An Alternative Approach to Hardware Benchmarking of CAESAR Candidates Based on the Use of High-Level Synthesis Tools

Ekawat Homsirikamol and Kris Gaj
George Mason University USA

Based on work partially supported by the National Science Foundation under Grant No. 1314540
First Author

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”
### Number of Candidates in Cryptographic Contests

<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>
</tbody>
</table>
Pros & Cons of Multiple Designers

Pros:

• Distribution of effort
• Larger talent pool
• Potential for design space exploration

Cons:

• Different skills of designers
• Different amount of time and effort
• Misunderstandings regarding API and optimization target
• Requests for extending the deadline or disregarding ALL results
Potential Solution: High-Level Synthesis (HLS)

- High Level Language (preferably C or C++)
- High-Level Synthesis
- Hardware Description Language (VHDL or Verilog)
Case for High-Level Synthesis & Crypto

- Each submission includes reference implementation in C
- Development time potentially decreased 3-10 times
- All candidates can be implemented by the same group, and even the same designer
- Results from High-Level Synthesis could have a large impact in early stages of the competitions and help narrow down the search
- RTL code and results from previous contests form excellent benchmarks for High-Level Synthesis tools, which can generate fast progress targeting cryptographic applications
Potential Additional Benefits

BEFORE: Early feedback for designers of 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

DURING: Faster design space exploration

- Multiple hardware architectures (folded, unrolled, pipelined, etc.)
- Multiple variants of the same algorithms (e.g., key, nonce, tag size)
- Detecting suboptimal manual designs
Typical Doubts (from reviewers of our papers)

- How can we trust these tools?
- Isn’t manual design always better?
- Is it fair to compare manual designs with HLS designs?
- Won’t the number of candidates saturate soon anyway?
Typical Doubts (from reviewers of our papers)

• How can we trust these tools?
• Isn’t manual design always better?
• Is it fair to compare manual designs with HLS designs?
• Won’t the number of candidates saturate soon anyway?
• Why did not you implement Serpent?

(the same reviewer at two major crypto conferences)
High-Level Synthesis: State of the Art

“A Survey and Evaluation of FPGA High-Level Synthesis Tools”


Razvan Nane, Vlad-Mihai Sima, Koen Bertels: Delft University of Technology, The Netherlands

Christian Pilato, Fabrizio Ferrandi: Politecnico di Milano, Italy

Jongsok Choi, Blair Fort, Andrew Canis, Yu Ting Chen, Hsuan Hsiao, Stephen Brown, Jason Anderson: University of Toronto, Canada
### Number of Tools

<table>
<thead>
<tr>
<th></th>
<th>C, C++, or Extended C</th>
<th>Other Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Use</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>Abandoned</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>Status Unknown</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>26</td>
<td>7</td>
</tr>
</tbody>
</table>
### Number of Tools supporting C, C++, Extended C

<table>
<thead>
<tr>
<th>Status</th>
<th>Commercial</th>
<th>Academic</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Use</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>Abandoned</td>
<td>1 (C2H)</td>
<td>6</td>
</tr>
<tr>
<td>Status Unknown</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>12</td>
<td>14</td>
</tr>
</tbody>
</table>
In-Use Tools supporting C, C++, Extended C

Commercial:

- **CHC**: Altium; **CoDeveloper**: Impulse Accelerated; **Cynthesizer**: FORTE; **eXCite**: Y Explorations; **ROCCC**: Jacquard Comp.
- **Catapult-C**: Calypto Design Systems; **CtoS**: Cadence; **DK Design Suite**: Mentor Graphics; **Symphony C**: Synopsys
- **Vivado HLS**: Xilinx

Academic:

- **Bambu**: Politecnico di Milano, Italy
- **DWARV**: Delft University of Technology, The Netherlands
- **GAUT**: Universite de Bretagne-Sud, France
- **LegUp**: University of Toronto, Canada
Crypto-related Benchmarks (C programs)

CHStone Benchmark Program Suite for Practical C-based High-Level Synthesis
http://www.ertl.jp/chstone/

aes-encrypt:
Key scheduling + Encryption of 1 128-bit block

aes-decrypt:
Key scheduling + Decryption of 1 128-bit block

sha:
Hashing of 256 512-bit blocks using SHA-1

blowfish:
Key scheduling + Encryption of 650 64-bit blocks in CFB64 mode
### Benchmarking Results in Number of Clock Cycles Before Optimization

<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>Bambu</td>
<td>1,574</td>
<td>2,766</td>
<td>111,762</td>
<td>57,590</td>
</tr>
<tr>
<td>DWARV</td>
<td>5,135</td>
<td>2,579</td>
<td>71,163</td>
<td>70,200</td>
</tr>
<tr>
<td>LegUp</td>
<td>1,564</td>
<td>7,367</td>
<td>168,886</td>
<td>75,010</td>
</tr>
<tr>
<td>Commercial</td>
<td>3,976</td>
<td>5,461</td>
<td>197,867</td>
<td>101,010</td>
</tr>
<tr>
<td>Manual</td>
<td>20</td>
<td>20</td>
<td>20,480</td>
<td>18,736</td>
</tr>
<tr>
<td>Best/Manual</td>
<td>78</td>
<td>129</td>
<td>3.5</td>
<td>3.1</td>
</tr>
</tbody>
</table>
# Benchmarking Results in Number of Clock Cycles After Optimization

<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>Bambu</td>
<td>1,485</td>
<td>2,585</td>
<td>51,399</td>
<td>57,590</td>
</tr>
<tr>
<td>DWARV</td>
<td>3,282</td>
<td>2,579</td>
<td>71,163</td>
<td>70,200</td>
</tr>
<tr>
<td>LegUp</td>
<td>1,191</td>
<td>4,847</td>
<td>81,786</td>
<td>64,480</td>
</tr>
<tr>
<td>Commercial</td>
<td>3,735</td>
<td>3,923</td>
<td>124,339</td>
<td>96,460</td>
</tr>
<tr>
<td>Manual</td>
<td>20</td>
<td>20</td>
<td>20,480</td>
<td>18,736</td>
</tr>
<tr>
<td>Best/Manual</td>
<td>60</td>
<td>129</td>
<td>2.5</td>
<td>3.1</td>
</tr>
</tbody>
</table>
Our Choice of the HLS Tool: Vivado HLS

- **Integrated** into the primary Xilinx toolset, Vivado, and released in 2012
- **Free** (or almost free) licenses for academic institutions
- Good **documentation** and user **support**
- The largest number of **performance optimizations**
  - **8 out of 8**: Operation Chaining, Bitwidth Analysis and Optimization, Memory Space Allocation, Loop Optimizations, Hardware Resource library, Speculation and Code Motion, If-Conversion [Bambu, LegUp: 6 out of 8, DWARV: 5 out of 8]
  - On average the **highest clock frequency** of the generated code
Licensing Limitations of Vivado HLS

1. Results cannot be compared with results obtained using other HLS tools
2. Designers are not allowed to target ASICs
3. Designers are not allowed to target devices of other FPGA vendors (e.g., Altera)
GMU (Ice’s) Previous Efforts (1)

AES-128-ECB-ENC (Spartan 6):
ReConFig (Reconfigurable Computing and FPGAs), Dec. 2014

HLS/RTL ratios:
• Clock cycles: \(12/10 = 1.2\)
• Area: \(343/354 = 0.97\)

RTL/HLS ratios:
• Frequency: \(230/231 = 0.996\)
• Throughput: \(2943/2467 = 1.19\)
• Throughput/Area: \(8.31/7.19 = 1.16\)
GMU (Ice’s) Previous Efforts (2)

5 Final SHA-3 Candidates & SHA-2 (Virtex 6):
ARC (Applied ReConfigurable Computing, Apr. 2015)
Our Hypotheses

- **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
  - possibly Post-quantum Cryptography?
18 months of unsuccessful publishing attempts and unread/ignored rebuttals

1. Why not other HLS tools?
2. Why not ASICs?
3. Why not other FPGA vendors (e.g., Altera)?
4. Why no previous work by other teams?
5. Why another publication?
18 months of **unsuccessful** publishing attempts and **unread/ignored** rebuttals

1. Why not **other HLS tools**?
2. Why not **ASICs**?
3. Why not **other FPGA vendors** (e.g., Altera)?
4. Why no **previous work** by other teams?
5. Why another **publication**?
6. Why not **Serpent**?
DIAC 2016 vs. DIAC 2015

- **CAESAR HW API 1.0** (02/2016) vs. GMU API 1.1 (09/2015)
- Comparison vs. **RTL implementations** developed by other groups
- **New candidates** (e.g., MORUS, AEGIS, NORX, SILC)
- Block-based => **stream-based implementation**
- Easily **adjustable** algorithm-dependent **port widths**
- **C++** testbench independent of hardware architecture
- Automated generation of **test vectors at the CipherCore (C++) level**
Traditional Register-Transfer Level (RTL) Development & Benchmarking Flow

- Informal Specification
  - Manual Design
  - HDL Code
  - Xilinx ISE + ATHENA
  - Netlist
  - Post Place & Route Results
- Functional Verification
- Timing Verification
- Test Vectors
Proposed HLS-Based Development and Benchmarking Flow

Reference Implementation in C

Manual Modifications (pragmas, tweaks)

HLS-ready C code

High-Level Synthesis

HDL Code

Xilinx ISE + ATHENa

Netlist

Post Place & Route Results

Test Vectors

Functional Verification

Timing Verification
Language Partitioning
Basic handshaking signals (valid, ready) added automatically
Easily Adjustable Port Widths

```c
struct public_bus {
    data_t           data;
data_bytes_t       valid_bytes;
data_bytes_t       pad_loc;
bsize_t            size;
ap_uint<3>         type;
bool               eoi;
bool               eot;
bool               partial;
};
```

```c
struct secret_bus {
    secret_t          data;
};
```

```c
struct output_bus {
    data_t           data;
    bsize_t          size;
};
```

```c
/* Interface data types */
typedef ap_uint<KEYBUS> secret_t;
typedef ap_uint<BLOCKSIZE> data_t;
typedef ap_uint<BLOCKSIZE/8> data_bytes_t;
typedef ap_uint<LBS_BYTES+1> bsize_t;
```

```c
/* Interface parameters */
#define KEYSIZE 128
#define KEYBUS 32
#define LBS_BYTES 4
```
## Reference C vs. HLS-ready C/C++

<table>
<thead>
<tr>
<th>Data</th>
<th>Reference C</th>
<th>HLS-ready C/C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Random</td>
<td>Serial</td>
</tr>
<tr>
<td></td>
<td>Data can be accessed at any location multiple times</td>
<td>Previously accessed data must be maintained inside of the code if required</td>
</tr>
<tr>
<td>Width</td>
<td>Byte/Word</td>
<td>Block size</td>
</tr>
<tr>
<td>Total Size</td>
<td>Known</td>
<td>Unknown</td>
</tr>
<tr>
<td>Status</td>
<td>Always available</td>
<td>Availability unknown until the time of read</td>
</tr>
</tbody>
</table>
Reference C vs. HLS-ready C/C++

Reference C
- Encryption
- Decryption

HLS-ready C/C++
- Encryption/Decryption

Use of pragmas possible but unreliable
Low-Level Code Rewriting

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 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
```
HLS-Ready C/C++ Code Generation

Phase I

1. Step-by-step designer’s guide (under development)
   - Code rewriting
   - Pragmas insertion

2. Multiple examples (AES, SHA-3, CAESAR contests)

Phase II

1. Automated insertion of pragmas for Vivado HLS

2. Translation of Vivado HLS pragmas to pragmas for academic tools: Bambu, DWARV, LegUp
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
Tentative Results

Post-Round 2 RTL,
First Time with CAESAR API
and RTL designers from multiple groups
RTL vs. HLS Throughput [Mbits/s]

Different hardware architectures in HLS vs. RTL
RTL vs. HLS Ratios for Throughput in Virtex 6

Suboptimal HLS

Joltik: 1.63
ICEPOLE: 1.58
NORX: 1.41
OCB: 1.31
AES-COPA: 1.24
POET: 1.22
JAMBU-SIMON: 1.16
Ascon: 1.14
CLOC: 1.01
PAEQ: 0.96
AEGIS: 0.92
Deoxys: 0.90
PRIMATES GIBBON: 0.90
PRIMATES HANUMAN: 0.83
AES-GCM: 0.82
MORUS: 0.79
SCREAM: 0.71
SILC: 0.56
MINALPHER: 0.55

> 1.30
< 0.70
RTL vs. HLS Area [LUTs]

Different hardware architectures in HLS vs. RTL

Small difference in RTL
RTL vs. HLS Ratios for Area in Virtex 6

Sub-optimal RTL: > 1.30
Sub-optimal HLS: < 0.70

Bar chart showing the area ratios for various benchmarks.

- Minalpher: 0.50
- SCREAM: 0.57
- Ascon: 0.69
- Joltik: 0.69
- PRIMATES GIBBON: 0.71
- PRIMATES HANUMAN: 0.76
- Deoys: 0.76
- AES-GCM: 0.82
- PAEQ: 0.83
- OCB: 0.83
- MORUS: 0.87
- NORX: 0.88
- SILC: 0.90
- ICEPOL: 0.95
- CLOC: 1.00
- AES-COPA: 1.07
- JAMBU-SIMON: 1.24
- POET: 1.37
- AEGIS: 1.50
RTL vs. HLS Throughput/Area [(Mbits/s)/LUTs]

Different hardware architectures in HLS vs. RTL
RTL vs. HLS Ratios for Throughput/Area in Virtex 6

- **Suboptimal HLS**
  - > 1.30
  - [0.90, 1.30]
  - (0.70, 0.90]
  - < 0.70

- **RTL**
  - may be improved
  - acceptable
  - Suboptimal RTL
Possible Future Uses of HLS

Identifying **suboptimal RTL implementations** in Round 3 of the CAESAR Contest

Designing **new building blocks** [e.g., rounds, steps, etc.] **for** hardware-friendly block ciphers, hash functions, and authenticated ciphers

---

Post-Quantum Cryptography

Early Rounds of Future Contests
Remaining Difficulties

- **Suboptimal control unit** of HLS implementations
  \#cycles per block \( \geq \# \text{rounds} + 2 \)

- **Wide range of RTL to HLS performance metric ratios**
  Wide range of **RTL designer skills** and **selected architectures**

- A few potentially **suboptimal HLS or RTL implementations**

- **Dependence** of results on particular FPGA family

- Efficient and reliable **generation of HLS-ready C/C++ code**

- **Portability among HLS tools**

- **Licensing limitations of commercial tools**
HLS vs. RTL Ratios for Number of Clock Cycles
Best HLS/RTL reported so far

• “A Survey and Evaluation of FPGA High-Level Synthesis Tools”


  • 12 leading researchers in the HLS field
  • Co-developers of top 3 academic HLS Tools

<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>Best/Manual</td>
<td>60</td>
<td>129</td>
<td>2.5</td>
<td>3.1</td>
</tr>
</tbody>
</table>
Typical Doubts (from reviewers of our papers)

- How can we trust these tools?
  If HLS used efficiently, maximum 20% penalty in the number of clock cycles per block. Easy to verify by comparing vs. the number of rounds.

- Isn’t manual design always better?
  Multiple HLS designs with one or more metrics better. 7 out of 19 HLS designs with better Throughput/Area.

- Is it fair to compare manual designs with HLS designs?
  It is not our intention. HLS results are supposed to be compared with HLS only. However if an existing RTL result worse, it is OK to use HLS result temporarily.
Ekawat Homsirikamol
a.k.a “Ice”

- Main developer of the **RTL Round 2 Benchmarking Framework** and Developer’s Package
- **RTL Designer for 12 Round 2 Candidates:** AES-GCM, AEZ, Ascon, Deoxys, HS1-SIV, ICEPOLE, Joltik, NORX, OCB, PAEQ, Pi-Cipher, STRIBOB
- Developer of the HLS-based methodology and framework for crypto applications
Thank you!

Comments? Questions? Suggestions?

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