

# 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

# First Author

---



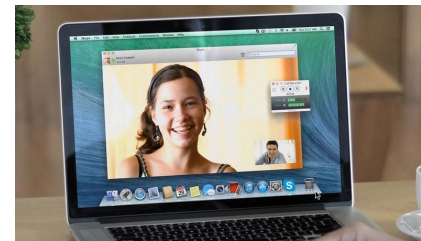
**Ekawat Homsirikamol**  
a.k.a “Ice”

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.

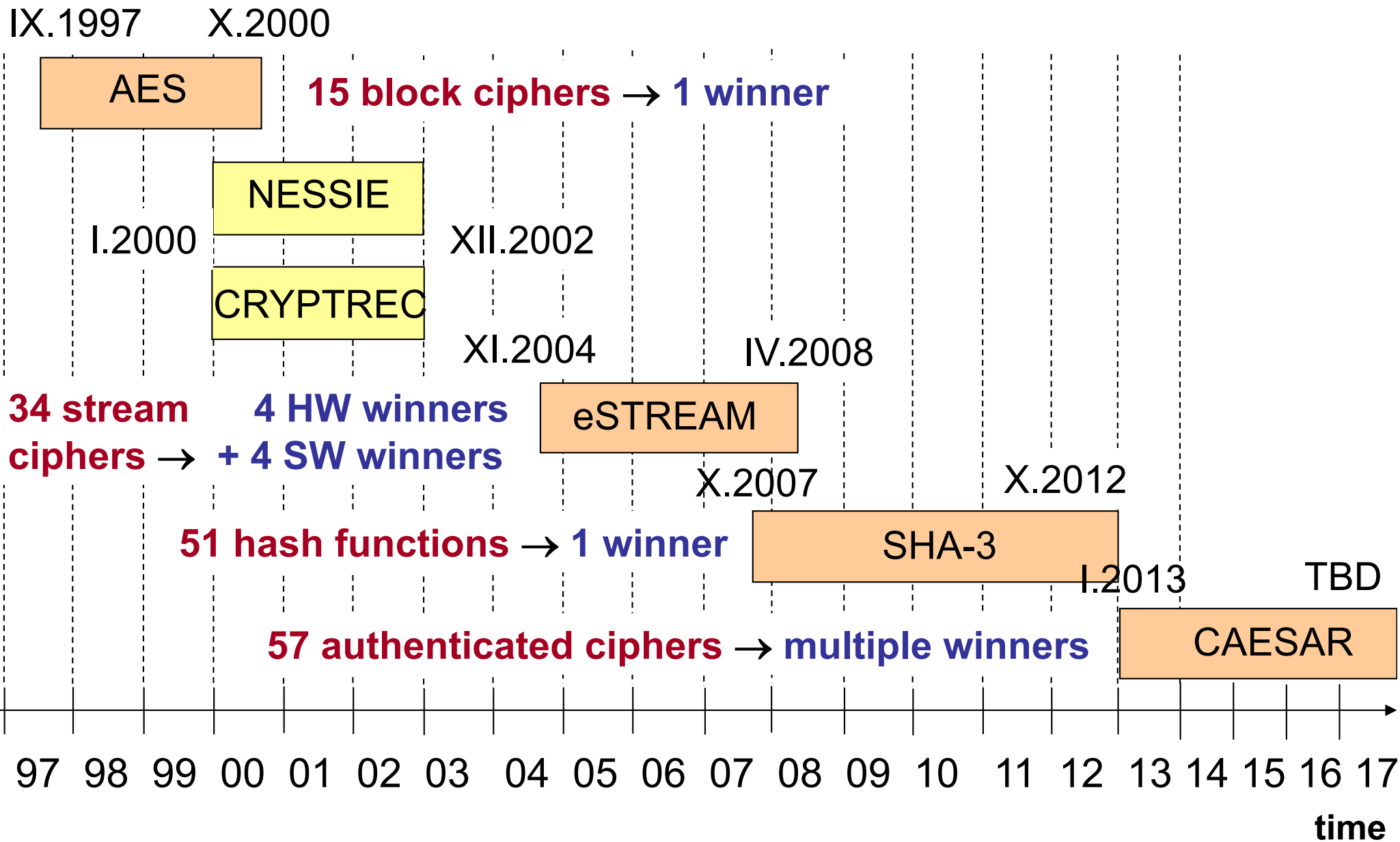


# Need for Standards



Sources: <http://geology.com>, <http://bluebuddies.com>

# Cryptographic Contests



# Evaluation Criteria

---

**Security**

**Software Efficiency**

$\mu$ Processors       $\mu$ Controllers

**Hardware Efficiency**

**FPGAs**      **ASICs**

**Flexibility**

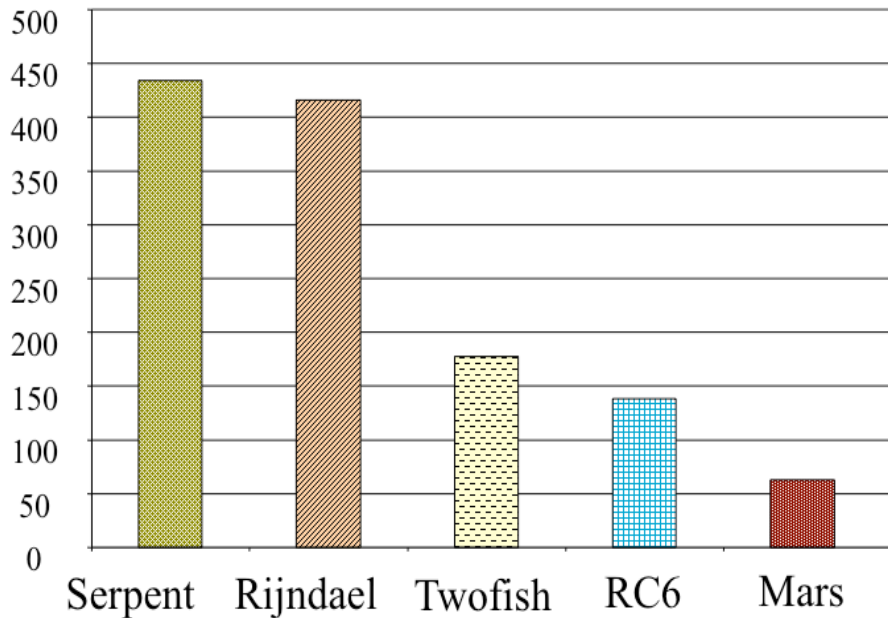
**Simplicity**

**Licensing**

# AES Final Round: 5 candidates

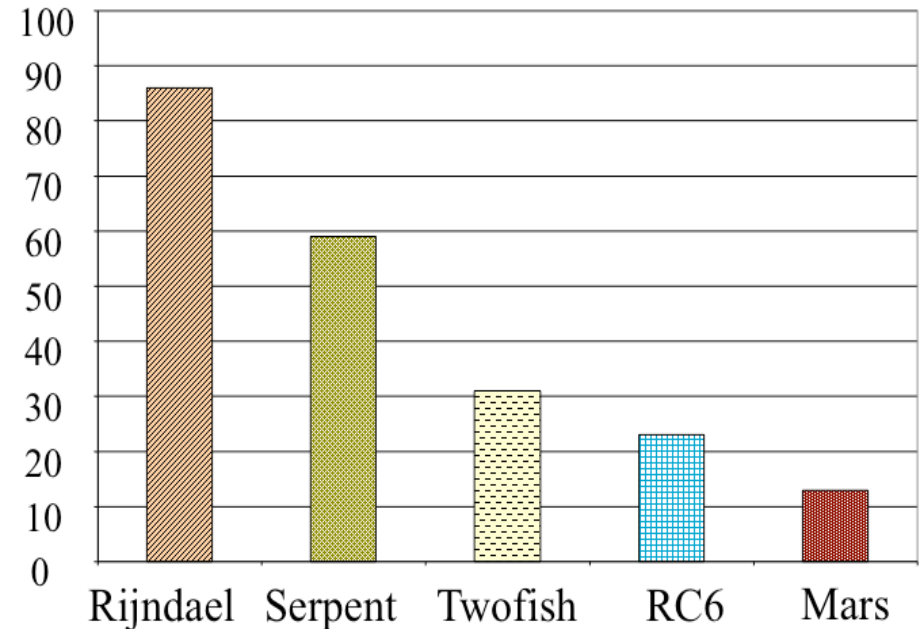
## GMU FPGA Results

Speed [Mbit/s]



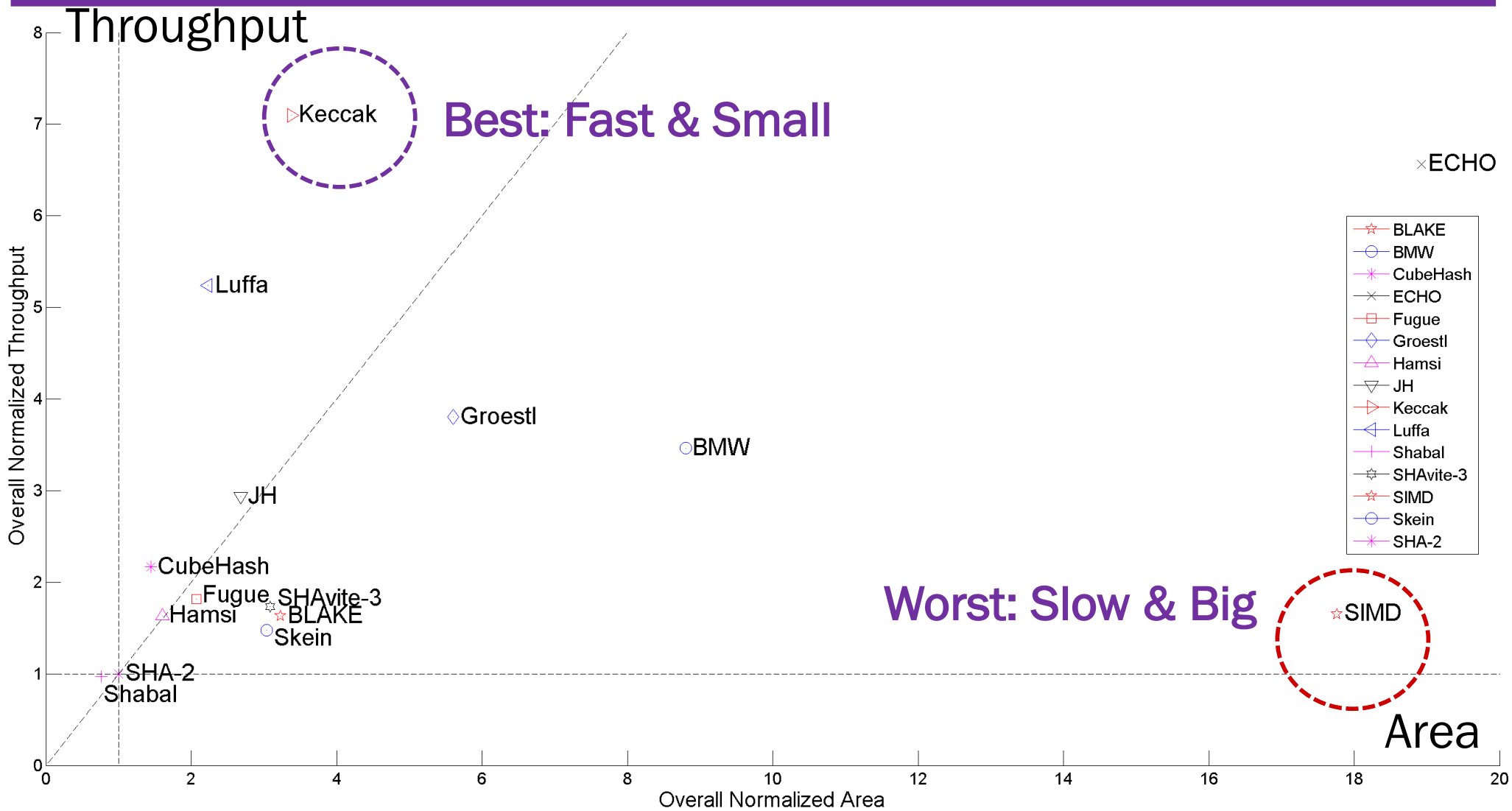
## Straw Poll @ AES 3 conference

# votes



**Rijndael second best in FPGAs,  
selected as a winner due to much better performance  
in software**

# SHA-3 Round 2: 14 candidates

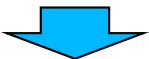



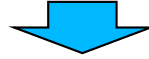


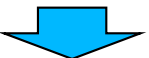



**Throughput vs. Area: Normalized to Results for SHA-2 and Averaged over 7 FPGA Families**



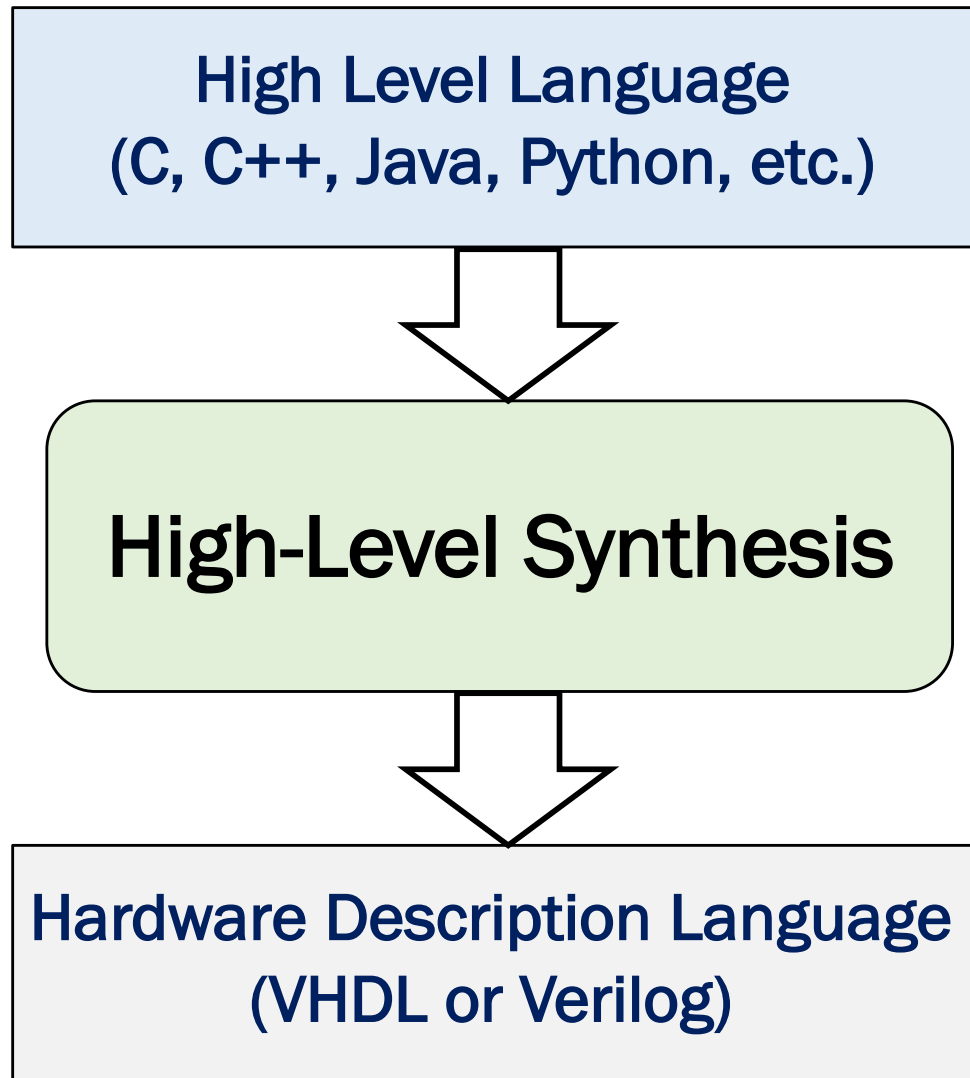
# Percentage of Candidates in Hardware

---

	Initial number of candidates	Implemented in hardware	Percentage
AES	15	5	33.3%
			
eSTREAM	34	8	23.5%
			
SHA-3	51	14	27.5%
			
CAESAR	57	28	49.1%
			
PQC	69	?	?

# High-Level Synthesis (HLS)

---



# 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

# HLS & Crypto: State-of-the-Art

## Cryptographic Benchmarks

Tools	aes-encrypt	aes-decrypt	sha	blowfish
<b>Best HLS</b>	1,191	2,579	51,399	57,590
Manual RTL	20	20	20,480	18,736
<b>Best HLS</b> <b>/Manual RTL</b>	<b>60</b>	<b>129</b>	<b><u>2.5</u></b>	<b>3.1</b>

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

Based on “A Survey and Evaluation of FPGA High-Level Synthesis Tools”, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 35, no. 10, Oct. 2016.

# 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:  $12/10 = 1.20$

★ Area:  $343/354 = 0.97$

## 🌐 Manual/HLS ratios:

★ Frequency:  $230/231 = 0.996$

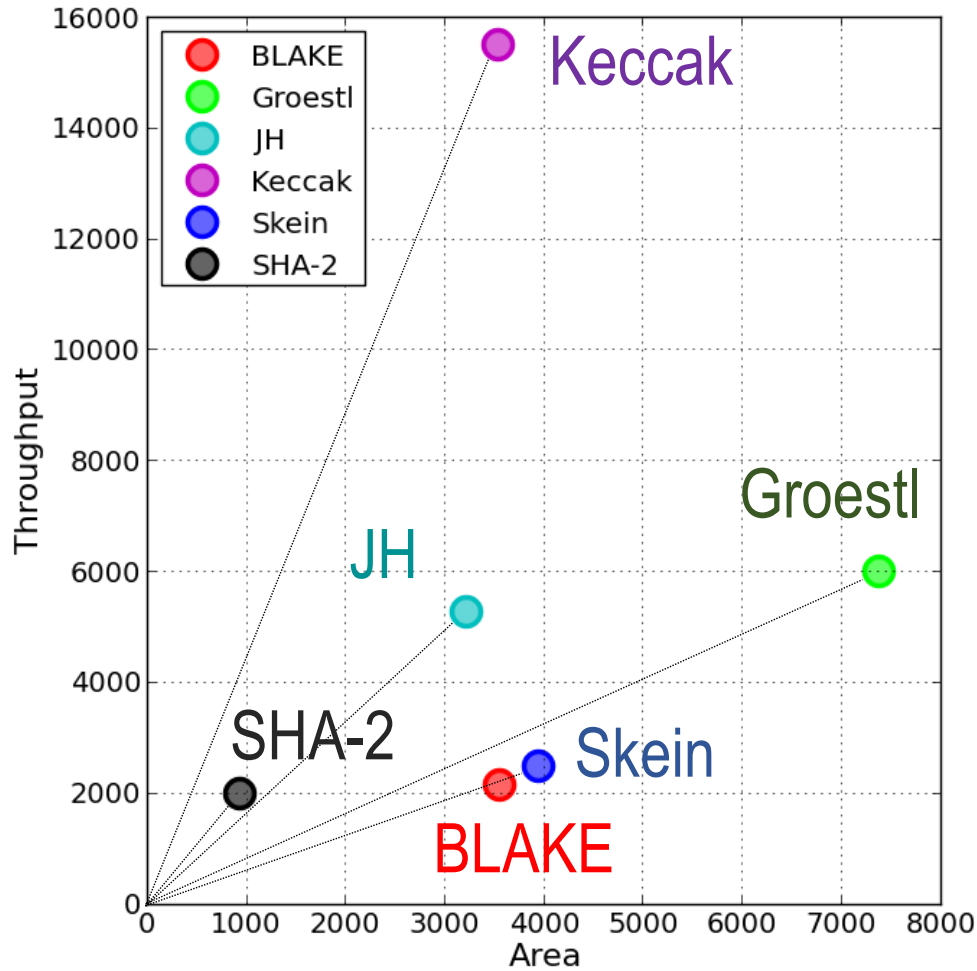
★ Throughput:  $2943/2467 = 1.19$

★ Throughput/Area:  $8.31/7.19 = 1.16$

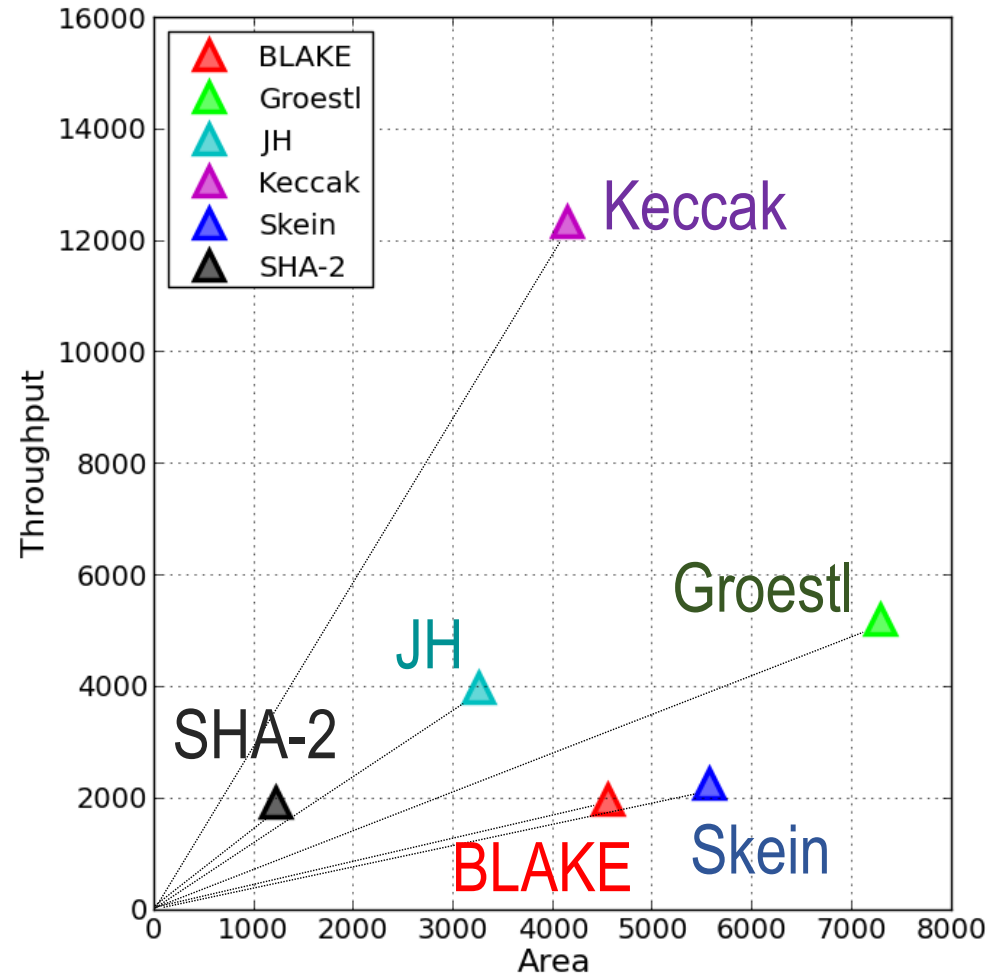


# ARC 2015: SHA-3 Candidates Revisited

## Altera Stratix III FPGA



RTL



HLS





# 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

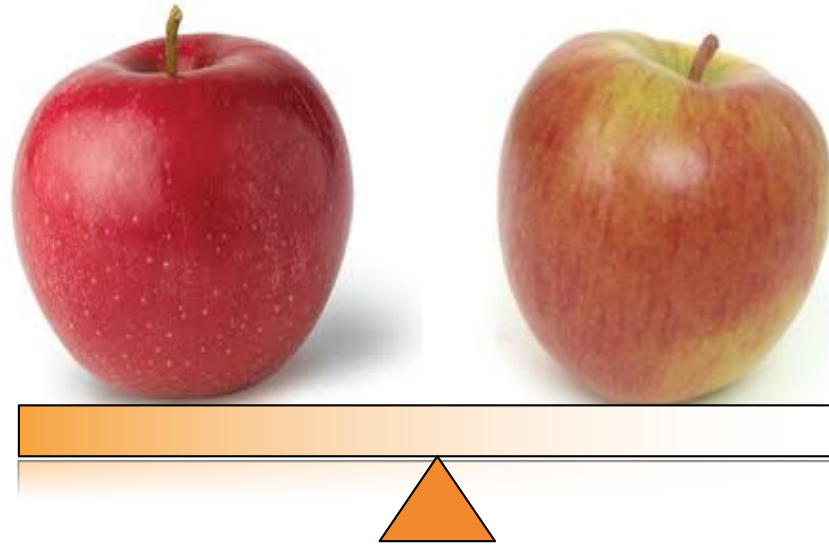
---

- 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

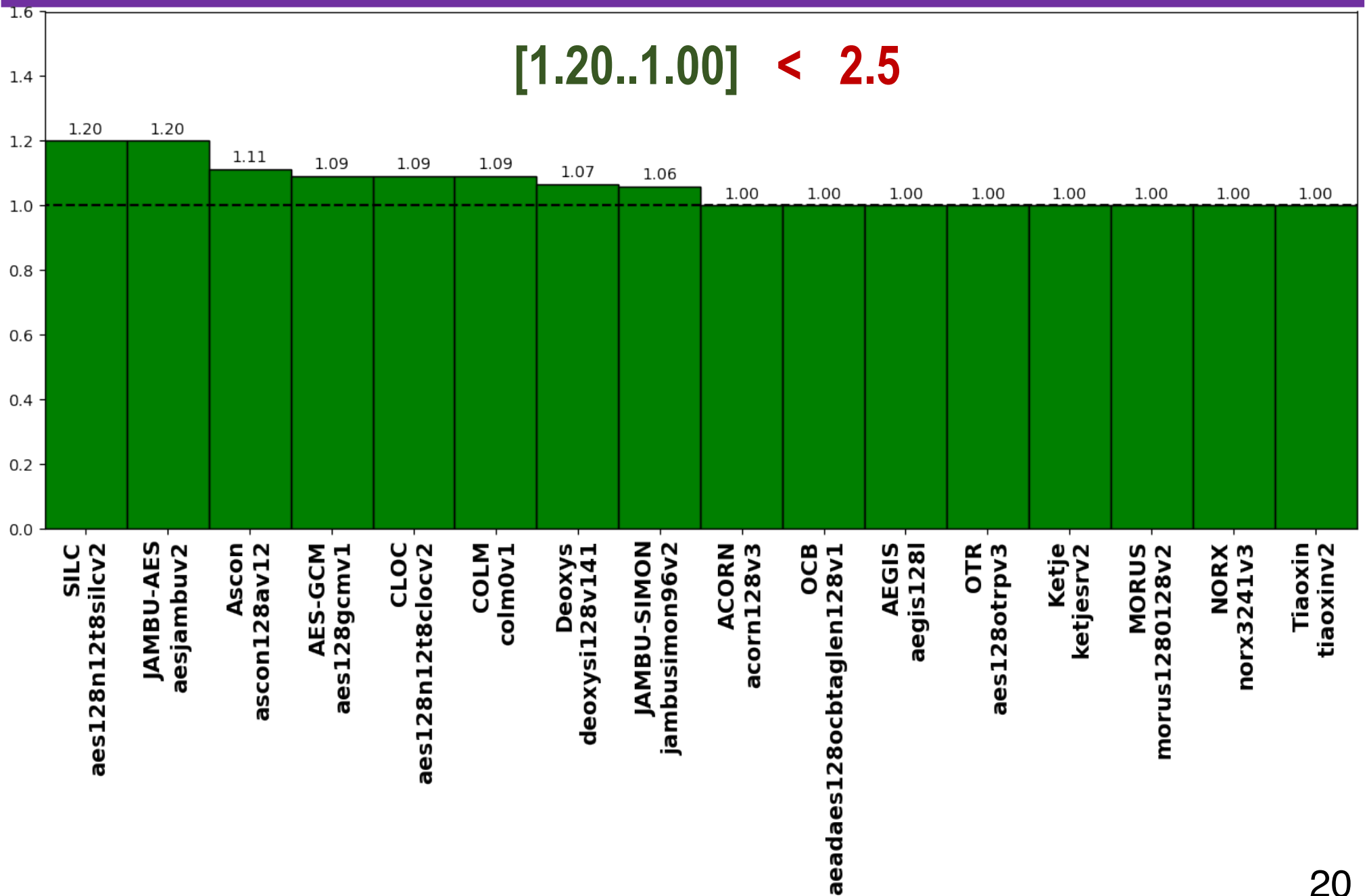
# CAESAR Case Study

---

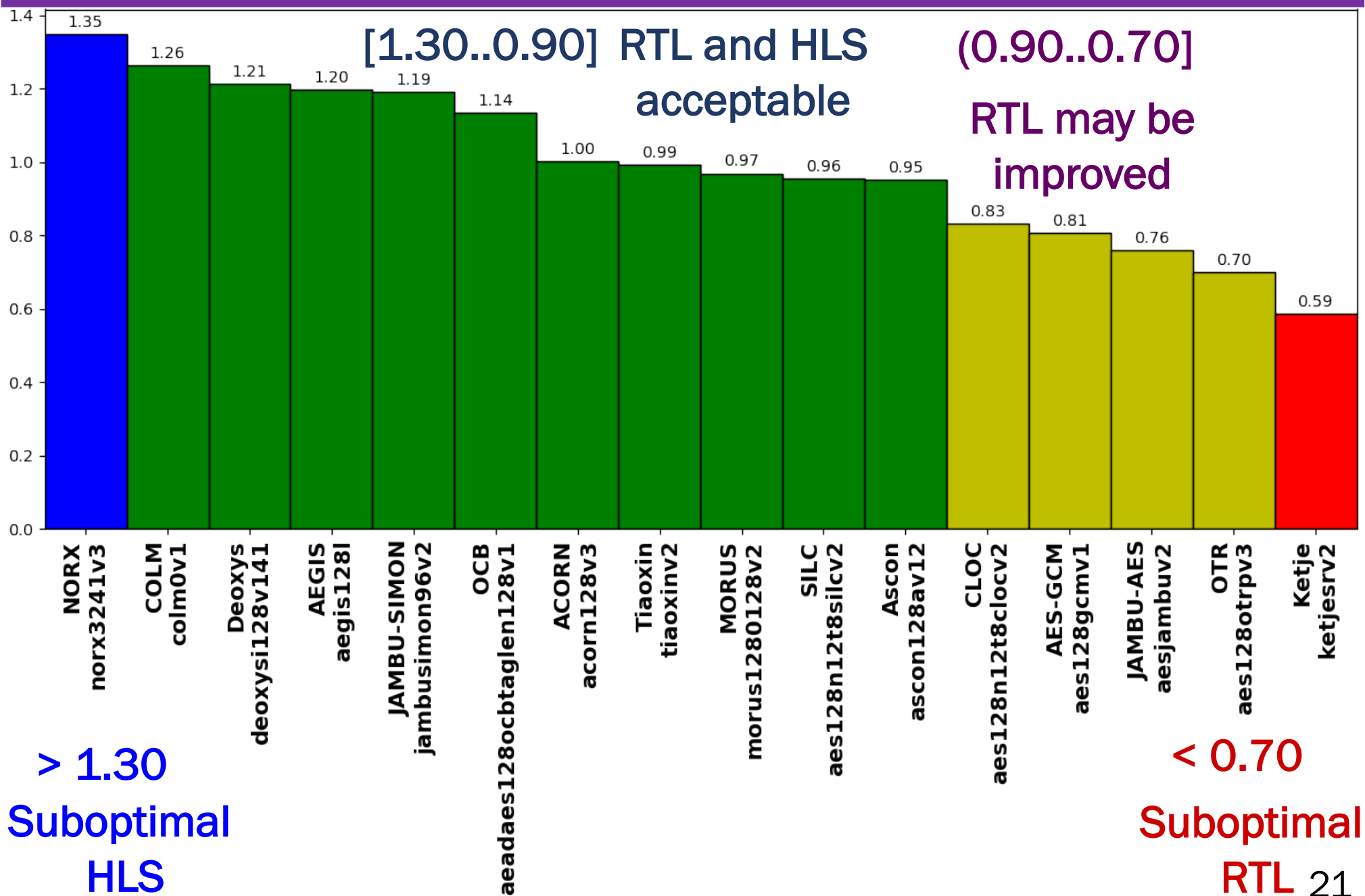


- **Uniform Hardware API**
- **Uniform PreProcessor & PostProcessor**
- **Uniform Benchmarking Methodology**
- **Two Platforms**
  - **Xilinx Virtex 6**
  - **Xilinx Virtex 7**

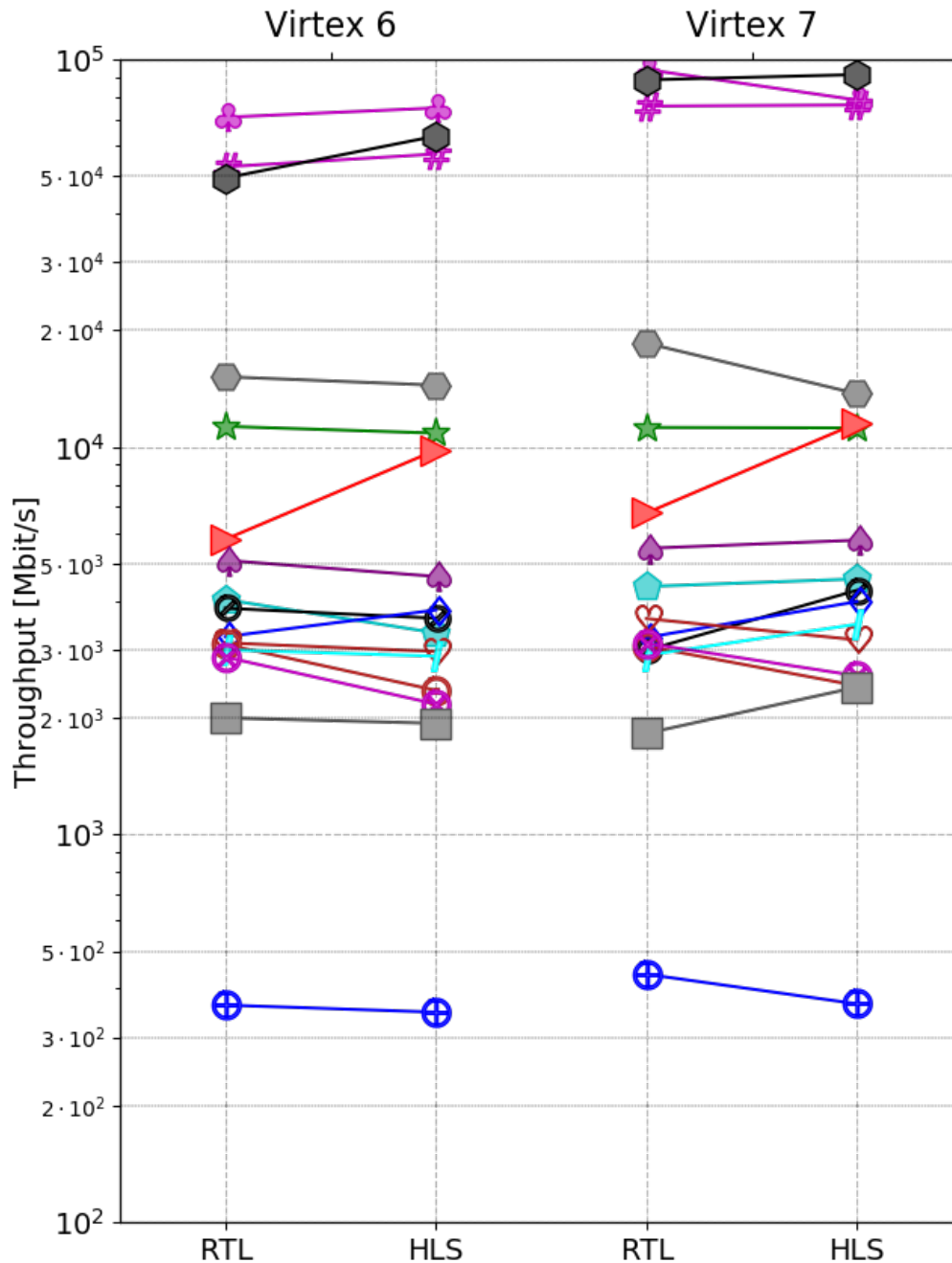
# HLS vs. RTL Ratios for Number of Clock Cycles



# Throughput RTL / Throughput HLS for Xilinx Virtex-7



# RTL vs. HLS Throughput [Mbits/s]

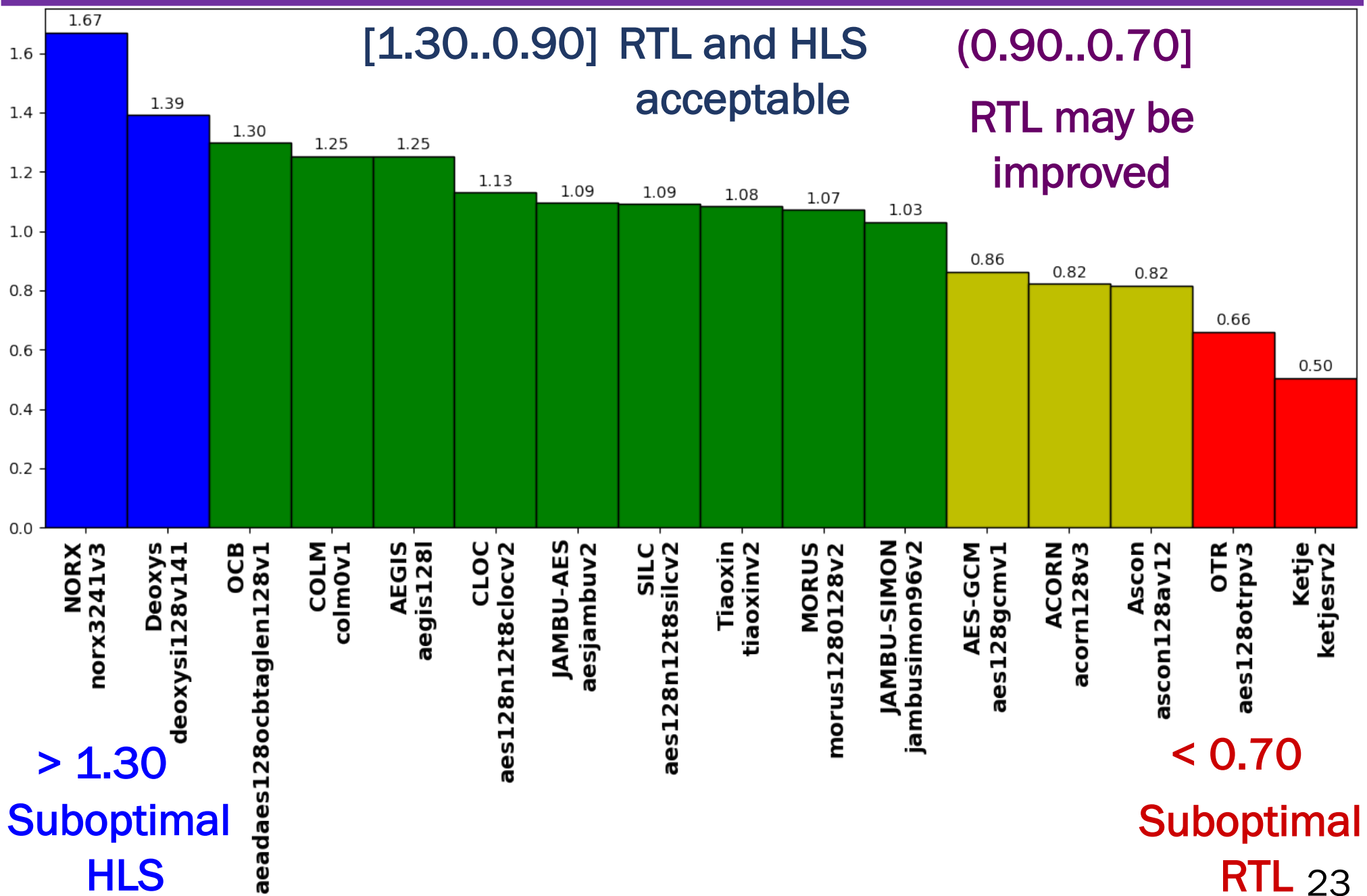


Algorithm	X-V6	X-V7
aegis128l	(1,1)	(1,2)
tiaoxinv2	(2,3)	(3,3)
morus1280128v2	(3,2)	(2,1)
norx3241v3	(4,4)	(4,4)
acorn128v3	(5,5)	(5,6)
ketjesrv2	(6,6)	(6,5)
ascon128av12	(7,7)	(7,7)
aes128n12t8 silcv2	(8,10)	(8,8)
aes128otrpv3	(9,9)	(13,9)
aes128gcmv1	(10,8)	(10,10)
aeadaes128 ocbtaglen128v1	(11,11)	(9,12)
colm0v1	(12,13)	(12,14)
aes128n12t8 clocv2	(13,12)	(14,11)
deoxysi128v141	(14,14)	(11,13)
aesjambuv2	(15,15)	(15,15)
jambusimon96v2	(16,16)	(16,16)

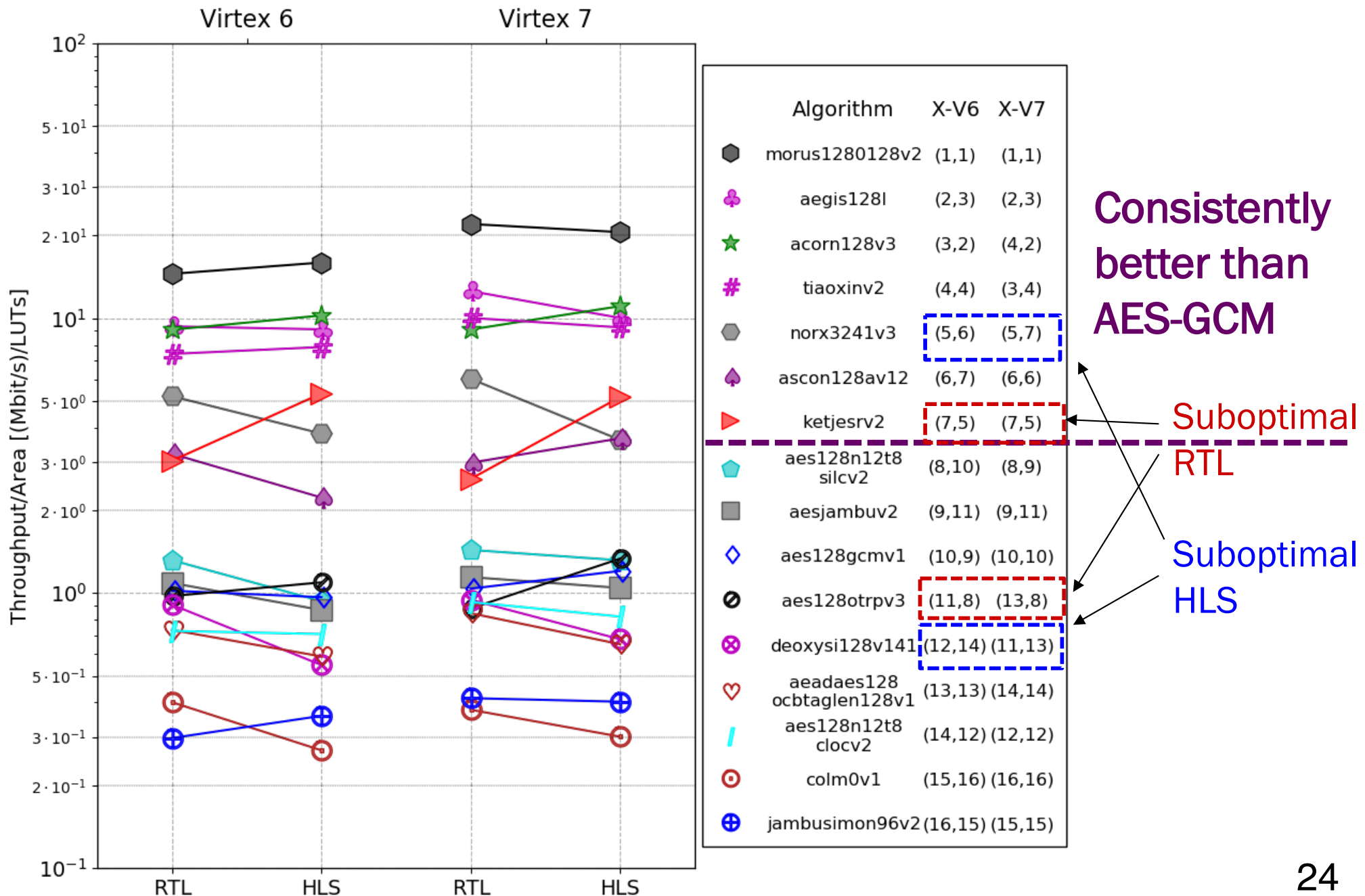
Consistently  
better than  
AES-GCM

Suboptimal  
RTL

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



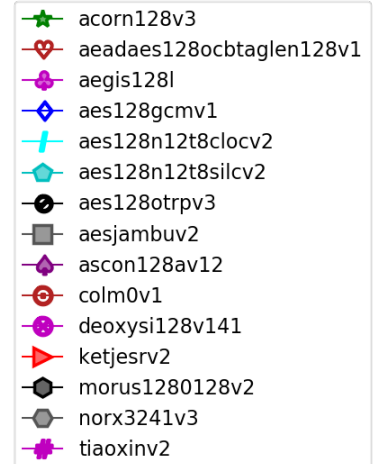
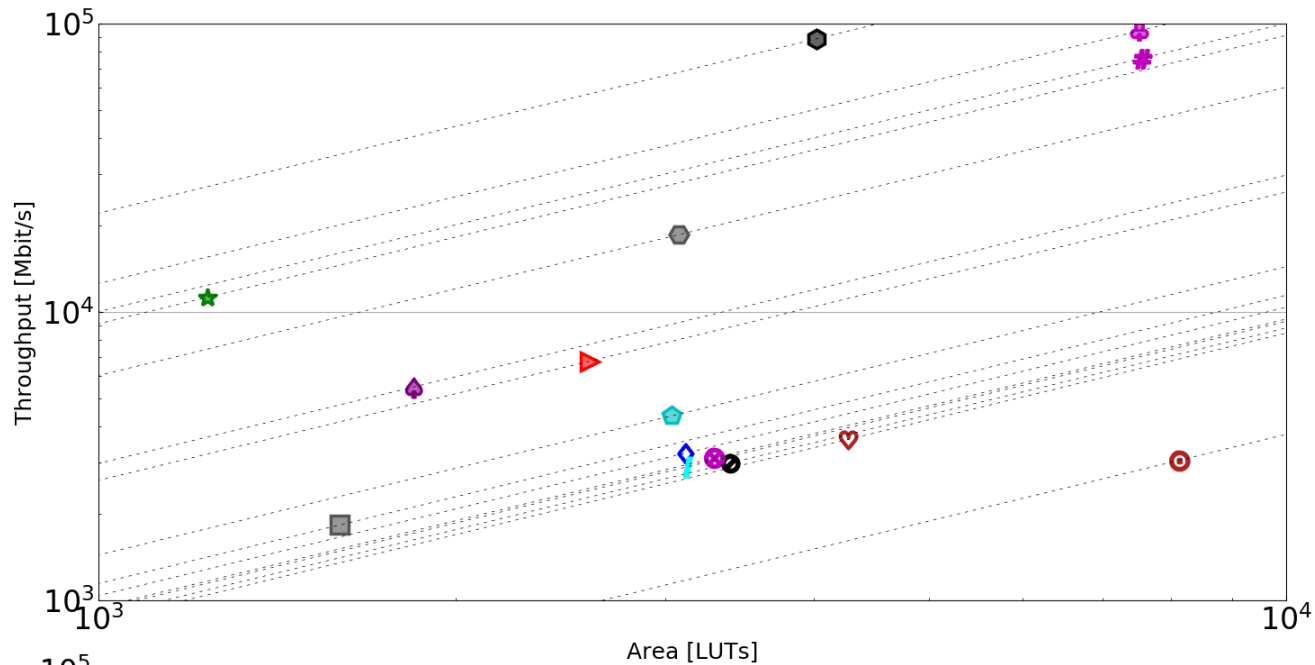
# RTL vs. HLS Throughput/Area [(Mbits/s)/LUTs]



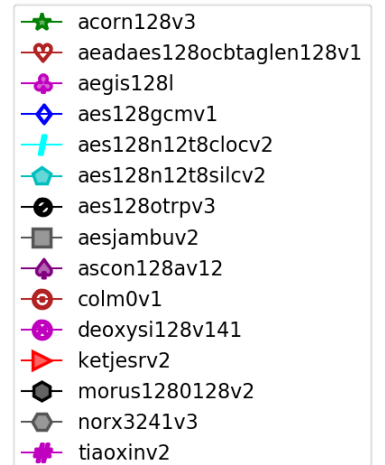
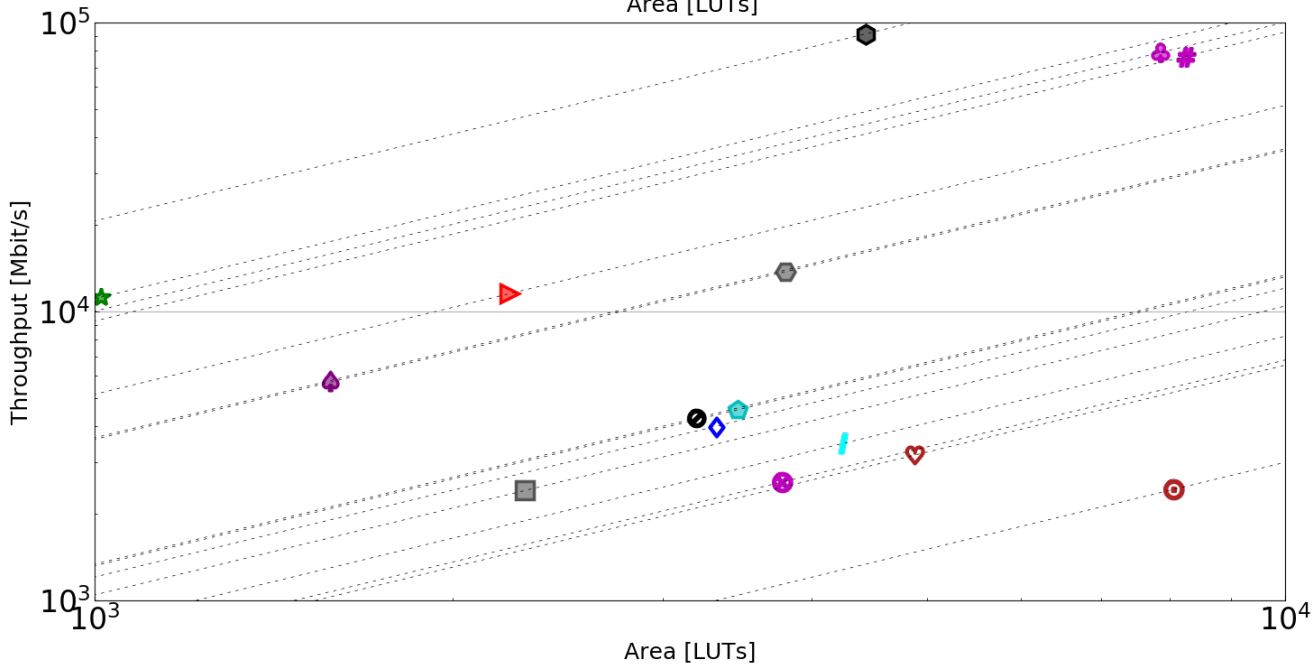


# HLS vs. RTL Throughput vs. Area

RTL



HLS

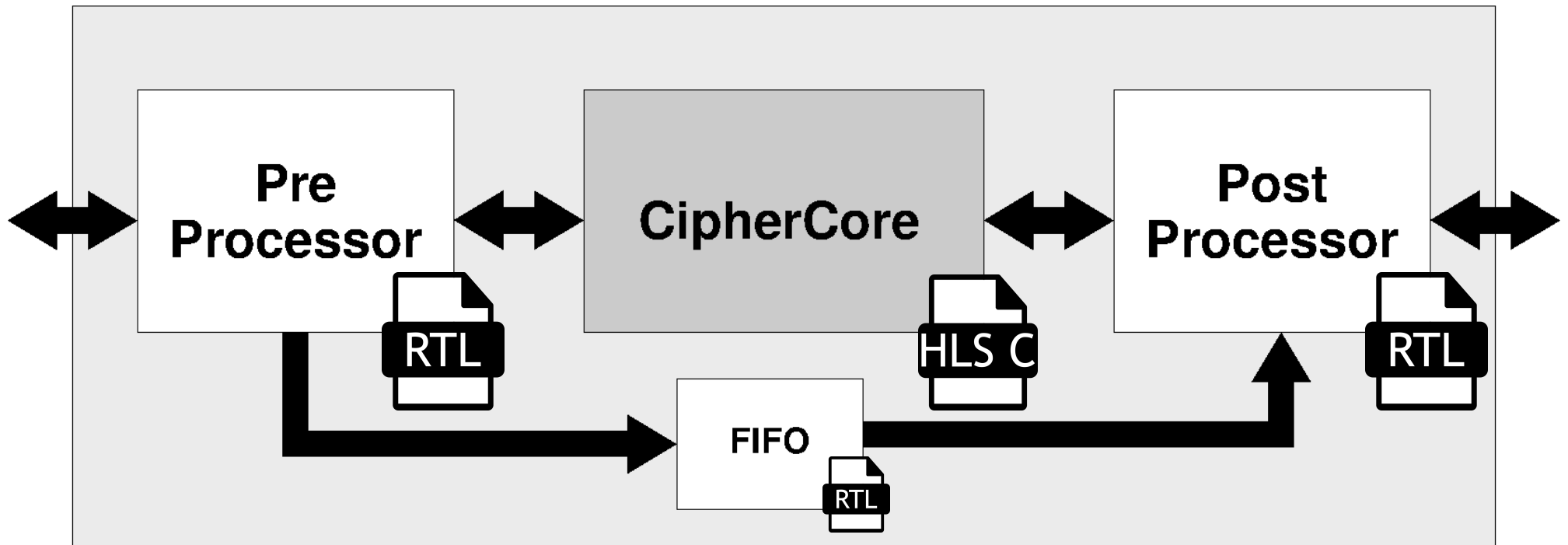


# Transformation to HLS-ready C/C++ Code

---

1. Language partitioning and interface mapping
2. Addition of HLS Tool directives (pragmas)
3. Hardware-driven code refactoring

# Language Partitioning



# Mapping Hardware to Software Interface

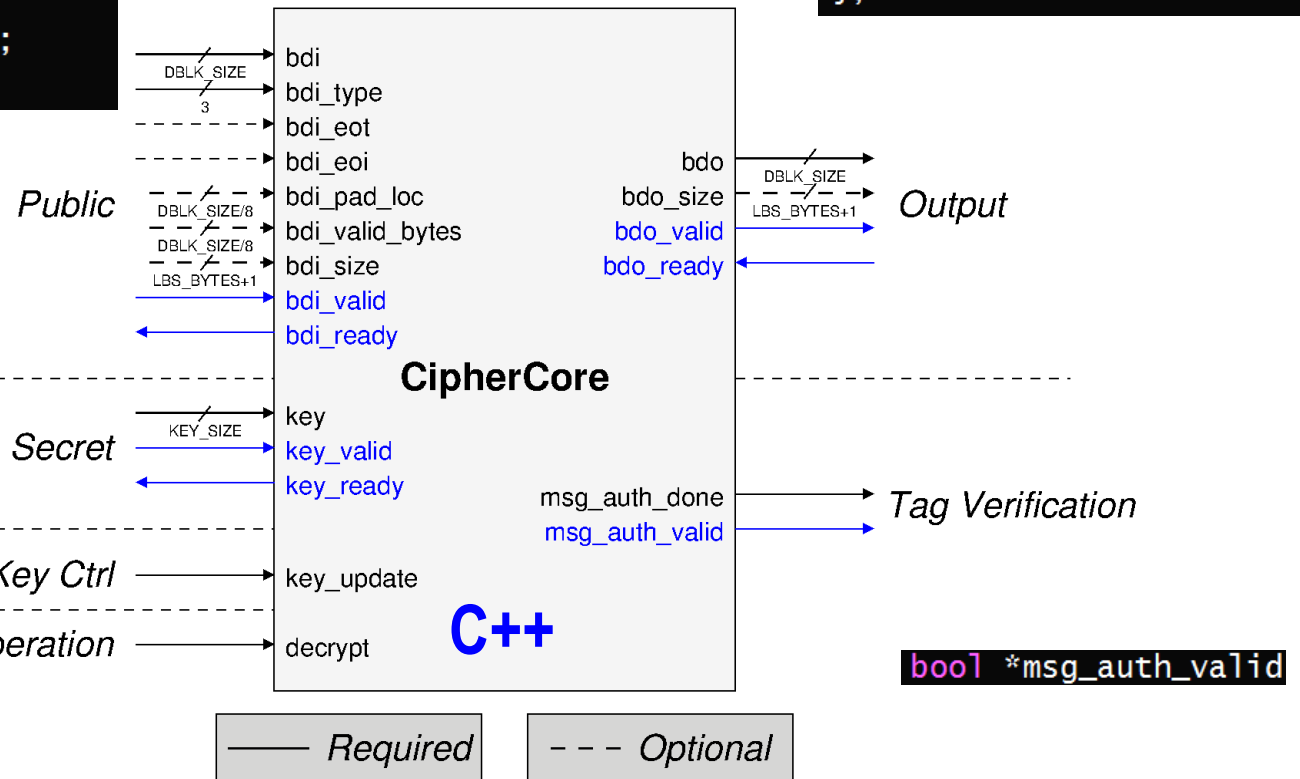
```
struct public_bus {
    data_t      data;
    data_bytes_t valid_bytes;
    data_bytes_t pad_loc;
    bsize_t     size;
    ap_uint<3>  type;
    bool        eoi;
    bool        eot;
    bool        partial;
};
```

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

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

```
volatile bool key_update
```

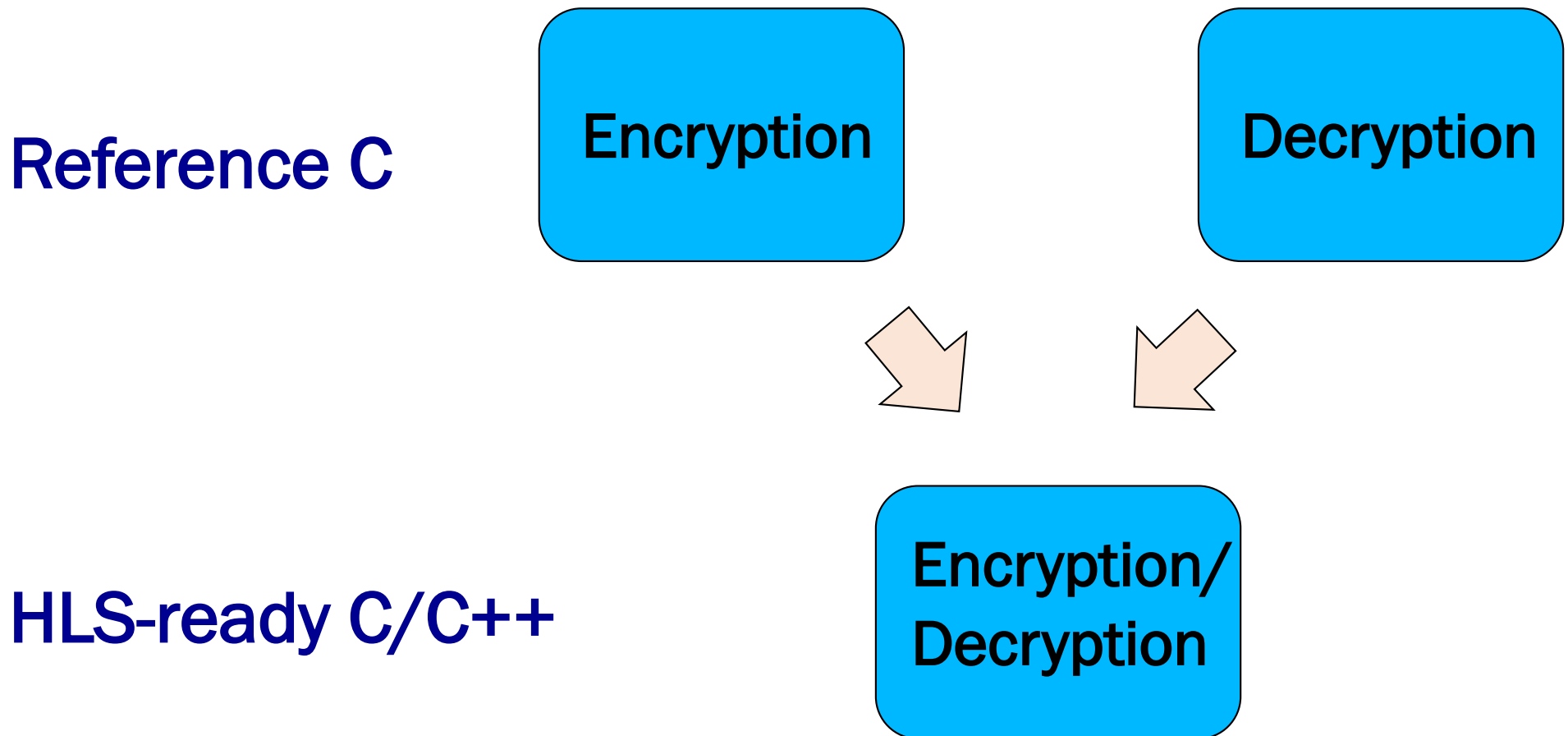
```
bool decrypt
```



Basic handshaking signals (valid, ready) added automatically

# Code Refactoring – High-Level

---



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:

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

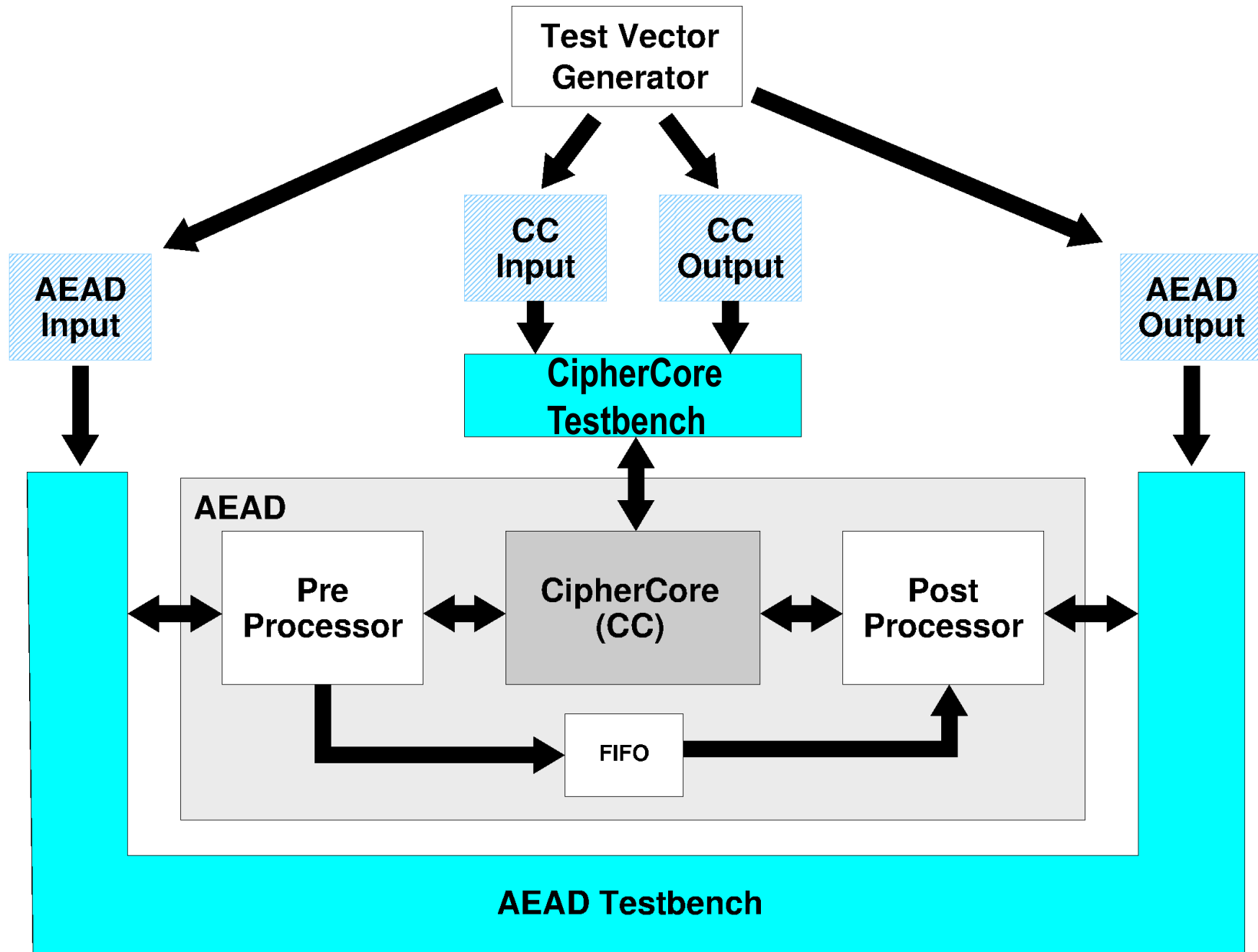
## Flattening function's hierarchy:

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

## Change array shapes:

```
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

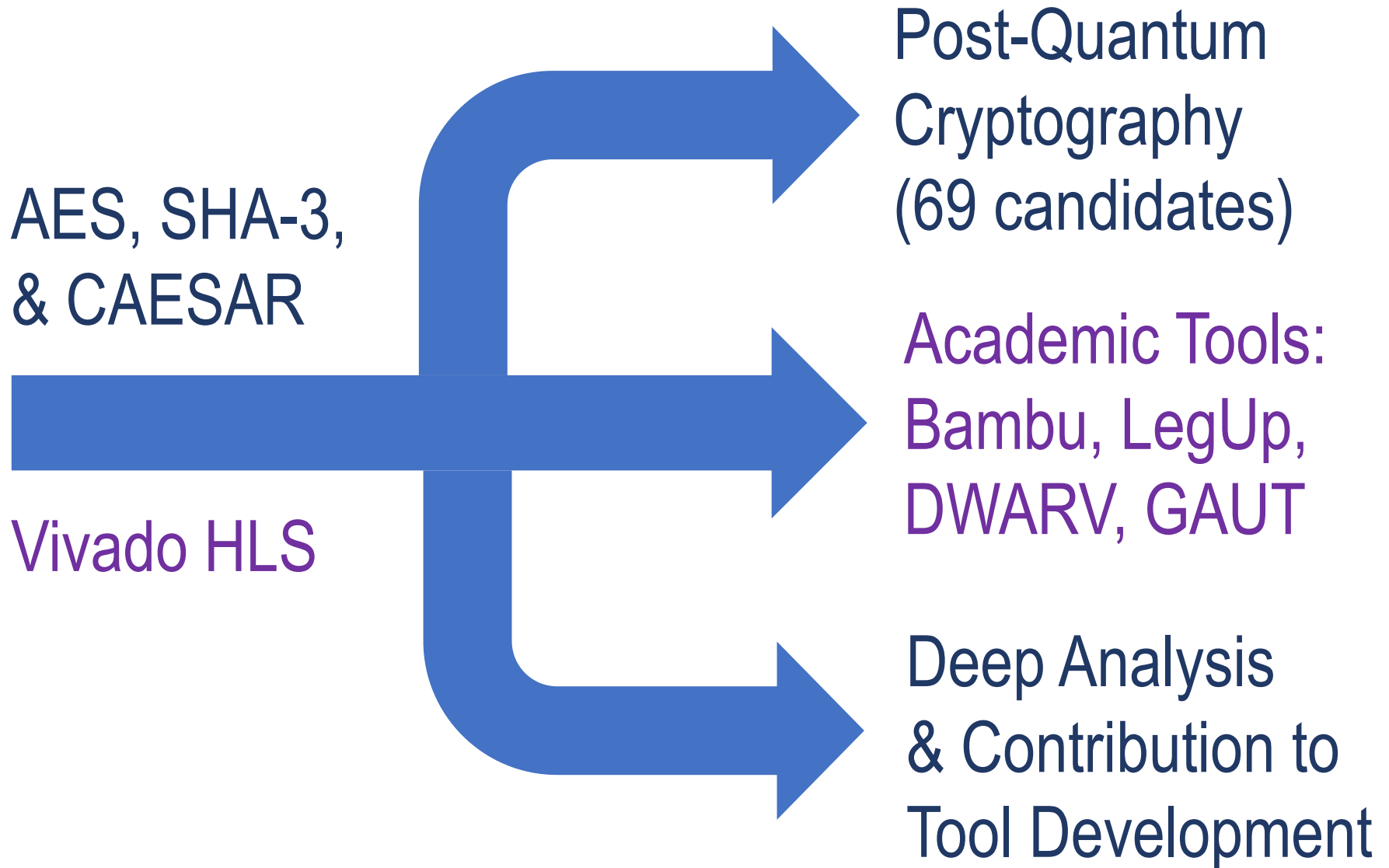
<https://cryptography.gmu.edu/athena>

under CAESAR ⇒

GMU Implementations of Authenticated Ciphers  
and Their Building Blocks

# Future Work: High-Level Synthesis

---



## Thank You!

Questions?



Comments?

Suggestions?

CERG: <http://cryptography.gmu.edu>

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