C vs. VHDL: Comparing Performance of CAESAR Candidates Using High-Level Synthesis on Xilinx FPGAs

Ekawat Homsirikamol,
William Diehl, Ahmed Ferozpuri,
Farnoud Farahmand,
and Kris Gaj
George Mason University
USA

http://cryptography.gmu.edu
https://cryptography.gmu.edu/athena
Primary Support for This Particular Project

Ekawat Homsirikamol
a.k.a “Ice”

Working on the PhD Thesis entitled
“A New Approach to the Development of Cryptographic Standards Based on the Use of High-Level Synthesis Tools”

RTL codes developed by:
William Diehl,
Farnoud Farahmand,
Ahmed Ferozpuri, and Ekawat Homsirikamol.
Cryptographic Standard Contests

IX.1997  X.2000

AES

15 block ciphers → 1 winner

1.2000

NESSIE  XII.2002

CRYPTREC  XI.2004

4 HW winners
+ 4 SW winners

34 stream ciphers

IV.2008

eSTREAM

X.2007

51 hash functions → 1 winner

XI.2004

SHA-3

X.2012

57 authenticated ciphers → multiple winners

V.2008

CAESAR  XII.2017

time
Evaluation Criteria

Security

Software Efficiency
μProcessors  μControllers

Hardware Efficiency
FPGAs  ASICs

Flexibility
Simplicity
Licensing
Traditional Development & Benchmarking Flow

Informal Specification

Manual Design

HDL Code

Manual Optimization

FPGA Tools

Netlist

Functional Verification

Timing Verification

Test Vectors

Post Place & Route Results
Extended Traditional Development & Benchmarking Flow

Informal Specification

Manual Design

HDL Code

Automated Optimization

FPGA Tools

Netlist

Functional Verification

Xilinx ISE + ATHENa
Vivado + Default Strategies

Timing Verification

Test Vectors

Post Place & Route Results
Remaining Difficulties of Hardware Benchmarking

• Large number of candidates
• Long time necessary to develop and verify RTL (Register-Transfer Level) Hardware Description Language (HDL) codes
• Multiple variants of algorithms (e.g., multiple key, nonce, and tag sizes)
• High-speed vs. lightweight algorithms
• Multiple hardware architectures
• Dependence on skills of designers
High-Level Synthesis (HLS)

- High Level Language (e.g. C, C++, SystemC)
- High-Level Synthesis
- Hardware Description Language (e.g., VHDL or Verilog)
Short History of High-Level Synthesis

Generation 1 (1980s-early 1990s): research period

Generation 2 (mid 1990s-early 2000s):
- Commercial tools from Synopsys, Cadence, Mentor Graphics, etc.
- Input languages: behavioral HDLs  Target: ASIC

Outcome: Commercial failure

Generation 3 (from early 2000s):
- Domain oriented commercial tools: in particular for DSP
- Input languages: C, C++, C-like languages (Impulse C, Handel C, etc.), Matlab + Simulink, Bluespec
- Target: FPGA, ASIC, or both

Outcome: First success stories
AutoESL Design Technologies, Inc. (25 employees)

Flagship product:

AutoPilot, translating C/C++/System C to VHDL or Verilog

• Acquired by the biggest FPGA company, Xilinx Inc., in 2011
• AutoPilot integrated into the primary Xilinx toolset, Vivado, as Vivado HLS, released in 2012

“High-Level Synthesis for the Masses”
Our Hypotheses

- **Ranking** of candidate algorithms in cryptographic contests in terms of their performance in modern FPGAs & All-Programmable SoCs will remain the same independently whether the HDL implementations are *developed manually* or *generated automatically* using High-Level Synthesis tools.

- The development time will be reduced by at least an order of magnitude.
Potential Additional Benefits

Early feedback for designers of cryptographic algorithms

• Typical design process based only on security analysis and software benchmarking
• Lack of immediate feedback on hardware performance
• Common unpleasant surprises, e.g.,
  ▪ Mars in the AES Contest
  ▪ BMW, ECHO, and SIMD in the SHA-3 Contest
Proposed HLS-Based Development and Benchmarking Flow

Reference Implementation in C

Manual Modifications (pragmas, tweaks)

HLS-ready C code

High-Level Synthesis

HDL Code

Automated Optimization

FPGA Tools

Netlist

Functional Verification

Xilinx ISE + ATHENa Vivado + Default Strategies

Timing Verification

Test Vectors

Post Place & Route Results
Examples of Source Code Modifications

Unrolling of loops:

```c
for (i = 0; i < 4; i++)
    #pragma HLS UNROLL
    for (j = 0; j < 4; j++)
        #pragma HLS UNROLL
        b[i][j] = s[i][j];
```

Flattening function's hierarchy:

```c
void KeyUpdate (word8 k[4][4],
                word8 round)
{
    #pragma HLS INLINE
    ...
}
```

Function Reuse:

```c
// (a) Before modification
for(round=0; round<NB_ROUNDS; ++round)
{
    if (round == NB_ROUNDS-1)
        single_round(state, 1);
    else
        single_round(state, 0);
}
```

```c
// (b) After modification
for(round=0; round<NB_ROUNDS; ++round)
{
    if (round == NB_ROUNDS-1)
        x = 1;
    else
        x = 0;
    single_round(state, x);
}
```
Our Test Case

- 8 Round 1 CAESAR candidates + current standard AES-GCM
- Basic iterative architecture
- GMU AEAD Hardware API
- Implementations developed in parallel using RTL and HLS methodology
- 2-3 RTL implementations per student, all HLS implementations developed by a single student (Ice)
- Starting point: Informal specifications and reference software implementations in C provided by the algorithm authors
- Post P&R results generated for
  - Xilinx Virtex 6 using Xilinx ISE + ATHENA, and
  - Virtex 7 and Zynq 7000 using Xilinx Vivado with 26 default option optimization strategies
- No use of BRAMs or DSP Units in AEAD Core
### Parameters of Authenticated Ciphers

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key size</th>
<th>Nonce size</th>
<th>Tag size</th>
<th>Basic Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Block Cipher Based</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES-COPA</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>AES</td>
</tr>
<tr>
<td>AES-GCM</td>
<td>128</td>
<td>96</td>
<td>128</td>
<td>AES</td>
</tr>
<tr>
<td>CLOC</td>
<td>128</td>
<td>96</td>
<td>128</td>
<td>AES</td>
</tr>
<tr>
<td>POET</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>AES</td>
</tr>
<tr>
<td>SCREAM</td>
<td>128</td>
<td>96</td>
<td>128</td>
<td>TLS</td>
</tr>
<tr>
<td><strong>Permutation Based</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICEPOLE</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>Keccak-like</td>
</tr>
<tr>
<td>Keyak</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>Keccak-f</td>
</tr>
<tr>
<td>PRIMATEs- GIBBON</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>PRIMATE</td>
</tr>
<tr>
<td>PRIMATEs- HANUMAN</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>PRIMATE</td>
</tr>
</tbody>
</table>
## Parameters of Ciphers & GMU Implementations

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Word Size, w</th>
<th>Block Size, b</th>
<th>#Rounds</th>
<th>Cycles/Block RTL</th>
<th>Cycles/Block HLS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Block-cipher Based</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES-COPA</td>
<td>32</td>
<td>128</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>AES-GCM</td>
<td>32</td>
<td>128</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>CLOC</td>
<td>32</td>
<td>128</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>POET</td>
<td>32</td>
<td>128</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>SCREAM</td>
<td>32</td>
<td>128</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td><strong>Permutation Based</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICEPOLE</td>
<td>256</td>
<td>1024</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Keyak</td>
<td>128</td>
<td>1344</td>
<td>12</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>PRIMATEs- GIBBON</td>
<td>40</td>
<td>40</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>PRIMATEs- HANUMAN</td>
<td>40</td>
<td>40</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>
Datapath vs. Control Unit

Datapath Determines
- Area
- Clock Frequency

Control Unit Determines
- Number of clock cycles
Encountered Problems

Control Unit suboptimal

- Difficulty in inferring an overlap between completing the last round and reading the next input block
- One additional clock cycle used for initialization of the state at the beginning of each round
- The formulas for throughput:

  HLS: Throughput = Block_size / ((#Rounds+2) * T_{CLK})

  RTL: Throughput = Block_size / (#Rounds+C * T_{CLK})

  C=0, 1 depending on the algorithm
RTL vs. HLS Clock Frequency in Zynq 7000

![Graph showing the comparison between RTL and HLS clock frequencies for various benchmarks in Zynq 7000. The graph indicates that HLS generally achieves higher clock frequencies than RTL for most benchmarks. The legend on the right lists the benchmarks and their respective clock frequencies.]
RTL vs. HLS Throughput in Zynq 7000

![Graph showing throughput comparison between RTL and HLS for different designs.]
RTL vs. HLS Ratios in Zynq 7000

Clock Frequency

Throughput

<table>
<thead>
<tr>
<th></th>
<th><em>PRIMATES HUMAN</em></th>
<th><em>ICEPOLE</em></th>
<th><em>Keyak</em></th>
<th><em>SCREAM</em></th>
<th><em>AES-GCM</em></th>
<th><em>PRIMATES GIBBON</em></th>
<th><em>POET</em></th>
<th><em>CLOC</em></th>
<th><em>AES-COPA</em></th>
<th><em>ICEPOLE</em></th>
<th><em>Keyak</em></th>
<th><em>PRIMATES HUMAN</em></th>
<th><em>PRIMATES GIBBON</em></th>
<th><em>SCREAM</em></th>
<th><em>AES-GCM</em></th>
<th><em>POET</em></th>
<th><em>CLOC</em></th>
<th><em>AES-COPA</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1.08</td>
<td>1.04</td>
<td>1.01</td>
<td>0.94</td>
<td>0.94</td>
<td>0.92</td>
<td>0.90</td>
<td>0.89</td>
<td>0.89</td>
<td>1.39</td>
<td>1.17</td>
<td>1.16</td>
<td>1.05</td>
<td>1.03</td>
<td>1.02</td>
<td>0.98</td>
<td>0.97</td>
<td>0.97</td>
</tr>
</tbody>
</table>
RTL vs. HLS #LUTs in Zynq 7000
RTL vs. HLS Throughput/#LUTs in Zynq 7000
RTL vs. HLS Ratios in Zynq 7000
Throughput vs. LUTs in Zynq 7000

- **RTL**

- **HLS**

![Graph showing throughput vs. LUTs for RTL and HLS](image)
RTL vs. HLS Throughput

![Graph showing throughput comparison between RTL and HLS for Virtex 6, Virtex 7, and Zynq]
RTL vs. HLS #LUTs
RTL vs. HLS Throughput/#LUTs

![Graph showing the comparison of throughput to area for different algorithms across different FPGA types: Virtex 6, Virtex 7, and Zynq. The algorithms include ICEPOLE, Keyak, AES-GCM, CLOC, PRIMATEs, GIBBON, PRIMATEs HANUMAN, SCREAM, POET, and AES-COPA. The graph illustrates the trade-off between throughput and area for each algorithm across the different FPGA types.]
ATHENa Database of Results for Authenticated Ciphers

- Available at http://cryptography.gmu.edu/athena
- Developed by John Pham, a Master’s-level student of Jens-Peter Kaps
- Results can be entered by designers themselves. If you would like to do that, please contact me regarding an account.
- The ATHENa Option Optimization Tool supports automatic generation of results suitable for uploading to the database
Ordered Listing with a Single-Best (Unique) Result per Each Algorithm

## Database of FPGA Results for Authenticated Ciphers

### Show Help

- Compare Selected

Show 25 entries

<table>
<thead>
<tr>
<th>Result ID</th>
<th>Algorithm</th>
<th>Key Size [bits]</th>
<th>Implementation Approach</th>
<th>Platform</th>
<th>Family</th>
<th>Timing [Mbits/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>ICEPOLE</td>
<td>128</td>
<td>HLS</td>
<td>Virtex 7</td>
<td></td>
<td>32,902</td>
</tr>
<tr>
<td>107</td>
<td>Keyak</td>
<td>128</td>
<td>HLS</td>
<td>Virtex 7</td>
<td></td>
<td>22,594</td>
</tr>
<tr>
<td>97</td>
<td>AES-GCM</td>
<td>128</td>
<td>HLS</td>
<td>Virtex 7</td>
<td></td>
<td>3,015</td>
</tr>
<tr>
<td>101</td>
<td>CLOC</td>
<td>128</td>
<td>HLS</td>
<td>Virtex 7</td>
<td></td>
<td>2,459</td>
</tr>
<tr>
<td>111</td>
<td>POET</td>
<td>128</td>
<td>HLS</td>
<td>Virtex 7</td>
<td></td>
<td>1,795</td>
</tr>
<tr>
<td>93</td>
<td>AES-COPA</td>
<td>128</td>
<td>HLS</td>
<td>Virtex 7</td>
<td></td>
<td>1,670</td>
</tr>
<tr>
<td>116</td>
<td>PRIMATEs-GIBBON</td>
<td>120</td>
<td>HLS</td>
<td>Virtex 7</td>
<td></td>
<td>1,590</td>
</tr>
<tr>
<td>89</td>
<td>SCREAM</td>
<td>128</td>
<td>HLS</td>
<td>Virtex 7</td>
<td></td>
<td>1,414</td>
</tr>
<tr>
<td>121</td>
<td>PRIMATEs-HANUMAN</td>
<td>120</td>
<td>HLS</td>
<td>Virtex 7</td>
<td></td>
<td>809</td>
</tr>
</tbody>
</table>

- Result ID
- Algorithm
- Key Size [bits]
- HLS
- Timing [Mbits/s]
Details of Result ID 97

Algorithm
- IV or Nonce Size [bits]: 96
- Transformation Category: Cryptographic
- Transformation: Authenticated Cipher
- Group: Standards
- Algorithm: AES-GCM
- Tag Size [bits]: 128
- Associated Data Support: -
- Key Size [bits]: 128
- Secret Message Number: -
- Secret Message Number Size [bits]: -
- Message Block Size [bits]: 128
- Other Parameters: -
- Specification: SP-800-38D.pdf
- Formula for Message Size After Padding: -

Design
- Design ID: 21
- Impl Approach: HLS
- Hardware API: GMU_AESD_Core_API_V1
- Primary Optimization Target: Throughput/Area
- Secondary Optimization Target: -
- Architecture Type: Basic Iterative
- Description Language: VHDL
- Use of Megafunctions or Primitives: No
- List of Megafunctions or Primitives: -
- Maximum Number of Streams Processed in Parallel: 1
- Number of Clock Cycles per Message Block in a Long Message: 12
- Datapath Width [bits]: 128
- Padding: Yes
- Minimum Message Unit: -
- Input Bus Width [bits]: 32
- Output Bus Width [bits]: 32
### Comparison of Result #s 95 and 97

#### Algorithm

<table>
<thead>
<tr>
<th></th>
<th>96</th>
<th>95</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV or Nonce Size [bits]:</td>
<td>Cryptographic</td>
<td>Cryptographic</td>
</tr>
<tr>
<td>Transformation Category:</td>
<td>Authenticated Cipher</td>
<td>Authenticated Cipher</td>
</tr>
<tr>
<td>Transformation:</td>
<td>Standards</td>
<td>Standards</td>
</tr>
<tr>
<td>Group:</td>
<td>AES-GCM</td>
<td>AES-GCM</td>
</tr>
<tr>
<td>Algorithm:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Tag Size [bits]:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Associated Data Support:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Key Size [bits]:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Secret Message Number Size [bits]:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Secret Message Number Size [bits]:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Message Block Size [bits]:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Other Parameters:</td>
<td>SP-800-38D.pdf</td>
<td>SP-800-38D.pdf</td>
</tr>
<tr>
<td>Specification:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formula for Message Size After Padding:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Design

<table>
<thead>
<tr>
<th></th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impl Approach:</td>
<td>RTL</td>
<td>HLS</td>
</tr>
<tr>
<td>Hardware API:</td>
<td>GMU_AEAD_Core_API_v1</td>
<td>GMU_AEAD_Core_API_v1</td>
</tr>
<tr>
<td>Primary Optimization Target:</td>
<td>Throughput/Area</td>
<td>Throughput/Area</td>
</tr>
<tr>
<td>Secondary Optimization Target:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture Type:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description Language:</td>
<td>Basic Iterative</td>
<td>Basic Iterative</td>
</tr>
<tr>
<td>Use of Megafunctions or Primitives:</td>
<td>VHDL</td>
<td>VHDL</td>
</tr>
<tr>
<td>List of Megafunctions or Primitives:</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Maximum Number of Streams Processed in Parallel:</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Number of Clock Cycles per Message Block in a Long Message:</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Datapath Width [bits]:</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Padding:</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Minimum Message Unit:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input Bus Width [bits]:</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Platform</td>
<td>Xilinx</td>
<td>Xilinx</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>Device</td>
<td>xc7vx485tffg1761-2</td>
<td>xc7vx485tffg1761-2</td>
</tr>
<tr>
<td>Timing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encryption/Authentication Throughput [Mbits/s]:</td>
<td>3261</td>
<td>3015</td>
</tr>
<tr>
<td>Decryption/Authentication Throughput [Mbits/s]:</td>
<td>3261</td>
<td>3015</td>
</tr>
<tr>
<td>Authentication-Only Throughput [Mbits/s]:</td>
<td>3261</td>
<td>3015</td>
</tr>
<tr>
<td>Synthesis Clock Frequency [MHz]:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Key Scheduling Time [ns]:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Requested Implementation Clock Frequency [MHz]:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Implementation Clock Frequency [MHz]:</td>
<td>280.27</td>
<td>282.65</td>
</tr>
<tr>
<td>(Encryption/Authentication Throughput)/LUT [(Mbits/s)/LUT]:</td>
<td>0.909</td>
<td>0.879</td>
</tr>
<tr>
<td>(Encryption/Authentication Throughput)/Slice [(Mbits/s)/Slice]:</td>
<td>2.797</td>
<td>2.728</td>
</tr>
<tr>
<td>(Decryption/Authentication Throughput)/LUT [(Mbits/s)/LUT]:</td>
<td>0.909</td>
<td>0.879</td>
</tr>
<tr>
<td>(Decryption/Authentication Throughput)/Slice [(Mbits/s)/Slice]:</td>
<td>2.797</td>
<td>2.728</td>
</tr>
<tr>
<td>(Auth-Only Throughput)/LUT [(Mbits/s)/LUT]:</td>
<td>0.909</td>
<td>0.879</td>
</tr>
<tr>
<td>(Auth-Only Throughput)/Slice [(Mbits/s)/Slice]:</td>
<td>2.797</td>
<td>2.728</td>
</tr>
<tr>
<td>Resource Utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLB Slices:</td>
<td>1166</td>
<td>1105</td>
</tr>
<tr>
<td>LUTs:</td>
<td>3588</td>
<td>3430</td>
</tr>
<tr>
<td>Flip Flops:</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DSPs:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BRAMs:</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Conclusions

• High-level synthesis offers a potential to facilitate hardware benchmarking during the design of cryptographic algorithms and at the early stages of cryptographic contests

• Case study based on 8 Round 1 CAESAR candidates & AES-GCM demonstrated correct ranking for majority of candidates using all major performance metrics

• More research needed to overcome remaining difficulties
  • Suboptimal control unit
  • Wide range of RTL to HLS performance metric ratios
  • Efficient and reliable generation of HLS-ready C codes
Thank you!

Comments?

Questions?

Suggestions?

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