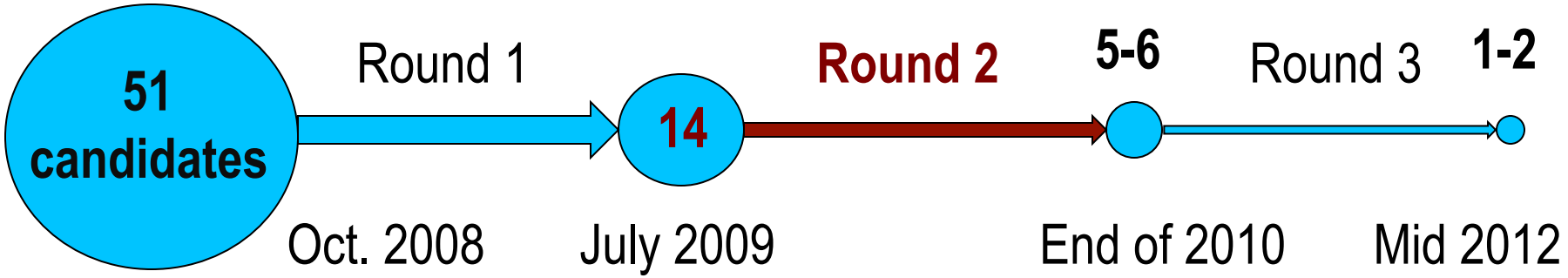
SHA-3 Competition in Hardware

Methodology, Tools, and Results of Comparing Fourteen Round 2 SHA-3 Candidates using Reconfigurable Hardware

Marcin Rogawski,
Ekawat Homsirikamol, and
Kris Gaj
George Mason University
U.S.A.

NIST SHA-3 Contest

- Timeline

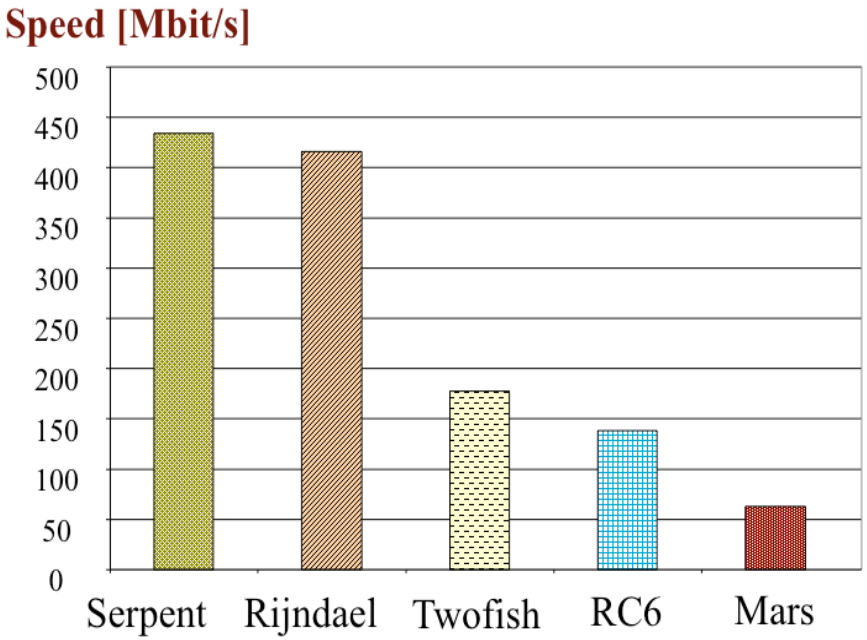


- Evaluation criteria
  - Security
  - Performance in software
  - **Performance in hardware**
  - Flexibility

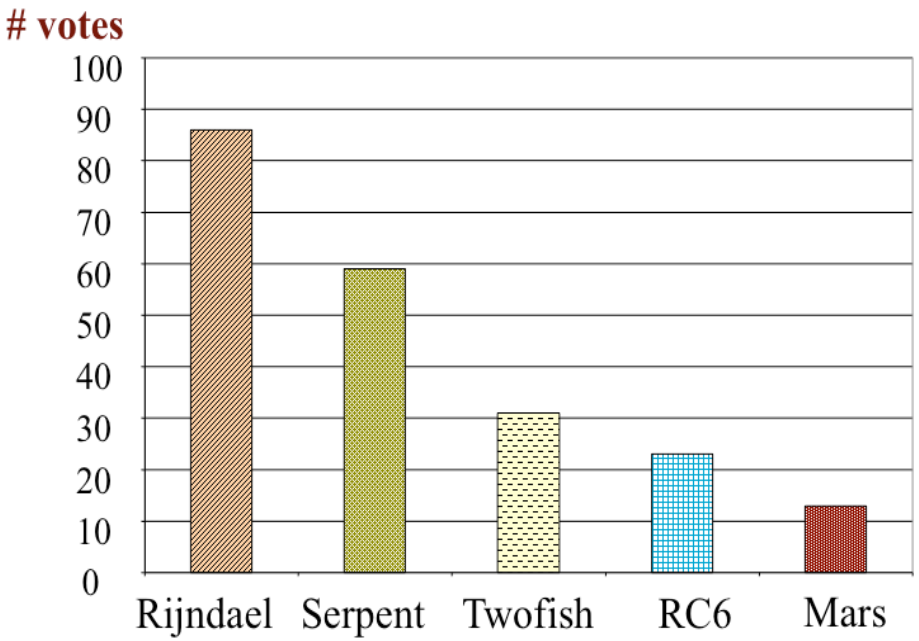
Lessons from the Past - AES Contest – 1997-2000

Round 2 of AES Contest, 2000

Speed in FPGAs



Votes at the AES 3 conference



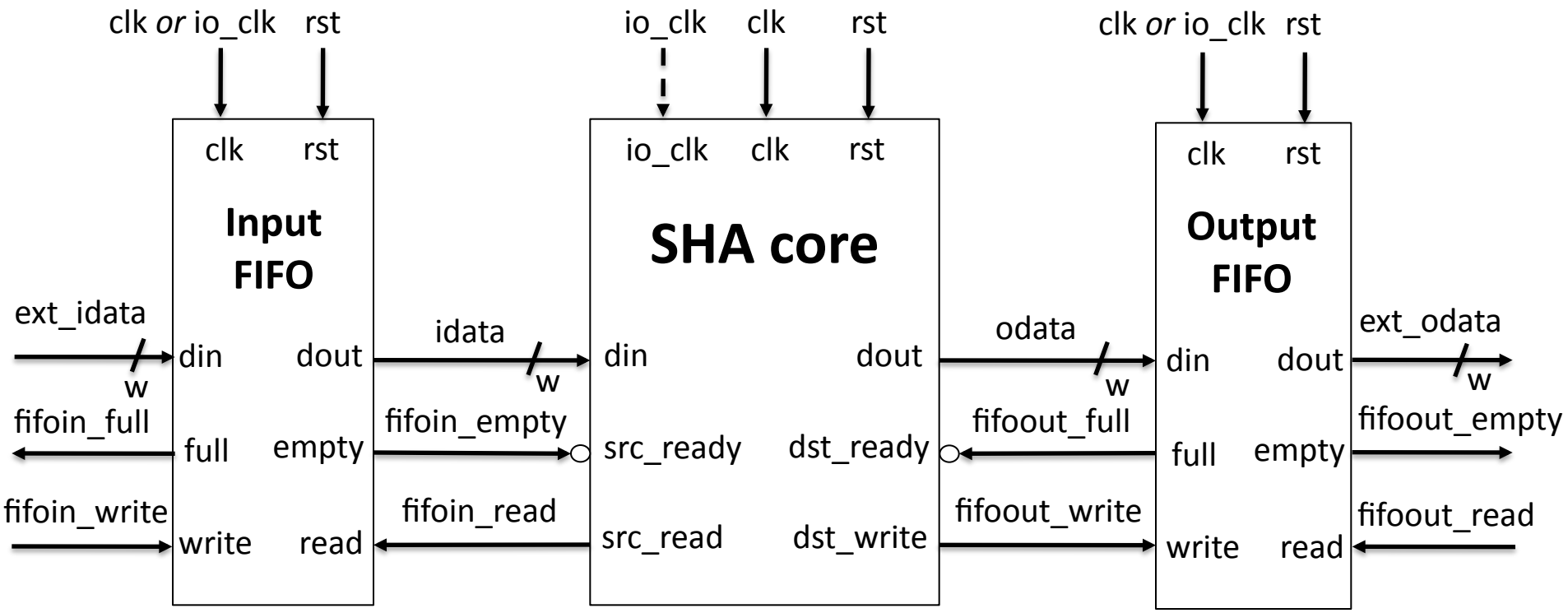
Our Methodology

- Uniform assumptions
  - 256-bit variant & 512-bit variant of each function, padding in software
- Uniform & practical interface
- Clear performance metrics
- Uniform optimization criteria
- Use of multiple FPGAs from two major vendors

  Xilinx: Spartan 3, Virtex 4, Virtex 5

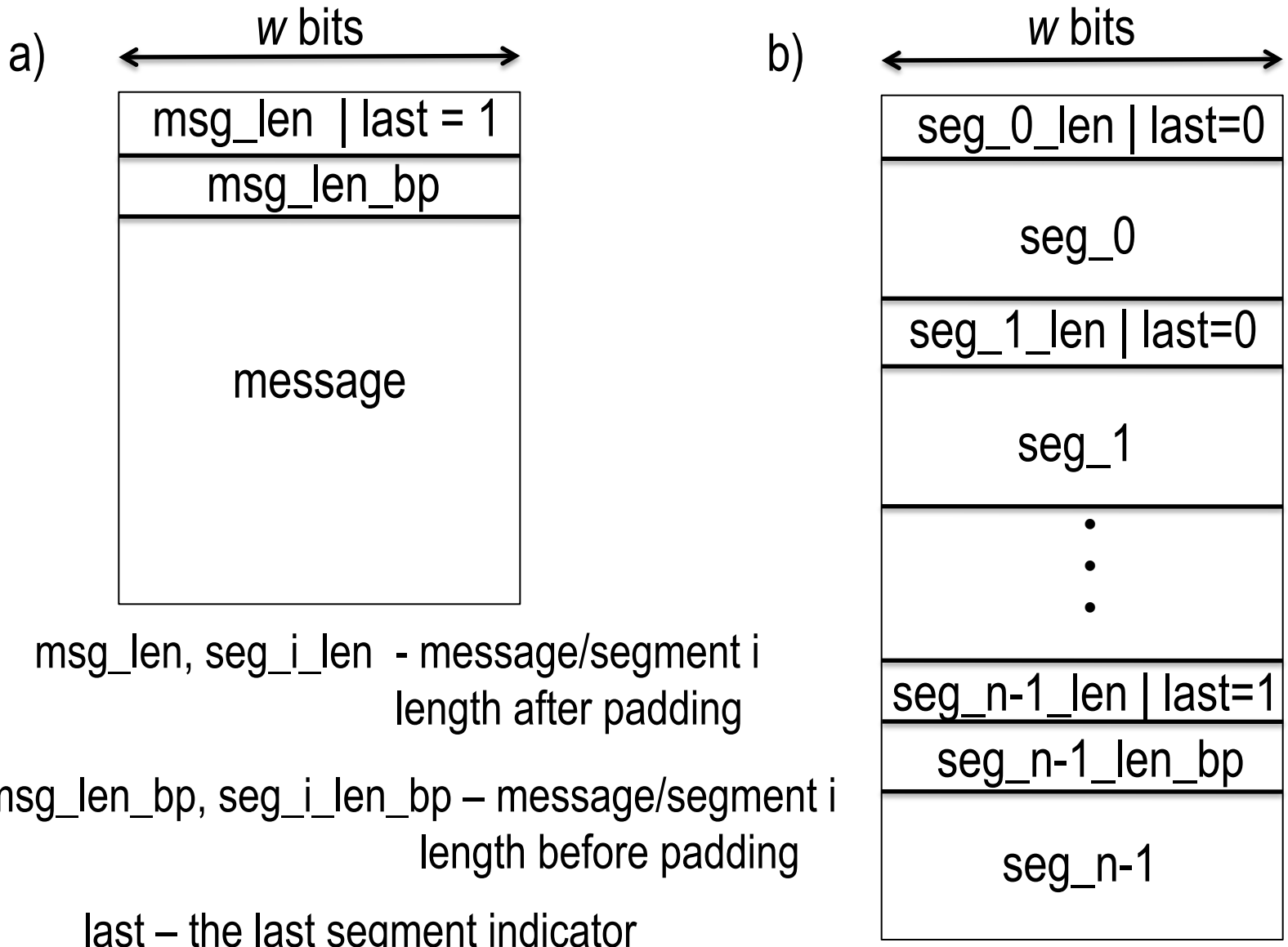
  Altera: Cyclone II, Cyclone III, Stratix II, Stratix III
- Use of ATHENa for generation, optimization, and comparative analysis of results
- Techniques for normalizing and averaging results

SHA Core Interface & Typical Configuration



- SHA core is an active component, surrounding FIFOs are passive and widely available
- Input interface is separate from an output interface
- Processing a current block, reading the next block, and storing a result for the previous message can be all done in parallel
- Separate I/O clock is optional, and is used only if necessary to maintain throughput

Format of Input Data for Padded Messages
a) one-segment message, b) multi-segment message



msg_len, seg_i_len - message/segment i length after padding

msg_len_bp, seg_i_len_bp – message/segment i length before padding

last – the last segment indicator

Hash Time in Clock Cycles (of the main clock)

$$HTime(N) = c_{INIT} + \lceil c_{IN}/r_{IO} \rceil + c_{BLOCK} \cdot N + c_{FINAL} + \lceil c_{OUT}/r_{IO} \rceil$$

N - number of message blocks after padding. *N=msg_len/block_size.*

c_{INIT} - number of clock cycles necessary to establish communication with the source of data (typically, Input FIFO)

c_{IN} - number of clock cycles required to read the very first block of the message. c_{IN} = *block_size/w.*

r_{IO} - ratio of the I/O clock frequency to the main clock frequency

c_{BLOCK} - number of clock cycles required to process one block of the message

c_{FINAL} - number of clock cycles required for the finalization (once per message)

c_{OUT} - number of clock cycles required to write hash value to the destination circuit (typically Output FIFO). c_{OUT}=*output_size/w.*

Performance Metrics - Speed

Message block size, from specification

Maximum Throughput (for long messages) = $\dfrac{\text{block\_size}}{T * (Htime(N+1) - Htime(N))}$

Minimum clock period, from the place & route report and/or static timing analysis report

Time of processing of a single block in clock cycles, from analysis of a block diagram and/or functional simulation

Performance Metrics - Area

$$Resource\ Utilization_{\text{Spartan3}} = (\#CLB\ slices,\ \#BRAMs,\ \#MULs)$$

$$Resource\ Utilization_{\text{Cyclone III}} = (\#LE,\ \#memory\_bits,\ \#MULs).$$

We force these vectors to look as follows

through the synthesis and implementation options

$$Resource\ Utilization_{\text{Spartan3}} = (\#CLB\ slices,\ 0\ ,\ 0\ )$$

$$Resource\ Utilization_{\text{Cyclone III}} = (\#LE,\ 0\ ,\ 0\ ).$$

Area

Choice of Optimization Target

Primary Optimization Target: **Throughput to Area Ratio**

Features:

- practical: good balance between speed and cost
- very reliable guide through the entire design process, facilitating the choice of
  - high-level architecture
  - implementation of basic components
  - choice of tool options

Disadvantages of other Optimization Targets

Throughput:

- leads to highly unrolled architectures: minor increase in speed at the cost of major increase in area

Latency:

- depends highly on message size (one block of message has a different meaning for different functions)
- quite dependent on the interface
- influence of padding

Area:

- leads to highly sequential and relatively slow designs
- result dependent on the maximum amount of area available

High-level Architecture

- identifying most complex task that can be executed in an iterative fashion (without significant overhead)
- either multiple rounds or a fraction thereof may be appropriate
- trade-off: number of clock cycles per block vs. clock period (length of the critical path)

Function	Main Iterative Task	Function	Main Iterative Task
BLAKE	$G_i..G_{i+3}$	**JH**	Round function R_8
BMW	entire function	**Keccak**	Round R
CubeHash	one round	**Luffa**	The Step Function, Step
ECHO	AES round/AES round/ BIG.SHIFTROWS, BIG.MIXCOLUMNS	**Shabal**	Two iterations of the main loop
Fugue	2 subrounds (ROR3, CMIX, SMIX)	**SHAvite-3**	AES round
Groestl	Modified AES round	**SIMD**	4 steps of the compression function
Hamsi	Truncated Non-Linear Permutation P	**Skein**	8 rounds of Threefish-256

I/O Bus Width, Hash Time, & Throughput

	256-bit variants			512-bit variants		
	I/O Bus Width	Hash Time [cycles]	Throughput [Mbit/s]	I/O Bus Width	Hash Time [cycles]	Throughput [Mbit/s]
BLAKE	64	1+8+10·N+4	512/(10·T)	64	1+**16/2**+14·N+**8/2**	**1024**/(**14**·T)
BMW	64	1+8/8+N+1	512/T	64	1+**16/16**+N+**8/16**	**1024**/T
CubeHash	64	1+4+16·N+160+4	256/(16·T)	64	1+4+16·N+160+**8**	256/(16·T)
ECHO	64	2+24+25·N+4	1536/(25·T)	64	2+**16**+**31**·N+**8**	**1024**/(**31**·T)
Fugue	32	1+2·N+18+8	32/(2·T)	32	1+**4**·N+**21**+**16**	32/(**4**·T)
Groestl	64	1+8+21·N+4	512/(21·T)	64	1+**16/2**+29·N+**8/2**	**1024**/(**29**·T)
Hamsi	32	2+1+3·(N-1)+6+8	32/(3·T)	**64**	2+1+6·(N-1)+6+8	**64**/(**6**·T)
JH	64	2+8+36·N+4	512/(36·T)	64	2+8+36·N+**8**	512/(36·T)
Keccak	64	1+16+24·N+4	1088/(24·T)	64	1+16+24·N+**8**	**576**/(24·T)
Luffa	64	2+4+8·N+8+4	256/(8·T)	64	2+4+8·N+8+**8**	256/(8·T)
Shabal	64	2+8+1+49·N+ 3·49+4	512/(49·T)	64	2+8+1+49·N+ 3·49+**8**	512/(49·T)
SHAvite-3	64	2+8+37·N+4	512/(37·T)	64	2+16+**57**·N+**8**	**1024**/(**57**·T)
SIMD	64	2+8+8+9·N+4	512/(9·T)	64	2+16+9+9·N+**8**	**1024**/(9·T)
Skein	64	1+4+9·N+4	256/(9·T)	64	1+8+9·N+**8**	**512**/(9·T)
SHA-2	32	1+1+65·N+8	512/(65·T)	**64**	1+1+81·N+8	**1024**/(**81**·T)

Basic Operations of SHA-3 Candidates

Function	NTT	Linear code	S-box	GF MUL	MUL	mADD	ADD /SUB	Boolean
BLAKE						mADD3	ADD	XOR
BMW						mADD17	ADD,SUB	XOR
CubeHash							ADD	XOR
ECHO			AES 8x8	x02, x03				XOR
Fugue			AES 8x8	x04..x07				XOR
Groestl			AES 8x8	x02..x07				XOR
Hamsi		LC[128, 16,70]	Serpent 4x4					XOR
JH			Serpent 4x4	x2, x5				XOR
Keccak								NOT,AND,XOR
Luffa			4x4	x2				XOR
Shabal					x3, x5		ADD,SUB	NOT,AND,XOR
SHAvite-3			AES 8x8	x02, x03				NOT,XOR
SIMD	NTT_{128}				x185, x233	mADD3	ADD	NOT,AND,OR
Skein							ADD	XOR
SHA-256						mADD5		NOT,AND,XOR

NTT – Number Theoretic Transform, GF MUL – Galois Field multiplication,
MUL – integer multiplication, mADDn – multioperand addition with n operands

Optimizing Basic Operations

- at least two alternative architectures for several most complex operations (NTT, linear code, AES SubBytes, multioperand addition)
- all basic operation designs applied uniformly to all functions (through a common package)
- any possible doubts resolved experimentally

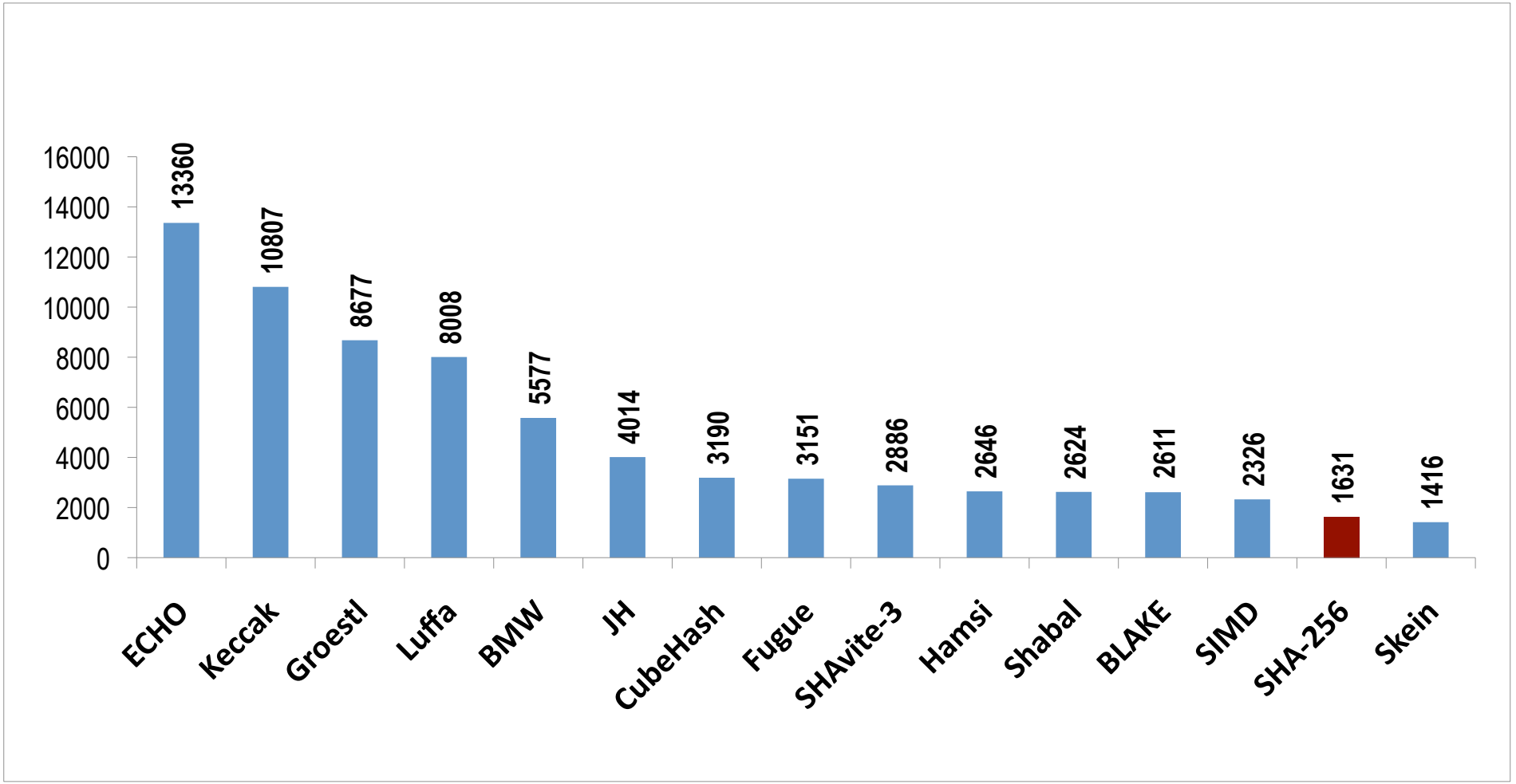
Verification of Codes

- universal testbench common for SHA-2 and all SHA-3 candidates
- special padding script developed in Perl to pad messages included in KAT (Known Answer Test) files
- final result tested using known answer tests
- final verification automated using ATHENa

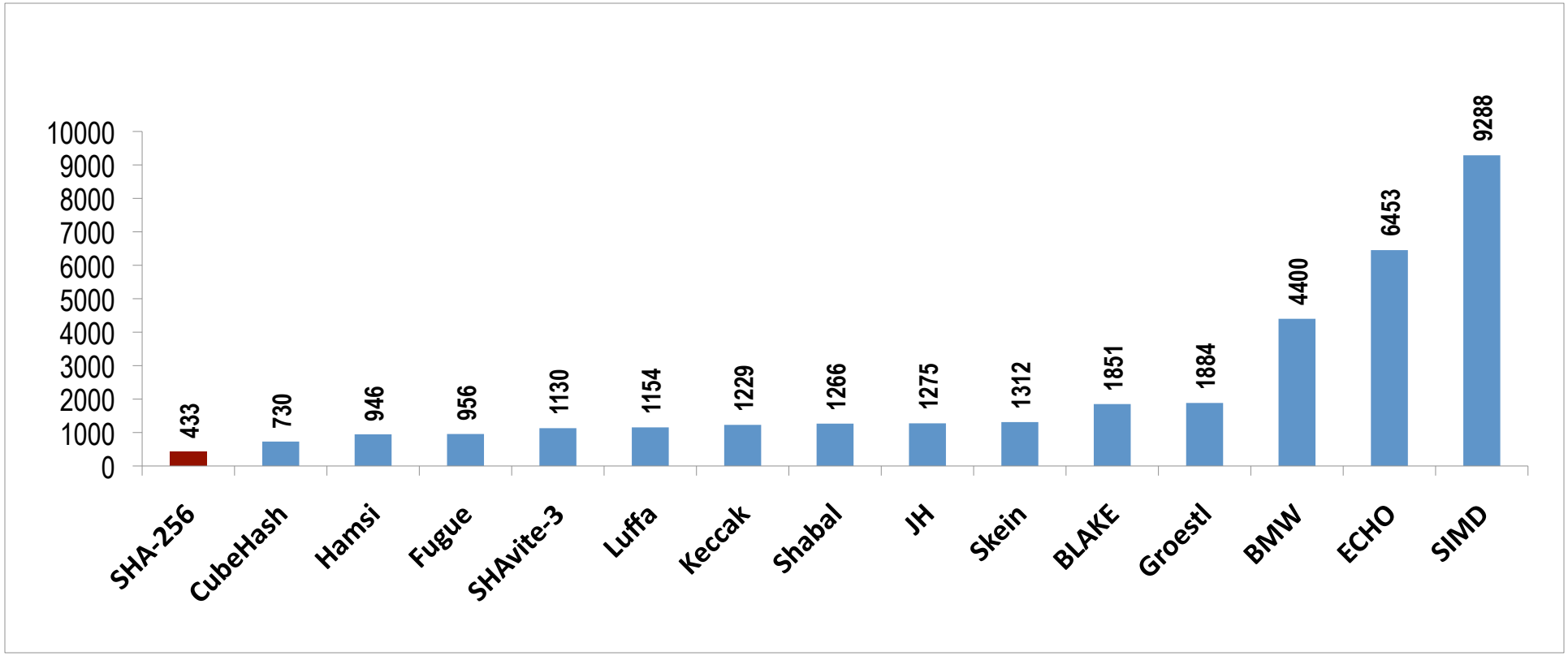
Generation of Results Using ATHENa

- generation of results facilitated by ATHENa
  - batch mode of FPGA tools
  - ease of extraction and tabulation of results
  - uniform optimizations using ATHENa
    - Frequency search for the best requested synthesis and implementation clock frequencies (effective for Xilinx only)
    - Exhaustive search for the best set of tool options
    - Placement search for the best starting point of placement

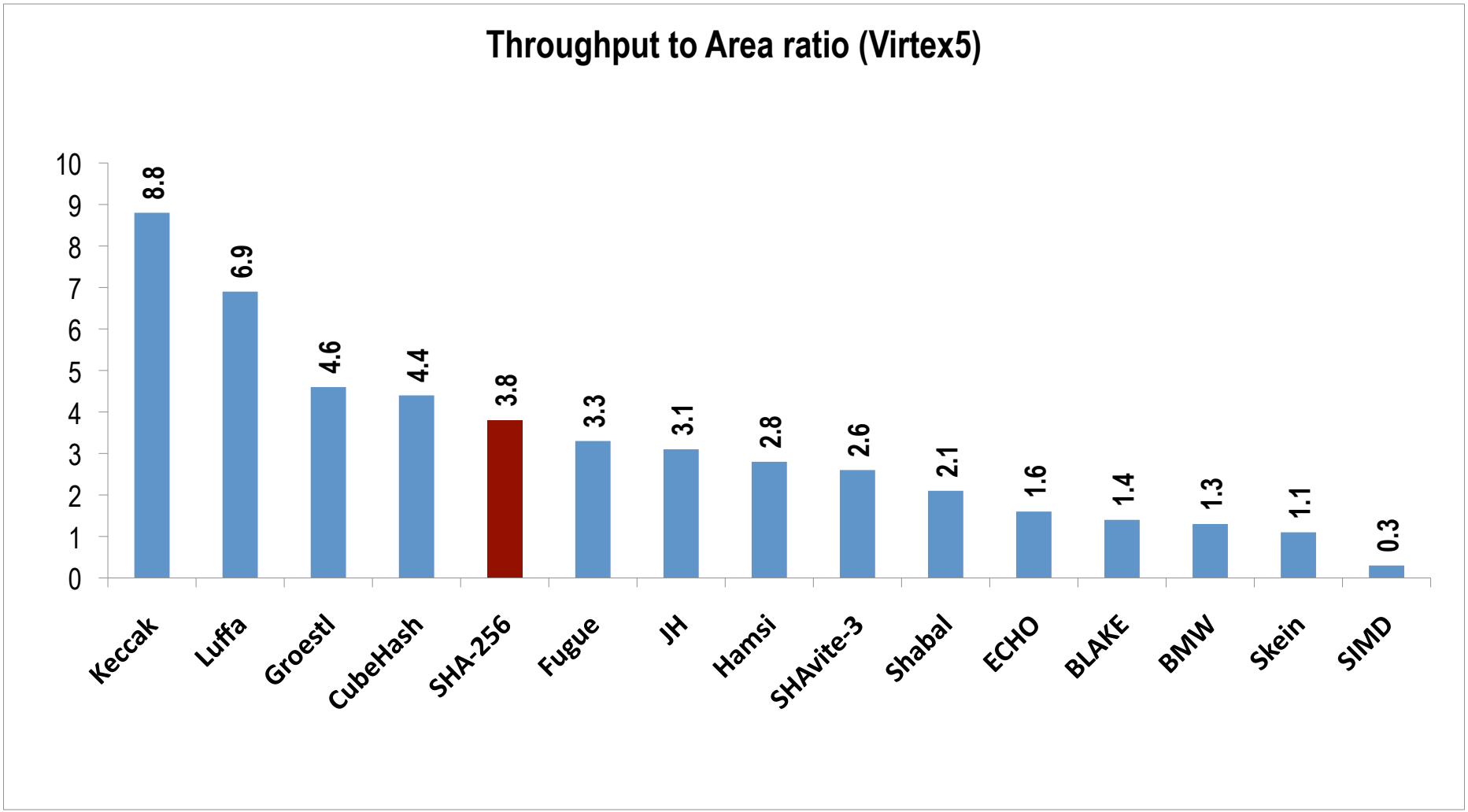
Throughput [Mbit/s] for Xilinx Virtex 5 – 256-bit variants



Area [CLB slices] for Xilinx Virtex 5 – 256-bit variants



Throughput/Area for Xilinx Virtex 5 – 256-bit variants

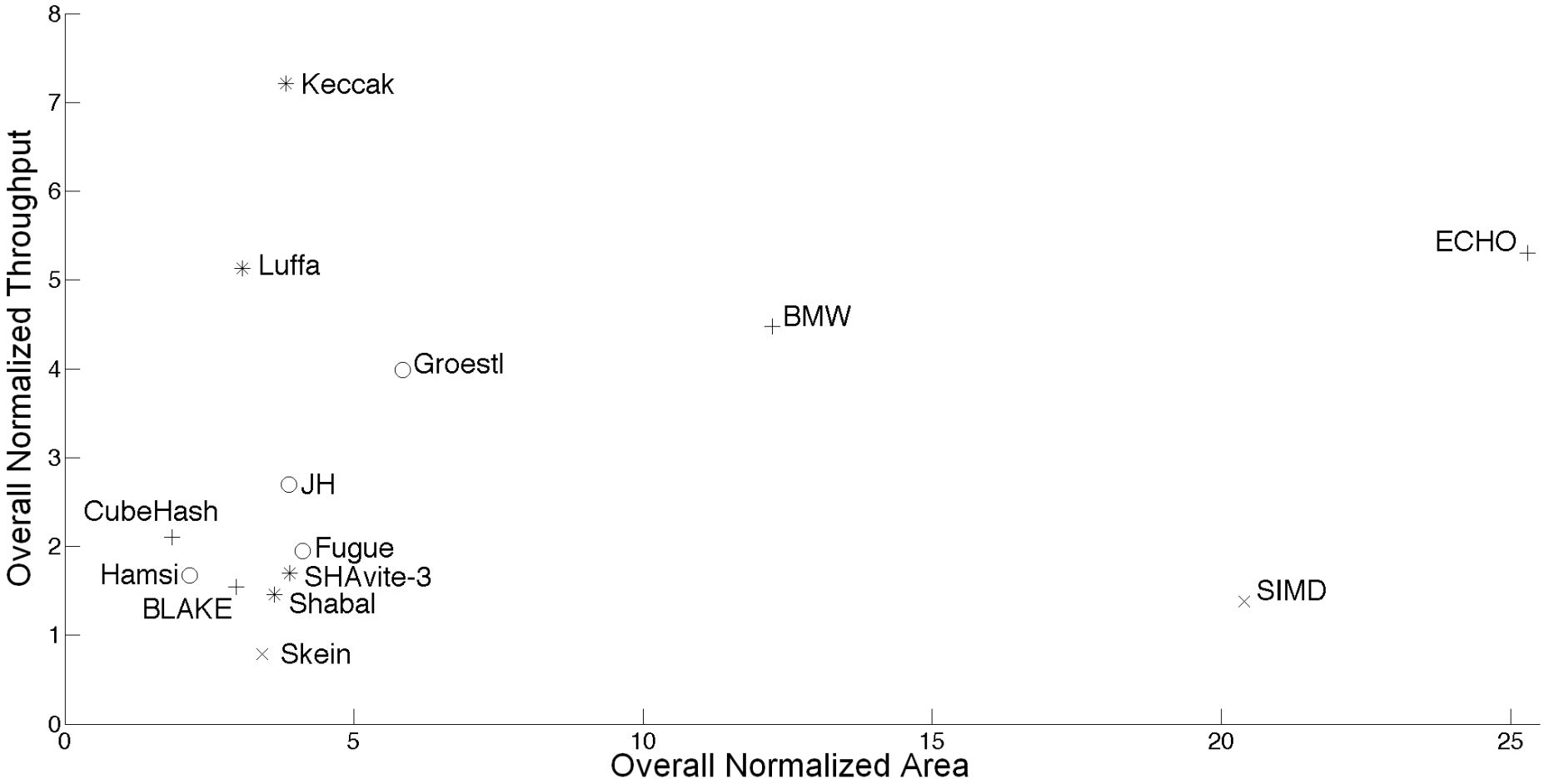


Throughput to Area Ratio Normalized to the Results for SHA-256

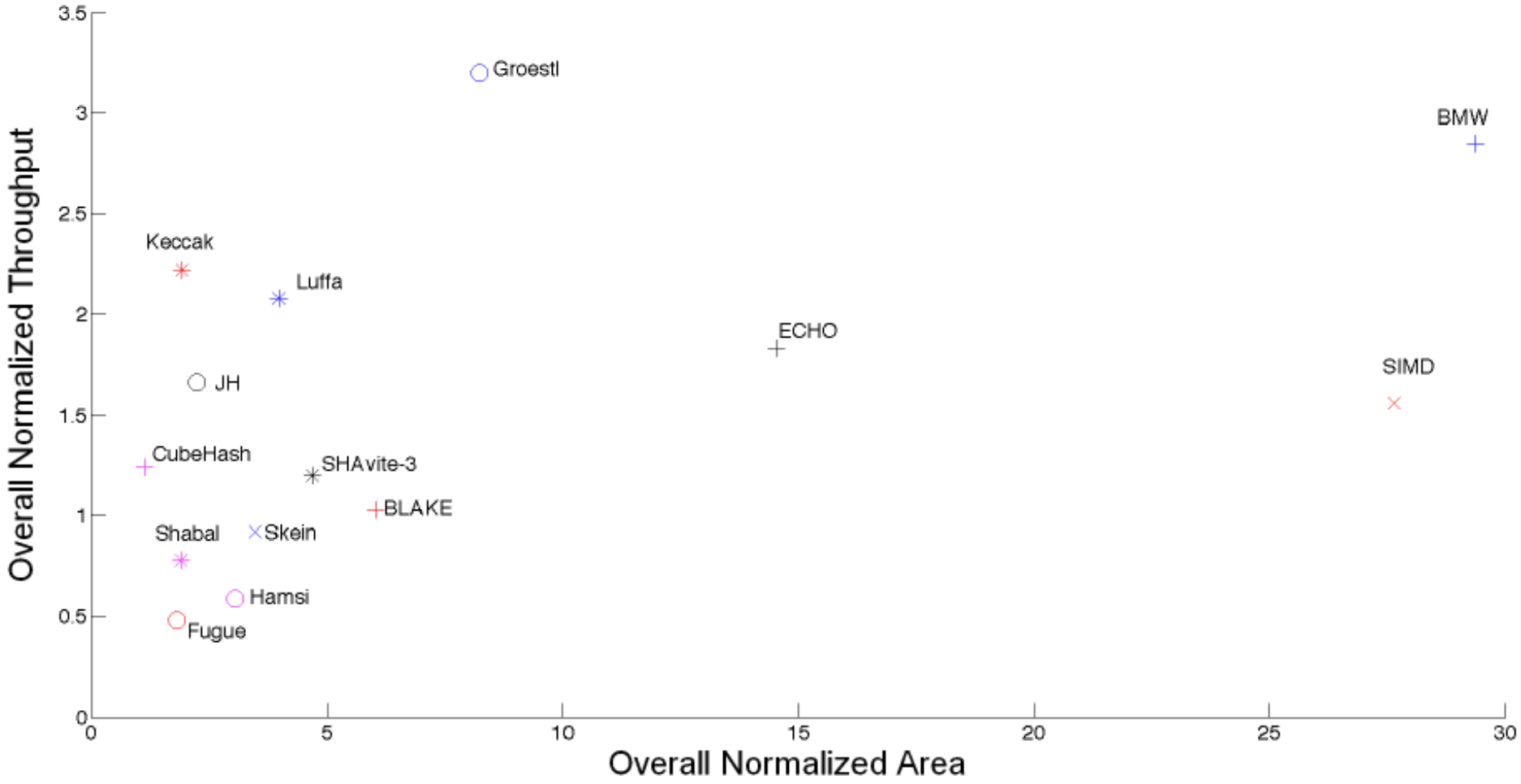
Function	Spartan 3	Virtex 4	Virtex 5	Cyclone II	Cyclone III	Stratix II	Stratix III	Overall
Keccak	1.54	1.60	2.34	2.27	2.18	1.72	1.73	1.89
Luffa	1.57	1.56	1.84	2.04	1.78	1.48	1.52	1.67
CubeHash	1.04	1.15	1.16	1.13	1.14	1.16	1.13	1.13
Hamsi	0.62	0.69	0.74	0.94	1.01	0.69	0.78	0.77
JH	0.49	0.46	0.84	0.65	0.71	0.96	0.95	0.70
Groestl	0.23	0.25	1.22	0.80	0.81	1.32	1.22	0.69
BLAKE	0.30	0.29	0.37	0.71	0.62	0.88	0.82	0.52
Fugue	0.42	0.36	0.88	0.33	0.33	0.58	0.63	0.47
SHAvite-3	0.33	0.30	0.68	0.27	0.28	0.74	0.81	0.44
Shabal	0.24	0.42	0.55	0.44	0.38	0.44	0.41	0.40
BMW	0.25	0.33	0.34	0.38	0.36	0.43	0.38	0.37
Skein	0.21	0.22	0.29	0.22	0.21	0.24	0.24	0.23
ECHO	0.13	0.18	0.41	N/A	0.15	0.22	0.25	0.21
SIMD	0.07	0.06	0.07	0.08	0.07	0.07	0.07	0.07

Overall = Geometric Mean of Results for all FPGA families

Throughput vs. Area Normalized to Results for SHA-256 and Averaged over 7 FPGA Families – 256-bit variants



Throughput vs. Area Normalized to Results for SHA-512 and Averaged over 7 FPGA Families – 512-bit variants



Conclusions

- Large differences among competing candidates;

  e.g., the best to worst result ratio:

  9 for Throughput (Keccak-256 vs. Skein-256)

  13 for Area (CubeHash-256 vs. ECHO-256)

  27 for Throughput/Area (Keccak-256 vs. SIMD-256)

- Only three candidates:

  Keccak-256, Luffa-256, & CubeHash-256

  outperform SHA-256 in terms of Throughput/Area

- Only two candidates:

  Keccak-512 and CubeHash-512

  outperform SHA-512 in terms of Throughput/Area