EMNESS

2020-1-FR01-KA203-080184

WP3.3.1: Verification and Test of Secure Circuits

Vincent Beroulle Grenoble INP- UGA, Esisar

04/05/2023

La Région Auvergne-Rhône-Alpes
Agence Erasmus+ France · Education Formation
Erasmus+

Outline

1. Introduction
2. **Hardware Functional Verification**
   1. Introduction
   2. Simulation
   3. Emulation & Prototyping
   4. Formal verification
   5. **Security verification**
3. Hardware Testing
4. HW/SW Co-verification

Security Verification

- This part has been funded by ERAMUS+ program, and AURA region through the EMNESS project (with several European University)

References

- Bhunia, S., Ray, S., & Sur-Kolay, S. (Eds.). (2017). *Fundamentals of IP and SoC security*. New York: Springer.
- Azar, K. Z., Hossain, M. M., Vafaei, A., Al Shaikh, H., Mondol, N. N., Rahman, F., ... & Farahmandi, F. (2022). Fuzz, Penetration, and AI Testing for SoC Security Verification: Challenges and Solutions. *Cryptology ePrint Archive*.
- Schneider, T., & Moradi, A. (2015, September). Leakage assessment methodology. In *International Workshop on Cryptographic Hardware and Embedded Systems* (pp. 495-513). Springer, Berlin, Heidelberg.
- Athanasios Papadimitriou, PhD on « RTL Modeling of laser fault attacks for the evaluation of integrated secure circuit and countermeasure design », 2016

Security Verification Outline

Security verification

1. Introduction
2. Functional verification of security primitives
   1. Dynamic analysis
      1. Fuzzing
      2. Hackaton/Pentesting
   2. Static analysis
3. Robustness verification against FIA
   1. Introduction
   2. Fault models
      1. Glitch fault attacks
      2. Laser fault attacks
4. Leakage verification

Security verification

Introduction

- Section requirements:
  - Security primitives (encryption, hash, …)
  - Design for Trust, Design for test, Design for Debug
  - Functional verification (test plan, dynamic and static methods: equivalence checking, model checking …)
  - Robustness evaluation (fault simulation, fault modeling, …)
  - Fault Injection Attacks (FIA), Hardware Trojan (HT), Side Channel Attacks (SCA)

Security verification

Introduction

- SoC integrate numerous primitives or services related to security

Security primitives

- Cryptographic cores (Encryption/decryption or Hash functions)
- PUF or TRNG (Physically Unclonable Functions or True Random Number Generation)

Services related to security

- Design for Trust
- Design for Test (DfT) or for Debug (DfD)

- Unauthorized or malicious use of these primitives or services can result in company trade secrets

Security verification
Introduction

- **Security objectives are difficult to specify**:
  - which test plan? which coverage/success criteria?
  - Test plan must include a large number of activities
- **Security verification is still on the rise**
  - In the following, we focus only on the hardware-level security verification: RTL and gate-level netlist verification
    - At these levels, security verification is very important (finding a bug latter generate higher costs = "*rule of ten*")
    - White box verification (rather than SoC level is more black box or grey box)
    - Overall security is a multi-stages and cross-layers problems

HEMNESS Erasmus+ La Région Auvergne-Rhône-Alpes

Security verification

Introduction

- **The steps for the security verification**

1. Identification of *vulnerabilities*
2. Identification of *security assets*
3. Definition and formalization of *security properties* (for example using PSL)
4. Test plan definition
5. Test plan implementation

Security verification

Introduction

- **Security vulnerability examples:**
  - Designer mistakes => insecure implementation
  - Rogue employee => manipulating hardware to facilitate obtaining security assets
  - Untrusted third-party IP vendors => IP watching the bus to obtain information
  - EDA insecure optimizations => resource sharing containing secret information
  - DfT functionalities => Unauthorized access to secret registers

[Fuzz, Penetration, and AI Testing for SoC Security Verification: Challenges and Solutions - FUTURE MICROELECTRONICS SECURITY RESEARCH SERIES 2022

EMNESS Erasmus+ La Région Auvergne-Rhône-Alpes

Security verification

Introduction

Security Assets

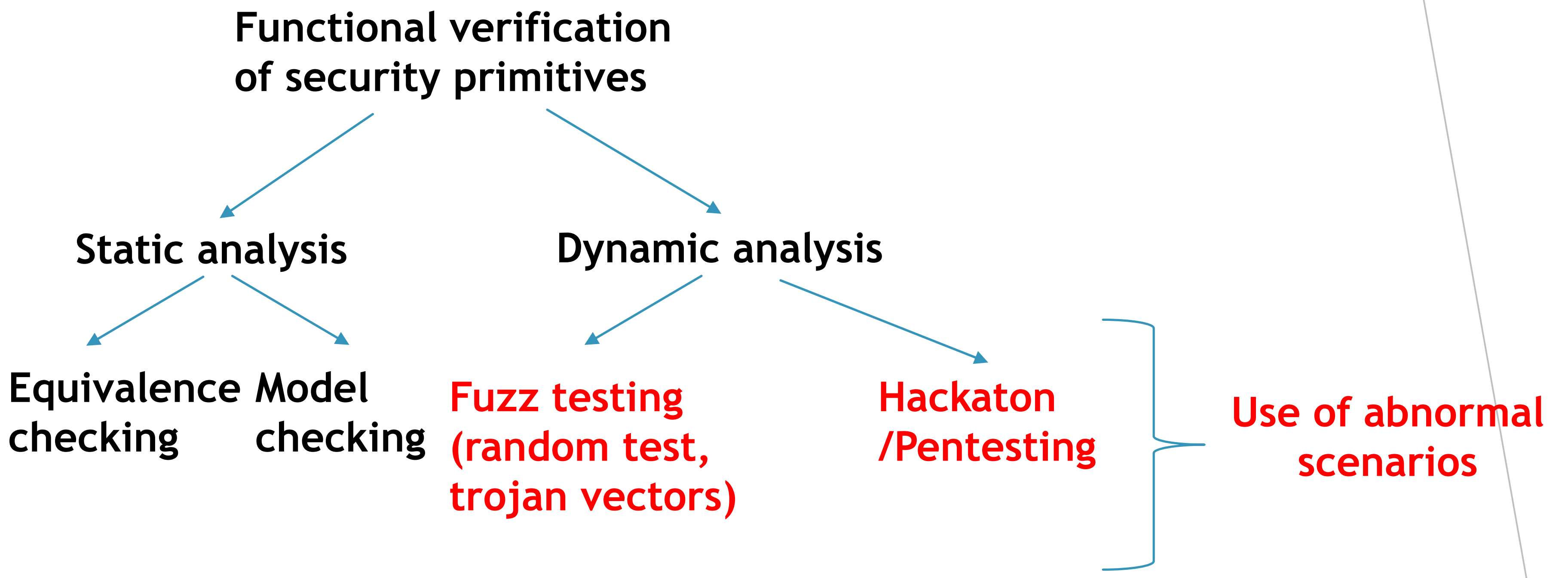
- **List of information** whose leakages can lead to catastrophic consequences
  - On-device key
  - Manufacture firmware
  - On-device protected data
  - Device configuration

Security policies

- A **set of requirements** related to security assets
  - Access restrictions
  - Data/control flow restrictions
  - HALT/OTS/DOS restrictions

Security verification

Introduction – security verification in a nutshell

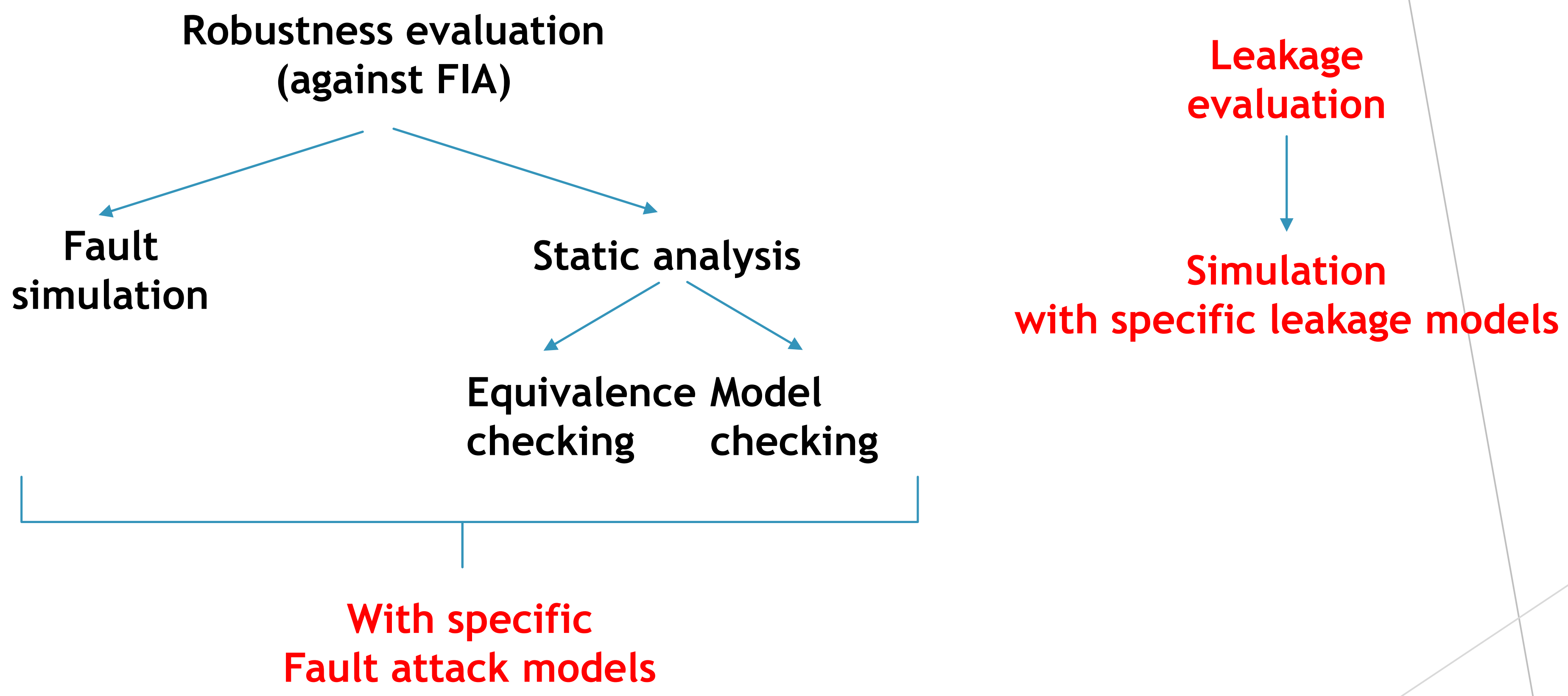

Functional verification of security primitives
Static analysis
Dynamic analysis
Equivalence checking
Model checking
Fuzz testing (random test, trojan vectors)
Hackaton /Pentesting
Use of abnormal scenarios

Objectives → **Security property verification**
+
Hardware Trojan (or **backdoors**) **detection**

In red, the new tasks for security verification

Security verification

Introduction – security verification in a nutshell


Robustness evaluation (against FIA)
Fault simulation
Static analysis
Equivalence checking
Model checking
With specific Fault attack models
Leakage evaluation
Simulation with specific leakage models

In red, the new tasks for security verification

Security verification

Functional verification of security primitives – Dynamic analysis

- Fuzzing (or *fuzz testing*) is a **testing technique** that involves providing **invalid/unexpected random inputs** and monitoring the results for finding exceptions (*crashes, corner coverage*)
  - Unexpected inputs can generate: buffer overflows, exceptions, race conditions, access violations, denial of service

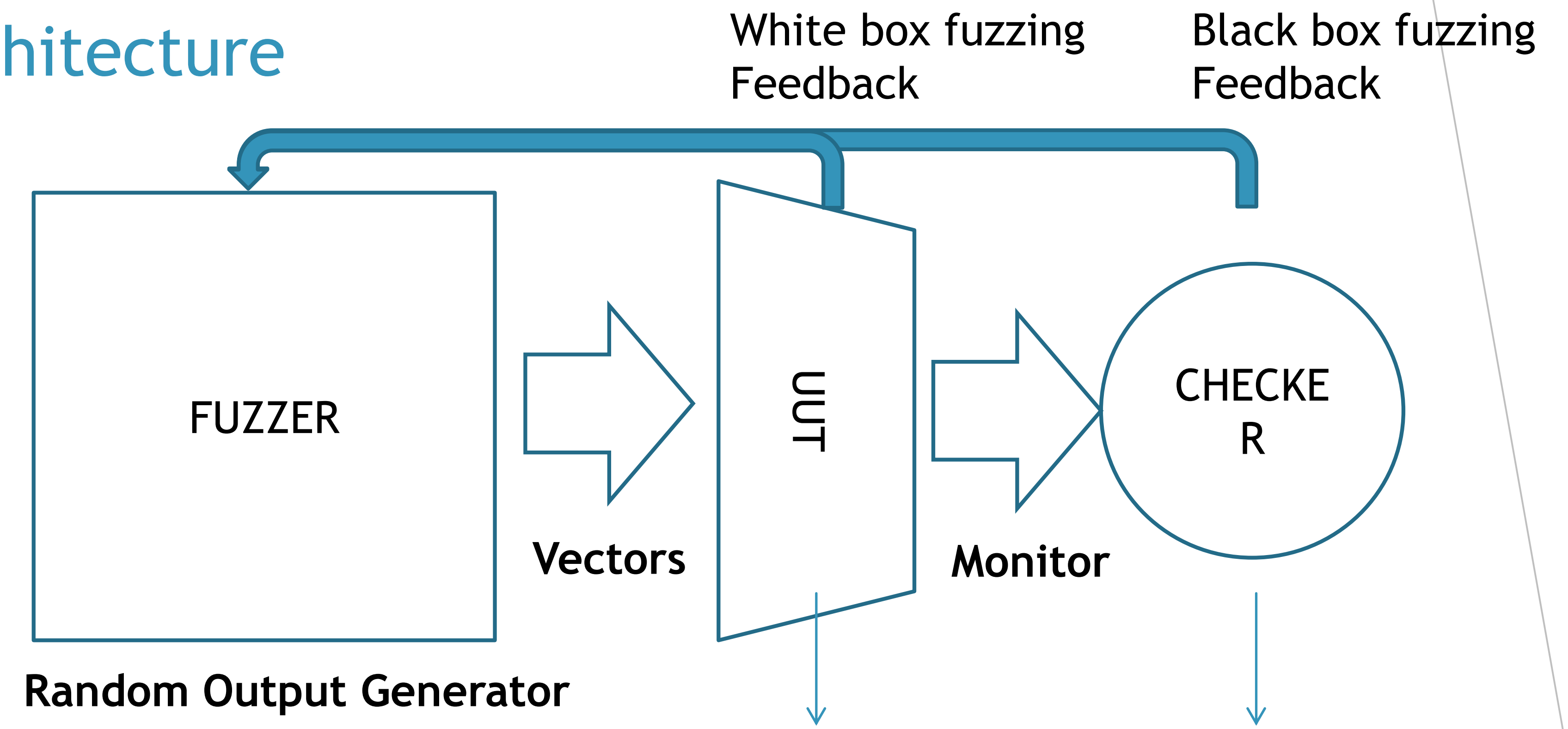
Security verification

Functional verification of security primitives – Dynamic analysis

- Blind fuzz = no feedback => low coverage
- Fuzzers generally use feedback (coverage, security property coverage) for evaluating **the cost function of genetic algorithms**
- Examples of software fuzzers: American Fuzzy Lop (AFL) and Hongfuzz

Security verification

Fuzzing architecture



- Bit flipping/Mutations
- Long strings/Large integers
- « No rules »/Strange scenarios

White/grey/black box fuzzing

- Code coverage
- Security property coverage

Observed behaviors:

- Crash (not at RTL or gate level)
- Hang/freeze
- Security property coverage

Security verification

Functional validation of security primitives – Dynamic verification

- **Hackathon/pentesting (H/P)**: **white-box hacking** to break security properties
  - H/P definition: test methodology that propagates the effects of vulnerability to an observable point
- Hardware H/P: vulnerabilities in hardware can be purely hardware-oriented
  - malicious hardware modification, side channel leakage, fault injection vulnerability
- RTL H/P is more important than post-silicon H/P as the silicon can not be patched

EMNESS Erasmus+ La Région Auvergne-Rhône-Alpes

Security verification

Functional verification of security primitives – Static verification

- **Equivalence checking** can help designers to confirm the desired functionalities
  - Specification = implementation
  - *Nothing more and nothing less*
- But this technique is limited to medium size circuits
- **Model checking** can also help designers to confirm the security properties

Security verification

Functional validation of security primitives – Static verification

- The verification of the absence of Hardware Trojan is a difficult challenge
  - Hardware Trojan (HT) are triggered by very rare conditions: **hard to activate with a normal dynamic test**
  - **Unused code identification approaches** look for unused portions in a circuit description => this allows detecting HT
  - At Silicon-Level, Side Channel Analysis (delay or power leakages) can also help detecting the HT **even without triggering it**

Erasmus+

La Région Auvergne-Rhône-Alpes

Security verification

Robustness evaluation - introduction

- Fault Injection Attack (FIA) can modify the circuit functionalities (to break some security properties)
- It is possible to **reuse functional safety verification or robustness evaluation** techniques for detecting security vulnerabilities due to these attacks

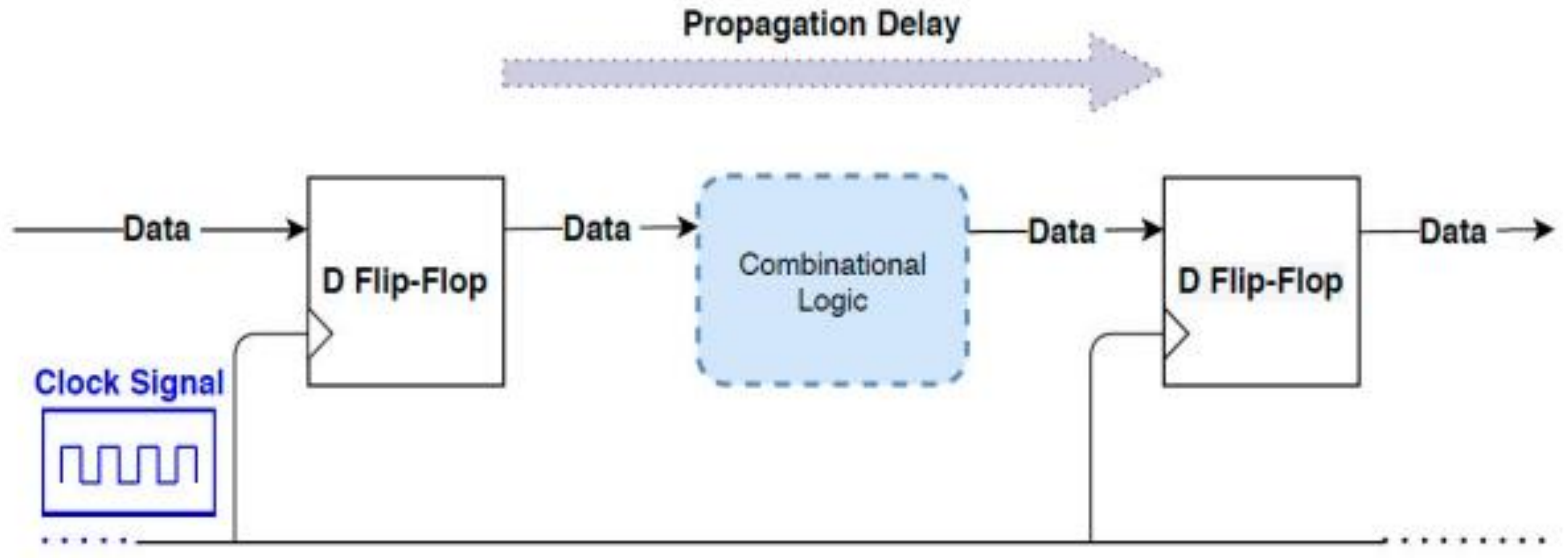
Security verification

Robustness evaluation – Fault model

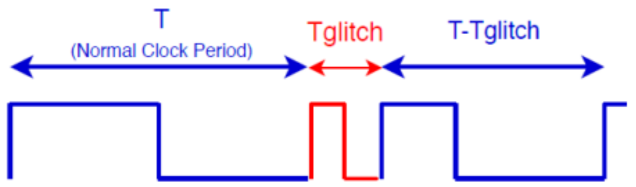
- FIA can be simulated (or emulated) with specific fault models
  - For glitch attacks (Clk/Voltage/EM): single or multiple faults **in the FFs involved in the most critical paths**
  - For laser fault attacks (with localized effects): single or multiple faults **in the FFs involved in the intersecting logical cones**

Security verification

Robust



Clk glitch injection



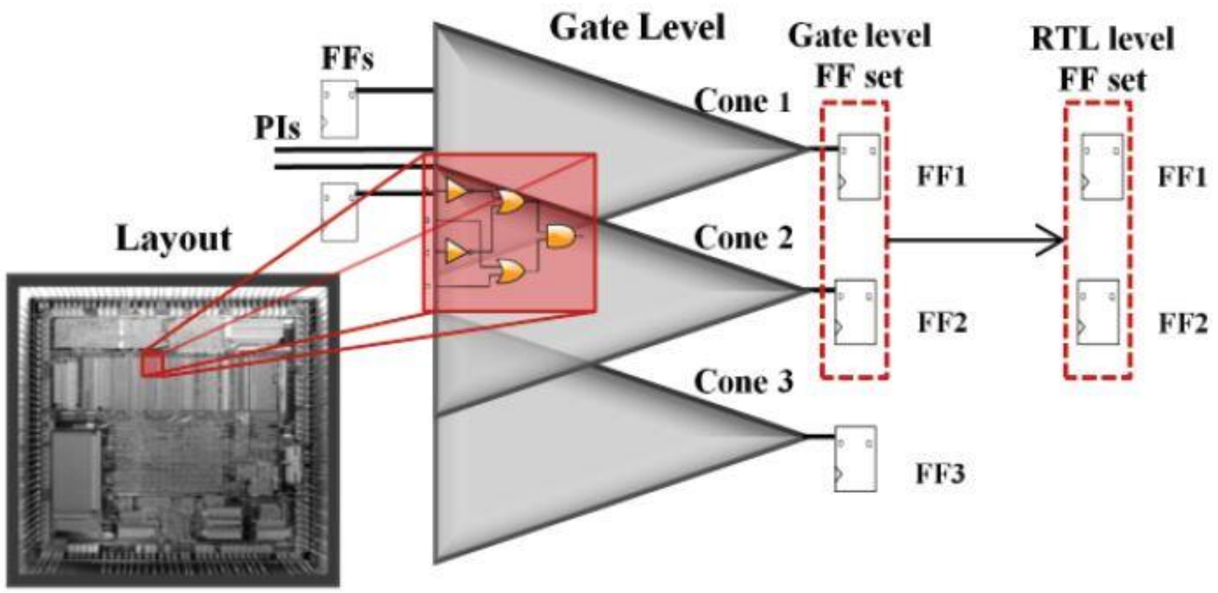
- Faults are more probable in FFs related to the most critical paths
- As Voltage Glith Attacks and EMA involve higher gate delays, then more faults are probable in FFs related to the most critical paths

Fault model is: no FF update

EMNESS Erasmus+ La Région Auvergne-Rhône-Alpes

Security verification

Robustness evaluation – Fault model – Laser fault attacks



Layout Fault Model [Athanasios Papadimitriou, PhD on « RTL Modeling of laser fault attacks for the evaluation of integrated secure circuit and countermeasure design », 2016
]

- More gates in the logic cone more probability to inject a fault in the FF
- Cone intersections => multi-bits fault injections

- **SEU or MBU or MCU**
- **Bit flip or bit set or bit reset**

Security verification

Leakage evaluation - introduction

- Side channel simple tests measure the information leakage from the design
- Does not require neither understanding the hardware design, nor attack model, nor realizing an attack
- The most known test is “Test Vector Leakage Assessment” (TVLA) (similar to T-test.)

This test can report that the DUT fails to provide the desired security level

Security validation
Leakage evaluation – TVLA

- TVLA procedure:

1. Create 2 datasets Q1 and Q2, each with n instances of plain-text and both the same key.
   - Q1: same plain-texts
   - Q2: random plain-texts
2. Obtain n power traces (which can be estimated at the RTL level thanks to specific leakage models) for each dataset and compute the

$$TVLA = \frac{mean(Q_1) - mean(Q_2)}{\sqrt{\frac{Var(Q_1)}{n} + \frac{Var(Q_2)}{n}}}$$

3. If TVLA>4,5 => the device is not secure against side channel attack