Low-Area Implementations of SHA-3 Candidates

Jens-Peter Kaps

Cryptographic Engineering Research Group (CERG)
http://cryptography.gmu.edu
Department of ECE, Volgenau School of IT&E,
George Mason University, Fairfax, VA, USA

SHA-3 Project Review Meeting

Outline

Motivation

- There have been several comparison of Throughput/Area optimized implementations [Gaj],[Matsuo],[Baldwin],[Guo].
- Only few low-area implementations of single SHA-3 algorithms on FPGAs.
- Not all fully autonomous, Varying interface assumptions.
- Low-area implementations highlight flexibility of algorithm designs.

Motivation

- There have been several comparison of Throughput/Area optimized implementations [Gaj],[Matsuo],[Baldwin],[Guo].
- Only few low-area implementations of single SHA-3 algorithms on FPGAs.
- Not all fully autonomous, Varying interface assumptions.
- Low-area implementations highlight flexibility of algorithm designs.

Goal

- First comprehensive comparison of low-area implementations of Round 2 SHA-3 Candidates.
- All use the same standardized interface.
- All optimized for the same parameters under the same assumptions.

Assumptions

- Implementing for minimum area alone can lead to unrealistic run-times.
- $\Rightarrow$ Goal: 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.

Assumptions

- Implementing for minimum area alone can lead to unrealistic run-times.
- $\Rightarrow$ Goal: 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.

Target

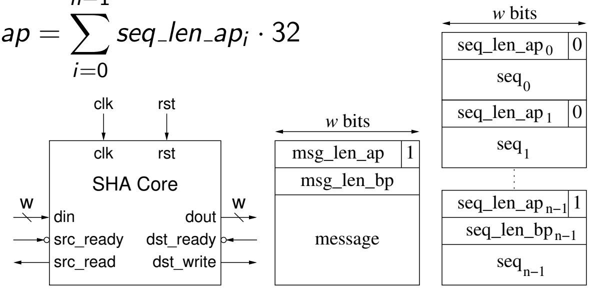
- Xilinx Spartan 3e, low cost FPGA family
- Budget: 500 slices, 1 Block RAM (BRAM)

Interface

- Based on Interface and I/O Protocol from [Gaj], w=16.
- msg_len_ap, seq_len_ap (after padding) in 32-bit words.
- msg_len_bp, seq_len_bp (before padding) in bits.

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

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

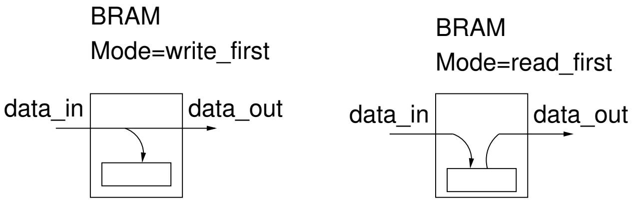


a)SHA Interface b)SHA Protocol

Minimization Technique

- Datapath
  - Use BRAM to store state, initialization vectors, constants
  - Use BRAM in each clock cycle
  - Avoid temporary storage or use:
    - Free registers, i.e. unused flip-flops after LUTs
    - Shift Registers (1x16 bit / Distributed RAM (1x16 bit) $\Rightarrow$ 1 LUT $= \frac{1}{2}$ Slice
- Control Logic
  - Small main state machine, up-to 8 states
  - Counter for clock cycles in longest state
  - Stored Program Control within states
  - BRAM addressing must follow regular sequence, can have offset between rounds

Block RAM



- We use exclusively *read_first* mode, i.e. old value is read, new value is written.
- Saves clock cycles, however, leads to address offset.
- Control logic might become difficult.

Limits

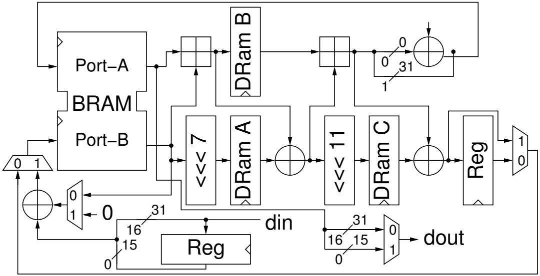
- Maximum 2 input and 2 output ports, 2 addresses (dual port).
- Maximum single port w/ 64 bits or dual port w/ 32 bits each.

BLAKE

	1	2	3	4	5	6	7	8	9	10	11	12
1	G1			G2		G1	G2					
2		G1			G2		G1	G2				
3			G1			G2		G1	G2			
4				G1			G2		G1			
A	G1	G1	G1	G2	G2	G1	G2	G2	G1	G1	G2	G2
B				G1		G2	G1	G1	G1	G1	G2	G2

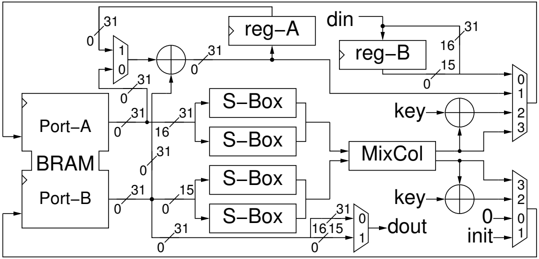
- Smallest implementation would be $\frac{1}{2}$ G-function $\rightarrow$ BRAM contention.
- Best result: 2 G-functions, pipelined as shown above. Keeps data in-flight longer, eliminates BRAM contention
- However, generation of addresses difficult $\Rightarrow$ Large control logic.

CubeHash



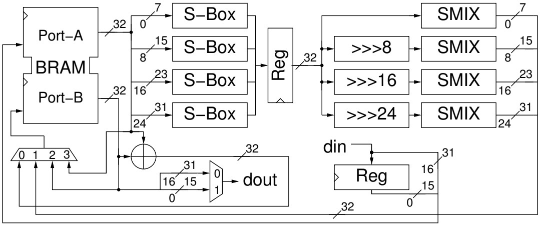
- Initialization vector pre-processed stored in BRAM.
- BRAM contention requires swapping data between BRAM and Distributed RAM (DRAM).
  - This requires additional clock cycles.
  - Leads to simpler control logic.
- Finalization very costly at 9,296 clock cycles.

ECHO



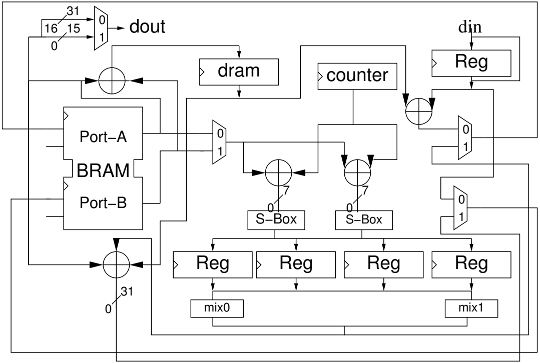
- Mix-Columns contains 2 Mix-Column units with total 8x8 bit register.
- 4 logic based S-Boxes with integrated pipeline stage.
- Faster Mix-Columns would exceed 500 slices.
- Key Generation uses DRAM, 32 bit adder. Allows store salt.
- Small control unit with room for improvement.

Fugue



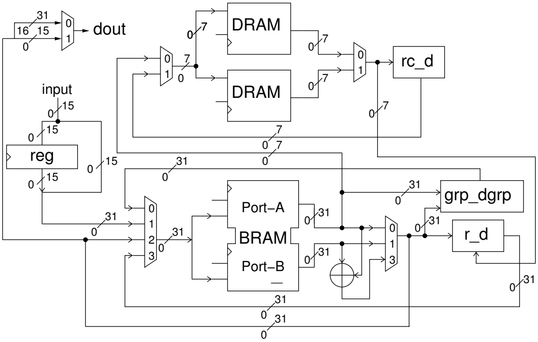
- 4 logic based S-Boxes followed by pipeline stage.
- Only fixed rotations.
- Most time consuming function: SMIX.
- Challenge to store the S-Mix table.

Grøstl



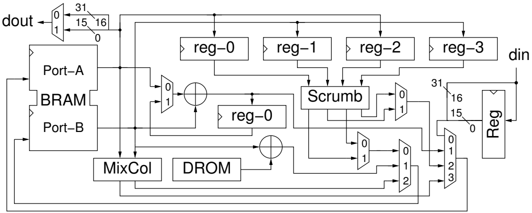
- Serialized P and Q.
- Only 2 S-Boxes due to Shift Rows → can only use 2x8 bits.
- Uses single set of registers for two Mix Column units.
- 16 / 32 bit datapath → 7 clock cycles per column.

JH



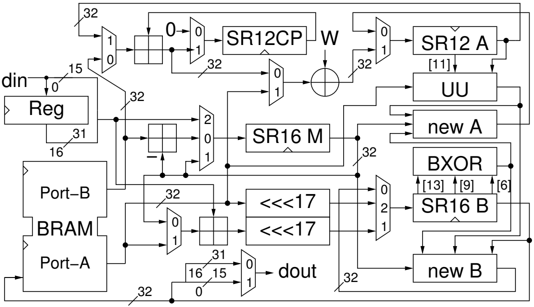
- Uses BRAM and two 32x8 DRAMs.
- Grouping and de-grouping are the most expensive operations.
  - 160 clock cycle operation.
  - multiple narrow memory accesses.

Luffa



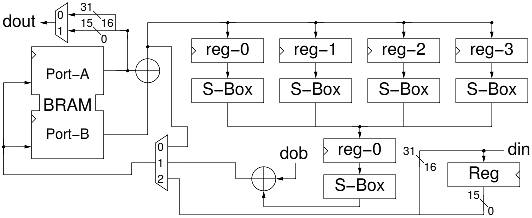
- Message injection uses serialized XORs.
- SubCrumb is implemented as DROM.
- Constraints: DRAM had to be used to keep control logic simple.

Shabal



- Based on paper by [Detrey].
- BRAM contains state register C and initialization vectors.
- Our I/O is more complex than [Detrey].
- BRAM makes controller more complex.

SHAvite-3



- 4 ROM based S-Boxes.
- Regular path of the mds matrix is an advantage.
- Mix column is realized through a shift register.

SHA-2

- Full Datapath
  - 700 slices
  - 65 clock cycles

SHA-2

- Full Datapath
  - 700 slices
  - 65 clock cycles
- Small Datapath
  - 520 slices
  - 595 clock cycles

SHA-2

- Full Datapath
  - 700 slices
  - 65 clock cycles
- Small Datapath
  - 520 slices
  - 595 clock cycles
- SHA-2 is not well suited for very small implementations.

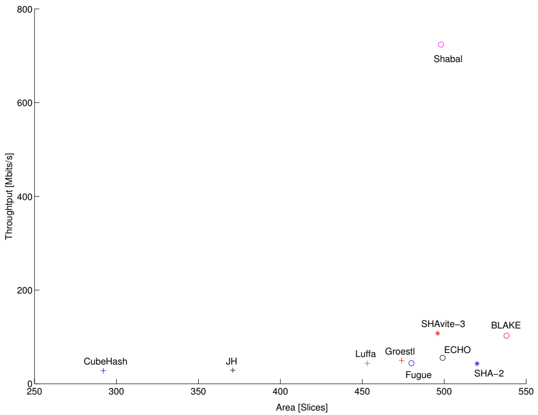
Performance Equations

Algorithm	Block Size (bits) b	Clock Cycles to hash N blocks $clk = st + (l + p) \cdot N + end$	Throughput $\frac{b}{(l+p) \cdot T}$
BLAKE	512	$18 + (32 + 480) \cdot N + 65$	$512/(512 \cdot T)$
CubeHash	256	$2 + (16 + 928) \cdot N + 9312$	$256/(928 \cdot T)$
ECHO	1536	$16 + (96 + 2449) \cdot N + 17$	$1536/(2545 \cdot T)$
Fugue	32	$33 + (2 + 61) \cdot N + 990$	$32/(63 \cdot T)$
Grøstl	512	$2 + (32 + 1120) \cdot N + 577$	$512/(1152 \cdot T)$
JH	512	$35 + (32 + 1574) \cdot N + 17$	$512/(1606 \cdot T)$
Luffa	256	$2 + (16 + 606) \cdot N + 647$	$256/(622 \cdot T)$
Shabal	512	$36 + (32 + 48) \cdot N + 208$	$512/(80 \cdot T)$
SHAvite-3	512	$18 + (32 + 648) \cdot N + 17$	$512/(680 \cdot T)$
SHA-256	512	$18 + (32 + 563) \cdot N + 17$	$256/(595 \cdot T)$

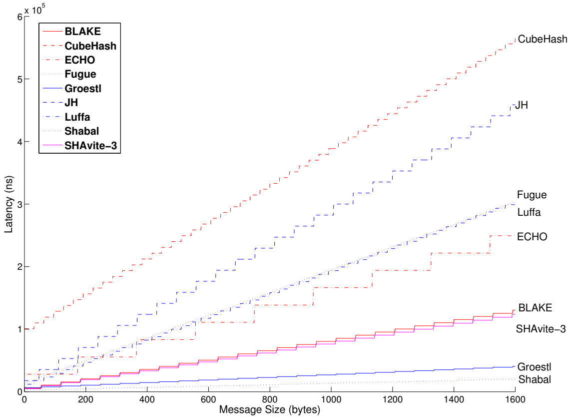
Implementation Results

Algorithm	Area (slices)	Block RAMs	Maximum Delay (ns) T	Large m: Throughput (Mbps)	Large m: Throughput/Area (Mbps/slice)	Small m: Throughput (Mbps)	Small m: Throughput/Area (Mbps/slice)
BLAKE	538	1	9.73	102.7	0.19	88.4	0.164
CubeHash	292	1	9.84	28.0	0.09	2.5	0.008
ECHO	499	1	10.87	55.5	0.11	54.8	0.110
Fugue	480	1	11.52	44.0	0.09	2.5	0.005
Grøstl	474	1	8.96	49.6	0.10	33	0.070
JH	371	1	10.98	29.0	0.08	28.6	0.077
Luffa	453	1	9.42	43.6	0.10	21.4	0.047
Shabal	498	1	8.84	723.9	1.45	178.8	0.359
SHAvite-3	496	1	6.99	107.7	0.22	102.4	0.206
SHA-256	520	1	10.01	42.9	0.08	40.6	0.070

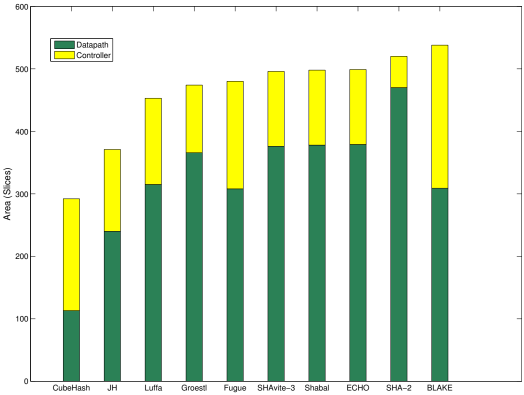
Results for Large Messages



Results for Short Messages



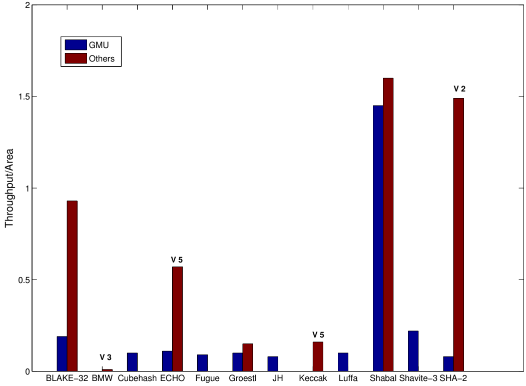
Control Logic vs. Datapath



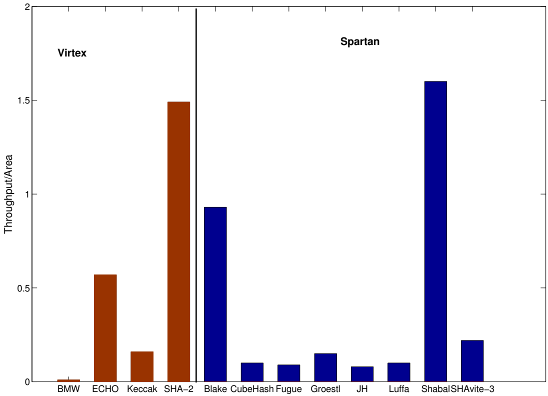
Implementation Results Comparison

Algorithm	Reference	Area (slices)	Block RAMs	Maximum Delay (ns)	I/O Width	Datapath Width	Clock Cycles $(I+p)$	Device	Functionality	Throughput (Mbps)	Throughput/ Area (Mbps/slice)
BLAKE-32	[?]	124	2	5.20	32	32	816	xc3s50	FA	115.0	0.93
BLAKE-32	TW	538	1	9.73	16	32	512	xc3s100e	FA	102.7	0.19
BMW	[?]	895	1	26.00	32	32	1060	xcv300	FA	9.0	0.01
CubeHash	[TW]	292	1	9.84	16	32	944	xc3s100e	FA	28.0	0.10
ECHO	[TW]	499	1	10.87	16	32	2545	xc3s100e	FA	55.5	0.11
ECHO	[?]	127	1	2.80	8	8	6593	xc5vlx50-2	FA	72.0	0.57
Fugue	[TW]	480	1	11.52	16	32	63	xc3s100e	FA	44.0	0.09
Grøstl	[?]	1276	0	16.67	64	64		xc3s1500	FA	192.0	0.15
Grøstl	[TW]	474	1	8.96	16	32	1152	xc3s100e	FA	49.6	0.10
JH	[TW]	371	1	10.98	16	32	1641	xc3s100e	FA	29.0	0.08
Keccak	[?]	444	1	3.77		64	3870	xc5vlx50	FA	70.0	0.16
Luffa	[TW]	453	1	9.42	16	32	622	xc3s100e	FA	43.6	0.10
Shabal	[?]	499	0	1.25		32	64	xc3s200	FA	800	1.60
Shabal	[TW]	498	1	8.84	16	32	80	xc3s100e	FA	723.9	1.45
Shavite-3	[TW]	496	1	6.99	16	32	680	xc3s100e	FA	107.7	0.22

Comparison of Candidate Implementations



Best Candidate Implementations



Thanks for your attention.