GMU SHA Core Interface & Hash Function Performance Metrics
**Why Interface Matters?**

- **Pin limit**
  
  Total number of i/o ports \( \leq \) Total number of an FPGA i/o pins

- **Support for the maximum throughput**
  
  Time to load the next message block \( \leq \) Time to process current block

---

**Interface: Two possible solutions**

<table>
<thead>
<tr>
<th>length of the message communicated at the beginning</th>
<th>msg_bitlen</th>
<th>end_of_msg</th>
<th>SHA core</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>message</td>
<td>end_of_msg</td>
<td>SHA core</td>
</tr>
<tr>
<td>zero_word</td>
<td>zero_word</td>
<td>SHA core</td>
<td>SHA core</td>
</tr>
</tbody>
</table>

- **easy to implement**
  - passive source circuit

- **area overhead for the counter of message bits**

- **more intelligent source circuit required**

- **no need for internal message bit counter**
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

SHA Core Interface
SHA Core Interface + Surrounding FIFOs

Input FIFO
- clk, rst
- ext_idata
- fifoin_empty
- fifoin_full
- fifoin_write
- fifoin_read
- din, dout
- iodata
- src_ready, src_read
- dst_ready, dst_write

SHA core
- clk, rst
- iodata
- src_ready, dst_ready
- fifoin_empty
- fifoin_read
- din, dout
- w, w
- w

Output FIFO
- clk, rst
- ext_odata
- fifoout_empty
- fifoout_full
- fifoout_write
- fifoout_read
- din, dout
- full, empty
- write, read
- w

Operation of FIFO

<table>
<thead>
<tr>
<th>clk</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>din</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dout</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>empty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFO operation</td>
<td>write</td>
<td>write</td>
<td>read</td>
<td>write</td>
<td>write</td>
<td>read</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>FIFO data</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Communication Protocol for Unpadded Messages

a) $w$ bits

- $\text{msg\_bitlen}$
- $\text{message}$
- $\text{zero\_word}$

b) $w$ bits

- $\text{seg\_0\_bitlen}$
- $\text{seg\_0}$
- $\text{seg\_1\_bitlen}$
- $\text{seg\_1}$
- ...
- $\text{seg\_n-1\_bitlen}$
- $\text{seg\_n-1}$
- $\text{zero\_word}$

SHA Core Interface with Additional Faster I/O Clock

- $\text{io\_clk}$
- $\text{clk}$
- $\text{rst}$

- $\text{SHA core}$
- $\text{io\_clk}$
- $\text{clk}$
- $\text{rst}$
- $\text{dout}$
- $\text{dout}$
- $\text{src\_ready}$
- $\text{dst\_ready}$
- $\text{src\_read}$
- $\text{dst\_write}$
**SHA Core Interface with Two Clocks + Surrounding FIFOs**

**Communication Protocol for Padded Messages Without Message Splitting**

msg_len_ap – message length after padding [bits]
msg_len_bp – message length before padding [bits]
**Communication Protocol for Padded Messages With Message Splitting**

<table>
<thead>
<tr>
<th>w bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>seg_0 len_ap</td>
</tr>
<tr>
<td>seg_0</td>
</tr>
<tr>
<td>seg_1 len_ap</td>
</tr>
<tr>
<td>seg_1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>seg_n-1 len_ap</td>
</tr>
<tr>
<td>seg_n-1 len_bp</td>
</tr>
<tr>
<td>seg_n-1</td>
</tr>
</tbody>
</table>

seg_i_len_ap – segment i length after padding* [bits]

seg_i_len_bp – segment i length before padding [bits]

* For all $i < n-1$ segment i length after padding is assumed to be a multiple of the message block size, $b$ [characteristic to each function], and thus also the word size, $w$. The last segment cannot consist of only padding bits. It must include at least one message bit.

---

**Performance Metrics**
Performance Metrics - Speed

Throughput for Long Messages [Mbit/s]
Throughput for Short Messages [Mbit/s]
Execution Time for Short Messages [ns]

Allows for easy cross-comparison among implementations in software (microprocessors), FPGAs (various vendors), ASICs (various libraries)

Performance Metrics - Speed

Time to hash N blocks of message [cycles] = Htime(N)

The exact formula from analysis of a block diagram, confirmed by functional simulation.

Minimum Clock Period [ns] = T

From a place & route and/or static timing analysis report file.
Time to Hash N Blocks of the Message [clock cycles]

\[ HTime(N) = c_{INIT} + \left[ c_{IN}/r_{IO} \right] + c_{BLOCK} \cdot N + c_{FINAL} + \left[ c_{OUT}/r_{IO} \right] \]

- \( c_{INIT} \) is the number of clock cycles necessary to establish communication with the source of data (typically, Input FIFO) and read the length of the message (in our formulas we assume that the length of the message is smaller than 2\(^8\)).
- \( c_{IN} \) is the number of clock cycles required to read the very first block of the message. \( c_{IN} = Block\_size/w \).
- \( c_{BLOCK} \) is the number of clock cycles required to process one block of the message.
- \( c_{FINAL} \) is the number of clock cycles required for the finalization. We assume that only one finalization is required per entire message (if the finalization needs to be repeated for every block of the message, its number of clock cycles is included in \( c_{BLOCK} \)).
- \( c_{OUT} \) is the number of clock cycles required to write hash value to the destination circuit (typically Output FIFO). \( c_{OUT} = output\_size/w \).

Performance Metrics - Speed

Minimum time to hash N blocks of message [ns] = \( Htime(N) \cdot T \)

Maximum Throughput (for long messages) = \[
\frac{block\_size}{T \cdot (Htime(N+1) - Htime(N))}
\]

= \[
\frac{block\_size}{T \cdot block\_processing\_time}
\]

Effective maximum throughput for short messages:

\[ Throughput_{eff} = \frac{N \cdot block\_size}{(T \cdot HTime(N))} \]
Performance Metrics - Speed

Maximum Throughput (for long messages) = \( \frac{\text{block_size}}{T \times \text{block_processing_time}} \)

- From specification
- From place & route report and/or static timing analysis report
- From analysis of block diagram and/or functional simulation

Performance Metrics - Area

\[ \text{Resource Utilization}_{\text{Spartan3}} = (\#\text{CLB slices}, \#\text{BRAMs}, \#\text{MULs}) \]
\[ \text{Resource Utilization}_{\text{Cyclone III}} = (\#\text{LE}, \#\text{memory_bits}, \#\text{MULs}). \]

For the basic, folded, and unrolled architectures, we force these vectors to look as follows through the synthesis and implementation options:

\[ \text{Resource Utilization}_{\text{Spartan3}} = \begin{bmatrix} \#\text{CLB slices} \\ 0 \\ 0 \end{bmatrix} \]
\[ \text{Resource Utilization}_{\text{Cyclone III}} = \begin{bmatrix} \#\text{LE} \\ 0 \\ 0 \end{bmatrix} \]
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
- leads to high-speed, close-to-maximum-throughput designs

Our Design Flow

```
  Specification
     ↓ Datapath
     ↓ Block diagram
  Interface
     ↓ Controller
     ↓ ASM Chart
     ↓ VHDL Code
  Controller Template
  Library of Basic Components
  Formulas for
  Throughput & Hash time
  ↓ Max. Clock Freq.
  ↓ Resource Utilization
  ↓ Throughput, Area, Throughput/Area, Hash Time for Short Messages
```
How to compare hardware speed vs. software speed?

EBASH reports (http://bench.cr yp.to/results-hash.html)

In graphs

Time(n) = Time in clock cycles vs. message size in bytes for n-byte messages, with n=0, 1, 2, 3, ... 2048, 4096

In tables

Performance in cycles/byte for n=8, 64, 576, 1536, 4096, long msg

\[
\text{Performance for long message} = \frac{\text{Time(4096)} - \text{Time(2048)}}{2048}
\]

Throughput [Gbit/s] = \( \frac{8 \text{ bits/byte} \cdot \text{clock frequency [GHz]}}{\text{Performance for long message [cycles/byte]}} \)