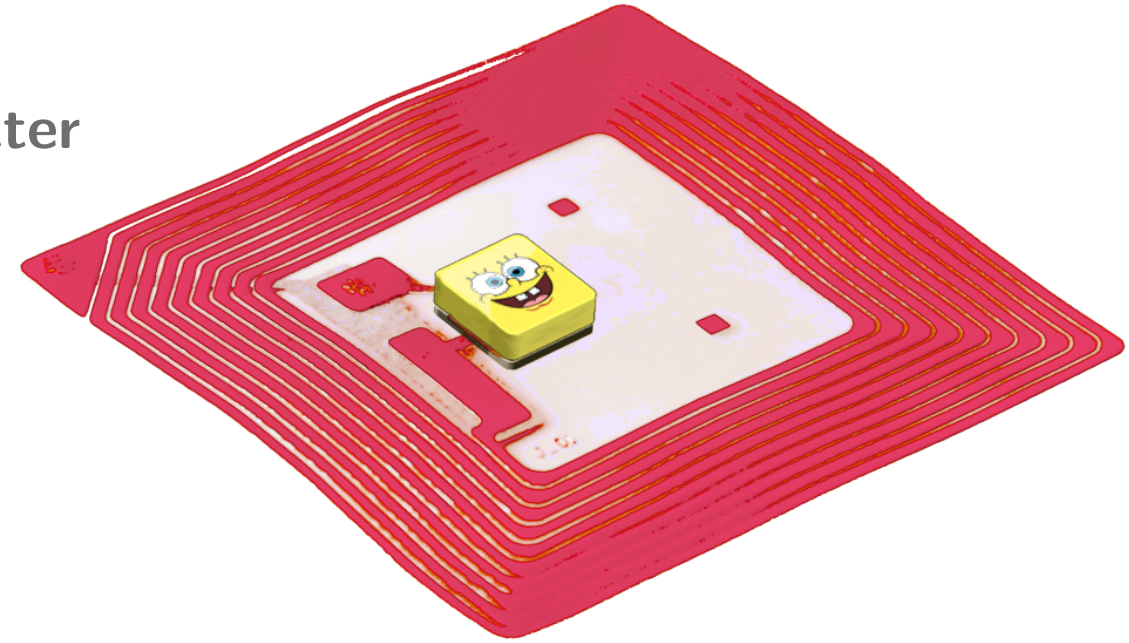
Pushing the Limits of SHA-3 Hardware Implementations to Fit on RFID

Peter Pessl and Michael Hutter



Co-Author

Peter Pessl

- VHDL implementation of KECCAK
- Currently working on integrating KECCAK into low-resource ECDSA

Outline

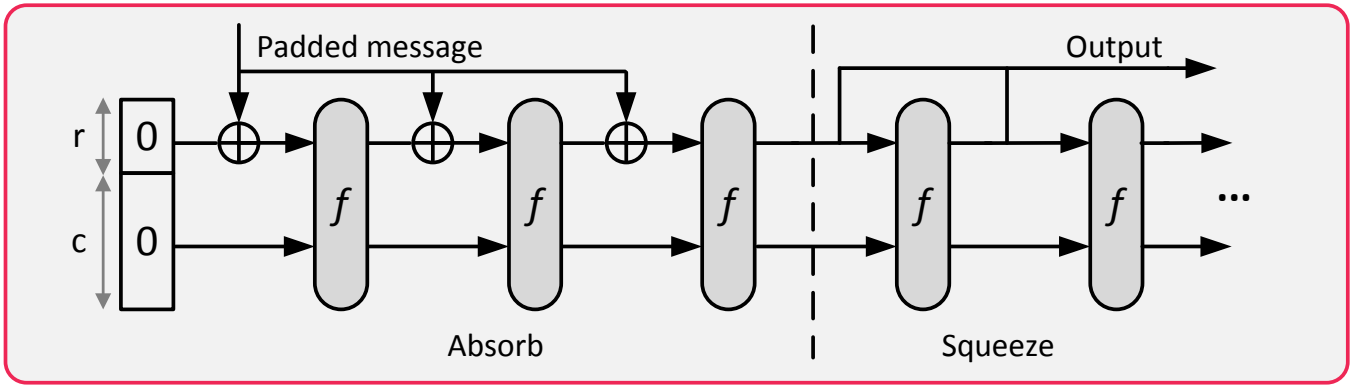
Motivation

- KECCAK as winner of the SHA-3 contest
- Main goal: what are the lower bounds of KECCAK in terms of area and power?
- How do highly serialized (8 or 16-bit) versions perform on ASICs?

- Suitable candidate for low-cost passive RFID?
  - Power should be less than 15 μW at 1 MHz (reading range)
  - Few milliseconds of response time OK (not recognizable by humans)
- Follow the RFID design principle: "*few gates and many cycles*" as suggested by S. Weis [10]

Keccak

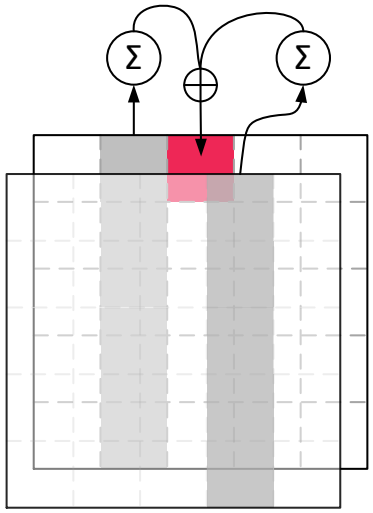
- Cryptographic *sponge* function family
- Instances call b-bit permutations f with parameters r, c:
  - r bits of rate
  - c bits of capacity (defines the security level of $2^{c/2}$)
  - $b = r + c = 25, 50, 100, 200, 400, 800$ or 1600
- SHA-3 instance example
  - $b = 1600$ with $r = 1088$ and $c = 512$
  - 256-bit security



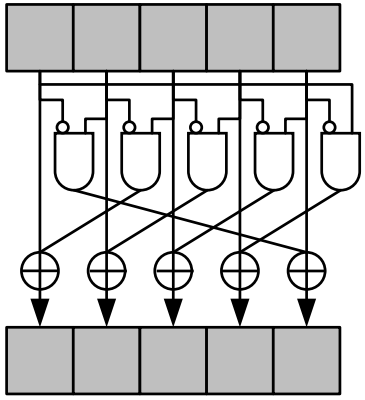
The Keccak-f Permutation

- Block permutations on a $b = 5 \times 5 \times 2^\ell$-bit state matrix, where $\ell \in [0, 6]$
- Consists of $12 + 2\ell$ rounds with 5 sub-functions:

Θ	Adds the parity (linear diffusion)
ρ	Cyclic shifts of lanes (slice dispersion)
π	Slice permutation (break alignment)
χ	Combination of rows (non-linearity)
ι	Add round constant (avoid symmetry)



Θ



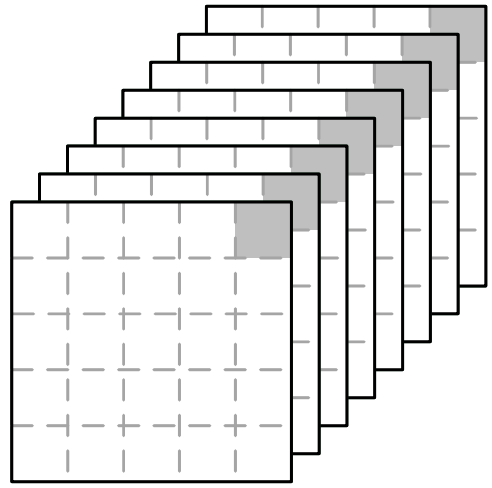
χ

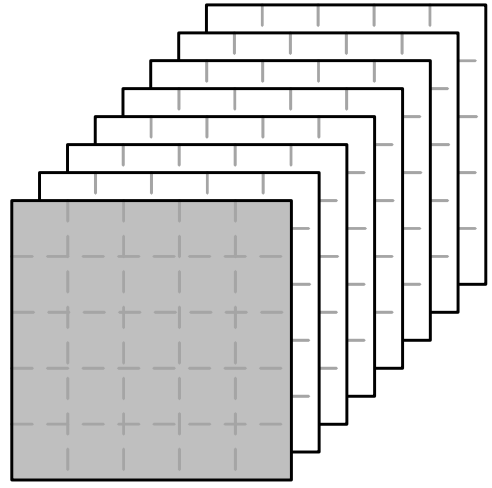
Design Exploration and Decisions

- We target KECCAK[1600] and KECCAK[800]
  - ...because most likely to be standardized
- For each target, we implement two versions:
  - 8-bit version: aims for lowest area
  - 16-bit version: trading area for higher throughput
- Memory type and I/O interface
  - Use of RAM macros for state storage
  - Standardized 8/16-bit AMBA APB interface
- Constants: LUT vs. LFSR
  - Round constants for ρ and ι stored in LUT
  - No dedicated LFSR unit required

Lane-wise vs. Slice-wise Processing

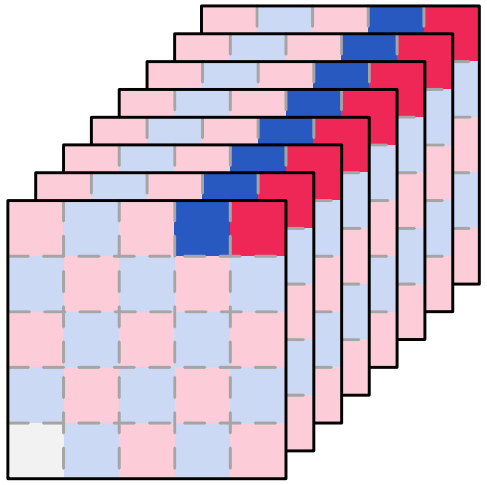
- Lane-wise processing
  - Often applied in SW
  - A lane with 2^ℓ bits is stored in $8, 16, 32$, or 64-bit registers
  - Can be combined with **bit interleaving:**
    - ✓ Helps to improve the performance of ρ
    - ✓ Reduces costly instructions necessary for rotation
- Slice-wise processing
  - More HW oriented
  - Round function has to be re-scheduled
  - Example: Jungk and Apfelbeck [6]
    - ✓ Processed 8 slices in parallel
    - ✓ ρ permutation required extra registers and special RAM addressing
    - ✓ Stored the state in 25 8×8 RAMs


lane

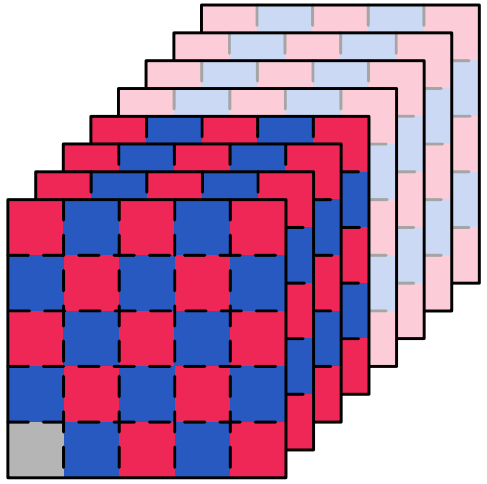

slice

Idea

- Apply **lane interleaving**
  - ✓ Store pairs of lanes interleaved in RAM
  - ✓ Each 8-bit word in RAM contains information about 2 lanes and 4 slices
  - ✓ Allows to efficiently process 4 slices instead of 8
- Combine lane and slice-wise processing in a single datapath
  1. Lane-processing phase:
     - ✓ Apply ρ on two entire 64-bit lanes
     - ✓ No RAM addressing issues (implicit rotation)
  2. Slice-processing phase:
     - ✓ Process 4 slices
- Allows usage of 200×8 RAM



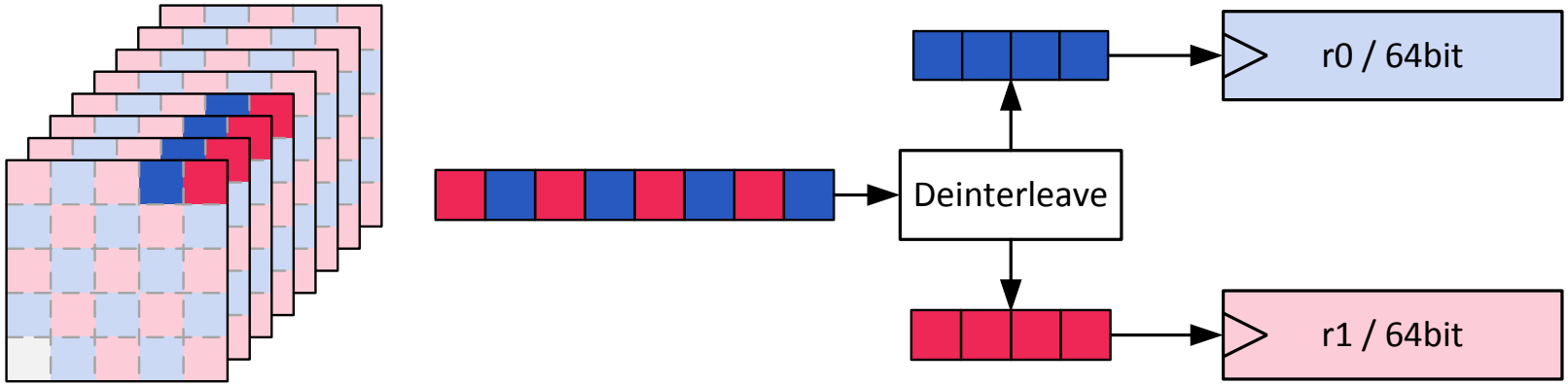
lane



slice

Lane Interleaving

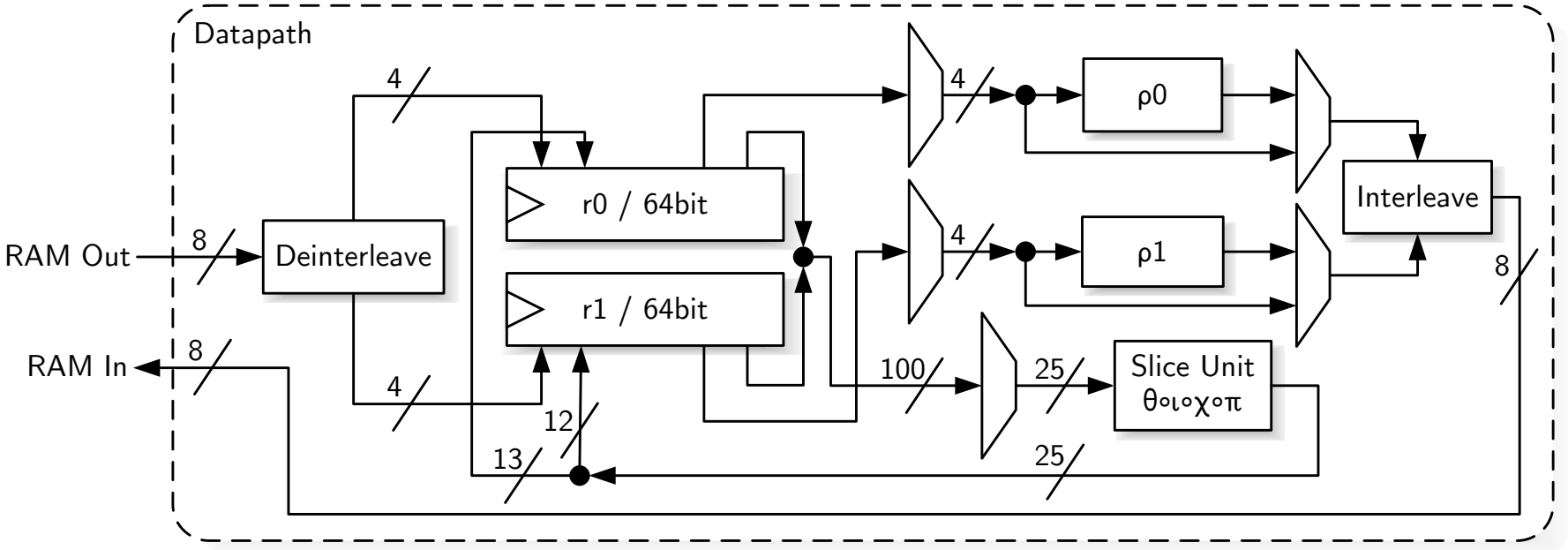
- Two shared 64-bit registers $r0$ and $r1$
  - Used to store 2 lanes or 4 slices
  - $r0$ stores odd lanes and $r1$ stores even lanes
- Only 24 lanes interleaved
  - Lane[0,0] has zero rotation offset



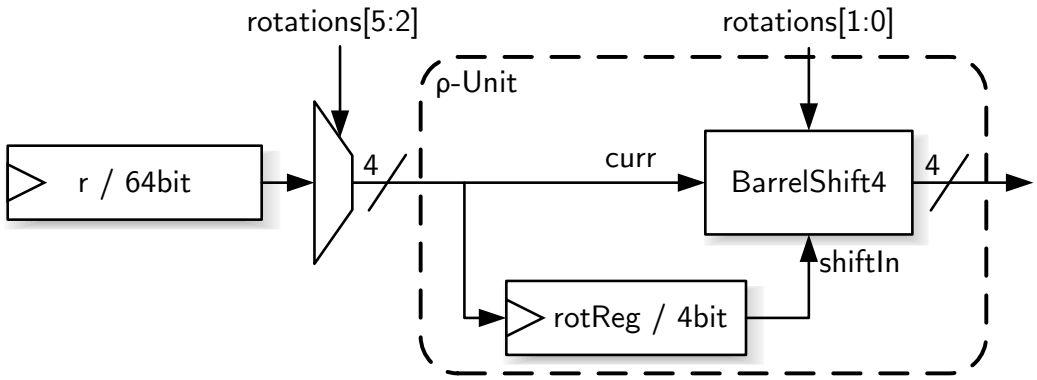
Ressource Requirements

- Two shared 64-bit registers
- Interleave/Deinterleave unit
- Two ρ units
  - Rotate two lanes in parallel
  - Two 4-bit rotation registers and Barrel shifters
- Slice unit
  - Reuse of rotation registers to store parities for Θ
- Re-schedule of round function (25 rounds):
  - First round: $\rho \circ \Theta$
  - 23 rounds: $\rho \circ \Theta \circ \iota \circ \chi \circ \pi$
  - Last round: $\iota \circ \chi \circ \pi$

The Datapath Architecture



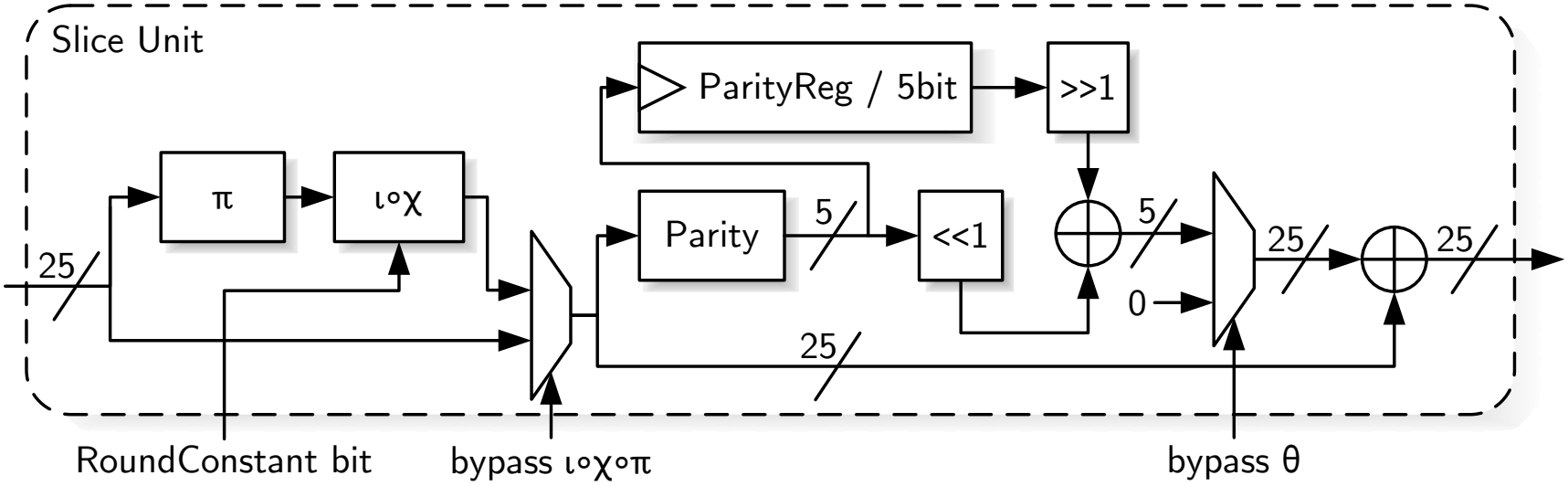
Lane Processing



- Load and deinterleave two 64-bit lanes (16 cycles)
- Apply ρ on entire lanes
  - 1 init cycle for pre-setting rotation register
  - Implicitly rotation by specified offsets using Barrel shifter
- Store two 64-bit lanes back interleaved (16 cycles)

Slice Processing

- Load and deinterleave 4 slices with consecutive z-coordinates (13 cycles)
- Permutation of Θ, ι, χ, π in a single cycle
- Parities of previous slice columns are stored in a 5-bit parity register
- Resources for parity register are shared with rotation registers for ρ



8-bit vs. 16-bit Version

- Drawbacks of 8-bit version
  - Narrow memory interface
  - Asymmetric datapath
    - ✓ 25-bits for slice unit
    - ✓ 8-bits for the two ρ units
- Trading area for higher throughput
  - 16-bit RAM macro instead of 8-bit
    - ✓ Allows writing of single bytes
  - Two 8-bit ρ units (instead of 4 bits)
    - ✓ Twice as fast
  - No modifications for slice unit (e.g., process 8 slices instead of 4)

Results

Table 1 : Area of chip components for our low-area version (8-bit)

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 (16-bit)

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**

Comparison with Related Work

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

	Techn. [nm]	Area [GEs]	Power [μW/MHz][a]	Cycles/ Block[b]	Throughput @1MHz [kbps]
Ours, 8-bit version	130	5 522	12.5	22 570	48.2
Ours, 16-bit version	130	5 898	13.7	15 427	70.5
KECCAK team [4]	130	9 300	N/A	5 160	210.9
Kavun et al. [7]	130	20 790	44.9	1 200	906.6
SHA-1 [9]	130	5 527	23.2	344	1 488.0
SHA-1 [5]	350	8 120	-	1 274	401.8
SHA-256 [8]	250	8 588	-	490	1 044.0
SHA-256 [5]	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 [3], SHA-1 & SHA-256: 512 bits

What About Keccak[800]?

- Optimizations
  - RAM size halved
  - Size reduction of internal registers
    - ✓ 100 bits (2×50) instead of 128 (2×64)
    - ✓ Memory needed to store 4 slices or 2 lanes (2×32)
  - KECCAK-f is twice as fast
  - Round reduction from 24 to 22
- Synthesis results:

Table 4 : KECCAK[800] results

Keccak[800]	Techn. [nm]	Area [GEs]	Power [μW/MHz]	Cycles Block[a]	Throughput @1MHz [kbps]
8-bit version	130	4 627	12.4	10 712	26.9
16-bit version	130	4 945	13.1	7 464	38.6

[a] Blocksizes: 800-bit KECCAK: $r = 288$ bits [3]

Further Research Suggestions

- Find own trade-off between area and speed
  - Broader memory interfaces (e.g., 32 bits) require more area...
  - Factor-n lane interleaving?
- Maybe more compact solutions that provide hashing capabilities, e.g., PRESENT, AES?
- Integration
  - External memory needed or is it already included in the system?
  - 8-bit AMBA APB interface available
- More "lightweight"? Change of KECCAK properties, e.g., collision resistance or security level (< 256 bits)
- Protection against implementation attacks, hiding (e.g., shuffling) or masking (e.g., secret sharing [1, 2])

Conclusions

- Serialized KECCAK[1600] requires $\approx 5.5 - 6$ kGEs
- Less than 15 μW at 1 MHz on 130 nm CMOS
- 8 vs. 16-bit version?
  - Spend 376 GEs for a 32 % speed improvement
  - No power differences

- KECCAK[800] preferred for RFIDs
  - 900 GEs smaller in size, i.e., 4.6 kGEs
  - With external memory available: only 2 016 GEs necessary
  - Twice as fast as KECCAK[1600]
  - 10.7 ms per block at 1 MHz
  - But almost no power savings

References I

G. Bertoni, J. Daemen, N. Debande, T.-H. Le, M. Peeters, and G. Van Assche.
Power Analysis of Hardware Implementations Protected with Secret Sharing.
Cryptology ePrint Archive: Report 2013/067, February 2013.

G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche.
Building Power Analysis Resistant Implementations of Keccak.
In *Second SHA-3 Candidate Conference, University of California, Santa Barbara, August 23-24, 2010*.

G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche.
The Keccak SHA-3 submission.
Submission to NIST (Round 3), 2011.

G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. V. Keer.
Keccak Implementation Overview, V3.2, 2012.

References II

M. Feldhofer and C. Rechberger.

A Case Against Currently Used Hash Functions in RFID Protocols.

In *Workshop on Information Security - IS, Montpellier, France, 2006.*

B. Jungk and J. Apfelbeck.

Area-Efficient FPGA Implementations of the SHA-3 Finalists.

In *Reconfigurable Computing and FPGAs–ReConFig 2011, International Conference, November 30-December 2, Cancun, Mexico, 2011*, pages 235–241, 2011.

E. B. Kavun and T. Yalcin.

A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications.

In S. B. O. Yalcin, editor, *Workshop on RFID Security – RFIDsec 2010, 6th Workshop, Istanbul, Turkey, June 7-9, 2010, Proceedings*, volume 6370, pages 258–269. Springer, 2010.

References III

M. Kim, J. Ryou, and S. Jun.

Efficient Hardware Architecture of SHA-256 Algorithm for Trusted Mobile Computing.

In *Information Security and Cryptology–Inscrypt 2008, 4th International Conference, Beijing, China, December 14-17, 2008, Revised Selected Papers*.

M. O'Neill.

Low-Cost SHA-1 Hash Function Architecture for RFID Tags.

In S. Dominikus, editor, *Workshop on RFID Security 2008 (RFIDsec08)*, pages 41–51, July 2008.

S. A. Weis, S. E. Sarma, R. L. Rivest, and D. W. Engels.

Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems.

In *Security in Pervasive Computing, 1st Annual Conference on Security in Pervasive Computing, Boppard, Germany, March 12-14, 2003, Revised Papers*.

Thanks for your attention!

Questions?

Michael Hutter
michael.hutter@iaik.tugraz.at
Graz University of Technology