Sabancı Üniversitesi

Efficient Hardware Implementations of High Throughput SHA-3 Candidates Keccak, Luffa and Blue Midnight Wish for Single- and Multi-Message Hashing

by

A. Akın, A. Aysu, O. C. Ulusel and E. Savaş

Introduction / Motivation

- In November 2007 NIST announced that it would organize the SHA-3 competition to select a new cryptographic hash function family by 2012.
- In the selection process, hardware performances of the candidates will play an important role.
- Single Message Hashing (SMH) assumes processing a single message stream at a time. Pipelined architectures are used to increase the performance for Multi-Message Hashing (MMH) [1], where the hardware processes more than one message concurrently.
- Hardware implementations of SHA-3 candidates that exploit MMH to achieve higher performance calls for a more thorough study of the issue.

Introduction / Motivation

- SMH design methodology usually favors single stage for each round, however in the case of MMH, the hardware can be fully pipelined and pipeline stages can be utilized.
- In MMH, hardware duplication and pipelining must be thought together
- In this paper, we present hardware evaluations of the candidate algorithms for MMH and SMH.

Candidate Selection

- The implementation results of [2] are summarized in the second column
- Other columns feature normalizations with respect to clock frequency or/and GE

Algorithm	Throughput (Gbit/s)	Throughput Normalized to 100 MHz	Throughput Normalized to 100 GE	Throughput Normalized to 100 MHz and 100 GE
BLAKE	3.97	2.33	8.70	5.10
BMW	5.36	51.22	3.16	30.18
CubeHash	4.67	3.20	7.92	5.44
ECHO	2.25	1.58	1.59	1.12
Fugue	4.09	1.60	8.85	3.46
Grøstl	6.29	2.33	10.77	3.99
Hamsi	5.57	3.20	9.49	5.46
JH	4.99	1.31	8.49	2.23
Keccak	21.23	4.35	37.70	7.73
Luffa	13.74	2.84	30.56	6.33
Shabal	3.28	1.02	5.98	1.87
SHAvite	3.15	1.38	5.49	2.41
SIMD	0.92	1.42	0.89	1.37
Skein	2.50	5.12	2.45	5.02

Why Keccak, Luffa and BMW?

- [2] targets high throughput architectures for SMH applications.
- To gain an insight on true potentials of the SHA-3 candidates in hardware implementation with the MMH in mind, we normalize the throughput results according to area or/and frequency values given in [2].
- Figures obtained through normalization *are in abstract level and not used as an objective in our design process,*
  - provide a deeper understanding and intuition to the multi-variable (e.g. area, clock frequency, and throughput) optimization problem for throughput comparison in MMH.
- Three candidates;
  - Keccak, Luffa and BMW to implement in SMH and MMH.

Keccak Algorithm

$Round[b](A, RC)$

θ STEP

$C[x] = A[x, 0] \oplus A[x, 1] \oplus A[x, 2] \oplus A[x, 3] \oplus A[x, 4]$ $\quad \forall x \text{ in } 0\ldots4$

$D[x] = C[x\text{-}1] \oplus ROT(C[x+1], 1)$ $\quad \forall x \text{ in } 0\ldots4$

$A[x, y] = A[x, y] \oplus D[x]$ $\quad \forall (x, y) \text{ in } (0\ldots4, 0\ldots4)$

ρ AND π STEPS

$B[y, 2x+3y] = ROT(A[x, y], r[x, y])$ $\quad \forall (x, y) \text{ in } (0\ldots4, 0\ldots4)$

χ STEP

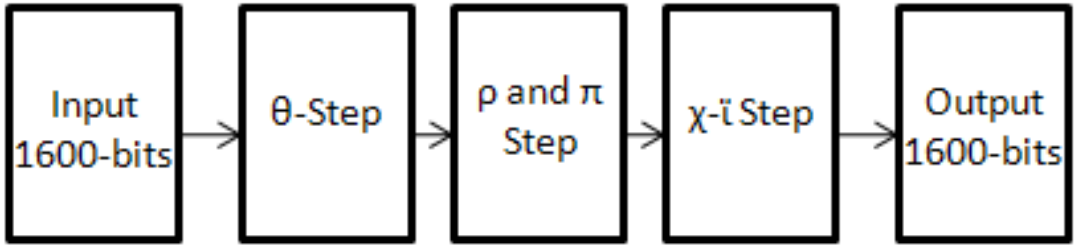
$A[x, y] = B[x, y] \oplus ((NOT\ B[x+1, y])\ AND\ B[x+2, y])$ $\quad \forall (x, y) \text{ in } (0\ldots4, 0\ldots4)$

ι STEP

$A[0, 0] = A[0, 0] \oplus RC$

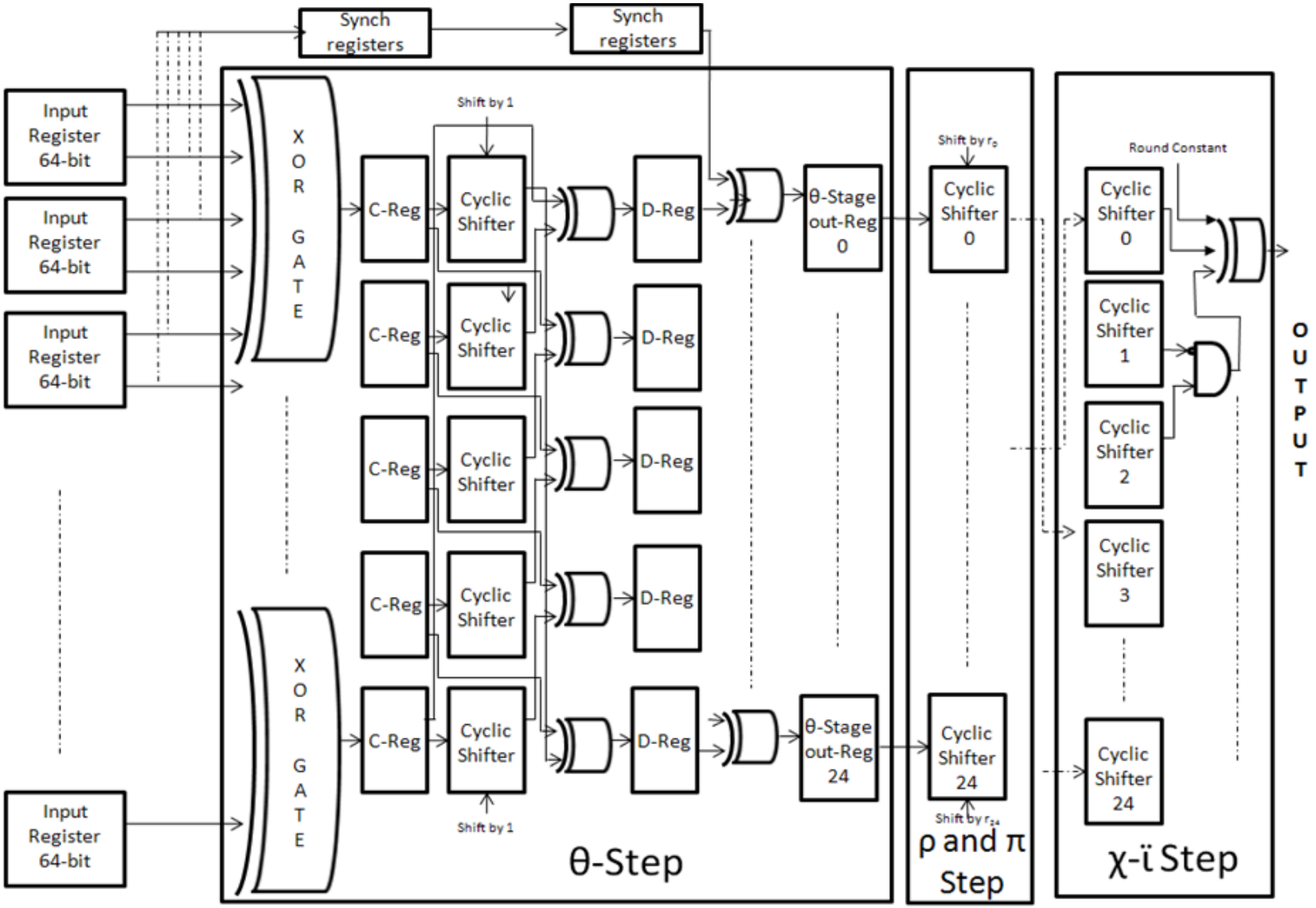
- Keccak algorithm [4] uses *KECCAK-f* permutation
  - 24 simple rounds
  - Each round consists of 5 steps
- The input and output of a round are 5×5 matrices of 64-bit words.
- The input message is divided into blocks of 1088-bit and *XOR*ed with a part of the state (which is initially zero and 1600-bit long) and the result is passed through a *KECCAK-f* permutation.

Keccak Hardware



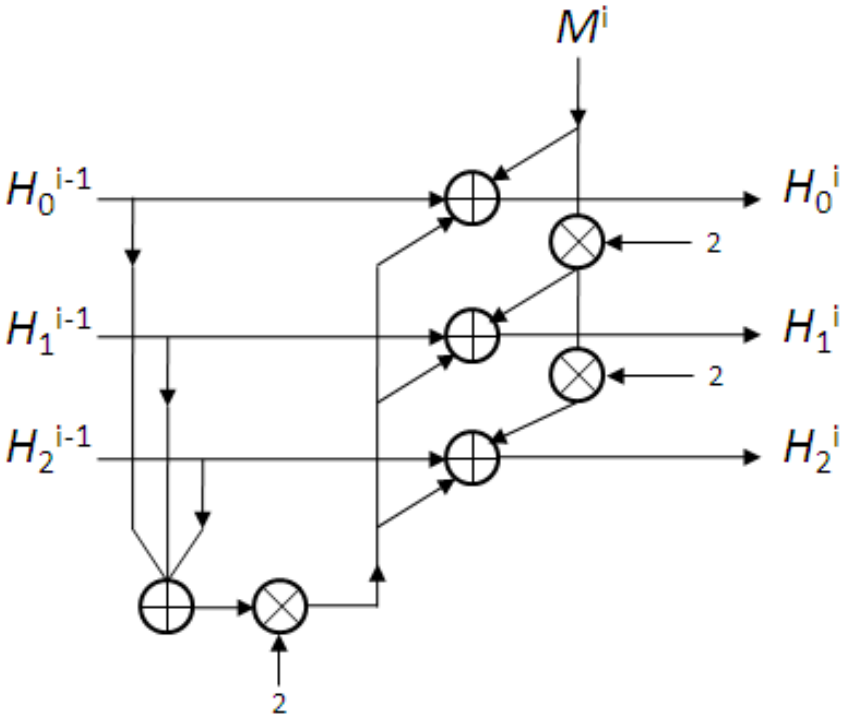
- One round consists of θ, ρ, π, χ, and *ï* steps
  - Three stage pipeline is used to implement one round
- First stage implements θ step.
- Second stage implements *ρ* and *π* steps .
- The final stage combines both *χ* and *ï* steps.
- Since new input arrives at each clock cycle in MMH, additional (synchronization) registers are required to forward the input to the later stages in addition to pipeline registers.
- SMH implementation,
  - no pipelining, one round is completed in one clock cycle, resulting in a lower operating frequency.

Keccak Hardware



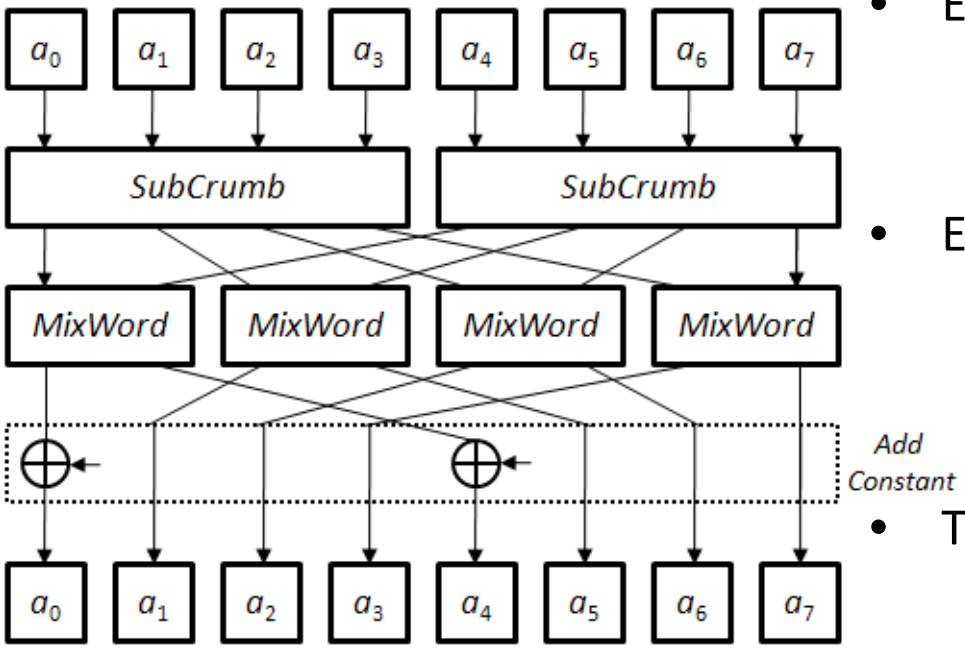
Luffa Algorithm

- Luffa's compression function is a round function made up of one Message Injection (*MI*) and one Permutation (*P*) stage.
- *MI* module combines the hash values of previous message blocks (i.e., H_0^{i-1}, H_1^{i-1}, H_2^{i-1}), with the current message block (M^i).
- Each of the inputs and ouputs are 256 bits.
- ⊗ represents a single multiplication in $GF(2^8)^{32}$ by constant 2.



Block diagram of the Message Injection (MI) module

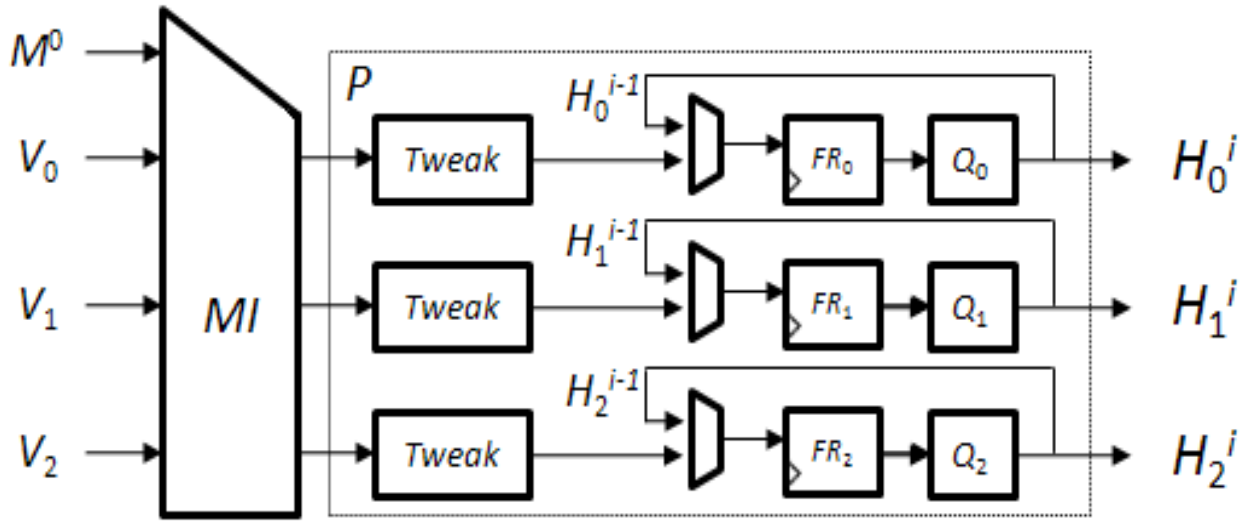
Luffa Algorithm



Block diagram of the *Step* module

- The permutation stage (*P*) for Luffa-256
  - 3 permute blocks for H_0^i, H_1^i, H_2^i.
- Each *Permute* block
  - starts with *tweak (permutation)*
  - iterates 8 rounds of the *Step* function.
- Each *Step* function
  - processes data in 32-bit words
  - Submodules: *SubCrumb*, *MixWord*, and *AddConstant*.
- The *SubCrumb*
  - nonlinear permutation by 32 identical s-boxes (4-bit input, 4-bit output).
- *MixWord*
  - Feistel ladder of 4 rounds, which is used to mix two words together.

Luffa Hardware

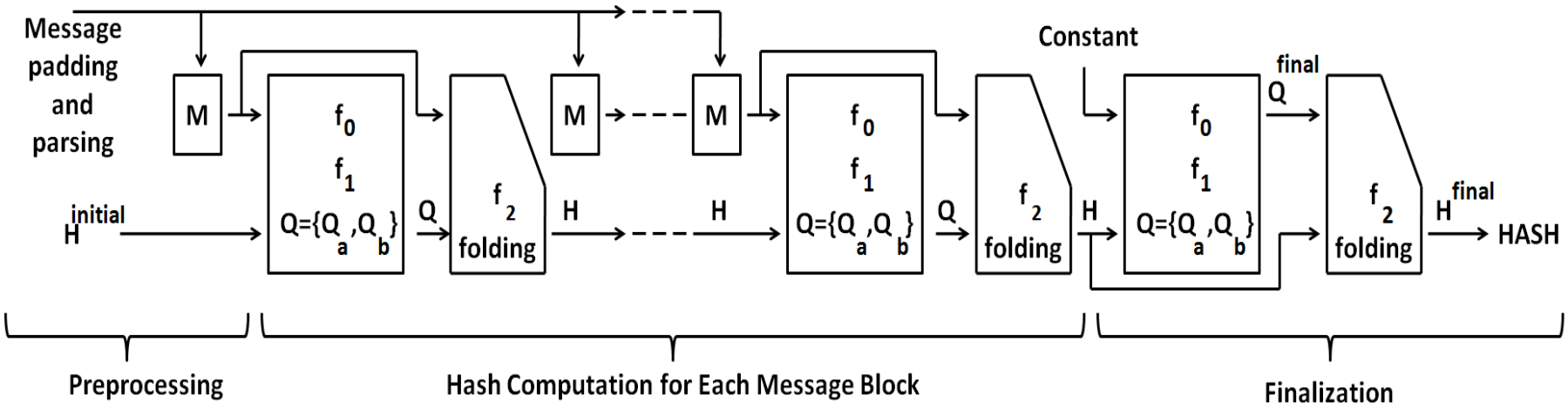


- For each message block, *MI* and *Tweak* are performed only once while the *Step* modules Q_0, Q_1, and Q_2 are used for eight consecutive rounds as shown.
- The critical path of *MI* module consists of 3 *XOR* gates while the critical path of permutation module consists of 5 *XOR* gates and a read operation from ROM for *S-Boxes*.

Luffa Hardware

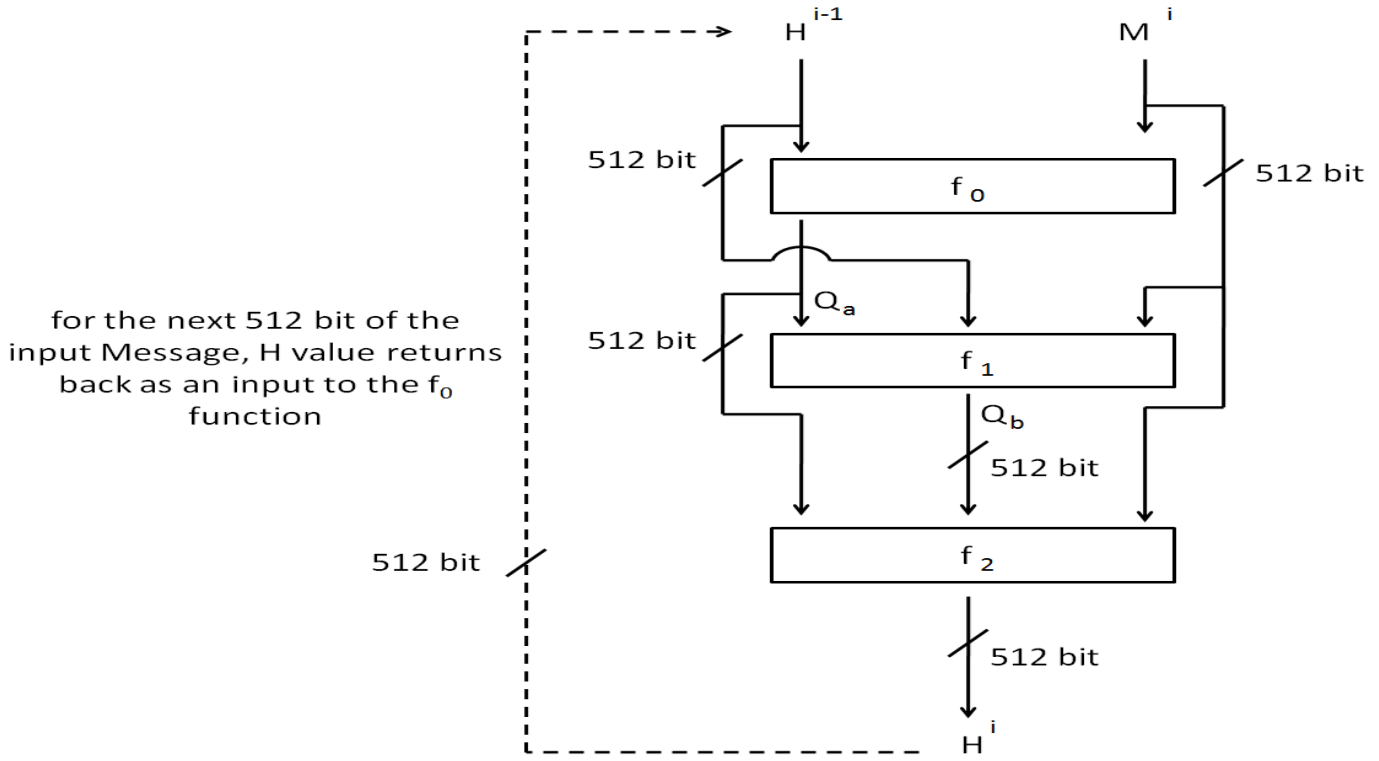
- Two hardware architectures:
  - SMH design:
    - a set of registers (FR_0, FR_1, FR_2) to forward the results of *Step* modules to the following round.
    - Placed between the *Tweak* and the *Step* to decrease the combination delay of a single round and increase the frequency at the cost of one extra cycle for message injection.
  - MMH design
    - It has already a small combinational delay,
    - we partition the *Step* function into two pipeline stages.
    - The first stage consists of a ROM and two *XOR* gates
    - The second stage has 3 *XOR* gates.

BMW algorithm



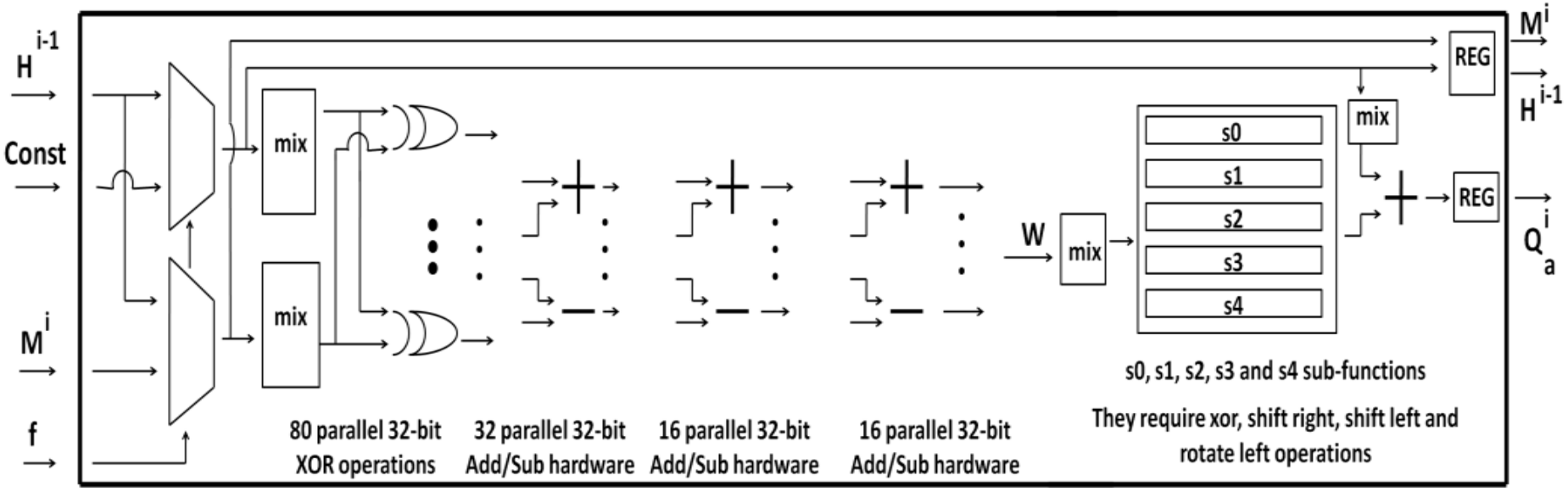
- Three steps: *preprocessing*, *hash computation* and *finalization*.
- The last two steps involve three functions f_0, f_1 and f_2.
- The function f_0 is used to compute Q_a
- The function f_1 is used with two sub-functions *expand1* and *expand2*.
- The function f_2 is used in folding (compression) part reducing 3m-bit of M, Q_a, and Q_b to m-bit new double pipe H.
- *Finalization* step is similar to the *hash computation*; but uses a constant value instead of message block to form m-bit H^{final}, which is truncated to n-bit hash value.

BMW Hardware



- BMW algorithm does not use multiple rounds for hashing of a single message block.
- the resulting value H^i is used as input in the processing of the next message block. Therefore, datapath of the BMW is longer than Luffa and Keccak.

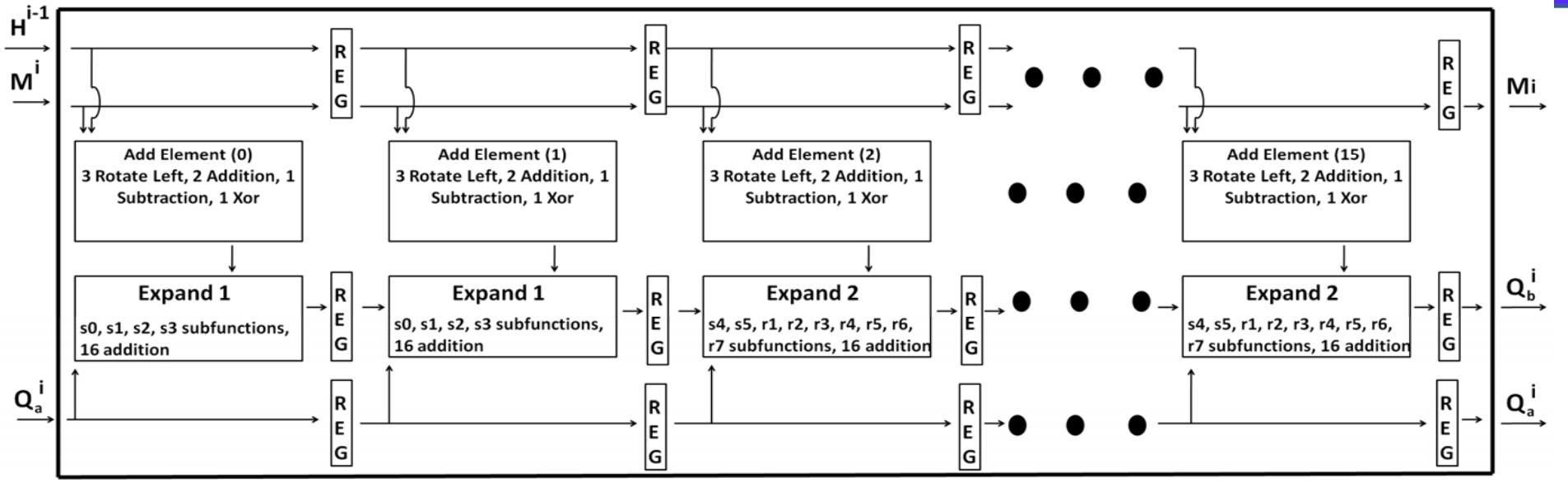
BMW Hardware



Block diagram of the f_0 module

- f_0 function of MMH implementation starts with mixing the
- bits of the previous hash block (H^{i-1}) and message block (M^{i}).
- Since the operations are not in the critical path, f_0 is implemented as a single pipeline stage.
- In SMH version of the architecture, the registers at the output of f_0 module are removed.

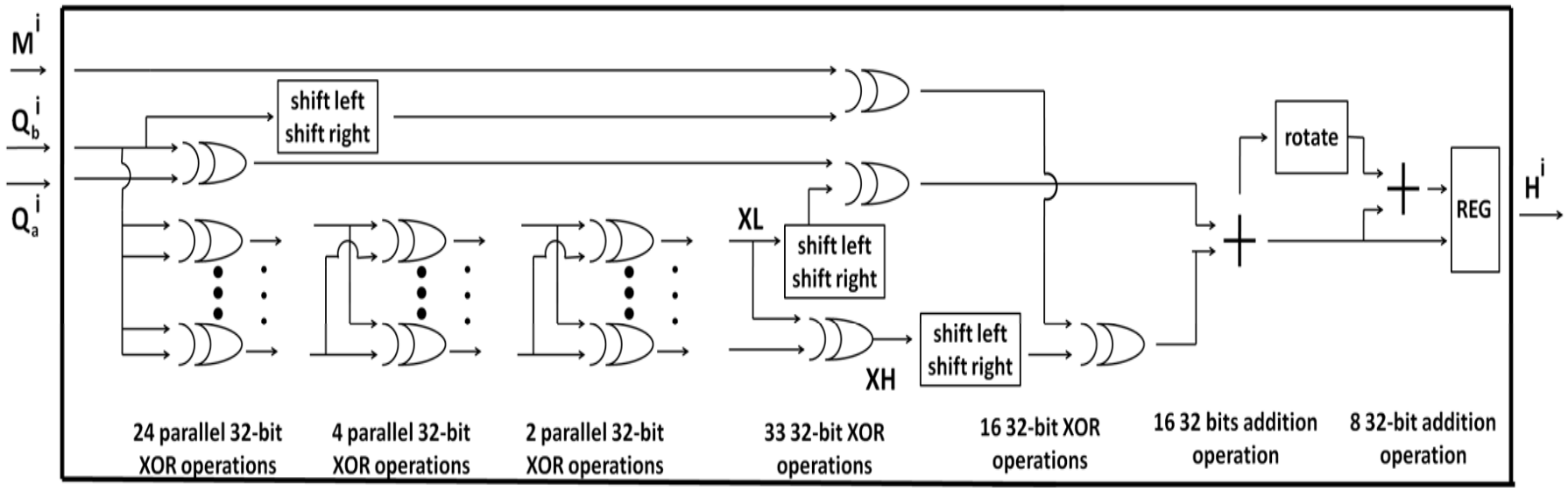
BMW Hardware



Block diagram of the f_1 module

- f_1 requires 2 *expand*1 and 14 *expand*2 sub-functions,
- Each *expand* function waits for the previous *expand* function to complete; this necessitates registering of 512-bit H^{i-1}, M^i and Q^i to forward them in the pipeline.
- MMH version, 16 pipeline stages are used to implement f_1
- for SMH hardware, all pipeline and synchronization registers are removed.

BMW Hardware



Block diagram of the f_2 module

- f_2 takes 512-bit M^i, $Q_a{}^i$ and $Q_b{}^i$ as inputs, and generates 512-bit H^i.
- It essentially compresses 1536-bit input to the 512 bits.
- In the finalization step, the leftmost 256 bits of the H^{final} forms the hash of the message.
- f_2 is implemented as a single pipeline stage and the same for both SMH and MMH.

Implementation Results

Algorithm	Hashing Method	Target Technology	Frequency (MHz)	# of Rounds	Latency (cycles)	Area (Slices/GE)	Throughput (Gbits/s)	Efficiency = Throughput/ (Area×10^6)
Keccak	SMH	Virtex 4	142.9	24	25	2,024	6.07	3.00
Keccak	MMH		508.7	24	121	4,356	22.33	5.13
Luffa	SMH		308.2	8	9	2,989	8.56	2.86
Luffa	MMH		470.8	8	17	3,719	13.85	3.72
BMW	SMH		9.01	1	1	10,486	4.51	0.43
BMW	MMH		115.96	1	18	12,497	57.98	4.64
Keccak	SMH	90nm ASIC	454.5	24	25	10.5k	19.32	1.84
Keccak	MMH		1,694.9	24	121	23.2k	74.41	3.21
Luffa	SMH		769.2	8	9	11.5k	21.37	1.86
Luffa	MMH		1,204.8	8	17	12.5k	35.44	2.83
BMW	SMH		52.63	1	1	55.9k	26.32	0.47
BMW	MMH		265.96	1	18	160.1k	132.98	0.83

- For SMH,
  - Luffa has the highest throughput on FPGA and BMW has highest throughput on ASIC.
  - However, efficiency of Keccak and Luffa are similar and better than BMW for both FPGA and ASIC.
- For MMH, BMW has the highest throughput on FPGA and ASIC, however Keccak has the highest efficiency.

Implementation Results

	Algorithm	Hashing Method	Target Technology	Frequency (MHz)	# of Rounds	Latency (cycles)	Area (Slices/GE)	Throughput (Gbits/s)	Efficiency = Throughput/ (Area×10^6)
ours	Keccak	SMH	90nm ASIC	454.5	24	25	10.5k	19.32	1.84
	Keccak	MMH		1,694.9	24	121	23.2k	74.41	3.21
	Luffa	SMH		769.2	8	9	11.5k	21.37	1.86
	Luffa	MMH		1,204.8	8	17	12.5k	35.44	2.83
	BMW	SMH		52.63	1	1	55.9k	26.32	0.47
	BMW	MMH		265.96	1	18	160.1k	132.98	0.83
[2]	Keccak	SMH	180nm ASIC	487.80	24	25	56.31k	21.23	0.38
	Luffa	SMH	180nm ASIC	483.09	8	9	44.9k	13.74	0.31
	BMW	SMH	180nm ASIC	10.46	1	1	169k	5.358	0.03
[1]	Luffa	SMH	130nm ASIC	1124	8	9	30.8k	31.9	1.07
	Luffa	MMH	130nm ASIC	508	1	9	156.6k	115.5	0.74
[3]	BMW	SMH	90nm ASIC	52.08	1	1	60.0k	26.66	0.54
	Luffa	SMH	90nm ASIC	100.4	1	1	68.9k	25.70	0.70

- Our Luffa SMH variant is superior to the architecture in [3], implemented using a comparable ASIC technology, both in terms of area and efficiency.
- Our MMH variant of Luffa provides a much better alternative to the only MMH implementation in literature [1], thanks to its high efficiency metric (*cf*. 2.83 and 0.74).
- Our BMW implementation of SMH variant along with the design in [3] is the fastest in ASIC realization;
- Keccak has the highest efficiency when it is compared with previous architectures.

Conclusion

- We presented efficient and high throughput hardware implementations of SHA-3 candidates Keccak, Luffa and Blue Midnight Wish with an emphasis on MMH applications.
- We employed pipelining, parallelism and re-timing techniques to improve the performances and efficiencies of our designs.
- To the best of our knowledge, this is the first work that provides a comparative analysis of three high performance SHA-3 candidates for MMH applications.

References

- [1] M. Knezevic, I. Verbauwhede: Hardware Evaluation of the Luffa Hash Family. *4th Workshop on Embedded Systems Security 2009*, Grenoble, France.
- [2] S. Tillich et al: High-Speed Hardware Implementations of BLAKE, Blue Midnight, Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD and Skein. Cryptology ePrint Archive. October 2009.
- [3] A. H. Namin and M. A. Hasan, “Hardware Implementation of the Compression Function for Selected SHA-3 Candidates”.