

# Lightweight Implementations of SHA-3 Finalists on FPGAs

**Jens-Peter Kaps**   Panasayya Yalla

Kishore Kumar Surapathi   Bilal Habib   Susheel Vadlamudi  
Smriti Gurung

Cryptographic Engineering Research Group (CERG)

<http://cryptography.gmu.edu>

Department of ECE, Volgenau School of Engineering,  
George Mason University, Fairfax, VA, USA

The Third SHA-3 Candidate Conference

# Outline

- 1 Introduction
- 2 Methodology
- 3 Results

# Lightweight Team



Panasayya Yalla

Ph.D. Student

SHA-2



Bilal Habib

Ph.D. Student

Skein



Smriti Gurung

Masters Student

Keccak



Susheel Vadlamudi

Masters Student

BLAKE-256, JH42



Kishore Surapathi

Masters Student

Grøstl

## Previous Work on SHA-3 Candidates

- Kerckhof et al. [CARDIS 2011]
  - Compact FPGA implementations of finalists.
  - 256-bit hash only on Virtex-6, 64-bit I/O.
  - Fastest algorithm is 7.8 times faster while 2.2 times larger than slowest.
- Jungk and Apfelbeck [ReConFig 2011]
  - Area-efficient implementation of finalists.
  - Designed for Virtex-5, also results for Spartan-3, 32-bit I/O.
  - Most implementations occupy similar area.

## Previous Work on SHA-3 Candidates

- Kerckhof et al. [CARDIS 2011]
  - Compact FPGA implementations of finalists.
  - 256-bit hash only on Virtex-6, 64-bit I/O.
  - Fastest algorithm is 7.8 times faster while 2.2 times larger than slowest.
- Jungk and Apfelbeck [ReConFig 2011]
  - Area-efficient implementation of finalists.
  - Designed for Virtex-5, also results for Spartan-3, 32-bit I/O.
  - Most implementations occupy similar area.

### Problem: Rating algorithm performance when

- Implementations are on different devices,
- made with different implementation goals and features,
- vary in both: area and throughput, and
- support different I/O interface widths.

# Motivation

## Assumption

Keeping either area or throughput constant makes fair comparison of lightweight implementations possible.

- Implementing for minimum area alone can lead to unrealistic run-times.
- $\Rightarrow$  Target: Achieve the maximum Throughput/Area ratio for a given area budget.
- Realistic scenario:
  - System on Chip: Certain area only available.
  - Standalone: Smaller Chip, lower cost, but limit to smallest chip available, e.g. 768 slices on smallest Spartan 3 FPGA.

# Our Goals

- All optimized for the same target:
  - Xilinx Spartan 3, low cost FPGA family
  - Maximum Throughput to Area ratio for given area budget.
  - Implemented 256 bit digest versions only
- All use the same standardized interface.
- Implemented on several FPGA families for fair comparison with other reported results.

## Target 1:

- Budget: 400-600 slices, 1 Block RAM (BRAM)

## Target 2:

- Budget: 768 slices, 0 Block RAM (BRAM)

# Our Goals

- All optimized for the same target:
  - Xilinx Spartan 3, low cost FPGA family
  - Maximum Throughput to Area ratio for given area budget.
  - Implemented 256 bit digest versions only
- All use the same standardized interface.
- Implemented on several FPGA families for fair comparison with other reported results.

## Target 1:

- Budget: 400-600 slices, 1 Block RAM (BRAM)

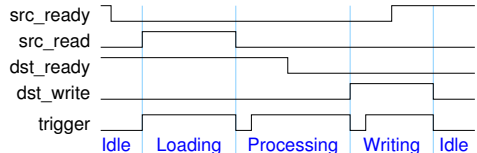
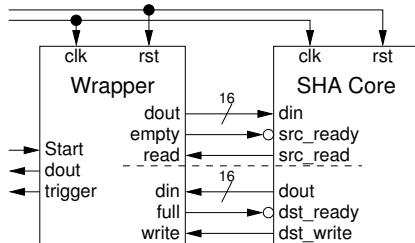
## Target 2:

- Budget: 768 slices, 0 Block RAM (BRAM)

- Additionally: **Measuring of Power Consumption**

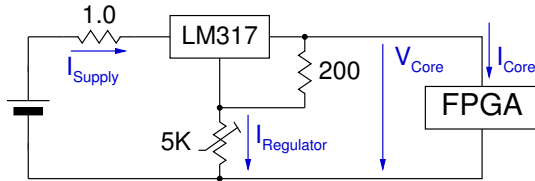


# Power Measurement Setup



- Separation of driver circuit from SHA-Core.
- SHA-Core state inferred through monitoring FIFO signals.
- Driver on Nexys2, SHA-Core on Spartan-3E Starter Kit.

# Power Measurement Circuit



- $P_{Total} = P_{Static} + P_{Dynamic}$
- 3 supply voltages  $V_{Core}$ ,  $V_{I/O}$ , and  $V_{Auxiliary}$ .
- We measure only  $I_{Core}$  w/o clock to obtain  $P_{Static}$  and at 50 MHz for  $P_{Total}$ .
- Measure  $I_{Core}$  average for Loading, Processing, and Writing stages.

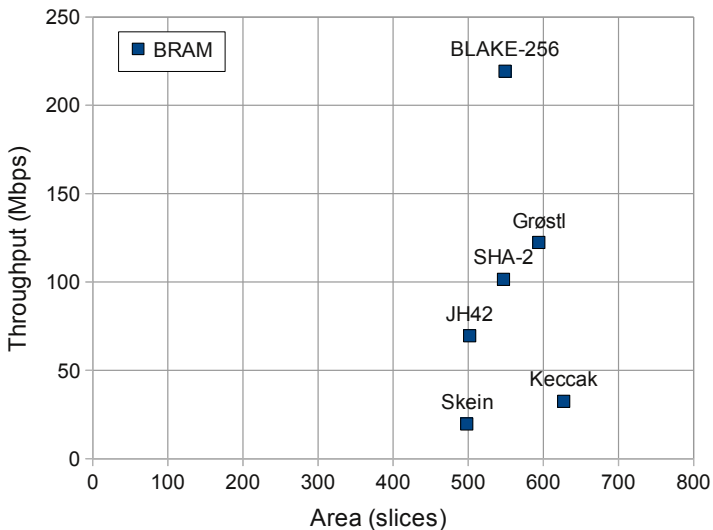
# Implementation Summary of Finalists for Long Messages

	Algorithm	Block Size (bits) $b$	Clock Cycles to hash $N$ blocks $clk =$		Throughput $\frac{b}{(l + p) \cdot T}$
			$st + ($	$l + p) \cdot N + end$	
BRAM	BLAKE-256	512	$2 + ($	$32 + 258) \cdot N + 17$	$512 / (290 \cdot T)$
	Grøstl	512	$2 + ($	$32 + 515) \cdot N + 532$	$512 / (547 \cdot T)$
	JH42	512	$2 + ($	$64 + 737) \cdot N + 33$	$512 / (801 \cdot T)$
	Keccak	1088	$2 + ($	$68 + 3696) \cdot N + 17$	$1088 / (3764 \cdot T)$
	Skein	512	$5 + ($	$32 + 2407) \cdot N + 2362$	$512 / (2439 \cdot T)$
	SHA-2	512	$2 + ($	$32 + 563) \cdot N + 17$	$512 / (595 \cdot T)$
Logic only	BLAKE-256	512	$2 + ($	$32 + 258) \cdot N + 17$	$512 / (290 \cdot T)$
	Grøstl	512	$2 + ($	$32 + 357) \cdot N + 374$	$512 / (389 \cdot T)$
	JH42	512	$2 + ($	$64 + 736) \cdot N + 32$	$512 / (800 \cdot T)$
	Keccak	1088	$2 + ($	$68 + 2328) \cdot N + 17$	$1088 / (2396 \cdot T)$
	Skein	512	$5 + ($	$32 + 2366) \cdot N + 2319$	$512 / (2398 \cdot T)$
	SHA-2	512	$2 + ($	$32 + 404) \cdot N + 17$	$512 / (436 \cdot T)$

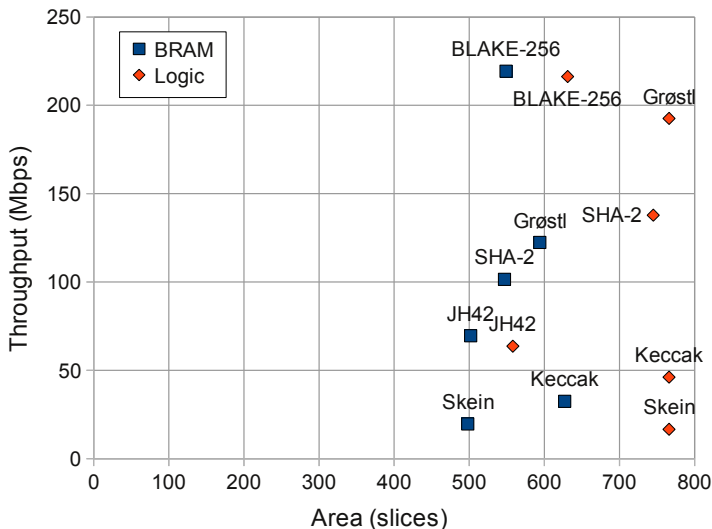
# Implementation Results of Finalists on Xilinx Spartan-3

Algorithm	Area (slices)	Block RAMs	Maximum Delay (ns) $T$	Long Messages		Short Messages	
				Throughput (Mbps)	TP/Area (Mbps/slices)	Throughput (Mbps)	TP/Area (Mbps/slices)
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
BLAKE-256	631	0	8.16	216.3	0.34	203.6	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

# Throughput versus Area on Spartan-3



# Throughput versus Area on Spartan-3



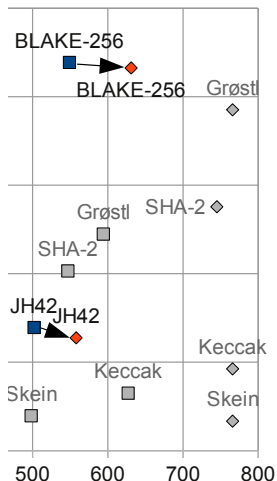
# BRAM vs. Logic only implementations

## BLAKE-256

- Replaced BRAM with two Distributed RAMs.
- Same number of clock cycles.

## JH

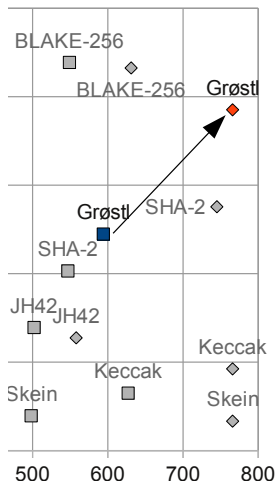
- Replaced BRAM with two Distributed RAMs.
- Not sufficient space for additional optimizations.



# BRAM vs. Logic only implementations

## Grøstl

- BRAM stores only intermediate hash and IV.
- Removing BRAM increases size only marginally.
- Additional area allows implementing full GF multiplier.
- ⇒ One new column every 2 clock cycles (earlier 3).

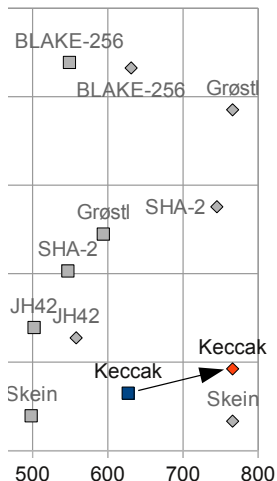




# BRAM vs. Logic only implementations

## Keccak

- Fixed rotations turn into variable rotator for small datapaths.
- Replaced BRAM with four single port Distributed RAMs.
- Decoupling of  $\theta$  and  $\rho$  &  $\pi$  through additional register and Distributed RAM eliminates address contention.
- $\Rightarrow$  Eliminates dedicated write cycles.



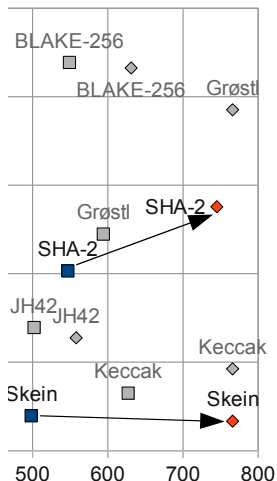
# BRAM vs. Logic only implementations

## Skein

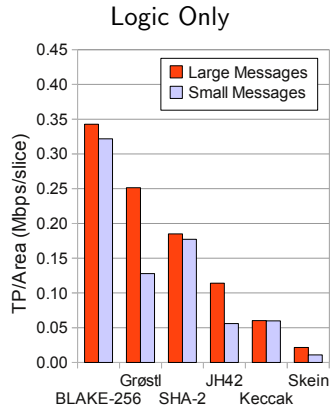
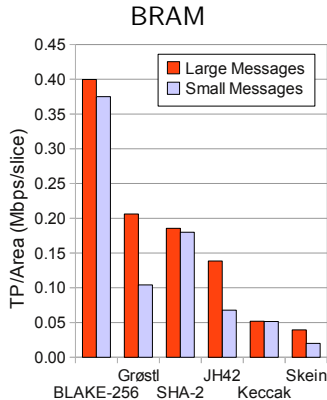
- BRAM circuit uses 32-bit adder to reduce critical path.
- Barrel shifter is largest block (192 slices).
- Logic only uses 64-bit datapath as dualport Distributed RAM is too big (162  $\rightarrow$  356 slices)

## SHA-2

- Logic only version uses registers for state variables  $\rightarrow$  reduces number of clock cycles per round.

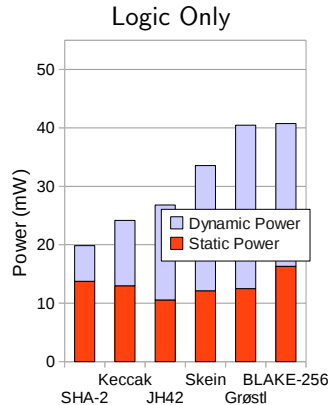
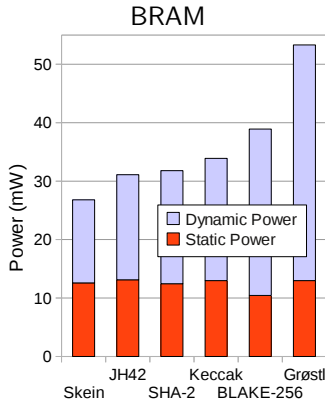


# Ranking by Throughput over Area on Spartan-3

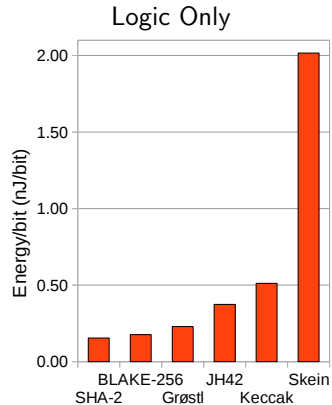
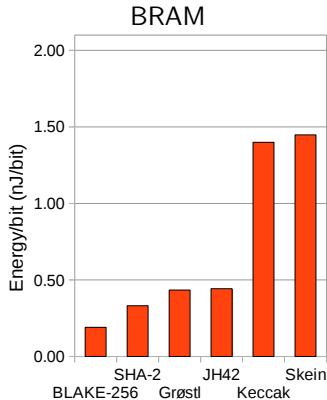


Algorithms with finalization rounds perform worse for short messages.

# Ranking by Power Consumption on Spartan-3E



# Ranking by Relative Energy Consumption on Spartan-3E



# Spartan-3 Shoot Out

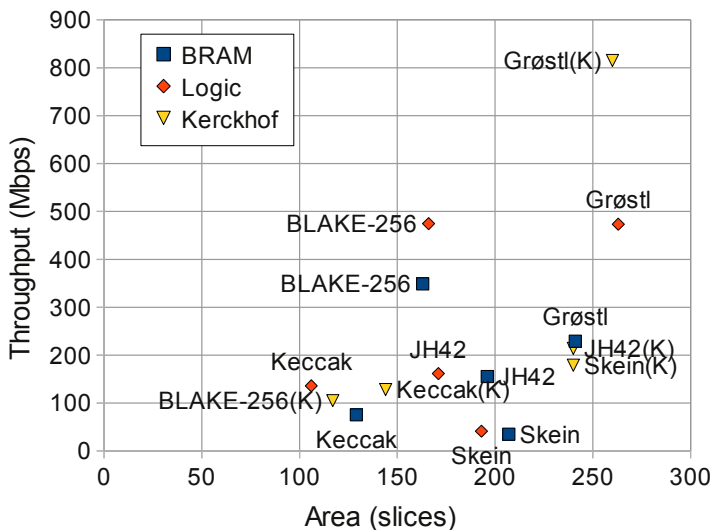
Algorithm	BRAM				Logic Only			
	TP/A		Power	Energy /bit	TP/A		Power	Energy /bit
	Large	Small			Large	Small		
BLAKE-256	2.15	2.08	1.22	0.53	1.85	1.81	4.00	1.15
Grøstl	1.11	0.58	1.68	1.31	1.36	0.72	4.58	1.49
JH42	0.75	0.38	0.98	1.33	0.62	0.22	2.66	2.43
Keccak	0.28	0.29	1.07	4.21	0.33	0.34	1.83	3.32
Skein	0.21	0.11	0.84	4.36	0.12	0.06	3.51	13.08
SHA-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

# Implementation Shoot Out

Algorithm		Xilinx										Altera	
		Spartan-3		Spartan-6		Aritx-7		Virtex-5		Virtex-6		Cyclone-II	
		L	S	L	S	L	S	L	S	L	S	L	S
BRAM	BLAKE-256	2.15	2.08	1.99	1.93	1.77	1.71	1.80	1.74	2.06	2.00	2.06	1.99
	Grøstl	1.11	0.58	0.69	0.36	0.70	0.36	1.02	0.54	0.92	0.48	1.95	1.02
	JH42	0.75	0.38	0.54	0.27	0.66	0.33	1.01	0.51	0.76	0.39	1.06	0.54
	Keccak	0.28	0.29	0.43	0.44	0.33	0.34	0.49	0.50	0.56	0.58	0.84	0.86
	Skein	0.21	0.11	0.15	0.08	0.12	0.06	0.18	0.10	0.16	0.09	0.36	0.19
	SHA-2	1.00	1.00	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	2.23	2.19	1.86	1.82	2.24	2.19	3.76	3.69
	Grøstl	1.36	0.72	1.43	0.76	1.26	0.67	1.50	0.80	1.41	0.75	1.93	1.02
	JH42	0.62	0.32	0.74	0.38	0.75	0.39	0.99	0.50	0.74	0.38	0.37	0.19
	Keccak	0.33	0.34	0.90	0.93	0.49	0.51	0.48	0.50	1.00	1.04	0.27	0.28
	Skein	0.12	0.06	0.14	0.08	0.14	0.07	0.17	0.09	0.17	0.09	0.07	0.04
	SHA-2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

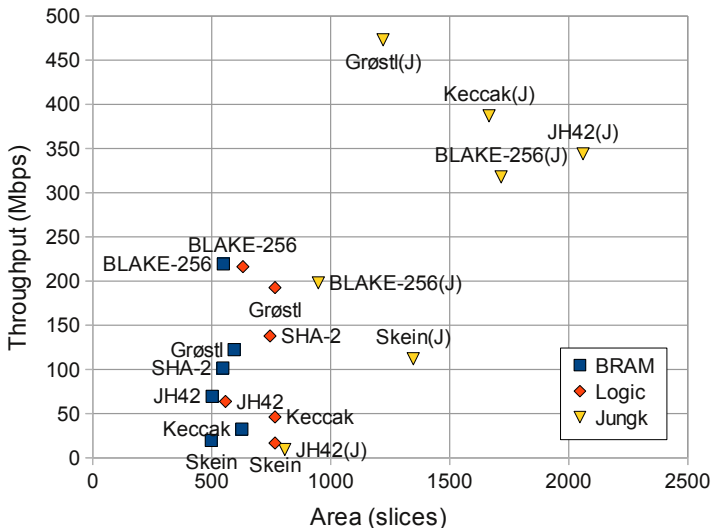
L: Large Message, S: Small Message

# Comparison with [Kerckhof] Results (Virtex 6)

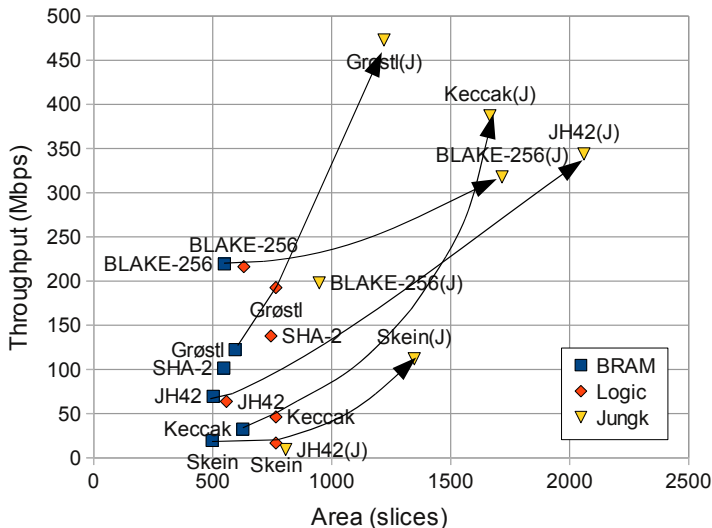




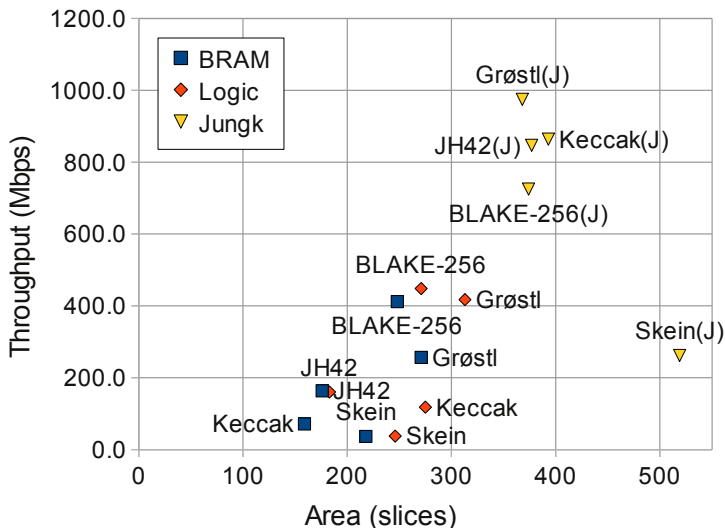
# Comparison with [Jungk] Results (Spartan-3)



# Comparison with [Jungk] Results (Spartan-3)



## Comparison with [Jungk] Results (Virtex 5)



Thanks for your attention.

# Backup Slides

# Interface and Protocol

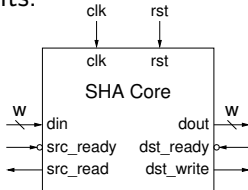
Based on Interface and I/O Protocol from Gaj et al.[CHES 2010].

- msg\_len\_ap, seq\_len\_ap (after padding ) in 32-bit words.
- msg\_len\_bp, seq\_len\_bp (before padding) in bits.

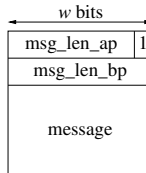
$$msg\_len\_bp = \sum_{i=0}^{n-2} seq\_len\_ap_i \cdot 32 + seq\_len\_bp_{n-1}$$

$$msg\_len\_ap = \sum_{i=0}^{n-1} seq\_len\_ap_i \cdot 32$$

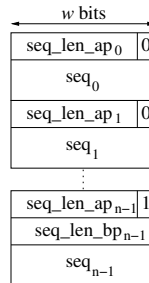
$w = 16$  bits.



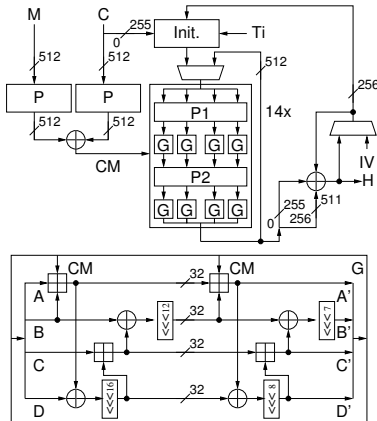
a)SHA Interface



b)SHA Protocol



# BLAKE-256 Algorithm

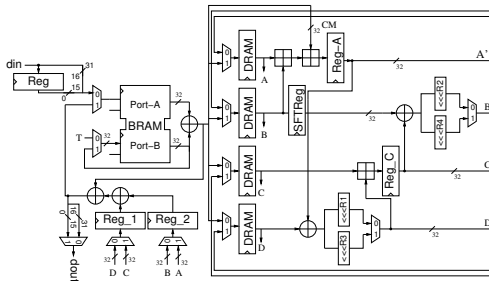


## Memory Requirements: 2,176 bits

- State: 512 bits
- IV: 256 bits
- Intermediate Hash: 256 bits
- Message: 512 bits
- Constants: 512 bits
- Salt: 128 bits all 0

- Blake scales very well.
- Folded up to 4 times vertically and 4 times horizontally.

# BLAKE-256 Implementation BRAM



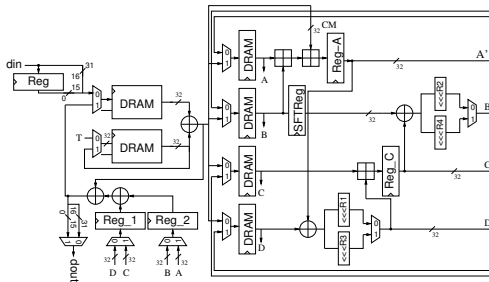
## Performance (Clock Cycles)

- Initialization: 2
- Loading: 32
- Block Initialization: 16
- $8 \times G$ :  $14 \times 16 + 2 = 226$
- Block Finalization: 16
- Total per Block: 258

- Implemented quasi-pipelined  $1/2$  G-function computing 2 in parallel.
- Permutation causes a large controller with 210 addresses.
- BRAM contains constants, message, IV, intermediate hash.
- **Scalability:** Unfolding leads to worse TP/A.



# BLAKE-256 Implementation Logic Only

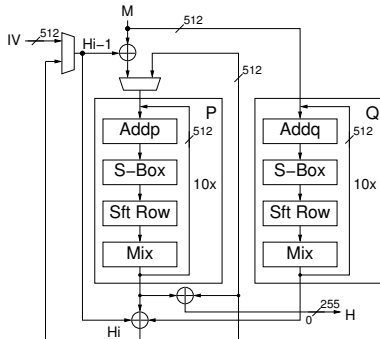


## Performance (Clock Cycles)

- Initialization: 2
- Loading: 32
- Block Initialization: 16
- $8 \times G$ :  $14 \times 16 + 2 = 226$
- Block Finalization: 16
- Total per Block: 258

- Replaced BRAM with two Distributed RAMs.
  - One stores IV, intermediate hash.
  - Other stores constants, salt, counter, chaining values.
- Therefore same number of clock cycles as BRAM version.

# Grøstl Algorithm

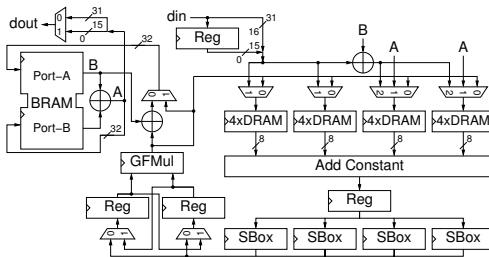


- Grøstl scales well, like AES.
- Folded up to 8 times vertical.
- Small storage requirements.
- Uses many narrow memory accesses in parallel (8 per column).

## Memory Requirements: 2,048 bits

- State: 1024 bits
- IV: 512 bits
- Intermediate Hash: 512 bits

# Grøstl Implementation BRAM

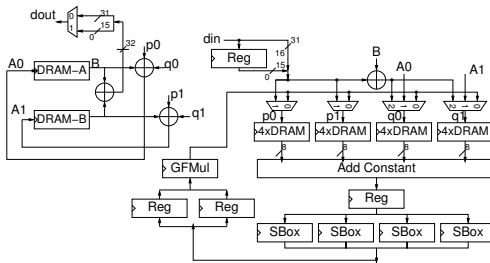


## Performance (Clock Cycles)

- Initialization: 2
- Loading: 32
- P&Q:  $10 \times 48 + 3 = 483$
- XORs: 32
- Total per Block: 515

- Finalization takes as many clock cycles as 1 block.
- BRAM stores only intermediate hash and IV.
- One new column every 3 clock cycles, P & Q interleaved.
- **Scalability:** Reducing number of clock cycles per column by adding S-Boxes and/or GF-Multiplier.

# Grøstl Implementation Logic Only

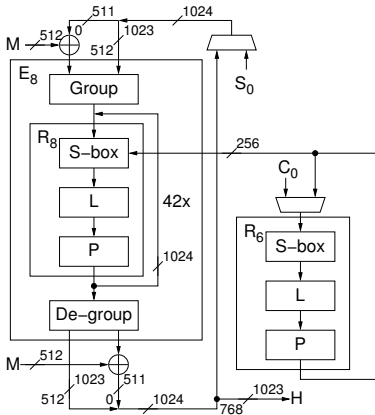


## Performance (Clock Cycles)

- Initialization: 2
- Loading: 32
- P&Q:  $10 \times 32 + 5 = 325$
- XORs: 32
- Total per Block: 357

- Removing BRAM increases size only marginally.
- The higher area budget allows implementing a full GF multiplier.
- $\Rightarrow$  One new column every 2 clock cycles.

# JH Algorithm



- Permutation  $P$ , grouping, and de-grouping makes scaling difficult.
- Folding increases size.

## Memory Requirements: 3,072 bits

- State: 1024 bits
- Precomputed  $S_0$ : 1024 bits
- Message: 512 bits
- Constants State: 256 bits
- Constants  $C_0$ : 256 bits



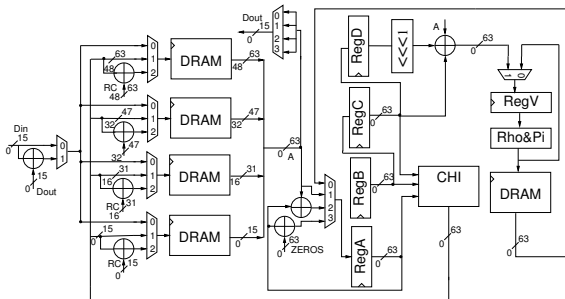








# Keccak Implementation Logic Only



## Performance (Clock Cycles)

- Initialization: 2
- Loading: 68
- $\theta, \rho, \pi$ :  $58 \times 24 = 1,392$
- $\chi, \iota$ :  $39 \times 24 = 936$
- Total per Block: 2,328

- Decoupling of  $\theta$  and  $\rho$  &  $\pi$  through additional register and Distributed RAM eliminates address contention.
- $\Rightarrow$  Eliminates dedicated write cycles.



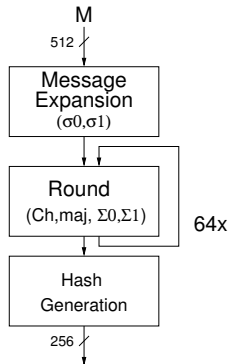
- Initialization: 5
- Loading: 32
- MIX:  $18 \times 4 \times 20 = 1440$
- Keygen:  
 $19 \times 45 + 3 = 858$
- Permutations: 109
- Total per Block: 2407

- Lightweight Implementations of SHA-3 on FPGAs 16 / 20

- Initialization: 5
- Loading: 32
- MIX:  $18 \times 4 \times 20 = 1440$
- Keygen:  
 $19 \times 47 + 1 = 894$
- Permutations: 32
- Total per Block: 2366

- Lightweight Implementations of SHA-3 on FPGAs 17 / 20

# SHA-2 Algorithm

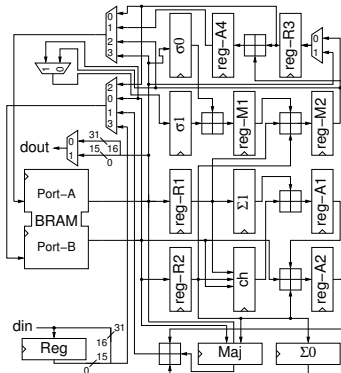


- Uses six logic functions  $Ch$ ,  $Maj$ ,  $\Sigma_0$ ,  $\Sigma_1$ ,  $\sigma_0$ , and  $\sigma_1$ .
- Algorithm cannot be folded.

## Memory Requirements: 3,072 bits

- State: 512 bits
- Constants:  $64 \times 32 = 2,048$  bits
- Hash values:  $2 \times 8 \times 32 = 512$  bits

# SHA-2 Implementation BRAM

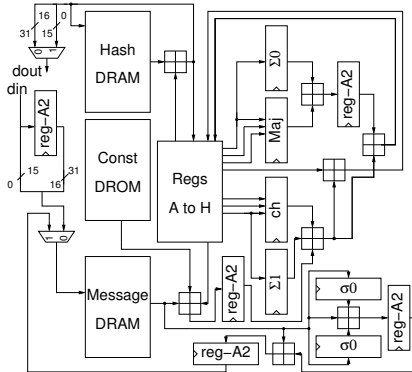


## Performance (Clock Cycles)

- Initialization: 2
- Loading: 32
- Message expansion=99
- Round:  $64 \times 7 = 448$
- Hash generation: 16
- Total per Block: 563

- BRAM contains constants, message, working variables and intermediate hash.
- Quasi-pipelined the round operation.

# SHA-2 Implementation Logic Only



## Performance (Clock Cycles)

- Initialization: 2
- Loading: 32
- Message expansion=196
- Round:  $64 \times 3 = 192$
- Hash generation: 16
- Total per Block: 404

- Constants, message and intermediate hash are stored in DRAMs and working variable in registers.
- Use of registers for working variable reduced the number of clock cycles per round.