Fault-Resistant Partitioning of Secure CPUs for System Co-Verification against Faults*

Simon Tollec[1], Vedad Hadžić[2], Pascal Nasahl[2,3], Mihail Asavoae[1], Roderick Bloem[2], Damien Couroussé[4], Karine Heydemann[5,6], Mathieu Jan[1], and Stefan Mangard[2]

[1] Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
[2] Graz University of Technology, Graz, Austria
[3] lowRISC C.I.C., Cambridge, United Kingdom
[4] Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France
[5] Sorbonne Univ., CNRS, LIP6, F-75005, Paris, France
[6] Thales DIS, France

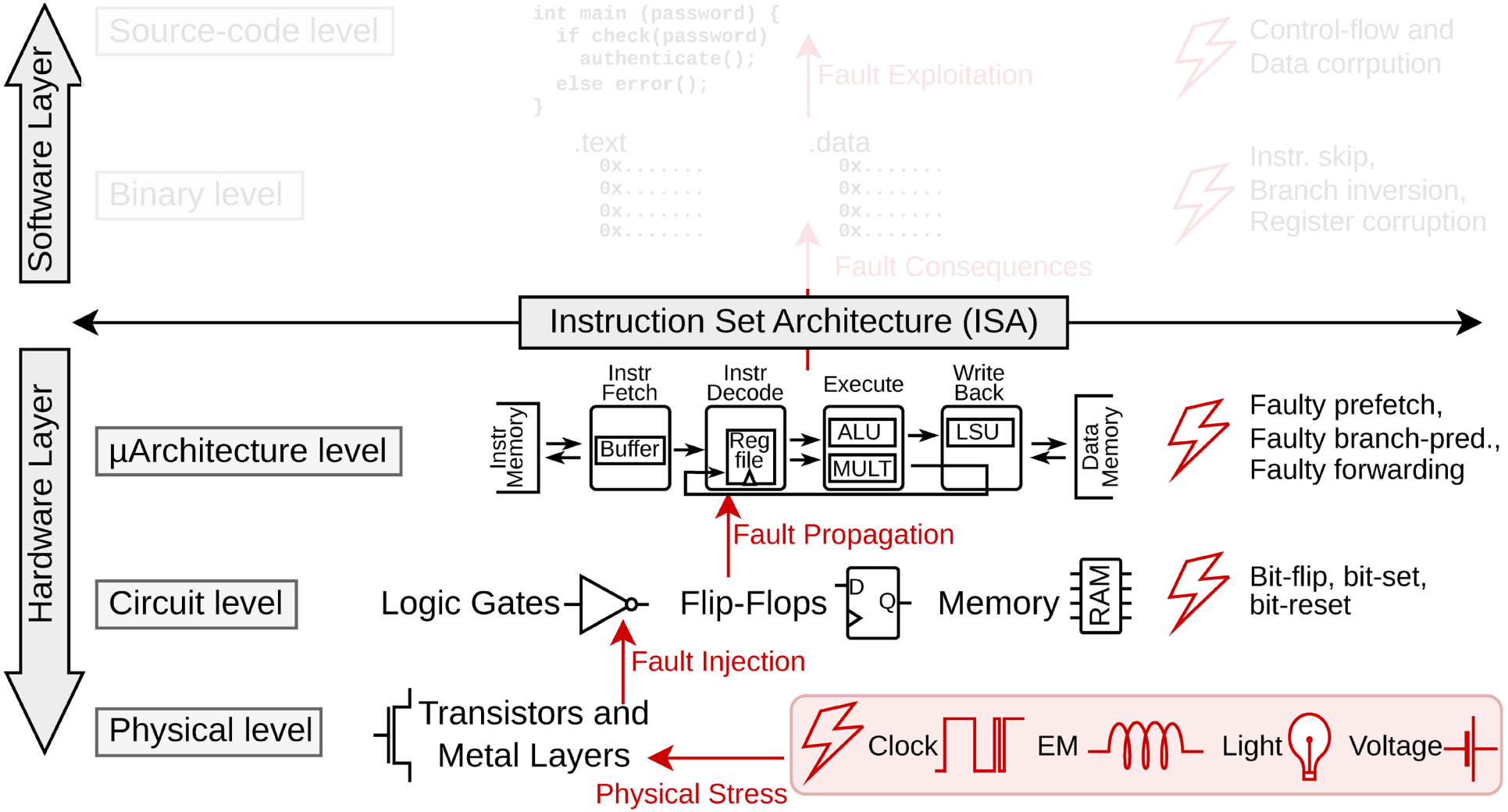
* Publié à CHES'24

JAIF, Oct 1, 2024

1. Introduction

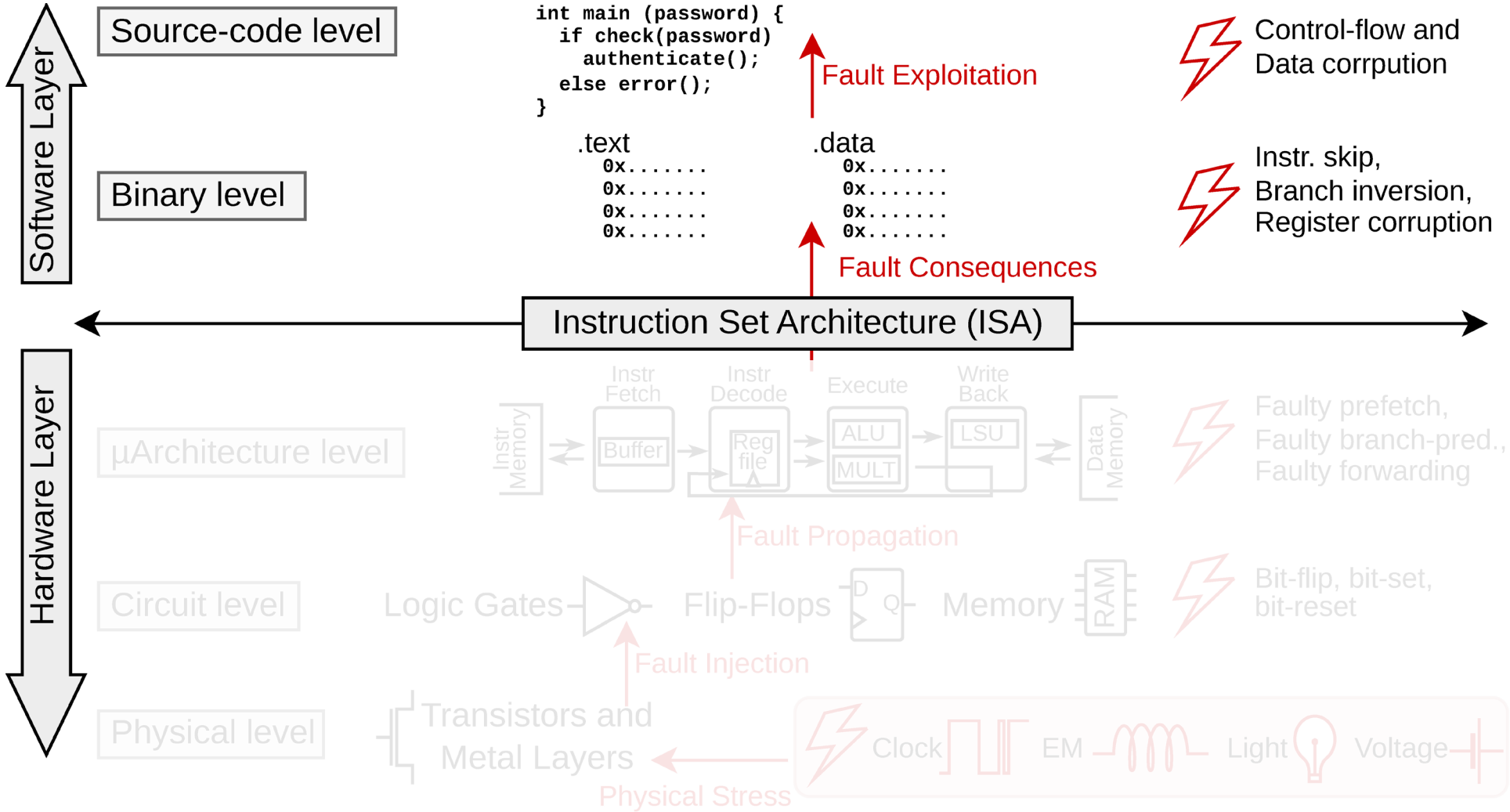
State of the Art & Motivations

CPU's Abstraction Levels during FIs



State of the Art & Motivations

CPU's Abstraction Levels during FIs

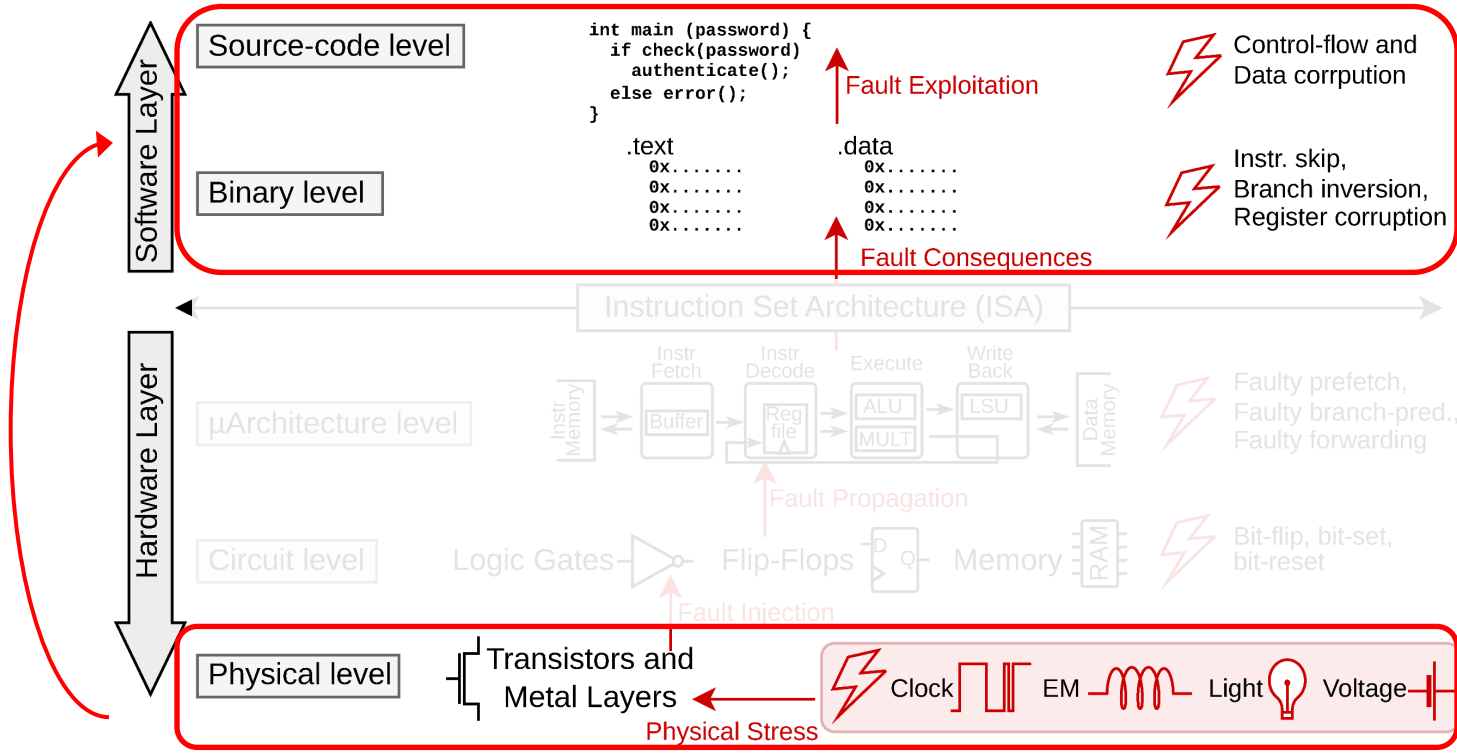


State of the Art & Motivations

Experimental Characterization

- Post-Silicon (often a black box)
- Apply physical stress
- Observe the consequences on software

Observe consequences



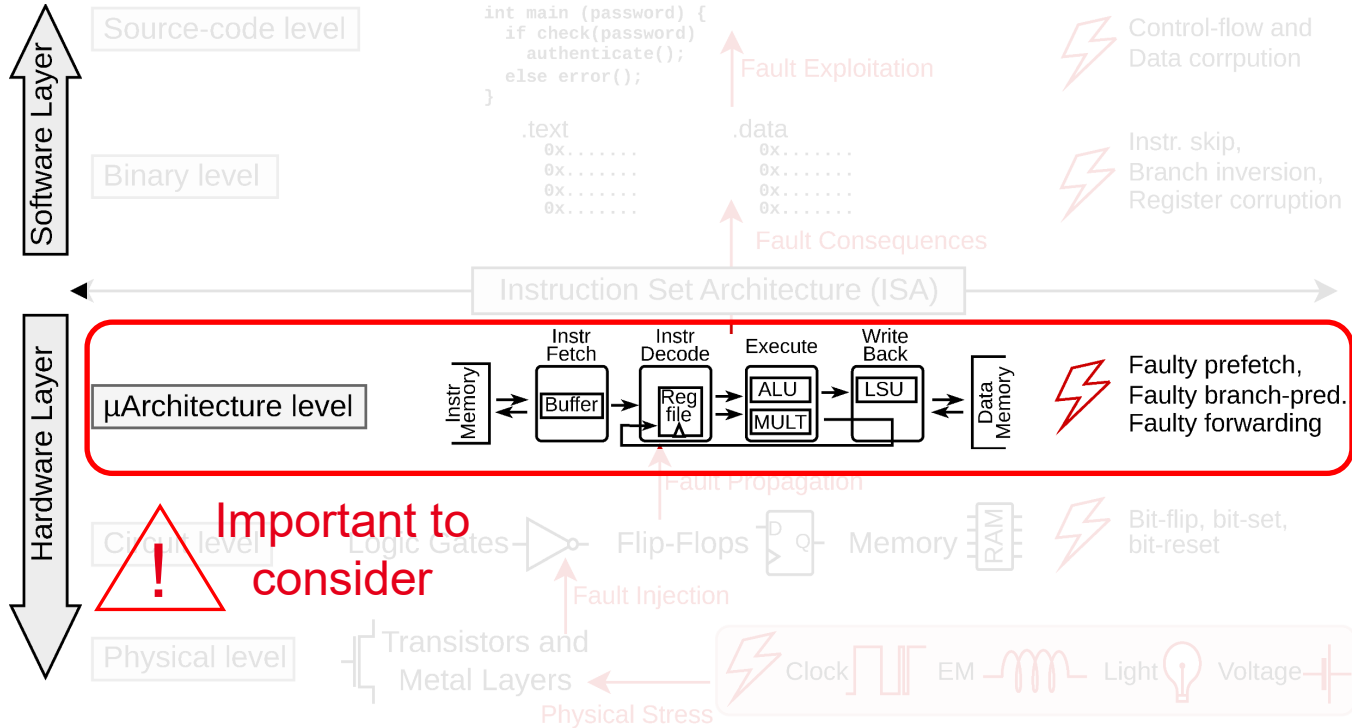
State of the Art & Motivations

Experimental Characterization

- Post-Silicon (often a black box)
- Apply physical stress
- Observe the consequences on software

Need to Open the Box

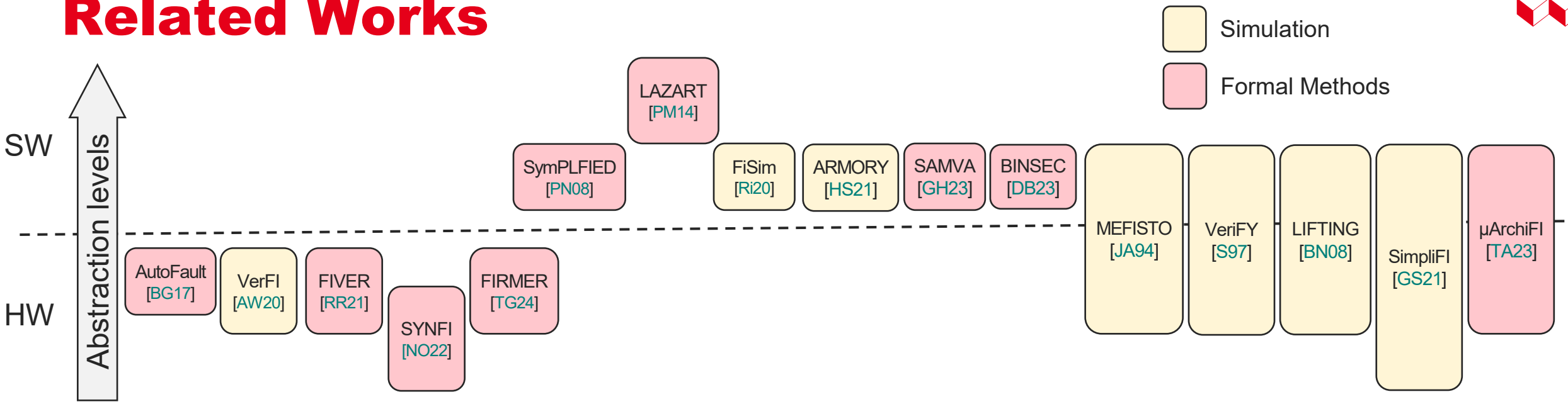
- Unexplainable observed effects [PH19]
- Importance of microarchitectural mechanisms
  - Pipelining [BG11, YG15]
  - Prefetch Buffer [TA22]
  - Forwarding [LB18, LB19]
  - Memory Cache [TB21]



- ***Pre-silicon*** *analyses should now consider* **processor microarchitecture**
- ***Security*** *evaluations require* ***exhaustive*** *methods*

[BG11] Balasch et al. "An in-depth and black-box characterization of the effects of clock glitches on 8-bit MCUs." *FDTC* 2011.
[YG15] Yuce and Schaumont. "Improving fault attacks on embedded software using RISC pipeline characterization." *FDTC* 2015.
[PH19] Proy et al. "A first ISA-level characterization of EM pulse effects on superscalar microarchitectures: a secure software perspective." *ARES* 2019.
[LB18] Laurent et al. "On the importance of analysing microarchitecture for accurate software fault models." *DSD* 2018.
[LB19] Laurent et al. "Fault injection on hidden registers in a risc-v rocket processor and software countermeasures." *DATE* 2019.
[TB21] Trouchkine et al. "Electromagnetic fault injection against a complex CPU, toward new micro-architectural fault models." *Journal of Cryptographic Engineering* 2021.

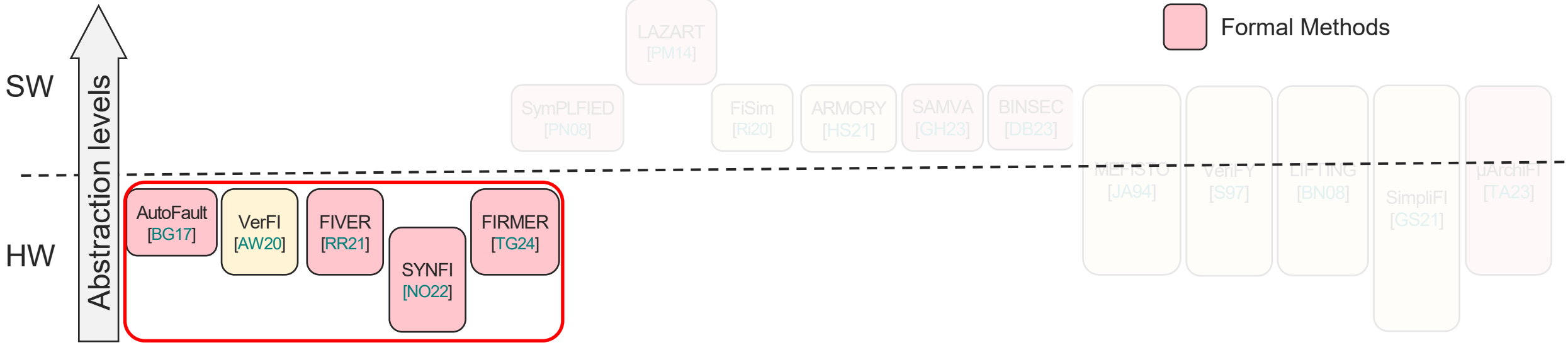
Related Works



Pre-Silicon Evaluation: Simulation vs Formal Methods

- Simulation: Execute concrete instances of the system
- Formal Methods: Reason on a system model to prove properties

Related Works



Pre-Silicon Evaluation: Simulation vs Formal Methods

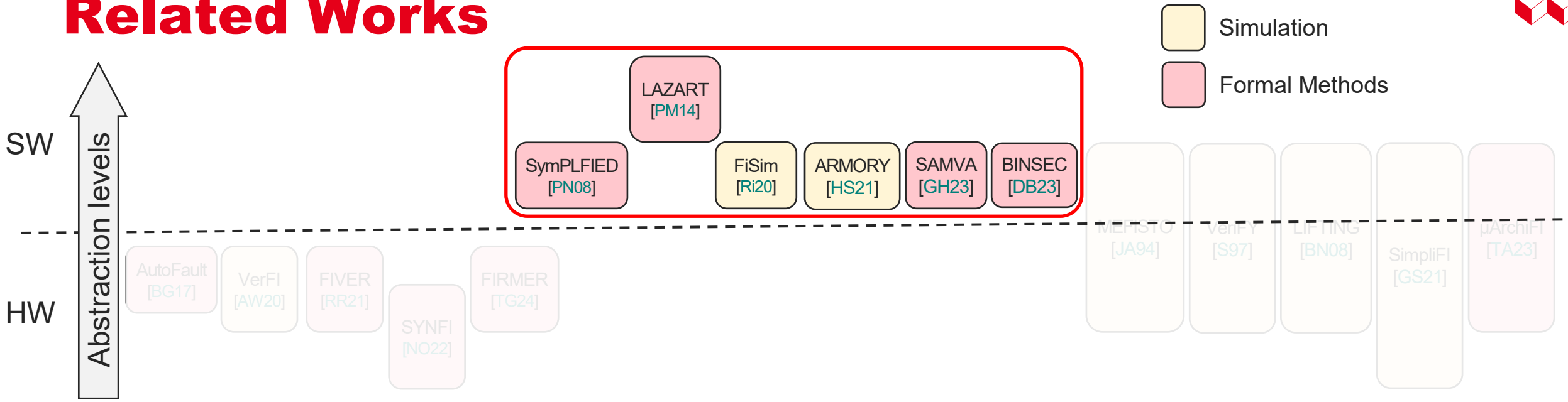
- Simulation: Execute concrete instances of the system
- Formal Methods: Reason on a system model to prove properties

Hardware Methods

- Dedicated to evaluate crypto circuits
- Compare fault-free vs. faulted circuits
- Performance (FIVER on AES):
  - 1 fault in **22 sec.**
  - 2 faults in **130 hours**

➢ Cannot model program execution

Related Works



Pre-Silicon Evaluation: Simulation vs Formal Methods

- Simulation: Execute concrete instances of the system
- Formal Methods: Reason on a system model to prove properties

Hardware Methods

- Dedicated to evaluate crypto circuits
- Compare fault-free vs. faulted circuits
- Performance (FIVER on AES):
  - 1 fault in **22 sec.**
  - 2 faults in **130 hours**

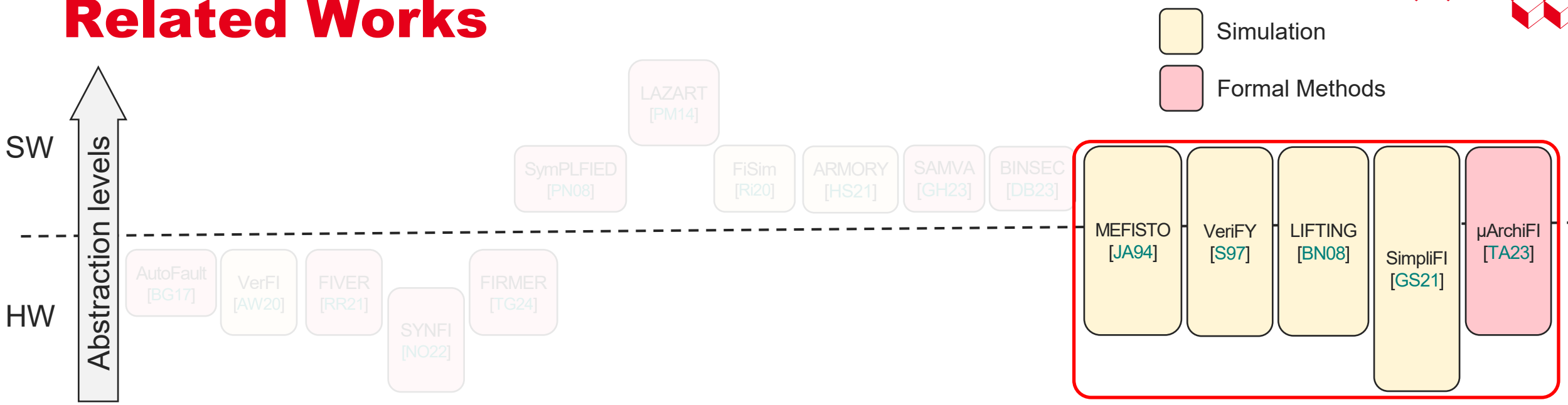
➢ Cannot model program execution

Software Methods

- ISA fault models, e.g., inst. skip
- Performance (BINSEC):
  - Bootloader: 1 fault in **9 sec.**
  - PIN comp.: 10 faults < **1 sec.**

➢ Ignore the underlying processor implementation

Related Works



Pre-Silicon Evaluation: Simulation vs Formal Methods

- Simulation: Execute concrete instances of the system
- Formal Methods: Reason on a system model to prove properties

Hardware Methods

- Dedicated to evaluate crypto circuits
- Compare fault-free vs. faulted circuits
- Performance (FIVER on AES):
  - 1 fault in **22 sec.**
  - 2 faults in **130 hours**
- ➢ Cannot model program execution

Software Methods

- ISA fault models, e.g., inst. skip
- Performance (BINSEC):
  - Bootloader: 1 fault in **9 sec.**
  - PIN comp.: 10 faults < **1 sec.**
- ➢ Ignore the underlying processor implementation

Combined Methods

- Originally for safety analysis with simulation
- Identify complex interplay between HW and SW
- ➢ Scalability issues (µArchiFI in **14 hours**)
  - 100 instructions
  - Small in-order CPU (46 kGE)
  - 1 fault injection

Problem Statement and Contributions

Observation

- HW- or SW-only evaluations are insufficient

Need

- **Pre-silicon combined HW/SW** analyses
- **Exhaustive** techniques, e.g., formal methods ➔ Security guarantees

Problem Statement and Contributions

Observation

- HW- or SW-only evaluations are insufficient

Need

- **Pre-silicon combined HW/SW** analyses
- **Exhaustive** techniques, e.g., formal methods ➔ Security guarantees

Challenge: State space explosion problem

- *Exploring* **the entire state space ➔ Scalability issues**
- *Addressing scalability is essential for* **practical applicability**

Problem Statement and Contributions

Observation

- HW- or SW-only evaluations are insufficient

Need

- **Pre-silicon combined HW/SW** analyses
- **Exhaustive** techniques, e.g., formal methods ➔ Security guarantees

Challenge: State space explosion problem

- *Exploring* **the entire state space ➔ Scalability issues**
- *Addressing scalability is essential for* **practical applicability**

Contributions

- **Decompose** HW/SW co-verification ➔ **Contain the state space explosion**
- **Address previously intractable** use cases with our new methodology

Outline

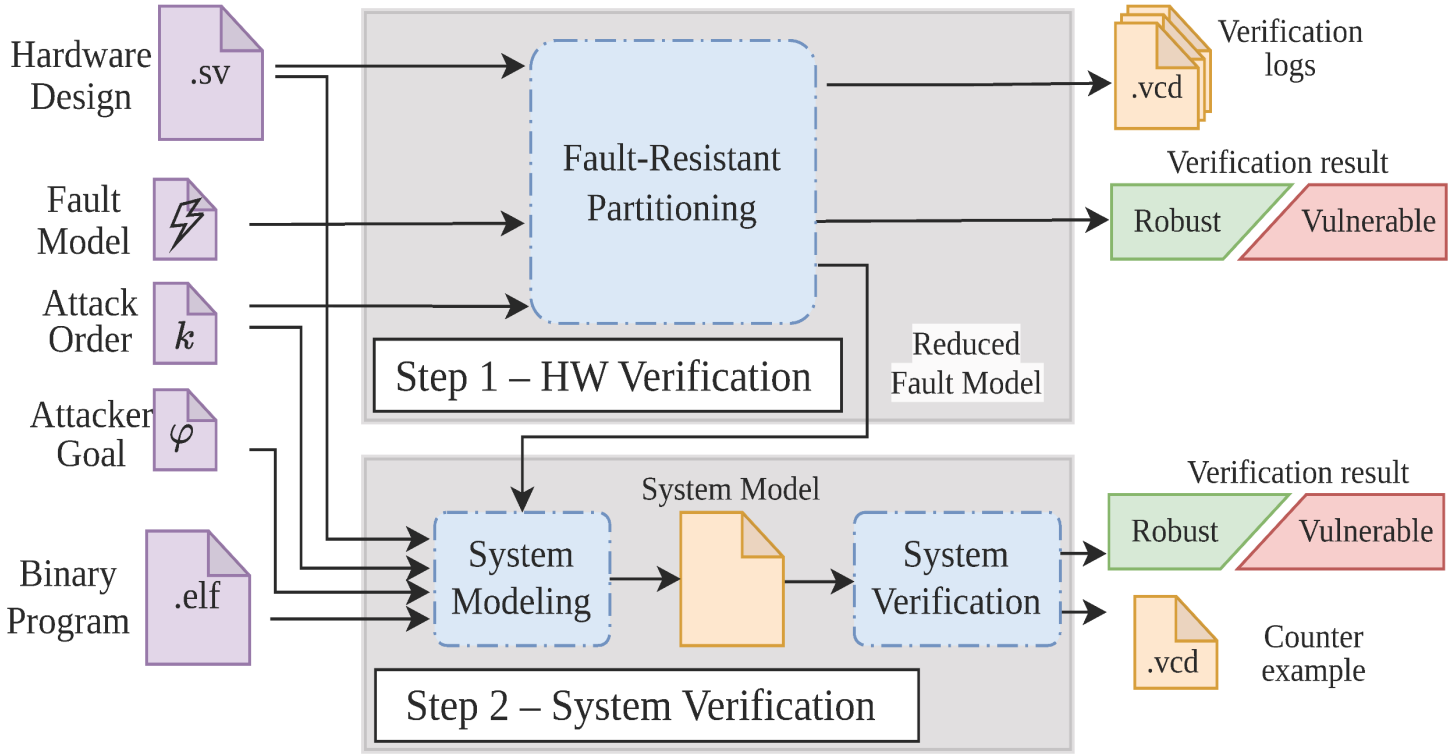
1. Introduction
2. Methodology
3. Validation on Impeccable Circuits
4. Evaluation of OpenTitan
5. Conclusion

2. Methodology

Methodology Overview

Principle

- Security-critical systems implement **HW protections**
- **Preliminary evaluation** of the hardware security
  - Must be run only once
- Faults not detected by HW must be detected by SW



Methodology Overview

Principle

- Security-critical systems implement **HW protections**
- **Preliminary evaluation** of the hardware security
  - Must be run only once
- Faults not detected by HW must be detected by SW

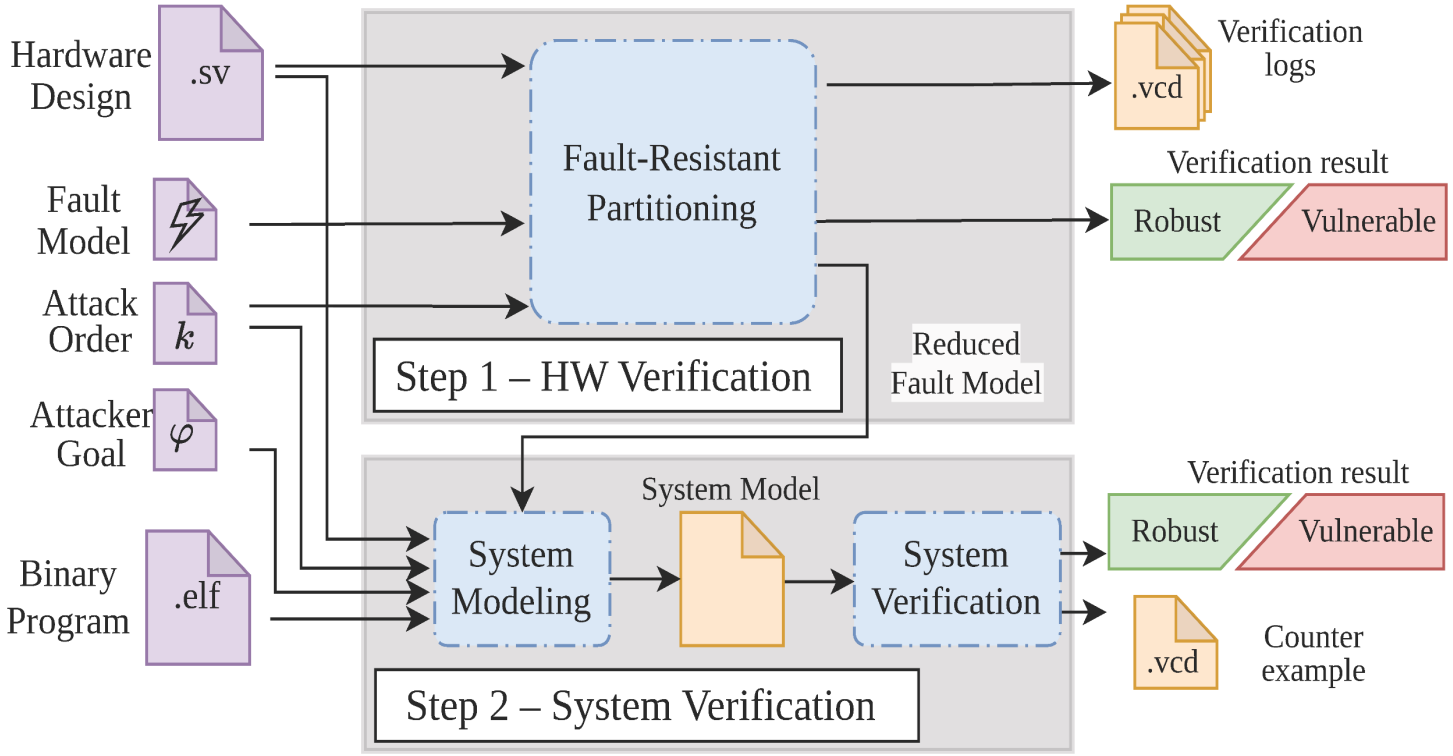
Requirements

➢ Need formal HW security guarantees to remove ineffective faults during SW co-verification

- Existing HW Techniques
  - Compare *fault-free trace* (*golden*) vs. *faulty trac*
  - Only provide **bounded** guarantees → **Insufficient**

- Faults can have long-term effects: e.g., hidden in microarchitectural registers

➢ Need **unbounded guarantees**

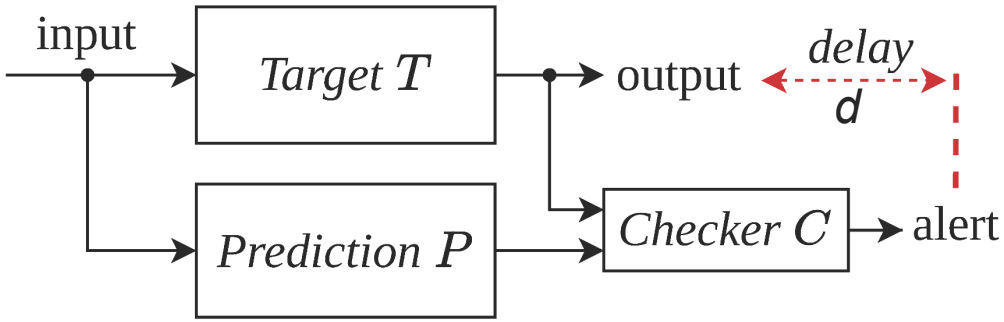


Step 1 — HW Verification

Concurrent Error Detection Scheme

- Spatial redundancy, e.g.,
  - Duplication
  - Triplication
- Informational redundancy e.g.,
  - Error detection codes

➢ Raise an alert on a mismatch (with a potential delay *d*)

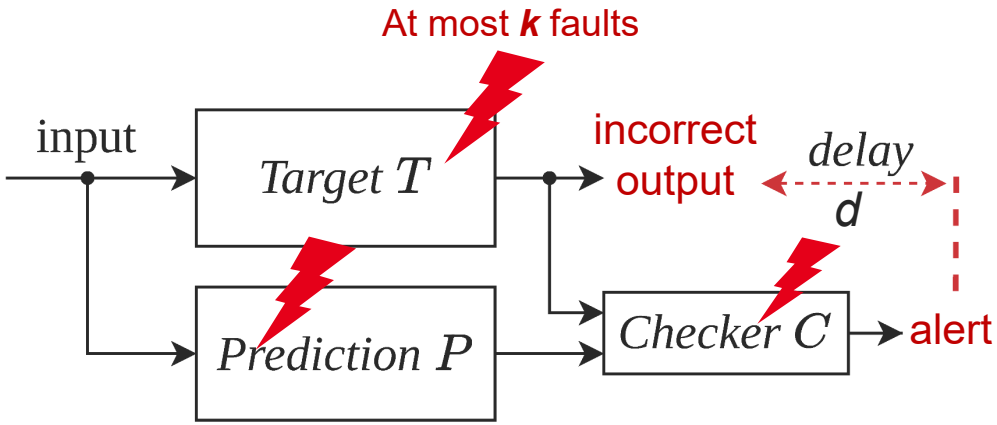


Step 1 — HW Verification

Concurrent Error Detection Scheme

- Spatial redundancy, e.g.,
  - Duplication
  - Triplication
- Informational redundancy e.g.,
  - Error detection codes

➢ Raise an alert on a mismatch (with a potential delay *d*)



Definition (*k*-Fault Security)

Assumptions

- At most ***k*** faults are injected in the circuit

Guarantees

- Circuit's outputs are correct, or an alert is raised after at most ***d*** clock cycles

Fault-Resistant Partitioning

Contribution

- Build and prove a *fault-resistant partitioning* of registers

Definition (*k*-Fault-Resistant Partitioning)

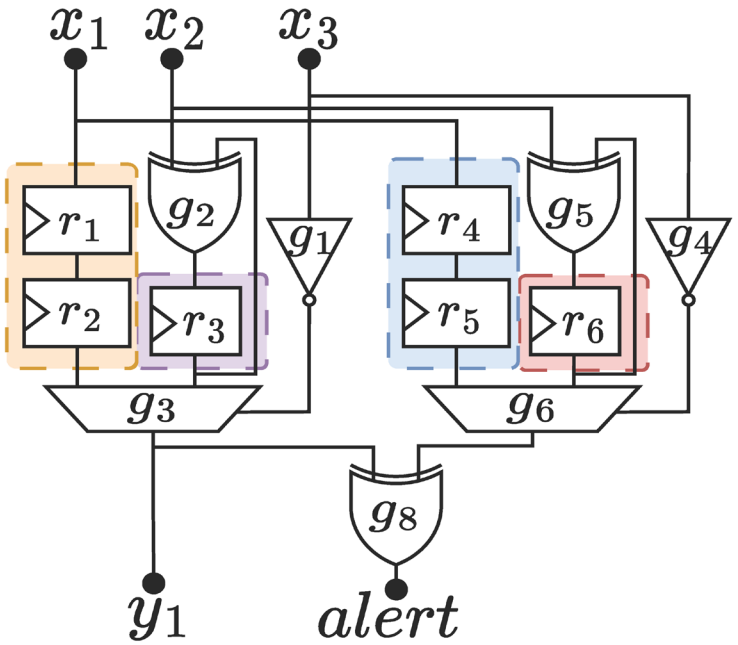
Assumptions

- Maximum budget of k faults, i.e., $k_1 + k_2 \leq k$
- At most k_1 faults in partitions at clock cycle j
- At most k_2 faults in logic gates at clock cycles $[j, j + d]$
- No alert between cycles j and $j + d$

Guarantees

- At most $k_1 + k_2$ faulty partitions at clock cycle $j + 1$
- No corrupted outputs at clock cycle j

Example with *k = 1*



Fault-Resistant Partitioning

Contribution

- Build and prove a *fault-resistant partitioning* of registers

Definition (*k*-Fault-Resistant Partitioning)

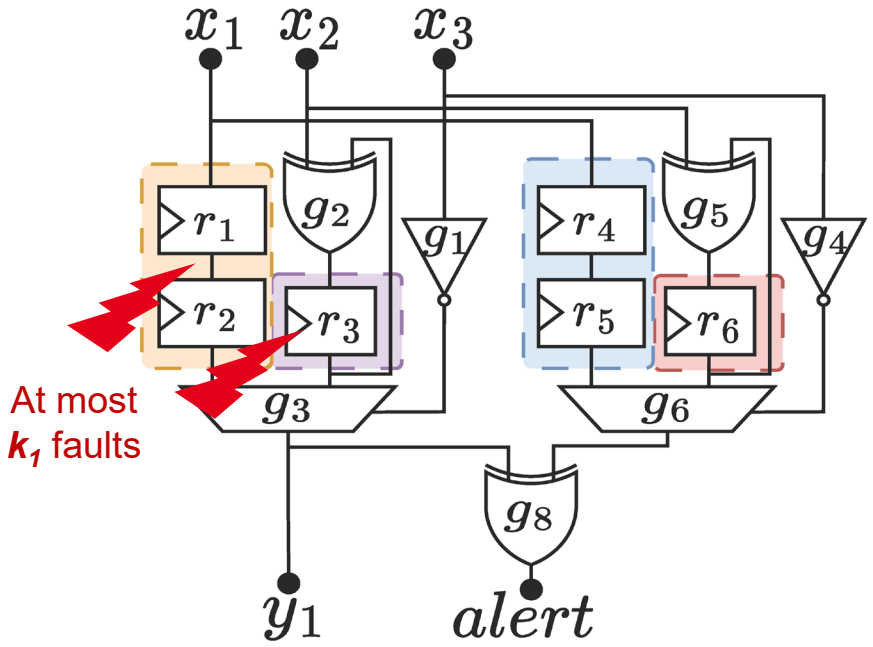
Assumptions

- Maximum budget of k faults, i.e., $k_1 + k_2 \leq k$
- At most k_1 faults in partitions at clock cycle j
- At most k_2 faults in logic gates at clock cycles $[j, j + d]$
- No alert between cycles j and $j + d$

Guarantees

- At most $k_1 + k_2$ faulty partitions at clock cycle $j + 1$
- No corrupted outputs at clock cycle j

Example with *k = 1*



Fault-Resistant Partitioning

Contribution

- Build and prove a *fault-resistant partitioning* of registers

Definition (*k*-Fault-Resistant Partitioning)

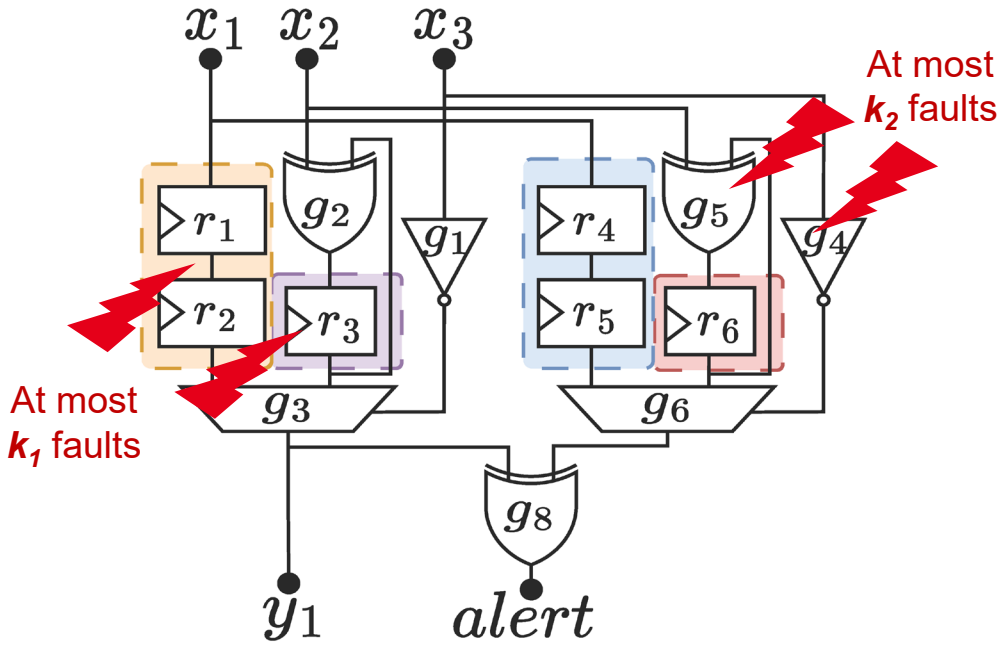
Assumptions

- Maximum budget of k faults, i.e., $k_1 + k_2 \leq k$
- At most k_1 faults in partitions at clock cycle j
- At most k_2 faults in logic gates at clock cycles $[j, j + d]$
- No alert between cycles j and $j + d$

Guarantees

- At most $k_1 + k_2$ faulty partitions at clock cycle $j + 1$
- No corrupted outputs at clock cycle j

Example with $k = 1$



Fault-Resistant Partitioning

Contribution

- Build and prove a *fault-resistant partitioning* of registers

Definition (*k*-Fault-Resistant Partitioning)

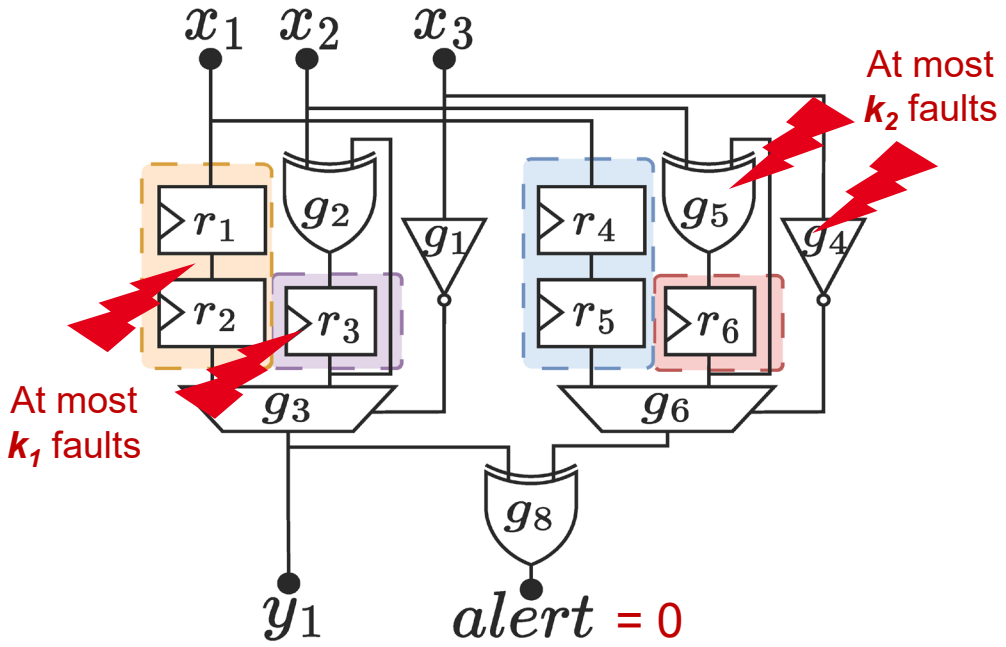
Assumptions

- Maximum budget of k faults, i.e., $k_1 + k_2 \leq k$
- At most k_1 faults in partitions at clock cycle j
- At most k_2 faults in logic gates at clock cycles $[j, j + d]$
- No alert between cycles j and $j + d$

Guarantees

- At most $k_1 + k_2$ faulty partitions at clock cycle $j + 1$
- No corrupted outputs at clock cycle j

Example with *k = 1*



Fault-Resistant Partitioning

Contribution

- Build and prove a *fault-resistant partitioning* of registers

Definition (*k*-Fault-Resistant Partitioning)

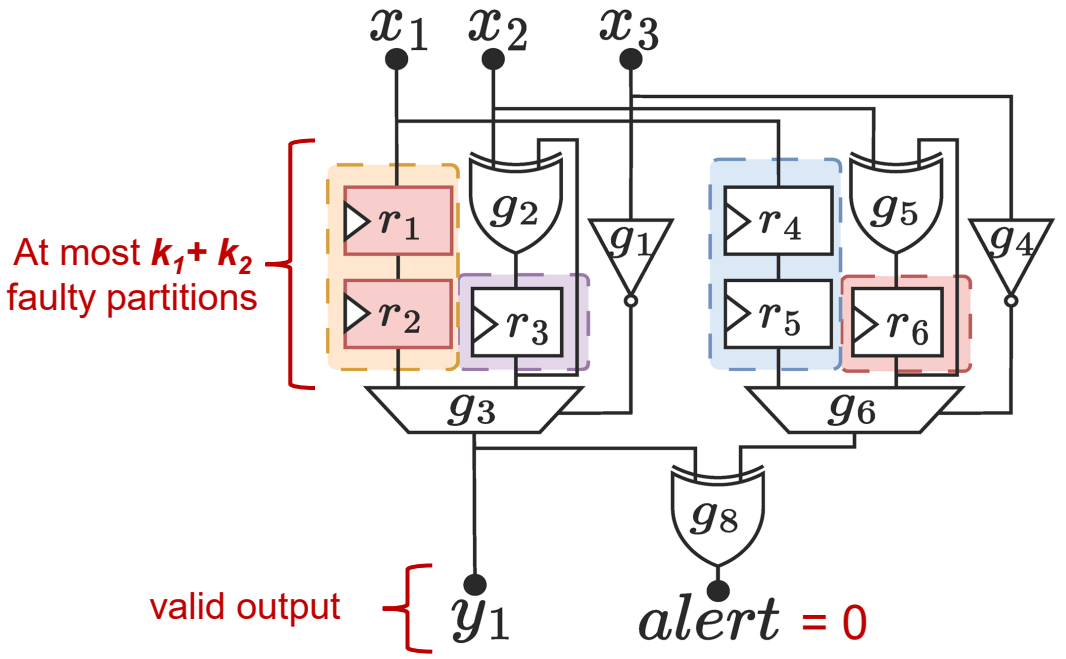
Assumptions

- Maximum budget of k faults, i.e., $k_1 + k_2 \leq k$
- At most k_1 faults in partitions at clock cycle j
- At most k_2 faults in logic gates at clock cycles $[j, j + d]$
- No alert between cycles j and $j + d$

Guarantees

- At most $k_1 + k_2$ faulty partitions at clock cycle $j + 1$
- No corrupted outputs at clock cycle j

Example with *k = 1*



Fault-Resistant Partitioning

Contribution

- Build and prove a *fault-resistant partitioning* of registers

Definition (*k*-Fault-Resistant Partitioning)

Assumptions

- Maximum budget of k faults, i.e., $k_1 + k_2 \leq k$
- At most k_1 faults in partitions at clock cycle j
- At most k_2 faults in logic gates at clock cycles $[j, j + d]$
- No alert between cycles j and $j + d$

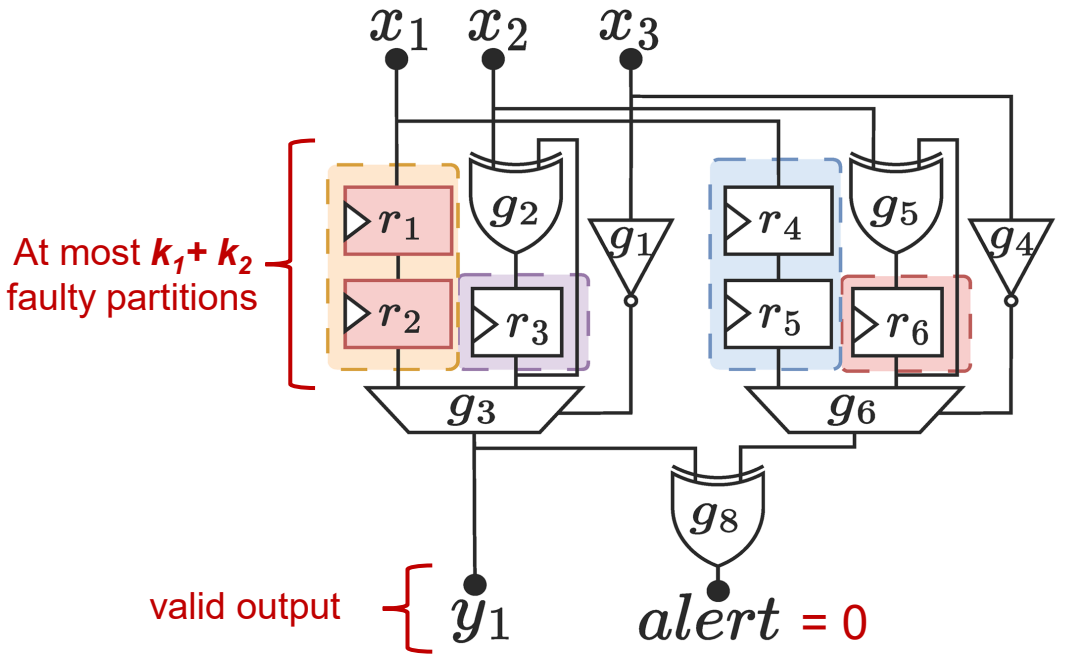
Guarantees

- At most $k_1 + k_2$ faulty partitions at clock cycle $j + 1$
- No corrupted outputs at clock cycle j

Intuition

➢ Faulty values are confined in partitions and cannot alter the circuit behavior without being detected by countermeasures.

Example with *k = 1*



Fault-Resistant Partitioning

Contribution

- Build and prove a *fault-resistant partitioning* of registers

Definition (*k*-Fault-Resistant Partitioning)

Assumptions

- Maximum budget of k faults, i.e., $k_1 + k_2 \leq k$
- At most k_1 faults in partitions at clock cycle j
- At most k_2 faults in logic gates at clock cycles $[j, j + d]$
- No alert between cycles j and $j + d$

Guarantees

- At most $k_1 + k_2$ faulty partitions at clock cycle $j + 1$
- No corrupted outputs at clock cycle j

Intuition

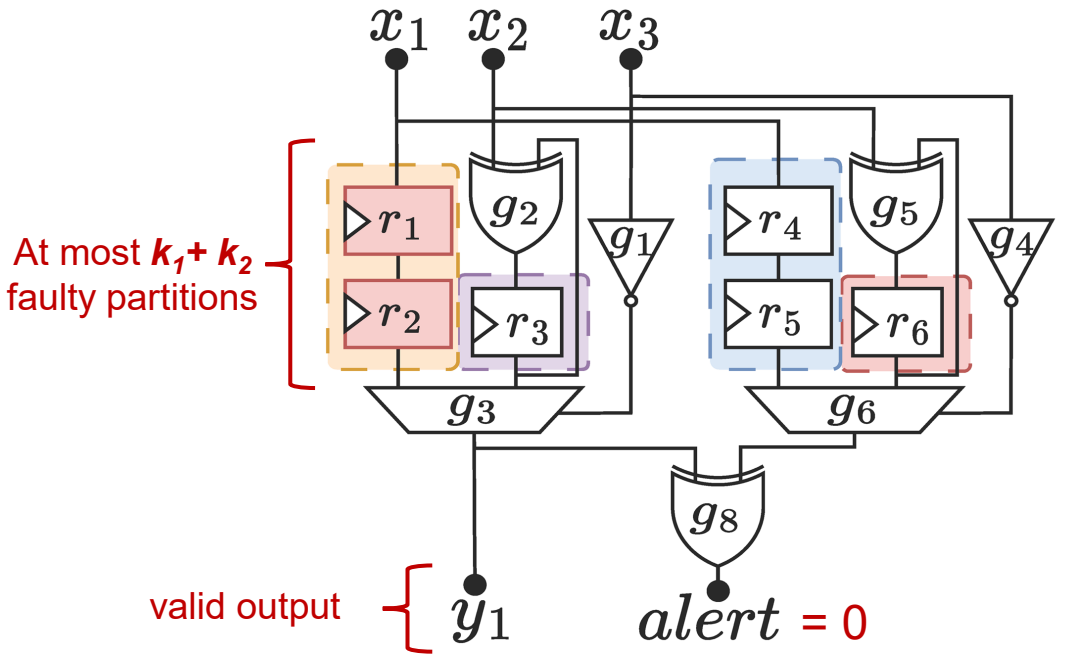
➢ Faulty values are confined in partitions and cannot alter the circuit behavior without being detected by countermeasures.

Theorem

k-fault resistant partitioning ⇒ k-fault security

➢ *Formal definitions and proofs are in the paper*

Example with *k = 1*



Advantages

- No need to unroll the circuit — fault propagation is abstracted
- Provide unbounded guarantees

Limitations

- Over-approximation of *k*-fault security
- Some circuits are *k*-fault secure, but we cannot prove it

3. Validation on Impeccable Circuits

Validation on Impeccable Circuits

Why Impeccable Circuits?

- No similar work exists on CPUs for comparison
- Validate the first step of our methodology
  - Against prior work like FIVER [RR21]
  - Evaluate verification performance
  - With multiple-fault attacks

Impeccable Circuits [AM19]

- Symmetric Bloc Ciphers (AES, LED, Simon, Skinny …) protected with Error Detection Codes (EDC).
- Designed to detect up to 3 faults (up to 7 faults for AES).
- An error signal is raised on fault detection and circuit outputs are zeroed out.

[AM19] Aghaie, Anita, et al. "Impeccable circuits." *IEEE Transactions on Computers* 2019.
[RR21] Richter-Brockmann, Jan, et al. "Fiver–robust verification of countermeasures against fault injections". *CHES 2021.*

Validation on Impeccable Circuits

Why Impeccable Circuits?

- No similar work exists on CPUs for comparison
- Validate the first step of our methodology
  - Against prior work like FIVER [RR21]
  - Evaluate verification performance
  - With multiple-fault attacks

Impeccable Circuits [AM19]

- Symmetric Bloc Ciphers (AES, LED, Simon, Skinny …) protected with Error Detection Codes (EDC).
- Designed to detect up to 3 faults (up to 7 faults for AES).
- An error signal is raised on fault detection and circuit outputs are zeroed out.

Experimental Results

- With 2 faults, we **prove** security of:
  - Skinny in **10 sec.**
  - AES in **4 hours** — FIVER took **130 hours**
- With 3 faults (assuming no faults in the checker), we **prove** security of:
  - Skinny in **40 sec.**
  - AES in **55 hours** — **never** been done before

[AM19] Aghaie, Anita, et al. "Impeccable circuits." *IEEE Transactions on Computers* 2019.
[RR21] Richter-Brockmann, Jan, et al. "Fiver–robust verification of countermeasures against fault injections". *CHES 2021.*

4. OpenTitan Evaluation

OpenTitan's Processor: Secure Ibex

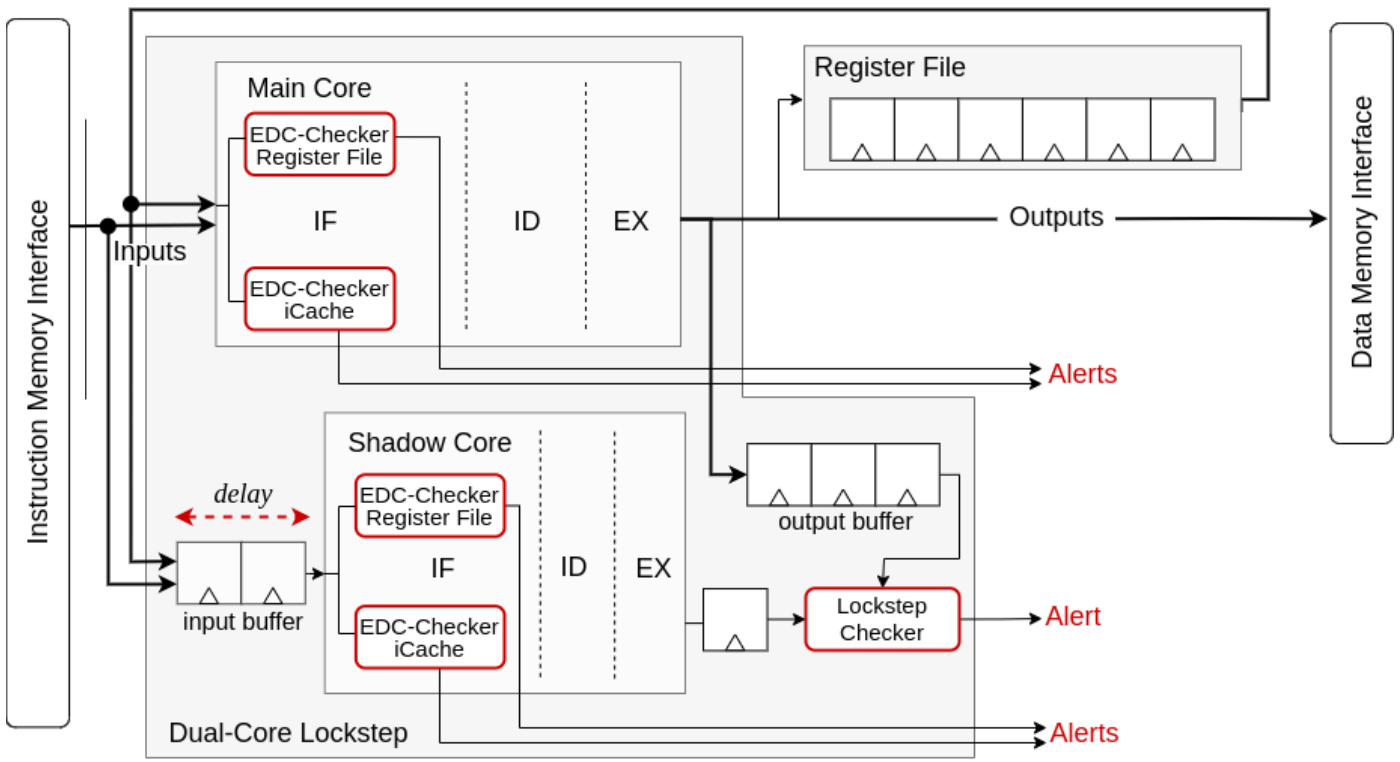


Figure - Secure Ibex Block Diagram

Secure Element OpenTitan [JR18]

opentitan

Secure-Ibex (development version) [Lo18]

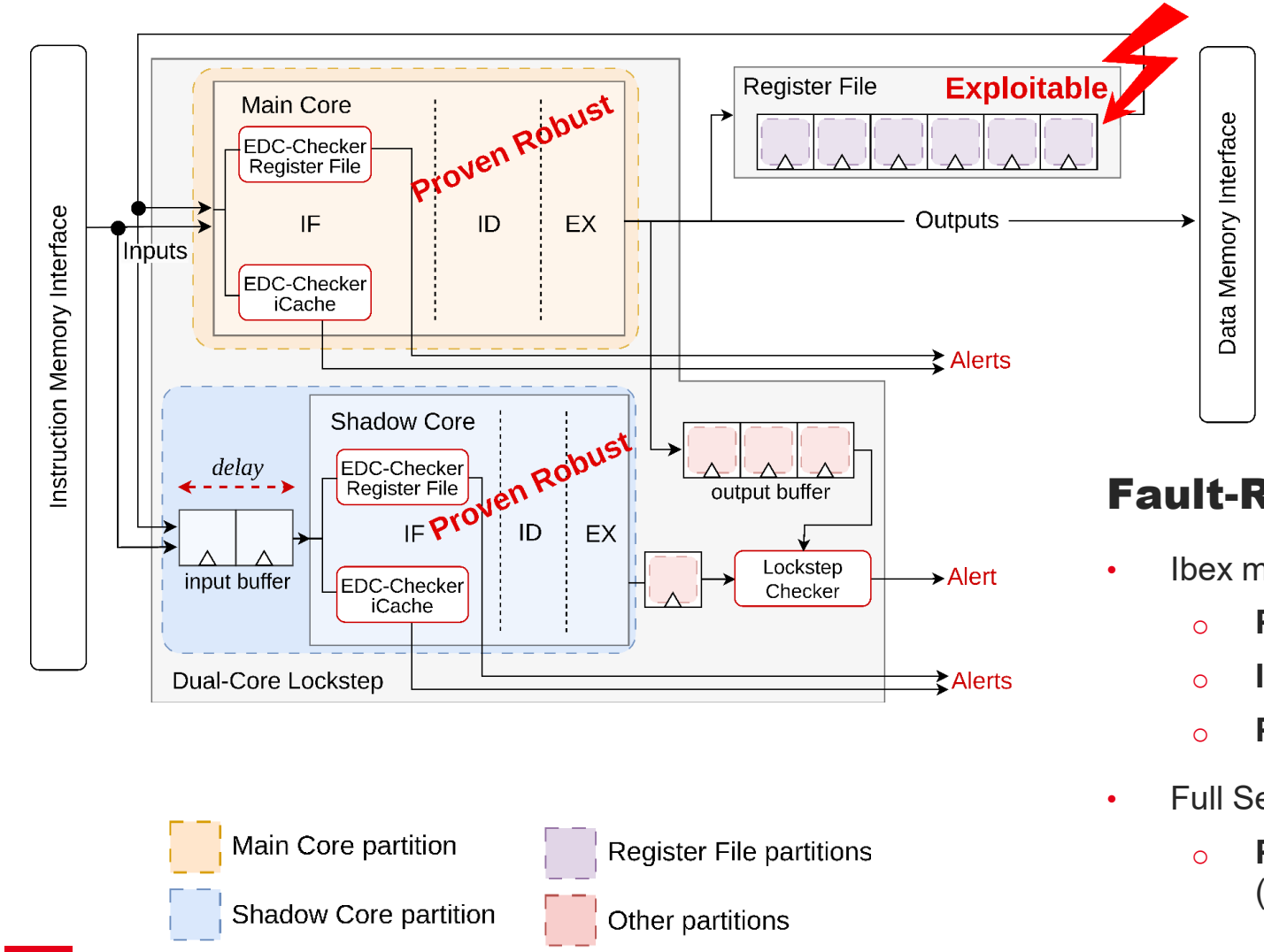
- RISC-V processor, 3 stages, in-order
- Concurrent Error Detection schemes, e.g.,
  - Dual Core Lockstep (DCLS) with delay d
  - Error Detection Codes in Register File

Fault Model

- Attacker with physical access to the processor
- Single transient bit-flip everywhere at any time

[JR18] Johnson, Scott, et al. "Titan: enabling a transparent silicon root of trust for cloud." *Hot Chips: A Symposium on High Performance Chips*. Vol. 194. 2018.
[Lo18] lowRISC. Ibex RISC-V Core github repository. https://github.com/lowRISC/ibex. Accessed: February 22, 2024.

HW Evaluation: Results



Fault-Resistant Partitioning Results

- Ibex modules
  - **Prove** 1-fault security of DCLS in **26 hours**
  - **Identify** 172 exploitable locations in Reg. File in **1 min 30**
  - **Prove** 3-seq.-fault security of Reg. File in **12 mins**
- Full Secure Ibex
  - **Prove** 1-fault security in **42 hours (+ 26 h)**
    (assuming no faults in the 172 exploitable locations identified)

System Co-Verif: SW Case Studies + Results

First Stage of Secure Boot

- *Provided by the OpenTitan project*

- Check authenticity and integrity of the next boot stage
- Implement software protections e.g.,
  - Step counters
  - Test duplications

- Goal: **Bypass memory signature check**
- # instructions: **2 526**
- # faults: **122 048**
- Performance: **2.5 hours** (8 threads)
- Results: **Secure**

[DP16] Dureuil, Louis, et al. "FISSC: A fault injection and simulation secure collection." *SAFECOMP* 2016.
[Ko19] kokke. Tiny AES, release 1.0. https://github.com/kokke/tiny-AES-c, 2019. Accessed: February 22, 2024.

System Co-Verif: SW Case Studies + Results

First Stage of Secure Boot

- *Provided by the OpenTitan project*

- Check authenticity and integrity of the next boot stage
- Implement software protections e.g.,
  - Step counters
  - Test duplications

- Goal: **Bypass memory signature check**
- # instructions: **2 526**
- # faults: **122 048**
- Performance: **2.5 hours** (8 threads)
- Results: **Secure**

VerifyPIN [DP16]

- ***Not*** *provided by the OpenTitan project*

- Compare two PINs for authentication
- 8 versions with various mix of protections

- Goal1: **Bypass authentication**
- Goal2: **Increase max number of tries**
- # instructions: **187**
- # faults: **7 424**
- Performance: **6 mins** (1 thread)
- Results: **Insecure**

DFA on tiny AES [Ko19]

- ***Not*** *provided by the OpenTitan project*

- SW implementation of AES

- Goal1: **DFA on key schedule**
- # instructions: **221**
- # faults: **5 760**
- Performance: **7 mins** (2 threads)
- Results: **Insecure**

- Goal2: **DFA on 7th AES round**
- # instructions: **1 144**
- # faults: **38 912**
- Performance: **29 mins** (8 threads)
- Results: **Insecure**

[DP16] Dureuil, Louis, et al. "FISSC: A fault injection and simulation secure collection." *SAFECOMP* 2016.
[Ko19] kokke. Tiny AES, release 1.0. https://github.com/kokke/tiny-AES-c, 2019. Accessed: February 22, 2024.

4. Conclusion

Conclusion

Co-verification Methodology

- **Two-step** methodology to evaluate fault security
  - Step 1 — **Preliminary analysis** of HW countermeasures
  - Step 2 — **Remaining faults** not detected by the HW are evaluated with the SW

Step 1 — Fault-Resistant Partitioning

- Provide **unbounded security guarantees** which are crucial for SW co-verification
- **Outperform** state-of-the-art solvers like FIVER
- First work to evaluate AES against **3 faults**

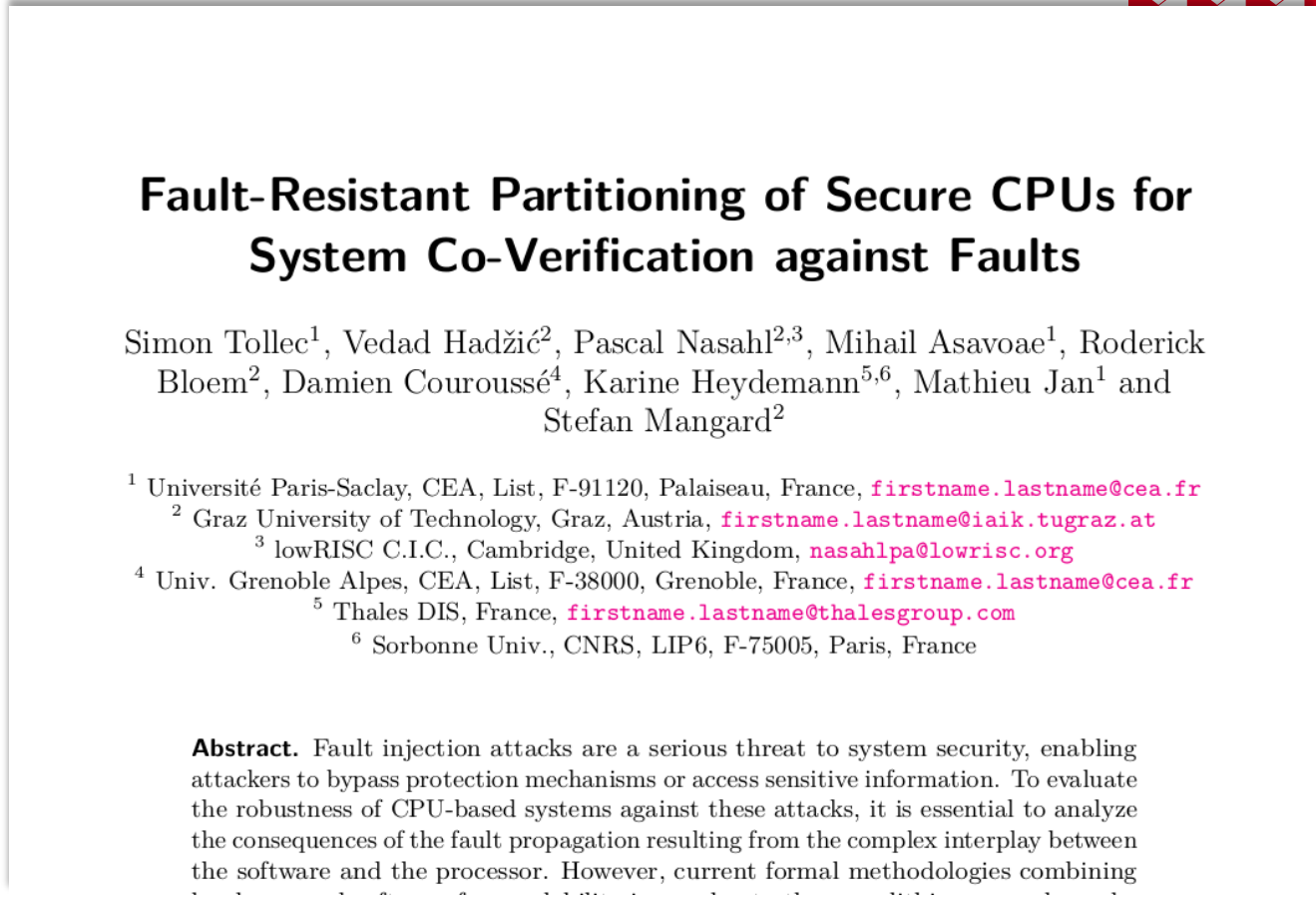
Step 2 — System co-verification

- Address **previously intractable** software verification of thousands of instructions
- First work to prove the security of the first stage of a **secure boot against fault attacks**

Thank you

Questions?

Come and see my poster



Fault-Resistant Partitioning of Secure CPUs for System Co-Verification against Faults

Simon Tollec[1], Vedad Hadžić[2], Pascal Nasahl[2,3], Mihail Asavoae[1], Roderick Bloem[2], Damien Couroussé[4], Karine Heydemann[5,6], Mathieu Jan[1] and Stefan Mangard[2]

[1] Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France, firstname.lastname@cea.fr
[2] Graz University of Technology, Graz, Austria, firstname.lastname@iaik.tugraz.at
[3] lowRISC C.I.C., Cambridge, United Kingdom, nasahlpa@lowrisc.org
[4] Univ. Grenoble Alpes, CEA, List, F-38000, Grenoble, France, firstname.lastname@cea.fr
[5] Thales DIS, France, firstname.lastname@thalesgroup.com
[6] Sorbonne Univ., CNRS, LIP6, F-75005, Paris, France

Abstract. Fault injection attacks are a serious threat to system security, enabling attackers to bypass protection mechanisms or access sensitive information. To evaluate the robustness of CPU-based systems against these attacks, it is essential to analyze the consequences of the fault propagation resulting from the complex interplay between the software and the processor. However, current formal methodologies combining

Bibliography

[JA94] Jenn, Eric, et al. "Fault injection into VHDL models: the MEFISTO tool." *Predictably Dependable Computing Systems*. Springer Berlin Heidelberg, 1995.

[ST97] Sieh, Volkmar, Oliver, Frank Balbach. "VERIFY: Evaluation of reliability using VHDL-models with embedded fault descriptions." *27th International Symposium on Fault Tolerant Computing*. 1997.

[BN08] Bosio, Alberto, and Giorgio Di Natale. "Lifting: A flexible open-source fault simulator." *2008 17th Asian Test Symposium*. IEEE, 2008.

[GS21] Grycel, Jacob, and Patrick Schaumont. "Simplifi: hardware simulation of embedded software fault attacks." *Cryptography* 5.2 (2021): 15.

[BG17] Burchard, Jan, et al. "Autofault: towards automatic construction of algebraic fault attacks." *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2017.

[AW20] Arribas, Victor, et al. "Cryptographic fault diagnosis using VerFI." *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020.

[RR21] Richter-Brockmann, Jan, et al. "Fiver–robust verification of countermeasures against fault injections." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021).

[NO22] Nasahl, Pascal, et al. "SYNFI: pre-silicon fault analysis of an open-source secure element." *arXiv preprint arXiv:2205.04775* (2022).

[PN08] Pattabiraman, Karthik, et al. "SymPLFIED: Symbolic program-level fault injection and error detection framework." *International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 2008.

[Ri20] Riscure. Fisim. https://github.com/Riscure/FiSim. Accessed: February 22, 2024.

[HG21] Hauschild, Florian, et al. "ARCHIE: A QEMU-Based framework for architecture-independent evaluation of faults." *Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, 2021.

[HS21] Hoffmann, Max, et al. "ARMORY: fully automated and exhaustive fault simulation on ARM-M binaries." *IEEE Transactions on Information Forensics and Security* 16 (2020).

[PM14] Potet, Marie-Laure, et al. "Lazart: A symbolic approach for evaluation the robustness of secured codes against control flow injections." *International Conference on Software Testing, Verification and Validation*. IEEE, 2014.

[GJ17] Given-Wilson, Thomas, et al. "An automated formal process for detecting fault injection vulnerabilities in binaries and case study on present." *2017,* IEEE, 2017.

[BH19] Bréjon, Jean-Baptiste, et al. "Fault attack vulnerability assessment of binary code." *Proceedings of the Sixth Workshop on Cryptography and Security in Computing Systems*. 2019.

[GH23] Gicquel, Antoine, et al. "SAMVA: static analysis for multi-fault attack paths determination." *Workshop on Constructive Side-Channel Analysis and Secure Design*, 2023.

[DB23] Ducousso, Soline, Sébastien Bardin, and Marie-Laure Potet. "Adversarial Reachability for Program-level Security Analysis." *ESOP*. 2023.

[TA23] Tollec, Simon, et al. " µArchiFI: Formal Modeling and Verification Strategies for Microarchitectural Fault Injections." Formal Methods in Computer Aided Design (FMCAD), 2023.

[DP16] Dureuil, Louis, et al. "FISSC: A fault injection and simulation secure collection." *Computer Safety, Reliability, and Security: 35th International Conference, SAFECOMP* 2016.

[kok19] kokke. Tiny AES. https://github.com/kokke/tiny-AES-c. Accessed: February 22, 2024.

[TG24] Tan,Huiyu, et al. "SAT-based Formal Fault-Resistance Verification of Cryptographic Circuits." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2024).

Implementation 1/2

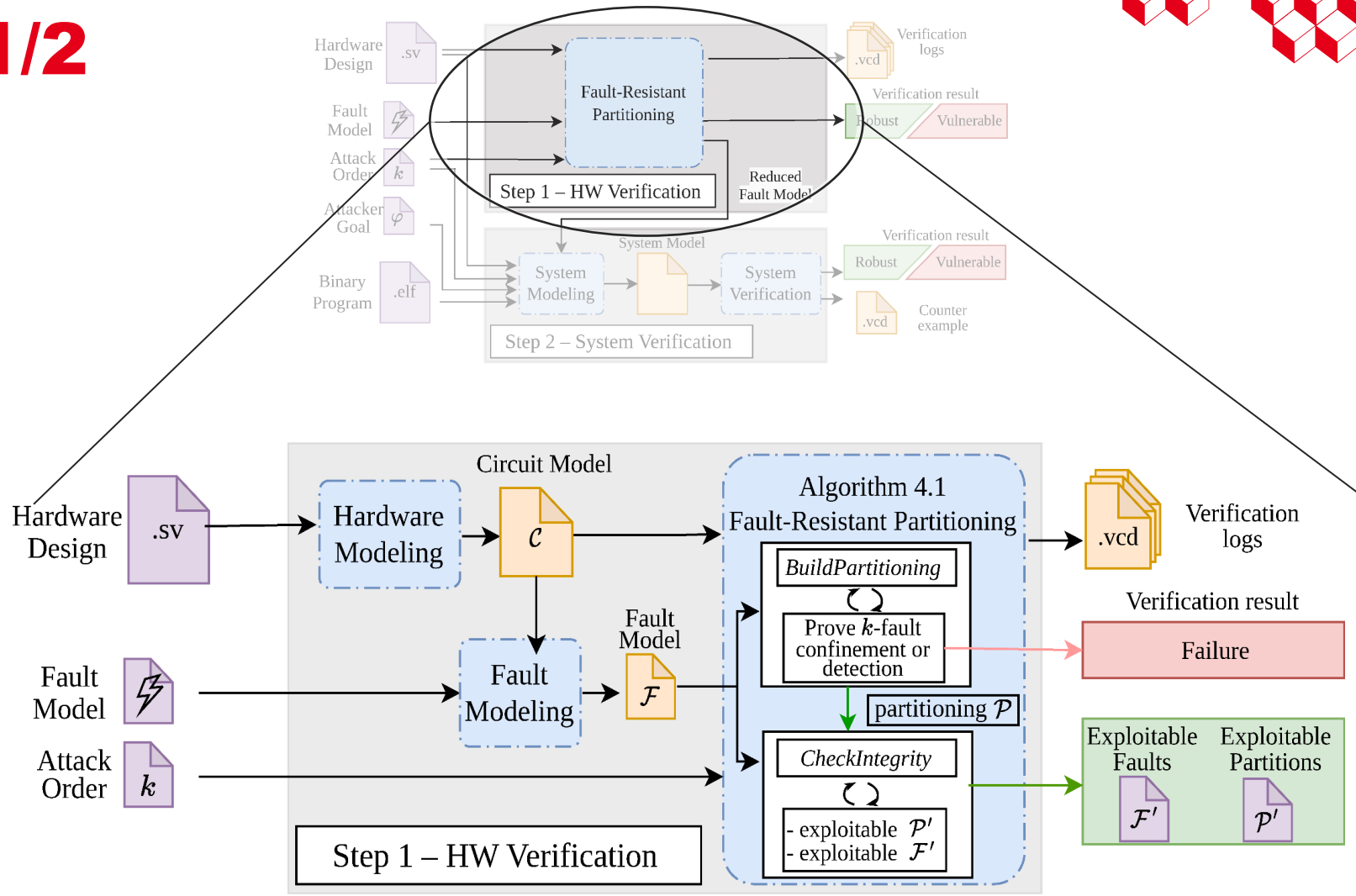
Fault-Resistant Partitioning

1. *BuildPartitioning* procedure
2. *CheckIntegrity* procedure

➢ Algorithm outputs are **unprotected/ exploitable** faults

- Rely on CaDiCaL SAT solver [Bi20]
- About 4 000 lines of code
- Open-source:

https://github.com/CEA-LIST/Fault-Resistant-Partitioning

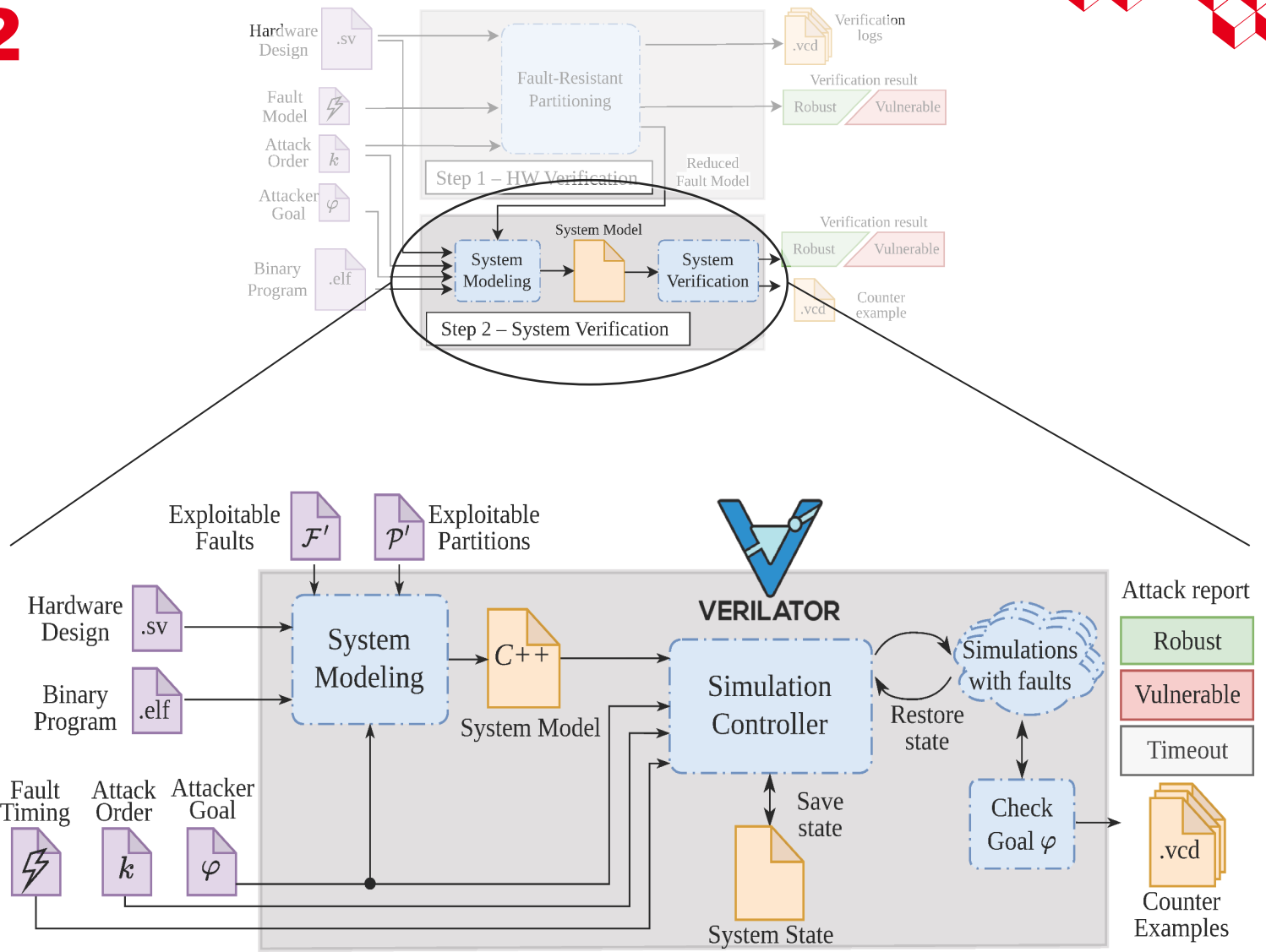


[Bi20] Biere, Armin « CaDiCaL SAT solver ». https://github.com/arminbiere/cadical. 2020. Accessed February 2024.

Implementation 2/2

System Co-Verification

- Evaluate whether exploitable faults with SW
- Based on the Verilator environment
- Exhaustively simulates exploitable faults



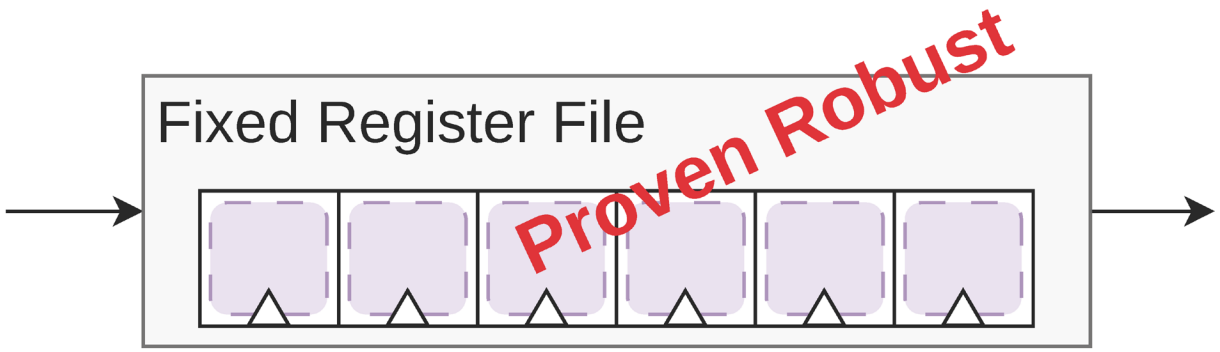
Fix of the Register File

Vulnerability Report

- 172 exploitable faults — allow reading from an incorrect register location
- We **reported the vulnerability** to the OpenTitan project
- They acknowledged our findings

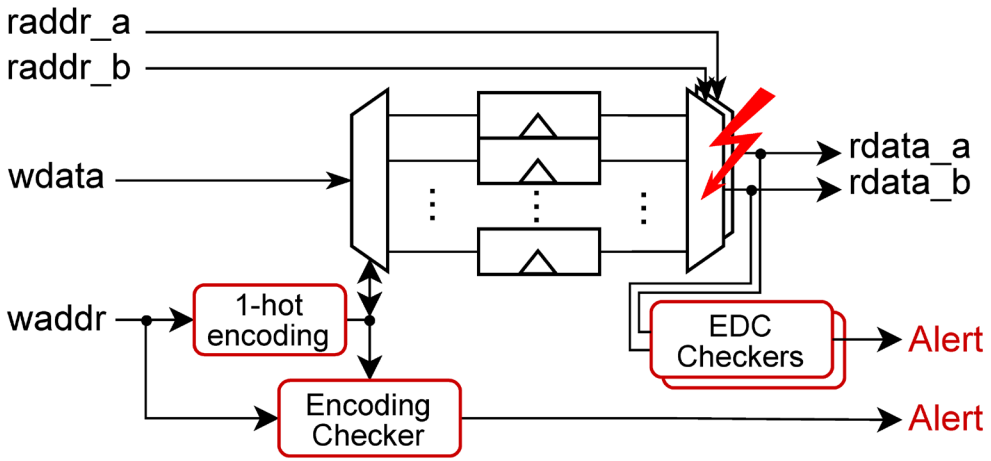
Vulnerability Fix

- We **proposed a security fix** and formally **prove it** using our methodology
- Our fix was integrated into the OpenTitan project
- **Secure Ibex** is now **proven 1-fault secure** unconditionally of the executed software

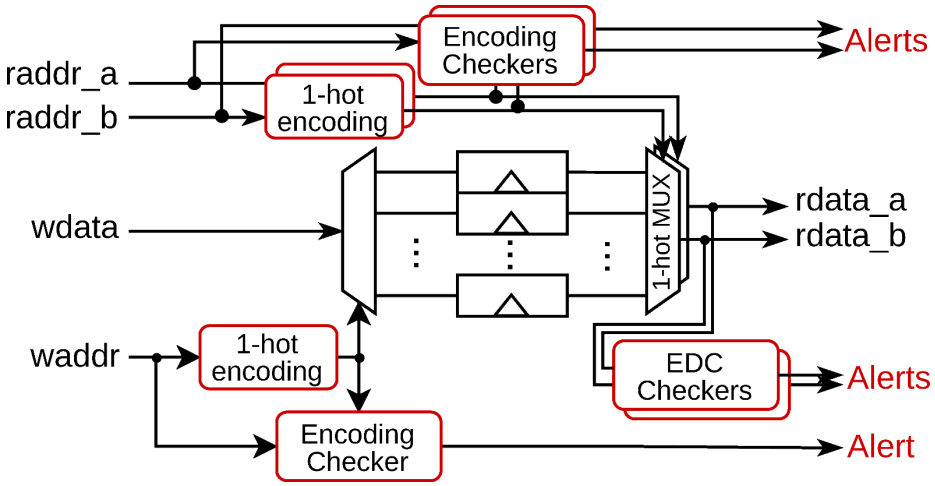


Hardware Fix + Prove

Register File Vulnerability



Register File Security Fix



Fault Propagation and Hidden Faults