SoC Security Verification

Mark Tehranipoor and Farimah Farahmandi

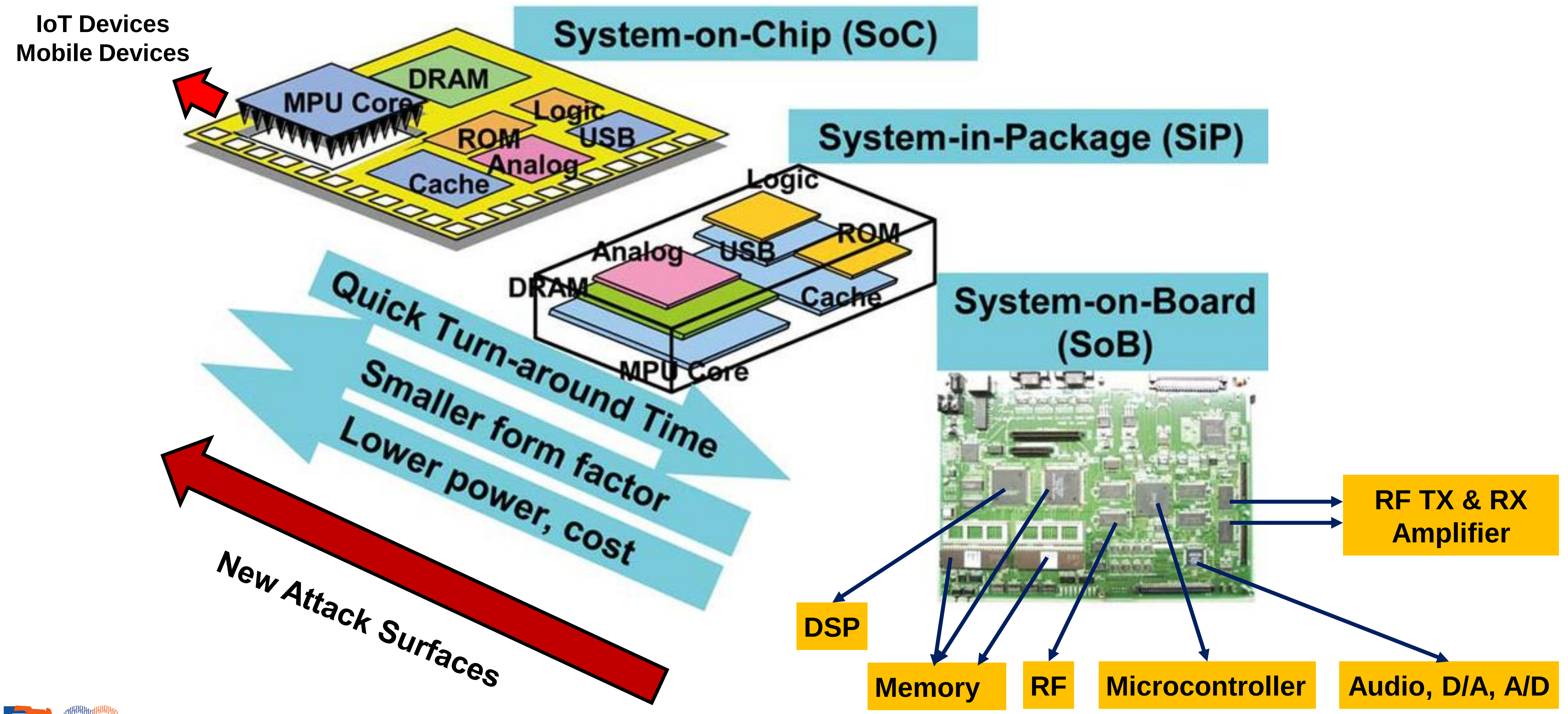
Florida Institute for Cybersecurity Research
University of Florida

Download the slides from: http://farimah.ece.ufl.edu/CADforSecurity

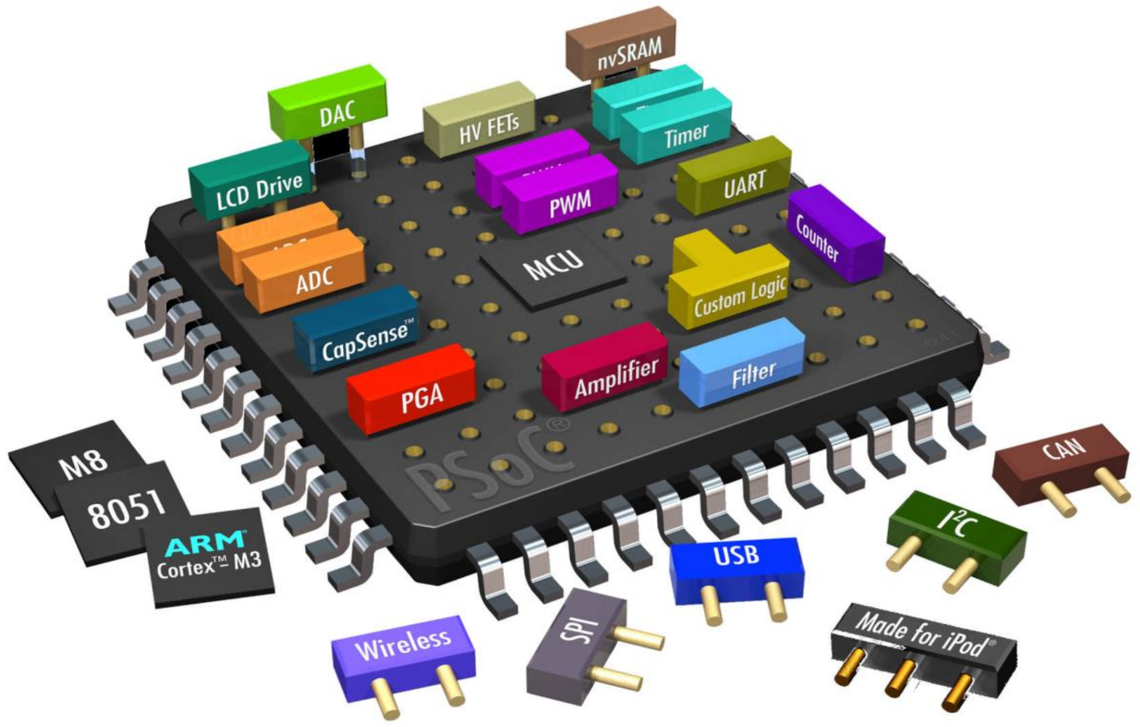
Outline

- Problem Statement
- Design Flow
- Supply Chain
- Hardware Attacks
- Need for Automation
- CAD for Security Demos
- Challenges

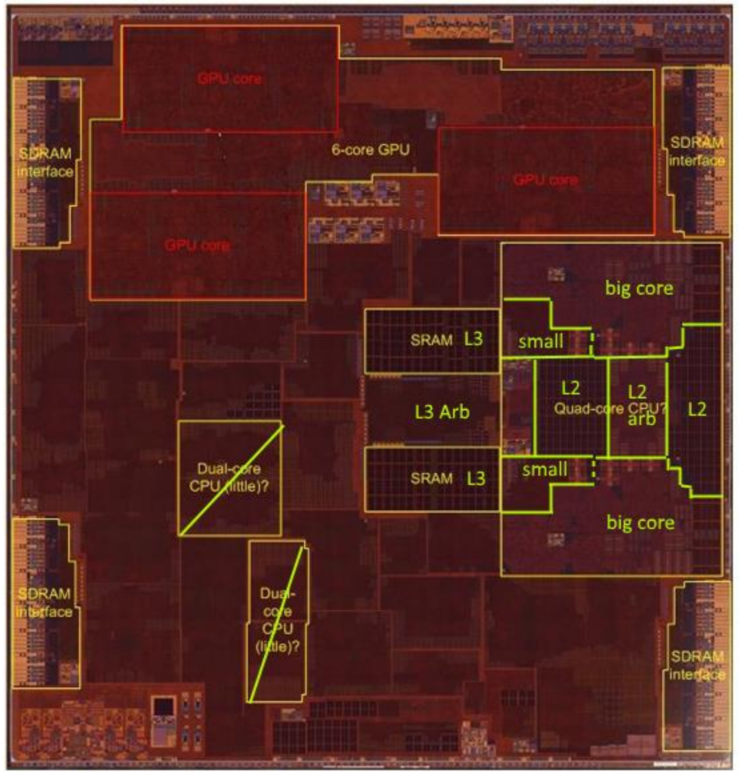
VLSI Integration



Modern SoCs – Heterogeneous Architecture



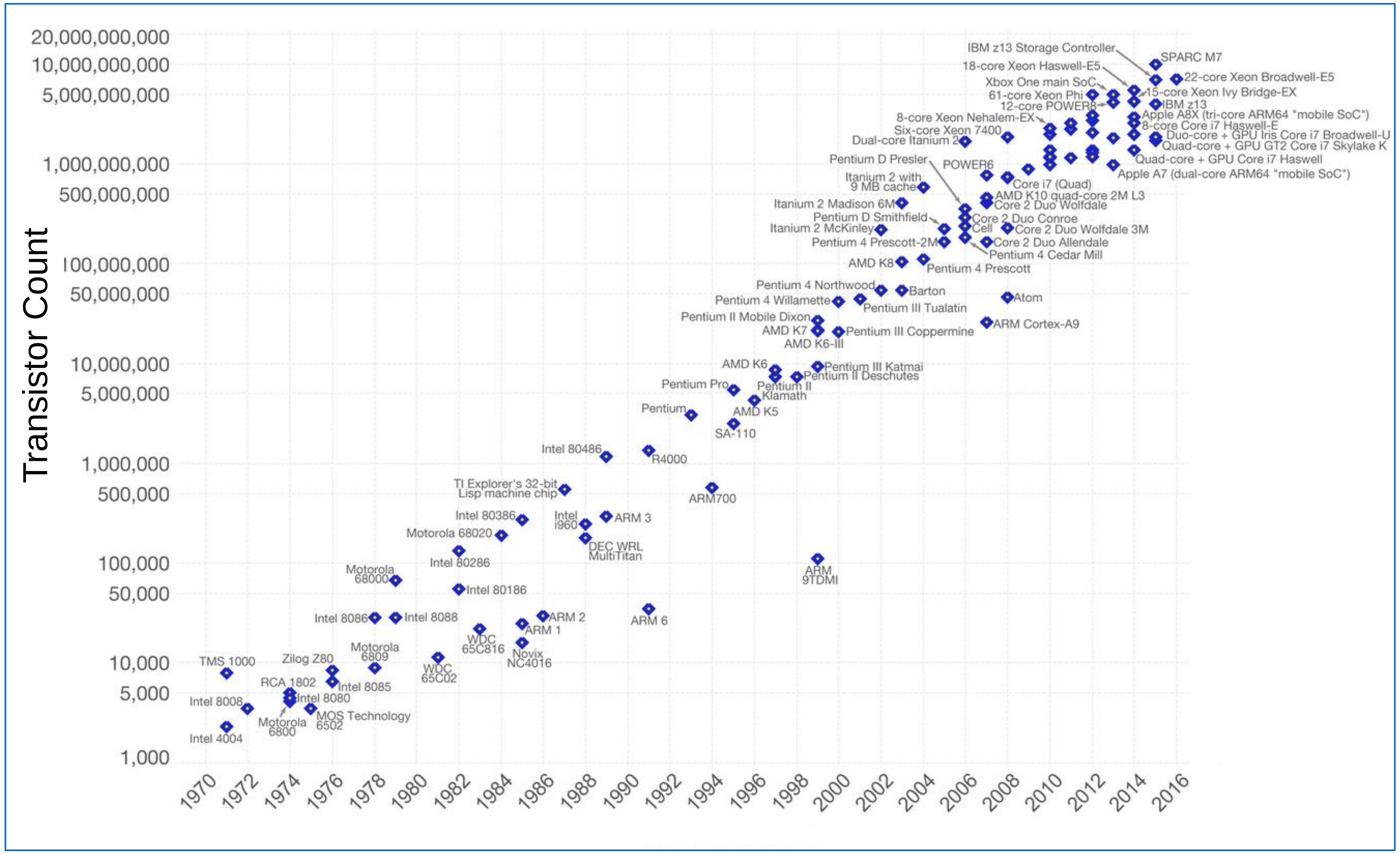
Apple A10 Quad Core SoC



- TSMC's **16 nm** FinFET
- **3.3 billion** transistors
- Die size: 125 mm^2

@Chipworks

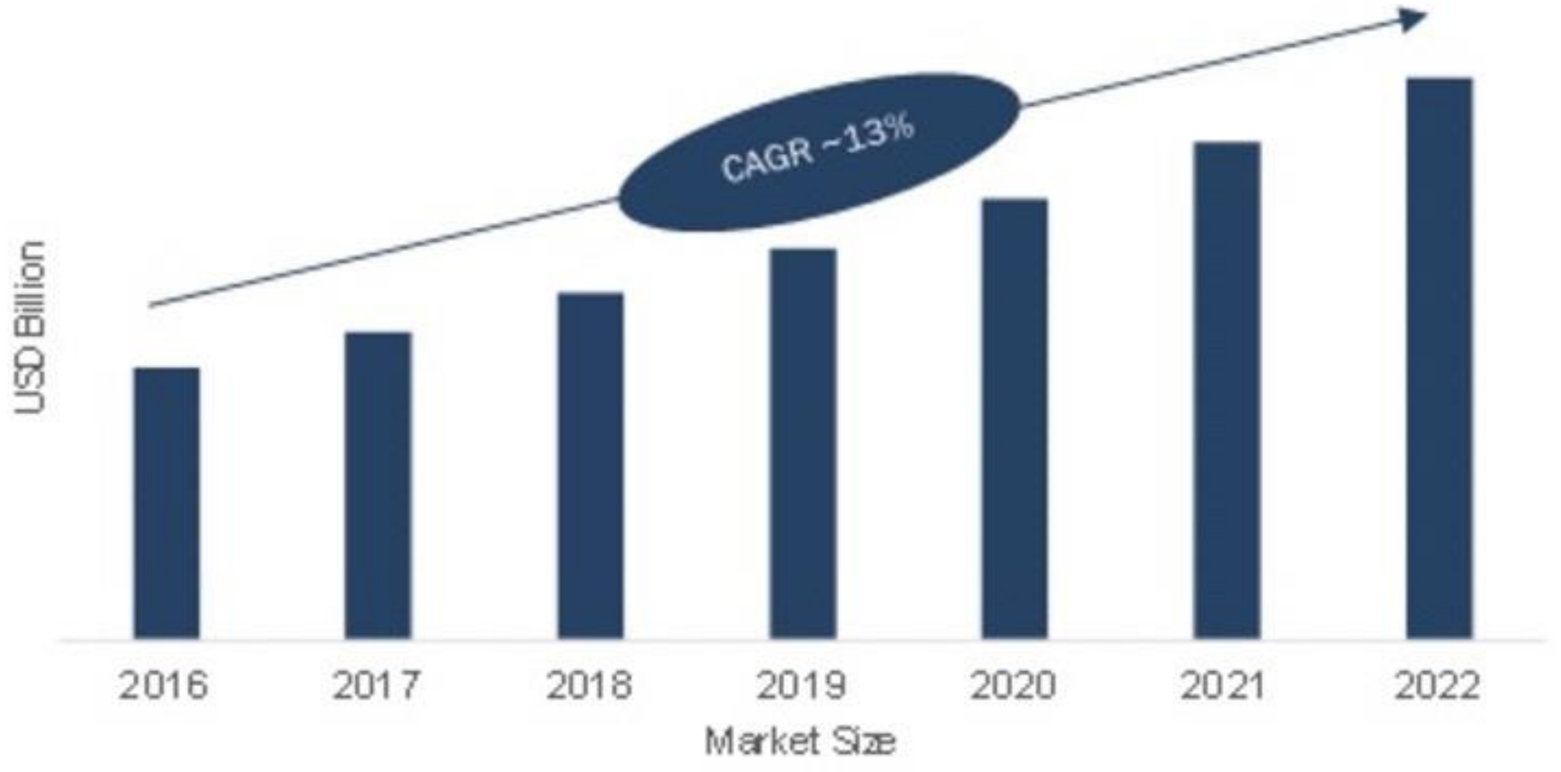
SoC’s Growth



Year of Introduction

SoC's Growth

SoC Market Size



Design Challenges

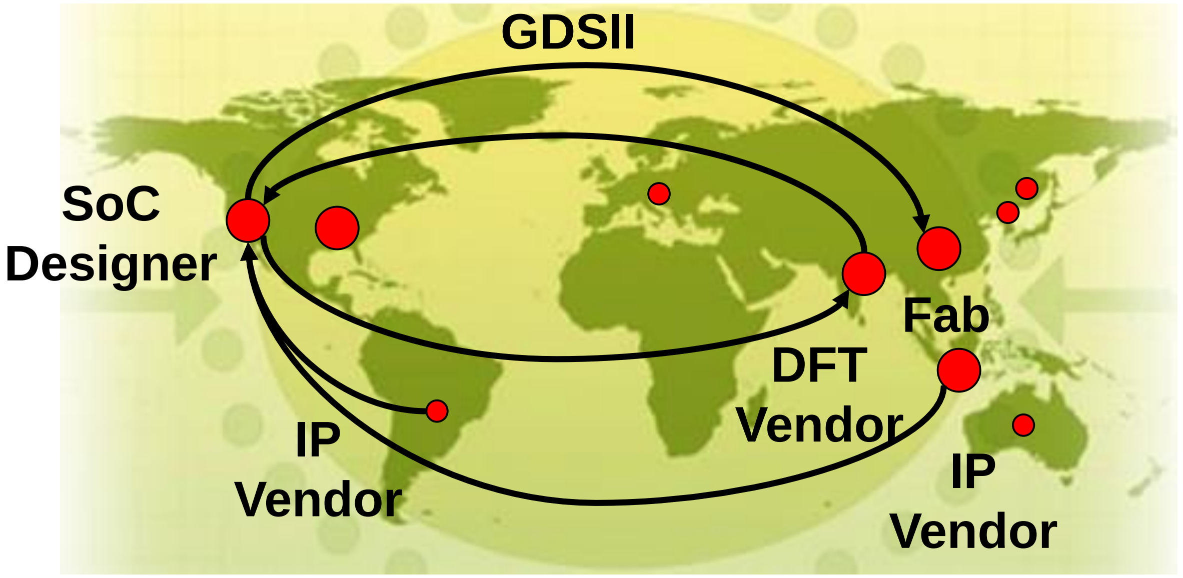
- **High complexity of devices**
  - Tens of billions transistors
- **Aggressive time-to-market requirements**
  - Severely constrains functional validation → vulnerability escapes to silicon or in-field
- **High diversity in computing devices**
  - Security requirements **vary** significantly
  - Cannot be “pre-verified” at the IP level
- **Connectivity**
  - More SoCs being connected → not originally designed to be connected

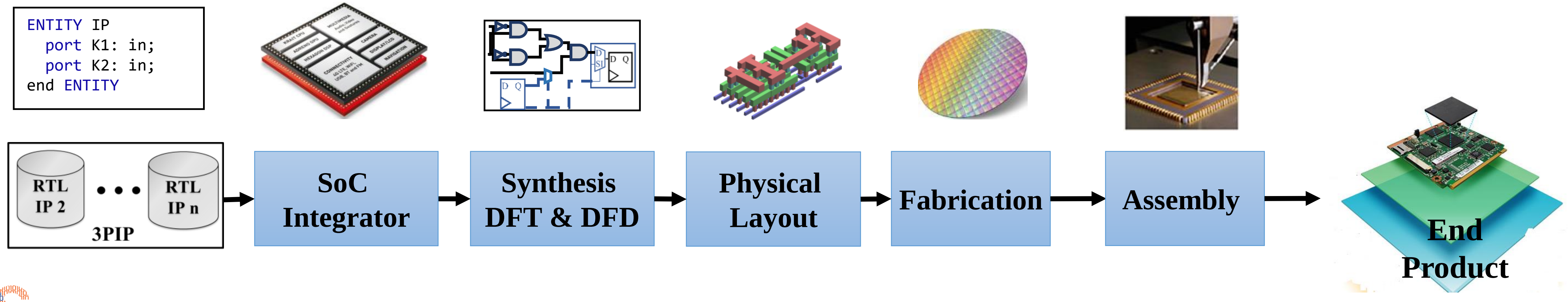
Contains 3.3 Billions of transistors

Shrunk to less than a year → mobile device

Everything is connect to Internet

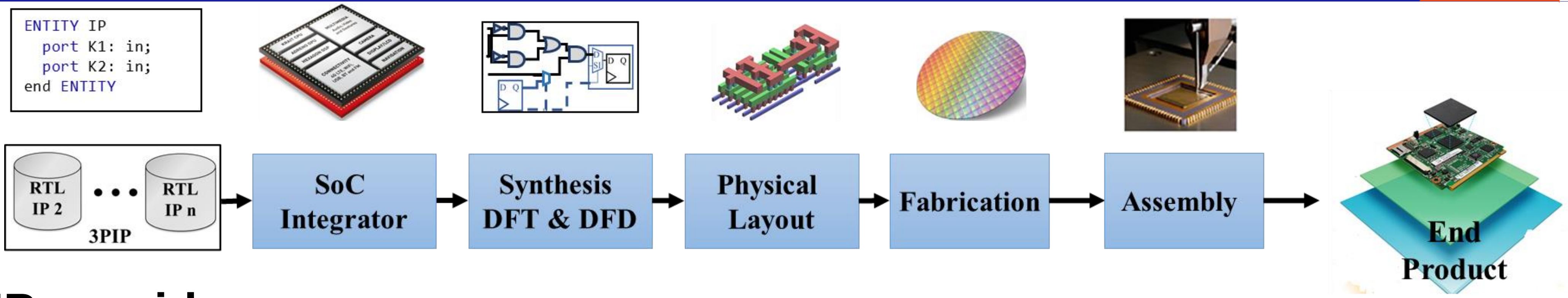
Design Flow





```
ENTITY IP
  port K1: in;
  port K2: in;
end ENTITY
```

Security & Trust Issues: Supply Chain



- **3PIP providers**
  - Working under aggressive schedules → **design mistakes, poor IP validation**
  - Can insert malicious implants (hardware **Trojans**)
- **CAD tools**
  - **Not equipped** with understanding security vulnerabilities
  - Vulnerabilities during **optimization, synthesis, DFT**, etc.
- **Foundry**
  - Access to the entire design → hardware Trojan, Counterfeit
  - **Counterfeits** → low-quality clones, overproduced chips in untrusted foundry

Challenges

Tens of billions transistors

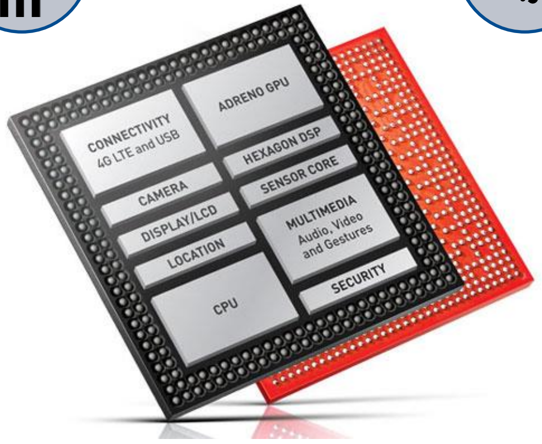
Many custom/legacy functionality

Many security critical assets

Designed around the globe

Tens of IPs from 3P vendors

Aggressive time-to-market



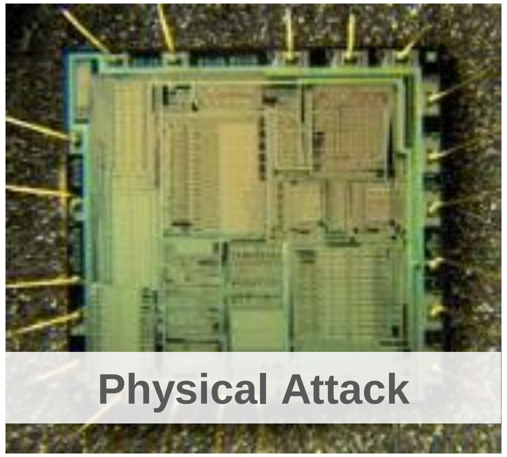
Ensuring security is a challenge

HW Attacks

Trojans

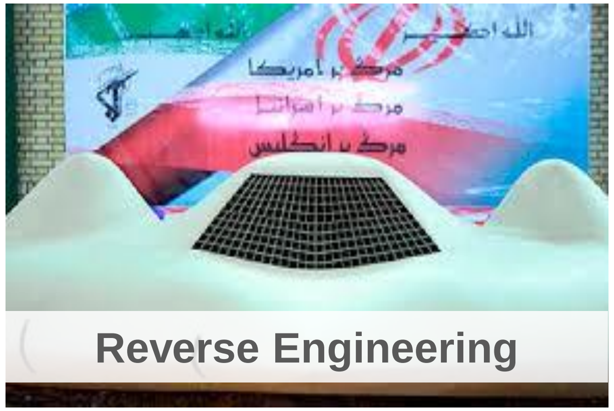
Untrusted Foundry

Counterfeit ICs


Physical Attack

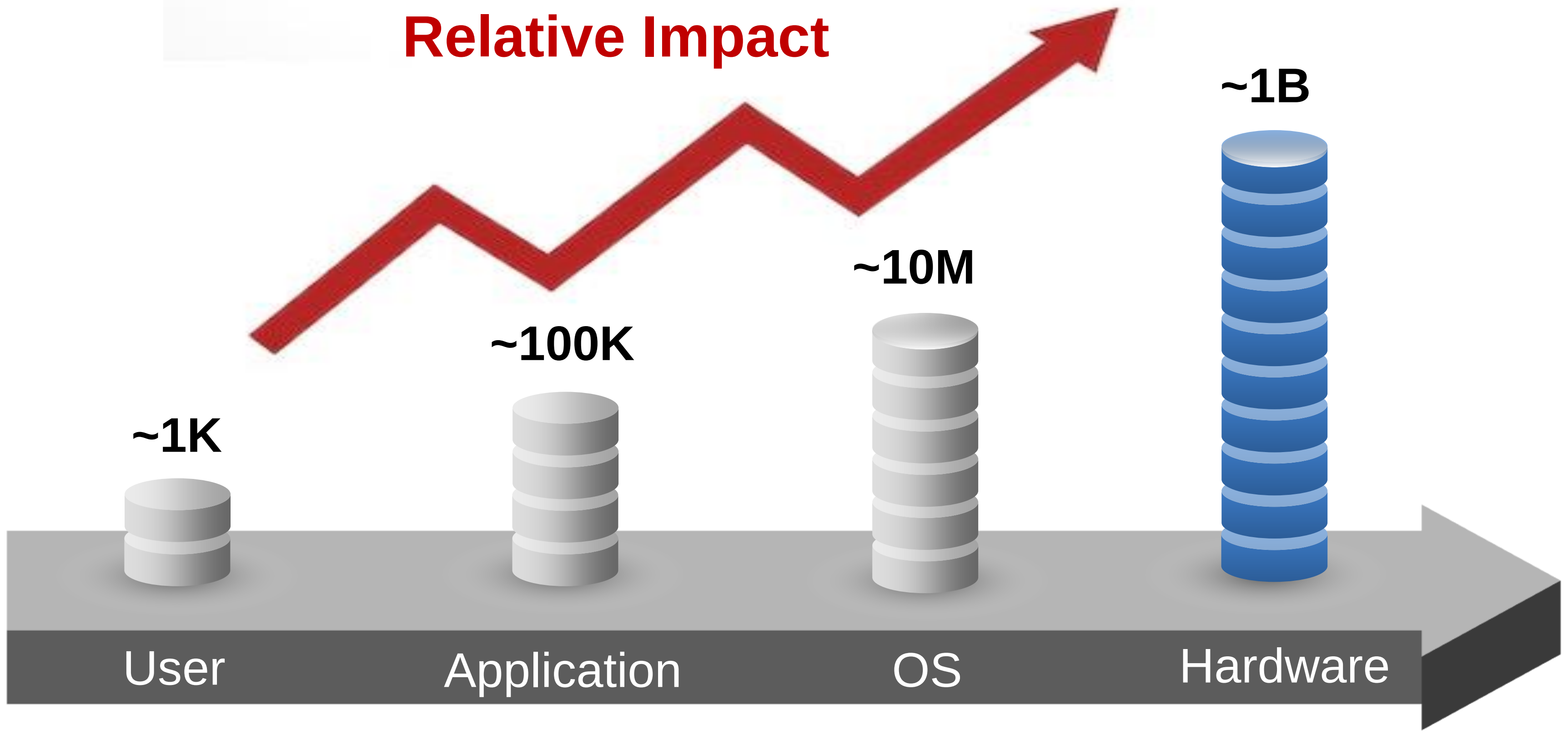
Side-channel Attacks

Fault Injection Attacks


Reverse Engineering

Counterfeit/Fake Parts

Impact: HW Security Compromise

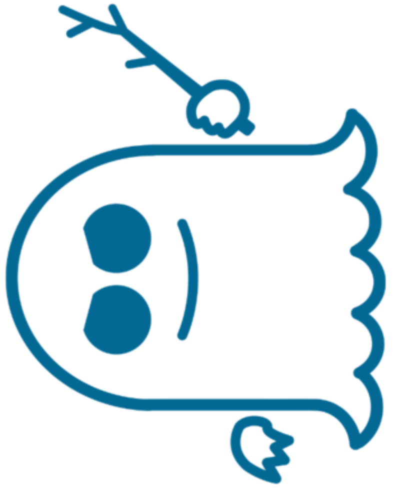


Social engineering (phishing)

Malwares (information harvesting)

Virus/ Trojan (Hijacking/ DDoS)

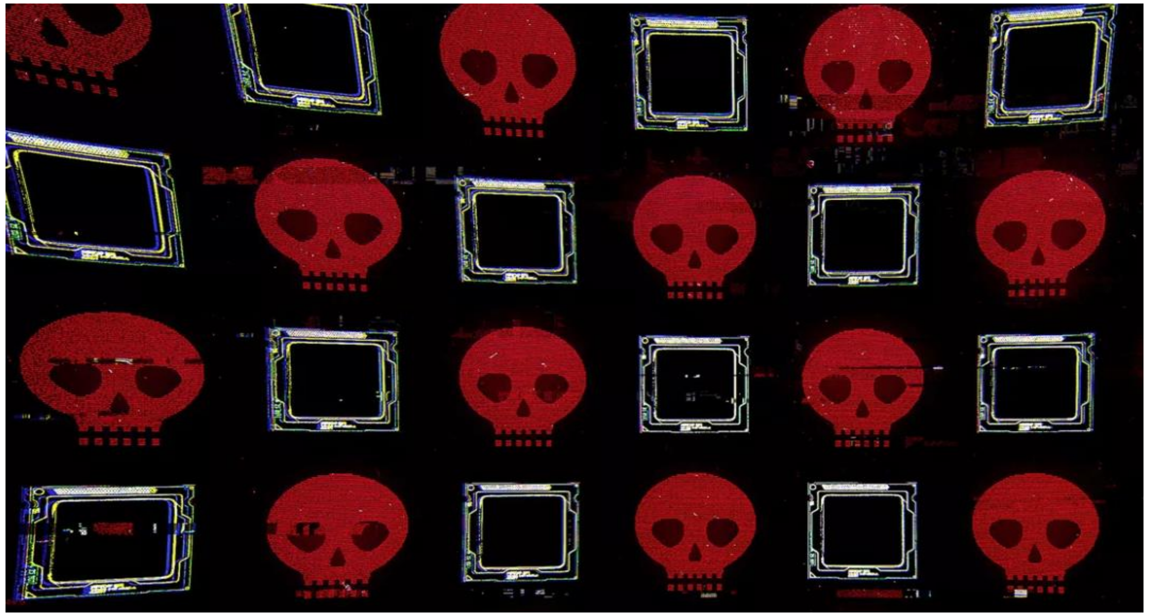
Hardware compromise (low grade/ backdoor)



Impact of Hardware Compromise

THE VERGE

Intel Facing 32 Lawsuits Over Meltdown and Spectre CPU Security Flaws



Jan 4, 2018

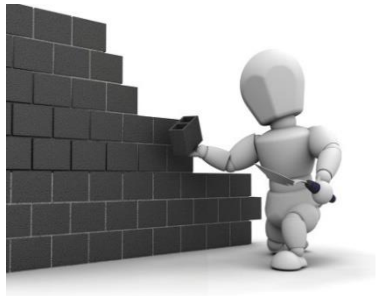
Intel sells off for a second day as massive security exploit shakes the stock

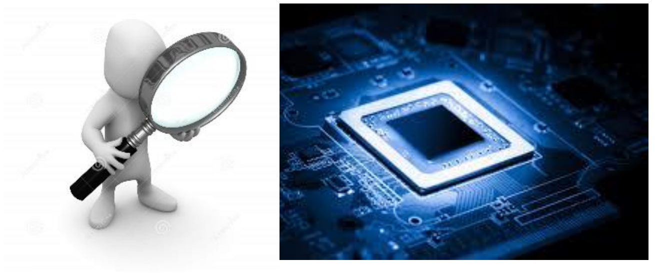
BUSINESS INSIDER

The company accused of selling Apple and Amazon data servers compromised by Chinese spies is getting crushed — it's lost half of its value today

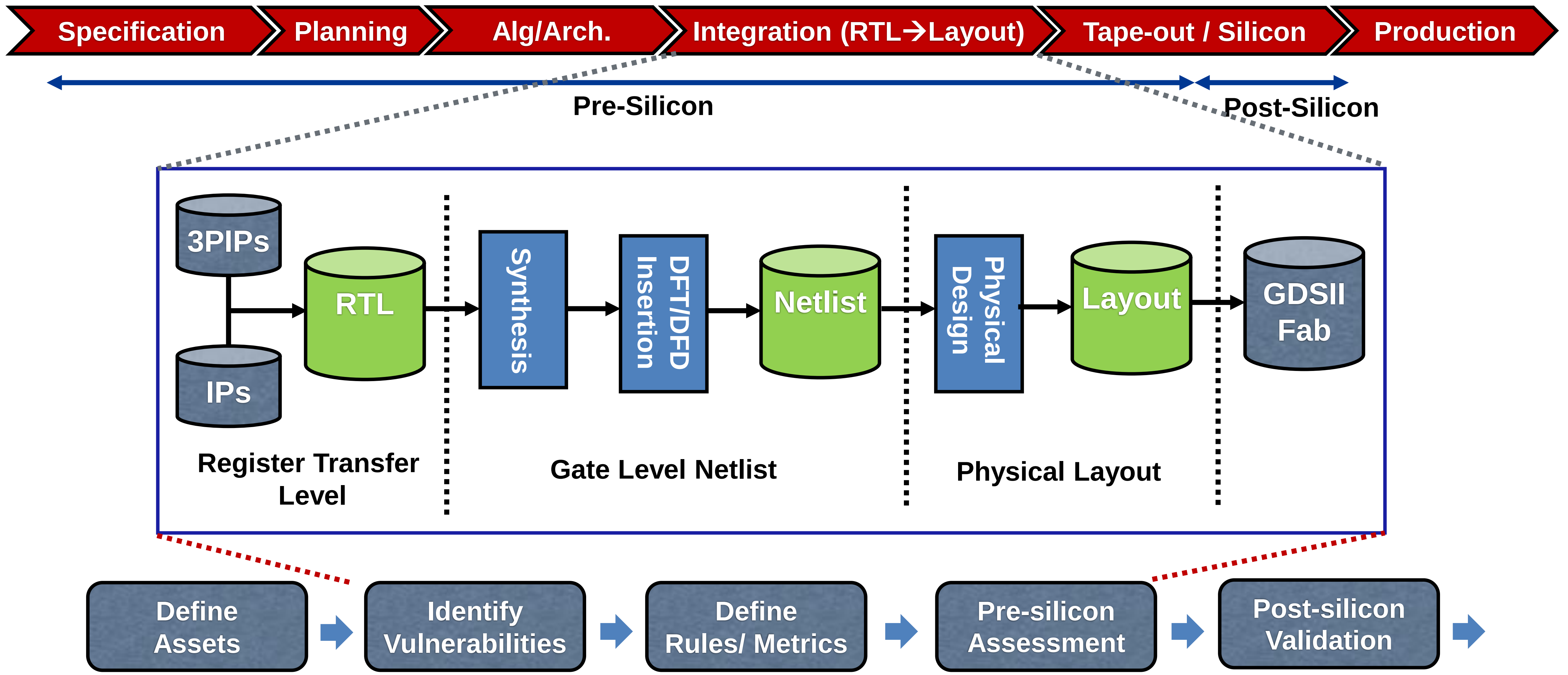
Building a Secure Design

- Consider security from very **beginning**
- Identify what needs to be protected (**assets, IPs,**)
- Evaluate **right level** of security for each asset
  - A door may be sufficient to protect cloths, but a safe should be needed to protect jewelry
- Identify potential **vulnerabilities**
  - Need to develop a vulnerability database
- **Analyze if vulnerabilities exists**
  - **Need to develop CAD tools for security assessment**
- Develop proper **countermeasures**


Security from the start


Security assessment

Security along Design Life-cycle

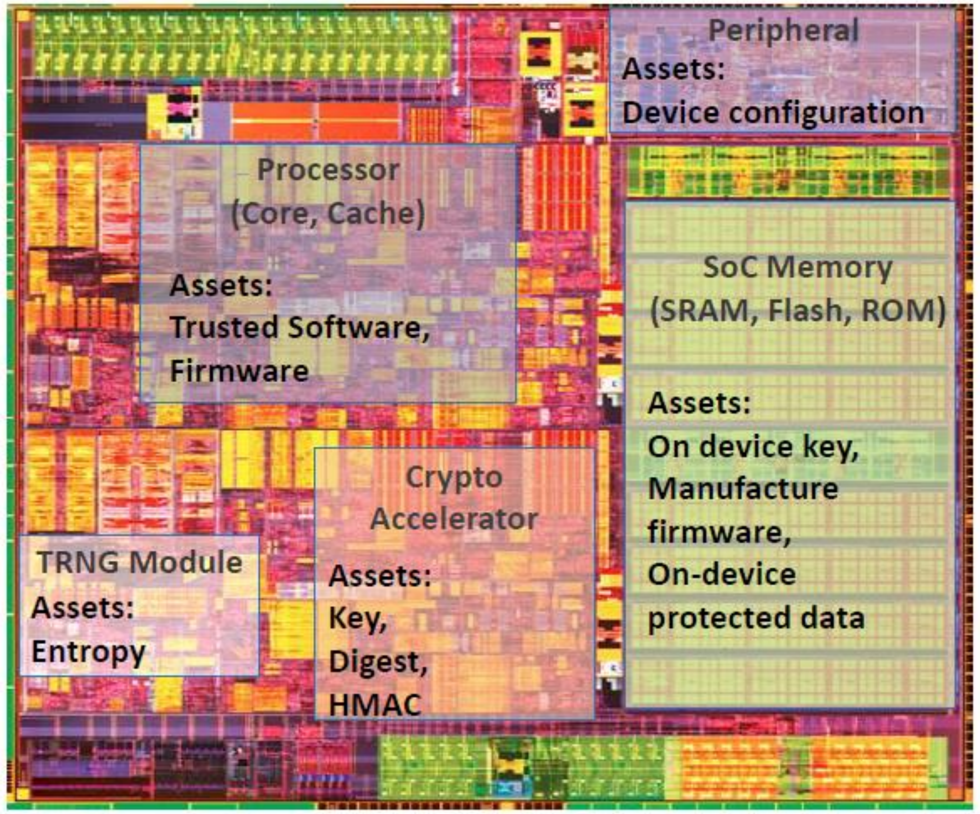


Security Assets

Asset: A resource of value worth protecting from an adversary

Security Assets in SoCs:

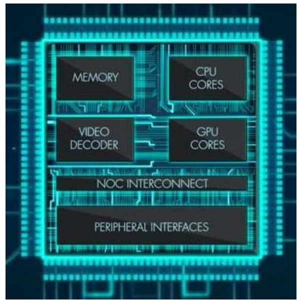
- On-device keys (developer/OEM)
- Device configuration
- Manufacturer Firmware
- Application software
- On-device sensitive data
- Communication credentials
- Random number or entropy
- E-fuse,
- PUF, and more…



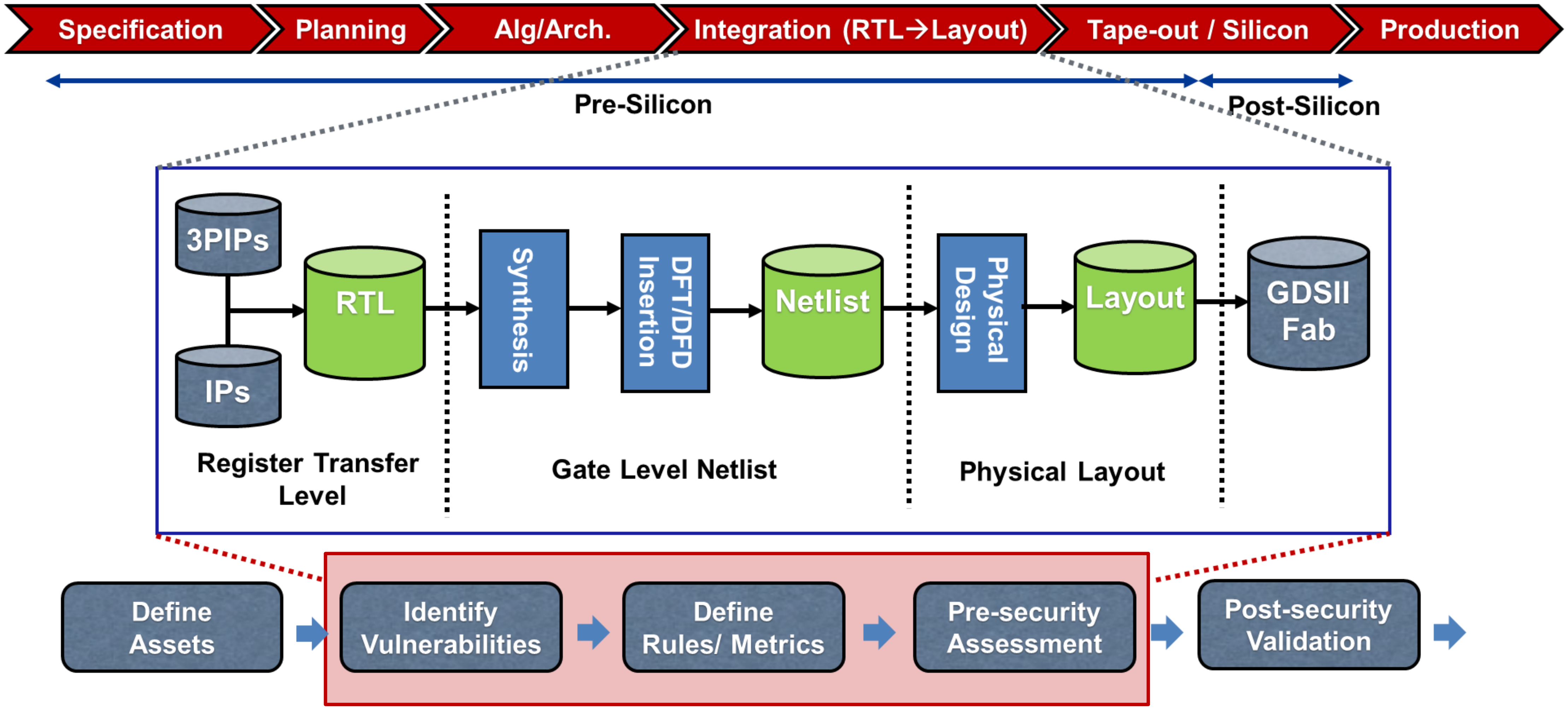
Source: Intel

Assets

- **On device key:** Secret encryption key material permanently embedded on the device
  - Confidentiality violated if compromised
- **Random Number/Entropy:** Cryptographic primitives rely on a good quality and unbiased random number generator
  - Weaken cryptographic algorithms if tampered
- **On-device sensitive data:** Information about the user credential, meter readings, counters
  - Privacy violated if compromised/tampered
- **Chip manufacturer's code:** Low level program instructions, proprietary firmware
  -



Security along SoC Design Life-cycle



CAD for Security

Current Practices

Manual Security Assessment

- **Certification Schemes**: Security verification by an independent official 3rd party
  - Example: payment Card Industry (PCI-DSS and PTS Finance industry)
- **Process overview:**

Security claims

3P Assessment

Final report

- **Suffer from various flaws**
  - Security review depends greatly on the experience
  - No proof that the design is secure against possible attack scenarios

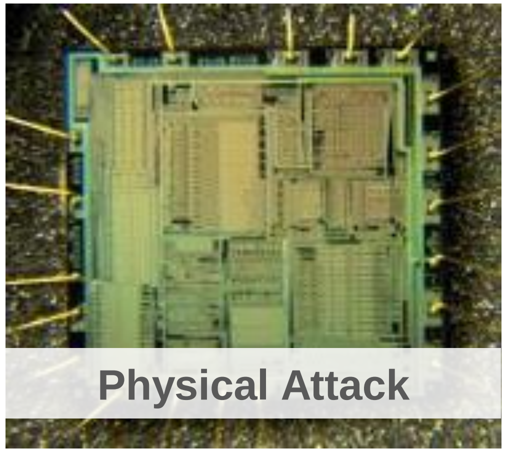
Automation

- Automation made design of modern ICs possible
- Tools made design of chips optimized for different applications possible, i.e., optimized for power, performance, and area
- Metrics played major role
  - Power
  - Performance
  - Area
  - Testability

Automation

- Security is a generic term
  - Vulnerabilities are quite diverse
  - No silver bullet and no one size fits all
  - Relying on SMEs is no longer possible
  - There is a lack of understanding of security issues by designers
  - Emerging vulnerabilities
    - How quickly one can understand it? Mitigate it?
    - Best to be automated
  - Focus on the known vulnerabilities

Fault Injection Attacks

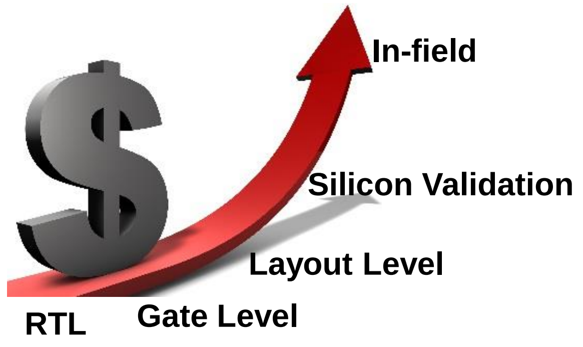

Physical Attack

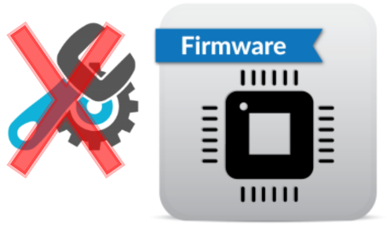
Side-channel Attacks

Untrusted Foundry

Automation

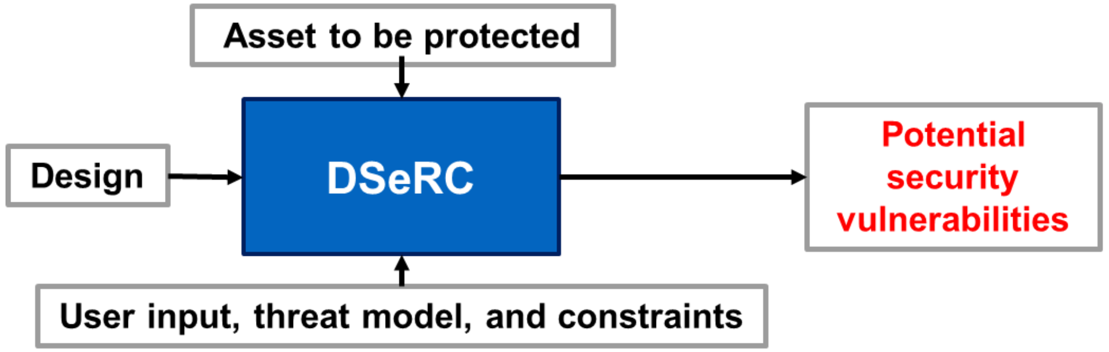
- ***No comprehensive solution to guide security check for SoCs***
- Cost of fixing vulnerabilities found at later stages is **significantly higher – Rule of 10**
  - Unlike software or firmware → **no flexibility** in changing or releasing post-shipment patches for hardware
- Identify security issues during **design phase**
- Address them as early as possible in the design process



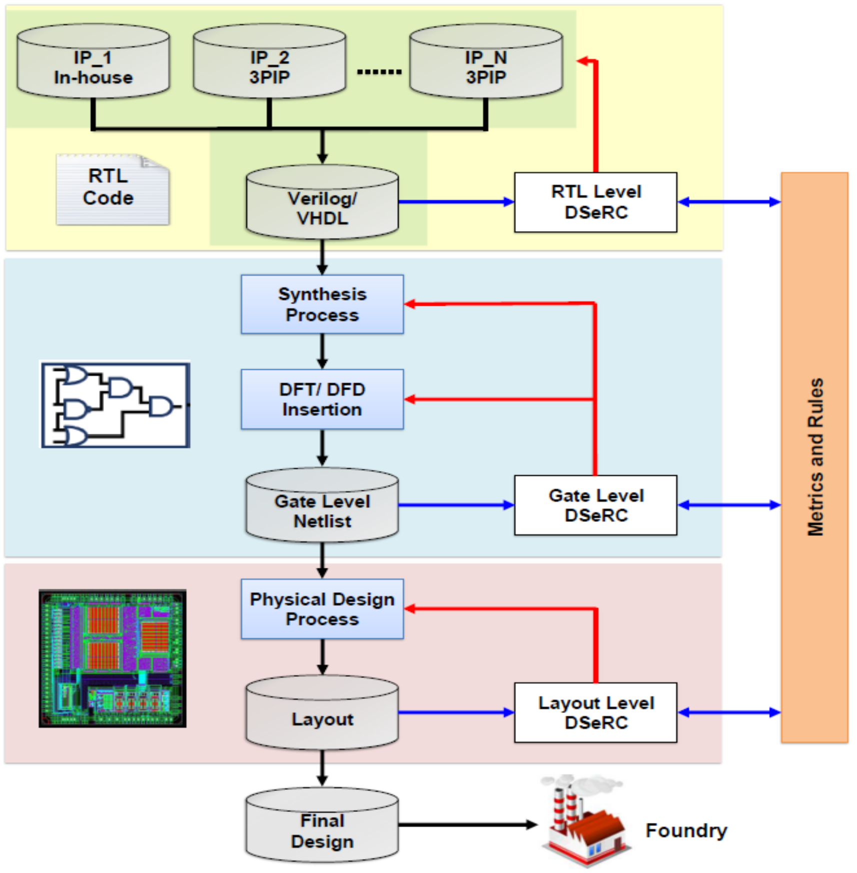


Security Assessment

- A **comprehensive framework** for analyzing **known** security issues in SoCs
- DSeRC framework:
  - **reads** the design files, constraints, threat model, and user input data
  - checks for vulnerabilities at all levels of **abstraction** (RTL, gate, layout, and architectural levels)
- Each vulnerability is tied with a set of **rules** and **metrics** → security can be quantitatively measured



Asset to be protected
Design
DSeRC
Potential security vulnerabilities
User input, threat model, and constraints



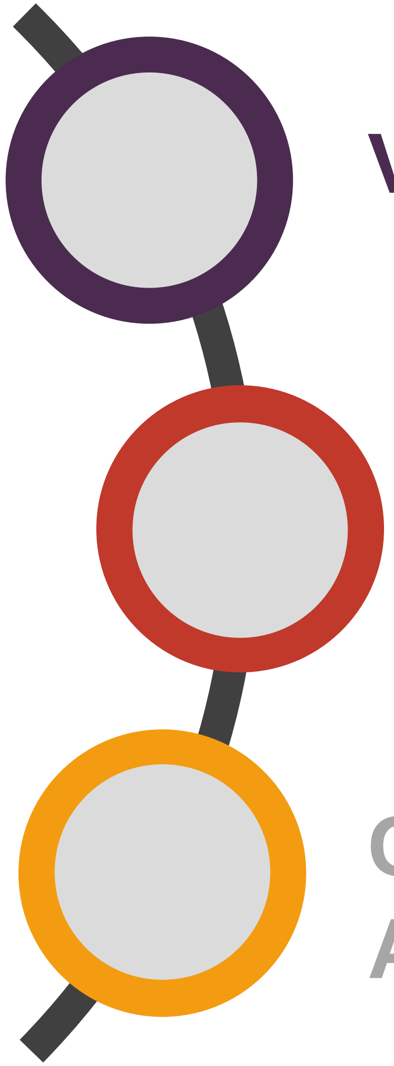
Security Assessment

Vulnerabilities

Rules & Metrics

CAD for Security Assessment

Security Assessment

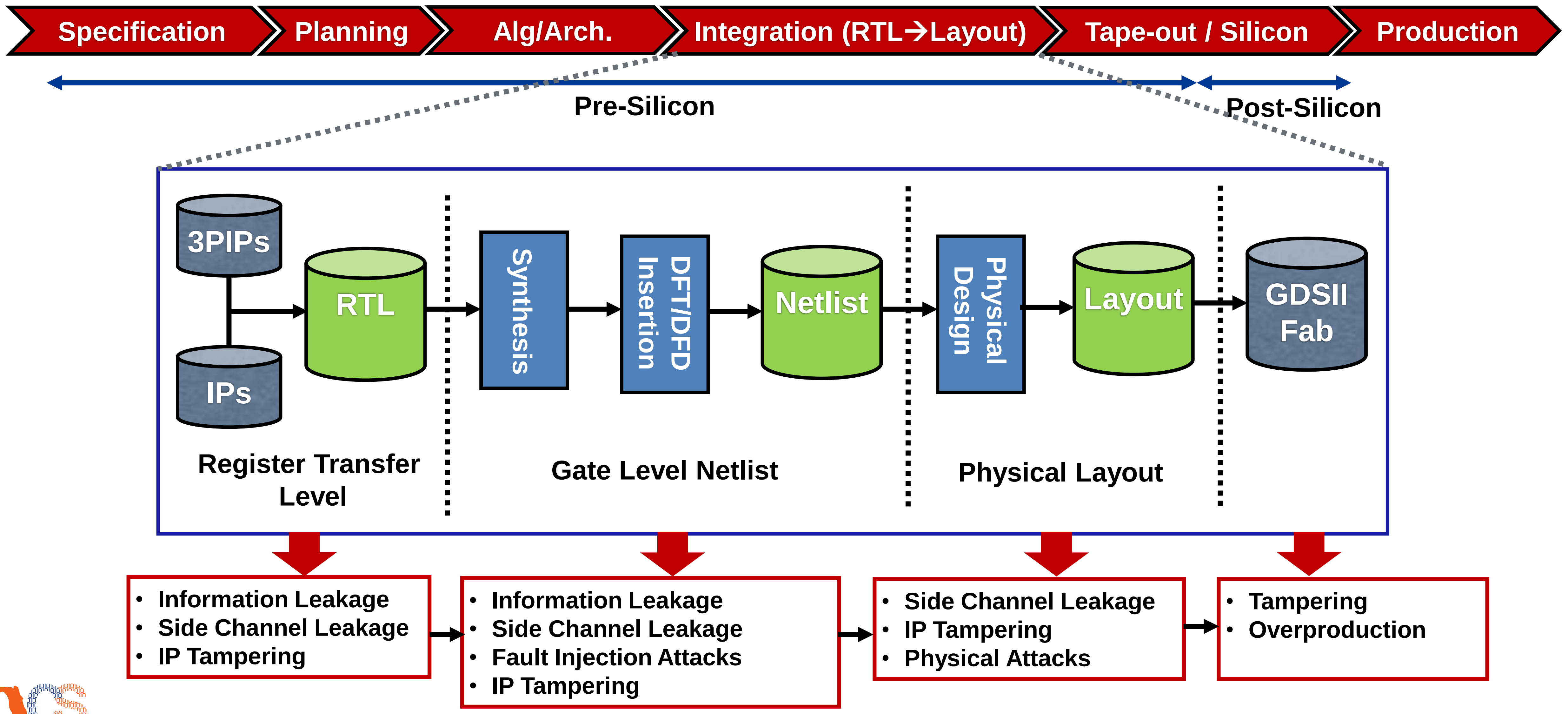


Vulnerabilities

Rules & Metrics

CAD for Security Assessment

Comprehensive Vulnerability Database



- Information Leakage
- Side Channel Leakage
- IP Tampering

- Information Leakage
- Side Channel Leakage
- Fault Injection Attacks
- IP Tampering

- Side Channel Leakage
- IP Tampering
- Physical Attacks

- Tampering
- Overproduction

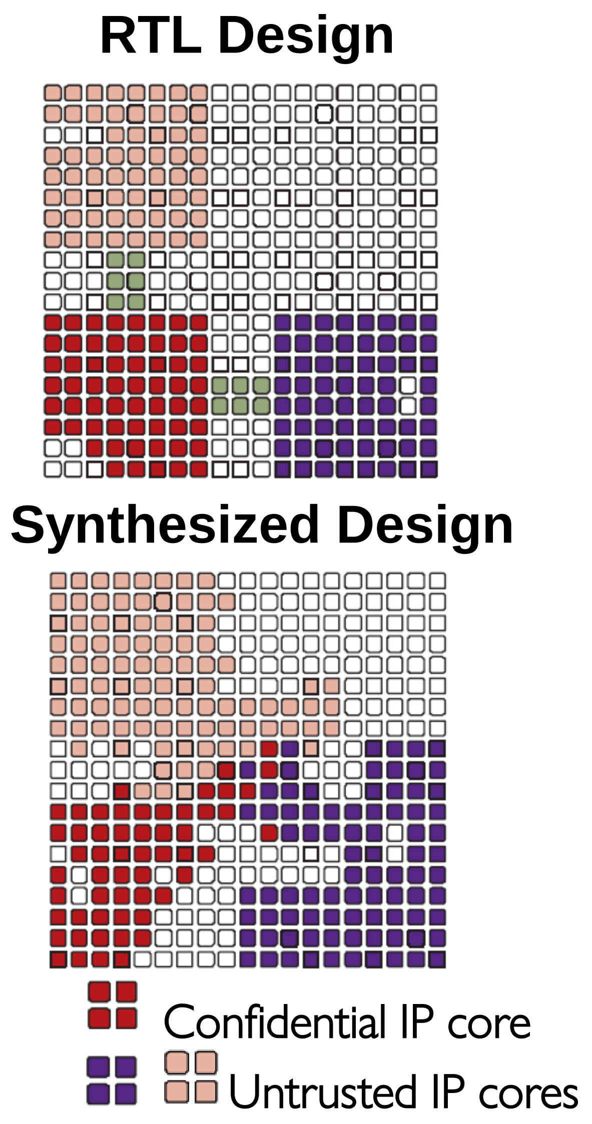
Sources of Vulnerabilities

- **Design Issues**
  - Unintentionally created by (i) **designer's mistakes**, (ii) designer's **lack of understanding** of security problems and requirements in a complex SoC.

- **CAD Tools**
  - Tools are designed to focus on **power**, **performance**, and **area**
  - Can introduce vulnerabilities during **optimization/synthesis – leak information**

Synthesis tools "melt" the IP cores into one circuit – Circuit Flattening

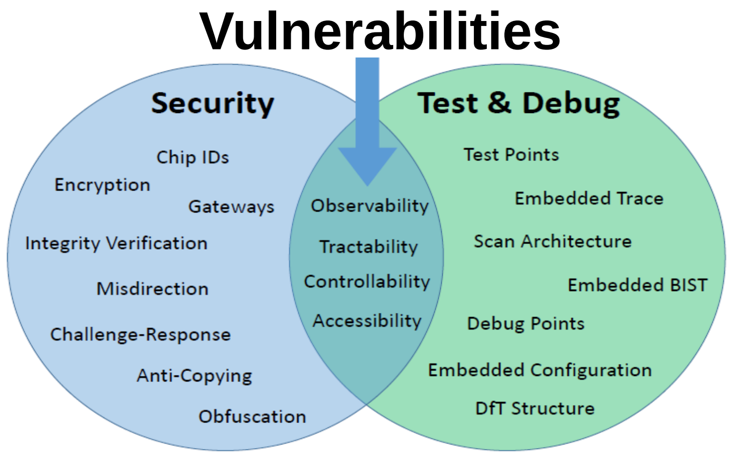
T. Huffmire et al., Moats and Drawbridges: An Isolation Primitive for Reconfigurable Hardware Based Systems , ieee-sp'07.



Sources of Vulnerabilities

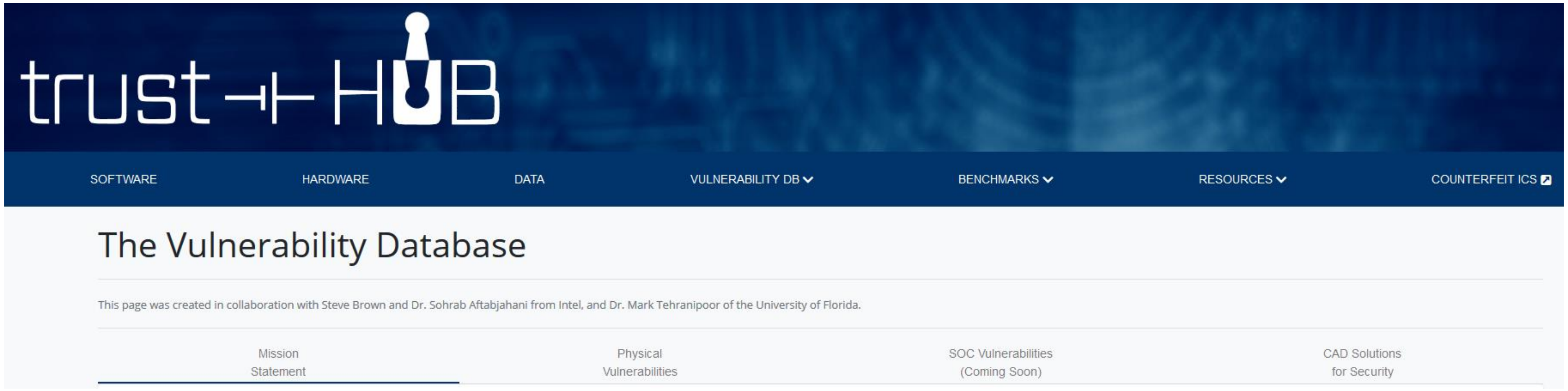
- **DFT and DFD Structures**
  - The increased **controllability** and **observability** added by DFT and DFD structures can create **additional** vulnerabilities

- **Black and White Hats**
  - Side channel attacks, fault injection attacks, information leakage, IP issues, and more



Trust-Hub / TAME Vulnerability Database

- An effort by industry and academic research leaders to provide awareness to researchers and practitioners of hardware security on SoC vulnerabilities
- **Goal**:
  - Develop the National Hardware Vulnerability Database (NHVD) to be shared with the potential of being used as a standard approach for enumerating and screening of various dimensions of security risks for SoCs



The Vulnerability Database

This page was created in collaboration with Steve Brown and Dr. Sohrab Aftabjahani from Intel, and Dr. Mark Tehranipoor of the University of Florida.

Trust-Hub Vulnerability Database

The Vulnerability Database

Timing
- Delay Analysis
- Clock Glitching Injection
- Overclocking
- Underclocking

Fault Injection
- Photon(Laser) Induced current
- Ambient / Ultra - violet
- Ionizing Radiation
- E and M Field
- Voltage Spike
- Temperature
- Over / Under Voltage

Side - Channel Observation Methods
- Acoustic
- Photoemission
- Voltage, Charge contrast
- SEM Inspection
- IREM Inspection
- Temperature Imaging
- E or M Fields
- Current & Power Measurement
- Voltage Measurement
- Indirect Voltage Measurement
- Data Remanence
- Black Box I / O

Logical Attacks
- Brute Force Algorithm
- Protocol Attacks

Die Analysis
- Delayering, Netlist Reconstruction
- Grind
- Section
- Dimple Down
- Photon(Laser) Induced Current
- Focused Ion Beam Deposition
- Focused Ion Beam Removal
- Ion Milling
- Direct Metal or Contact Probing
- Light Sensing
- Circuit Parameter Sensing

Board Analysis
- Delayering, Netlist Reconstruction

Design or FAB Injection
- HW Trojan

Security Assessment

Vulnerabilities

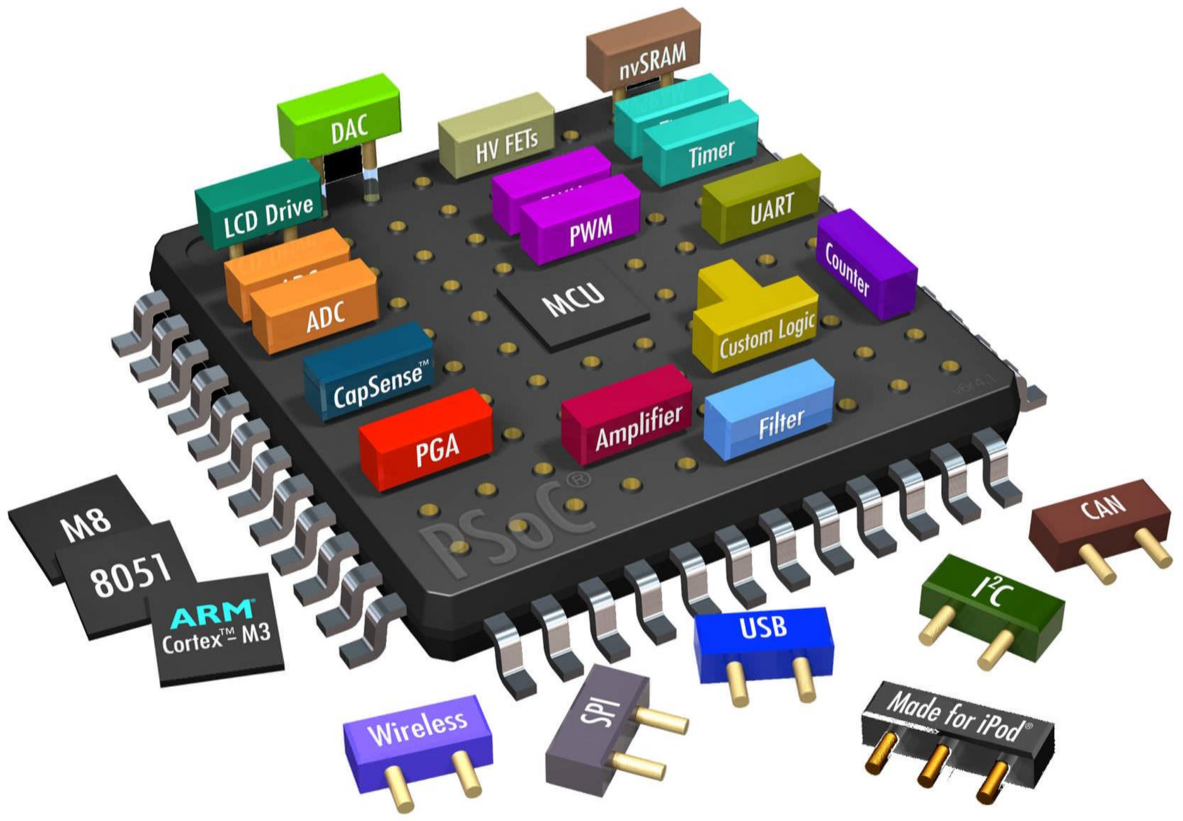
Rules & Metrics

CAD for Security Assessment

Abstraction Levels

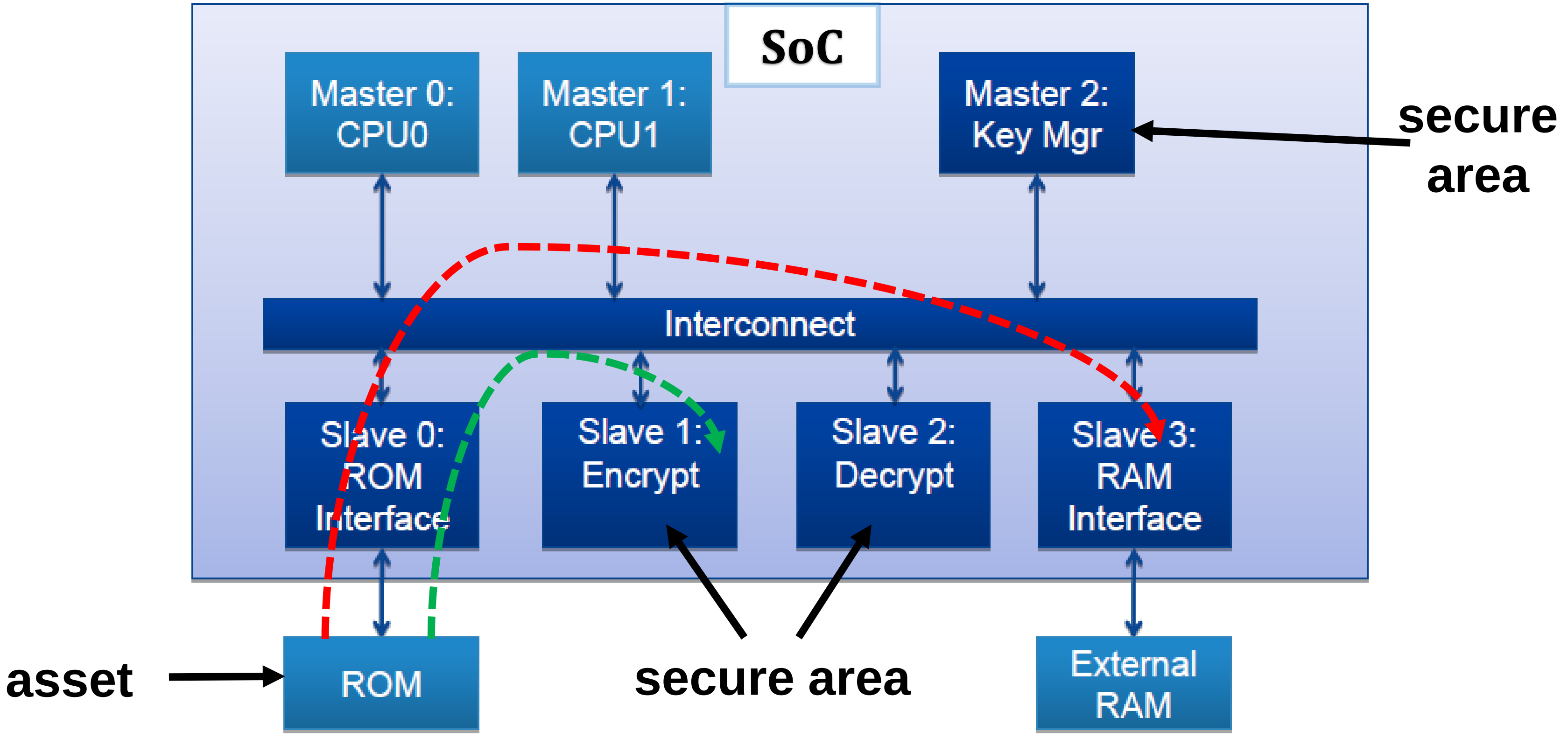
- **IP Level:** Vulnerabilities considered in modular basis at RTL, gate, and physical layout levels

- **SoC Level:** Vulnerabilities considered from system (e.g., SoC) level perspective – interaction between different cores



Vulnerabilities and Rules

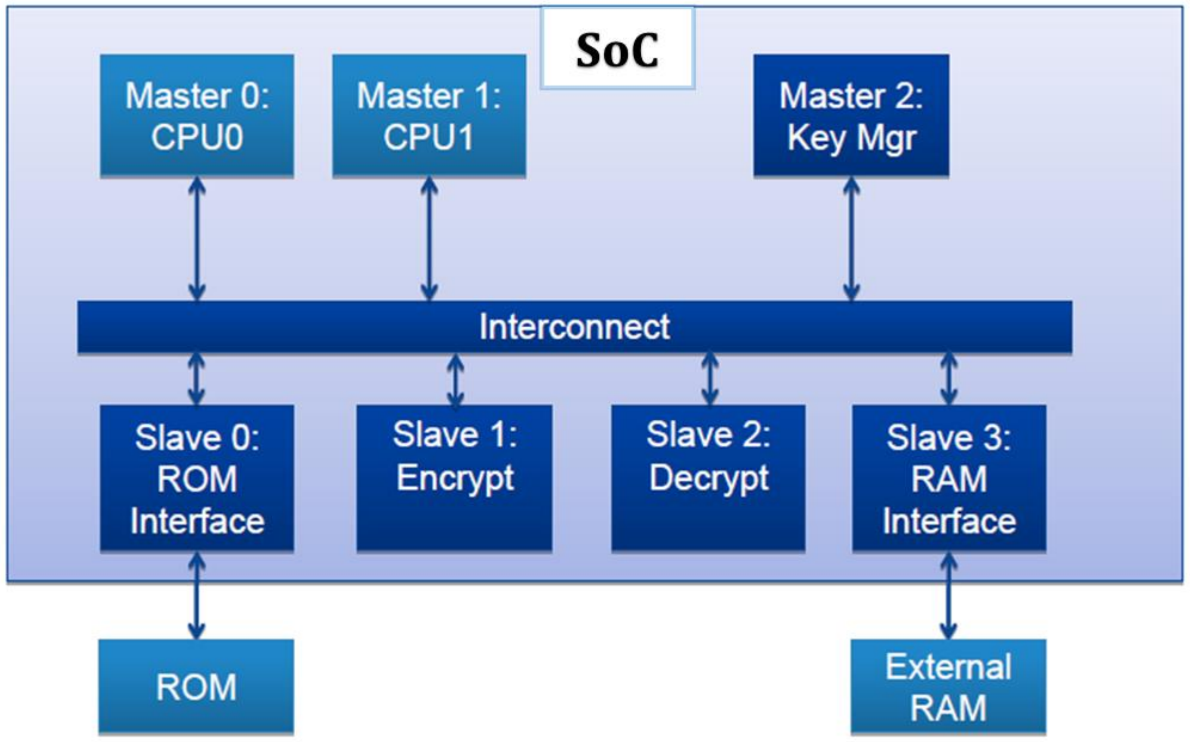
- **Vulnerability:** Asset leakage
- **Rule:** An asset should never propagate to any location where an attacker can observe it



Source: Jasper

More Examples of Rules

- uP in user mode should **never access** OS kernel memory
- During crypto operation reset, reading intermediate results, changing keys, and data operations **are prohibited**
- During cryptographic asset (e.g. key) transfer from the system memory to the crypto-core registers, all other IP accesses to the bus **are disabled**
- The power management module **can enable** a modification in the clock frequencies only when the core is not in active mode
- During debug, **no accesses** are allowed to the security critical part of memory



Source: Jasper

Vulnerabilities, Metrics and Rules

	Vulnerability	Metric	Rule	Attack (Attacker)
RTL Level	Dangerous Don't Cares	Identify all 'X' assignments and check if 'X' can propagate to observable outputs	'X' assignments should not be propagated to observable output	Hardware Trojan (Insider)
	Hard-to-control & hard-to-observe Signals	Statement hardness and signal observability	Statement hardness (signal observbility) should be lower (higher) than a predefined threshold	Hardware Trojan (Insider)
	Asset leakage	Structure checking and IFT	Security sensitive assets should not be exposed to observable points	Asset hacking (End user)
	….			
Gate Level	Hard-to-Control & hard-to-observe Nets	Net controllability and observability	Controllability and observability should be higher than a threshold value	Hardware Trojan (Insider)
	Vulnerable FSM	Vulnerability factor of fault injection (VF_{FI}) and Trojan insertion (VF_{Tro})	VF_{FI} and VF_{Tro} should be zero	Fault injection, Hardware Trojan (Insider, end user)
	Asset Leakage	Confidentiality and integrity assessment	Assets should not be leaked through observable points	Asset hacking (End user)
	Design-for-Test (DFT), JTAG/IJTAG Vulnerabilities	Confidentiality and integrity assessment	Assets should not be leaked or accessed through DFT structure	Asset hacking (End user)
	Design-for-Debug structure Vulnerabilities	Confidentiality and integrity assessment	Assets should not be leaked or accessed through DFD structure	Asset hacking (End user)
	…..			
Layout Level	Side-Channel Leakage	Side-channel vulnerability (SCV)	SVF should be lower than a threshold value	Side-channel attack (End user)
	Microprobing Vulnerability	Exposed area of the security-critical nets which are vulnerable to microprobing attack	The exposed area should be lower than a threshold value	Micro-probing attack (Professional attacker)
	Trojan Insertion – unused space	Unused space analysis	Unused space should be lower than a threshold value	Untrusted foundry
	…..			

Security Assessment

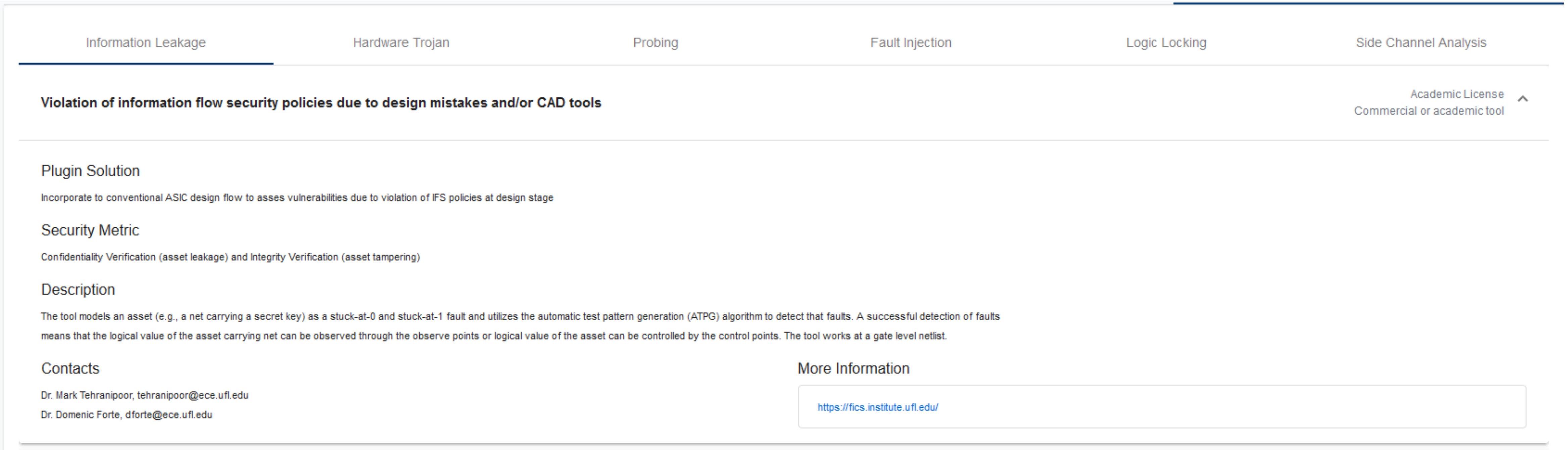
Vulnerabilities

Rules & Metrics

CAD for Security Assessment

Trust-Hub CAD for Security

The Vulnerability Database



Violation of information flow security policies due to design mistakes and/or CAD tools

Academic License
Commercial or academic tool

Plugin Solution

Incorporate to conventional ASIC design flow to asses vulnerabilities due to violation of IFS policies at design stage

Security Metric

Confidentiality Verification (asset leakage) and Integrity Verification (asset tampering)

Description

The tool models an asset (e.g., a net carrying a secret key) as a stuck-at-0 and stuck-at-1 fault and utilizes the automatic test pattern generation (ATPG) algorithm to detect that faults. A successful detection of faults means that the logical value of the asset carrying net can be observed through the observe points or logical value of the asset can be controlled by the control points. The tool works at a gate level netlist.

Contacts

Dr. Mark Tehranipoor, tehranipoor@ece.ufl.edu
Dr. Domenic Forte, dforte@ece.ufl.edu

More Information

https://fics.institute.ufl.edu/

CAD for Security

C/C++

- Information Leakage
- Control flow violations
- Side channel leakage

RTL

- Susceptibility to Trojan Insertion
- Test Generation for Trust Verification
- Power Side-channel Leakage Assessment (RTL-PSC)
- PSC Test Generation (TG)
- EM Leakage Assessment
- Formal Verification of Security Properties

Gate

- Susceptibility of Fault Injection (AVFSM)
- Automated Mapping of Security Properties from C to Gate Level (AutoMap)
- Information Flow Security (IFS)
- Power Side-channel Leakage Assessment (SCRIPT)
- Fault Injection Assessment

Layout

- Susceptibility of Probing Attacks
- Power Side-channel Leakage Assessment (TVLA)

CAD for Security

C/C++

- Information Leakage
- Control flow violations
- Side channel leakage

RTL

- Susceptibility to Trojan Insertion
- Test Generation for Trust Verification
- Power Side-channel Leakage Assessment (RTL-PSC)
- PSC Test Generation (TG)
- EM Leakage Assessment
- Formal Verification of Security Properties

Gate

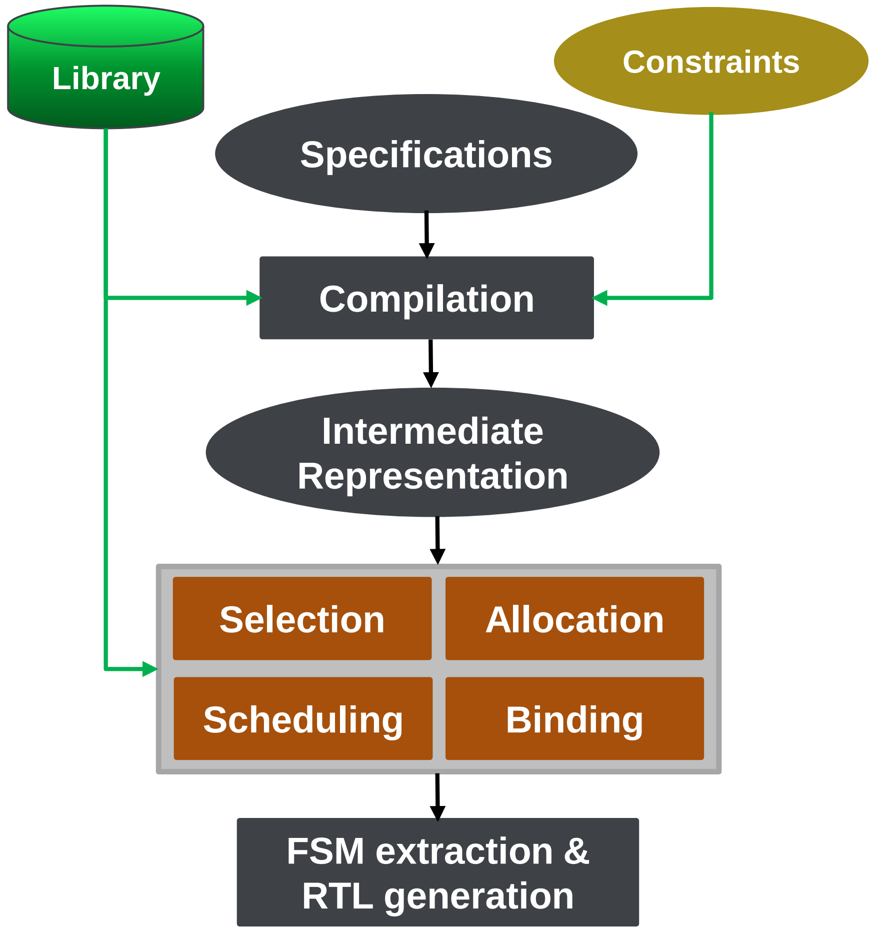
- Susceptibility of Fault Injection (AVFSM)
- Automated Mapping of Security Properties from C to Gate Level (AutoMap)
- Information Flow Security (IFS)
- Power Side-channel Leakage Assessment (SCRIPT)
- Fault Injection Assessment

Layout

- Susceptibility of Probing Attacks
- Power Side-channel Leakage Assessment (TVLA)

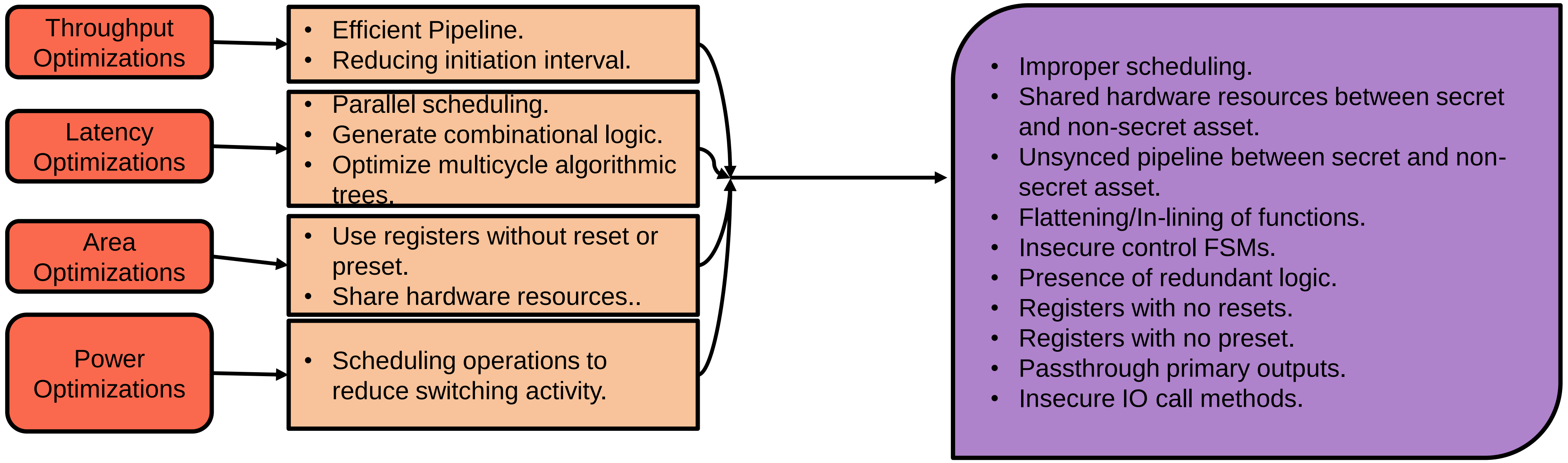
HLS Overview

- High-Level Synthesis (HLS) translates high-level C/C++ code to HDL-level VHDL/Verilog. Advantages:
  - reduced time-to-market
  - easier implantation of complex RTL designing
  - suitable for Crypto modules, Machine Leaning, and AI
- However, due to prioritizing performance, the security aspects are overlooked in specific scenarios
- This tool explores some of the security vulnerabilities introduced by HLS



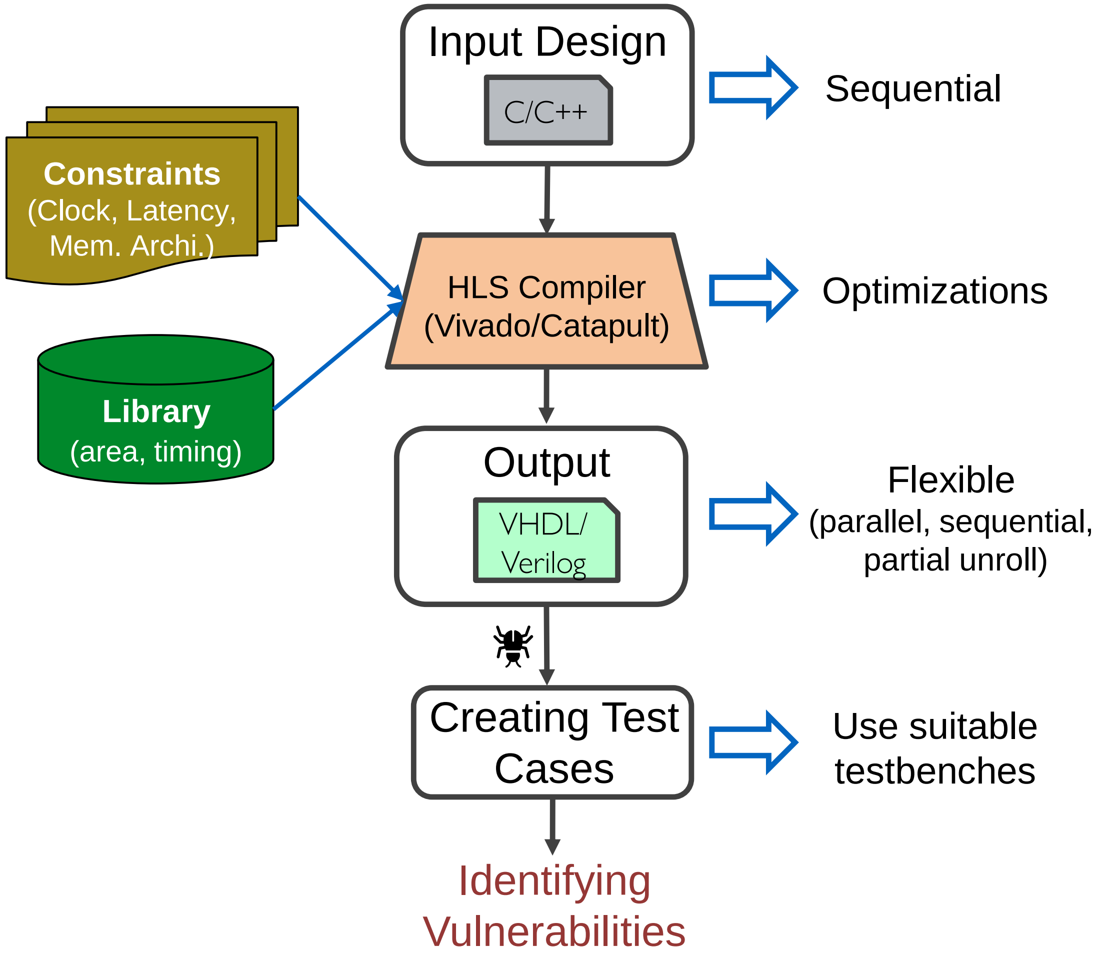
High-Level Synthesis steps

HLS-related Potential Hardware Vulnerabilities


Throughput Optimizations
Latency Optimizations
Area Optimizations
Power Optimizations
• Efficient Pipeline.
• Reducing initiation interval.
• Parallel scheduling.
• Generate combinational logic.
• Optimize multicycle algorithmic trees.
• Use registers without reset or preset.
• Share hardware resources..
• Scheduling operations to reduce switching activity.
• Improper scheduling.
• Shared hardware resources between secret and non-secret asset.
• Unsynced pipeline between secret and non-secret asset.
• Flattening/In-lining of functions.
• Insecure control FSMs.
• Presence of redundant logic.
• Registers with no resets.
• Registers with no preset.
• Passthrough primary outputs.
• Insecure IO call methods.

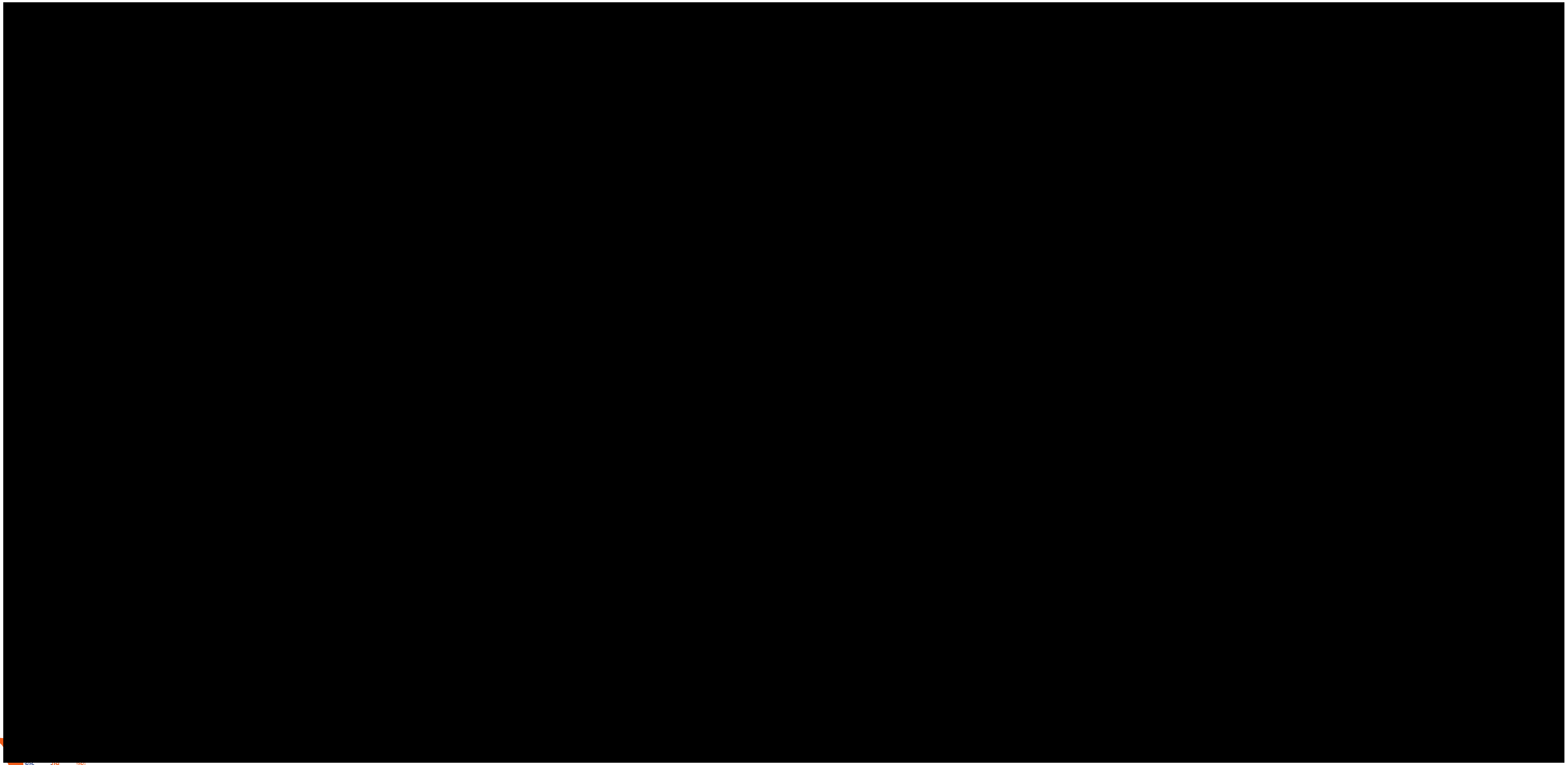
Workflow

- The first step is to use benchmark designs (C/C++) as input to HLS compiler
- The compiler outputs the HDL form of the design
- This HDL is simulated with suitable test conditions
- Assessing if any kind of security vulnerability can be found
- Common vulnerabilities: confidentiality and integrity violations (e.g., information leakage, and inadequate access control)



HLS Vulnerability Detection Demo

Demo Video



CAD for Security

C/C++

- Information Leakage
- Control flow violations
- Side channel leakage

RTL

- Susceptibility to Trojan Insertion
- Test Generation for Trust Verification
- Power Side-channel Leakage Assessment (RTL-PSC)
- PSC Test Generation (TG)
- EM Leakage Assessment
- Formal Verification of Security Properties

Gate

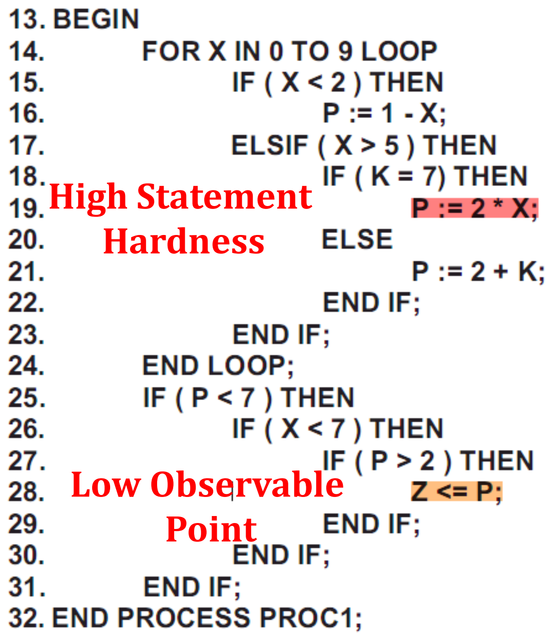
- Susceptibility of Fault Injection (AVFSM)
- Automated Mapping of Security Properties from C to Gate Level (AutoMap)
- Information Flow Security (IFS)
- Power Side-channel Leakage Assessment (SCRIPT)
- Fault Injection Assessment

Layout

- Susceptibility of Probing Attacks
- Power Side-channel Leakage Assessment (TVLA)

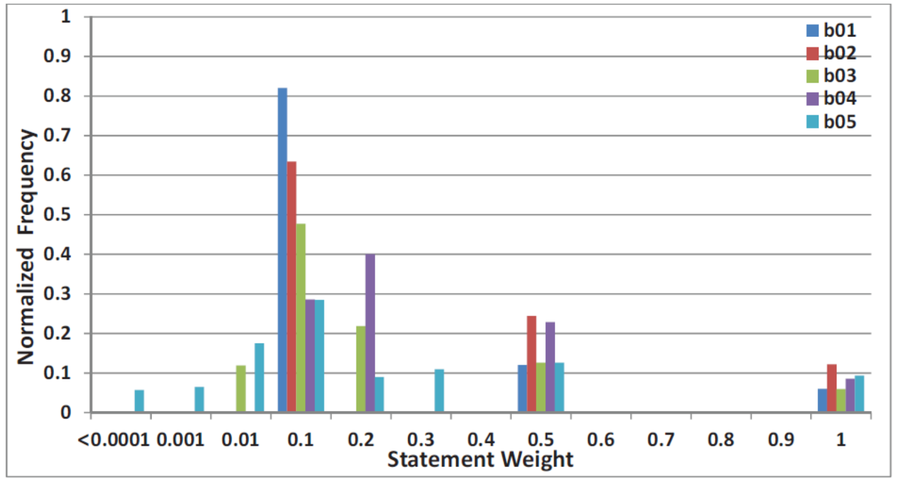
Susceptibility to Trojan Insertion

- Sections in a circuit with **low controllability and observability** are considered potential areas for implementing **Trojans**
- **Metrics:**
  - **Statement hardness**: Difficulty of executing a statement
  - **Observability**: Difficulty of observing a signal
- **Rule 1:** Statement hardness of each statement should be lower than a predefined threshold
- **Rule 2:** Observability of each observable signal should be higher than a predefined threshold

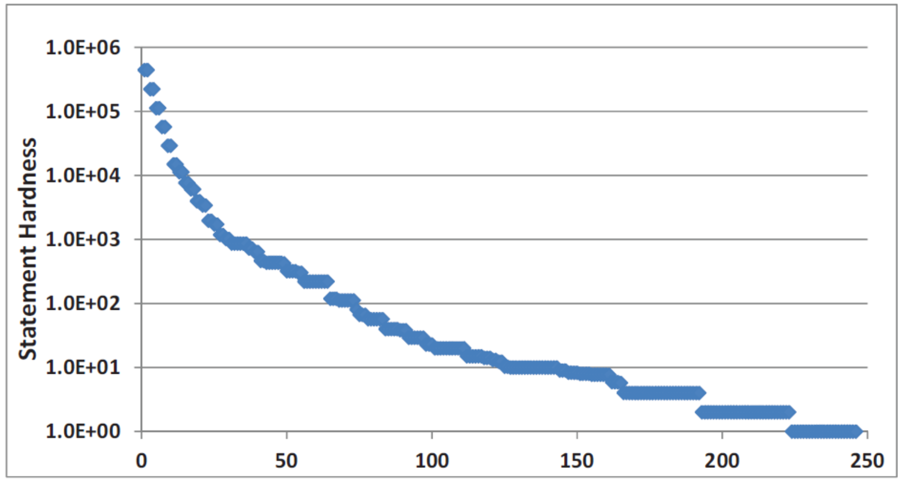

High Statement Hardness
Low Observable Point

```
BEGIN
        FOR X IN 0 TO 9 LOOP
                IF ( X < 2 ) THEN
                        P := 1 - X;
                ELSIF ( X > 5 ) THEN
                        IF ( K = 7) THEN
                                P := 2 * X;
                        ELSE
                                P := 2 + K;
                        END IF;
                END IF;
        END LOOP;
        IF ( P < 7 ) THEN
                IF ( X < 7 ) THEN
                        IF ( P > 2 ) THEN
                                Z <= P;
                        END IF;
                END IF;
        END IF;
END PROCESS PROC1;
```

Susceptibility to Trojan Insertion



Statement weight analysis.



Statement hardness for b05.

- Application of the Tool:
  - Can be used to determine which parts of a circuit are more susceptible to Trojan insertion
  - Can be used to track and identify malicious part included in the code by a rogue employee (insider threat)

CAD for Security

C/C++

- Information Leakage
- Control flow violations
- Side channel leakage

RTL

- Susceptibility to Trojan Insertion
- Test Generation for Trust Verification
- Power Side-channel Leakage Assessment (RTL-PSC)
- PSC Test Generation (TG)
- EM Leakage Assessment
- Formal Verification of Security Properties

Gate

- Susceptibility of Fault Injection (AVFSM)
- Automated Mapping of Security Properties from C to Gate Level (AutoMap)
- Information Flow Security (IFS)
- Power Side-channel Leakage Assessment (SCRIPT)
- Fault Injection Assessment

Layout

- Susceptibility of Probing Attacks
- Power Side-channel Leakage Assessment (TVLA)

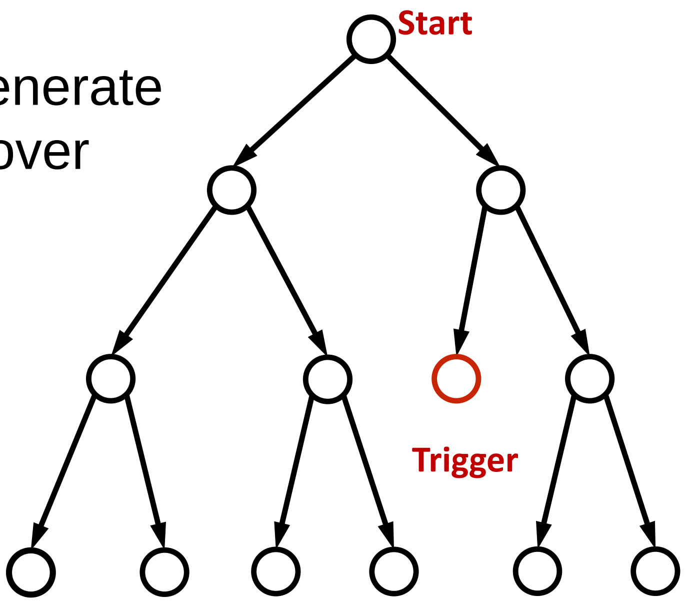
Motivation

- ***Goal:*** **Given a RTL design, we need to generate test for covering all suspicious targets**

```
func (a) {
    if (a == 5)
        activate Trojan
    else
        normal operation
}
```

Target

We want to generate test case to cover *target*



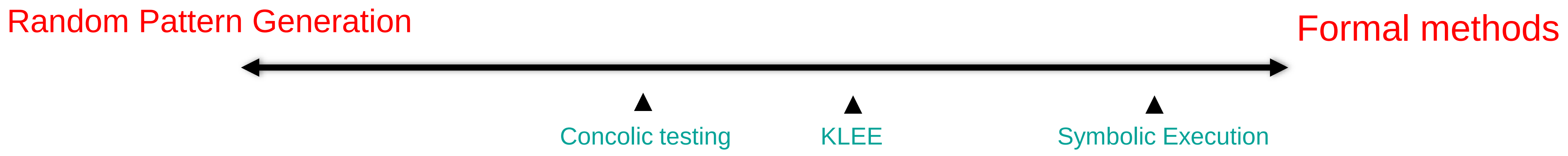
Manual Test Writing

- Small program – Doable
- Large program – Hard
- RTL designs - Harder (complex designs, concurrency, multiple clock domain etc.)
- Trojans - Even harder (Occurs only on extremely rare scenarios)

Test Generation for Hardware Trojan Detection

- **Problem:**
  - Threat: Hardware Trojan inserted in a RTL design that leaks an asset to the outside world.
  - Finding Test patterns that trigger the Trojan.
  - Rareness of certain regions of code is our metric to find candidates.

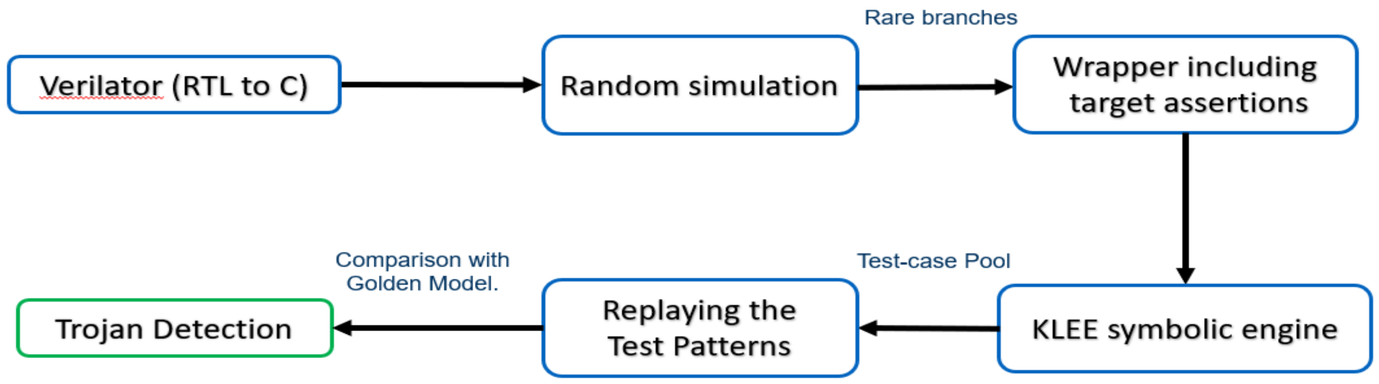
- The WhiteBox vs BlackBox.



- Tradeoff between scalability and coverage.

Test Generation Steps

- Formal methods are not scalable and random test generation does not provide good coverage → Symbolic Execution.
  - Steps in obtaining the trigger patterns:
    - Step 1: Instrumentation of the RTL code and Translating it to C level
    - Step 2: Random simulation for sufficient cycles to identify rare branches.
    - Step 3: Translating the rare branches to KLEE assertions.
    - Step 3: Symbolic execution with Cone of influence analysis to cover the assertions.
    - Step 4: Test pool of all the test patterns that are candidates for Trojan trigger.



Test Generation for Hardware Trojan Detection

- Symbolic execution generates the test patterns by using a SMT solver at its core.
- The Platform was tested on AES trojan inserted designs.

	Conquest		Klee	
Benchmark	Timing(s)	Covered Rare Branches/total rare branches	Timing(s)	Covered Rare Branches/total rare branches
AES-T500	427.8	5/5	31.3	5/5
AES-T1000	22.1	2/2	4.5	2/2
AES-T1100	144.3	4/5	15.7	5/5
AES-T1300	178.7	7/9	57.8	9/9
AES-T2000	298.3	4/5	16.0	5/5
Cb_aes_01	1.38	2/2	4.37	2/2
Cb_aes_05	6.81	2/2	2.8	2/2
Cb_aes_10_15	22.3	2/2	8.37	2/2

Test Generation Demo

Demo Video

CAD for Security

C/C++

- Information Leakage
- Control flow violations
- Side channel leakage

RTL

- Susceptibility to Trojan Insertion
- Test Generation for Trust Verification
- Power Side-channel Leakage Assessment (RTL-PSC)
- PSC Test Generation (TG)
- EM Leakage Assessment
- Formal Verification of Security Properties

Gate

- Susceptibility of Fault Injection (AVFSM)
- Automated Mapping of Security Properties from C to Gate Level (AutoMap)
- Information Flow Security (IFS)
- Power Side-channel Leakage Assessment (SCRIPT)
- Fault Injection Assessment

Layout

- Susceptibility of Probing Attacks
- Power Side-channel Leakage Assessment (TVLA)

Objective and Motivations

- Side-channel attacks have been a major concern to security community

- Side-channel countermeasures and leakage assessment have been studied

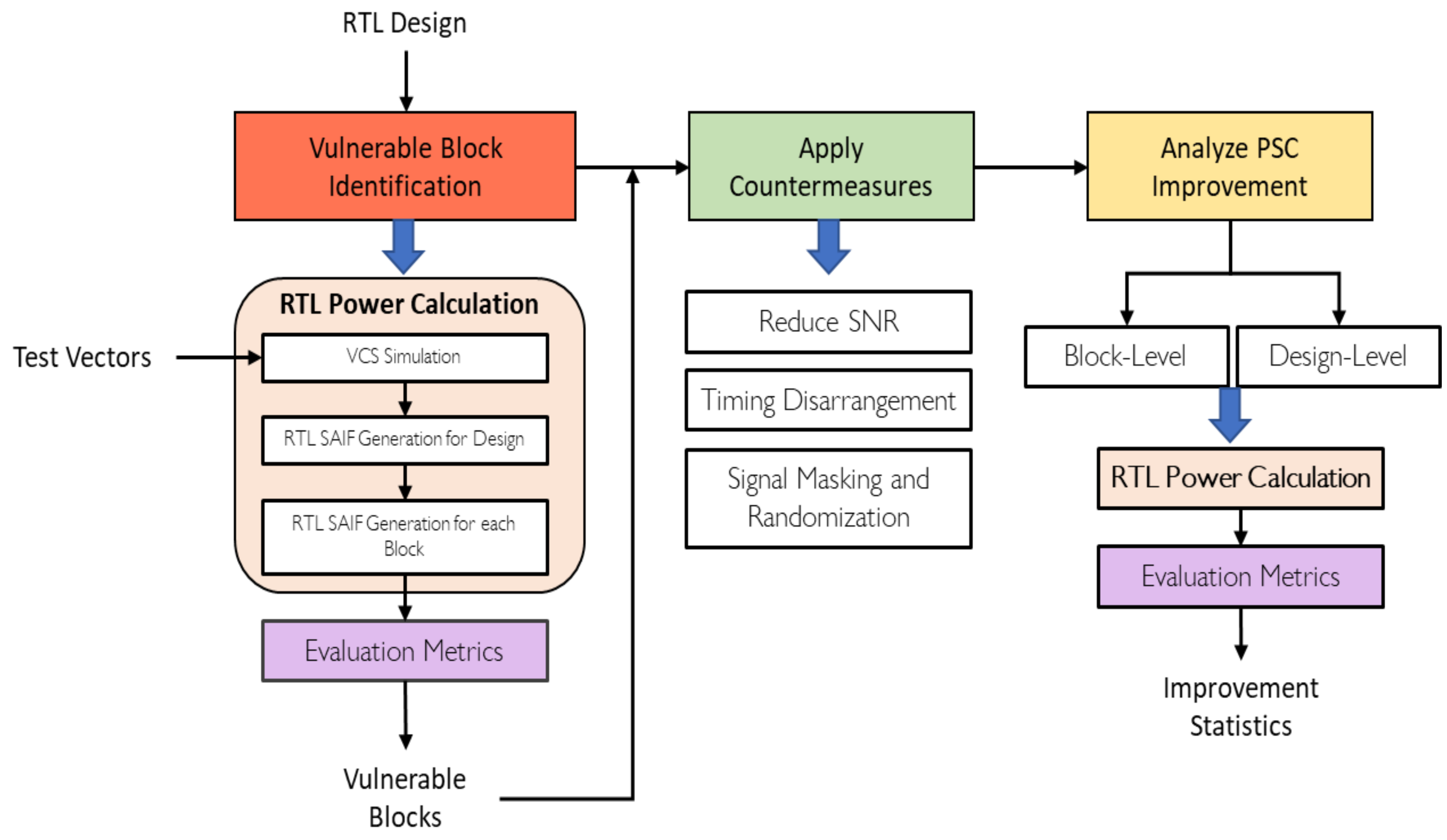
- However, they mostly focus on *post-silicon* side-channel assessment
  - Difficult to find the leakage sources or modules
  - Too *expensive* in modifying designs to address leakage issues

- Two proposed frameworks PSC-Sim/TG to fulfill side-channel assessment at RTL
  - Leakage evaluation at the earliest phase allows more flexibility
  - Technology independent analysis
  - CAD tools for flow automation

Overview: PSC-Sim

KL Divergence Metric

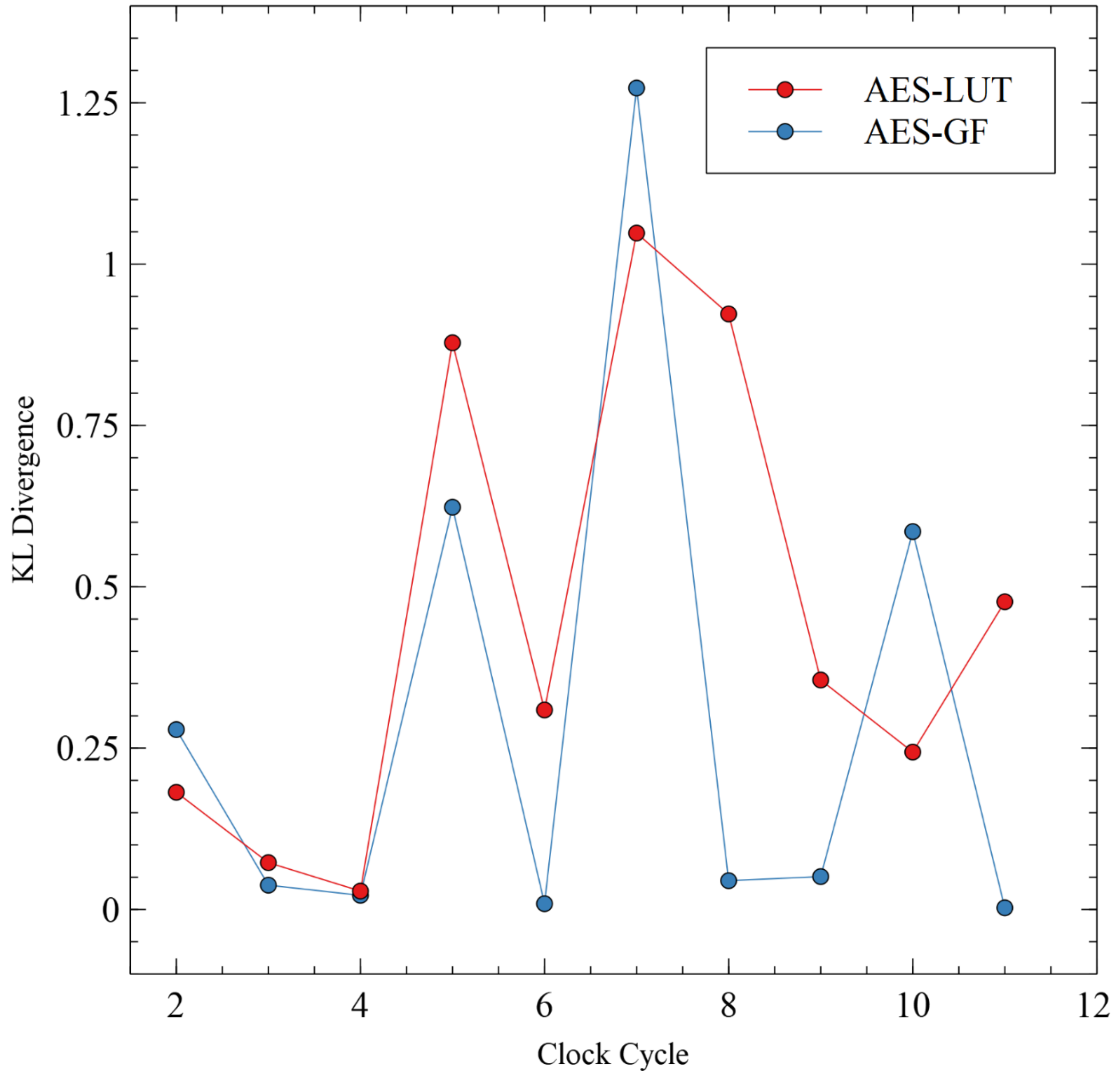
- For any two given secret keys, metric helps to visualize by how much the power distribution functions (PDFs) associated with the keys differ.
- Larger the distance, higher probability of an attacker guessing the key correctly in fewer number traces by performing differential power analysis.
- Exhaustive testing for all the key-pairs at design-level is time consuming, hence key pairs are intelligently selected.
- Exhaustive testing at modular-level is done to test for all possible secret inputs to the block. It also helps to replicate the scenario where intrinsic noise from other modules hide the vulnerable block which attacker may exploit after pre-processing of power traces post-silicon.



Identifying Power Leakage

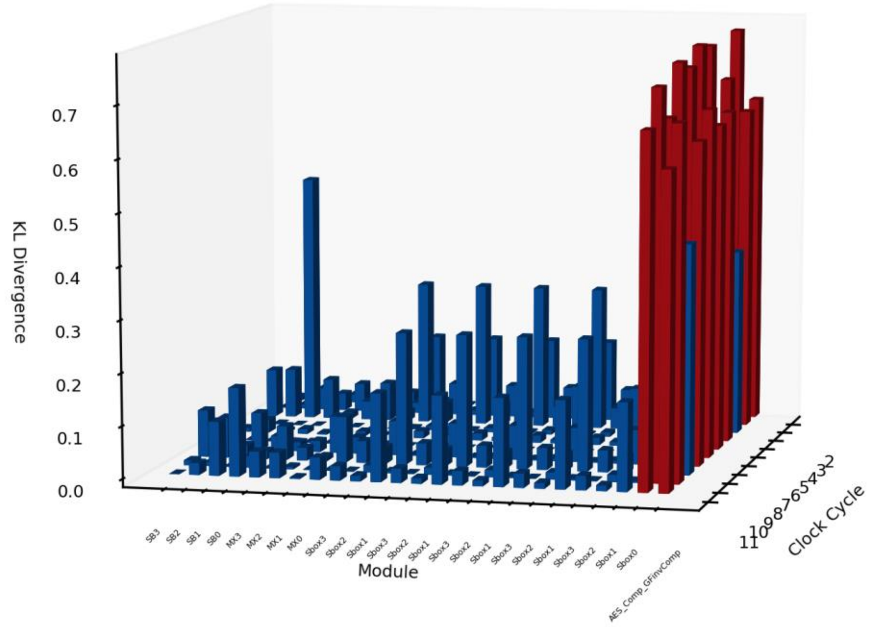
- Total power KL Divergence between key pairs all 0’s and all F’s.

- KL divergence of more than 0.03 shows that there is more than 90% probability of attacker able to distinguish between key pairs by side channel analysis.

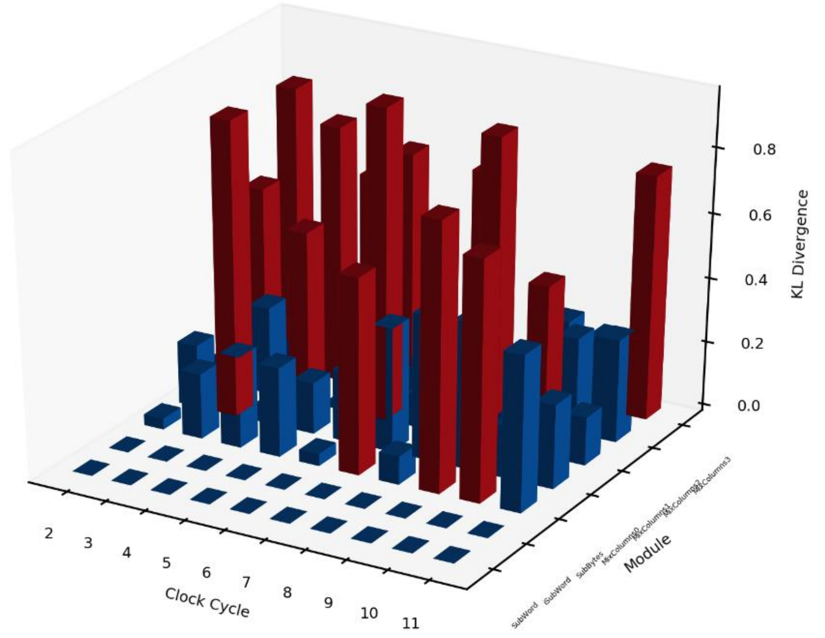


Identification of Vulnerable Module

Worst-case Key Pair
Key0: 0x0000_0000_0000_0000
Key2: 0xFFFF_FFFF_FFFF_FFFF



AES-GF Design: Vulnerable module identification

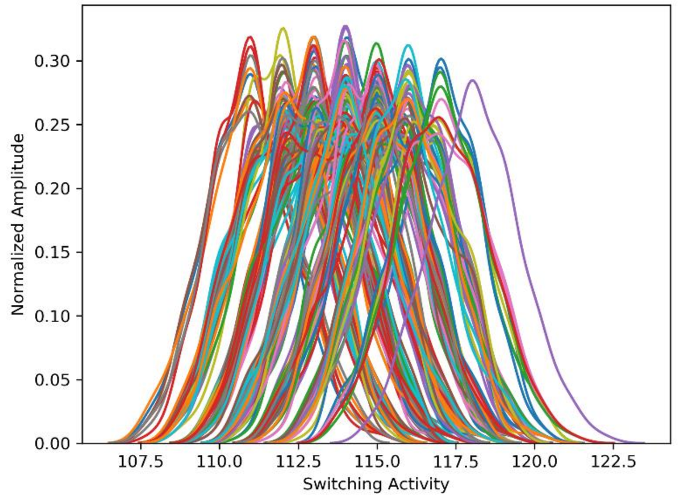


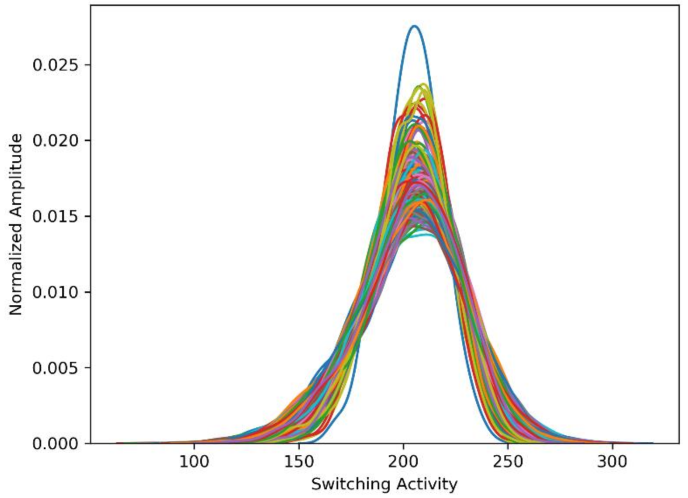
AES-LUT Design: Vulnerable module identification

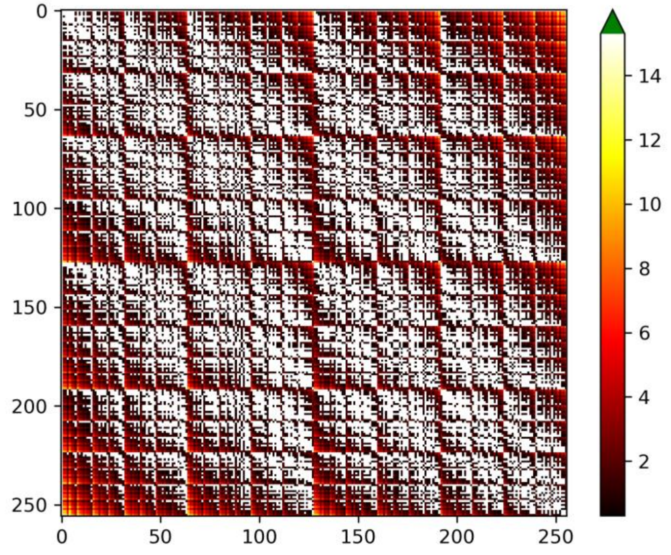
- It can be seen that framework is able to identify the Sbox and mix column modules as leaky.

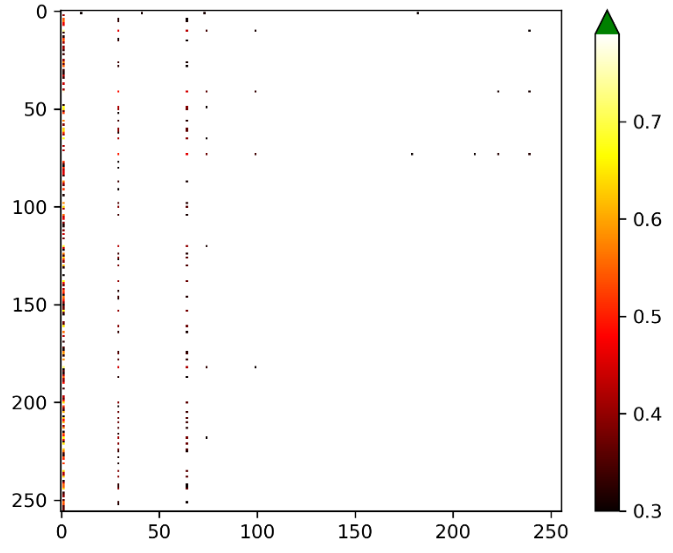
PSC Improvement at Module-Level

- Unprotected LUT-Sbox Module is replaced with threshold implementation of Sbox.
- Top-Left: Power distribution functions for different subkeys in the normal AES Sbox Module.
- Top-Right: Power distribution functions for different subkeys in the Sbox with TI implementation.
- Bottom-Left: KL divergence between every possible subkey pair for unprotected Sbox.
- Bottom-Right: KL divergence between every possible subkey pair for Sbox-TI.





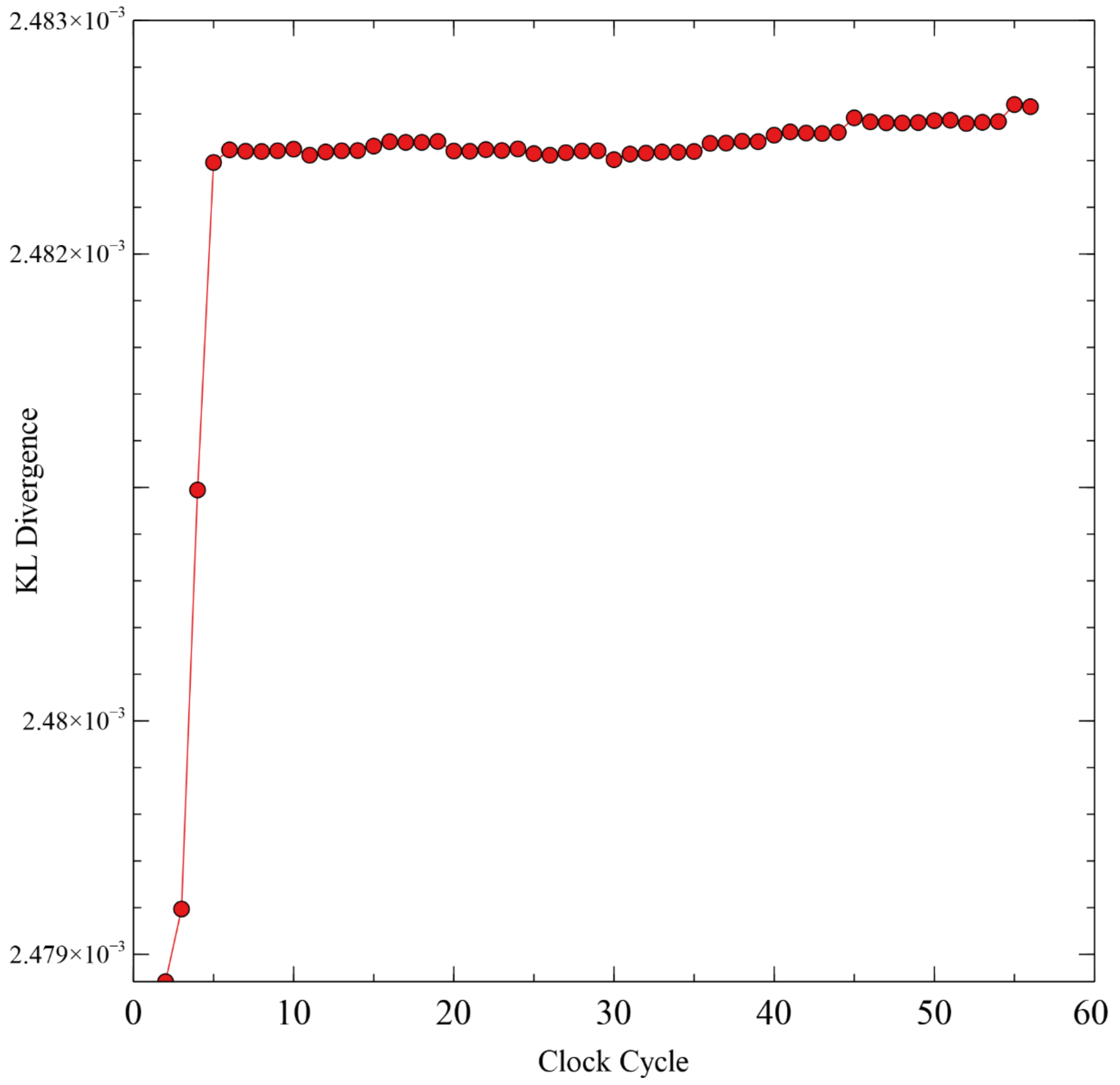




White spaces reflect the KL divergence for the particular keypair is less than 0.3

PSC Improvement at Design-level

- The unprotected Sbox is replaced with Sbox-TI.
- The generation of random bytes in the design also lead to additional switching. Thus, reducing SNR.
- Figure shows the total power KL divergence between key pairs all 0's vs all F's.
- Since KL divergence is less than 0.03 it can be fairly assumed that it will challenging for an attacker to distinguish between the key pairs.

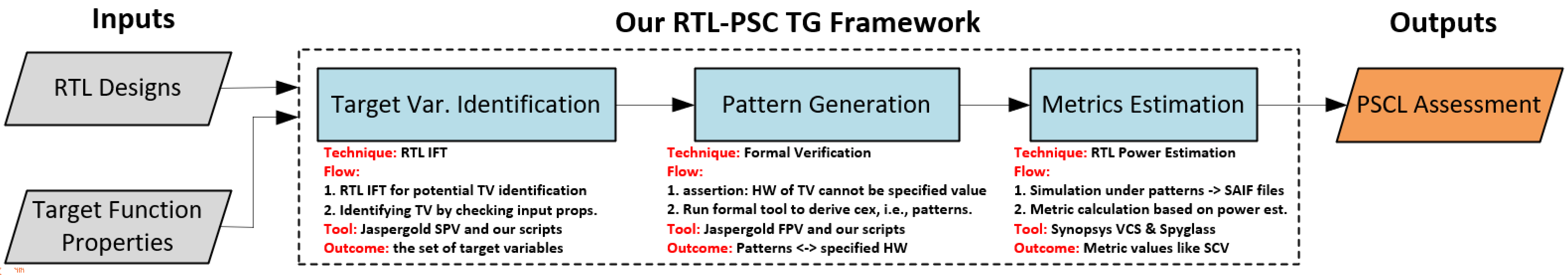


Overview: PSC-TG

- PSC-TG framework
  - Goal: Deriving few patterns to cause worst-case scenario in terms of PSCL for the target design and calculate metrics for PSCL assessment
  - Overview
    - Target properties definition
    - RTL information flow tracking
    - Pattern generation
    - Metrics calculation for non-masked implementations -> SCV metric

Target Function Properties

Secret •Key of encryption operation	**Controllability** •Plaintext of encryption operation
Confusion •One output bit depends on multiple key bits	**Divide-and-conquer** •Depends on a subset of key bits



Technique: RTL IFT
Flow:
1. RTL IFT for potential TV identification
2. Identifying TV by checking input props.
Tool: Jaspergold SPV and our scripts
Outcome: the set of target variables

Technique: Formal Verification
Flow:
1. assertion: HW of TV cannot be specified value
2. Run formal tool to derive cex, i.e., patterns.
Tool: Jaspergold FPV and our scripts
Outcome: Patterns <-> specified HW

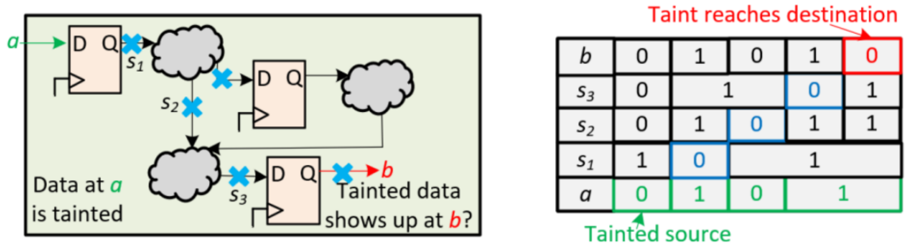
Technique: RTL Power Estimation
Flow:
1. Simulation under patterns -> SAIF files
2. Metric calculation based on power est.
Tool: Synopsys VCS & Spyglass
Outcome: Metric values like SCV

Target Variable Identification

- Target variables identification
  - Aim: identify the variables which satisfy all four properties
  - Method: RTL IFT with Jaspergold SPV
    - Starting from key bits -> check tainted variables cycle-by-cycle -> potential target variables
    - If other 3 properties are also satisfied -> target variables

Jaspergold_SPV(*a*, *b*, *r*): failed -> information flow **exists** between *a* and *b*

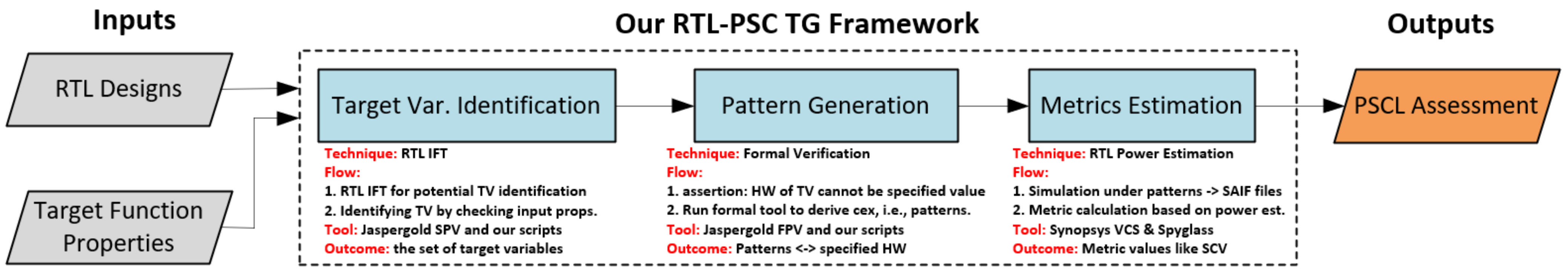
check_spv -create from {*a*} -from_precond {round == 0} -to {*b*} -to_precond {round == *r*}



b	0	1	0	1	0
s_3	0	1		0	1
s_2	0	1	0	1	1
s_1	1	0	1		
a	0	1	0	1	

Target Function Properties

Secret •Key of encryption operation	**Controllability** •Plaintext of encryption operation
Confusion •One output bit depends on multiple key bits	**Divide-and-conquer** •Depends on a subset of key bits



Technique: RTL IFT
Flow:
1. RTL IFT for potential TV identification
2. Identifying TV by checking input props.
Tool: Jaspergold SPV and our scripts
Outcome: the set of target variables

Technique: Formal Verification
Flow:
1. assertion: HW of TV cannot be specified value
2. Run formal tool to derive cex, i.e., patterns.
Tool: Jaspergold FPV and our scripts
Outcome: Patterns <-> specified HW

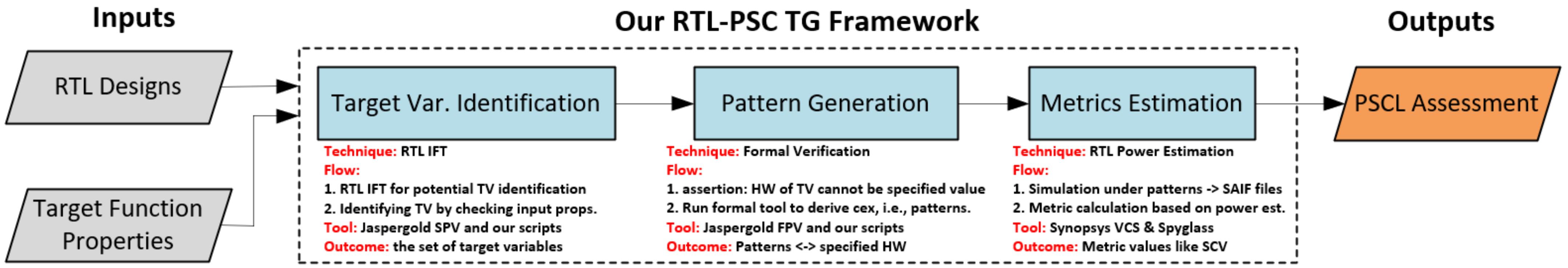
Technique: RTL Power Estimation
Flow:
1. Simulation under patterns -> SAIF files
2. Metric calculation based on power est.
Tool: Synopsys VCS & Spyglass
Outcome: Metric values like SCV

Pattern Generation

- Pattern generation
  - Aim: derive the patterns which can cause worst-case scenario
  - Method: Formal verification with Jaspergold FPV
    - HW model: HW(TVs) at N_{th} round != specified value
    - HD model: HD(TVs) at N_{th} and $(N\text{-}1)_{th}$ round != specified value
    - Derived patterns will be reported in the counterexample

Algorithm 2: Pattern Generation Procedure

```
Input: D - The RTL design with key K and plaintext X
Input: $TV_R$ - Identified target variables at round $R$
Input: $N$ - Specified value for HW/HD model
Output: $P$ - The derived test pattern
1  Load the RTL designs D
2  Specify the top module, clock and reset
3  Apply the input constraint $K \leftarrow 0$
4  Apply the constant constraint to X
5  Run reset analysis
6  Assertion: $HW(TV_R) \neq N$ or $HW(TV_{R-1} \oplus TV_R) \neq N$
7  Prove
8  Extract $P$ from the counter-example
```



Technique: RTL IFT
Flow:
1. RTL IFT for potential TV identification
2. Identifying TV by checking input props.
Tool: Jaspergold SPV and our scripts
Outcome: the set of target variables

Technique: Formal Verification
Flow:
1. assertion: HW of TV cannot be specified value
2. Run formal tool to derive cex, i.e., patterns.
Tool: Jaspergold FPV and our scripts
Outcome: Patterns <-> specified HW

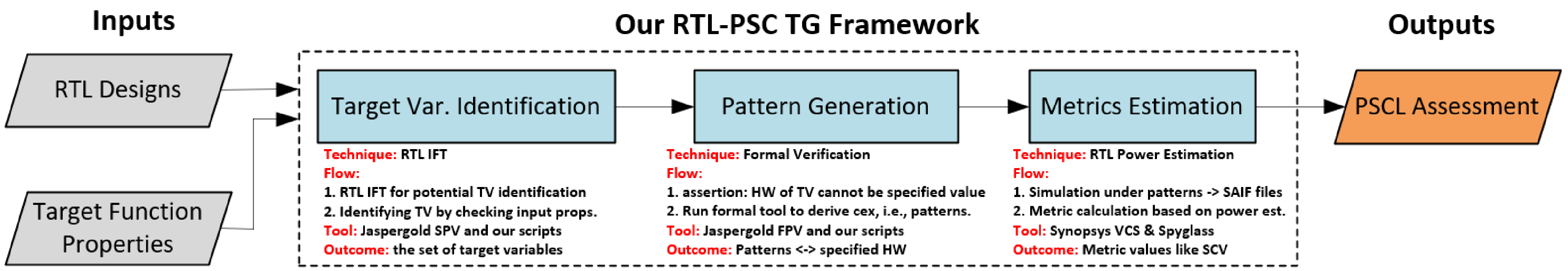
Technique: RTL Power Estimation
Flow:
1. Simulation under patterns -> SAIF files
2. Metric calculation based on power est.
Tool: Synopsys VCS & Spyglass
Outcome: Metric values like SCV

SCV Metric Calculation

- SCV metric calculation for non-masked design
  - Signal-noise-ratio (SNR) is popular at post-silicon assessment
  - Similar metric side-channel vulnerability (SCV) is proposed at pre-silicon assessment

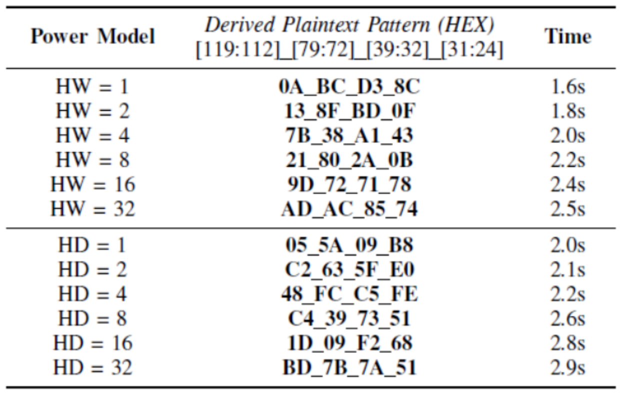
$$SCV = \frac{P_{signal}}{P_{noise}} = \frac{P_{T.hi} - P_{T.hj}}{P_{noise}}$$

  - Derived patterns are used during simulation with Synopsys VCS
  - The generated SAIF files are fed to Synopsys Spyglass for power estimation

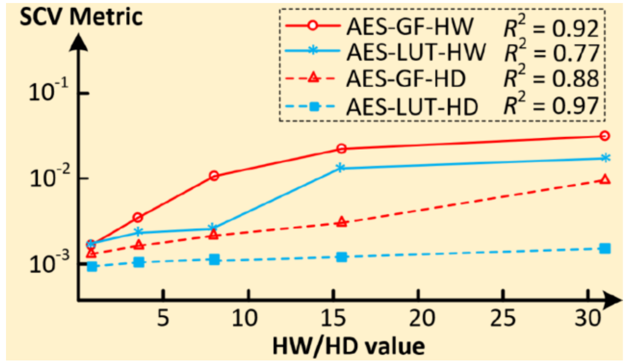

Inputs
Our RTL-PSC TG Framework
Outputs
RTL Designs
Target Function Properties
Target Var. Identification
Pattern Generation
Metrics Estimation
PSCL Assessment
Technique: RTL IFT
Flow:
1. RTL IFT for potential TV identification
2. Identifying TV by checking input props.
Tool: Jaspergold SPV and our scripts
Outcome: the set of target variables
Technique: Formal Verification
Flow:
1. assertion: HW of TV cannot be specified value
2. Run formal tool to derive cex, i.e., patterns.
Tool: Jaspergold FPV and our scripts
Outcome: Patterns <-> specified HW
Technique: RTL Power Estimation
Flow:
1. Simulation under patterns -> SAIF files
2. Metric calculation based on power est.
Tool: Synopsys VCS & Spyglass
Outcome: Metric values like SCV

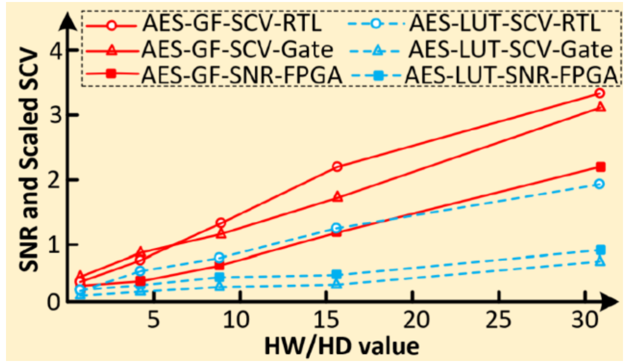
Experimental Results

- SCV calculation
  - Evaluation under both HW and HD models
  - Positive correlation between SCV metric and specified HW/HD value
  - R^2 values indicate the great linearity between SCV and HW/HD
  - AES-GF > AES-LUT in terms of PSCL -> consistent w/ previous results
- SCV validation at gate-level and post-silicon (FPGA) levels
  - Pearson correlation coefficient
  - Gate-level: Xilinx Vivado
    - AES-GF = 0.986
    - AES-LUT = 0.967
  - FPGA level: Xilinx Spartan-6 with Tektronix MDO3102 oscilloscope
    - Design running at 24 MHz
    - AES-GF = 0.985
    - AES-LUT = 0.924



Power Model	*Derived Plaintext Pattern (HEX)* [119:112]_[79:72]_[39:32]_[31:24]	**Time**
HW = 1	**0A_BC_D3_8C**	1.6s
HW = 2	**13_8F_BD_0F**	1.8s
HW = 4	**7B_38_A1_43**	2.0s
HW = 8	**21_80_2A_0B**	2.2s
HW = 16	**9D_72_71_78**	2.4s
HW = 32	**AD_AC_85_74**	2.5s
HD = 1	**05_5A_09_B8**	2.0s
HD = 2	**C2_63_5F_E0**	2.1s
HD = 4	**48_FC_C5_FE**	2.2s
HD = 8	**C4_39_73_51**	2.6s
HD = 16	**1D_09_F2_68**	2.8s
HD = 32	**BD_7B_7A_51**	2.9s





Demo Video

CAD for Security

C/C++

- Information Leakage
- Control flow violations
- Side channel leakage

RTL

- Susceptibility to Trojan Insertion
- Test Generation for Trust Verification
- Power Side-channel Leakage Assessment (RTL-PSC)
- PSC Test Generation (TG)
- EM Leakage Assessment
- Formal Verification of Security Properties

Gate

- Susceptibility of Fault Injection (AVFSM)
- Automated Mapping of Security Properties from C to Gate Level (AutoMap)
- Information Flow Security (IFS)
- Power Side-channel Leakage Assessment (SCRIPT)
- Fault Injection Assessment

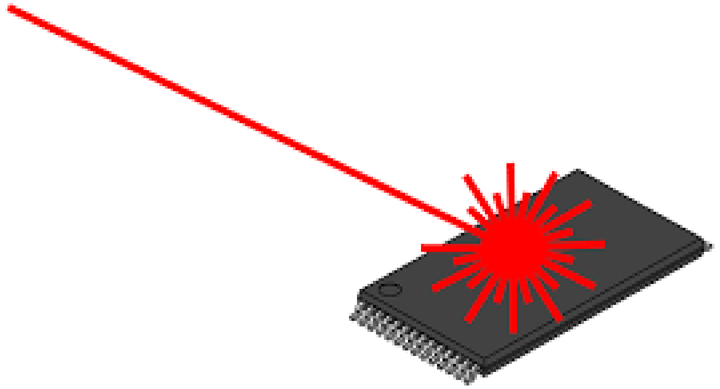
Layout

- Susceptibility of Probing Attacks
- Power Side-channel Leakage Assessment (TVLA)

Vulnerability Analysis of FSM

- Finite State Machine → controls overall functionality of most digital systems

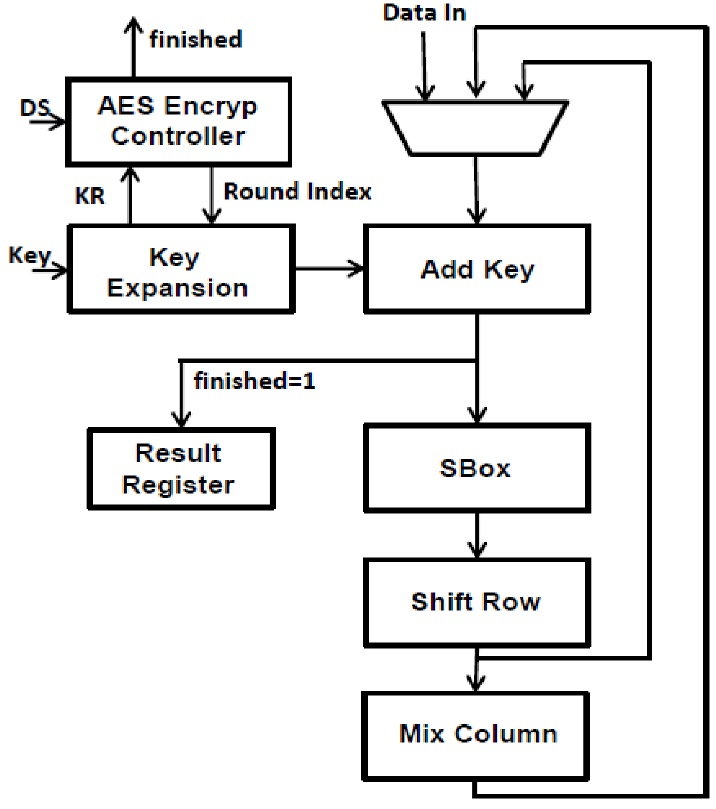
- **Attacks on FSM**
  - **Fault Injection Attack:** Inject a fault to cause transition to a protected state from an unauthorized state



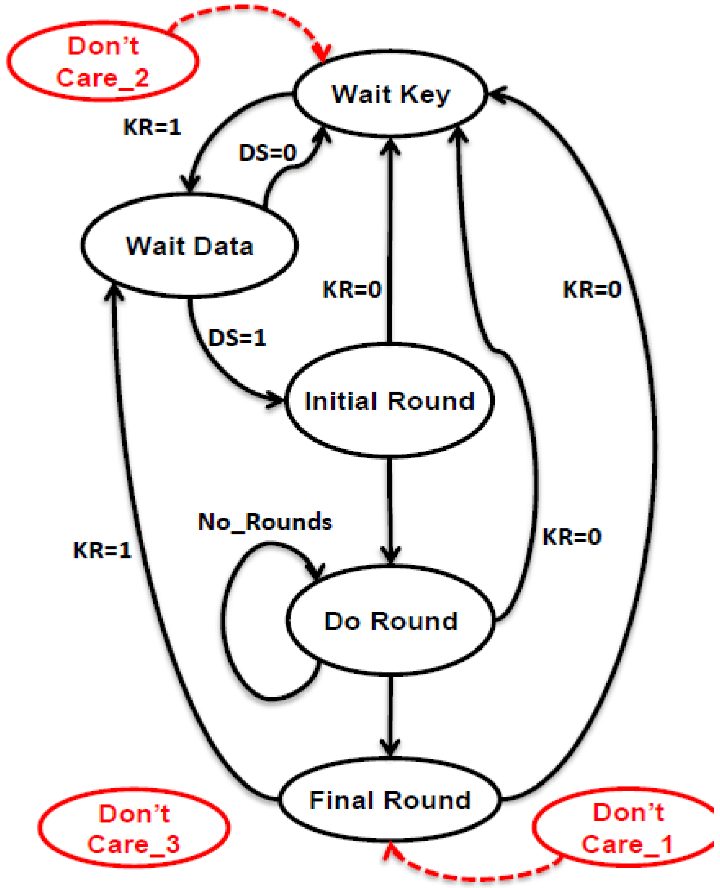
- **Sources of Vulnerabilities**
  - Synthesis tools introduce **don't-care states and transitions** → facilitate fault and Trojan based attacks
  - **Encoding scheme** and **design constraints** → create unintentional vulnerabilities in FSM

Example: AES Encryption

Attacker's objective:
Bypass the intermediate rounds and go directly to the Final Round.



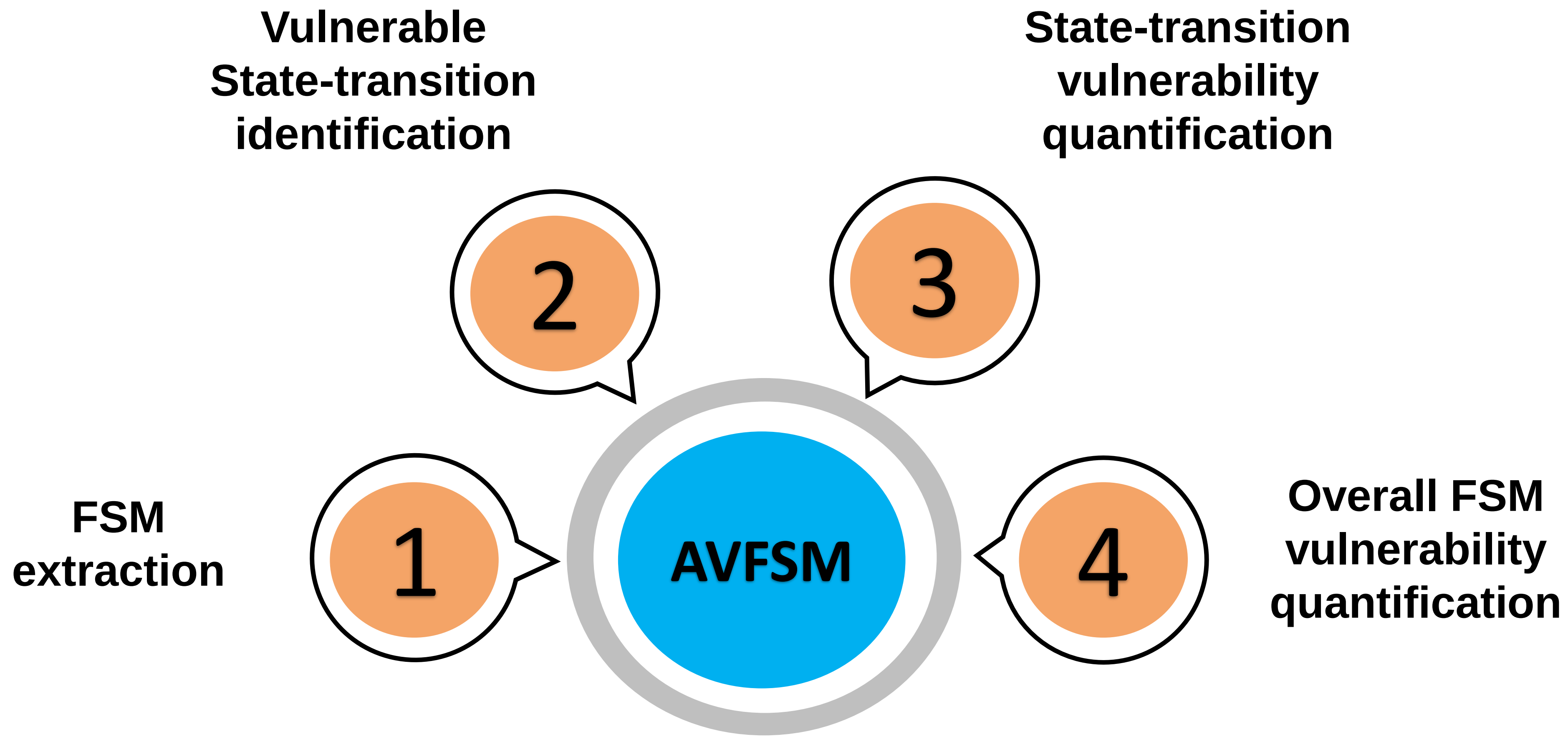
(a)



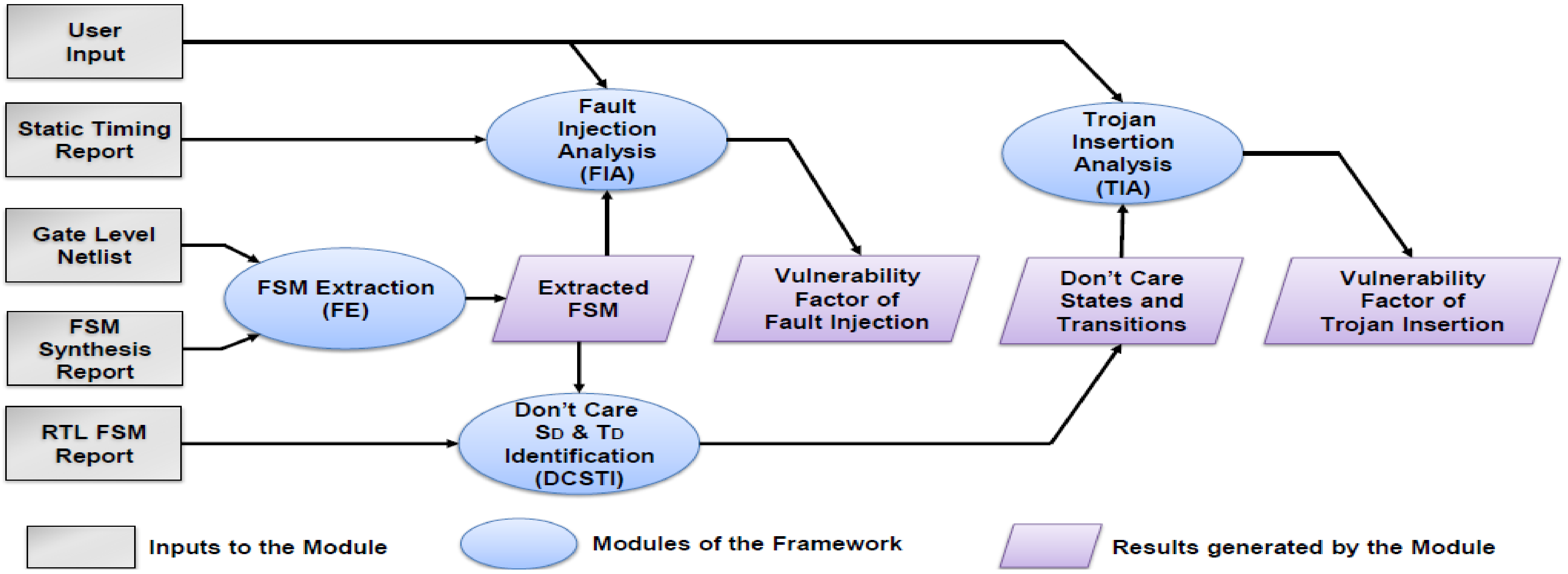
(b)

Source: Datasheeet AES 128/192/256 (ECB) AVALON

AVFSM (Analyzing Vulnerabilities in FSM)


Vulnerable State-transition identification
State-transition vulnerability quantification
2
3
FSM extraction
1
AVFSM
4
Overall FSM vulnerability quantification

AVFSM Framework



User Input
Static Timing Report
Gate Level Netlist
FSM Synthesis Report
RTL FSM Report
FSM Extraction (FE)
Fault Injection Analysis (FIA)
Extracted FSM
Vulnerability Factor of Fault Injection
Trojan Insertion Analysis (TIA)
Don't Care States and Transitions
Vulnerability Factor of Trojan Insertion
Don't Care SD & TD Identification (DCSTI)
Inputs to the Module
Modules of the Framework
Results generated by the Module

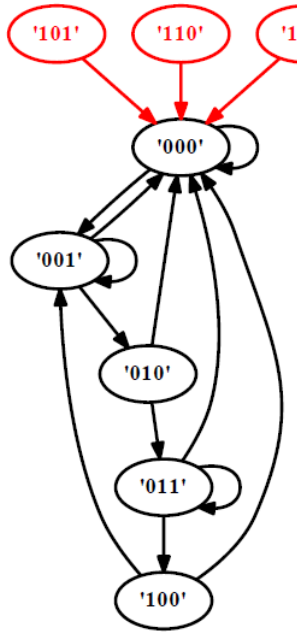
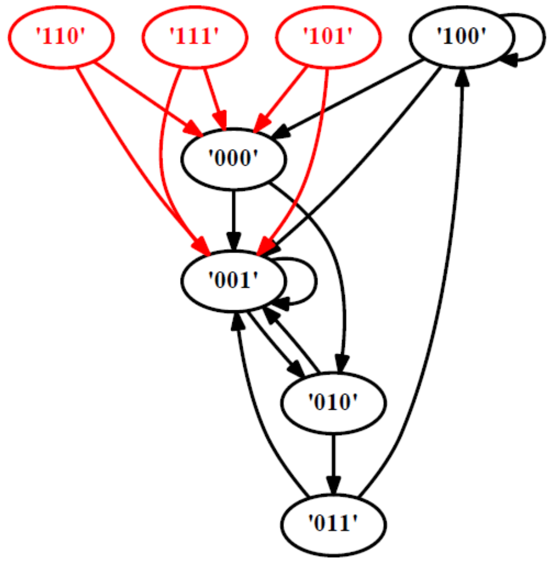
Fault Injection Vulnerability Metric

$$VF_{FI} = \{PVT(\%), ASF\}$$

$$PVT(\%) = \frac{Total_{Vulnerable\_Transition}(N_{VT})}{Total_{Transition}},\ ASF = \frac{\sum_{i=1}^{N_{VT}} SF(i)}{N_{VT}}$$

- **Rule**: For Secure FSM VF_{FI} should be zero (or minimized)
- **ASF** is a measure of susceptibility to fault attack
- **PVT(%)** indicates percentage of total transitions vulnerable to fault injection attack

Impact of Encoding Schemes

Encoding Scheme 1	Encoding Scheme 2
 '101' '110' '111' '000' '001' '010' '011' '100' State Encodings and Order: WAIT_KEY : 000 WAIT_DATA : 001 INITIAL_ROUND : 010 DO_ROUND : 011 FINAL_ROUND : 100 Area: 2226.7	 '110' '111' '101' '100' '000' '001' '010' '011' State Encodings and Order: FINAL_ROUND : 000 WAIT_KEY : 001 WAIT_DATA : 010 INITIAL_ROUND : 011 DO_ROUND : 100 Area: 2038.5

- **Takeaway**
  - **State encodings** impacts the vulnerabilities of a FSM

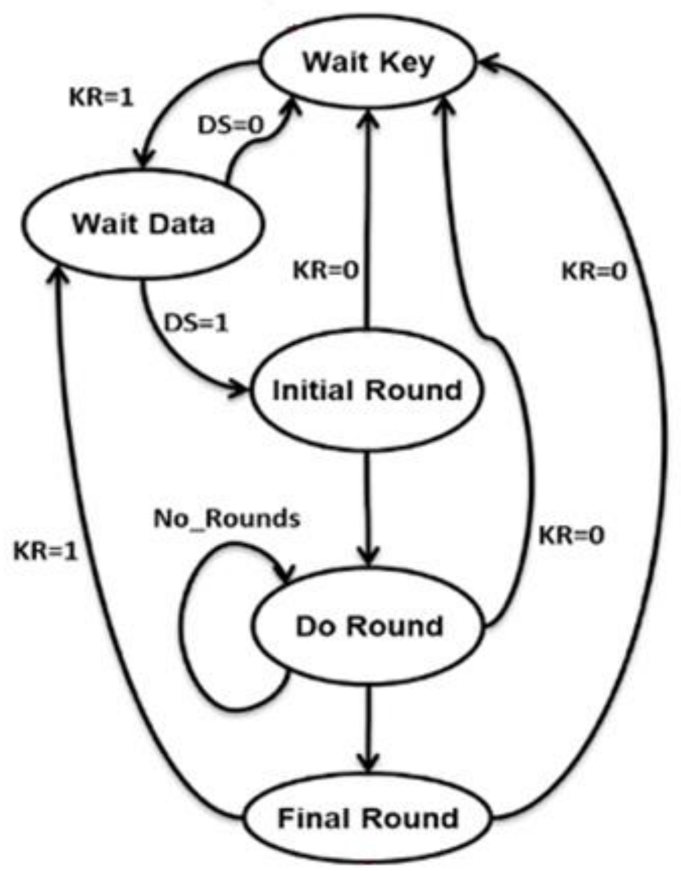
Vulnerability analysis of AES

	scheme 1	scheme 2
VF_{FI}	(0,0)	(58.9%,0.15)

FICS Research

Case Study

- **Design:** AES controller's FSM
- **Abstraction level:** Gate-level netlist
- **State Encoding**:
  - WAIT_KEY : 001
  - WAIT_DATA : 010
  - INITIAL_ROUND : 011
  - DO_ROUND : 100
  - FINAL_ROUND : 000
- **Protected State**: 000



(b)

Figure: Finite-state machine of AES controller

Demo Video

CAD for Security

C/C++

- Information Leakage
- Control flow violations
- Side channel leakage

RTL

- Susceptibility to Trojan Insertion
- Test Generation for Trust Verification
- Power Side-channel Leakage Assessment (RTL-PSC)
- PSC Test Generation (TG)
- EM Leakage Assessment
- Formal Verification of Security Properties

Gate

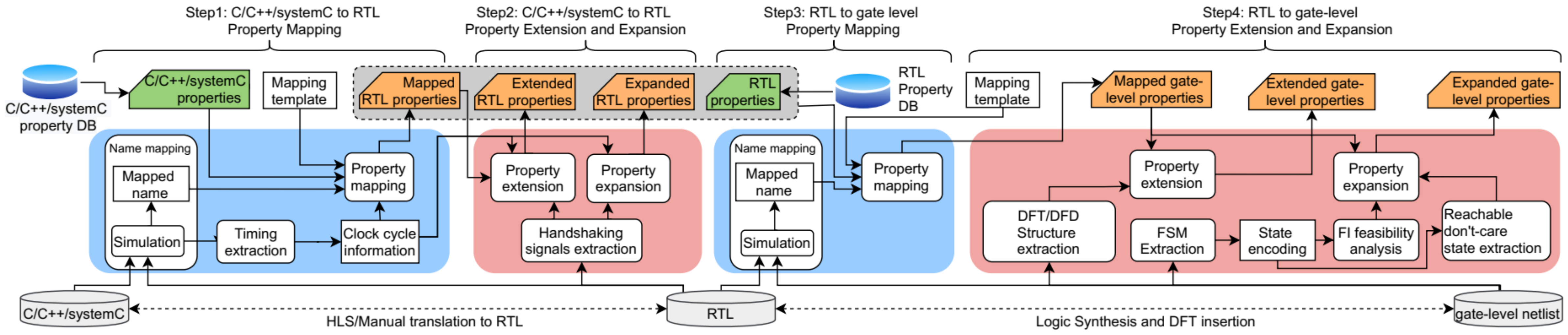
- Susceptibility of Fault Injection (AVFSM)
- Automated Mapping of Security Properties from C to Gate Level (AutoMap)
- Information Flow Security (IFS)
- Power Side-channel Leakage Assessment (SCRIPT)
- Fault Injection Assessment

Layout

- Susceptibility of Probing Attacks
- Power Side-channel Leakage Assessment (TVLA)

High-Level Overview

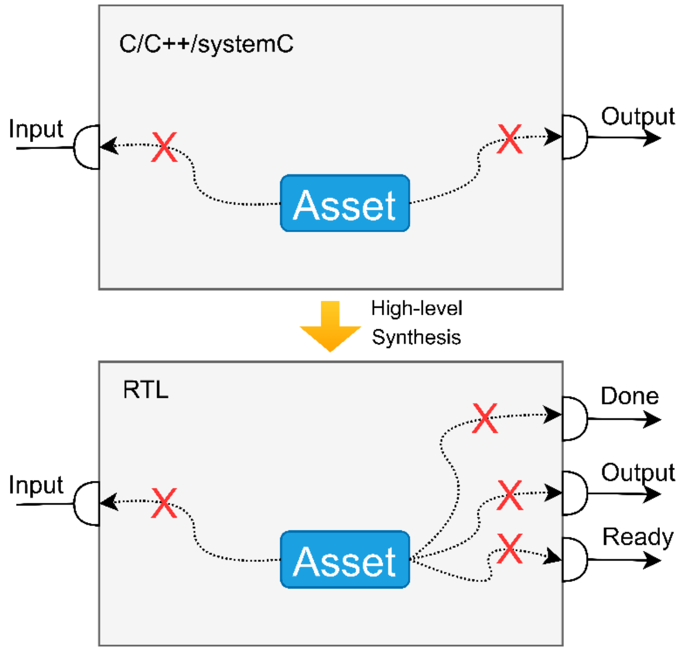
- **Tool Name:** Automated Security Property mapping tool
- **Goal:**
  - ***Mapping*** *Security Property between design abstraction levels (C to RTL to GATE)*
  - ***Extension*** *of the security properties*
  - ***Expansion*** *of the security properties*



Step1: C/C++/systemC to RTL Property Mapping
Step2: C/C++/systemC to RTL Property Extension and Expansion
Step3: RTL to gate level Property Mapping
Step4: RTL to gate-level Property Extension and Expansion
C/C++/systemC property DB
C/C++/systemC properties
Mapping template
Mapped RTL properties
Extended RTL properties
Expanded RTL properties
RTL properties
RTL Property DB
Mapping template
Mapped gate-level properties
Extended gate-level properties
Expanded gate-level properties
Name mapping
Mapped name
Simulation
Timing extraction
Clock cycle information
Property mapping
Property extension
Property expansion
Handshaking signals extraction
Name mapping
Mapped name
Simulation
Property mapping
DFT/DFD Structure extraction
Property extension
FSM Extraction
State encoding
FI feasibility analysis
Property expansion
Reachable don't-care state extraction
C/C++/systemC
HLS/Manual translation to RTL
RTL
Logic Synthesis and DFT insertion
gate-level netlist

Important Terms

- **Security Property Mapping:** Translation of one abstraction level's security property to another.
- **Security Property Extension:** Extension of the argument list of a security property.



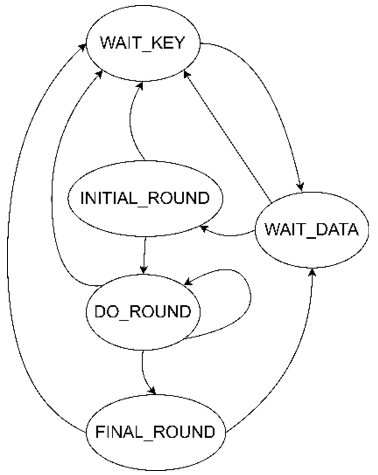
Property:
Asset should not leak to ***Input*** *and* ***output***

Extended Property
Asset should not leak to ***Input****,* ***output,*** *and* ***Ready***

Important Terms (Cont.)

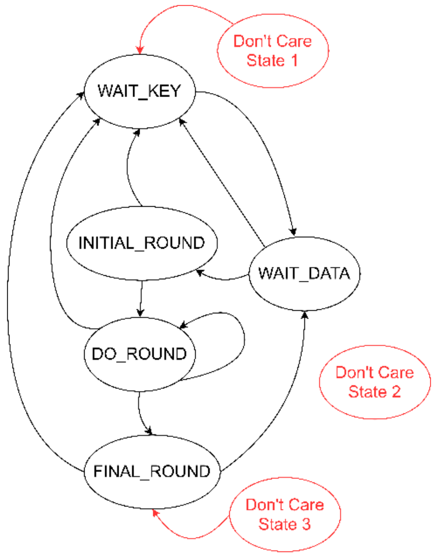
- **Security Property Expansion:** Additional security property created to check new vulnerability that violates a specific security goal.

RTL



Property:
FINAL_ROUND should be accessed only from DO_ROUND

Gate

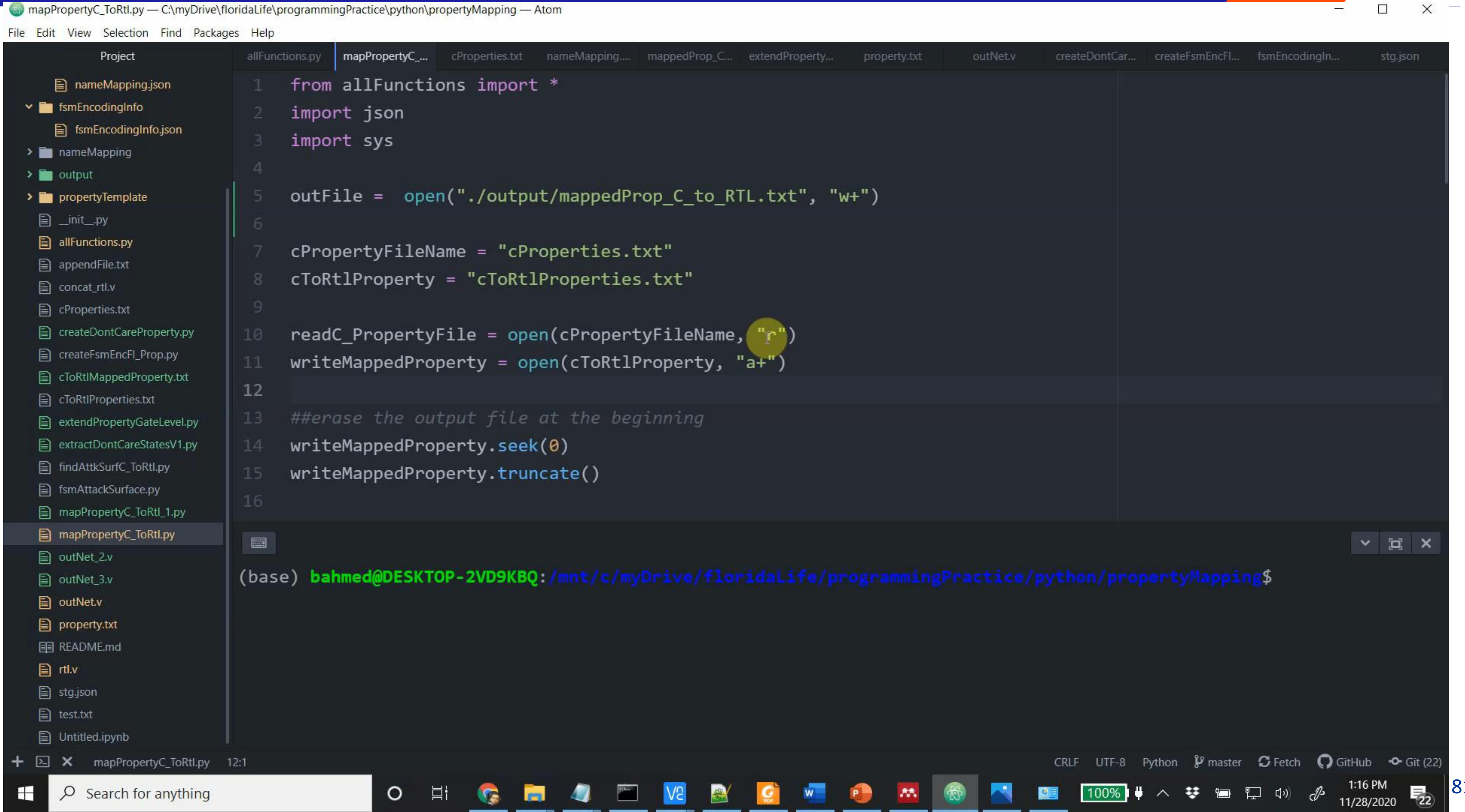


Expanded Property:
FINAL_ROUND should not be connected to any don’t care states

Case Study

- Design: AES
- Abstraction level: C, RTL, Gate
- Input in C-level:
  - C Design
  - C properties
  - RTL design
- Input in RTL:
  - RTL design
  - RTL Property
  - Gate-level netlist.
  - DFT structure.
- Output:
  - Mapped, extended, and expanded properties

Demo Video



mapPropertyC_ToRtl.py — C:\myDrive\floridaLife\programmingPractice\python\propertyMapping — Atom

```python
from allFunctions import *
import json
import sys

outFile =  open("./output/mappedProp_C_to_RTL.txt", "w+")

cPropertyFileName = "cProperties.txt"
cToRtlProperty = "cToRtlProperties.txt"

readC_PropertyFile = open(cPropertyFileName, "r")
writeMappedProperty = open(cToRtlProperty, "a+")

##erase the output file at the beginning
writeMappedProperty.seek(0)
writeMappedProperty.truncate()

```

```
(base) bahmed@DESKTOP-2VD9KBQ:/mnt/c/myDrive/floridaLife/programmingPractice/python/propertyMapping$
```

CAD for Security

C/C++

- Information Leakage
- Control flow violations
- Side channel leakage

RTL

- Susceptibility to Trojan Insertion
- Test Generation for Trust Verification
- Power Side-channel Leakage Assessment (RTL-PSC)
- PSC Test Generation (TG)
- EM Leakage Assessment
- Formal Verification of Security Properties

Gate

- Susceptibility of Fault Injection (AVFSM)
- Automated Mapping of Security Properties from C to Gate Level (AutoMap)
- Information Flow Security (IFS)
- Power Side-channel Leakage Assessment (SCRIPT)
- Fault Injection Assessment

Layout

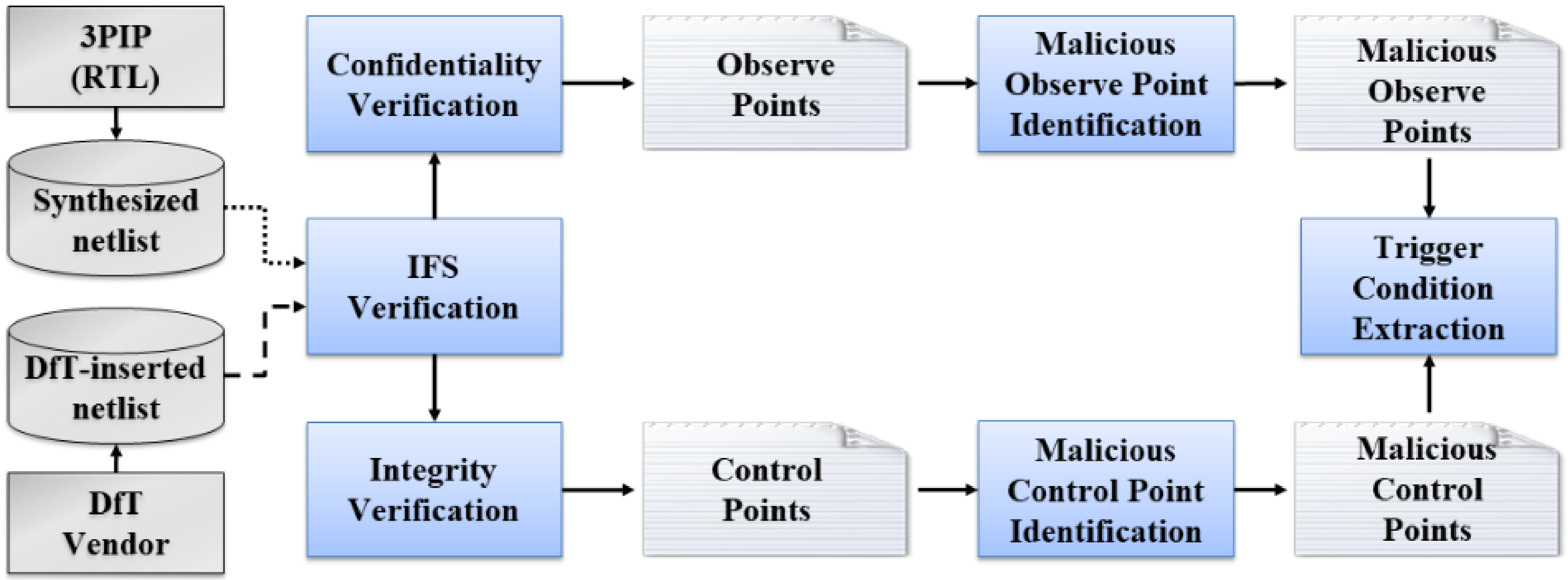
- Susceptibility of Probing Attacks
- Power Side-channel Leakage Assessment (TVLA)

High Level Overview

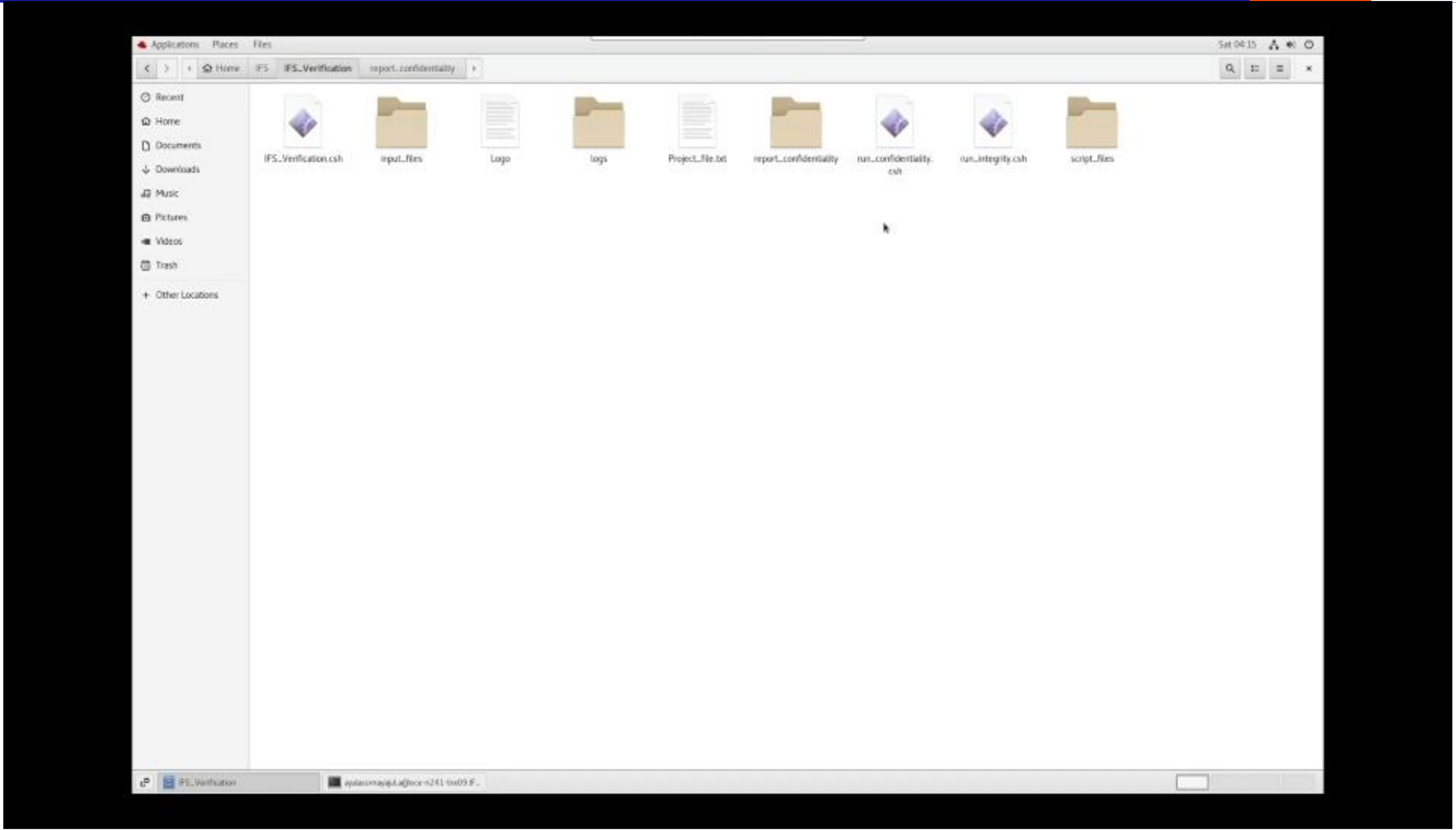
- **Tool Name**: **I**nformation **F**low **S**ecurity Verification Tool (**IFS**).
- **Goal:** To detect the presence of Trojans in a design using Information Flow Security (IFS) by checking for any violation of information flow policies.
- **Description:**
  - Our tool tries to leverage the violations of '**confidentiality'** or '**integrity'** of information flow by a Trojan.
  - By leveraging fault models such as stuck-at-faults and Automatic Test Pattern Generation (ATPG) tools, we can detect malicious change that cause CI violations.
  - The trigger condition for the Trojans can also be extracted using the IFS tool.

Framework

- **Input**: Design Library (.v), synthesized netlist (.v) and test protocol (.spf)
- **Output**: Confidentiality Report, Integrity Report.
- **CAD Tool Used:** Synopsys TetraMax

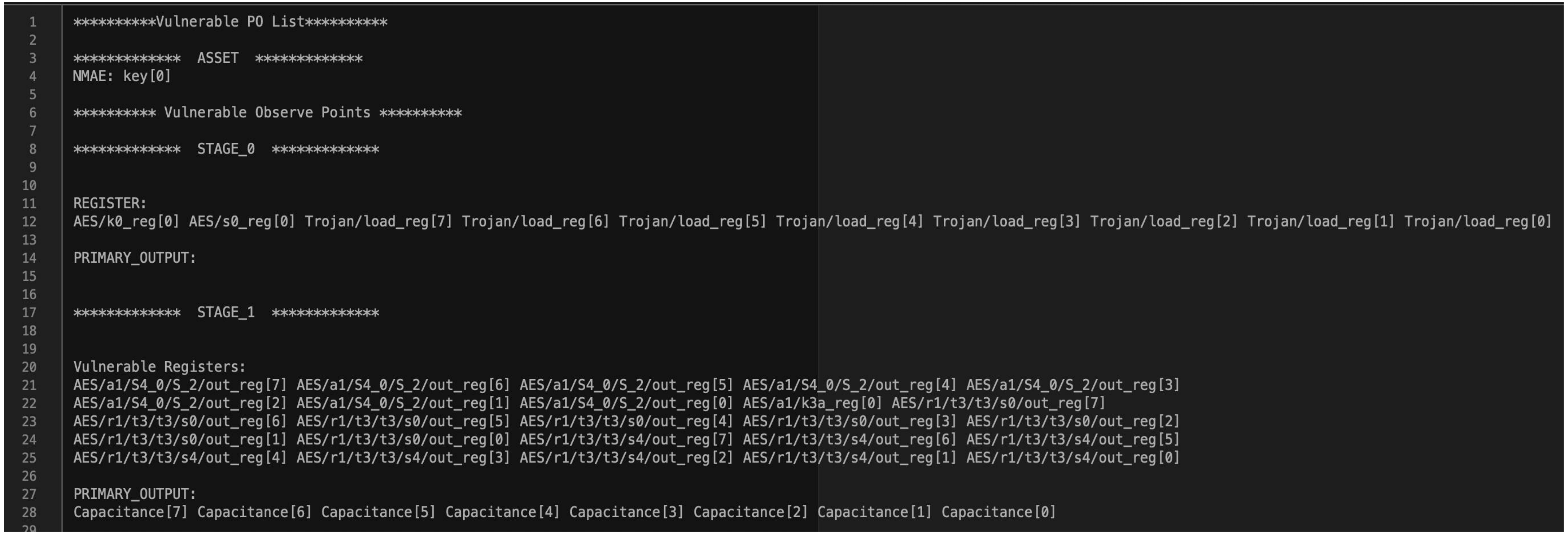


Demo Video


Applications
Places
Files
Sat 04:15
Home
IFS
IFS_Verification
report_confidentiality
Recent
Home
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
IFS_Verification.csh
input_files
Logo
logs
Project_file.txt
report_confidentiality
run_confidentiality.csh
run_integrity.csh
script_files
IFS_Verification

Results

- We have performed a **“confidentiality”** verification on an AES-T100 Trojan benchmark which has an always-on Trojan.
- Too detected different malicious observe points to which our asset **(key[0])** leaks to.



```
**********Vulnerable PO List**********

*************  ASSET  *************
NMAE: key[0]

********** Vulnerable Observe Points **********

*************  STAGE_0  *************


REGISTER:
AES/k0_reg[0] AES/s0_reg[0] Trojan/load_reg[7] Trojan/load_reg[6] Trojan/load_reg[5] Trojan/load_reg[4] Trojan/load_reg[3] Trojan/load_reg[2] Trojan/load_reg[1] Trojan/load_reg[0]

PRIMARY_OUTPUT:


*************  STAGE_1  *************


Vulnerable Registers:
AES/a1/S4_0/S_2/out_reg[7] AES/a1/S4_0/S_2/out_reg[6] AES/a1/S4_0/S_2/out_reg[5] AES/a1/S4_0/S_2/out_reg[4] AES/a1/S4_0/S_2/out_reg[3]
AES/a1/S4_0/S_2/out_reg[2] AES/a1/S4_0/S_2/out_reg[1] AES/a1/S4_0/S_2/out_reg[0] AES/a1/k3a_reg[0] AES/r1/t3/t3/s0/out_reg[7]
AES/r1/t3/t3/s0/out_reg[6] AES/r1/t3/t3/s0/out_reg[5] AES/r1/t3/t3/s0/out_reg[4] AES/r1/t3/t3/s0/out_reg[3] AES/r1/t3/t3/s0/out_reg[2]
AES/r1/t3/t3/s0/out_reg[1] AES/r1/t3/t3/s0/out_reg[0] AES/r1/t3/t3/s4/out_reg[7] AES/r1/t3/t3/s4/out_reg[6] AES/r1/t3/t3/s4/out_reg[5]
AES/r1/t3/t3/s4/out_reg[4] AES/r1/t3/t3/s4/out_reg[3] AES/r1/t3/t3/s4/out_reg[2] AES/r1/t3/t3/s4/out_reg[1] AES/r1/t3/t3/s4/out_reg[0]

PRIMARY_OUTPUT:
Capacitance[7] Capacitance[6] Capacitance[5] Capacitance[4] Capacitance[3] Capacitance[2] Capacitance[1] Capacitance[0]
```

CAD for Security

C/C++

- Information Leakage
- Control flow violations
- Side channel leakage

RTL

- Susceptibility to Trojan Insertion
- Test Generation for Trust Verification
- Power Side-channel Leakage Assessment (RTL-PSC)
- PSC Test Generation (TG)
- EM Leakage Assessment
- Formal Verification of Security Properties

Gate

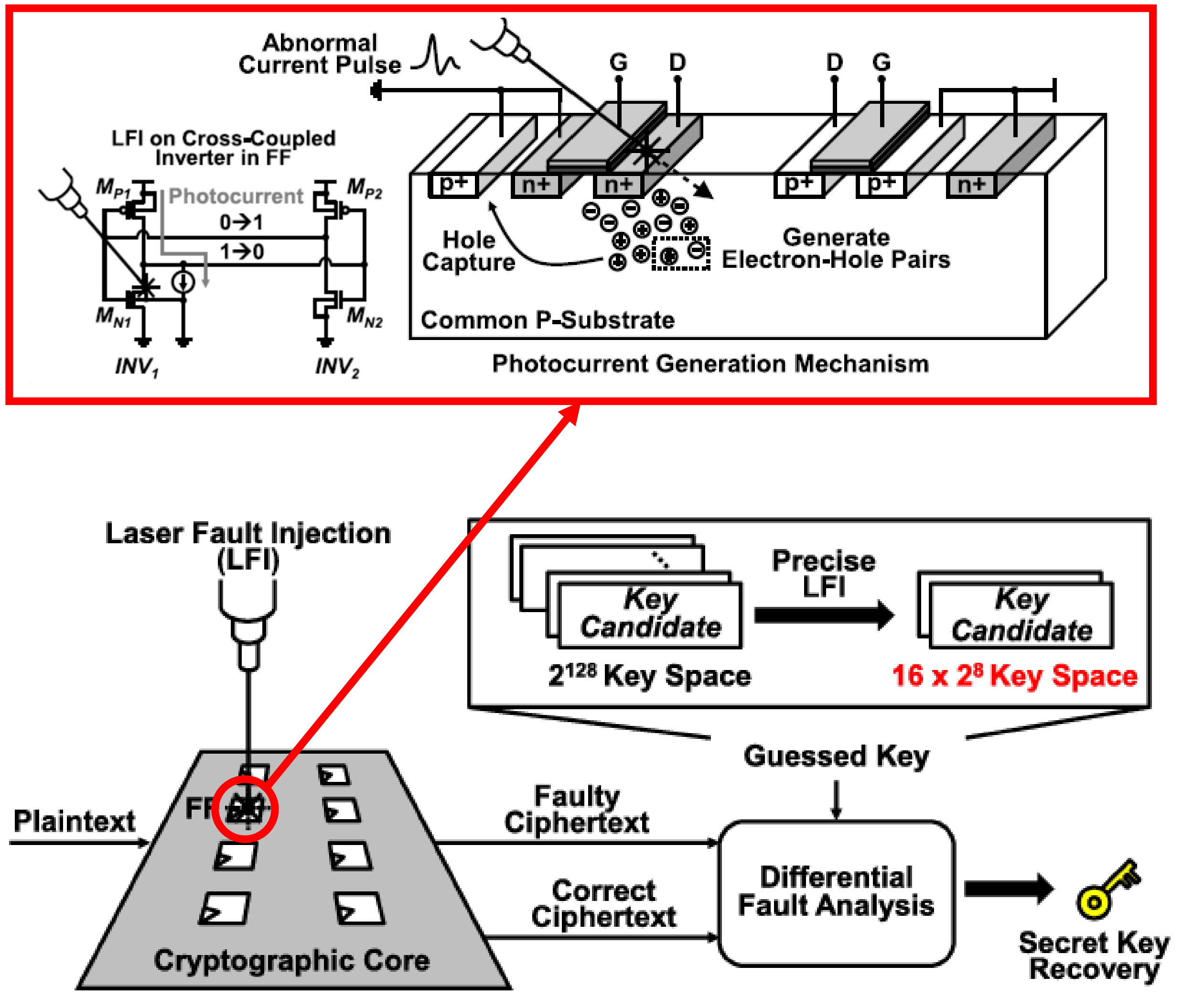
- Susceptibility of Fault Injection (AVFSM)
- Automated Mapping of Security Properties from C to Gate Level (AutoMap)
- Information Flow Security (IFS)
- Power Side-channel Leakage Assessment (SCRIPT)
- Fault Injection Assessment

Layout

- Susceptibility of Probing Attacks
- Power Side-channel Leakage Assessment (TVLA)

Motivation

- Fault injection is a powerful attack to tamper with the device and extract secrets
- Current countermeasures, e.g., hardware/time redundancy, may involve 100% area or timing overhead
- Lack of research in assessing the design's vulnerability to fault-injection attacks at an early stage (gate-level)
- No automated framework to perform such assessment

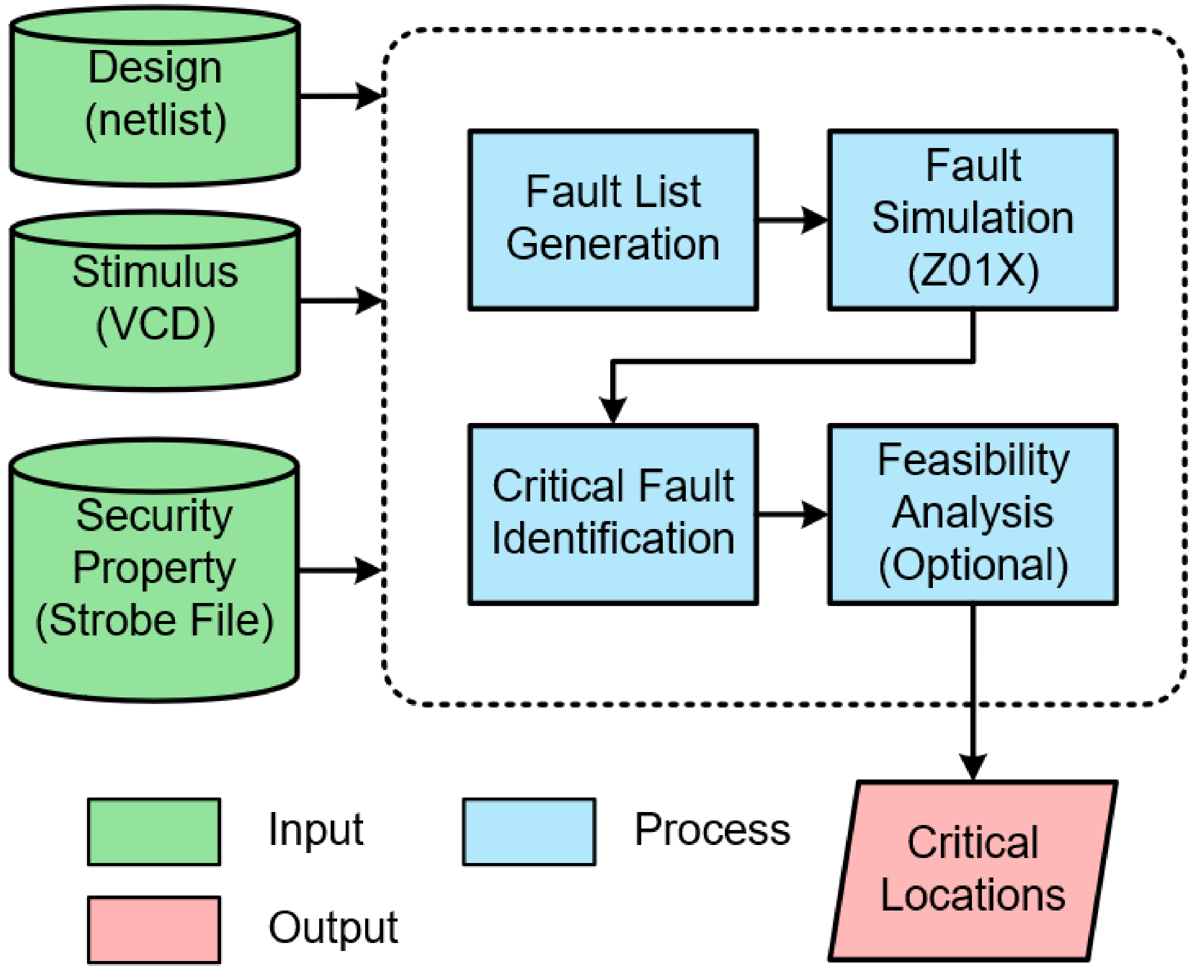


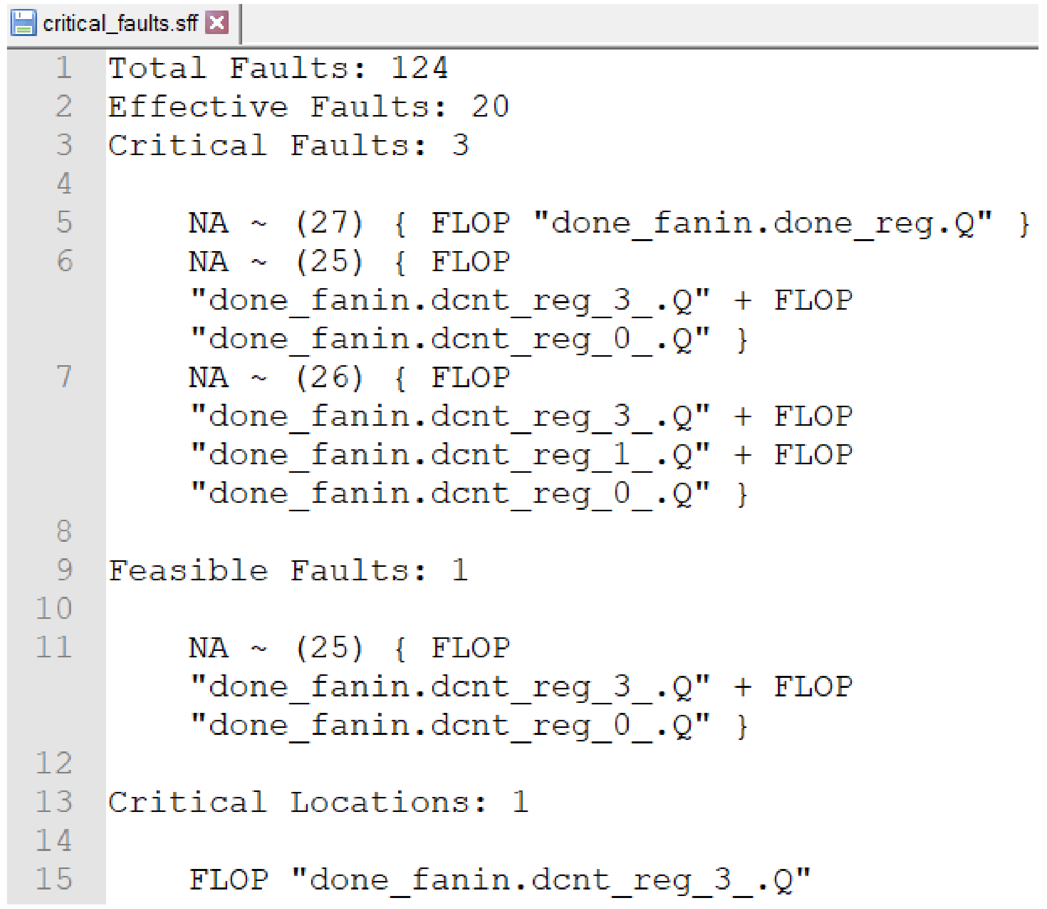
K. Matsuda *et al*., "A 286 F2/Cell Distributed Bulk-Current Sensor and Secure Flush Code Eraser Against Laser Fault Injection Attack on Cryptographic Processor," in *IEEE Journal of Solid-State Circuits*, vol. 53, no. 11, pp. 3174-3182, Nov. 2018

SoFI Framework

- Assessment tool for ICs against fault injection attacks at gate-level
- Security property driven
- Critical locations are identified
- Considers the capability of specific fault injection technique
- Fault feasibility analysis
- Provides opportunity for local countermeasures to lower overhead

SoFI Overview and Sample Output

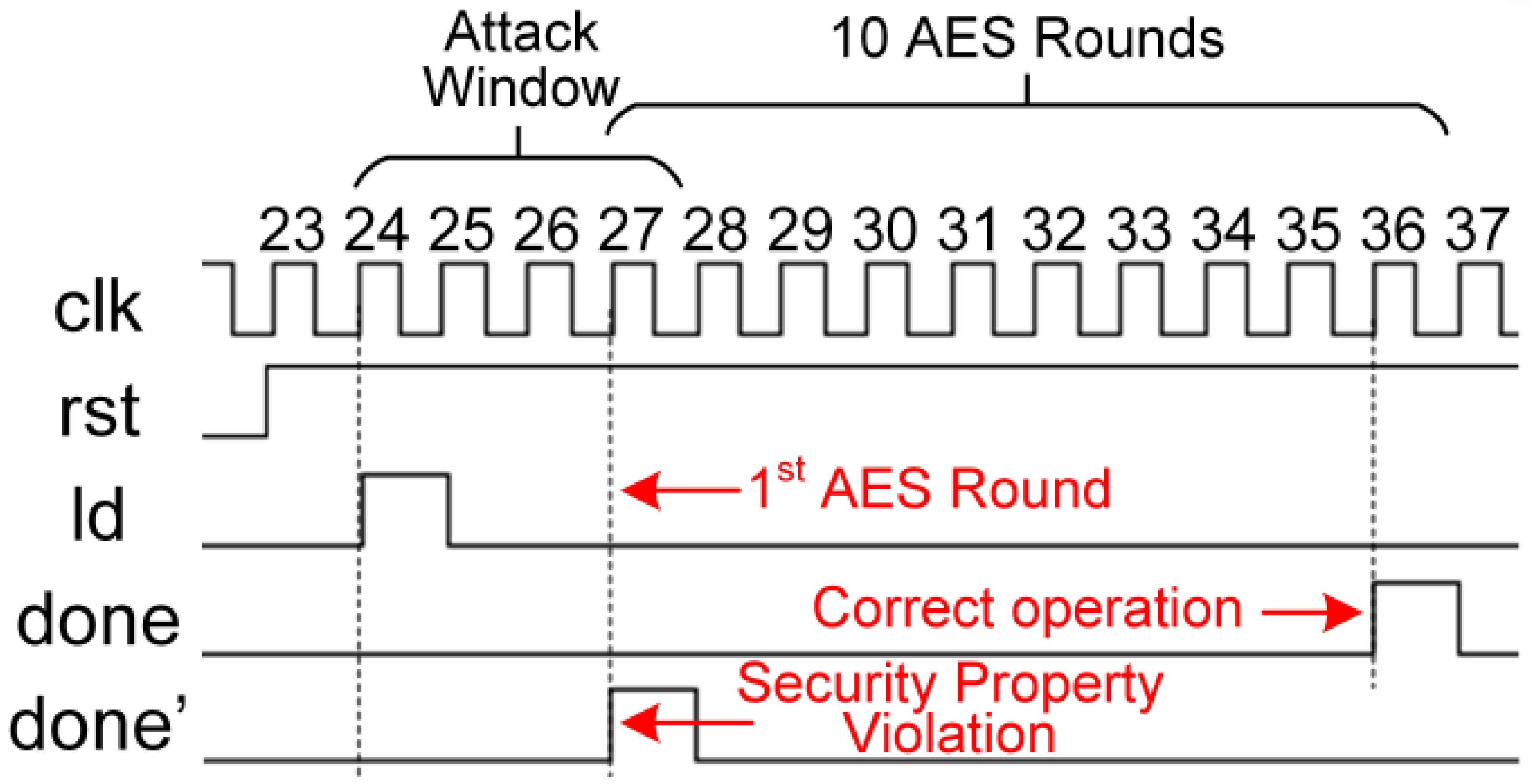



critical_faults.sff

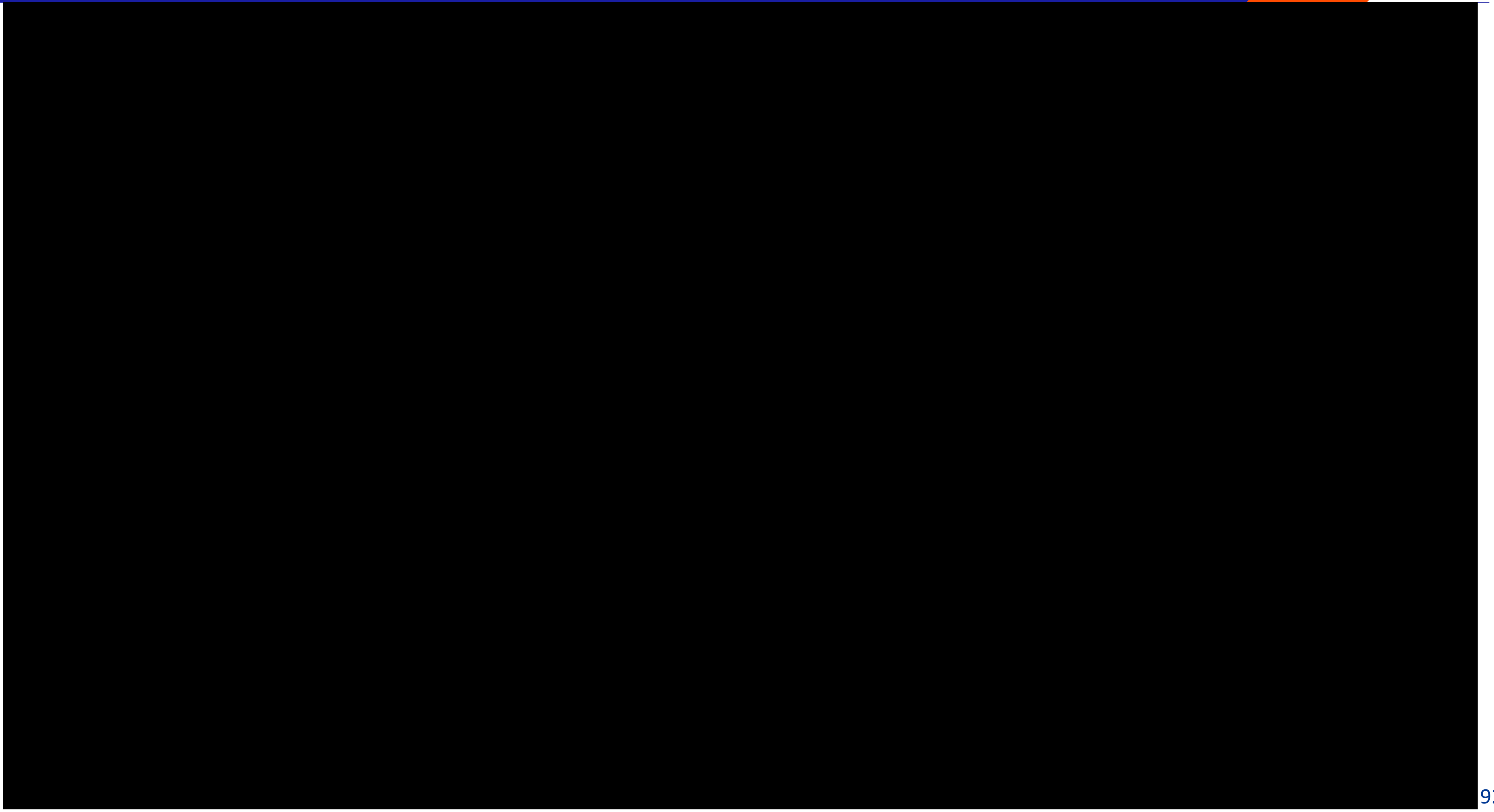
```
Total Faults: 124
Effective Faults: 20
Critical Faults: 3

    NA ~ (27) { FLOP "done_fanin.done_reg.Q" }
    NA ~ (25) { FLOP
    "done_fanin.dcnt_reg_3_.Q" + FLOP
    "done_fanin.dcnt_reg_0_.Q" }
    NA ~ (26) { FLOP
    "done_fanin.dcnt_reg_3_.Q" + FLOP
    "done_fanin.dcnt_reg_1_.Q" + FLOP
    "done_fanin.dcnt_reg_0_.Q" }

Feasible Faults: 1

    NA ~ (25) { FLOP
    "done_fanin.dcnt_reg_3_.Q" + FLOP
    "done_fanin.dcnt_reg_0_.Q" }

Critical Locations: 1

    FLOP "done_fanin.dcnt_reg_3_.Q"
```

Demo on an AES Controller

- **Example Security Property:** The *done* signal that indicates the completion of ten AES rounds cannot be raised in the 1st AES round.



Demo Video



CAD for Security

C/C++

- Information Leakage
- Control flow violations
- Side channel leakage

RTL

- Susceptibility to Trojan Insertion
- Test Generation for Trust Verification
- Power Side-channel Leakage Assessment (RTL-PSC)
- PSC Test Generation (TG)
- EM Leakage Assessment
- Formal Verification of Security Properties

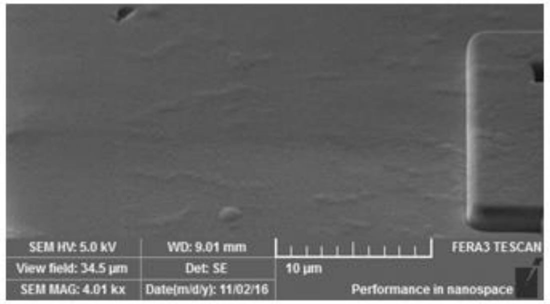
Gate

- Susceptibility of Fault Injection (AVFSM)
- Automated Mapping of Security Properties from C to Gate Level (AutoMap)
- Information Flow Security (IFS)
- Power Side-channel Leakage Assessment (SCRIPT)
- Fault Injection Assessment

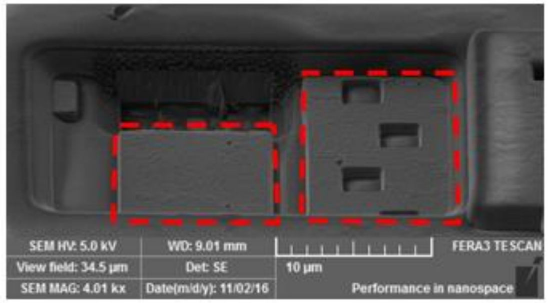
Layout

- Susceptibility of Probing Attacks
- Power Side-channel Leakage Assessment (TVLA)

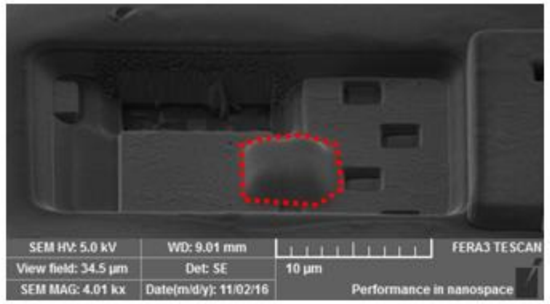
Motivation



Pre-FIB surface



FIB milling to expose adjacent interconnects



FIB deposition to short adjacent interconnects

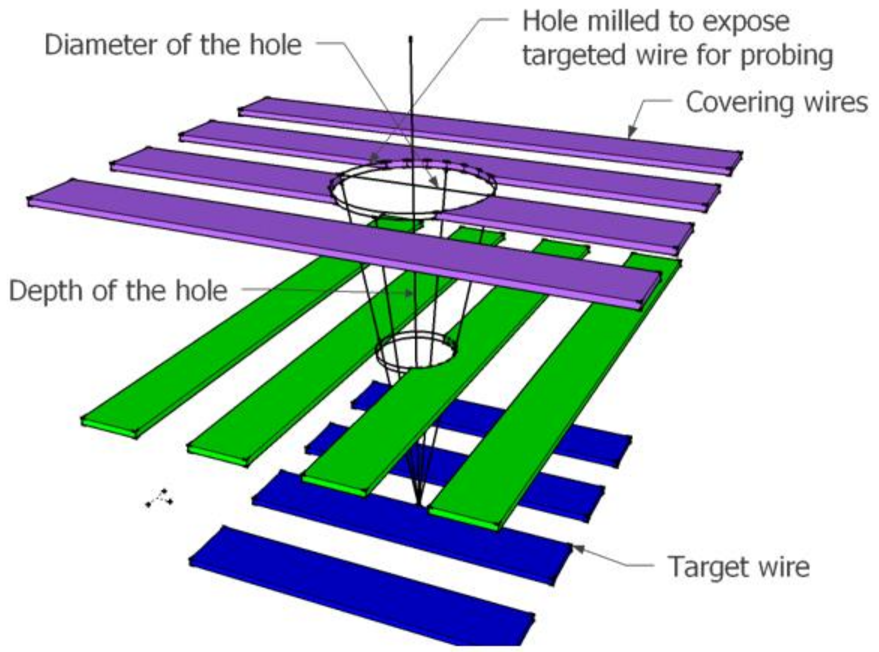
Source: FICS Research (Fera system)

Focused Ion Beam (FIB)

- A powerful tool commonly used in the development, manufacturing, and editing of ICs in nm level precision

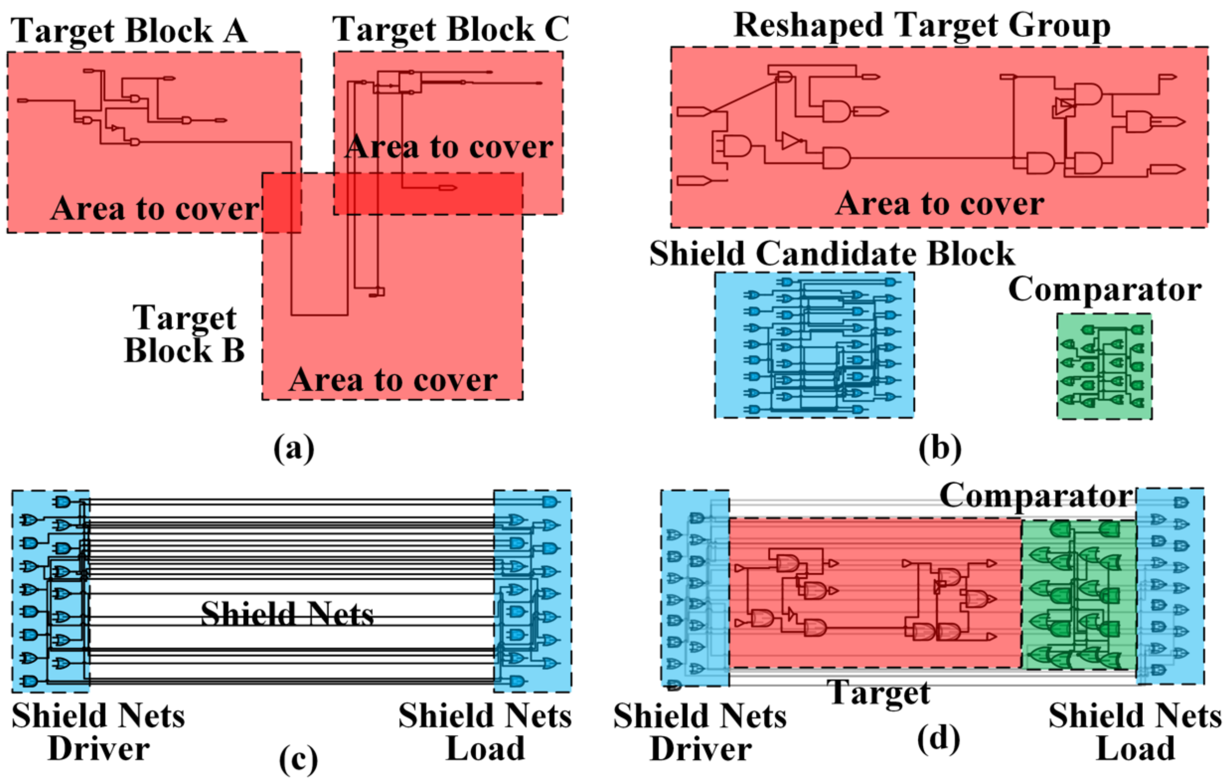
Probing Attack

- Get physical access to signal wires to extract security critical information
- Front-side attack and back-side attack

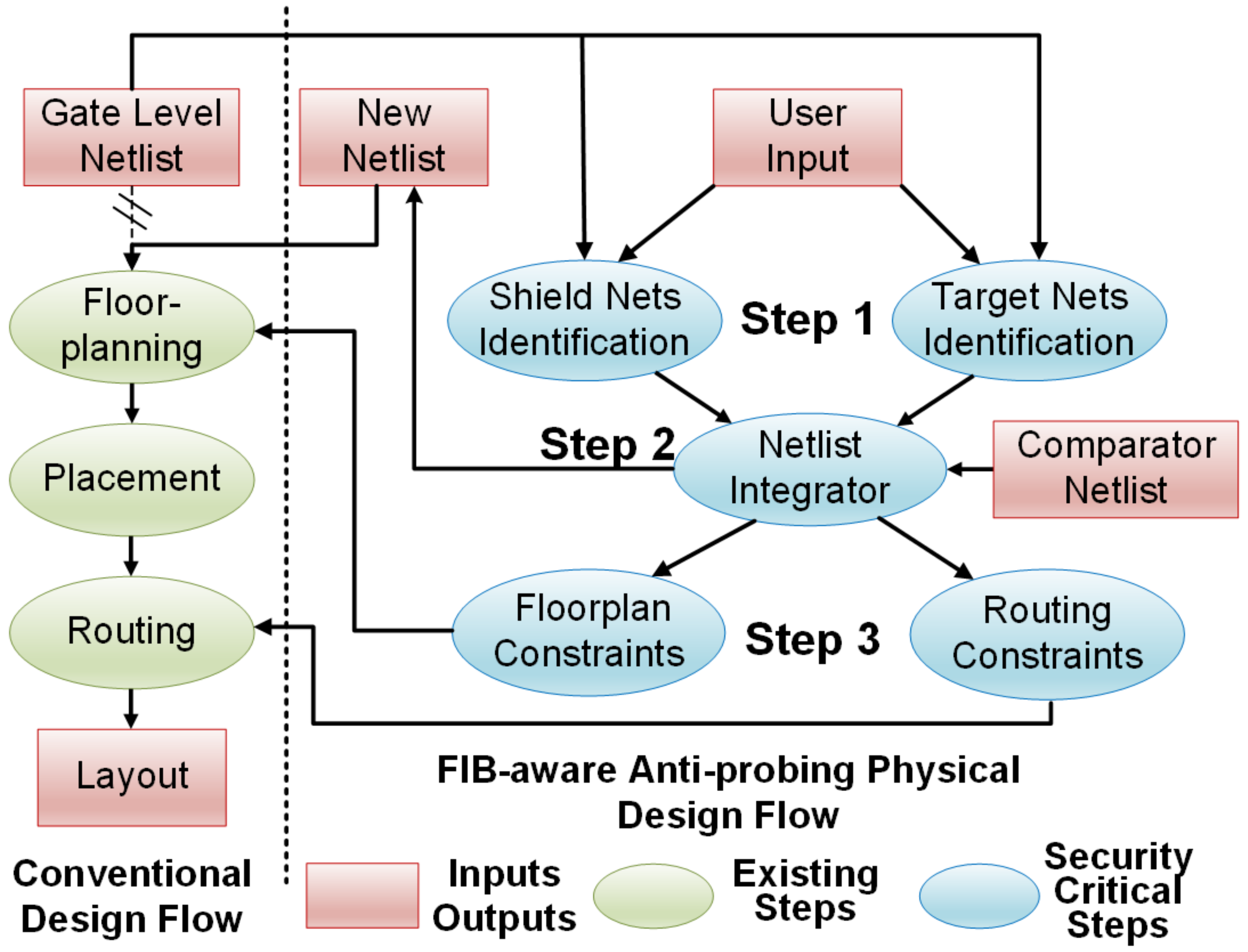


iPROBE Framework

- Automatically identifies target and shield nets
- Groups target nets under internal shield by constrained place and route
- Compares the shield signal and a lower copy to detect milling
- Exposed area metric to assess the vulnerability against probing attacks

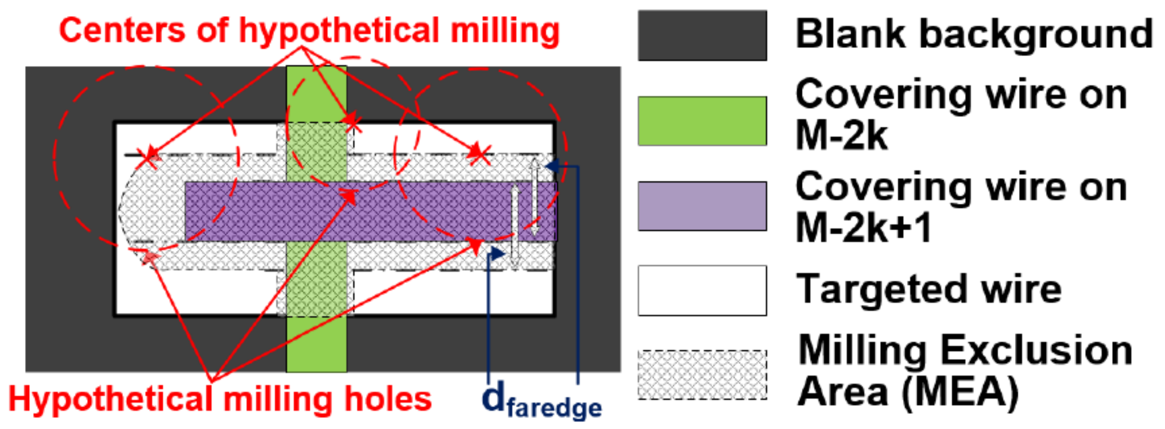


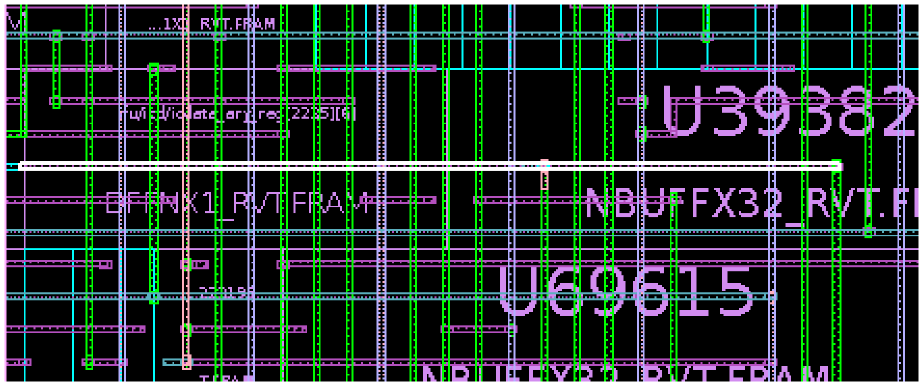
iPROBE Overview



- **Internal Shield**
  - Save area, no pattern generator
  - Mitigate bypass and reroute attack
- **Three new steps**
  - Automatically **identify** target and shield nets
  - Integrate **Comparator** gates in the new netlist
  - **Group** target nets under internal shield by **constrained** floorplan and routing

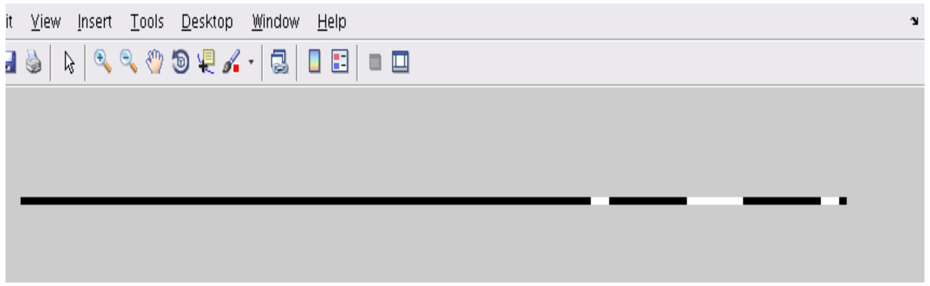
Probing Assessment: Exposed Area





Layout view of targeted wire

- **Milling-exclusion Area (MEA)**
  - If milling center falls in MEA, a covering wire will be completely cut
- **Exposed Area (EA)**
  - Complement of MEA on target wires
  - Free to probe area without impacting signal transmission
  - Designs with large exposed area are vulnerable to probing



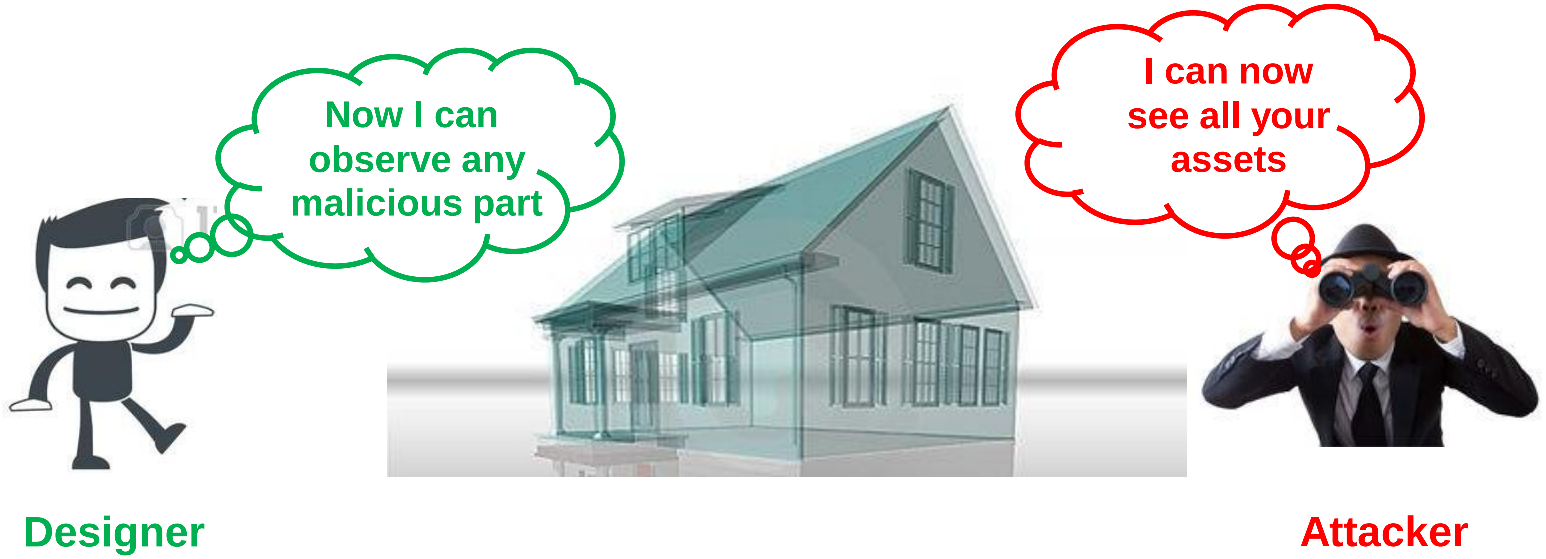
White: Exposed Area -- 11%
Black: Milling-exclusion Area

Demo on AES

Challenges

Challenges

- Some rules can have **conflicting requirement**
  - For malicious change detection → high observability is **desired**
  - For asset leakage → high observability (of asset) is a **serious threat**


Now I can observe any malicious part
I can now see all your assets
Designer
Attacker

- **Risk-cost Analysis**: Invest in addressing threats that matters the most within the given budget/risk
  - Blindly applying rules → unnecessary **design overhead and loss of testability**

Challenges

- Need to develop comprehensive **SoC vulnerability database**
  - Effort underway by TAME working group
- **Formally expressing** security policies and rules
- Metrics
- Need to develop **standards** -- IEEE
- Automated security validation
  - Done at higher levels of abstraction, i.e., C/C++ or RTL
  - Evaluation times need to be **scalable** with the design size
  - Outputs generated should be easily **interpretable** by design engineer

Challenges

- **Usable Security**:
  - Development of **design guidelines** for security → avoid some common security problems
  - **Do-s** & **Don't-s** for designers
  - Best security practices
  - **Low-cost countermeasure** techniques for each vulnerability

References

[1] Salmani, Hassan, and Mohammed Tehranipoor. "Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level." *2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*. IEEE, 2013.

[2] He, Miao., Park, J., Nahiyan, A., Vassilev, A., Jin, Y., & Tehranipoor, M. (2019). RTL-PSC: Automated Power Side-Channel Leakage Assessment at Register-Transfer Level. *IEEE VLSI Test Symposium 2019*. 2019

[3] Y. Jin et al., EMLA: Metrics and Tools for Automated EM-Channel Leakage Analysis at Pre-Silicon, in preparation

[4] Jasper. (2014). JasperGold: Security Path Verification App. [Online].

[5] Contreras, Gustavo K., et al. "Security vulnerability analysis of design-for-test exploits for asset protection in SoCs." *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017.

[6] Nahiyan, Adib, et al. "Hardware Trojan detection through information flow security verification." *2017 IEEE International Test Conference (ITC)*. IEEE, 2017.

[7] A. Nahiyan, F. Farahmandi, D. Forte, P. Mishra and M. Tehranipoor, \Security-aware FSM Design Flow for Mitigating Vulnerabilities to Fault Injection Attacks", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), submitted.

[8] Nahiyan, A., Xiao, K., Yang, K., Jin, Y., Forte, D., & Tehranipoor, M. (2016, June). AVFSM: a framework for identifying and mitigating vulnerabilities in FSMs. In *Proceedings of the 53rd Annual Design Automation Conference* (p. 89). ACM.

References

[9] H. Salmani, M. Tehranipoor, R. Karri, On design vulnerability analysis and trust benchmarks development, in: Computer Design (ICCD), 2013 IEEE 31st International Conference on, IEEE, pp. 471–474.
[10] Wang, Huanyu, et al. "Probing Assessment Framework and Evaluation of Antiprobing Solutions." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2019).
[11] F. Farahmandi and P. Mishra. Automated test generation for debugging arithmetic circuits. In 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 1351– 1356. IEEE, 2016.
[12] F. Farahmandi, Y. Huang, and P. Mishra. Trojan localization using symbolic algebra. In Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific, pages 591–597. IEEE,
2017.
[13] N. Fern, I. San, C. K. Koc¸, and K.-T. T. Cheng. Hardware trojans in incompletely specified on-chip bus systems. In Proceedings of the 2016 Conference on Design, Automation & Test in Europe, pages 527–530. EDA Consortium, 2016.
[14] X. Guo, R. G. Dutta, Y. Jin, F. Farahmandi, and P. Mishra. Pre-silicon security verification and validation: A formal perspective. In ACM/IEEE Design Automation Conference (DAC), 2015.
[15] J. Rajendran, V. Vedula and R. Karri. Detecting malicious modifications of data in third party intellectual property cores. In ACM/IEEE Design Automation Conference (DAC), pages 112–118, 2015.
[16] Jonathan Cruz, Farimah Farahmandi, Alif Ahmed, and Prabhat Mishra, “Hardware Trojan Detection using ATPG and Model Checking,” International Conference on VLSI Design (VLSI Design), pages 91-96, Pune, India, January 6 – 10, 2018.

See Trust-Hub to access benchmarks, tools, hardware platforms, etc.
www.trust-hub.org

SoC Security
http://trust-hub.org/vulnerability-db/cad-solutions

Mark Tehranipoor, tehranipoor@ece.ufl.edu
Farimah Farahmandi, farimah@ece.ufl.edu