black hat USA 2024

AUGUST 7-8, 2024

BRIEFINGS

# The Hack@DAC* Story: Learnings from Organizing the World's Largest Hardware Hacking Competition

**Arun Kanuparthi**, Hareesh Khattri, Jason Fung (Intel Corporation, USA)

JV Rajendran (Texas A&M University, USA), Ahmad-Reza Sadeghi (TU Darmstadt, Germany)

*Design Automation Conference

# The Team

**Arun Kanuparthi**
Principal Engineer,
Offensive Security Researcher
Intel Corporation, USA

Hareesh Khattri
Principal Engineer,
Offensive Security Researcher
Intel Corporation, USA

Jason Fung
Sr. Director
Offensive Security Research
Intel Corporation, USA

## Offensive Security Research at Intel

- 50+ years of combined experience
- CPUs, Servers, Clients, Networking, Cellular, Storage, Security technologies, …
- 500+ vulnerabilities identified
- Vulnerability root causing and categorization
- MITRE HW CWE SIG* members

Jeyavijayan (JV) Rajendran
Associate Professor
Texas A&M University, USA

Ahmad-Reza Sadeghi
Professor
TU Darmstadt, Germany

## Security Research

- 35+ years of combined experience
- Circuits, system security, network security, cryptography, microarchitecture, etc.
- 44000+ citations!

*Special Interest Group (SIG)

# Full Team

**Texas A&M University**

- Rahul Kande*
- Chen Chen*
- Patrick Haney
- Garrett Persyn
- Bhagyaraja Adapa

**TU Darmstadt**

- Mohammadreza Rostami*
- Ghada Dessouky
- David Gens
- Pouya Mahmoody
- Shaza Zeitouni

**Synopsys**

- Shylaja Sen*
- Yann Antonioli*
- Jagminder Chugh
- Meriav Nitzan

* Part of most recent team

# Overview

- Introduction
- Value of Organizing HW CTFs
- How Hack@DAC is Unique
- Organizing Hack@DAC
- Key Takeaways & Summary

# Overview

- Introduction
- Value of Organizing HW CTFs
- How Hack@DAC is Unique
- Organizing Hack@DAC
- Key Takeaways & Summary

# Computing Stack - Refresher

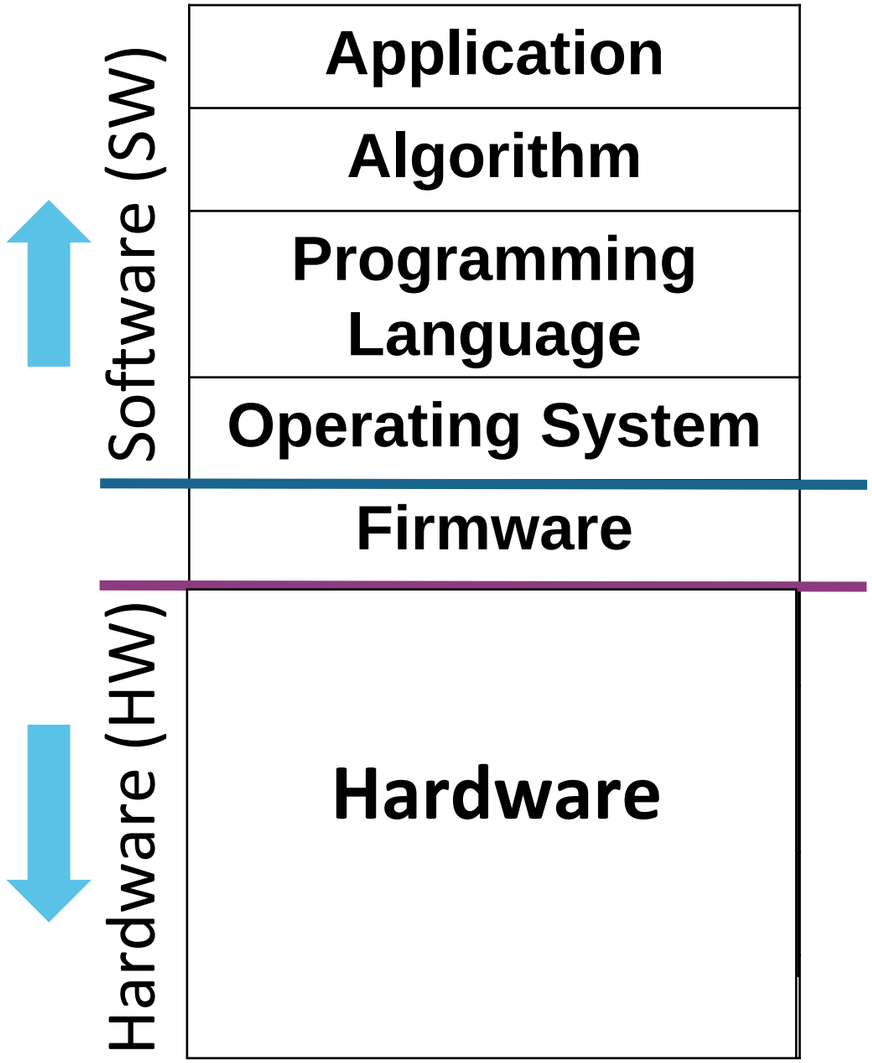





Image3 Source Image4 Source

# Computing Stack - Refresher

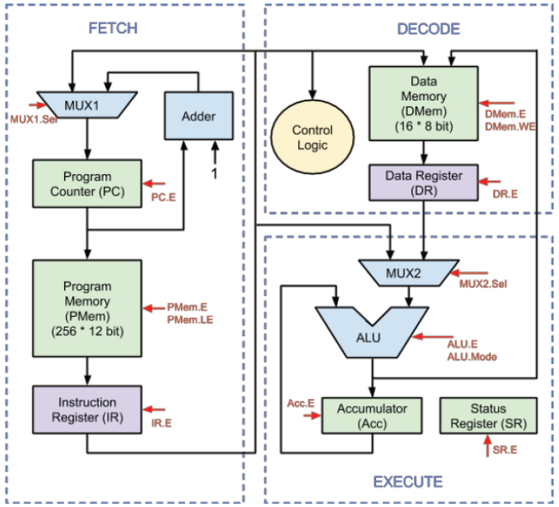
Software (SW)

- Application
- Algorithm
- Programming Language
- Operating System

Firmware

Hardware (HW)

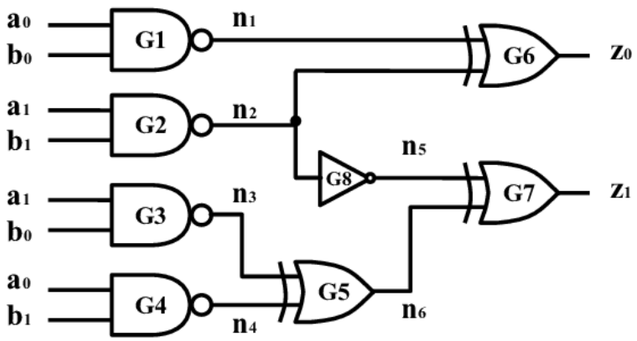
- Microarchitecture
- Register Transfer Level (RTL)
- Gate Level
- Transistor



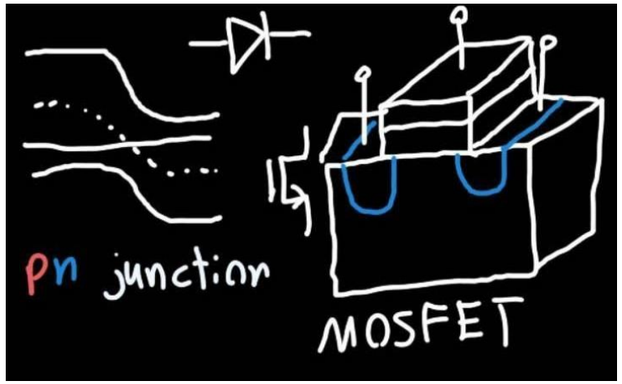

Microarchitecture

```
assign ADD_result = reg_A + reg_B;
assign SUB_result = reg_A – reg_B;
assign AND_result = reg_A & reg_B;
…
if (IR_opcode_field == 0)
        case (IR_function_field)
                6'b100000: ALU_result <= ADD_result;
                6'b100010: ALU_result <= SUB_result;
                6'b100100: ALU_result <= AND_result;
…
```

Register Transfer Level (RTL)




Gate Level

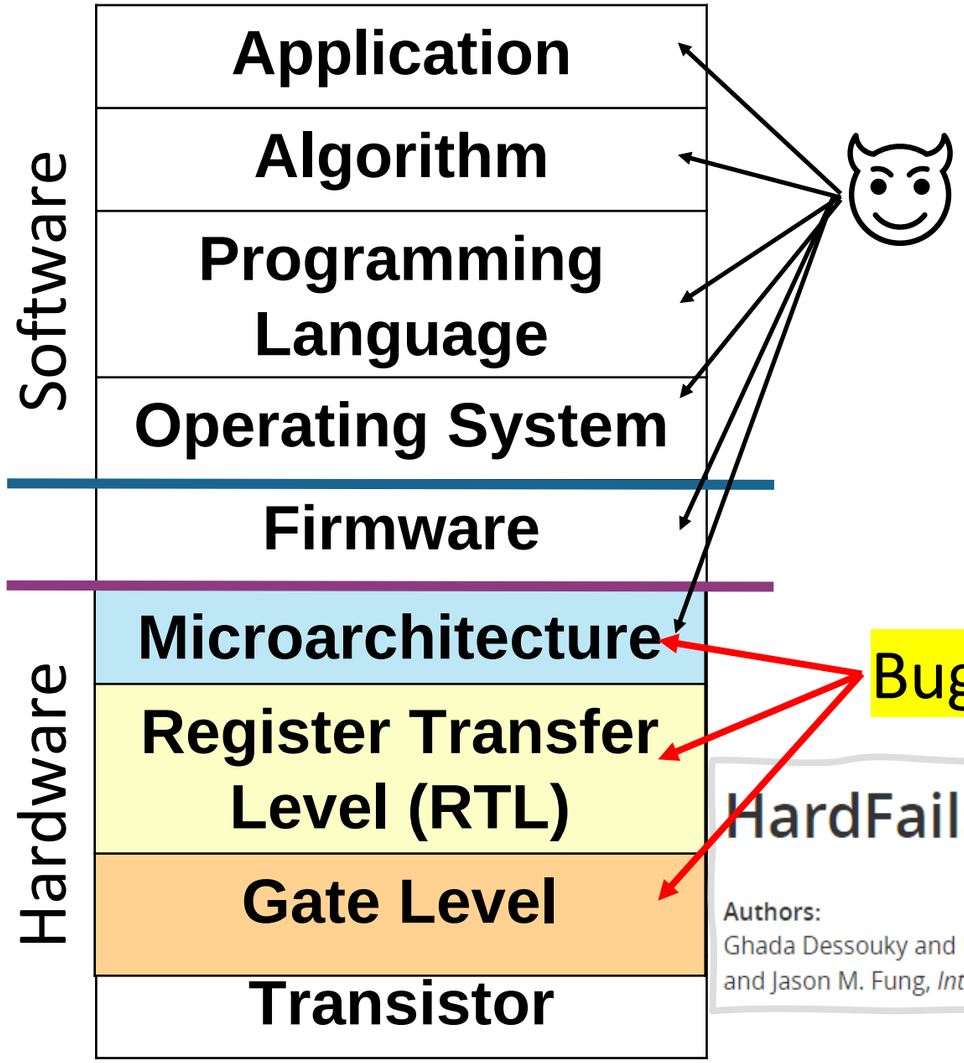



Transistor



Image3 Source Image4 Source

# Challenge #1: Limited Awareness of HW Security Weaknesses




RACE TO THE BOTTOM

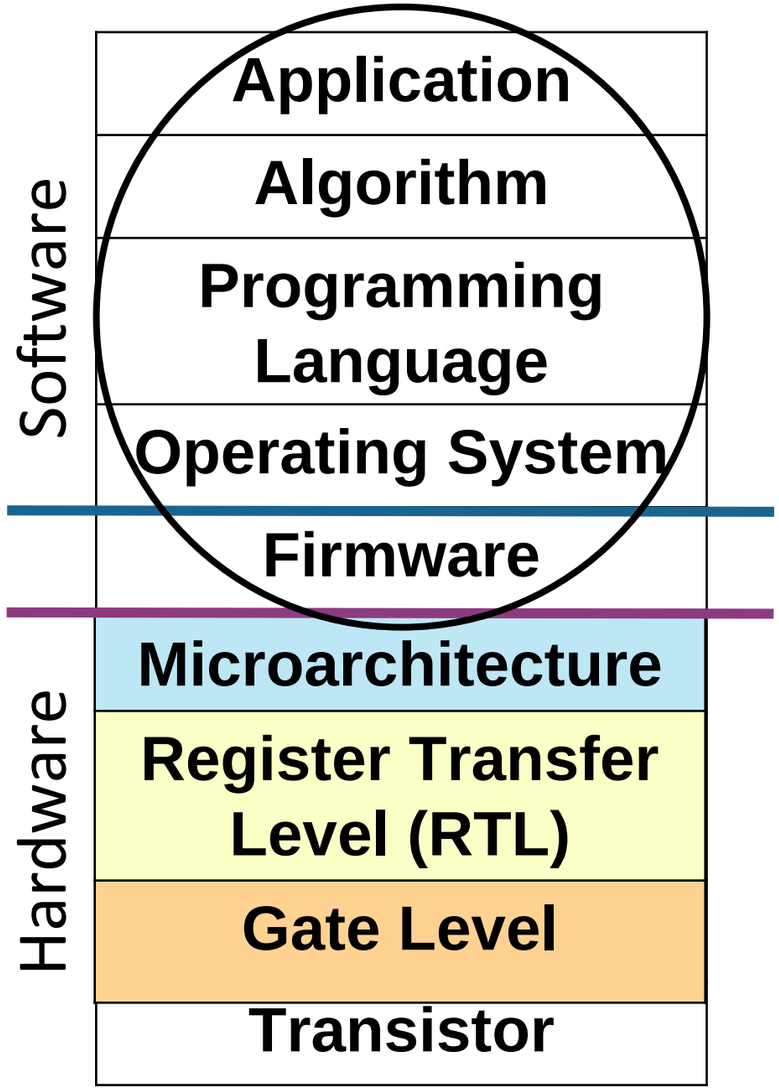
Bugs in hardware could be exploitable by software!

HardFails: Insights into Software-Exploitable Hardware Bugs

Authors:
Ghada Dessouky and David Gens, *Technische Universität Darmstadt;* Patrick Haney and Garrett Persyn, *Texas A&M University;* Arun Kanuparthi, Hareesh Khattri, and Jason M. Fung, *Intel Corporation;* Ahmad-Reza Sadeghi, *Technische Universität Darmstadt;* Jeyavijayan Rajendran, *Texas A&M University*

*USENIX Security 2019*

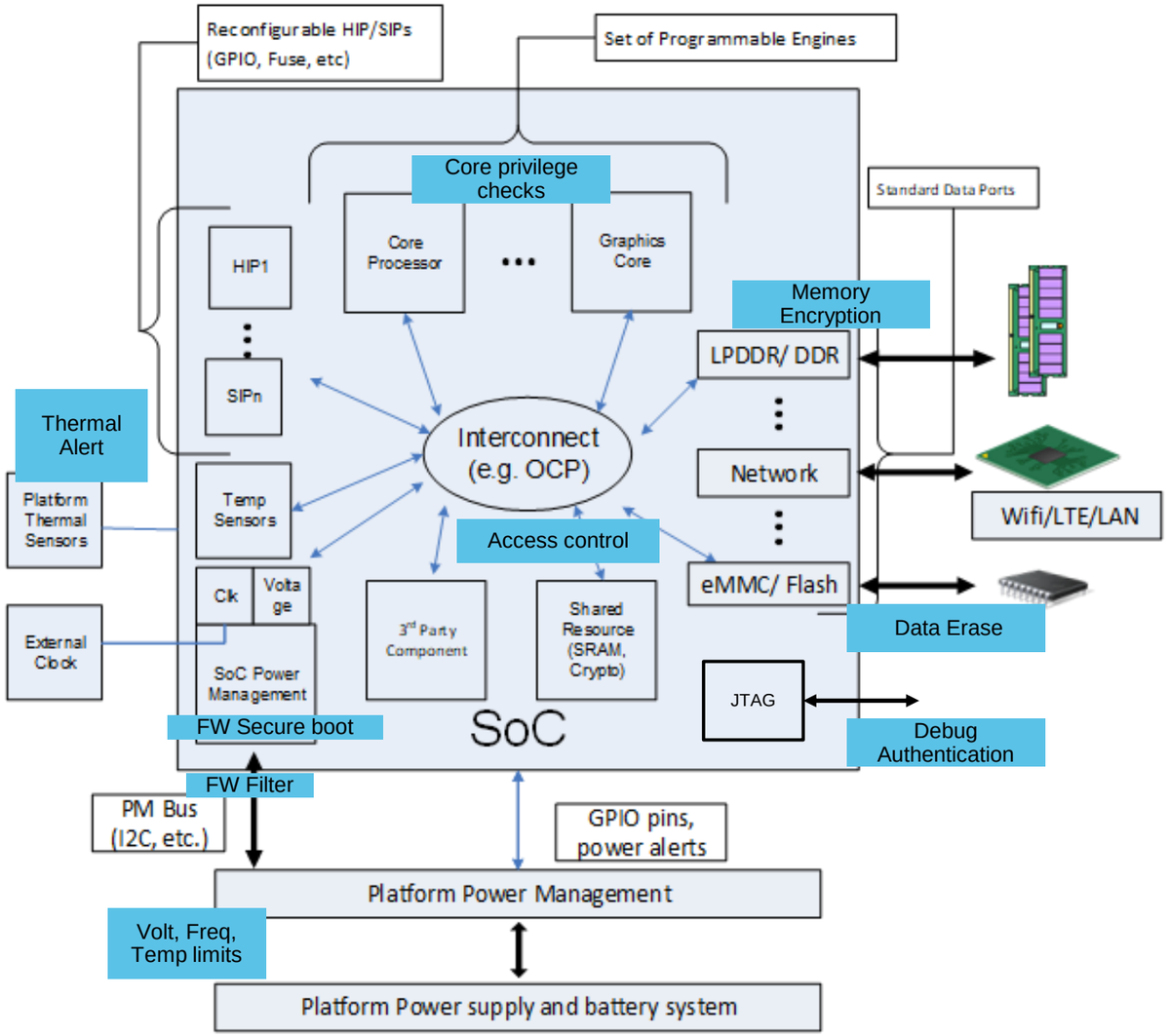
# Tools for Security – SW vs HW




Lots of tools for SW/FW security!

- Code scanners
- Protocol checkers
- Configuration checkers
- Decompilers & RE tools

# System on a Chip (SoC) Security




- Typical Security Objectives for Assets
  - Data Confidentiality
  - Data Integrity
  - Availability
- Example Security Features
  - Execution core & debug privilege checks
  - Access control
  - Memory encryption & integrity
  - Secure data erase
  - Power and thermal critical trip alerts

```
module aes0_wrapper #(
    parameter int unsigned AXI_ADDR_WIDTH = 64,
    parameter int unsigned AXI_DATA_WIDTH = 64,
    parameter int unsigned AXI_ID_WIDTH   = 10
)(
        clk_i,
        rst_ni,
        reglk_ctrl_i,
        acct_ctrl_i,
        debug_mode_i,
        axi_req_i,
        axi_resp_o
    );
```

# HW Vulnerability Example – Key Clear

| | |
|---|---|
| **Asset (Objective)** | Secret Keys in AES block (Confidentiality) |
| **Threat** | HW debug adversary extracts keys |
| **Mitigation** | Return all 0s for all reads when chip is in debug mode |

When in debug mode, return 0 when keys are read

debug_mode is not checked for key_big2

Attacker can extract key_big2 during debug

Security Impact

```
// signals from AXI 4 Lite
logic [AXI_ADDR_WIDTH-1:0] address;
logic                      en, en_acct;
logic                      we;
logic [63:0] wdata;
logic [63:0] rdata;

assign p_c_big    = {p_c[0], p_c[1], p_c[2], p_c[3]};
assign state_big  = {state[0], state[1], state[2], state[3]};
assign key_big0   = debug_mode_i ? 192'b0 : {key0[0], key0[1], key0[2], key0[3], key0[4], key0[5]};
assign key_big1   = debug_mode_i ? 192'b0 : {key1[0], key1[1], key1[2], key1[3], key1[4], key1[5]};
assign key_big2   = {key2[0], key2[1], key2[2], key2[3], key2[4], key2[5]};
```

https://github.com/PrincetonUniversity/openpiton

# HW Vulnerability Example – Key Leak

| | |
|---|---|
| Asset (Objective) | Secret Keys in AES block (Confidentiality) |
| Threat | Keys should not be readable by untrusted software code |
| Mitigation | A read lock signal (when enabled) returns ‘0’ when keys are read by software |

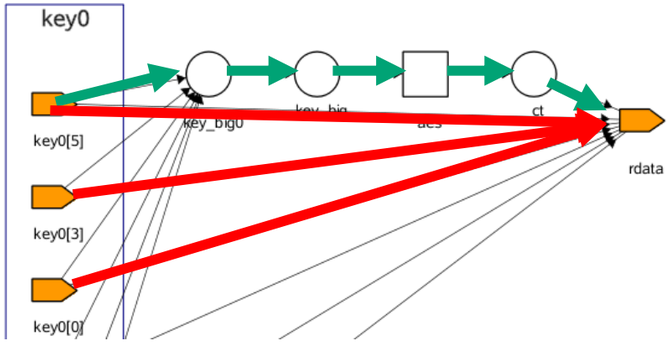
Logic to read the key. If lock is set, reads return ‘0’

When valid read is detected (en is high), key0 is passed to read data without checking for lock

Key0 leaks to attacker observable interface

Security Impact

```
// Read side
//always @(~write)
always @(*)
    begin
      rdata = 64'b0;
      if (en) begin
        rdata = key0[address[8:3]];
        case(address[8:3])
            0:
                rdata = reglk_ctrl_i[0] ? 'b0 : {31'b0, start};
            1:
                rdata = reglk_ctrl_i[2] ? 'b0 : p_c[3];
            2:
                rdata = reglk_ctrl_i[2] ? 'b0 : p_c[2];
```



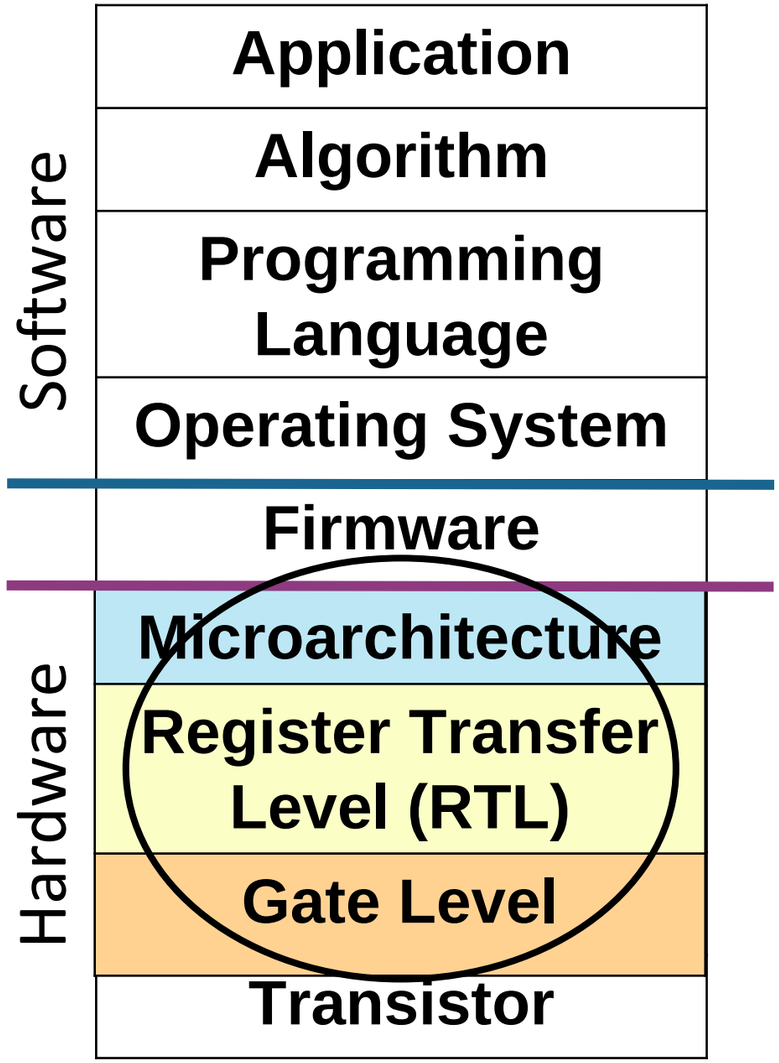

Expected Path through AES engine

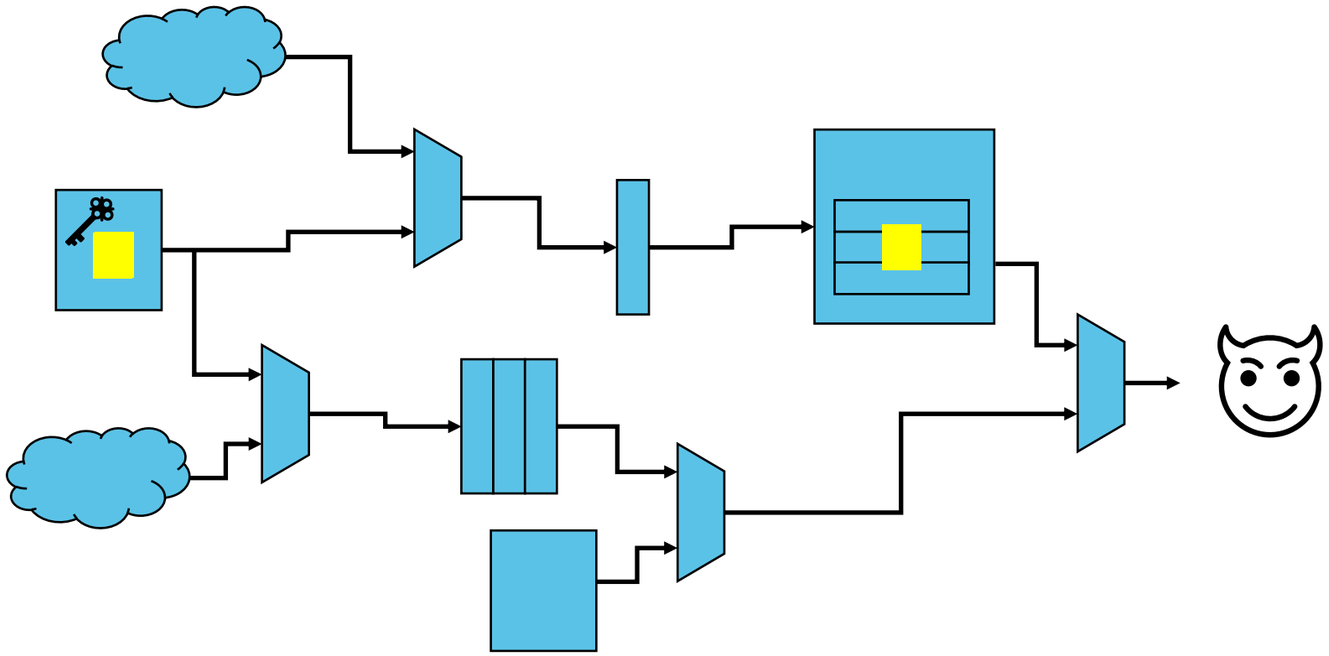
Unexpected/hidden path leaks key

https://github.com/PrincetonUniversity/openpiton

# Tools for Security – SW vs HW

## Challenge #2: Need for Security-Aware Design Automation Tools



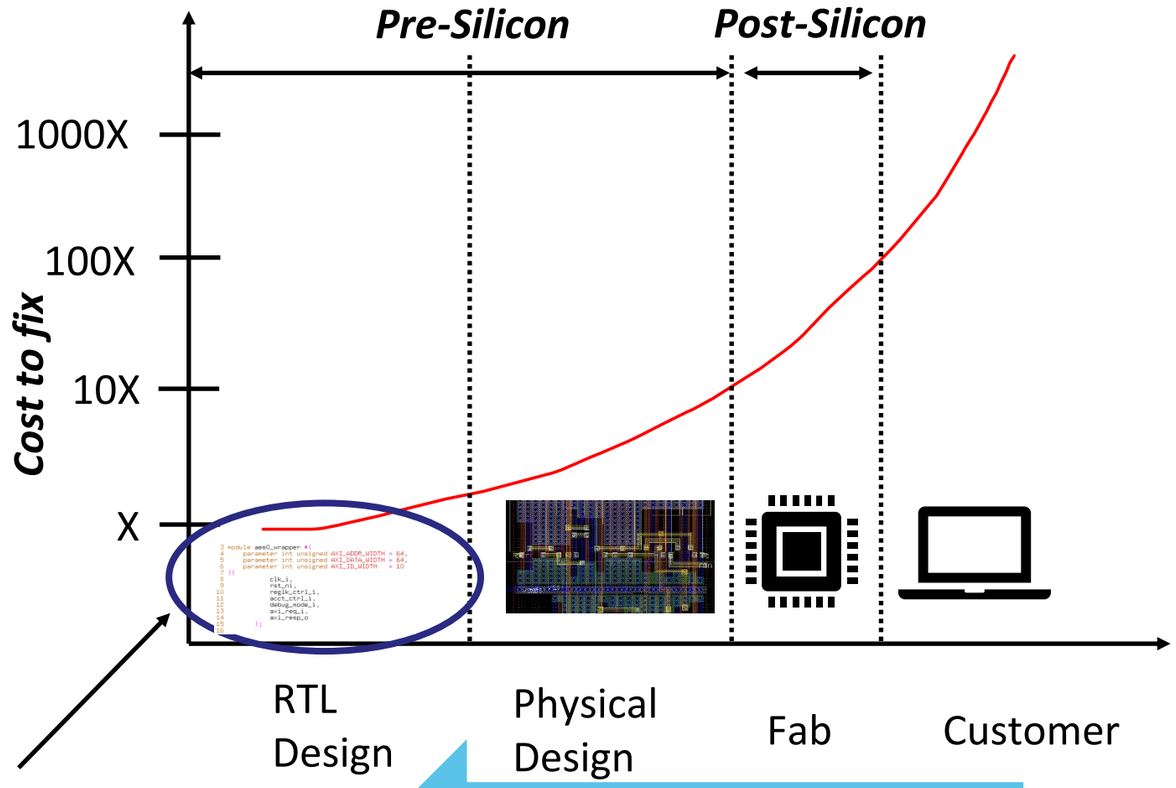



HW security tools (at RTL level) are limited

# Cost of Fixing Bugs

## Challenge #3: Need to Detect/Fix Bugs at RTL Design Phase

- SW bugs fixed with patches
- HW bugs are complicated to fix
  - Time consuming
  - Expensive
  - Cause brand damage




Fix bugs here!

"Shift Left"

# Motivation for Hack@DAC

Awareness of Hardware Common Weaknesses

**CONCEPTS**

Security-Aware Design Automation

**TOOLS**

"Shift-Left" to Detect & Fix Bugs in RTL

**BEST PRACTICES**

**Hack@DAC**

- Hackathons, trainings
- Open-source hardware as target?
- What about hardware CTF?

# Overview

- Introduction
- Value of Organizing HW CTFs
- How Hack@DAC is Unique
- Organizing Hack@DAC
- Key Takeaways & Summary

# Fostering Awareness for HW Security

- Continuous race between attackers and defenders
- Defenders need to up their game!
- Hardware CTFs foster greater awareness about
  - Common hardware security weaknesses
  - Constraints of chip design teams

Image source

# What's in it for Academia & Industry?

- A buggy SoC* framework for **furthering innovation**
  - Realistic security features, threat model, and security objectives
  - Vulnerabilities inspired by CVEs and real-world bugs
  - Open source and commercial tool support
- Benchmark for **developing and testing HW security tools**
  - Closest to commercial chip designs
- Participants **gain hardware security assurance experience**
  - Develop hacker mindset
  - Launchpad for researchers from adjacent areas (e.g., Firmware)

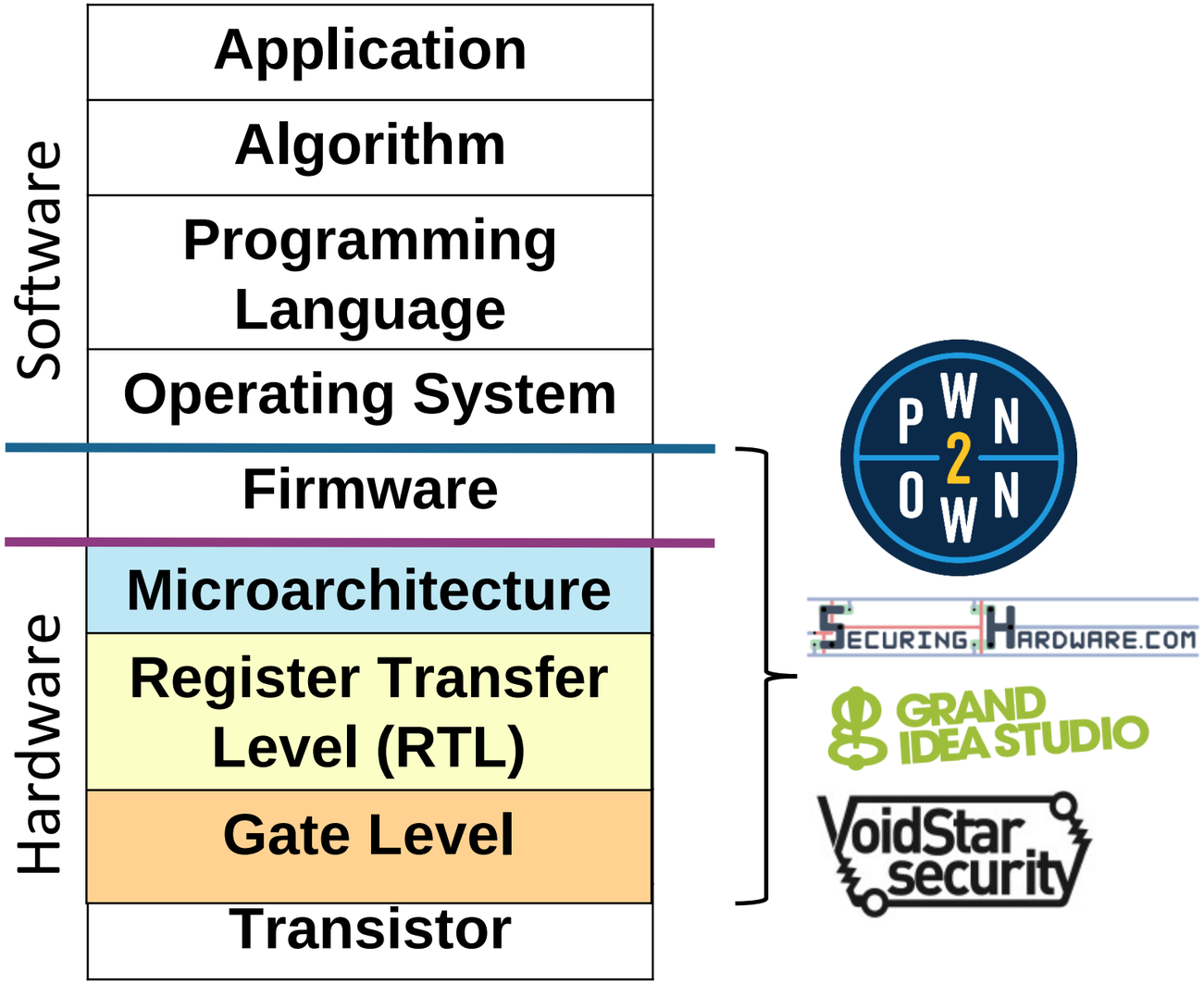
trust HUB

*SoC = System on a chip

# Overview

- Introduction
- Value of Organizing HW CTFs
- How Hack@DAC is Unique
- Organizing Hack@DAC
- Key Takeaways & Summary

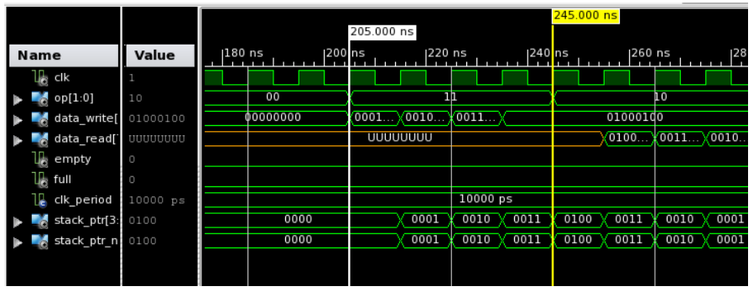
# Popular HW CTFs




- Popular HW CTFs are "closed-box"
- Adopt a hacker-centric approach
  - Involve physical interaction with target chip
    - Probing input/output ports
    - Desoldering and reverse engineering attacks
    - Physical side channel attacks, etc.
  - No insights into the RTL code of the chip
- Very important research!
- Does not address "shift-left" challenge

# Closed-box vs Open-box CTFs

- Hack@DAC is “Open-box”
  - Participants given a buggy SoC RTL
  - Finer grained scope
- Participants attempt to break security features
  - RTL Simulation/ Emulation
  - Formal Verification
  - RTL Static Analysis
  - Manual reviews
- **<u>Designer-centric approach</u>**




# Overview

- Introduction
- Value of Organizing HW CTFs
- How Hack@DAC is Unique
- Organizing Hack@DAC
- Key Takeaways & Summary

# Hack@DAC – The Process

## Participants

5 Registration

6 Bug submission

8 Scoreboard

11 Opportunities

## Design Team

1 Open-source design

Security features
Threat model

2

Bug list

3

Buggy design (RTL)

4

Updated spec

Tool support

## Judges

7 Bug evaluation

## Cloud Team

FPGA

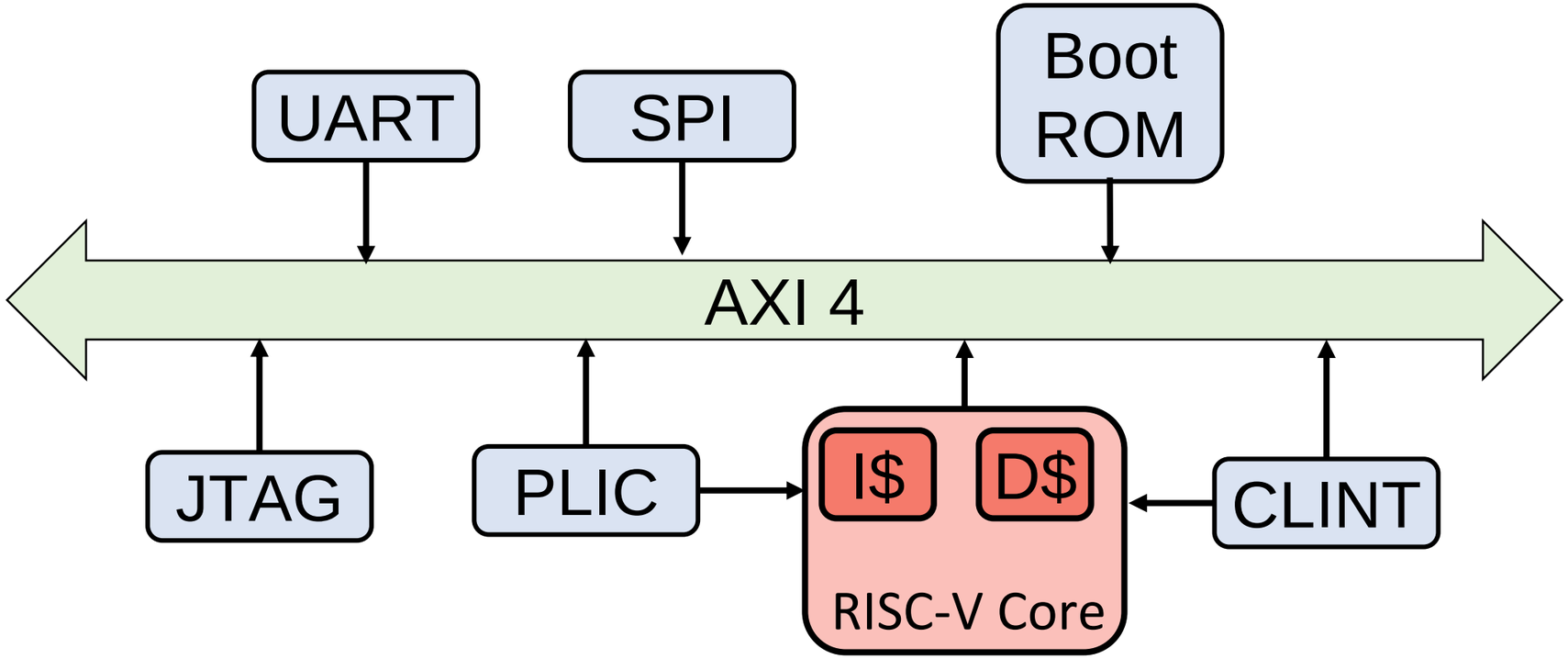
9 FPGA support for emulation

Commercial tools on cloud

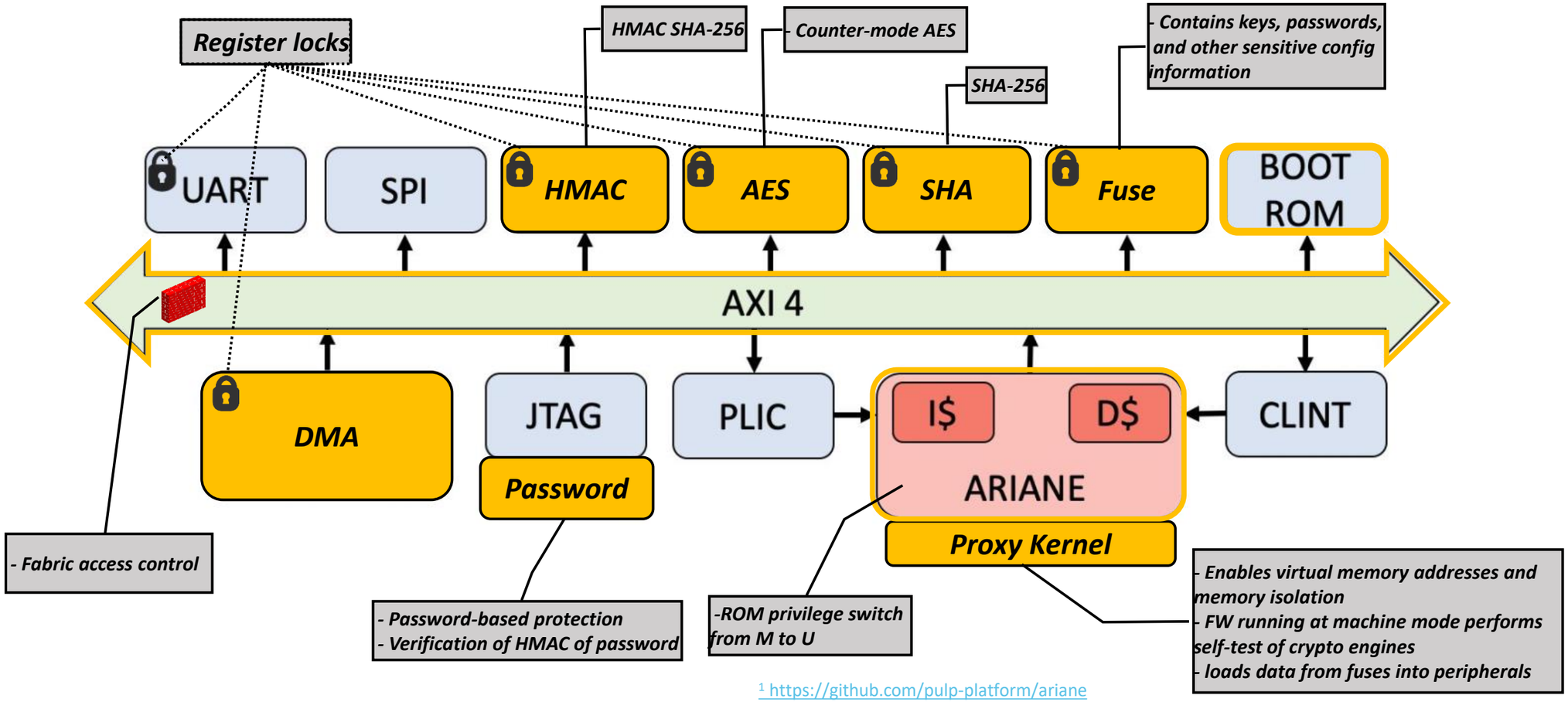
10 Winners announced

# Selection of Target

- Survey various open-source hardware designs and pick full SoC
- Priority given to designs with support for hardware simulation (open-source tool support), stability
- Reduced Instruction Set Computing (RISCV) RISC-V architecture based SoCs
  - Pulpino -> Pulpissimo -> OpenPiton -> Open Titan



# Adding Security Features to HW




[1] https://github.com/pulp-platform/ariane
[2] https://content.riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf

# Threat Modeling & Security Objectives

- Threat Model

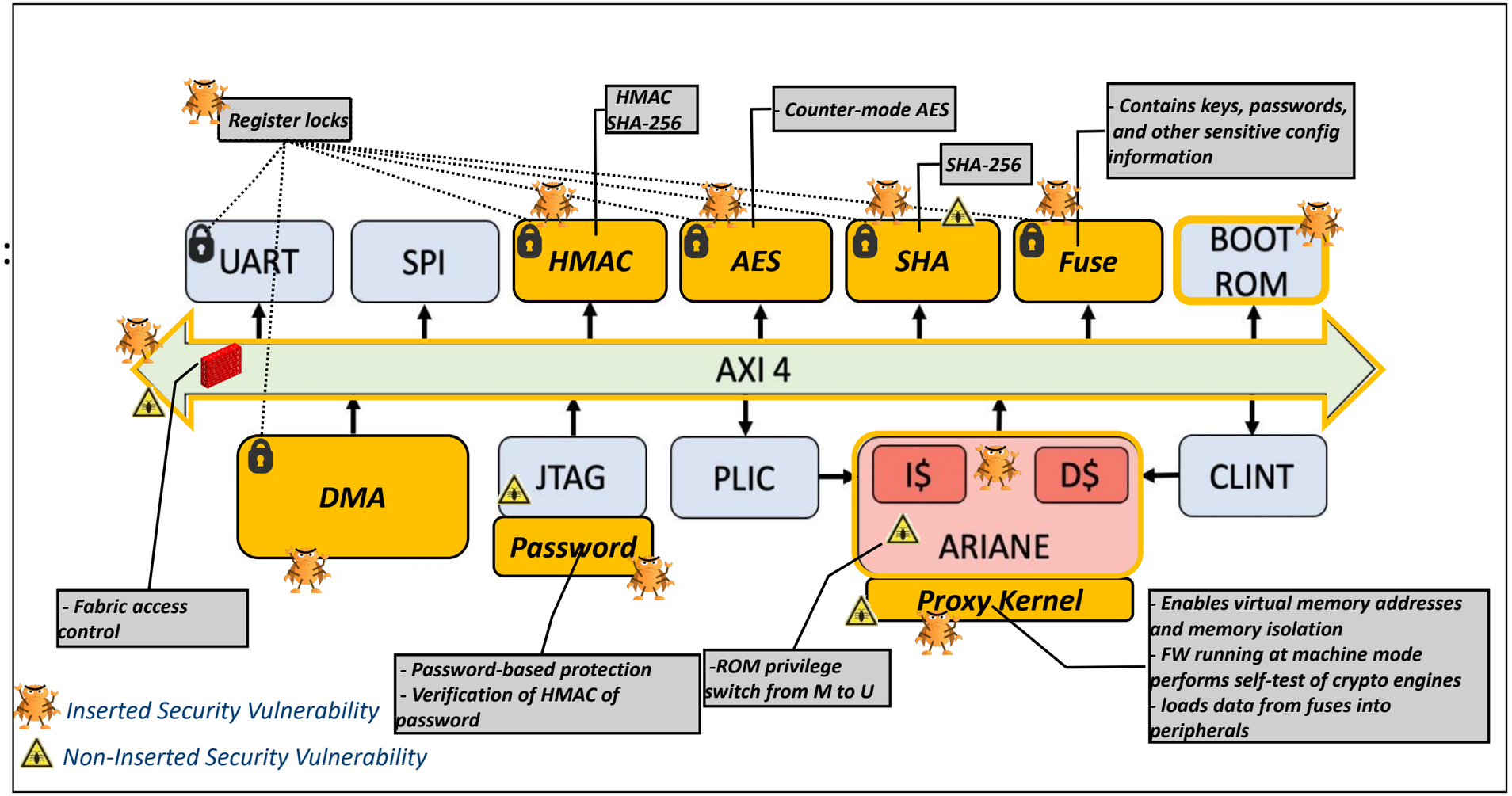
| # | Type | Description |
|---|---|---|
| 1 | Unprivileged software at user-level mode | Executes on core with user-level privileges but may exploit bugs to mount privilege escalation attacks |
| 2 | Physical attacker | Has physical possession of the device |
| 3 | Authorized debug access | Has the ability to unlock and debug production device |

- Security Objectives
  - Unprivileged code in core should not be able to compromise privilege level
  - Internal registers of crypto blocks should not be accessible from JTAG

# Inserting Vulnerabilities

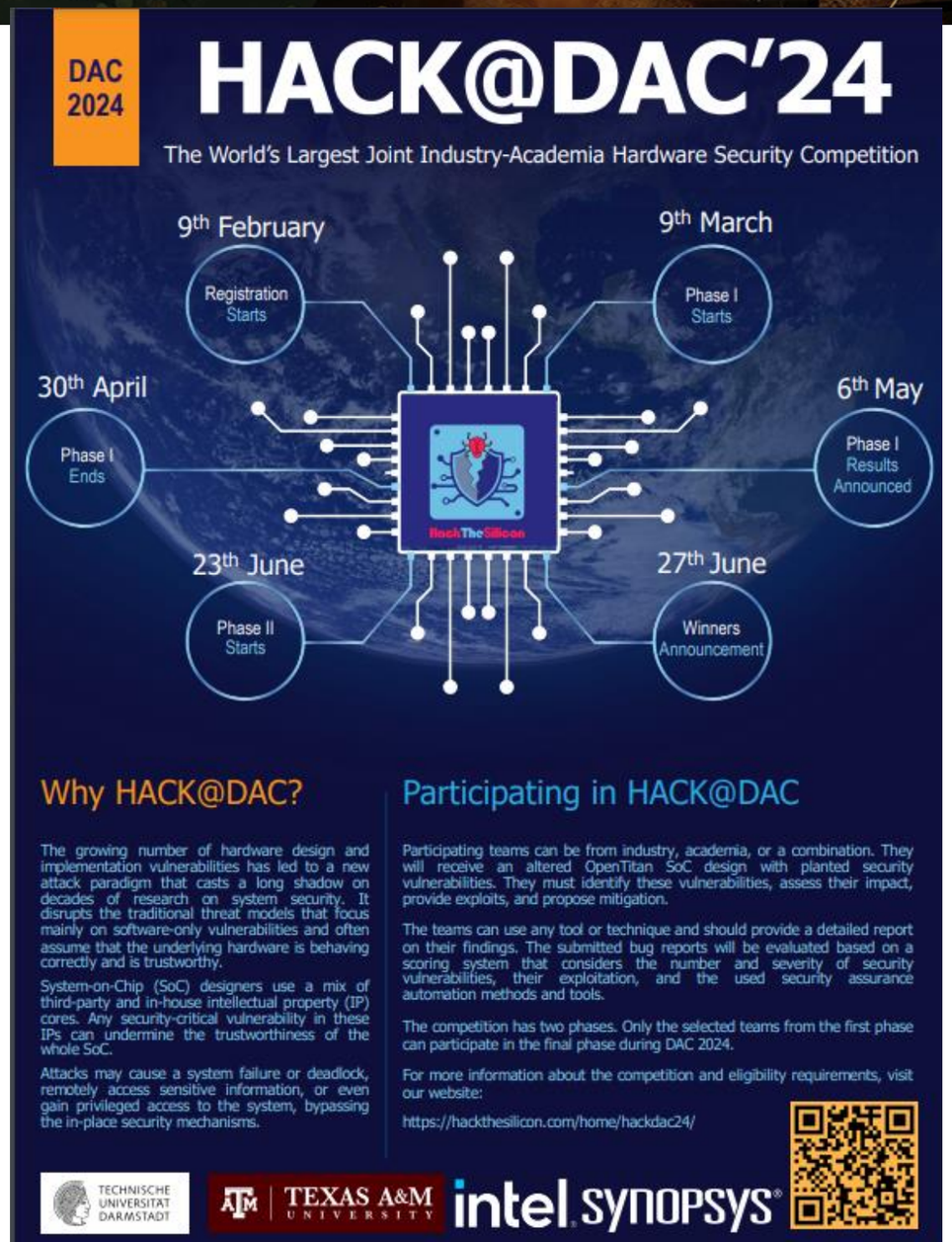
Vulnerabilities inspired by:

- CVEs
- Security advisories
- Our experience



# Advertisement

- Website updated with Call for Participation
- Advertised on social media

# Competition: Phase 1

- Phase 1 is offline
- Participants have over 2 months to:
  - Analyze entry points
  - Identify assets
  - Develop security test cases
  - Develop custom tools to detect bugs
  - Submit bugs for evaluation by judges
- Extended duration allows for equal access to participants from various backgrounds.

# Submission and Scoring

| B | D | E | F |
|---|---|---|---|
| Team name | Security feature bypassed | Finding | Location or code reference |
| | | | |
| | Register Lock Control signal unset | In access control register wrapper file, reglk_ctrl signal is responsible for reading/writing the signal for locked peripherals. All bits set to '1' of reglk_ctrl signal indicates the peripheral is locked otherwise bits set to '0' indicate normal operation. Therefore, by default reglk_ctrl should always be set high to prevent unauthorized access. We found that only lower half of the reglk_ctrl is set from 8-bit input reglk_ctrl_i and higher bits are set to 0. Thus, all bits from 8-15 are set to 0 and should not be accessed for any read/write operation. In acc_wrapper.sv, at line 96, 98 and | piton/design/chip/tile/ariane/src/acct/acct_wrapper.sv, Line 96, 98 and 100. |

Specific security feature that participants managed to bypass

# Submission and Scoring

| B | D | E | F | G |
|---|---|---|---|---|
| Team name | Security feature bypassed | Finding | Location or code reference | Detection method |
| | | | | |
| | Register Lock Control signal unset | In access control register wrapper file, reglk_ctrl signal is responsible for reading/writing the signal for locked peripherals. All bits set to '1' of reglk_ctrl signal indicates the peripheral is locked otherwise bits set to '0' indicate normal operation. Therefore, by default reglk_ctrl should always be set high to prevent unauthorized access. We found that only lower half of the reglk_ctrl is set from 8-bit input reglk_ctrl_i and higher bits are set to 0. Thus, all bits from 8-15 are set to 0 and should not be accessed for any read/write operation. In acc_wrapper.sv, at line 96, 98 and | piton/design/chip/tile/ariane/src/acct/acct_wrapper.sv, Line 96, 98 and 100. | Manual analysis + User level assertion generation + Formal property verification using Synopsys VCStatic |

How was the vulnerability identified?

- Simulation
- Formal Verification?
- Custom tool?
- Manual code review?

# Submission and Scoring

| B | D | E | F | G | H |
|---|---|---|---|---|---|
| Team name | Security feature bypassed | Finding | Location or code reference | Detection method | Security impact |
| | | | | | other wrappers, all the secure data can be read out. |
| | Register Lock Control signal unset | In access control register wrapper file, reglk_ctrl signal is responsible for reading/writing the signal for locked peripherals. All bits set to '1' of reglk_ctrl signal indicates the peripheral is locked otherwise bits set to '0' indicate normal operation. Therefore, by default reglk_ctrl should always be set high to prevent unauthorized access. We found that only lower half of the reglk_ctrl is set from 8-bit input reglk_ctrl_i and higher bits are set to 0. Thus, all bits from 8-15 are set to 0 and should not be accessed for any read/write operation. In acc_wrapper.sv, at line 96, 98 and | piton/design/chip/tile/ariane/src/acct/acct_wrapper.sv, Line 96, 98 and 100. | Manual analysis + User level assertion generation + Formal property verification using Synopsys VCStatic | This bug will lead to accessing peripheral device even when its register is in locked state (which ideally should have restricted its access). |

What is the security impact of bypassing security feature?

# Submission and Scoring

| B | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|
| Team name | Security feature bypassed | Finding | Location or code reference | Detection method | Security impact | Adversary profile | Proposed mitigation |
| | | | | | other wrappers, all the secure data can be read out. | | |
| | Register Lock Control signal unset | In access control register wrapper file, reglk_ctrl signal is responsible for reading/writing the signal for locked peripherals. All bits set to '1' of reglk_ctrl signal indicates the peripheral is locked otherwise bits set to '0' indicate normal operation. Therefore, by default reglk_ctrl should always be set high to prevent unauthorized access. We found that only lower half of the reglk_ctrl is set from 8-bit input reglk_ctrl_i and higher bits are set to 0. Thus, all bits from 8-15 are set to 0 and should not be accessed for any read/write operation. In acc_wrapper.sv, at line 96, 98 and | piton/design/chip/tile/ariane/src/acct/acct_wrapper.sv, Line 96, 98 and 100. | Manual analysis + User level assertion generation + Formal property verification using Synopsys VCStatic | This bug will lead to accessing peripheral device even when its register is in locked state (which ideally should have restricted its access). | Unprivileged software at user-level mode | One line verilog change in acct_wrapper.sv: reglk_ctrl[13] -> reglk_ctrl[3] |

Mitigation suggestions

# Submission and Scoring

| B | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|
| Team name | Security feature bypassed | Finding | Location or code reference | Detection method | Security impact | Adversary profile | Proposed mitigation | CVSSv3.1 score and severity | CVSSv3.1 Details |
| | | | | | other wrappers, all the secure data can be read out. | | | | |
| | Register Lock Control signal unset | In access control register wrapper file, reglk_ctrl signal is responsible for reading/writing the signal for locked peripherals. All bits set to '1' of reglk_ctrl signal indicates the peripheral is locked otherwise bits set to '0' indicate normal operation. Therefore, by default reglk_ctrl should always be set high to prevent unauthorized access. We found that only lower half of the reglk_ctrl is set from 8-bit input reglk_ctrl_i and higher bits are set to 0. Thus, all bits from 8-15 are set to 0 and should not be accessed for any read/write operation. In acc_wrapper.sv, at line 96, 98 and | piton/design/chip/tile/ariane/src/acct/acct_wrapper.sv, Line 96, 98 and 100. | Manual analysis + User level assertion generation + Formal property verification using Synopsys VCStatic | This bug will lead to accessing peripheral device even when its register is in locked state (which ideally should have restricted its access). | Unprivileged software at user-level mode | One line verilog change in acct_wrapper.sv: reglk_ctrl[13] -> reglk_ctrl[3] | Medium (6.1) | CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:H/A:N/RC:C<br>Attack vector: Local. A person having read/write/execute access on the SoC can mount the attack.<br>Attack complexity: Low. An exploit code developed can sureshot obtain access control of |

CVSS scoring details to determine severity of issue

# Submission and Scoring

| B | D | E | F | G | H | I | J | K | L | N |
|---|---|---|---|---|---|---|---|---|---|---|
| Team name | Security feature bypassed | Finding | Location or code reference | Detection method | Security impact | Adversary profile | Proposed mitigation | CVSSv3.1 score and severity | CVSSv3.1 Details | Judges comments |
| | | | | | other wrappers, all the secure data can be read out. | | | | | |
| | Register Lock Control signal unset | In access control register wrapper file, reglk_ctrl signal is responsible for reading/writing the signal for locked peripherals. All bits set to '1' of reglk_ctrl signal indicates the peripheral is locked otherwise bits set to '0' indicate normal operation. Therefore, by default reglk_ctrl should always be set high to prevent unauthorized access. We found that only lower half of the reglk_ctrl is set from 8-bit input reglk_ctrl_i and higher bits are set to 0. Thus, all bits from 8-15 are set to 0 and should not be accessed for any read/write operation. In acc_wrapper.sv, at line 96, 98 and | piton/design/chip/tile/ariane/src/acct/acct_wrapper.sv, Line 96, 98 and 100. | Manual analysis + User level assertion generation + Formal property verification using Synopsys VCStatic | This bug will lead to accessing peripheral device even when its register is in locked state (which ideally should have restricted its access). | Unprivileged software at user-level mode | One line verilog change in acct_wrapper.sv: reglk_ctrl[13] -> reglk_ctrl[3] | Medium (6.1) | CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:H/A:N/RC:C Attack vector: Local. A person having read/write/execute access on the SoC can mount the attack. Attack complexity: Low. An exploit code developed can sureshot obtain access control of | Valid issue (5) + correc impact analysis (5) + FPV usage to detect bug (50) |

Scoring based on:

- Validity of issue
- Novelty of methodology used
- Correctness of security impact, mitigation, CVSS
- Conference theme based bonus
  - New tool bonus at DAC
  - Exploit bonus at USENIX Security

Special award for “cool” finds!

Manual vs Automated scoring

# Competition: Phase 2 (Finals)

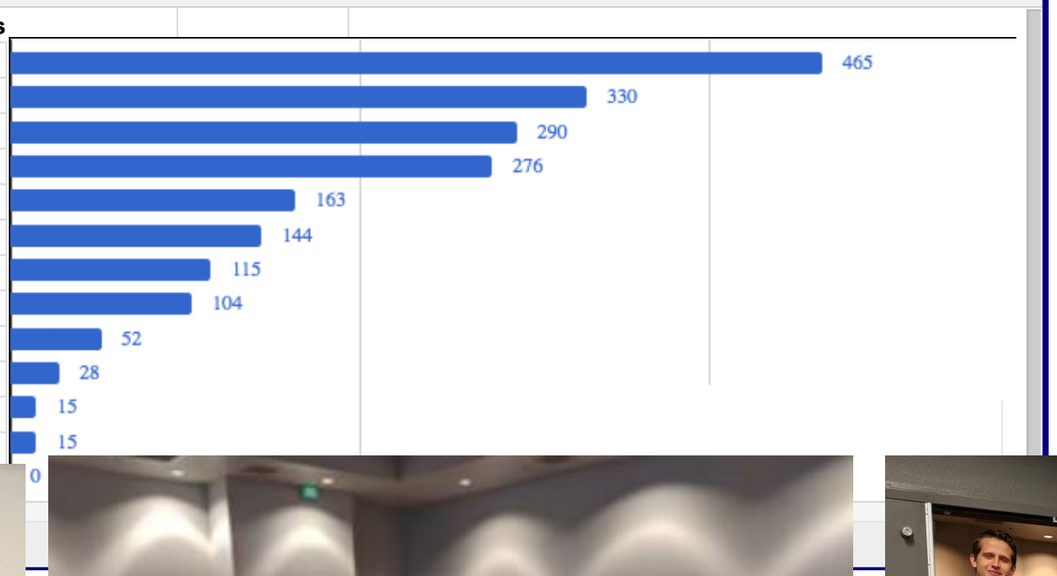
- Top 10 teams invited to participate in finals
- Phase 2 live at the conference
- Partnership with Synopsys
  - All necessary tools hosted on Synopsys cloud
  - Buggy design ported to cloud
  - Tool trainings provided to all finalists
- Travel grants to US-based finalists to attend in person
- Duration of 48 hours

# Competition: Phase 2 (Finals)

## Live Scoreboard

Hack@DAC'19 Beta Scoreboard : Live

| Team name | Points |
|---|---|
| Hackin' Aggies* | 465 |
| NOPS | 330 |
| Always@Posedge | 290 |
| NotATrojan | 276 |
| Alpha4 | 163 |
| .:hackamole:. | 144 |
| SEC | 115 |
| Team 11 | 104 |
| Chipsters | 52 |
| Tribe | 28 |
| CCNY | 15 |
| CICA* | 15 |



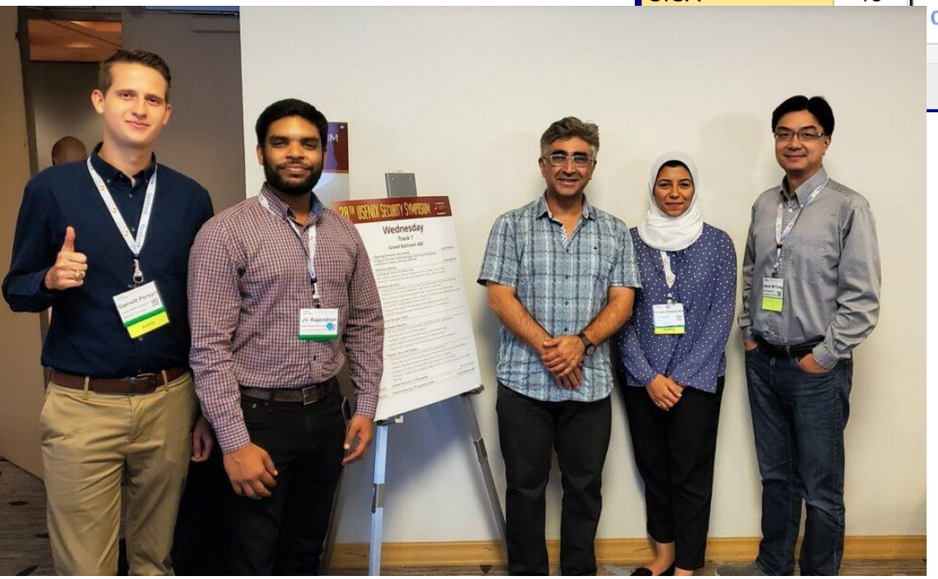




Image: “Hacking SoC IP Under Pressure”, SemiEngineering 2018 source

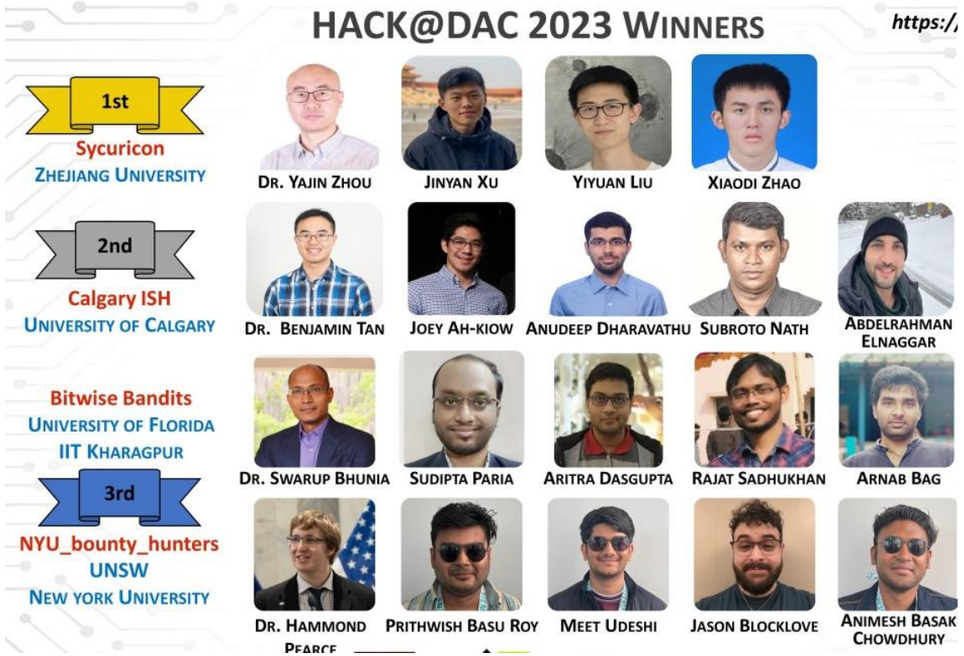
# Competition: Phase 2 (Finals)

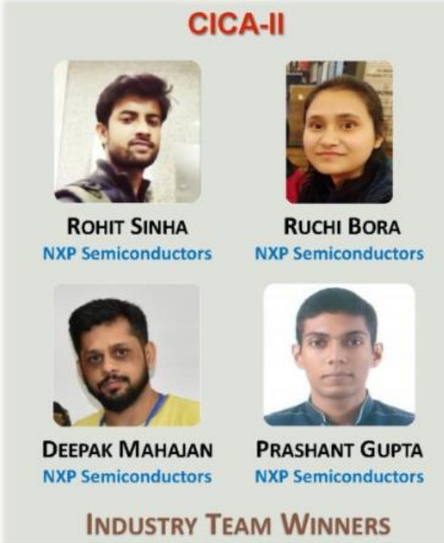
## Winners Honored


## Publications

# So Far..

- Extended to USENIX Security (Hack@SEC) and CHES (Hack@CHES)
- 300+ teams participated from all over the world; 1000+ participants
- Industry participation too!
- Past winners now working in hardware security roles at top companies




# Overview

- Introduction
- Value of Organizing HW CTFs
- How Hack@DAC is Unique
- Organizing Hack@DAC
- Key Takeaways & Summary

# Recap of 3 Top Challenges

Awareness of Hardware Common Weaknesses

Security-Aware Design Automation

“Shift-Left” to Detect & Fix Bugs in RTL

# MITRE Hardware CWE

https://cwe.mitre.org

**1194 - Hardware Design**

- C Manufacturing and Life Cycle Management Concerns - *(1195)*
- C Security Flow Issues - *(1196)*
- C Integration Issues - *(1197)*
- C Privilege Separation and Access Control Issues - *(1198)*
- C General Circuit and Logic Design Concerns - *(1199)*
- C Core and Compute Issues - *(1201)*
- C Memory and Storage Issues - *(1202)*
- C Peripherals, On-chip Fabric, and Interface/IO Problems - *(1203)*
- C Security Primitives and Cryptography Issues - *(1205)*
- C Power, Clock, Thermal, and Reset Concerns - *(1206)*
- C Debug and Test Problems - *(1207)*
- C Cross-Cutting Problems - *(1208)*
- C Physical Access Issues and Concerns - *(1388)*

- 75+/110 CWE entries contributed by Intel
- Hack@DAC vulnerability and mitigation examples now added to several CWE entries
- "Hardware Security Failure Scenarios"

NIST

## CWE-1245: Improper Finite State Machines (FSMs) in Hardware Logic

**Weakness ID: 1245**
**Vulnerability Mapping: ALLOWED**
**Abstraction:** Base

*View customized information:* Conceptual | Operational | Mapping Friendly | **Complete** | Custom

### Description

Faulty finite state machines (FSMs) in the hardware logic allow an attacker to put the system in an undefined state, to cause a denial of service (DoS) or gain privileges on the victim's system.

### Extended Description

The functionality and security of the system heavily depend on the implementation of FSMs. FSMs can be used to indicate the current security state of the system. Lots of secure data operations and data transfers rely on the state reported by the FSM. Faulty FSM designs that do not account for all states, either through undefined states (left as don't cares) or through incorrect implementation, might lead an attacker to drive the system into an unstable state from which the system cannot recover without a reset, thus causing a DoS. Depending on what the FSM is used for, an attacker might also gain additional privileges to launch further attacks and compromise the security guarantees.

### Relationships

***Relevant to the view "Research Concepts" (CWE-1000)***

| Nature | Type | ID | Name |
|---|---|---|---|
| ChildOf | C | 684 | Incorrect Provision of Specified Functionality |

***Relevant to the view "Hardware Design" (CWE-1194)***

| Nature | Type | ID | Name |
|---|---|---|---|
| MemberOf | C | 1199 | General Circuit and Logic Design Concerns |

### Modes Of Introduction

| Phase | Note |
|---|---|
| Architecture and Design | |
| Implementation | |

### Applicable Platforms

**Languages**

Class: Not Language-Specific *(Undetermined Prevalence)*

**Operating Systems**

# Security-Aware Tooling & Bug Detection

- Security Test Case Generation and Bug Patching using GenAI/ LLMs
  - (Security) Assertions by Large Language Models *(IEEE TIFS 2024)*
  - Examining Zero Shot Vulnerability Repair with Large Language Models *(IEEE Security and Privacy 2*
  - Fixing Hardware Security Bugs with Large Language Models *(arXiv)*
  - On Prompting Hardware Security Bug Code Fixes by Prompting Large Language Models *(IEEE TIFS*
  - DIVAS: An LLM-based End to End Framework for SoC Security Analysis and Policy-based Protection *(arXiv)*
- Formal Verification
  - Sylvia: Countering the Path Explosion Problem in the Symbolic Execution of Hardware Designs *(FMCA*
  - All Artificial, Less Intelligence: GenAI Through the Lens of Formal Verification *(arXiv)*
- Static Analysis
  - Don’t CWEAT It: Toward CWE Analysis Techniques in Early Stages of Hardware Design *(IEEE/ACM IC*
- Concolic Testing
  - RTL-ConTest: Concolic Testing on RTL for Detecting Security Vulnerabilities *(IEEE TCAD 2022)*
- Hardware Information Flow Tracking
  - Cell-IFT: Leveraging Cells for Scalable & Precise Dynamic Information Flow Tracking in RTL *(USENIX Security 2022)*

# Key Takeaways for Academia

- Hack@DAC SoC framework
  - Realistic threat model and security objectives
  - Closest available to commercial chip designs
  - Uncover new classes of security vulnerabilities
- Get invaluable hardware security assurance skills!
  - Mimic security teams at a chip design company
  - Develop a hacker mindset
- Competition format
  - provides equal access to participants from diverse backgrounds
    - Strong technical female participation
  - Facilitates participation from various geos/ time zones

Hack@DAC 2018 finals
at San Francisco, CA

Image: “Hacking SoC IP Under Pressure”, SemiEngineering 2018 source

# Takeaways for Industry

- Improve in-house security assurance best practices
  - Exposure to new kinds of weaknesses
  - Planning for survivability features
  - Easier for functional verification teams to pick up security assurance
- New tools for identifying weakness classes
  - Publish guides on detection of classes of hardware security weaknesses
- Add security capabilities to today's functional tools
  - Address gaps of today's security verification tools to detect classes of vulnerabilities

# Media Coverage

EE Times

Capture-the-Flag Competitions Need to Include Hardware

CYBER DEFENSE MAGAZINE

Learning Hardware Security Via Capture-The-Flag Competitions

DEVOPS digest

Why Do We Need a Standardized Framework to Enumerate Hardware Security Weaknesses?

techspective ...a unique perspective on technology ©

Intel Hardware CTF Competitions Drive Innovation for Next-Gen Secure Computing Platforms

SEMICONDUCTOR ENGINEERING DEEP INSIGHTS FOR THE TECH INDUSTRY

Hacking SoC IP Under Pressure

DARKREADING

Intel Harnesses Hackathons to Tackle Hardware Vulnerabilities

Hack@DAC has resulted in:

- Increased HW Security Awareness
  - *MITRE HW CWE* (*https://cwe.mitre.org*)
  - *Corpus of weaknesses and code examples*
- Availability of Open-sourced buggy SoCs
  - *Realistic security features*
  - *CVE-inspired vulnerabilities*
  - *Complexity matching commercial chips*
- Innovations in HW security tooling
  - *Tools that detect and patch bugs at RTL*
- Participants developed hacker mindset

**Contact**

**Website**: https://hackthesilicon.com/

**Email**: hackatevent@gmail.com

black hat®
USA 2024

# HW Vulnerability Example - Locks

Lock signal for sensitive registers

Security sensitive register core_lock_reg is not locked

Attacker can overwrite security sensitive register

```
45      input logic [16-1:0][53:0]      pmpaddr_i;
46      input logic [7 :0]              reglk_ctrl_i; // register lock values
129             else if(en && we)
130                 case(address[7:3])
131                     0:
132                         start_reg <= wdata;
145                     7:
146                         core_lock_reg <= (wdata==0) ? 0 : ((core_lock_reg==0) ? wdata : 0);
```

Security Impact

https://github.com/PrincetonUniversity/openpiton

# HW Vulnerability Example - Debug

State machine implementing password authentication logic for secure debug access

When opcode is DTM_PASS (for entering password), state change changes to WRITE

Attacker wants to enter password – but gets write access to chip internals through debug interface

Security Impact

```
case (state_q)
    Idle: begin
        // make sure that no error is sticky
        if (dmi_access && update_dr && (error_q == DMINoError)) begin
            // save address and value
            address_d = dmi.address;
            data_d = dmi.data;
            if ( (dm::dtm_op_e'(dmi.op) == dm::DTM_READ) && (pass_check | ~we_flag == 1) ) begin
                state_d = Read;
            end else if ( (dm::dtm_op_e'(dmi.op) == dm::DTM_WRITE) && (pass_check == 1) ) begin
                state_d = Write;
            end else if (dm::dtm_op_e'(dmi.op) == dm::DTM_PASS) begin
                state_d = Write;
                pass_mode = 1'b1;
            end
            // else this is a nop and we can stay here
        end
    end
```

When in debug mode, return 0 when keys are read

debug_mode is not checked for key_big2

Attacker can extract key_big2 during debug

Security Impact

```
assign p_c_big    = {p_c[0], p_c[1], p_c[2], p_c[3]};
assign state_big  = {state[0], state[1], state[2], state[3]};
assign key_big0    = debug_mode_i ? 192'b0 : {key0[0], key0[1], key0[2], key0[3], key0[4], key0[5]};
assign key_big1    = debug_mode_i ? 192'b0 : {key1[0], key1[1], key1[2], key1[3], key1[4], key1[5]};
assign key_big2    = {key2[0], key2[1], key2[2], key2[3], key2[4], key2[5]};
```

https://github.com/PrincetonUniversity/openpiton