Pushing the Limits of SHA-3 Hardware Implementations to Fit on RFID

Peter Pessl and Michael Hutter

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
peter.pessl@gmail.com, Michael.Hutter@iaik.tugraz.at

Abstract. There exists a broad range of RFID protocols in literature that propose hash functions as cryptographic primitives. Since Keccak has been selected as the winner of the NIST SHA-3 competition in 2012, there is the question of how far we can push the limits of Keccak to fulfill the stringent requirements of passive low-cost RFID. In this paper, we address this question by presenting a hardware implementation of Keccak that aims for lowest power and lowest area. Our smallest (full-state) design requires only 2 927 GEs (for designs with external memory available) and 5 522 GEs (total size including memory). It has a power consumption of 12.5 μW at 1 MHz on a low leakage 130 nm CMOS process technology. As a result, we provide a design that needs 40 % less resources than related work. Our design is even smaller than the smallest SHA-1 and SHA-2 implementations.

Keywords: Hardware Implementation, SHA-3, Keccak, ASIC, RFID, Low-Power Design, Embedded Systems.

1 Introduction

Radio Frequency Identification (RFID) is a technology that makes great demands on cryptographers to implement secure applications. The main challenges are the limited power consumption of tags that are in the field as well as the limited chip area that is available. In the past, several RFID-protocol designers proposed to use hash functions to provide cryptographic services. Hash functions are basic building blocks to implement, e.g., digital signatures or privacy-preserving protocols. However, it has been shown that these building blocks can not be implemented as efficient as other cryptographic primitives like AES or PRESENT as highlighted by M. Feldhofer, C. Rechberger [13] and A. Bogdanov et al. [12]. Until now it remains an open question if Keccak is a suitable candidate for those devices and if it can fulfill these demands.

Before Keccak has been selected as the winner of the NIST SHA-3 competition in October 2012, several authors reported performance results for ASIC platforms. Most of them target high-speed implementations which require between 27 and 56 kGEs (synthesized on 90 or 130 nm CMOS process technology).

G. Bertoni and J.-S. Coron (Eds.): CHES 2013, LNCS 8086, pp. 126–141, 2013.

The smallest design has been estimated by the KECCAK design team itself needing 9.3 kGEs and 5 160 clock cycles per message block. Kavun et al. [28] have been the first who evaluated KECCAK for RFID devices. They analyzed various KECCAK variants using different state sizes. Their 1 600-bit state version requires 20.8 kGEs, 1 200 clock cycles per block, and 44 μW per MHz which makes their design not well suitable for most passive low-cost tags.

Our Contribution. In this paper, we present a compact hardware implementation that aims to identify the lowest possible bound for KECCAK in terms of power and area. Goal is to meet the basic requirements of passive low-cost RFIDs. We focus on the most likely configuration of KECCAK that will be standardized by NIST in the near future, i.e., a 1 600 (or 800) bit state, 224/256/384/512-bit output lengths, and 24 (or 22) rounds. We present two different designs which are based on highly serialized 8 and 16-bit datapaths, respectively. Our smallest full-state design requires 2 927 GEs (core only) and 5 522 GEs including memory and hashes a block within 22 kCycles (thus following the RFID design principle *few gates and many cycles* as suggested by S. Weis [42]). Our second design is slightly larger (3 148 GEs and 5 898 GEs, respectively) but needs only 15 kCycles in total. Next to these results, we also analyzed KECCAK using a state size of 800 bits only (and using 22 rounds). In this case, our designs require 4 627 GEs and 4 945 GEs in total (including memory) while the cycle count decreases to 11 and 7 kCycles, respectively. All our designs consume less than 15 μW per MHz and thus meet the basic requirements of passive low-cost tags. Compared to the smallest reported 1 600-bit KECCAK implementation, our designs require about 40 % less resources. The numbers are also comparative with the smallest reported SHA-1 [33] (being slightly smaller in size but needing 50 % less power) and SHA-2 [31] implementations (40 % less area).

Roadmap. The paper is organized as follows. In Section 2, we give a brief introduction to state of the art RFID crypto and its requirements. In Section 3, the KECCAK algorithm is presented and low-resource optimizations are discussed. Section 4 presents the implemented hardware architectures. Section 5 provides results and a discussion about further optimizations. Conclusions are drawn in Section 6.

2 Crypto on RFID

Radio Frequency Identification (RFID) is a contactless communication technology that consists of three parts: tags, readers, and a back-end system. Tags are essentially composed of tiny microchips which are attached to an antenna. They can communicate with a reader via an electromagnetic field which is also used to power the tags in case of passive tags. Active tags, in contrast, have their own power source, e.g., a battery. Readers are connected to a back-end system that is typically composed of a database holding tag records.

Nowadays, RFID systems are widely used in many applications that help to improve, for example, logistics, inventory control, transportation, access control,

or contactless payment. In this context, RFID faces several security and privacy challenges. Most of these applications carry enough sensitive information to require strong cryptographic services. Secure RFID is essential also for new applications that require integrity of tag data, confidentiality during communication, and authentication or proof-of-origin to prevent counterfeiting—a major challenge where RFID might help to stop the process of piracy.

In the following, we list the principle design criteria and requirements for security-enabled RFID devices.

Reading Range and Power. The primary concern in passive RFID systems is the limited power that is available for the tags. Tags draw their energy from the electromagnetic field of a reader and use internal capacitors to buffer the energy to perform computations. The available energy depends thereby on various factors such as the distance to the reader, the size of the antenna, the operating frequency, and the field-strength of the reader. Inductively-coupled tags operating in the 13.56 MHz frequency range typically have enough power available. The magnetic field of the readers is quite high (1.5 to 7.5 A/m). This means that there are several milliwatts of power available for the tags to perform cryptographic operations. Long-range tags (e.g., UHF EPC Gen2 tags), in contrast, have a reading range of several meters. These tags have only a fraction of power available, i.e., a few microwatts that are drawn from the electromagnetic (far-)field of the reader. Thus, these tags have to operate in an environment where the power source is being up to 1 000 times lower compared to short-range HF systems. In practice, the total power consumption of those devices is typically limited to at most 10-15 μW per MHz on average and 3-30 μW peak power (depending on read or write operations) [34, 35].

Costs and Chip Area. During the last decade, several authors made chip area estimations for low-cost passive RFID tags. One of the first estimations have been made by S. Sarma from the MIT Auto-ID Center [36–38] and S. Weis in 2003 [42]. They predicted the costs for a low-cost tag to be 5 (dollar) cents in the near future and estimated the actual die size of a low-cost tag accordingly to be between 5 000 and 15 000 gate equivalents where only up to 2 000 gates are usable for security purposes. Similar estimations have been made by D. Ranasinghe and P. Cole in 2008—both from the Auto-ID Lab Adelaide—who reported numbers from 2 000 to 5 000 GEs for security-related functions [34]. They stated that the number of available gates naturally increases over the years due to improvements in manufacturing and process technology as also highlighted by M. Feldhofer and J. Wolkerstorfer in [15].

Speed and Response Time. Tags have to answer the reader within a specific response time. This time is usually very short, i.e., 15-250 μs for EPC Gen2 tags (nominal range), 320 μs for ISO/IEC 15693 tags, and 86-91 μs for ISO/IEC 14443 tags[1]. However, it is principally not required for a tag to finish the computation

[1] This number refers to the response time of ISO/IEC 14443-3 tags during anti-collision. For higher-level protocols like ISO/IEC 14443-4, the default response time is 4.8 ms and it can be extended up to 5 seconds if needed.

within this short period of time (if even possible). Instead, a challenge-response protocol is needed that allows a larger time frame for cryptographic operations (without causing a recognizable delay). Thus, challenging a tag that is for example clocked with 1.5 MHz, would take the reasonable time period of 4.8 ms to perform a computation needing 7 200 clock cycles.

2.1 Hash Functions for RFID

One of the first who proposed to use hash functions in RFID protocols in 2003 were S. Weis, S. Sarma, R. Rivest, and D. W. Engels [25, 42]. They made use of the difficulty to invert a one-way hash function to realize access control services for low-cost EPC tags. The so-called "hash-lock" protocol works as follows. First, the owner of the tag generates a random key and sends the hash to the tag (i.e., the MetaID). After that, the tag stores the hash and locks its memory. To unlock the memory, the owner has to send the original key to the tag which hashes it and compares the digest with the stored MetaID.

Another proposal has been made by A. Shamir who presented the RFID protocol SQUASH (squashed form of SQUare-hASH) in 2008 [39]. He described a tag authentication scenario using a challenge-response protocol where the tag and the reader share a secret key S. The reader issues a random number R and sends it to the tag. After that, the tag calculates $H(S, R)$ where H represents a public hash function. The tag sends the hash back to the reader which can independently calculate the same message digest to proof the authenticity of the tag. As a cryptographic primitive, A. Shamir proposed to use the 64-bit SQUASH function, which is based on the well-studied Rabin encryption scheme. Note that the SQUASH function does not provide collision resistance since it is not necessarily required for the given RFID authentication scenario (this however lowers the resource requirements for practical implementations).

An approach to calculate a message digest using block ciphers has been proposed by H. Yoshida et al. [43] in 2005 and by A. Bogdanov et al. [12] in 2008. The latter authors presented DM-PRESENT which is based on the 64-bit cipher PRESENT as well as H-PRESENT that provides a 128-bit security level.

The first sponge-construction based hash function has been presented by G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche at the ECRYPT Hash Workshop in 2007 [10]. Since then, several hash-function proposals were made with respect to RFID applications including KECCAK, QUARK [2], Spongent [11], and Photon [19].

Related Work on Keccak Implementations. There exist several KECCAK implementations where most of them have been designed for FPGAs. High-speed implementations have been reported by J. Strömbergson [40], B. Baldwin et al. [3], E. Homsirikamol et al. [23], K. Kobayashi et al. [32], F. Gürkaynak et al. [21], and K. Gaj et al. [17, 18]. Low-area FPGA designs have been presented by S. Kerckhof et al. [30], J.-P. Kaps et al. [27], and B. Jungk and J. Apfelbeck [26].

In view of ASIC designs, there exist many high-speed variants proposed by S. Tillich et al. [41], A. Akin et al. [1], L. Henzen et al. [22], and X. Guo et al. [20].

Note that there also recently exists an open-source project at OpenCores.org [24]. To the authors' knowledge, there are only two publications that report a low-area implementation of KECCAK on ASICs. The KECCAK team reported numbers for a low-area version of KECCAK needing 9.3 kGEs (including memory) on a 130 nm CMOS process technology [9]. In 2010, E. B. Kavun and T. Yalcin presented several low-resource designs of KECCAK for RFID in [28]. Their full-state version (1 600 bits) needs about 20 kGEs on the same process technology.

3 Keccak Specification and Design Exploration

In this section, we first give a brief overview about KECCAK with the focus on parameters likely to be integrated in the SHA-3 standard. Afterwards, we explore different design decisions and discuss various optimizations for practical implementations.

The Sponge Construction. KECCAK is based on a new cryptographic hash family, the so-called sponge function family [6]. As opposed to existing hash constructions, which are classically based on the Merkle-Damgård construction, a fixed length permutation f is used to allow the handling of arbitrary length input and to produce fixed length outputs, e.g., 224, 256, 384, or 512 bits. The permutations are performed on a state with a fixed size of b bits.

The state is cut into two parts of size r (rate) and c (capacity), respectively. The rate defines the number of input bits which are processed in one block permutation. The capacity c of the sponge function represents the remaining bits of the state, i.e., $c = b - r$. The authors of KECCAK proposed values for r and c in their submitted KECCAK specification [8], e.g., $b = 1\,600$ bits, $r = 1\,088$ bits, and $c = 2n = 512$, where n is the length of the output.

Hashing works as follows. First, the state is initialized with 0^b and the input is padded to a length that is a multiple of r using the very simple multi-rate padding scheme [7]. After that, it is cut into blocks of size r. During the initial absorbing phase, the message blocks are XORed with the first r bits of the state followed by a single state permutation f. After the sponge has absorbed the whole message, it switches to the squeezing mode in which r bits are output iteratively (again followed by single state permutations f).

The Keccak-f Permutation. The authors of KECCAK proposed seven different state-permutation functions KECCAK-f that can be used. These permutation functions are further denoted by KECCAK-$f[b]$, where $b = 25 \times 2^\ell$ and ℓ ranges from 0 to 6. Note that the two largest permutations are KECCAK-$f[1600]$ and KECCAK-$f[800]$.

KECCAK-f organizes the b-bit state as a 3-D matrix with dimension $5 \times 5 \times w$, with $w = 2^\ell$. This matrix can be split into *slices* and *lanes*. A *slice* is a matrix composed of 25 bits with constant z coordinate (5 bits in each row and 5 bits in each column). A *lane* is a simple array consisting of w bits of constant x and y coordinate. Figure 1 shows the structure of the state.

The KECCAK-f permutation is a round based function, each of the $12 + 2\ell$ rounds consists of five parts:

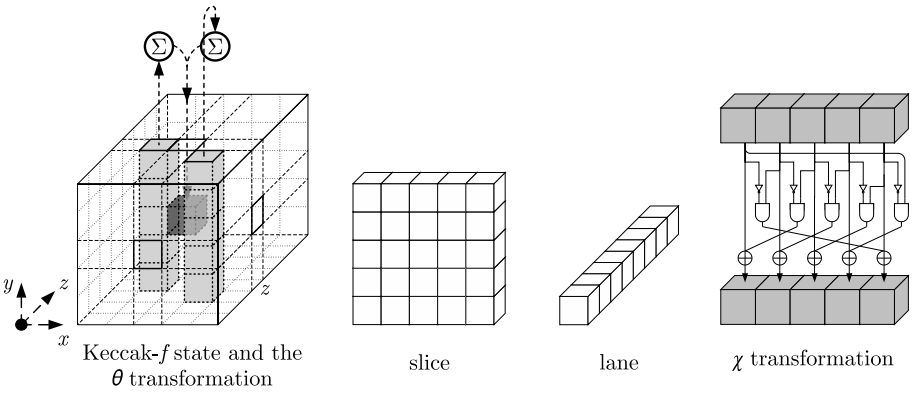

Fig. 1. Parts of the KECCAK-f state [7] including θ transformation (left) and χ transformation (right)[2]

θ **:** The parity of two nearby columns is added to each column, see left image in Figure 1.
ρ **:** All lanes are rotated by a defined offset.
π **:** The 25 lanes are transposed in a fixed pattern, i.e., the bits of each slice are permuted.
χ **:** The 5 bits of each row are non-linearly combined using AND gates and inverters and the result is added to the row as depicted in the right image of Figure 1.
ι **:** A w-bit round constant is added (XORed) to a single lane.

For a more in-depth explanation of KECCAK we refer to the KECCAK reference [7].

3.1 Design Exploration and Decisions

We decided to analyze the hardware complexity of KECCAK-f with a state size of both 1 600 (full-state) and 800 bits. For each design, we implemented two versions. The first version aims for lowest power and lowest area (Version 1). The second version (Version 2) targets the same goals but tries to find an optimal trade-off between power, area, and speed without causing a significant weight gain in one direction. For both designs, we decided to use low width datapaths, i.e., 8 and 16 bits. This is because lower datapath widths would result in unacceptable throughput penalties while higher datapath widths exceed the limited power and area requirements. Moreover, we serialized all operations and the applied components have been re-used as much as possible.

[2] The figures have been taken from the KECCAK website [29] and are available under the Creative Commons Attribution license.

Figure 2 shows the basic hardware architecture of our designs. It consists of a controller, a datapath, a Look-up Table (LUT) for constants, an input/output interface, and an external RAM block. As a requirement, our design should feature all necessary components for KECCAK (permutation calculation, sponge function, input handling including padding) and should be flexible (support multiple output lengths).

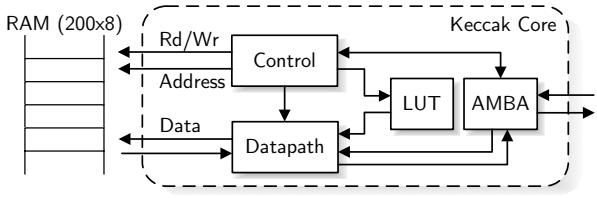

Fig. 2. Basic hardware architecture

Memory Type and I/O Interfaces. We decided to use RAM macros for state storage because they require typically less resources than standard-cell-based designs (in terms of power and area). For our first version, we decided to use an 8-bit interface; for the second version we use an 16-bit interface (to improve speed). As a major requirement, no more than b bits (the size of the state, e.g., 1 600 bits) should be used. As input/output interface, we chose to implement an 8-bit AMBA APB interface, which is very simple and provides a standardized communication interface.

Constants: LUT vs. LFSR. The round constants for the ι transformation as well as the ρ rotation offsets should be stored in a simple LUT. The round constants can be also generated using a 7-bit Linear Feedback Shift Register (LFSR) but this would require more power and area.

Lane- and Slice-Wise Processing. Software implementations as well as the compact co-processor described in [9] operate lane-wise, i.e., lanes are fetched from the memory and are subsequently processed. This approach however needs a lot of additional storage and is slow on the small data buses we are using.

An interesting alternative, namely slice-wise processing, was proposed by B. Jungk and J. Apfelbeck [26]. Although initially designed and implemented for FPGAs, slice-wise processing serves as an excellent starting point for a low-resource ASIC implementation. All operations except ρ can be performed on a slice-per-slice basis. In order to perform these four transformations on a slice in a single cycle, the rounds of the KECCAK-f permutation must be rearranged: the initial round solely consists of θ and ρ, followed by 23 rounds of π, χ, ι, θ and ρ, and the final round consists of π, χ and ι. This round schedule differs slightly from the one used by Jungk and Apfelbeck.

The ρ transformation as well as the sponge computations cannot be performed slice-wise but only on a lane-per-lane basis. For this reason, we use both

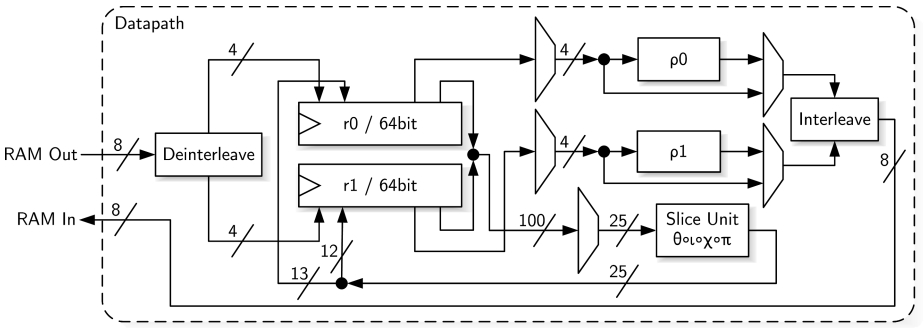

Fig. 3. Datapath architecture of our Keccak design

lane- and slice-wise processing and combine these two approaches into a single datapath. This combination is a challenge when using an external memory as it must both be possible to access slices as well as lanes while still using the full bandwidth of the memory bus and keeping the core's internal storage small. We tackle this problem using a technique called interleaving which will be explained in the next section.

Low-Power Optimizations. To reduce current drain, we integrated clock gating and operand isolation techniques. In the case of clock gating, registers are only clocked whenever new values should be stored. Operand isolation sets the inputs of combinational parts, whose outputs are not needed in the current cycle, to a constant value, i.e., to 0. Both these methods reduce switching activity which is the main contributor to power consumption in CMOS technology. Applying these techniques to our design helps us to drastically reduce power consumption while the area impact is kept low.

4 The Keccak Architectures

In this section, we first describe two hardware architectures for the full-state Keccak algorithm. Our first design (Version 1) aims for lowest power and area. Our second design (Version 2) trades area for higher throughput. After that, we discuss the implications of smaller state sizes and present two architectures using 800 bits only.

4.1 Version 1: Pushing the Limits towards Lowest Power and Area

Figure 3 shows the datapath architecture of our design. It provides an 8-bit memory interface and is mainly composed of an interleave and de-interleave unit, two 64-bit registers, one slice unit, and two ρ units.

Interleaved Storage. The 1 600-bit state is not stored linearly in the RAM (i.e., lane after lane) but interleaved: two adjacent lanes—each containing 64 bits—are interleaved into a single 128-bit word. On even positions of the interleaved

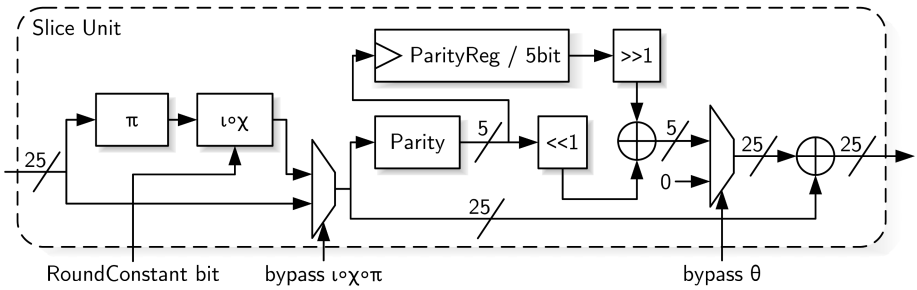

Fig. 4. Architecture of the slice-processing unit

word, bits of the lower lane are stored while odd positions contain the bits of the upper lane. Using this technique, a single n-bit memory word contains information about 2 lanes but only $n/2$ slices. This fact helps us to drastically decrease the size of the internal memory needed as will be explained later. Due to the fact that the state consists of an odd number of lanes, one selected lane has to be stored non-interleaved; we chose the lane $[0, 0]$, since this is the only one with a ρ offset equal to 0. Therefore, we can skip this single lane in this phase.

Combined Slice- and Lane-Processing. The two 64-bit registers $r0$ and $r1$ combined either store two lanes or four slices. In the latter case, only 100 out of 128 bits are used. The interleaved memory technique described above allows us to load and store two lanes at full bus speed (i.e., 16 memory cycles on an 8-bit bus) and four slices in only 13 cycles. When not using interleaving, the size of the registers need to be increased to 100 bits in order to store 8 slices.

Figure 4 shows the architecture of the slice-processing unit. The π operation is a rewiring of the input, χ is computed on the 5 rows of one slice in parallel, and ι is a single XOR with a bit of the round constant. For the θ transformation, the column parity of the previous slice is stored in a 5-bit register. The parity of a slice is computed and XORed to the stored parity. The result is then added to each of the 5 rows. In the initial and final round, some parts must be skipped. For this reason, two multiplexers allow bypassing of blocks.

A single ρ unit is made up of a barrel shifter and a register with half the size of the memory-bus width. The upper 4 bits of the rotation offset are handled by proper register addressing while the lower 2 bits are done by actual shifts to the left.

The Round Computation. The computation of a single modified round consists of two main phases: the slice-processing phase and the ρ transformation phase:

– In the slice-processing phase, the column parity of slice 63 (after having applied $\iota \circ \chi \circ \pi$) is first computed and stored in the parity register. Then, the following is repeated 16 times: four slices are loaded within 13 clock

cycles and after performing $\theta \circ \iota \circ \chi \circ \pi$ on each slice, the result is stored in memory.
- For the ρ phase, two lanes are fetched from memory. With the help of two separate ρ units, the lanes are implicitly rotated by the specified offsets and stored back to memory. This is done for all 24 lanes which have an offset other than 0.

4.2 Version 2: Trading Area for Higher Throughput

The previously described design requires low resources in terms of power and area but lacks in speed and throughput. The main drawback is the use of an 8-bit memory interface and the asymmetric datapath. During the slice processing, 25 bits are processed at once while the ρ phase operates on only 8 bits which is inefficient in terms of power.

We therefore make use of a 16-bit memory interface that allows writing of single bytes to trade some gates for higher speeds. The cycle count for the ρ phase is therefore cut into half. For the slice-processing unit, this is not the case. Instead, a single 16-bit word has information on 8 slices but only 4 slices can be stored in the 128-bit internal register. Thus, 8 bits have to be discarded. With further optimizations (reading the upper byte of a 16-bit memory word in the next cycle after writing the lower byte) the cycle count for the permutation can be decreased by about 30 %. The number of additional gates for these modifications is marginal and limited to the need of 8-bit wide ρ units (shifter and register) and the increase of the RAM-macro cell due to the additional 8-bit pre-charge logic, write logic, and sense amplifiers.

4.3 Adapting to an 800-Bit State

Our design can also be used with an 800-bit state, only small additions to the controller are necessary to support both state sizes. When restricting to 800 bits, some optimizations are possible. First, only half of the RAM size is required. Second, the size of the internal registers can be cut down to a total of 100 bits, i.e., the memory needed to store four slices. A single lane now consists of 32 bits, this reduces memory requirements in the lane-processing phase to 64 bits. Furthermore, the number of rounds is reduced from 24 to 22. The cycle count needed for a single KECCAK-f round is reduced by a factor of 2. For detailed implementation results see Section 5.

A possible trade-off between area and speed is to extend the used interleaving scheme to more than two lanes. When interleaving four 32-bit lanes into one 128-bit word, four lane registers and a 16-bit memory interface are needed. The core area will be comparable to that of the 1 600-bit version, while saving roughly 1 000 cycles per permutation compared to the 16-bit 2-lane case. However, we did not implement this approach to minimize the area requirements.

For even smaller state sizes, i.e., 400 or 200 bits, the number of lanes used in the interleaving scheme has to be chosen according to the desired cycle count and area requirements.

Table 1. Area of chip components for our low-area version (Version 1)

Component	GEs
Datapath	1 922
r0+r1	1 213
Slice unit	382
ρ units	38
Controller	598
LUT	144
AMBA IO	69
Core Total	**2 927**
RAM macro	2 595
Total	**5 522**

Table 2. Area of chip components for our higher-throughput version (Version 2)

Component	GEs
Datapath	2 083
r0+r1	1 205
Slice unit	382
ρ units	119
Controller	646
LUT	144
AMBA IO	69
Core Total	**3 148**
RAM macro	2 750
Total	**5 898**

5 Results

We implemented both designs in VHDL using a mixed tool design flow. For synthesis, we used the Synopsys Design Compiler 2012.06 that generates a netlist targeting the *FSC0L_D* standard-cell library from Faraday. This library is based on the UMC 0.13 μm low-leakage process which has a standard supply voltage of 1.2 V. The following area results have been obtained after synthesis (using low-area optimizations enabled); power values have been generated using Cadence Encounter Power System v8.10 after place and route (using Cadence Encounter RTL-to-GDSII). We further used low-leakage RAM macros from Faraday as storage blocks. Circuit size is expressed in terms of *gate equivalences* (GE), 1 GE is the area occupied by a 2-input NAND Gate. All values have been determined for a hash output length of 256 bits, the capacity c was set to 512 bits as suggested by the Keccak authors [8].

Table 1 and Table 2 show the area usage of our 1 600-bit designs for different chip components. For our lowest-area version, the two registers use almost 40 % of the occupied area. The slice unit needs the largest combinational part with 13 %. The higher-throughput version needs slightly more area mainly due to the larger ρ units, the controller, and the 16-bit RAM macro interface, i.e., 221 GEs for the core (and 155 GEs in addition for the larger RAM macro). In total it is 6.38 % larger.

Table 3 provides more results including throughput and power. It shows that our higher-throughput version needs 32 % less clock cycles (15 427 instead of 22 570); this translates to a throughput of 44.3 kbps (for Version 1) and 64.8 kbps (for Version 2) at a clock frequency of 1 MHz. The power consumption values are nearly the same: our low-area version needs 5.5 μW per MHz of power (core only) and 12.5 μW per MHz (with memory included) and our higher-throughput version needs 5.6 μW per MHz and 13.7 μW per MHz, respectively. The maximum frequency of the core is 61 MHz.

Table 3. Comparison of 1 600-bit KECCAK, SHA-1, and SHA-256 implementations

	Techn. [nm]	Area [GEs]	Power [μW/MHz][a]	Cycles/ Block[b]	Throughput @1MHz [kbps]
Ours, Version 1	130	5 522	12.5	22 570	44.3
Ours, Version 2	130	5 898	13.7	15 427	64.8
KECCAK team [9][c]	130	9 300	*N/A*	5 160	210.9
Kavun et al. [28]	130	20 790	44.9	1 200	906.6
SHA-1 [33]	130	5 527	23.2	344	1 488.0
SHA-1 [14]	350	8 120	-	1 274	401.8
SHA-256 [31]	250	8 588	-	490	1 044.0
SHA-256 [14]	350	10 868	-	1 128	454.0

[a] Power values of designs using different process technologies are omitted
[b] Blocksizes: 1 600-bit KECCAK: 1 088 bits [8], SHA-1 & SHA-256: 512 bits
[c] The KECCAK implementation of [9] is based on a 64-bit memory interface. The co-processor requires 5 kGEs and an external memory of 3 520 bits is required (9.3 kGEs in total). It does not feature sponge and padding functionality.

Comparison with Related Work. We compare our solutions with the two most relevant publications of low-resource full-state KECCAK implementations. It shows that our work requires significantly less area, i.e., 41 % compared to the implementation of [9] (note that the authors estimated the total size of their low-area design to 9.3 kGEs including an external 64-bit memory). Our design is also more compact than the work of E. B. Kavun and T. Yalcin [28] (about a factor of 4). We also compare our designs with the smallest SHA-1 and SHA-2 implementations from [33] and [31]. It shows that our design has about the same size as SHA-1 and needs about 36 % less area than SHA-2. The power values of our design are also compelling requiring less than 15 μW per MHz (including memory), this is 72 % less than [28].

5.1 Results for an 800-Bit State

We also adapted our design for use with an 800-bit state. As a result, the size of the core could be decreased by roughly 300 GEs (mainly due to the use of smaller registers, cf. Section 4.3). In fact, 2 611 GEs are needed for our low-area version (Version 1) and 2 837 GEs are needed for the higher-throughput variant (Version 2). In addition to these savings, the RAM size requirements are halved. The 8-bit RAM macro for the low-area version needs 2 016 GEs and the 16-bit RAM macro needs 2 108 GEs. Thus, our designs require 4 627 GEs and 4 945 GEs in total, respectively.

Regarding power consumption, the smaller state versions need slightly less power, i.e., 12.4 and 13.1 μW per MHz. The cycle count for both versions drops by more than 50 %. 10 712 clock cycles are needed for Version 1 and 7 464 clock cycles are required for Version 2. The throughput, however, suffers due to the

smaller chosen blocksize of $800 - 2 \times 256 = 288$ bits. It decreases to 26.9 and 38.6 kbps.

5.2 Discussion

As already stated in the introduction and in Section 2, our primary goal was to determine a lower bound for Keccak in terms of power and area. The following points invite to further discussions:

- The throughput of our design is relatively low but still acceptable for the targeted RFID applications. Increasing throughput is possible by adapting our design to broader memory interfaces (i.e., 32 bits). This of course will increase the area and power requirements.
- The use of 1 600 and 800-bit Keccak for low-cost passive RFID tags has to be considered with caution: our smallest design requires about 5.5 kGEs and 4.6 kGEs, respectively. But there exist more compact hardware implementations that use primitives like block ciphers which can be used in a mode to provide hashing capabilities [12, 13].
- Integration: if external memory is available, e.g., in implementations where other chip components share a common memory, only the core logic has to be integrated requiring around 3 kGEs. Note that our design makes use of an 8-bit (standardized) AMBA interface and can therefore be easily adopted for existing designs.
- The difference between the 1 600 and 800 bit versions of our Keccak implementations is significant. The 800-bit version is about 900 GEs smaller in size while being twice as fast.
- For even more "lightweight" applications, the properties of the design might be modified (though might not being standard conform anymore), e.g., modifying the level of collision-resistance property; or reducing the size of the state to 400 or less bits as suggested by [28]. Note that such smaller state versions are specified from the Keccak team but will not likely be part of the SHA-3 standard.
- We did not integrate any countermeasures against implementation attacks which has to be considered in scenarios where Keccak is used for authenticated encryption, for instance. Keccak can be protected using, for example, secret-sharing techniques as shown by G. Bertoni [4, 5]. Note that this will increase the area requirements. Future work has to evaluate low-resource SCA and fault-attack countermeasures for Keccak.

6 Conclusions

With the results given in this paper, we show that full-state Keccak can be implemented with less than 5.5 kGEs. There is room for improvements and it can be expected that the limits will be further pushed down towards an acceptable border where an integration into passive low-cost tags is getting more attractive. By now and without making any modification and restrictions for certain RFID applications, we obtain power values that are below 15 μW at 1 MHz (thus guaranteeing high reading ranges) while providing 128-bit of security.

Acknowledgements. The work has been supported by the European Commission through the ICT program under contract ICT-SEC-2009-5-258754 (Tamper Resistant Sensor Node - TAMPRES) and by the Austrian Science Fund (FWF) under the grant number TRP251-N23.

References

1. Akin, A., Aysu, A., Ulusel, O.C., Savaş, E.: Efficient Hardware Implementations of High Throughput SHA-3 Candidates Keccak, Luffa and Blue Midnight Wish for Single- and Multi-Message Hashing. In: 3rd International Conference Security of Information and Networks–SIN 2010, Taganrog, Russia, September 7-11, pp. 168–177 (2010)
2. Aumasson, J.-P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A Lightweight Hash. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 1–15. Springer, Heidelberg (2010)
3. Baldwin, B., Byrne, A., Lu, L., Hamilton, M., Hanley, N., O'Neill, M., Marnane, W.P.: FPGA Implementations of the Round Two SHA-3 Candidates. In: International Conference on Field Programmable Logic and Applications–FPL 2010, Milano, Italy, August 31-September 2, pp. 400–407 (2010)
4. Bertoni, G., Daemen, J., Debande, N., Le, T.-H., Peeters, M., Van Assche, G.: Power Analysis of Hardware Implementations Protected with Secret Sharing. Cryptology ePrint Archive: Report 2013/067 (February 2013)
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Building Power Analysis Resistant Implementations of Keccak. In: Second SHA-3 Candidate Conference (August 2010)
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic sponge functions. Submission to NIST (Round 3) (2011)
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak reference. Submission to NIST (Round 3) (2011)
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak SHA-3 submission. Submission to NIST (Round 3) (2011)
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Keer, R.V.: Keccak Implementation Overview, V3.2 (2012)
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: ECRYPT Hash Workshop, Barcelona, Spain, May 24-25 (2007), `http://sponge.noekeon.org/SpongeFunctions.pdf`
11. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: Spongent: A Lightweight Hash Function. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 312–325. Springer, Heidelberg (2011)
12. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash Functions and RFID Tags: Mind the Gap. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 283–299. Springer, Heidelberg (2008)
13. Feldhofer, M., Rechberger, C.: A Case Against Currently Used Hash Functions in RFID Protocols. In: Dominikus, S. (ed.) Workshop on RFID Security 2006 (RFIDSec06), Graz, Austria, July 12-14, pp. 109–122 (July 2006)
14. Feldhofer, M., Rechberger, C.: A Case Against Currently Used Hash Functions in RFID Protocols. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4277, pp. 372–381. Springer, Heidelberg (2006)

15. Feldhofer, M., Wolkerstorfer, J.: Hardware Implementation of Symmetric Algorithms for RFID Security. In: RFID Security: Techniques, Protocols and System-On-Chip Design, pp. 373–415. Springer (2008)
16. Finkenzeller, K.: RFID-Handbook, 2nd edn. Carl Hanser Verlag (April 2003) ISBN 0-470-84402-7
17. Gaj, K., Homsirikamol, E., Rogawski, M.: Fair and Comprehensive Methodology for Comparing Hardware Performance of Fourteen Round two SHA-3 Candidates using FPGAs. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 264–278. Springer, Heidelberg (2010)
18. Gaj, K., Homsirikamol, E., Rogawski, M., Shahid, R., Sharif, M.U.: Comprehensive Evaluation of High-Speed and Medium-Speed Implementations of Five SHA-3 Finalists Using Xilinx and Altera FPGAs. Cryptology ePrint Archive: Report 2012/368 (June 2012)
19. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841. Springer, Heidelberg (2011)
20. Guo, X., Huang, S., Nazhandali, L., Schaumont, P.: Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations. In: Second SHA-3 Candidate Conference 2010 (2010)
21. Gürkaynak, F.K., Gaj, K., Muheim, B., Homsirikamol, E., Keller, C., Rogawski, M., Kaeslin, H., Kaps, J.-P.: Lessons Learned from Designing a 65nm ASIC for Evaluating Third Round SHA-3 Candidates. In: Third SHA-3 Candidate Conference (March 2012)
22. Henzen, L., Gendotti, P., Guillet, P., Pargaetzi, E., Zoller, M., Gürkaynak, F.K.: Developing a Hardware Evaluation Method for SHA-3 Candidates. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 248–263. Springer, Heidelberg (2010)
23. Homsirikamol, E., Rogawski, M., Gaj, K.: Comparing Hardware Performance of Round 3 SHA-3 Candidates using Multiple Hardware Architectures in Xilinx and Altera FPGAs. In: CRYPT II Hash Workshop 2011 (May 2011)
24. Hsing, H.: Sha3 (keccak). OpenCores.org (January 2013)
25. Juels, A., Weis, S.A.: Defining Strong Privacy for RFID. Cryptology ePrint Archive, Report 2006/137 (April 2006), `http://eprint.iacr.org/`
26. Jungk, B., Apfelbeck, J.: Area-Efficient FPGA Implementations of the SHA-3 Finalists. In: International Conference on Reconfigurable Computing and FPGAs–ReConFig 2011, Cancun, Mexico, November 30-December 2, pp. 235–241 (2011)
27. Kaps, J.-P., Yalla, P., Surapathi, K.K., Habib, B., Vadlamudi, S., Gurung, S., Pham, J.: Lightweight Implementations of SHA-3 Candidates on FPGAs. In: Bernstein, D.J., Chatterjee, S. (eds.) INDOCRYPT 2011. LNCS, vol. 7107, pp. 270–289. Springer, Heidelberg (2011)
28. Kavun, E.B., Yalcin, T.: A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications. In: Ors Yalcin, S.B. (ed.) RFIDSec 2010. LNCS, vol. 6370, pp. 258–269. Springer, Heidelberg (2010)
29. Keccak Design Team. The Keccak sponge function family, `http://keccak.noekeon.org/`
30. Kerckhof, S., Durvaux, F., Veyrat-Charvillon, N., Regazzoni, F., de Dormale, G.M., Standaert, F.-X.: Compact FPGA Implementations of the Five SHA-3 Finalists. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 217–233. Springer, Heidelberg (2011)

31. Kim, M., Ryou, J., Jun, S.: Efficient Hardware Architecture of SHA-256 Algorithm for Trusted Mobile Computing. In: Yung, M., Liu, P., Lin, D. (eds.) Inscrypt 2008. LNCS, vol. 5487, pp. 240–252. Springer, Heidelberg (2009)
32. Kobayashi, K., Ikegami, J., Knežević, M., Guo, E.X., Matsuo, S., Huang, S., Nazhandali, L., Kocabas, Ü., Fan, J., Satoh, A., Verbauwhede, I., Sakiyama, K., Ohta, K.: Prototyping Platform for Performance Evaluation of SHA-3 Candidates. In: IEEE International Symposium on Hardware-Oriented Security and Trust– HOST 2010, Anaheim, California, USA, June 13-14, pp. 60–63 (2010)
33. O'Neill, M.: Low-Cost SHA-1 Hash Function Architecture for RFID Tags. In: Dominikus, S. (ed.) Workshop on RFID Security 2008 (RFIDsec 2008), pp. 41–51 (July 2008)
34. Ranasinghe, D.C., Cole, P.H.: Networked RFID Systems and Lightweight Cryptography. Springer, Berlin (2008)
35. Saarinen, M.-J.O., Engels, D.: A do-it-all-cipher for rfid: Design requirements (extended abstract). Cryptology ePrint Archive: Report 2012/317 (June 2012)
36. Sarma, S.: Towards the 5 Cent Tag. White paper, MIT Auto-ID Center (2001)
37. Sarma, S.E., Weis, S.A., Engels, D.W.: Radio Frequency Identification: Risks and Challenges. CryptoBytes (RSA Laboratories) 6(1), 325 (2003)
38. Sarma, S.E., Weis, S.A., Engels, D.W.: RFID Systems and Security and Privacy Implications. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 454–469. Springer, Heidelberg (2003)
39. Shamir, A.: SQUASH A New MAC with Provable Security Properties for Highly Constrained Devices Such as RFID Tags. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 144–157. Springer, Heidelberg (2008)
40. Strömbergson, J.: Implementation of the Keccak Hash Function in FPGA Devices. Technical report, InformAsic AB (2008)
41. Tillich, S., Feldhofer, M., Kirschbaum, M., Plos, T., Schmidt, J.-M., Szekely, A.: Hardware Implementations of the Round-Two SHA-3 Candidates: Comparison on a Common Ground. In: Proceedings of Austrochip 2010, Villach, Austria, October 6, pp. 43–48 (2010) ISBN 978-3-200-01945-4
42. Weis, S.A., Sarma, S.E., Rivest, R.L., Engels, D.W.: Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems. In: Hutter, D., Müller, G., Stephan, W., Ullmann, M. (eds.) Security in Pervasive Computing 2003. LNCS, vol. 2802, pp. 201–212. Springer, Heidelberg (2004)
43. Yoshida, H., Watanabe, D., Okeya, K., Kitahara, J., Wu, H., Küçük, Ö., Preneel, B.: MAME: A Compression Function with Reduced Hardware Requirements. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 148–165. Springer, Heidelberg (2007)