FPGA Benchmarking for High-Speed and Medium-Speed Implementations

Kris Gaj
George Mason University
U.S.A.
Codes developed and results generated by:

Ekawat Homsirikamol
Marcin Rogawski
Malik Umar Sharif
Rabia Shahid
Bilal Habib
Outline

- Flexibility of the SHA-3 candidates in hardware in terms of speed-area trade-offs using
  a. folding
  b. unrolling
  c. pipelining
  d. embedded FPGA resources (embedded DSP units, memory blocks, etc.)
- FPGA performance metrics
- Summary of GMU results
- Presentation and comparison of FPGA results from various groups
- Discussion of possible sources of discrepancies
- Pros and Cons of all candidates
High-Speed Architecture of a Hash Function

Features

• datapath width = state size
• optimization for throughput or throughput to area ratio [rather than for area or power]
• typically [but not always] one clock cycle per one round/step

Our Target

High-Speed Architecture Optimized for the Maximum Throughput to Area Ratio
Starting Point: Basic Iterative Architecture

- datapath width = state size
- one clock cycle per one round/step

Block processing time = \#R \cdot T

\#R = number of rounds/steps
T = clock period

Currently, most common architecture used to implement SHA-1, SHA-2, and many other hash functions.
Horizontal Folding

- datapath width = state size
- two clock cycles per one round/step

Block processing time = \((2 \cdot \#R) \times T'\)

\[ T/2 < T' < T \]

typically \(T' \approx T/2\)

Area/2 < Area' < Area

Typically Throughput/Area ratio increases
Horizontally Folded vs. Basic Iterative BLAKE

Blake

x1 – basic iterative
/2(h) – folded horizontally by a factor of 2
Horizontal folding of CubeHash
Horizontally Folded vs. Basic Iterative CubeHash

CubeHash

Throughput (Mbit/s)

Area (Slices)

x1 – basic iterative
/2(h) – folded horizontally by a factor of 2
x2 – two times unrolled

h=256,m=long
h=512,m=long
SHA-3 Candidates Benefiting from Horizontal Folding

BLAKE: two layers of G-functions - folding by 2 : /2

Fugue-256: two iterations of (ROR3, CMIX, SMIX) - folding by 2 : /2

Fugue-512: four iterations of (ROR3, CMIX, SMIX) - folding by 4: /4

ECHO: two layers of BIG.SubBytes - folding by 3/2: x2/3

SHAvite-3-256: three iterations of AES Round - folding by 3 : /3

SHAvite-3-512: four iterations of AES Round - folding by 4: /4
Unrolling

- datapath width = state size
- one clock cycle per two rounds

Block processing time = \((\#R/2) \times T'\)

\[ T < T' < 2 \times T \]

typically \(T' \approx 2 \times T\)

\[ \text{Area}/2 < \text{Area}' < 2 \times \text{Area} \]

Typically \(\text{Area}' \approx 2 \times \text{Area}\)

Typically Throughput/Area ratio decreases
Unrolling of CubeHash

CubeHash

Throughput (Mbit/s) vs. Area (Slices)

- Blue line: \( h=256, m=\text{long} \)
- Green line: \( h=512, m=\text{long} \)

- \( x_1 \) – basic iterative
- \( /2(h) \) – folded horizontally by a factor of 2
- \( x_2 \) – two times unrolled
Unrolling of Hamsi

Hamsi

Throughput (Mbit/s)

Area (Slices)

- x1 – basic iterative
- x3 – unrolled by a factor of 3

h=256, m=long
How Can Functions Benefit from Unrolling?

Functions having non-uniform rounds/steps can benefit from unrolling.

Examples:

**Skein**: 8 consecutive rounds use 8 different rotation amounts.

**SIMD**: 36 steps use 16 different rotation amounts.

The logic of a single round may be significantly simplified as a result of unrolling.
Unrolling Skein

Skein

- x1 – basic iterative
- x4 – unrolled by a factor of 4
- x8 – unrolled by a factor of 8
Basic operation in Skein x1 and Skein x4

Basic operation, MIX, in Skein x1
(basic iterative)

Basic operation, MIX, in Skein x4
(4 times unrolled)
Basic operation in SIMD x1 and SIMD x4

Operation required in SIMD x1
(basic iterative)

Equivalent operation in SIMD x4
(4 times unrolled)
How to Reduce Area? – The case for Vertical Folding

- datapath width = state size/2
- two clock cycles per one round/step

Block processing time = \((2 \cdot \#R) \cdot T'\)

typically \(T' \approx T\)

\(\text{Area}/2 < \text{Area}' < \text{Area}\)
Folding of Luffa

/N(v) – folded vertically by a factor of N
xN – unrolled by a factor of N
x1 – basic iterative
Folding of ECHO

/\N(v) – folded vertically by a factor of N
x1 – basic iterative
/x2/3(h) – folded horizontally by a factor of 3/2 [BIG.SubBytes logic reused]
Folding of Fugue

Fugue

x1 – basic iterative  \( /2(h) \) – folded horizontally by a factor of 2
\( /2(h) /N(v) \) – folded horizontally by a factor of 2, and vertically by a factor of N
Folding of Groestl

- $x_1(P+Q)$ – basic iterative with P&Q executing in parallel
- $x_1(P/Q \text{ pp2})$ – basic iterative with P&Q sharing the same logic with two stages of pipelining
- /2(v) (P/Q pp2) – folded vertically by a factor of 2, shared P/Q with two stages of pipelining
- /8(v) (P/Q pp2) – folded vertically by a factor of 8, shared P/Q with two stages of pipelining
Folding of SHAvite-3

/3(h) – folded horizontally by a factor of 3
/3(h) /N(v) – folded horizontally by a factor of 3, and vertically by a factor of N
Special Case - BMW

BMW: no clear round structure

Basic architecture combinational

Very large area

One clock cycle per message block
New Folded Architecture of BMW

New, previously not reported, folded architecture

33 clock cycles per message block

Significant reduction in the circuit area
Special Case - BMW

x1 – basic iterative
/16(h) – f1 folded horizontally by a factor of 16
Combined Results for 256-bit SHA-3 Variants
Combined Results for 512-bit SHA-3 Variants

SHA-3-512 Various Architectures

Throughput (MB/s) vs. Area (Slices)
# Algorithms Ranked According to the Flexibility (1)

## Highest Flexibility, Best Area Reduction Factors

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Flexibility Factors</th>
<th>Area Reduction Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAKE:</td>
<td>( x_1 ), ( \frac{2}{h} ), ( \frac{4}{h} )</td>
<td>( \frac{2}{h}, \frac{4}{h} )</td>
</tr>
<tr>
<td>Luffa-256:</td>
<td>( x_1 )</td>
<td>( \frac{1}{3} )</td>
</tr>
<tr>
<td>Luffa-512:</td>
<td>( x_1 )</td>
<td>( \frac{1}{5} )</td>
</tr>
</tbody>
</table>

## High Flexibility, Medium Area Reduction Factors

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Flexibility Factors</th>
<th>Area Reduction Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECHO:</td>
<td>( x_1 ), ( \frac{2}{3} )</td>
<td>( \frac{2}{h}, \frac{4}{h}, \frac{8}{h}, \frac{16}{h} )</td>
</tr>
<tr>
<td>Fugue-256:</td>
<td>( x_1 ), ( \frac{2}{h} )</td>
<td>( \frac{2}{h}, \frac{4}{h}, \frac{8}{h}, \frac{16}{h} )</td>
</tr>
<tr>
<td>Groestl:</td>
<td>( x_1(P+Q) ), ( x_1(P/Q \text{ pp2}) )</td>
<td>( \frac{2}{h}, \frac{4}{h}, \frac{8}{h}, \frac{16}{h} (P/Q \text{ pp2}) )</td>
</tr>
<tr>
<td>JH:</td>
<td>( x_1 )</td>
<td>( \frac{2}{h}, \frac{4}{h}, \frac{8}{h}, \frac{16}{h}, \frac{32}{h}, \frac{64}{h} )</td>
</tr>
</tbody>
</table>
Moderate Flexibility

BMW: \( \times 1 \) /16(h)  
CubeHash: \( \times 1 \) /2(h)  
SHAvite-3-256: \( /3(h) \) /2, /4 (v)  
SHAvite-3-512: \( /4(h) \) /2, /4 (v)  
Skein: \( x1, x4, x8 \)  
Shabil: \( x1, x2, x3, x4, x6 \)  

Unknown, Most Likely Low Flexibility

Keccak  
Hamsi
How to Increase the Speed? : The case for pipelining and parallel processing

- Protocols: IPSec, SSL, WLAN (802.11)
- Minimum Required Throughput Range: 100 Mbit/s - 40 Gbit/s (based on the specs of Security Processors from Cavium Networks, HiFn, and Broadcom)
- Supported sizes of packets: 40B - 1500B
  
  1500 B = Maximum Transmission Unit (MTU) for Ethernet v2
  576 B = Maximum Transmission Unit (MTU) for Internet IPv4 Path
- Most Common Operation Involving Hashing: HMAC
Cumulative Distribution of Packet Sizes
HMAC

\[
\text{KEY} \oplus \text{opad} = \text{KEY'}
\]

message m

\[
\begin{align*}
\text{KEY} \oplus \text{ipad} &= \text{KEY''} \\
\text{h} &\rightarrow \text{HMAC}
\end{align*}
\]

- American standard FIPS 198
- Arbitrary hash function and key size
Execution Time for Short Messages up to 1000 bits
Virtex 5, 256-bit variants of algorithms
Execution Time for Short Messages up to 1000 bits
Virtex 5, 512-bit variants of algorithms
Multiple Packets Available for Parallel Processing
Parallel Processing

Data Stream 1 . . . . . . . .

Data Stream k

IV \rightarrow H \rightarrow R \rightarrow \text{CLR}

\text{Step t}

W_t \rightarrow K_t

IV \rightarrow H \rightarrow R \rightarrow \text{CLR}

\text{Step t}

W_t \rightarrow K_t
Pipelining

Stage 1
Stage 2

IV → H → + → R1

IV → H → + → R2

W_t → Stage 2 → K_t
BMW vs. CubeHash (1)

In Virtex 5:

Clock period:
CubeHash    -    5 ns
BMW          -  100 ns

Let us assume that the pipeline can be inserted every 5 ns
(clock frequency = 200 MHz).

Number of pipeline stages:
CubeHash    -    1
BMW          -  20
BMW vs. CubeHash (2)

BMW:

Before pipelining:
Throughput = 512/(100ns) = 5.12 Gbit/s
Area = 4400 CLB slices

After pipelining:
Throughput' = 512/(5ns) = 100 Gbit/s
Area' = 4400 + 20*0.1*4400 CLB slices = 13,200 CLB slices
(assuming 10% increase in area per pipeline stage)

CubeHash:

Throughput = 256/(16*5ns) = 3.2 Gbit/s
Area = 700 CLB slices

In order to reach the speed of 100 Gbit/s,
the required area of CubeHash = 700 * (100/3.2) = 21,875 CLB slices
BMW vs. CubeHash (3)

50 Gbit/s

BMW:
N=10
Throughput' = \frac{512}{(10\text{ns})} = 51.2 \text{ Gbit/s}
Area' = 4400 + 10 \times 0.1 \times 4400 \text{ CLB slices} = 8,800 \text{ CLB slices}

CubeHash:
In order to reach the speed of 50 Gbit/s:

The required area of CubeHash = 700 \times \frac{50}{3.2} = 10,938 \text{ CLB slices}
Reported Pipelined Implementations by Savas et al.

Pipelined Implementations:
    Multi-Message Hashing by Savas et al.

Number of pipeline stages
Keccak   - 5
Luffa    - 2
BMW      - 18

Results for Spartan3, Virtex 2, Virtex 4, 90nm ASIC.
## Improvement of the Throughput/Area ratio

<table>
<thead>
<tr>
<th></th>
<th>Spartan 3</th>
<th>Virtex 2</th>
<th>Virtex 4</th>
<th>ASIC 90nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keccak</td>
<td>2.0</td>
<td>1.2</td>
<td>1.7</td>
<td>1.7</td>
</tr>
<tr>
<td>Luffa</td>
<td>1.1</td>
<td>1.2</td>
<td>1.3</td>
<td>1.5</td>
</tr>
<tr>
<td>BMW</td>
<td>11.5</td>
<td>11.0</td>
<td>10.8</td>
<td>1.8</td>
</tr>
</tbody>
</table>
### Maximum Throughput [Gbit/s] Reached

<table>
<thead>
<tr>
<th></th>
<th>Spartan 3</th>
<th>Virtex 2</th>
<th>Virtex 4</th>
<th>ASIC 90nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keccak</td>
<td>14.9</td>
<td>15.0</td>
<td>22.3</td>
<td>74.4</td>
</tr>
<tr>
<td>Luffa</td>
<td>6.0</td>
<td>12.5</td>
<td>13.9</td>
<td>35.4</td>
</tr>
<tr>
<td>BMW</td>
<td>28.7</td>
<td>43.2</td>
<td>58.0</td>
<td>133.0</td>
</tr>
<tr>
<td></td>
<td>Thr/Area</td>
<td>Thr</td>
<td>Area</td>
<td>Short msg.</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>-----</td>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>BLAKE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CubeHash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECHO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fugue</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Groestl</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamsi</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keccak</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Luffa</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shabal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHAvite-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skein</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

256-bit variants

512-bit variants

TH/REA  TR  AEA  SHORT MEG.
Overall Normalized Throughput/Area: 256-bit variants
Normalized to SHA-256, Averaged over 7 FPGA families

results reported at the SHA-3 conference; Fugue, Shabal, and SIMD improved since then
Overall Normalized Throughput/Area: 512-bit variants
Normalized to SHA-512, Averaged over 7 FPGA families

results reported at the SHA-3 conference; Fugue, Shabal, and SIMD improved since then
What is an FPGA?

- Configurable Logic Blocks
- I/O Blocks
- Block RAMs
RAM Blocks and DSP Units
In Xilinx and Altera FPGAs
<table>
<thead>
<tr>
<th>Hash Algorithm</th>
<th>DSP Adders</th>
<th>DSP Multipliers</th>
<th>Block Memories</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAKE</td>
<td>Yes</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>BMW</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CubeHash</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ECHO</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Fugue</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Groestl</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Hamsi</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>JH</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Keccak</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Luffa</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SHA-2</td>
<td>Yes</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Shabal</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>SHAvite-3</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>SIMD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Skein</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
BLOCK MEMORIES
Block Memories used to implement T-boxes/S-boxes

- ECHO, SHA
tite-3
  - AES-Sboxes (8x8)
  - AES-Tboxes (8x32)
- Fugue
  - AES-Sboxes (8x8)
  - Fugue-Tboxes (8x24) and (8x32)
- Groestl
  - AES-Sboxes (8x8)
  - Groestl-Tboxes (8x40)
Block Memories used to implement ROM and Round Constants

- Hamsi
  - ROM in message expansion
    - $8 \times 4 \times 256 \times 32 = 256 \text{ kbit}$ in Hamsi-256

- Keccak, JH, SHA-2
  - Round constants only

- BLAKE
  - Permutation
DSP ADDERS
&
MULTIPLIERS
DSP Adders

- CubeHash
  - 32-bit addition

- Skein
  - 64-bit addition

- BMW
  - 32-bit or 64-bit Multioperand Addition

- BLAKE
  - 32-bit Addition

- SHA-2
  - 32-bit Multioperand Addition
PRELIMINARY RESULTS
## DSP Adders & Multipliers

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CubeHash</td>
<td>Basic</td>
<td>215.33</td>
<td>3445.00</td>
<td>707, 0, 0</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>234.41</td>
<td>3751.00</td>
<td>706, 0, 32</td>
</tr>
<tr>
<td>Skein</td>
<td>Basic</td>
<td>104.34</td>
<td>2812.00</td>
<td>1463, 0, 0</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>50.55</td>
<td>1362.15</td>
<td>1111, 0.64</td>
</tr>
<tr>
<td>BMW</td>
<td>Basic</td>
<td>10.89</td>
<td>5576.70</td>
<td>4400, 0, 0</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>5.29</td>
<td>2707.50</td>
<td>3267, 0, 144</td>
</tr>
<tr>
<td>Shabal</td>
<td>Basic</td>
<td>144.47</td>
<td>2958.68</td>
<td>275, 0, 0</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>208.55</td>
<td>4271.00</td>
<td>298, 0, 70</td>
</tr>
<tr>
<td>SIMD</td>
<td>Basic</td>
<td>40.89</td>
<td>2325.90</td>
<td>9288, 0, 0</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>33.73</td>
<td>1919</td>
<td>7880, 0, 96</td>
</tr>
</tbody>
</table>

- Throughput increases
- Throughput decreases

(underlined) - Throughput increases

(most likely as a result of design error)
# Block Memory & Adders

- Throughput increases
- Throughput decreases (most likely as a result of design error)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamsi</td>
<td>Basic</td>
<td>248.08</td>
<td>2646.00</td>
<td>946, 0, 0</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>224.07</td>
<td>2390.02</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>549, 32, 0</td>
</tr>
<tr>
<td>Keccak</td>
<td>Basic</td>
<td>238.38</td>
<td>10807.00</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>242.94</td>
<td>10952.70</td>
<td>1320, 2, 0</td>
</tr>
<tr>
<td>SHA-2</td>
<td>Basic</td>
<td>207.00</td>
<td>1630.00</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>169.12</td>
<td>1332.00</td>
<td>291, 1, 11</td>
</tr>
<tr>
<td>BLAKE</td>
<td>Basic</td>
<td>117.06</td>
<td>2853.91</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td>164.42</td>
<td>4008.7</td>
<td>928, 10, 24</td>
</tr>
<tr>
<td>JH</td>
<td>Basic</td>
<td>278.09</td>
<td>3955.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Embedded</td>
<td></td>
<td></td>
<td>1108, 0, 0</td>
</tr>
</tbody>
</table>
Results by Other Groups: Comprehensive Comparisons

Baldwin et al.

Institutions: University College Cork, Ireland
RMIT University, Melbourne, Australia
Queen’s University Belfast, Belfast, UK

Presented at: SHA-3 Candidate Conference 2010, FPL 2010

Matsuo et al.

Institutions: National Institute of Information and Communications Technology, Japan
Katholieke Universiteit Leuven, Belgium
Virginia Tech, USA
National Institute of Advanced Industrial Science and Technology, Japan
The University of Electro-Communications, Japan

Presented at: HOST 2010 and SHA-3 Candidate Conference 2010
Results by Other Groups: Comprehensive Comparisons

Guo et al.

Institutions: Virginia Tech, USA
Presented at: ePrint 2010/536
Results by Other Groups: Interesting Studies of Selected Algorithms

Savas et al.

Institutions: Sabanci University, Istanbul, Turkey
Presented at: SHA-3 Candidate Conference 2010

Pipelined Architectures of BMW, Keccak, and Luffa.

Detrey et al.

Institutions: LORIA, INRIA / CNRS / Nancy Université / SGDSN / ANSSI, France
Presented at: Selected Areas in Cryptography, SAC 2010

Excellent Implementation of Shabal
Differences: Padding

Padding in hardware:

**Baldwin et al.**: yes
- under assumption that the message ends on a boundary of a 32-bit word
- counters in the padding unit of the following functions form the critical path and thus affect the maximum clock frequency
  - Echo-256, Fugue-256/512, JH

**Other groups**: no

**GMU**: no results with padding yet
- universal padding circuit under development
  - arbitrary SHA-3 candidate, SHA-2, SHA-1
  - message allowed to end on a boundary of a word, byte, or bit
Differences: Interface (1)

Baldwin et al.:

- 32-bit input bus
- one clock (the same clock for processing and i/o)

As a result, the interface substantially limits the throughput of the following algorithms:

BMW, Echo, Grøstl, Keccak.

Matsuo et al., Guo et al.:

- compatible with SASEBO boards
- 16-bit input bus
- one clock (the same clock for processing and i/o)

As a result, the interface substantially limits the throughput for majority of algorithms.
Differences: Interface (2)

GMU:

- 64-bit input bus (for all algorithms except those with the block size = 32 bits)
- two clocks, if needed to assure that
  
  Load Time $\leq$ Processing Time

(only BMW for basic architectures)

- the interface does not restrict the speed of processing for any SHA-3 candidate
In order to make the comparison fair, we make the following assumptions:

**Baldwin et al.**
- Padding in Software
- Ideal Input-Output Bus

**Matsuo et al., Guo et al.**
- Ideal Input-Output Bus

Slightly favors other groups in terms of area, because our interface includes serial-to-parallel and parallel-to-serial converters, as well as message length counters.
AREA

Slices

Algorithm

SHA-2, BLAKE, BMW, CubeHash, ECHO, Fugue, Groesi, Hamsi, JH, Keccak, Luffa, Shabal, SHA-vite-3, SIMD, Skein
## Explanation of Remaining Differences

### Baldwin et al. vs. GMU

<table>
<thead>
<tr>
<th>Function</th>
<th>Baldwin</th>
<th>GMU</th>
<th>Differences Baldwin vs. GMU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blake</td>
<td>40</td>
<td>21</td>
<td>horizontally folded by 4 vs. horizontally folded by 2</td>
</tr>
<tr>
<td>BMW</td>
<td>4</td>
<td>1</td>
<td>unbalanced architecture</td>
</tr>
<tr>
<td>ECHO</td>
<td>8</td>
<td>25</td>
<td>basic iterative vs. reuse of BIG/SubBytes</td>
</tr>
<tr>
<td>Fugue</td>
<td>7</td>
<td>2</td>
<td>unbalanced architecture</td>
</tr>
<tr>
<td>Groestl</td>
<td>10</td>
<td>21</td>
<td>parallel execution vs. quasi-pipelined architecture</td>
</tr>
<tr>
<td>Hamsi</td>
<td>6</td>
<td>3</td>
<td>2 vs. 1 clock cycle per round</td>
</tr>
<tr>
<td>SIMD</td>
<td>32</td>
<td>9</td>
<td>basic architecture vs. 4x unrolled architecture</td>
</tr>
<tr>
<td>Function</td>
<td>Matsuo</td>
<td>GMU</td>
<td>Differences Matsuo vs. GMU</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>-----</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ECHO</td>
<td>99</td>
<td>25</td>
<td>4x vertically folded architecture vs. 3/2 horizontally folded architecture</td>
</tr>
<tr>
<td>Groestl</td>
<td>10</td>
<td>21</td>
<td>parallel execution vs. quasi-pipelined architecture</td>
</tr>
<tr>
<td>SIMD</td>
<td>46</td>
<td>9</td>
<td>basic architecture vs. 4x unrolled architecture; message expansion and main rounds performed sequentially vs. in parallel</td>
</tr>
</tbody>
</table>

**Keccak-256:** 1024-bit block size vs. 1088-bit block size

**Skein-256-256 vs. Skein-512-256:** 256 vs. 512-bit message block size and state size
Throughput resulting best Ratio

- Keccak
- ECHO
- Groesi
- BMW
- Luffa
- JH
- Fugue
- SHA-vite-3
- BLAKE
- CubeHash
- Hamsi
- SIMD
- Shabal
- SHA-2
- Skein
## 512-bit variant vs. 256-bit variant – Predicted Behavior

<table>
<thead>
<tr>
<th>Group</th>
<th>Area:</th>
<th>Thr:</th>
<th>Thr/Area:</th>
<th>512-bit variant vs. 256-bit variant – Predicted Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CubeHash, JH, Shabal, Skein</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BMW, SIMD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLAKE, Groestl, SHAvite-3, SHA-2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ECHO, Keccak</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hamsi, Luffa</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fugue</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hints for Designers of Hash Functions

- Easy way to predict **approximately** the change in speed and area when moving from a 256-bit to a 512-bit variant in **high-speed** hardware implementations

\[
\frac{\text{Area}(512)}{\text{Area}(256)} \approx \frac{\text{Datapath\_width}(512)}{\text{Datapath\_width}(256)} = \frac{\text{State\_size}(512)}{\text{State\_size}(256)}
\]

\[
\frac{\text{Thr}(512)}{\text{Thr}(256)} \approx \frac{\text{Block\_size}(512)}{\text{Block\_size}(256)} = \frac{\text{Round\_no}(512)}{\text{Round\_no\_ratio}} = \frac{\text{Round\_no}(256)}{\text{Block\_size\_ratio}}
\]
Pros and Cons of SHA-3 Candidates (1)

BLAKE
+ extremely flexible, multiple architectures
  obtained by horizontal, vertical, and mixed folding

BMW
+ good potential for pipelining
+ area efficient for high throughputs
- irregular structure
- difficulties with placing & routing
- quite complex folded architecture (late discovery),
  smaller but less efficient than the basic architecture
- need for an extra input/output clock
Pros and Cons of SHA-3 Candidates (2)

CubeHash
- small area
- good throughput to area ratio
- very suitable for parallel processing
- easy replacement for SHA-2 (similar in size and speed)
- relatively weak performance for short messages
- does not offer any significant performance advantage over SHA-2

ECHO
- very flexible in terms of vertical folding
- suitable for use of embedded block memories	only implement AES S-boxes and/or T-boxes
- good performance for short messages
- large area of the basic architecture
Pros and Cons of SHA-3 Candidates (3)

**Fugue**

+ very flexible in terms of vertical folding
+ suitable for use of embedded block memories
to implement AES S-boxes and T-boxes
- relatively slow for very short messages
- area grows and throughput decreases for a 512-bit variant

**Groestl**

+ suitable for quasi-pipelining (pipelining with one message)
+ high throughput and throughput to area ratios
+ very flexible in terms of vertical folding
+ suitable for use of embedded block memories
to implement AES S-boxes and T-boxes
- relatively large area of the basic architecture
Pros and Cons of SHA-3 Candidates (4)

Hamsi
+ suitable for use of embedded block memories to implement message expansion
- limited flexibility, no known folded architectures

JH
+/- good potential for folding but with limited area improvement

Keccak
+ very high throughput and throughput to area ratio
  especially for a 256-bit variant
+ good potential for pipelining
- limited flexibility, no known folded architectures
Pros and Cons of SHA-3 Candidates (5)

Luffa
+ very high throughput and throughput to area ratio
  for both 256 and 512-bit variant
+ good flexibility: straightforward folded architectures for medium-speed implementations

Shabal
+ extremely small area and high throughput to area ratio for Xilinx FPGAs
  (does not carry to Altera FPGAs or ASICs)
+ very suitable for parallel processing
- relatively small throughput of the basic architecture
- relatively weak performance for short messages
Pros and Cons of SHA-3 Candidates (6)

SHAvite-3
+ flexible in terms of vertical folding
+ suitable for use of embedded block memories
to implement AES S-boxes and T-boxes
- complex key scheduling, difficult to fold or unroll

SIMD
- big area of the basic architecture
- by far the worst throughput to area ratio
- most time consuming to implement and debug
- complex message expansion unit
+ good potential for folding
Pros and Cons of SHA-3 Candidates (7)

Skein

+ good potential for pipelining

- relatively small throughput of the basic architecture before pipelining
More About our Designs & Tools

- CHES 2010 paper
  - Methodology
  - Results for 256-bit variants
- FPL 2010 paper
  - ATHENa features
  - Case studies

- Cryptology e-Print Archive, 2010/445, last updated on Oct. 10, 2010
  - Detailed hierarchical block diagrams,
    - 60 diagrams for 15 functions
  - Corresponding formulas for execution time and throughput

- ATHENa web site
  - Most recent results
  - Comparisons with results from other groups
  - Optimum options of tools
Thank you!

Questions?

CERG: http://cryptography.gmu.edu

ATHENa: http://cryptography.gmu.edu/athena