Toward a New HLS-Based Methodology for FPGA Benchmarking of Candidates in Cryptographic Competitions: The CAESAR Contest Case Study

Ekawat Homsirikamol and Kris Gaj
George Mason University
USA

Based on work partially supported by the National Science Foundation under Grant No. 1314540
Ekawat Homsirikamol
a.k.a “Ice”

Ph.D. Thesis Defense
November 2016

Currently with Cadence Design Systems,
San Jose, California
Cryptography is Everywhere!

- Software updates
- Mobile phones connecting to cell towers
- Credit/debit card authorizations
- On-line payments and tax declarations
- Skype
- WhatsApp, iMessage
- ePassports

- Crypto-currencies (e.g., Bitcoin)
- Cloud computing
- Internet of Things, etc.
Need for Standards

Cryptographic Contests

IX.1997  X.2000
AES

I.2000  XII.2002
NESSIE  CRYPTREC

XI.2004  IV.2008
eSTREAM

X.2007  X.2012
SHA-3

I.2013  TBD
CAESAR

15 block ciphers → 1 winner

34 stream ciphers → 4 HW winners + 4 SW winners

51 hash functions → 1 winner

57 authenticated ciphers → multiple winners
Evaluation Criteria

Security

Software Efficiency
- µProcessors
- µControllers

Hardware Efficiency
- FPGAs
- ASICs

Flexibility

Simplicity

Licensing
AES Final Round: 5 candidates

GMU FPGA Results

Speed [Mbit/s]

Straw Poll @ AES 3 conference

# votes

Rijndael second best in FPGAs,
selected as a winner due to much better performance
in software
SHA-3 Round 2: 14 candidates

Throughput vs. Area: Normalized to Results for SHA-2 and Averaged over 7 FPGA Families

Best: Fast & Small

Worst: Slow & Big
<table>
<thead>
<tr>
<th></th>
<th>Initial number of candidates</th>
<th>Implemented in hardware</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>15</td>
<td>5</td>
<td>33.3%</td>
</tr>
<tr>
<td>eSTREAM</td>
<td>34</td>
<td>8</td>
<td>23.5%</td>
</tr>
<tr>
<td>SHA-3</td>
<td>51</td>
<td>14</td>
<td>27.5%</td>
</tr>
<tr>
<td>CAESAR</td>
<td>57</td>
<td>28</td>
<td>49.1%</td>
</tr>
<tr>
<td>PQC</td>
<td>69</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
High-Level Synthesis (HLS)

High Level Language
(C, C++, Java, Python, etc.)

High-Level Synthesis

Hardware Description Language
(VHDL or Verilog)
Case for HLS in Crypto Competitions

- All submissions include reference implementations in C
- Development time potentially decreased several times
- All candidates can be implemented by the same group, and even the same designer, reducing the bias
- Results from High-Level Synthesis could have a large impact in early stages of the competitions and help narrow down the search (saving thousands of man-hours of cryptanalysis)
- Potential for quickly detecting suboptimal code written manually, using Register Transfer Level (RTL) approach
Popular HLS Tools

Commercial (FPGA-oriented):

• **Vivado HLS**: Xilinx – selected for this study
• **FPGA SDK for OpenCL**: Intel

Academic:

• **Bambu**: Politecnico di Milano, Italy
• **DWARV**: Delft University of Technology, The Netherlands
• **GAUT**: Universite de Bretagne-Sud, France
• **LegUp**: University of Toronto, Canada
## Cryptographic Benchmarks

<table>
<thead>
<tr>
<th>Tools</th>
<th>aes-encrypt</th>
<th>aes-decrypt</th>
<th>sha</th>
<th>blowfish</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Best HLS</strong></td>
<td>1,191</td>
<td>2,579</td>
<td>51,399</td>
<td>57,590</td>
</tr>
<tr>
<td><strong>Manual RTL</strong></td>
<td>20</td>
<td>20</td>
<td>20,480</td>
<td>18,736</td>
</tr>
<tr>
<td><strong>Best HLS</strong></td>
<td>60</td>
<td>129</td>
<td>2.5</td>
<td>3.1</td>
</tr>
</tbody>
</table>

**Best HLS:** Best result (minimum number of clock cycles) from among those generated by Vivado HLS, Bambu, DWARV, and LegUp

GMU Case Studies

- **AES**, winner
  ReConFig 2014, Dec. 2014

- **5 Final SHA_3** Candidates + SHA-2
  Applied Reconfigurable Computing,
  ARC 2015, Bochum, Apr. 2015

- **15 Round 3 CAESAR** Candidates
  + AES-GCM

This Talk

Ekawat Homsirikamol
a.k.a “Ice”
ReConFig 2014: AES

HLS/Manual ratios:

- Clock cycles: $\frac{12}{10} = 1.20$
- Area: $\frac{343}{354} = 0.97$

Manual/HLS ratios:

- Frequency: $\frac{230}{231} = 0.996$
- Throughput: $\frac{2943}{2467} = 1.19$
- Throughput/Area: $\frac{8.31}{7.19} = 1.16$
ARC 2015: SHA-3 Candidates Revisited

Alterra Stratix III FPGA

Throughput vs. Area

RTL

HLS

SHA-3 Candidates:
- BLAKE
- Groestl
- JH
- Keccak
- Skein
- SHA-2

Graphs showing performance metrics for different cryptographic algorithms on Alterra Stratix III FPGA.
Our Hypothesis

- **Ranking** of candidates in cryptographic contests in terms of their performance in modern FPGAs 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 a factor of 3 to 10.

- This hypothesis should apply to at least AES Contest, SHA-3 Contest, CAESAR Contest.
CAESAR Case Study

• **GMU HLS-ready C Code**
  • 15 Round 3 CAESAR Candidate Variants
  • AES-GCM

• **GMU RTL VHDL Code**
  • 10 Round 3 CAESAR Candidates Variants
  • AES-GCM

• **NTU Singapore RTL VHDL Code**
  • ACORN, JAMBU-SIMON, MORUS

• **NEC Japan RTL VHDL Code**
  • AES-OTR

• **Ketje-Keyak RTL VHDL Code**
  • KetjeSr
• Uniform Hardware API
• Uniform PreProcessor & PostProcessor
• Uniform Benchmarking Methodology
• Two Platforms
  • Xilinx Virtex 6
  • Xilinx Virtex 7
HLS vs. RTL Ratios for Number of Clock Cycles

\[1.20 \ldots 1.00\] < 2.5
### Throughput RTL / Throughput HLS for Xilinx Virtex-7

<table>
<thead>
<tr>
<th>Component</th>
<th>Throughput RTL</th>
<th>Throughput HLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORX norx3241v3</td>
<td>1.35</td>
<td></td>
</tr>
<tr>
<td>COLM colmv1</td>
<td>1.26</td>
<td></td>
</tr>
<tr>
<td>Deoxys deoxysi128v141</td>
<td>1.21</td>
<td></td>
</tr>
<tr>
<td>AEGIS aegis128i</td>
<td>1.20</td>
<td></td>
</tr>
<tr>
<td>JAMBU-SIMON jambusimon96v2</td>
<td>1.19</td>
<td></td>
</tr>
<tr>
<td>OCB</td>
<td>1.14</td>
<td></td>
</tr>
<tr>
<td>ACORN acorn128v3</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Tiaoxin tiaoxinv2</td>
<td>0.99</td>
<td></td>
</tr>
<tr>
<td>MORUS morus1280128V2</td>
<td>0.97</td>
<td></td>
</tr>
<tr>
<td>SILEM silc128n128silc2v1</td>
<td>0.96</td>
<td></td>
</tr>
<tr>
<td>Ascon ascon128av12</td>
<td>0.95</td>
<td></td>
</tr>
<tr>
<td>CLOC aes128n128clocv2</td>
<td>0.83</td>
<td></td>
</tr>
<tr>
<td>AES-GCM aes128gcmv1</td>
<td>0.81</td>
<td></td>
</tr>
<tr>
<td>JAMBU-AES aesjambuv2</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td>OTR aes128otrpv3</td>
<td>0.70</td>
<td></td>
</tr>
<tr>
<td>Ketje ketjesrv2</td>
<td>0.59</td>
<td></td>
</tr>
</tbody>
</table>

- **Suboptimal HLS**
  - Throughput HLS > 1.30
  - Throughput HLS < 0.70
- **RTL may be improved**
  - Throughput RTL [0.90..0.70]
- **RTL is acceptable**
  - Throughput RTL [1.30..0.90]
RTL vs. HLS Throughput [Mbits/s]

Consistently better than AES-GCM

Suboptimal RTL
<table>
<thead>
<tr>
<th>Name</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORX norx3241v3</td>
<td>1.67</td>
</tr>
<tr>
<td>Deoxy1 deoxy1128v141</td>
<td>1.39</td>
</tr>
<tr>
<td>OCB</td>
<td>1.30</td>
</tr>
<tr>
<td>COLM</td>
<td>1.25</td>
</tr>
<tr>
<td>CLOC</td>
<td>1.25</td>
</tr>
<tr>
<td>AEGIS aegis128i</td>
<td>1.13</td>
</tr>
<tr>
<td>AES aes128n128clcv2</td>
<td>1.09</td>
</tr>
<tr>
<td>JAMBU-AES aesjambuv2</td>
<td>1.09</td>
</tr>
<tr>
<td>SILC</td>
<td>1.08</td>
</tr>
<tr>
<td>MORUS</td>
<td>1.07</td>
</tr>
<tr>
<td>JAMBU-SIMON jambusimon96v2</td>
<td>1.03</td>
</tr>
<tr>
<td>AES-GCM</td>
<td>0.86</td>
</tr>
<tr>
<td>ACORN</td>
<td>0.82</td>
</tr>
<tr>
<td>Ascon</td>
<td>0.82</td>
</tr>
<tr>
<td>OTR</td>
<td>0.66</td>
</tr>
<tr>
<td>Ketje ketjesrv2</td>
<td>0.50</td>
</tr>
</tbody>
</table>

**Throughput-to-Area RTL / Throughput-to-Area HLS in Virtex 7**

- **[1.30..0.90]**: RTL and HLS acceptable
- **(0.90..0.70)**: RTL may be improved
- **> 1.30**: Suboptimal HLS
- **< 0.70**: Suboptimal RTL

23
RTL vs. HLS Throughput/Area [(Mbits/s)/LUTs]

Consistently better than AES-GCM

Suboptimal RTL
Suboptimal HLS
HLS vs. RTL Throughput vs. Area

Throughput [Mbit/s] vs. Area [LUTs]

- HLS
- RTL

Legend:
- acorn128v3
- aeadaes128octaglen128v1
- aegis128l
- aes128gcmv1
- aes128n12t8clcv2
- aes128n12t8silcv2
- aes128otpv3
- aesjambuv2
- ascon128av12
- colm0v1
- deoxyxsi128v141
- ketjersrv2
- morus1280128v2
- norx3241v3
- tiaoxinv2
Transformation to HLS-ready C/C++ Code

1. Language partitioning and interface mapping
2. Addition of HLS Tool directives (pragmas)
3. Hardware-driven code refactoring
Language Partitioning
Basic handshaking signals (valid, ready) added automatically
Code Refactoring – High-Level

Reference C

Encryption

Decryption

HLS-ready C/C++

Encryption/Decryption

Use of pragmas possible but unreliable
Code Refactoring: Low-Level

Single vs. Multiple Function Calls:

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

// (b) After modification
for (round = 0; round < NB_ROUNDS; ++round)
{
    if (round == NB_ROUNDS - 1)
        x = 1;
    else
        x = 0;
    single_round(state, x);
}
Adding HLS Tool Directives - Pragmas

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
        ...
    }
```

Change array shapes:

```c
void AES_encrypt (word8 a[4][4], word8 k[4][4], word8 b[4][4])
    {
    #pragma HLS ARRAY_RESHAPE variable=a[0] complete dim=1 reshape
    #pragma HLS ARRAY_RESHAPE variable=a[1] complete dim=1 reshape
    #pragma HLS ARRAY_RESHAPE variable=a[2] complete dim=1 reshape
    #pragma HLS ARRAY_RESHAPE variable=a[3] complete dim=1 reshape
    #pragma HLS ARRAY_RESHAPE variable=a complete dim =1 reshape
    ```
Verification Framework
Sources of Productivity Gains

- Higher-level of abstraction
- Focus on datapath rather than control logic
- Debugging in software (C/C++)
  - Faster run time
  - No timing waveforms
Conclusions

Accuracy:
- **Good** (but not perfect) correlation between algorithm rankings using RTL and HLS approaches

Efficiency:
- **3-10 shorter development time**
- Designer can focus on functionality: control logic inferred
- Much easier verification: C/C++ testbenches
- A single designer can produce implementations of multiple (and even all) candidates

Bottom Line:
- Manual design approach still predominant
- HLS design approach at the experimental stage – more research needed
Open Source

GMU HLS-ready C Code
- 15 Round 3 CAESAR Candidates
- AES-GCM

GMU RTL VHDL Code
- 10 Round 3 CAESAR Candidates
- AES-GCM

made available at https://cryptography.gmu.edu/athena under CAESAR ⇒

GMU Implementations of Authenticated Ciphers and Their Building Blocks
Future Work: High-Level Synthesis

AES, SHA-3, & CAESAR

Vivado HLS

Post-Quantum Cryptography (69 candidates)

Academic Tools: Bambu, LegUp, DWARV, GAUT

Deep Analysis & Contribution to Tool Development
Q&A

Thank You!

Questions?  Comments?

Suggestions?

CERG: http://cryptography.gmu.edu

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