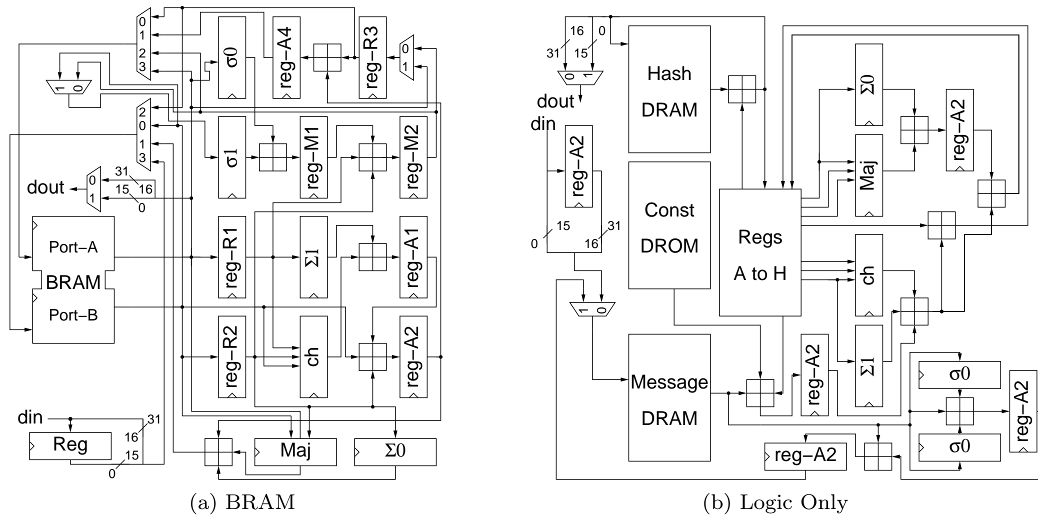
Figure 7: Block Diagram of Skein

Unfortunately, we could not use a 32-bit datapath for our Logic implementation version (7b). Replacing the dual-port BRAM with a Distributed RAM requires 2 dual-port 48x32 Distributed RAMs which will consume a total of 356 slices. A 64-bit datapath on the other hand allows us to use a single-port 48x64 Distributed RAM which consumes only 162 slices on a Spartan-3 FPGA. However, this reduction in area comes at the price of a slight increase in the critical path due to the 64-bit adder. We analyzed the throughput over area ratio of both, 32-bit and 64-bit datapath options, and came to the conclusion that for our Logic version a 64-bit datapath yields better results. The mix operation consists of a 64-bit addition and an XOR operation along with a rotation. Even on the 64-bit datapath, using the mix operation 4-times per round takes 20 clock cycles. Key injections take 47 clock cycles (48 for the first) however the permutations are faster at only 32 clock cycles instead of 109 for the 32-bit datapath. This results in a total of 2366 clock cycles per message block plus 32 clock cycles for loading the block.

3.6 SHA-2

The SHA-256 uses six logical functions Ch, Maj, $\Sigma0$, $Sigma1$, $sigma0$, and $sigma1$. Each of these functions operates on 32-bit words resulting in a new 32-bit words. These six functions are used in one of the three processing steps. The first processing step is message expansion where a 512-bit message block is expanded into 2048-bit message using two functions $\sigma0$ and $\sigma1$. The second processing step is round operation which uses eight working variables $a, b, .., h$. These eight working variables are initialized with initial hash values and updated using 2048-bit message, sixty four 32-bit round constants and four functions $\Sigma0$, $\Sigma1$, Ch and

Maj. The final step is intermediate hash generation where the eight working variables are added with initial hash values to obtain new intermediate hash values.



(a) BRAM

(b) Logic Only

Figure 8: Block Diagram of SHA-2

Our implementation of SHA-256 with BRAM uses it in Dual-port mode to store message, working variables, round constants, initial and final hash values (8a). The datapath is quasi-pipelined to reduce the critical time and clock cycles. Most of the pipeline registers except R1, R2 and R3 does not cost any additional area. The initialization of the working variables and intermediate hash values is performed while loading of message. Due to BRAM contention, message expansion takes 99 clock cycles while the round operation takes 448 clock cycles.

In the logic only version, the BRAM is replaced with three DRAMs and six registers (8b). The message, round constants and hash values are stored in DRAMs, the working variables in registers. Using registers for working variables reduces the required clock cycles for round operation to 192 clock cycles. The number of clock cycles for message expansion increases to 196 clock cyles due to use of single port DRAM. Using a dual-port DRAM can reduce the clock cyles but would increase area significantly. Using approximately additional 100 slices, the throughput can be doubled but it would violate the area constraint.

4 Results and Conclusions

4.1 Implementation Results

The results of our implementations are summarized by the graph shown in Fig. 9. It shows the area consumption of each implementation on the x-axis and the throughput on the y-axis. It can be seen that all implementations with BRAM (blue squares) fall within our narrow target range of 450 to 650 slices. Within this area constraint each algorithm was optimized for maximum throughput. All Logic only implementations (red diamonds) are below the maximum target of 768 slices, however Grøstl, Keccak and Skein use almost 100% of all logic resources. JH42 is still more than 200 slices smaller and BLAKE-256 more than 150 slices smaller than the maximum. Unfortunately, neither algorithm can take advantage of the additional area to increase their throughput. In fact, it would reduce their throughput over area ratio. Only Grøstl, SHA-2, and Keccack show an improved throughput when implemented without BRAM but with more area for logic. For the others, the throughput remained almost the same and for JH42 the throughput actually decreased.

The performance results of each implementation on a Xilinx Spartan-3 device is shown in Fig. 10. In each graph the order of the algorithm is sorted by best to worst performer. The top three graphs show results

Table 1: Throughput formulae for our implementations of SHA-3 candidates

Version	Algorithm	Specification	Block Size (bits) b	# Rounds	Clock Cycles per Round	Additional Clock Cycles	Clock Cycles to hash N blocks $clk = st + (\ l +\ p) \cdot N +\ end$	Throughput (long Messages) $\frac{b}{(l+p) \cdot T}$
BRAM	BLAKE-256	[22]	512	14	16	34	$2 + (32 + 258) \cdot N + 17$	$512/(290 \cdot T)$
	Grøstl	[19]	512	10	48	35	$2 + (32 + 515) \cdot N + 532$	$512/(547 \cdot T)$
	JH42	[23]	512	42	16	65	$2 + (64 + 737) \cdot N + 33$	$512/(801 \cdot T)$
	Keccak	[21]	1088	24	154	0	$2 + (68 + 3696) \cdot N + 17$	$1088/(3764 \cdot T)$
	Skein	[24]	512	72	20	967	$5 + (32 + 2407) \cdot N + 2362$	$512/(2439 \cdot T)$
	SHA-2	[24]	512	64	7	115	$2 + (32 + 563) \cdot N + 17$	$512/(595 \cdot T)$
Logic only	BLAKE-256	[22]	512	14	16	34	$2 + (32 + 258) \cdot N + 17$	$512/(290 \cdot T)$
	Grøstl	[19]	512	10	32	37	$2 + (32 + 357) \cdot N + 374$	$512/(389 \cdot T)$
	JH42	[23]	512	42	16	64	$2 + (64 + 736) \cdot N + 32$	$512/(800 \cdot T)$
	Keccak	[21]	1088	24	97	0	$2 + (68 + 2328) \cdot N + 17$	$1088/(2396 \cdot T)$
	Skein	[24]	512	72	20	926	$5 + (32 + 2366) \cdot N + 2319$	$512/(2398 \cdot T)$
	SHA-2	[24]	512	64	3	212	$2 + (32 + 404) \cdot N + 17$	$512/(436 \cdot T)$

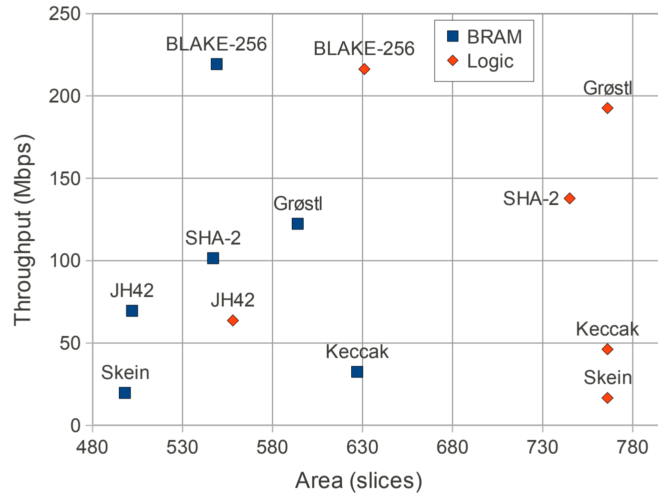
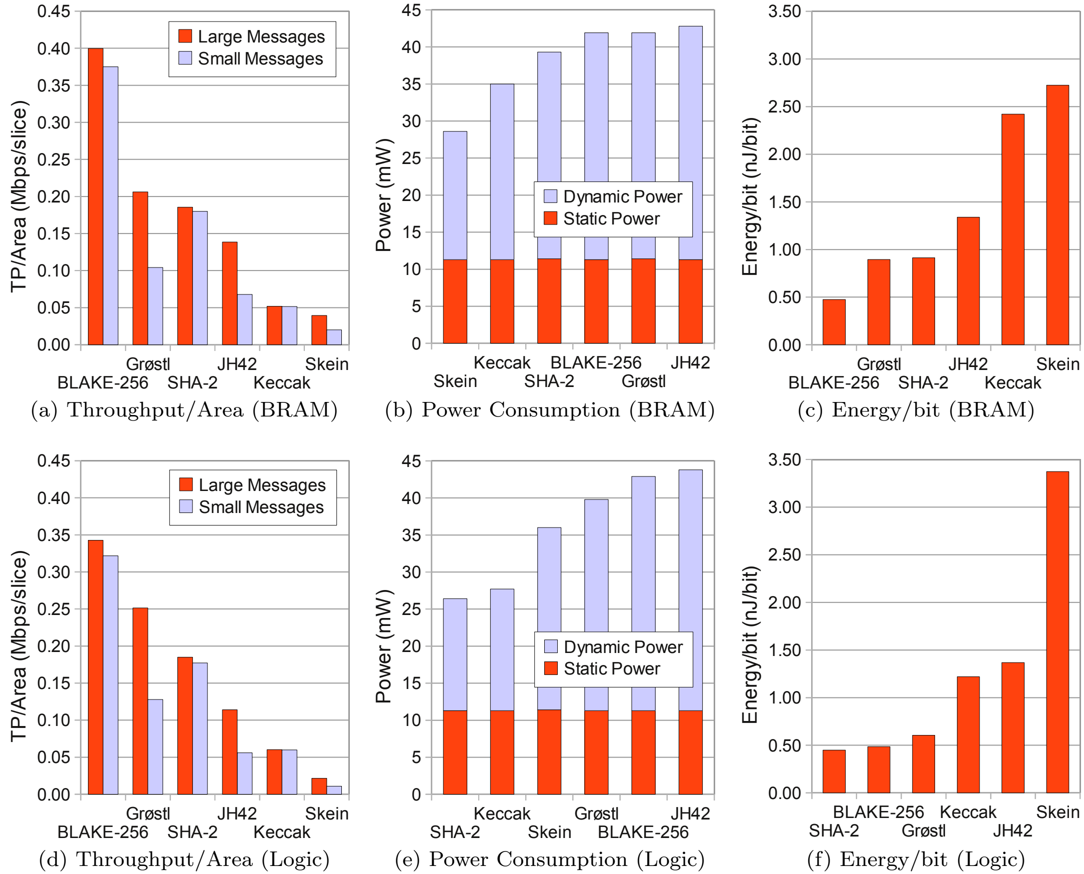


Figure 9: Throughput over area of our SHA-3 implementations on Xilinx Spartan-3

for the implementations with BRAM, the bottom three for Logic only implementations. The first graphs (Fig. 10a, Fig. 10d) show the throughput over area ratio of each implementation for long messages (red) and for short messages (light blue). It can clearly be seen, that algorithms that have a lengthy finalization step (Grøstl, Skein) or require an additional block just for padding (JH42) do not perform as well for short messages as for long messages. Only BLAKE-256 and Grøstl (for long messages) perform better than SHA-2. The order of the algorithms does not change between BRAM and Logic only implementations.

The next pair of graphs (Fig. 10b, Fig. 10e) shows the power consumption of each implementation which we measured on a Xilinx-3E FGPA. As expected, the static power consumption (red) is the same for all implementations as it depends only on the device, and not on how the device is configured. What is unique to each implementation is the dynamic power consumption (light blue). Skein and Keccak have a lower power consumption than SHA-2, but only for the BRAM implementations. The most power efficient implementations are SHA-2 (BRAM) and Keccak (BRAM). This picture changes dramatically when we look at the energy efficiency measured in Energy/bit (Fig. 10c, Fig. 10f). Again BLAKE-256, SHA-2 and Grøstl are leading. Skein has the worst energy efficiency.



(a) Throughput/Area (BRAM) (b) Power Consumption (BRAM) (c) Energy/bit (BRAM)

(d) Throughput/Area (Logic) (e) Power Consumption (Logic) (f) Energy/bit (Logic)

Figure 10: Performance Results for Lightweight Implementations of SHA-3 Candidates on Spartan-3

4.2 Implementation Shootout

Due to the multitude of results we generated it is difficult to determine a clear winner. The following tables 2 and 3 summarize the most important results. In each category they show how much different an algorithm performs as compared to SHA-2. Results that are better than SHA-2 are highlighted. Table 2 compares throughput over area, power and energy results on our target device Spartan-3. Table 3 compares only throughput over area results but accross all FPGA families for which we generated results. The only algorithm that is consistently better than SHA-2 is BLAKE-256 followed by Grøstl. The detailed results of our implementations on Xilinx Spartan-3 and on Spartan-6, Virtex-5, Virtex-6, and Altera Cyclone-II devices are summarized in the Appendix in Tables 6 and 5.

4.3 Comparison with Other Reported Results

We compare our results with previously reported ones by Kerckhof et al. [7] and Jungk et al. [8].Even though our primary design target is Xilinx Spartan-3, we synthesized our implementations for other devices to match the devices of reported results. This puts our designs at a disadvantage as we could not take full advantage of their features. Most notably, pipeline stages might become unbalanced when synthesizing a design for a device with 4-input LUTs on a 6-input LUT device. Figure 11 compares our results with results from Kerckhof et al. [7]. Our implementations with BRAM are indicated through blue squares, Logic only through red diamonds and Kerckhof's through yellow triangles. Even though the area range of our implementations was

Table 2: Shootout of Spartan-3 Implementations relative to SHA-2

TP/A: Throughput/Area

Algorithm	BRAM TP/A Large	BRAM TP/A Small	BRAM Power	BRAM Energy /bit	Logic Only TP/A Large	Logic Only TP/A Small	Logic Only Power	Logic Only Energy /bit
BLAKE-256	2.15	2.08	1.07	0.52	1.85	1.81	2.09	1.08
Grøstl	1.11	0.58	1.07	0.98	1.36	0.72	1.89	1.35
JH42	0.75	0.38	1.09	1.47	0.62	0.32	2.15	3.04
Keccak	0.28	0.29	0.89	2.65	0.33	0.34	1.09	2.71
Skein	0.21	0.11	0.73	2.98	0.12	0.06	1.63	7.50
SHA-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 3: Implementation Shootout, Throughput over Area relative to SHA-2

L: Large Message, S: Small Message

		Xilinx								Altera	
		Spartan-3		Spartan-6		Virtex-5		Virtex-6		Cyclone-II	
	Algorithm	L	S	L	S	L	S	L	S	L	S
BRAM	BLAKE-256	2.15	2.08	1.99	1.93	1.80	1.74	2.06	2.00	2.06	1.99
BRAM	Grøstl	1.11	0.58	0.69	0.36	1.02	0.54	0.92	0.48	1.95	1.02
BRAM	JH42	0.75	0.38	0.54	0.27	1.01	0.51	0.76	0.39	1.06	0.54
BRAM	Keccak	0.28	0.29	0.43	0.44	0.49	0.50	0.56	0.58	0.84	0.86
BRAM	Skein	0.21	0.11	0.15	0.08	0.18	0.10	0.16	0.09	0.36	0.19
BRAM	SHA-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Logic only	BLAKE-256	1.85	1.81	2.24	2.19	1.86	1.82	2.24	2.19	3.76	3.69
Logic only	Grøstl	1.36	0.72	1.43	0.76	1.50	0.80	1.41	0.75	1.93	1.02
Logic only	JH42	0.62	0.32	0.74	0.38	0.99	0.50	0.74	0.38	0.37	0.19
Logic only	Keccak	0.33	0.34	0.90	0.93	0.48	0.50	1.00	1.04	0.27	0.28
Logic only	Skein	0.12	0.06	0.14	0.08	0.17	0.09	0.17	0.09	0.07	0.04
Logic only	SHA-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

560 slices ±65, i.e. with a variability of less than ±12% on Spartan-3 for BRAM implementations, and 660 ±16% for logic only implementations our results show a large variability on Spartan-6. Kerckhof's designs occupy a similar range of sizes. The graph in Fig. 12 is more interesting in that the results from Jungk et al. occupy a different area than ours and with the exception of Skein, all their designs are in a tight area, just like ours. The data in Fig. 12 is newer than [8] and was obtained from the ATHENa database [11]. Figure 12 illustrates nicely how non linearly the throughput increases with an increase in available area.

4.4 Conclusions

In this paper we presented two sets of lightweight FPGA implementations of all SHA-3 finalists and SHA-2. All algorithms were implemented using the same assumptions, goals, tools, interface, and the same area optimization techniques and their power consumption were measured. The lightweight implementations were evaluated with regards to their throughput over area ratio and energy per bit. The resulting ranking of algorithms is very different from implementations for best throughput over area reported in the literature [25], [14], [26]. The finalists with the best throughput over area ratio and with the least energy per bit on Xilinx devices are BLAKE-256 followed by Grøstl.

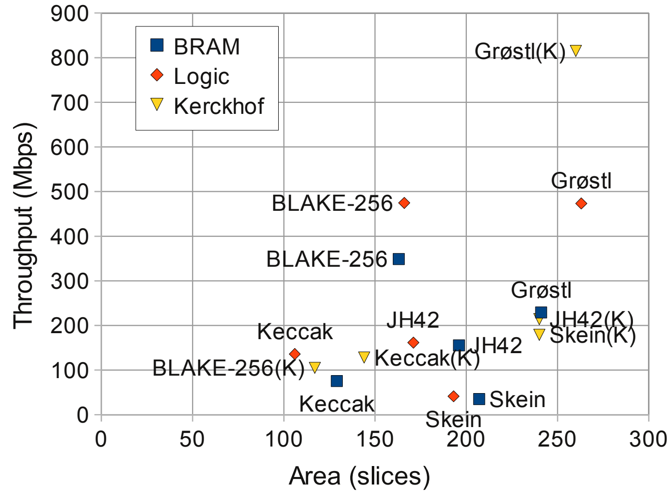


Figure 11: Throughput over area comparison with Kerckhof et al. on Virtex-6

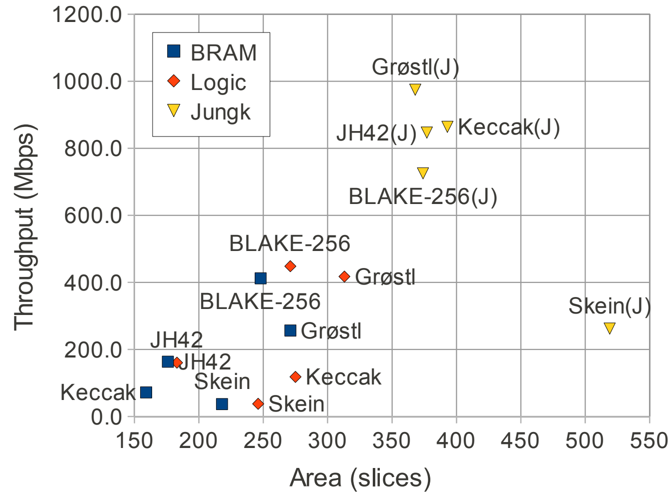


Figure 12: Throughput over area comparison with Jungk et al. on Virtex-5

References

[1] "Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family," Federal Register/ Vol. 72, No. 212, Nov 2007, notices 62212.

[2] J.-L. Beuchat, E. Okamoto, and T. Yamazaki, "Compact implementations of BLAKE-32 and BLAKE-64 on FPGA," Cryptology ePrint Archive, Report 2010/173, 2010.

[3] B. Jungk and S. Reith, "On FPGA-based implementations of Grøstl," Cryptology ePrint Archive, Report 2010/260, 2010.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak sponge function family main document," http://keccak.noekeon.org, Apr 2009, version 1.2.

[5] A. Namin and M. Hasan, "Implementation of the compression function for selected SHA-3 candidates on FPGA," in *International Parallel Distributed Processing Symposium, Workshops and Phd Forum (IPDPSW)*. IEEE, 2010, pp. 1–4.

[6] ——, "Hardware implementation of the compression function for selected SHA-3 candidates," Centre for Applied Cryptographic Research (CACR), University of Waterloo, Tech. Rep. 28, Jul 2009.

[7] S. Kerckhof, F. Durvaux, N. Veyrat-Charvillon, F. Regazzoni, G. M. de Dormale, and F.-X. Standaert, “Compact FPGA implementations of the five SHA-3 finalists,” in *10th Smart Card Research and Advanced Application Conference, CARDIS 2011*, Leuven, Belgium, Sep 2011.

[8] B. Jungk and J. Apfelbeck, “Area-efficient FPGA implementations of the SHA-3 finalists,” in *International Conference on ReConfigurable Computing and FPGAs*. IEEE: ReConfig’11, DEC 2011.

[9] J.-P. Kaps, P. Yalla, K. K. Surapathi, B. Habib, S. Vadlamudi, S. Gurung, and J. Pham, “Lightweight implementations of SHA-3 candidates on FPGAs,” in *Progress in Cryptology – INDOCRYPT 2011*, ser. Lecture Notes in Computer Science (LNCS), D. J. Bernstein and S. Chatterjee, Eds., vol. 7107. Springer Berlin / Heidelberg, Dec 2011, pp. 270–289.

[10] B. Westermann, D. Gligoroski, and S. Knapskog, “Comparision of the power consumption of the 2nd round sha-3 candidates,” 2010.

[11] “ATHENa results database,” http://cryptography.gmu.edu/athenadb/, Automated Tool for Hardware EvaluatioN project.

[12] K. Gaj, J.-P. Kaps, V. Amirineni, M. Rogawski, E. Homsirikamol, and B. 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*. IEEE, 2010, pp. 414–421, winner of the FPL Community Award.

[13] *Hardware Interface of a Secure Hash Algorithm (SHA)*, v. 1.4 ed., Cryptographic Engineering Research Group, George Mason University, Jan 2010.

[14] K. Gaj, E. Homsirikamol, and M. Rogawski, “Fair and comprehensive methodology for comparing hardware performance of fourteen round two SHA-3 candidates using FPGA,” in *Cryptographic Hardware and Embedded Systems, CHES 2010*, ser. LNCS, S. Mangard and F.-X. Standaert, Eds., vol. 6225. Springer Berlin / Heidelberg, 2010, pp. 264–278.

[15] E. Homsirikamol, M. Rogawski, and K. Gaj, “Throughput vs. area trade-offs architectures of five round 3 SHA-3 candidates implemented using Xilinx and Altera FPGAs,” in *Workshop on Cryptographic Hardware and Embedded Systems CHES 2011*, ser. LNCS, B. Preneel and T. Takagi, Eds. Springer Berlin / Heidelberg, Sep 2011.

[16] M. Rawski, H. Selvaraj, and T. Luba, “An application of functional decomposition in ROM-based FSM implementation in FPGA devices,” *J. Syst. Archit.*, vol. 51, no. 6-7, pp. 424–434, 2005.

[17] V. Skylarov, “Synthesis and implementation of RAM-based finite state machines in FPGAs,” in *Field-Programmable Logic and Applications – FPL’00*, ser. LNCS, R. W. Hartenstein and H. Grünbacher, Eds., vol. 1896. Springer-Verlag, 2000, pp. 718–728.

[18] I. García-Vargas, R. Senhadji-Navarro, G. Jiménez-Moreno, A. Civit-Balcells, and P. Guerra-Gutiérrez, “ROM-based finite state machine implementation in low cost FPGAs,” in *International Symposium on Industrial Electronics, ISIE 2007*. IEEE, June 2007, pp. 2342–2347.

[19] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schäffer, and S. S. Thomsen, “Grøstl – a SHA-3 candidate,” Submission to NIST (Round 3), 2011, http://www.groestl.info/Groestl.pdf.

[20] K. Gaj and P. Chodowiec, *Cryptographic Engineering*. Springer, 2009, ch. FPGA and ASIC Implementations of AES, pp. 235–294.

[21] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “The Keccak SHA-3 submission,” Submission to NIST (Round 3), 2011, http://keccak.noekeon.org/Keccak-submission-3.pdf.

[22] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan, “SHA-3 proposal BLAKE,” Submission to NIST (Round 3), 2010, http://131002.net/blake/blake.pdf.

[23] H. Wu, “The hash function JH,” Submission to NIST (round 3), 2011, http://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf.

[24] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker, “The Skein hash function family,” Submission to NIST (Round 3), 2010, http://www.skein-hash.info/sites/default/files/skein1.3.pdf.

[25] S. Matsuo, M. Knežević, P. Schaumont, I. Verbauwhede, A. Satoh, K. Sakiyama, and K. Ota, "How can we conduct "fair and consistent" hardware evaluation for SHA-3 candidate?" Second SHA-3 Candidate Conference, Tech. Rep., 2010.

[26] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. O'Neill, and W. P. Marnane, "FPGA implementations of the round two SHA-3 candidates," Second SHA-3 Candidate Conference, Tech. Rep., 2010.

Appendix

Table 4: Power Measurements on Spartan-3E

	Algorithm	P_{Static} [mW]	$P_{Dynamic}$ [mW]	P_{Total} [mW]	Energy/bit nJ/bit
BRAM	BLAKE-256	11.3	30.6	41.9	0.475
	Grøstl	11.4	30.5	41.9	0.895
	JH42	11.3	31.5	42.8	1.339
	Keccak	11.3	23.7	35.0	2.422
	Skein	11.3	17.3	28.6	2.725
	SHA-2	11.4	27.9	39.3	0.913
Logic only	BLAKE-256	11.3	31.6	42.9	0.486
	Grøstl	11.3	28.5	39.8	0.605
	JH42	11.3	32.5	43.8	1.369
	Keccak	11.3	16.4	27.7	1.220
	Skein	11.4	24.6	36.0	3.372
	SHA-2	11.3	15.1	26.4	0.450

Table 5: Implementation results of our implementations of SHA-3 candidates

Device	Version	Message / Algorithm	Area (LEs)	Memory Bits	Maximum Delay (ns) T	Long: Throughput (Mbps)	Long: TP/Area (Mbps/slice)	Short: Throughput (Mbps)	Short: TP/Area (Mbps/slice)
Altera Cyclone II (ep2c8f256c6)	BRAM	BLAKE-256	1,367	2,048	9.98	176.9	0.13	166.0	0.121
		Grøstl	1,221	3,072	6.26	149.6	0.12	75.7	0.062
		JH42	1,045	3,840	9.15	69.9	0.07	34.2	0.033
		Keccak	996	8,192	5.48	52.7	0.05	52.5	0.053
		Skein	930	4,096	9.89	21.2	0.02	10.8	0.012
		SHA-2	1,195	8,192	11.45	75.1	0.06	72.8	0.061
	Logic only	BLAKE-256	2,019	0	7.39	238.8	0.12	224.1	0.111
		Grøstl	3,937	0	5.52	238.4	0.06	121.2	0.031
		JH42	5,527	0	10.05	63.7	0.01	31.2	0.006
		Keccak	6,247	0	8.49	53.5	0.01	53.1	0.008
		Skein	6,141	0	15.83	13.5	0.001	6.8	0.001
		SHA-2	4,705	0	7.94	147.9	0.03	141.7	0.030

Table 6: Implementation results of our implementations of SHA-3 candidates

Device	Version	Algorithm	Area (slices)	Block RAMs	Maximum Delay (ns) T	Long: Throughput (Mbps)	Long: TP/Area (Mbps/slice)	Short: Throughput (Mbps)	Short: TP/Area (Mbps/slice)
Xilinx Spartan-3 (xc3s50-5)	BRAM	BLAKE-256	549	1	8.05	219.3	0.40	205.9	0.375
		Grøstl	594	1	7.65	122.4	0.21	61.9	0.104
		JH42	502	1	9.19	69.6	0.14	34.0	0.068
		Keccak	627	1	8.90	32.5	0.05	32.3	0.052
		Skein	498	1	10.65	19.7	0.04	10.0	0.020
		SHA-2	547	1	8.48	101.5	0.19	98.4	0.180
	Logic only	BLAKE-256	631	0	8.16	216.3	0.34	203.0	0.322
		Grøstl	766	0	6.83	192.6	0.25	97.9	0.128
		JH42	558	0	10.05	63.7	0.11	31.2	0.056
		Keccak	766	0	9.83	46.2	0.06	45.8	0.060
		Skein	766	0	12.83	16.6	0.02	8.5	0.011
		SHA-2	745	0	8.52	137.8	0.19	132.1	0.177
Xilinx Spartan-6 (xc6slx4csg-3)	BRAM	BLAKE-256	152	1	5.63	313.8	2.06	294.5	1.938
		Grøstl	271	1	4.80	195.0	0.72	98.7	0.364
		JH42	182	1	6.23	102.6	0.56	50.2	0.276
		Keccak	127	1	5.07	57.0	0.45	56.8	0.447
		Skein	182	1	7.19	29.2	0.16	14.8	0.081
		SHA-2	140	1	5.93	145.2	1.04	140.7	1.005
	Logic only	BLAKE-256	164	0	5.34	330.6	2.02	310.2	1.882
		Grøstl	230	0	4.43	297.3	1.29	151.2	0.657
		JH42	156	0	6.14	104.2	0.67	51.0	0.327
		Keccak	113	0	4.95	91.8	0.81	91.1	0.806
		Skein	190	0	8.77	24.3	0.13	12.4	0.065
		SHA-2	227	0	5.74	204.6	0.90	196.0	0.864
Xilinx Virtex-5 (xc5vlx20-2)	BRAM	BLAKE-256	248	1	4.29	411.9	1.66	386.6	1.559
		Grøstl	271	1	3.65	256.5	0.95	129.8	0.479
		JH42	176	1	3.91	163.5	0.93	80.0	0.454
		Keccak	159	1	4.04	71.6	0.45	71.3	0.448
		Skein	218	1	5.69	36.9	0.17	18.7	0.086
		SHA-2	234	1	3.98	216.2	0.92	209.5	0.895
	Logic only	BLAKE-256	271	0	3.94	448.2	1.65	420.7	1.552
		Grøstl	313	0	3.15	417.4	1.33	212.3	0.678
		JH42	183	0	3.99	160.3	0.88	78.5	0.429
		Keccak	275	0	3.85	118.1	0.43	117.2	0.426
		Skein	246	0	5.66	37.7	0.15	19.2	0.078
		SHA-2	312	0	4.24	277.0	0.89	265.4	0.851
Xilinx Virtex-6 (xc6vlx75T-1)	BRAM	BLAKE-256	163	1	5.06	348.7	2.14	327.3	2.008
		Grøstl	241	1	4.09	229.1	0.95	115.9	0.481
		JH42	196	1	4.11	155.4	0.79	148.9	0.760
		Keccak	129	1	3.84	75.2	0.58	74.9	0.580
		Skein	207	1	6.00	35.0	0.17	17.8	0.086
		SHA-2	155	1	4.84	177.8	1.15	172.3	1.111
	Logic only	BLAKE-256	166	0	3.72	474.6	2.86	445.4	2.693
		Grøstl	263	0	2.78	473.3	1.80	240.7	0.915
		JH42	171	0	3.96	161.5	0.94	154.9	0.906
		Keccak	106	0	3.34	136.0	1.28	135.0	1.273
		Skein	193	0	5.17	41.3	0.21	21.0	0.109
		SHA-2	238	0	3.86	304.2	1.28	291.5	1.225