Can High-Level Synthesis Compete Against a Hand-Written Code in the Cryptographic Domain?  
A Case Study

Ekawat Homsirikamol & Kris Gaj
George Mason University
USA

Project supported by NSF Grant #1314540
Outline

- High-level synthesis background
- Methodology and benchmarking routine
- Interface, protocol, and reference designs
- Modifications and results
- Lessons learned
- Conclusions
High-Level Synthesis (HLS)

High Level Language
(e.g. C, C++, Matlab, Bluespec)

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
Cinderella Story

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”
Can High-Level Synthesis Compete Against a Hand-Written Code in the Cryptographic Domain?
Traditional Development and Benchmarking Flow

Informal Specification

Manual Design

HDL Code

Manual Optimization

FPGA Tools

Netlist

Test Vectors

Functional Verification

Timing Verification

Post Place & Route Results
Extended Traditional Development and Benchmarking Flow

Informal Specification

Manual Design

HDL Code

Option Optimization

FPGA Tools

Netlist

Functional Verification

Timing Verification

GMU ATHENA

Test Vectors

Post Place & Route Results
ATHENa – Automated Tool for Hardware Evaluation

http://cryptography.gmu.edu/athena

Benchmarking open-source tool, written in Perl, aimed at an AUTOMATED generation of OPTIMIZED results for MULTIPLE hardware platforms

Currently under development at George Mason University
Generation of Results Facilitated by ATHENA

- batch mode of FPGA tools
- ease of extraction and tabulation of results
  - Text Reports, Excel, CSV (Comma-Separated Values)
- optimized choice of tool options
  - GMU_optimization_1 strategy
HLS-Based Development and Benchmarking Flow

Reference Implementation in C

Manual Modifications (pragmas, tweaks)

HLS-ready C code

High-Level Synthesis

HDL Code

Option Optimization

FPGA Tools

Netlist

Test Vectors

Functional Verification

GMU ATHENA

Timing Verification

Post Place & Route Results
Case Study

- Algorithm: AES-128
- Mode of operation: Counter (CTR)
- Protocol and interface: GMU proposal
- Two vendors: Xilinx & Altera
- Four different FPGA families
  - Xilinx Spartan-6 (X-S6)
  - Xilinx Virtex-7 (X-V7)
  - Altera Cyclone IV (A-CIV)
  - Altera Stratix V (A-SV)
Tools & Tool Versions

- Vivado HLS 2014.1
- Xilinx ISE v14.7
- Altera Quartus II v13.0sp1
- ATHENa v0.6.4 (with GMU_optimization_1)
Interface & Protocol

CRYPTO CORE

w
pdi
pdi_ready
pdi_read
do

w
sdi
do_ready
do_write

w
sdi_ready
sdi_read

a)

<table>
<thead>
<tr>
<th>instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>seg(hdr)</td>
</tr>
</tbody>
</table>

| seg(dat) |

b)

<table>
<thead>
<tr>
<th>instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>seg(hdr)</td>
</tr>
</tbody>
</table>

| seg(dat) |

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>

| seg(hdr) |

| seg(dat) |
Top-Level
Reference Hardware Design in RTL VHDL

Note: All buses are 128-bit wide
RTL Result

Latency = 11 cycles

Time between two consecutive outputs = 10 cycles
Software Design

Reference Code


HLSv0

• Removed support for decryption
• Removed support for different AES variants
HLSv0: Xilinx Results

Latency = 7367 cycles
HLSv1: Code Refactoring

Refactor the code to match the target AES architecture

- KeyScheduling is performed once per round
- Improved Galois field multiplication operation
- Included last round as part of the core loop
HLSv1: Xilinx Results

Latency = 3224 cycles
HLSv2: Optimization directives: ARRAY_RESHAPE

- Change an array shape in the output hardware

```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
```
HLSv2: Optimization directives: UNROLL & INLINE

- **Unroll a loop**
  - OutputLoop: for (i = 0; i < 4; i ++)
    
    ```c
    #pragma HLS UNROLL
    for (j = 0; j < 4; j ++)
    #pragma HLS UNROLL
    b[i][j] = s[i][j];
    ```

- **Flatten a function's hierarchy for improved performance**
  - void KeyUpdate (word8 k[4][4], word8 round) {
    #pragma HLS INLINE
    ...
    }

```c
```
HLSv2: Optimization directives: RESOURCE & INTERFACE

- Specify the type of FPGA resource to be used by the target variable

```c
word32 rcon[10] = {
    0x01, 0x02, 0x04, 0x08, 0x10,
    0x20, 0x40, 0x80, 0x1b, 0x36
};
#pragma HLS RESOURCE variable=Rcon0 core=ROM_1P_1S
```

- Direct how an input/output port should behave, i.e., registered or handshake mode

```c
void AES_encrypt (word8 a[4][4], word8 k[4][4], word8 b[4][4])
{
    #pragma HLS INTERFACE register port=b
```
HLSv2: Xilinx Results

Latency = 11 cycles
HLSv2: HLS vs. RTL, Frequency - Area
HLSv2: HLS vs. RTL, **Throughput - Area**

![Graphs comparing throughput to area for various FPGA types and versions.](image-url)
Source of Inefficiencies: Datapath vs. Control Unit

Datapath

Data Inputs

Data Outputs

Control Signals

Status Signals

Control Unit

Control Inputs

Control Outputs

Determines

• Area
• Clock Frequency

Determines

• Number of clock cycles
Source of Inefficiencies

Datapath inferred correctly

• **Frequency** and **area** within 10% of manual designs

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:
  
  RTL: Throughput = Block_size / (#Rounds * T_{CLK})
  
  HLS: Throughput = Block_size / ((#Rounds+2) * T_{CLK})
AES-ECB-ENC x2: HLS vs. RTL, Frequency - Area

![Graphs showing frequency vs. area for different FPGA types: Altera Cyclone IV, Altera Stratix V, Xilinx Spartan 6, Xilinx Virtex 7. The graphs compare iterative and x2 unroll implementations for both HLS and RTL.]
AES-ECB-ENC x2: HLS vs. RTL, Throughput - Area
AES-CTR

 AES-CTR

 AES
 ECB-ENC
AES-CTR Results

![Graph showing performance comparison between Altera Stratix V and Xilinx Virtex 7.]
Full AES-CTR with I/O processors
AES-CTR with IO Results

- **Altera Cyclone IV**
  - Throughput (Mbit/s) vs. Area (LEs/ALMs/SLICEs)
  - Red and blue dots indicating RTL and HLS performance

- **Altera Stratix V**
  - Throughput (Mbit/s) vs. Area (LEs/ALMs/SLICEs)
  - Red and blue dots indicating RTL and HLS performance

- **Xilinx Spartan 6**
  - Throughput (Mbit/s) vs. Area (LEs/ALMs/SLICEs)
  - Red and blue dots indicating RTL and HLS performance

- **Xilinx Virtex 7**
  - Throughput (Mbit/s) vs. Area (LEs/ALMs/SLICEs)
  - Red and blue dots indicating RTL and HLS performance
Conclusions

- Area and frequency of designs produced by High-Level Synthesis are comparable to handwritten RTL code
- Small increase in the number of clock cycles reduces throughput of HLS-based approach
- Complex I/O units can be created by HLS-based approach
- HLS-based design can compete against handwritten RTL code when we have a specific architecture and latency in mind while preparing an HLS-ready HLL code
Thank you!

Questions?

Suggestions?

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