High-Level Approaches to Hardware Security

Ramesh Karri
Polytechnic School of Engineering
rkarri@nyu.edu
http://cyber.nyu.edu

http://cyber.nyu.edu/
Mission

NYU Center for Cybersecurity (CCS) is an interdisciplinary center dedicated to:

- **Research** technical and other means to secure the cyber infrastructure
- **Educate** the next generation of cybersecurity professionals and
- **Shape** public discourse on the policy and legal aspects of cybersecurity.

NYU has a Reputation in Cyber Security

- One of the earliest to offer degrees in Cyber Security (circa 1998)
- Triple distinction
  - NSA Center of Excellence in Information Assurance Education
  - NSA Center of Excellence in Information Assurance Research
  - NSA Center of Excellence in Cyber Operations
- MS in Cybersecurity (Cyberscholars for US residents)
- MS in Cyberrisk
- Bridge to Cyber
- Significant funding for research/education over 10 years.
- Scholarship for Service
  - Strong research and training partnership with federal agencies.
  - Placed over 100 students in all agencies of the Govt.
- Signature programs and partnerships.
Signature Outreach Programs

- Cyber Security Awareness Week (CSAW)
  - Celebrating its 14th year
  - Largest student cyber competition in US
  - Largest Capture the Flag
  - 20,000+ HS and college students
  - CTF, ESC, Best paper, High school forensics,...
  - MENA(NYU-AD), India (IIT Kanpur), Europe (Valence France and Israel (U. Haifa))
- Summer Cyber Boot-camp High School STEM Educators
- Sloan Speaker Series
- Hackers in Residence from Industry
- Hosts NSF/NSA CyberCorps Program ~ 100 in government service

Applications of Integrated Circuits

- Aerospace
- Communications
- Healthcare
- Energy
- Appliances
- Consumer electronics
- Industrial Control
Securing Electronics Supply Chain

High-Level Synthesis (HLS/ESL)
HLS is a Productivity Tool

![Graph showing the relationship between productivity/complexity and process technology nodes.](Image)

Gates / cm² Moore's Law (59% CAGR)

Design Productivity (20-25% CAGR)

Source: Semico Research Corp.

---

3rd Party IPs in a Design

![Graph showing the increase in the average number of IP blocks with process technology nodes.](Image)

Average Number of IP Blocks

Process Technology Node (nanometers)

(16, 90)

(190, 22)

(International Business Strategies, 2012)
Accelerator-based Design!

HLS Design Flow

int main(int X, int *Y, int *Z1, int *Z2 : num16) {
    int int1 = (X * K1);
    Y = biquad(in1, K2, K3, K4, K5, *Z1, *Z2);
    return Y;
}

int biquad(int in, int a1, int a2, int b1, int b2, int *Z1, int *Z2) {
    int state = in + (a1 * *Z1) + (a2 * *Z2);
    return state + (b1 * *Z1) + (b2 * *Z2);
}

c-specification of biquad filter  Scheduling and binding  Finite state machine

Datapath+controller
Security-Aware HLS

Semantic Information

Hard to secure

C/C++ → HLS → RTL → Logic Synthesis → Netlist

for (i=0; i<N; i++)
c[i] = a[i]+b[i];
.....

always @ posedge clk
a[i] <= b[i] + c[i];
.....

High-Level View of Security

• Promising to add security constraints
• HLS in Hardware vs Programming Language/Compilers in Software
• Semantics: constants, operators, control flow, dependencies (sensitive IP)

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm-Level (HLS)</td>
<td>Programming Lang (Compiler)</td>
</tr>
<tr>
<td>RT Level</td>
<td>Assembly (HEX)</td>
</tr>
<tr>
<td>Gate Level</td>
<td>Binary</td>
</tr>
<tr>
<td>Layout</td>
<td></td>
</tr>
</tbody>
</table>
HLS Design Flow

```c
int main (int X, int *Y, int *Z1, int *Z2 : num16) {
    int in1 = (X * K1);
    Y = biquad(in1, K2, K3, K4, K5, *Z1, *Z2);
    return Y;
}
int biquad(int in, int a1, int a2, int b1, int b2, int *Z1, int *Z2) {
    int state = in + (a1 * in) + (a2 * *Z1);
    return state + (b1 * *Z1) + (b2 * *Z2);
}
```

Can ESL undermine security of the design?

Datapath+controller

Threat: Reverse Engineering

- Legal: to detect piracy
  - Identify device technology, functionality, design
  - Chipworks
- Illegal: piracy, IP theft and Trojan insertion
  - Malicious user or Malicious SoC integration house or Malicious foundry
**Attack: HLS-informed Rev. Engg.**


**Security Metric: # of CDFGs**

before attack: $2^{53}$ CDFGs
Security Metric: # of CDFGs

before attack: $2^{53}$ CDFGs

after attack: 1 CDFG

Datapath Constraints
Security Metric: # of CDFGs

<table>
<thead>
<tr>
<th>Design</th>
<th>ESL Constraints</th>
<th># 1</th>
<th># 1 - # 4</th>
<th># 1 - # 6</th>
<th># 1 - # 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>BQF</td>
<td></td>
<td>253</td>
<td></td>
<td>243</td>
<td></td>
</tr>
<tr>
<td>Arai</td>
<td></td>
<td>247</td>
<td></td>
<td>2118</td>
<td>244</td>
</tr>
<tr>
<td>Chem</td>
<td></td>
<td>23526</td>
<td></td>
<td>2606</td>
<td></td>
</tr>
<tr>
<td>Dir</td>
<td></td>
<td>2717</td>
<td></td>
<td>2606</td>
<td></td>
</tr>
<tr>
<td>Feig_det</td>
<td></td>
<td>23790</td>
<td></td>
<td>2812</td>
<td>23</td>
</tr>
<tr>
<td>Honda</td>
<td></td>
<td>2435</td>
<td></td>
<td>2118</td>
<td>244</td>
</tr>
<tr>
<td>Lee</td>
<td></td>
<td>25716</td>
<td></td>
<td>2118</td>
<td>244</td>
</tr>
<tr>
<td>Mcm</td>
<td></td>
<td>1000000</td>
<td></td>
<td>2118</td>
<td>244</td>
</tr>
<tr>
<td>Pr</td>
<td></td>
<td>2353</td>
<td></td>
<td>2118</td>
<td>244</td>
</tr>
<tr>
<td>Wang</td>
<td></td>
<td>2383</td>
<td></td>
<td>2118</td>
<td>244</td>
</tr>
<tr>
<td>Snow3g</td>
<td></td>
<td>280</td>
<td></td>
<td>2118</td>
<td>244</td>
</tr>
<tr>
<td>Kasumi</td>
<td></td>
<td>7557749</td>
<td></td>
<td>752363</td>
<td>29</td>
</tr>
<tr>
<td>MD5c</td>
<td></td>
<td>2100000</td>
<td></td>
<td>272105</td>
<td>29</td>
</tr>
<tr>
<td>AES</td>
<td></td>
<td>2100000</td>
<td></td>
<td>2598662</td>
<td>29</td>
</tr>
</tbody>
</table>

# of CDFGs reduce drastically using HSL constraints

Belled the CAD!

<table>
<thead>
<tr>
<th>Design</th>
<th>Tools A, B, C, D &amp; E: Non-pipelined and Resource-Constrained</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of compare points</td>
</tr>
<tr>
<td>BQF</td>
<td>16</td>
</tr>
<tr>
<td>Arai</td>
<td>128</td>
</tr>
<tr>
<td>Chem</td>
<td>240</td>
</tr>
<tr>
<td>Dir</td>
<td>128</td>
</tr>
<tr>
<td>Feig_det</td>
<td>1024</td>
</tr>
<tr>
<td>Honda</td>
<td>128</td>
</tr>
<tr>
<td>Lee</td>
<td>128</td>
</tr>
<tr>
<td>Mcm</td>
<td>128</td>
</tr>
<tr>
<td>Pr</td>
<td>128</td>
</tr>
<tr>
<td>Wang</td>
<td>128</td>
</tr>
<tr>
<td>Snow3g</td>
<td>32</td>
</tr>
<tr>
<td>Kasumi</td>
<td>64</td>
</tr>
<tr>
<td>MD5c</td>
<td>128</td>
</tr>
<tr>
<td>AES</td>
<td>128</td>
</tr>
</tbody>
</table>

All benchmarks reverse engineered in <30 minutes. Functionally equivalent and structurally identical!
Threat: Malicious 3PIP (Trojans)

- 3PIP vendors are not trusted; may insert trojans
  - Trojans cause wrong outputs
  - Distributed: in different modules from same vendor may collude
- SoC integrator is trusted
  - SoC integrator uses components from 3PIP vendors
  - 3PIPs are integrated into a system and synthesized
- SoC is manufactured at an off-shore foundry
- The manufactured hardware is tested and deployed

HLS-based Trojan Detection

While (x < a) {
  x1 = x + dx
  u1 = u – 3xudx – 3ydx
  y1 = y + udx
  x = x1; u = u1; y = y1
}

Control Data Flow Graph
Detect “Natural” Faults

Does reliability ensure security?
Malicious 3PIPs

3PIP Designer $\rightarrow$ SoC Integrator

$\rightarrow$ Multiplier (RTL)

Synthesis $\rightarrow$ Integration

Detect 3PIPs: Duplicate and Check

- Components with Trojans produce same “malicious” outputs
- Checkers cannot detect malicious outputs
- Violates assumption for reliability
• Components with Trojans produce same “malicious” outputs
• Checkers cannot detect malicious outputs
• Violates assumption for reliability

Collude (a.k.a Distributed Trojans)

- Wireless Video Capture SoC monitors a building entrance
- Normal: CPU processes camera output → generates video frames → crypto engine encrypts frames → UART transmits to control room
- In the control room, the frames are decrypted and viewed


(Parent-Child) Collusion

- Timer and bus controller obtained from malicious vendor
- Normal operation: Bus contr. controls bus when timer expires
- Malicious operation
  - Timer sends a trigger (within its packet) to bus contr.
  - Trojan in the bus contr. puts the bus in tri-state
  - Output of the SoC freezes
  - Attacker sneaks into the building
- Timer (parent module) colludes with bus contr. (child module)
Prevent Collusion

- Prevent collusions: Map operations to diverse components
- Parent-Child collusion: Map parent, child ops on diverse components

Prevent Collusion

- Prevent collusions: Map operations to diverse components
- Parent-Child collusion: Map parent, child ops on diverse components
- Parent-Parent collusion: Map at least one parent on a component from a different vendor
## # of Potential Vulnerabilities

<table>
<thead>
<tr>
<th>Design</th>
<th># of Ops</th>
<th># of Comm. Paths</th>
<th># of Potentially Untrustworthy IPs</th>
<th># of Parent to Child Collusion</th>
<th># of Parent to Parent Collusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diff, Eq</td>
<td>17</td>
<td>8</td>
<td>17</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Conv3X3</td>
<td>514</td>
<td>413</td>
<td>514</td>
<td>413</td>
<td>204</td>
</tr>
<tr>
<td>Cordic</td>
<td>194</td>
<td>338</td>
<td>194</td>
<td>338</td>
<td>247</td>
</tr>
<tr>
<td>DCT32</td>
<td>519</td>
<td>612</td>
<td>519</td>
<td>612</td>
<td>306</td>
</tr>
<tr>
<td>FIR16</td>
<td>63</td>
<td>30</td>
<td>63</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>Polynom</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Sobel</td>
<td>391</td>
<td>670</td>
<td>391</td>
<td>670</td>
<td>536</td>
</tr>
<tr>
<td>Ellipticlass</td>
<td>37</td>
<td>39</td>
<td>37</td>
<td>39</td>
<td>19</td>
</tr>
</tbody>
</table>

Opportunities to produce malicious outputs or opportunities to collude

## Detect 3PIPs: Duplicate+Diversify

Duplicate + Diversify: 3 vendors; 3 mults 4 adder/comparators/subs
Prevent Parent-Child Collusion and Parent-Parent Collusion
**Untrusted Foundry**

- Attacker capabilities
  - Is (in) the Foundry
  - Has the GDSII
  - Does not have access to a (activated/)functional IC
- Objective: Recover the design


**Algorithm Obfuscation**

```java
if (cond < N) {
    c[i] = a[i] + b[i];
    d[i] = c[i] * CONST_1;
    ...
} else { ... }
```

Several ways to obfuscate an algorithm
Algorithm Obfuscation

```
if (cond < N) {
    c[i] = a[i] + b[i];
    d[i] = c[i] * CONST_1;
    ...
} else { ... }
```

Obfuscate Control Flow

- Mask control condition with key bit
- Correct branch is taken only with correct key
- Reorder Branch: Ensures semantic equivalence + confuse attacker
Obfuscate Operations

- Gives intelligence on what the algorithm does
- Operator variants can camouflage correct operation
- Correct result is propagated only with the correct key

```
b[i] a[i] + b[i] = c[i]
```

"Fake" operations with same i/o

Input port decided by key

Obfuscate Constants

- Hard-coded values used by algorithm (coefficients, thresholds, …)
- Information is maintained at RTL
- Extensively optimized during logic synthesis

```
C/C++: d[i] = c[i] * CONST_1;
```

<table>
<thead>
<tr>
<th>Obfuscated</th>
<th>Not obfuscated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data co-efficients</td>
<td>Reset values</td>
</tr>
<tr>
<td>Signal extensions</td>
<td>Signal polarity</td>
</tr>
<tr>
<td>Mask values</td>
<td></td>
</tr>
</tbody>
</table>

No impact on security, less keys

No impact on semantics
Obfuscate Dependencies

- K-bit key is used to select $2^k$ DFG variants

Correct paths are activated only with the correct key

HLS Obfuscation

Integrate with HLS (e.g., Bambu) need access to HLS source

Design key

Compatible with RTL synthesis tool

Semantic Obfuscation: Branches, Dependencies, Operations, Constants
### Results

<table>
<thead>
<tr>
<th>Design name</th>
<th>Obfuscation</th>
<th># of key bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Constant</td>
<td>Branch</td>
</tr>
<tr>
<td>GSM</td>
<td>4 / 128</td>
<td>4</td>
</tr>
<tr>
<td>ADPCM</td>
<td>5 / 160</td>
<td>5</td>
</tr>
<tr>
<td>SOBEL</td>
<td>2 / 64</td>
<td>2</td>
</tr>
<tr>
<td>BACKPROP</td>
<td>12 / 384</td>
<td>11</td>
</tr>
<tr>
<td>VITERBI</td>
<td>117 / 3,744</td>
<td>9</td>
</tr>
</tbody>
</table>

**Bambu Open Source HLS (automatic generation from C-to-HDL)**

### Overhead

- Area overhead of each technique wrt the **baseline** version
  - Synopsys SAED 32nm @ 500 MHz
- Operation+Dependence obfuscation
**RTL Transformations for Security**

- **RTL Obfuscation**
- **Design key**
- **Synthesizable Verilog**
- **Compatible with any RTL synthesis tool**

**RTL Obfuscation: Results**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>add_only_decimator_par32x</td>
<td>80 / 240</td>
<td>24</td>
<td>32</td>
<td>296</td>
</tr>
</tbody>
</table>
### RTL Obfuscation: Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>add_only_decimator_par32x</td>
<td>80 / 240</td>
<td>24</td>
<td>32</td>
<td>296</td>
</tr>
<tr>
<td>aod_par32x_section0</td>
<td>64 / 208</td>
<td>100</td>
<td>128</td>
<td>436</td>
</tr>
<tr>
<td>aod_par32x_section1</td>
<td>32 / 120</td>
<td>69</td>
<td>80</td>
<td>269</td>
</tr>
<tr>
<td>aod_par32x_section2</td>
<td>16 / 68</td>
<td>55</td>
<td>56</td>
<td>179</td>
</tr>
<tr>
<td>coarse_time_delay_par2x</td>
<td>0 / 0</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>convert_to_cos_sin_32x_elem0</td>
<td>34 / 173</td>
<td>64</td>
<td>97</td>
<td>334</td>
</tr>
<tr>
<td>data_select_and_decimate_par4x</td>
<td>0 / 0</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>delay_i_par[2x0</td>
<td>2x1</td>
<td>4x0]</td>
<td>0 / 0</td>
<td>1</td>
</tr>
<tr>
<td>dotprod_par4x_16taps0</td>
<td>0 / 0</td>
<td>17</td>
<td>31</td>
<td>48</td>
</tr>
</tbody>
</table>

### Conclusions

1. **RTL is a promising level to Design-in Security**
2. **HLS can be used for Trojan Detection and Isolation**
3. **Watermark designs during High-Level Synthesis**
4. **Design obfuscation benefits from High-Level semantic information**
5. **Taint Propagation is seamless during HLS**
6. **HLS-generated designs can be reverse engineered !**
7. **One can use High-Level Synthesis for Black-Hat purposes**
   - C Pilato, K Basu, F Regazzoni, R Karri, Black-Hat High-Level Synthesis: Myth or Reality? IEEE Transactions on Very Large Scale Integration (VLSI) System, DOI: 10.1109/TVLSI.2018.2884742
Security: A Summary

Sensitive IP: Constants, control flow, dependencies, operations, CDFGs

NIST Post-Quantum Cryptography-
A Hardware Evaluation Study
Kanad Basu, Deepraj Soni, Mohammed Nabeel, and Ramesh Karri

Abstract—Experts forecast that quantum computers can break classical cryptographic algorithms. Scientists are developing post-quantum cryptographic (PQC) algorithms, that are invulnerable to quantum computer attacks. The National Institute of Standards and Technology (NIST) started a public evaluation process to standardize quantum-resistant public key algorithms. The objective of our study is to provide a hardware-based comparison of the NIST PQC candidates. For this, we use a High-Level Synthesis (HLS)-based hardware design methodology to map high-level C specifications of round 2 PQC candidates into both FPGA and ASIC implementations.

I. INTRODUCTION
Public key cryptography is a fundamental security protocol for all forms of digital communication, wired or wireless. Public key cryptography has three main cryptographic functions, namely (a) public key encryption, (b) digital signatures, and (c) key exchange [1]. RSA and Elliptic Curve-based public

1) Developed systematic FPGA and ASIC design flows for PQC evaluation starting from a C specification.
2) Studied performance vs area trade-offs for 11 PQC algorithms, including lattice, code, hash, and multivariate-based KEM and Signature algorithms.
3) Improved the latency of PQC implementations using optimizations such as loop unrolling and loop pipelining.
4) Performed a detailed study of three signature algorithms to explore area vs performance vs security trade-offs.

The paper is organized as follows. Section II gives a background on Post-Quantum Cryptography. Section III describes the design flow and Section IV presents experimental results. Section V describes case studies using three signature-based algorithms and Section VI enumerates the key takeaways.
Security-Aware HLS

Cell: 917 363 9703
rkarri@nyu.edu
http://cyber.nyu.edu
void preprocess (int v) {
    struct results ret;
    if (v > 0) ret.x = 1;
    else ret.x = 4;
    ret.y = 10;
    ret.z = 5;
    return ret;
}

struct results {
    int x;
    int y;
    int z;
};

get_IO (&v);
...
ret = preprocess (v);
...
elaborate (ret);

- Information flow tracking allows identification of malicious uses
- No existing support for hardware accelerators for intrinsic DIFT


C/C++ parsing

IR analysis

Taint Regs
Taint Modules
Taint Mem
Interconnect

Code Generation
Taint-HLS: Area Overhead

![Bar chart showing area overhead for different applications on Xilinx Virtex-7 FPGA at 100 MHz. Baseline (no DIFT), Word-level DIFT, Variable-level DIFT, and Bit-level DIFT are compared. The chart indicates a +31% overhead for a specific application.]