Exploration of Fault Effects on Formal RISC-V Microarchitecture Models*

Simon Tollec[1], Mihail Asavoae[1], Damien Couroussé[2], Karine Heydemann[3,4] and Mathieu Jan[1]

[1]Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
[2]Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France
[3]Sorbonne Univ., CNRS, LIP6, F-75005, Paris, France
[4]Thales DIS, France

GT MFS, March 29, 2023

*Published in *2022 Workshop on Fault Diagnosis and Tolerance in Cryptography*

1. Background on Fault Injection (FI) Attacks
2. Motivating Example and Goals
3. Contributions: Formal Verification Workflow
4. Use Case: CV32E40P and VerifyPIN

1 Background on Fault Injection (FI) Attacks

2 Motivating Example and Goals

3 Contributions: Formal Verification Workflow

4 Use Case: CV32E40P and VerifyPIN

Studying FIs on a Processor Executing a SW Program

Fault injection (FI) attacks

- Applying abnormal execution conditions
  - high temperature
  - electromagnetic radiation
- Induce computational errors
- Lead to an undesired behaviour

Create vulnerabilities in the system

- Retrieve sensitive data
- Acquire execution privilege

Studying fault injections

- Develop methodologies to analyze systems' security
- Develop countermeasures

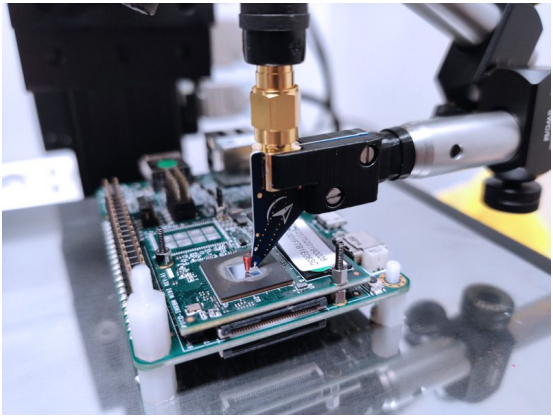
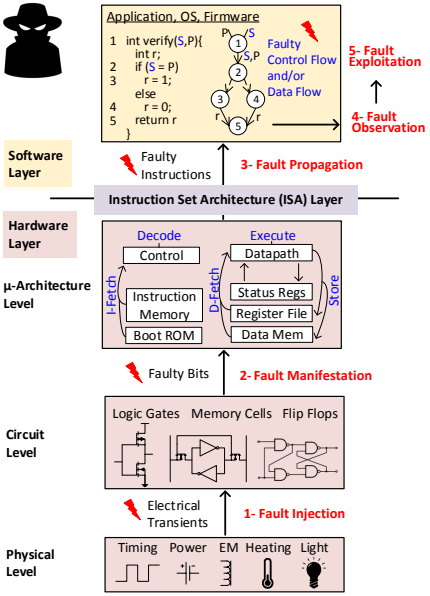


Photo credit: `https://eshard.com`

Basic Flow of FIs when Targetting a Secure Embedded Software



Propagations of the FI in the system

- Different abstraction layers involved
- *Circuit-level:* describe the initial effect of the FI
- *Software-level:* observe the consequences of the fault

FIs' effects depend on the executing context

- FIs can have no effect
- FIs can manifest after an unknown amount of time

Figure: Yuce, B., Schaumont, P., & Witteman, M. (2018). *Fault attacks on secure embedded software: Threats, design, and evaluation.* Journal of Hardware and Systems Security.

1. Background on Fault Injection (FI) Attacks
2. Motivating Example and Goals
3. Contributions: Formal Verification Workflow
4. Use Case: CV32E40P and VerifyPIN

Motivating Examples

Fault in the Prefetch Buffer:

illustrated on a RISC-V Core (IF stage)

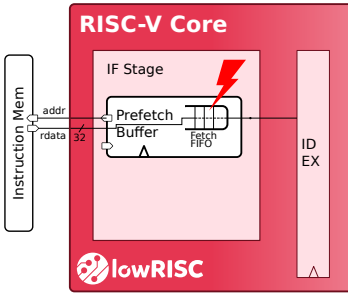


Image credit:
`https://github.com/lowRISC/ibex`

What is the Prefetch Buffer (PFB)?

- Reduce latency due to memory accesses
- Store a small number of instructions in a FIFO
- Hardware optimization invisible at the SW level

Motivating Examples

Fault in the Prefetch Buffer:

illustrated on a RISC-V Core (IF stage)

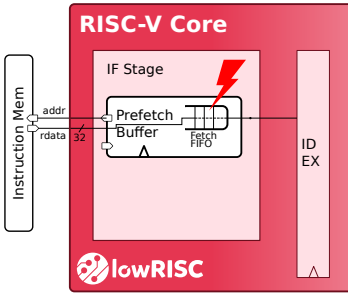


Image credit:
`https://github.com/lowRISC/ibex`

What is the Prefetch Buffer (PFB)?

- Reduce latency due to memory accesses
- Store a small number of instructions in a FIFO
- Hardware optimization invisible at the SW level

Fault Effects in the Prefetch Buffer:

1. **Immediate effect**: replay the PFB instructions
2. **Recurring effect**: incorrect executing order of instructions
3. **Long-term effect**: corruption of the next branch target

➔ *Resulting effect:* a combination of all of these effects

➔ Strongly depends on the internal state of the μarchitecture.

Motivating Examples

Fault in the Prefetch Buffer:

illustrated on a RISC-V Core (IF stage)

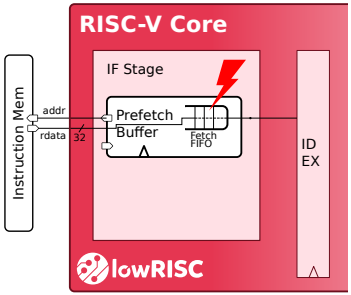


Image credit:
`https://github.com/lowRISC/ibex`

This effect cannot be analyzed with:

- **HW analysis**: Difficult to give meaning to the wrong behavior of the PFB
- **SW analysis**: Would not have detected the effect and is still difficult to model *a postreriori*

Need to consider the SW and the HW together:

- **HW** $\rightarrow$ the execution platform and fault models
- **SW** $\rightarrow$ the semantics of FIs with the ISA
  + makes possible to interpret their consequences

Motivating Examples

Fault in the Prefetch Buffer:

illustrated on a RISC-V Core (IF stage)

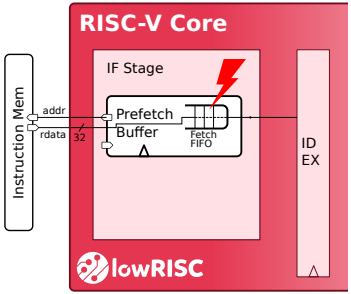


Image credit:
`https://github.com/lowRISC/ibex`

What do we need to model the system?

- µArchitecture implementation details
  - Data-path
  - Control-path
- Fault model
  - Location
  - Timing
  - Effect
  - Multiplicity

Motivating Examples

Fault in the Prefetch Buffer:

illustrated on a RISC-V Core (IF stage)

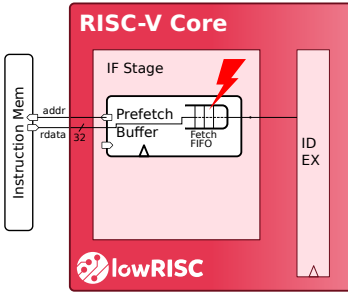


Image credit:
`https://github.com/lowRISC/ibex`

What do we need to model the system?

- µArchitecture implementation details
  - Data-path
  - Control-path
- Fault model
  - Location
  - Timing
  - Effect
  - Multiplicity
- Software program
- Security property

➔ *Chosen system modeling level:* Cycle-accurate, Word-level

Motivating Examples

Fault in the Prefetch Buffer:

illustrated on a RISC-V Core (IF stage)

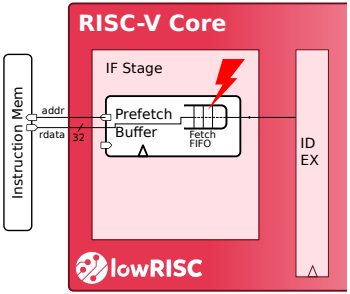


Image credit:
`https://github.com/lowRISC/ibex`

What do we need to model the system?

- µArchitecture implementation details
  - Data-path
  - Control-path
- Fault model
  - Location
  - Timing
  - Effect
  - Multiplicity
- Software program
- Security property

➔ *Chosen system modeling level:* Cycle-accurate, Word-level

Verification Techniques Requirements:

- *Exhaustiveness:* find corner case vulnerabilities (like PFB)
- *Unrolling the system:* observe the fault propagation
- *Difficult to induce invariants:* due to the transient nature of faults

➔ *Bounded verification techniques:* e.g., Bounded Model Checking

1. Background on Fault Injection (FI) Attacks
2. Motivating Example and Goals
3. Contributions: Formal Verification Workflow
4. Use Case: CV32E40P and VerifyPIN

Contributions

Goal:
Formal modeling of the SW/HW system to analyze microarchitectural fault effects on the software security

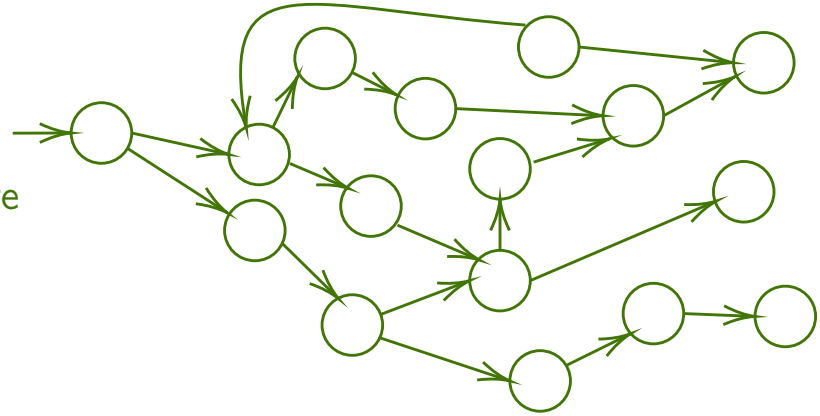
Contributions: Automated formal modeling of HW and SW

- ➔ For exploring microarchitectural fault effects on SW security
- ➔ For analyzing the robustness of HW or SW countermeasures

Transition System to model HW, SW and FIs

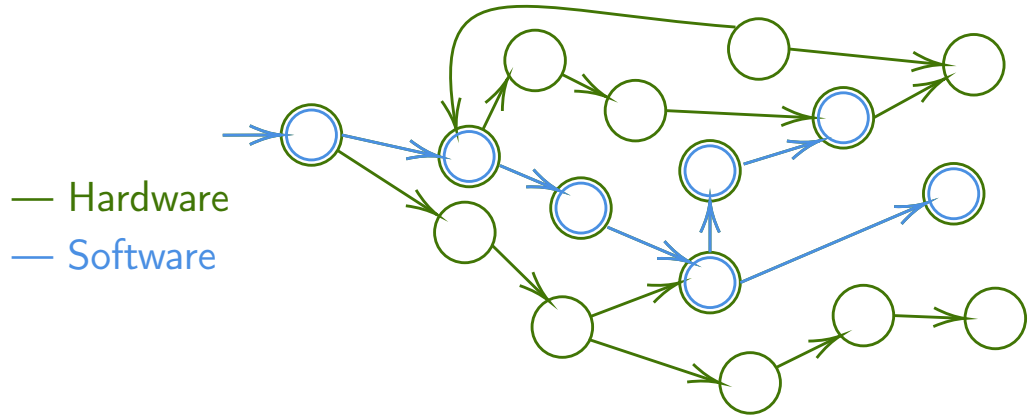
Modeling Steps

— Hardware



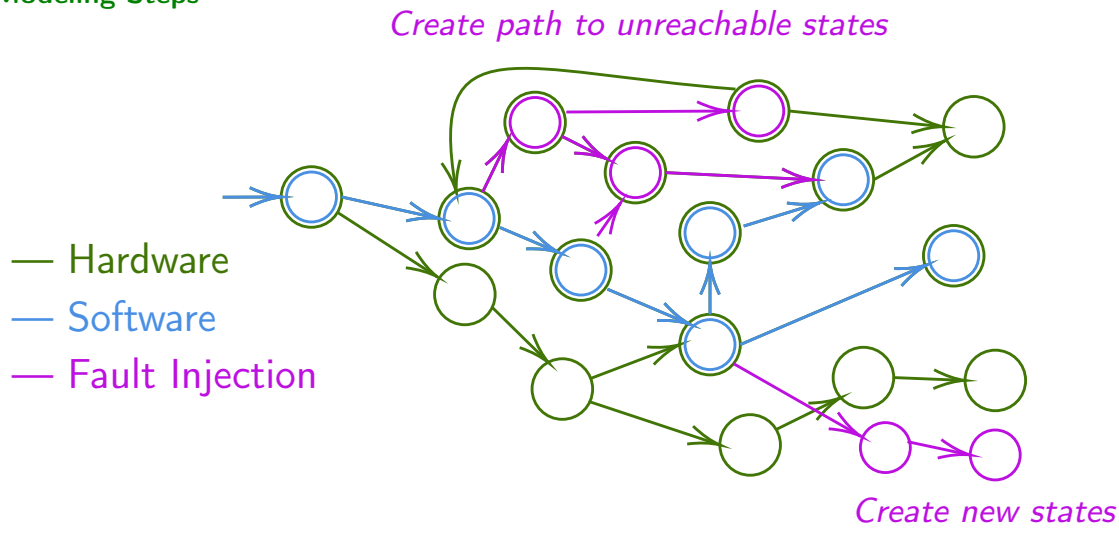
Transition System to model HW, SW and FIs

Modeling Steps



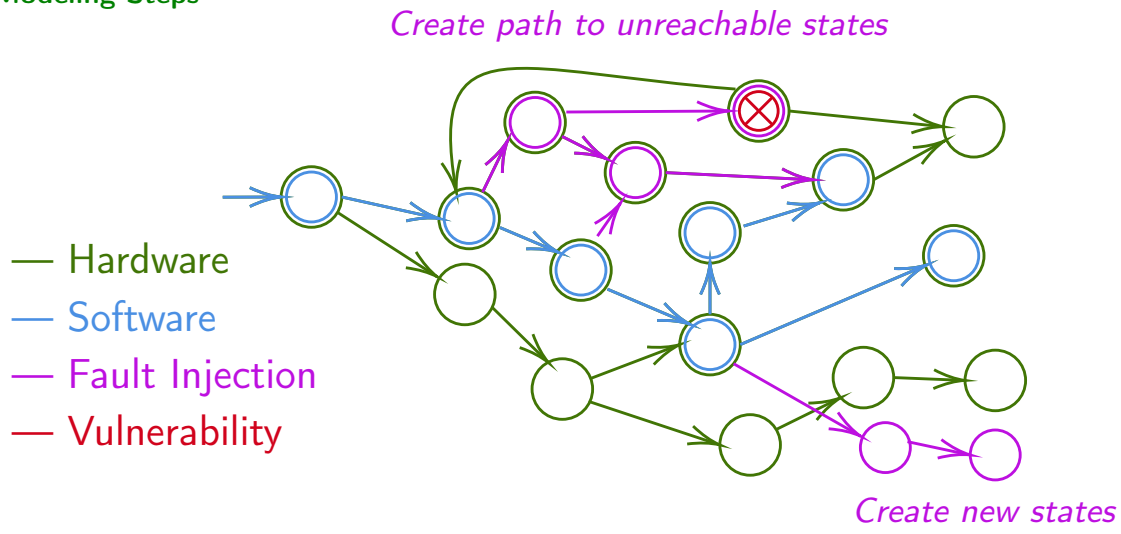
Transition System to model HW, SW and FIs

Modeling Steps



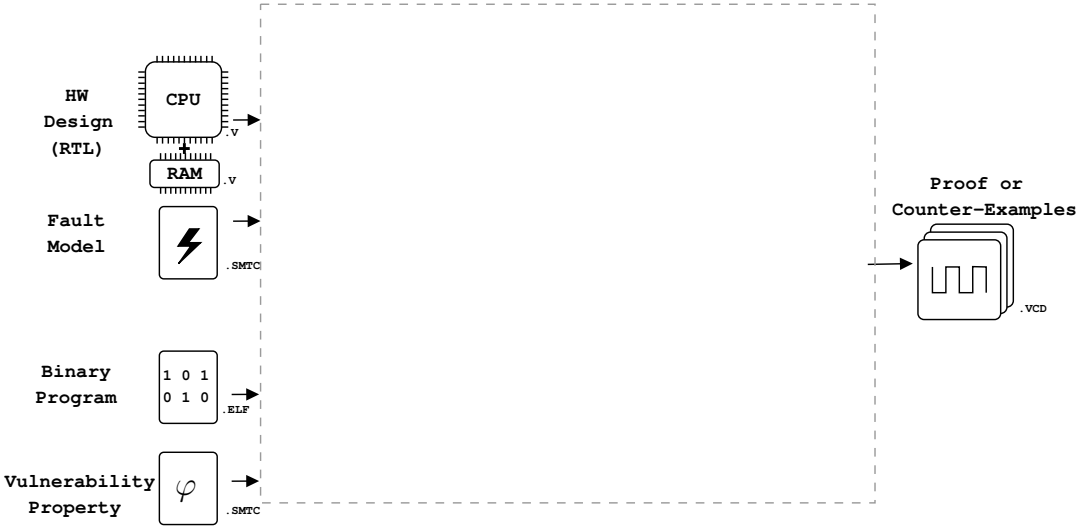
Transition System to model HW, SW and FIs

Modeling Steps



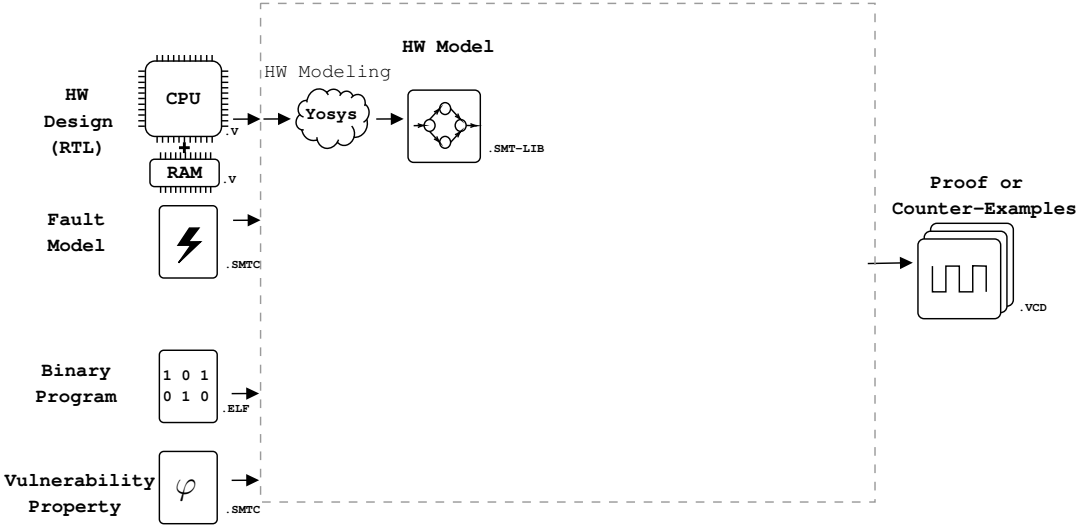
Workflow: Modeling Steps

Inputs / Outputs



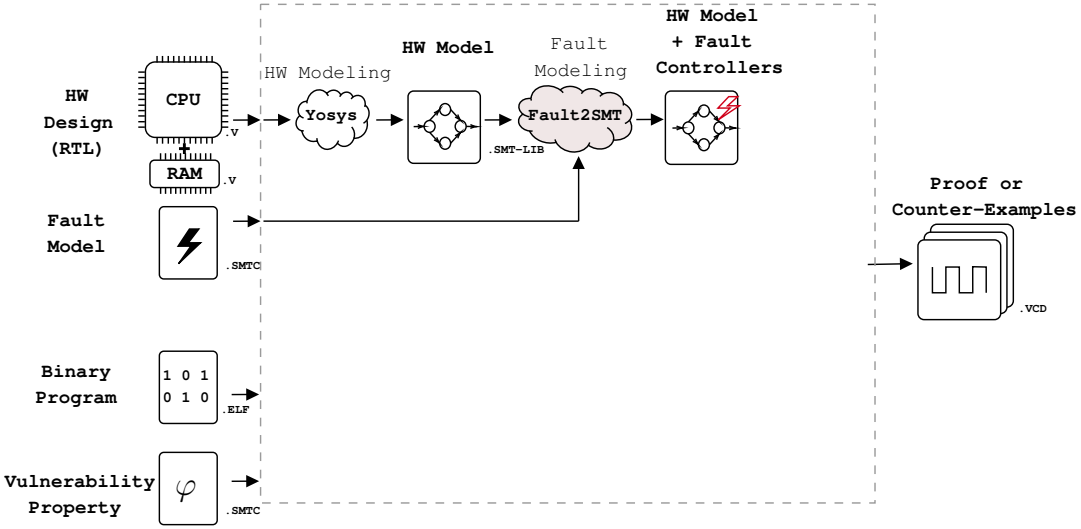
Workflow: Modeling Steps

Hardware Modeling



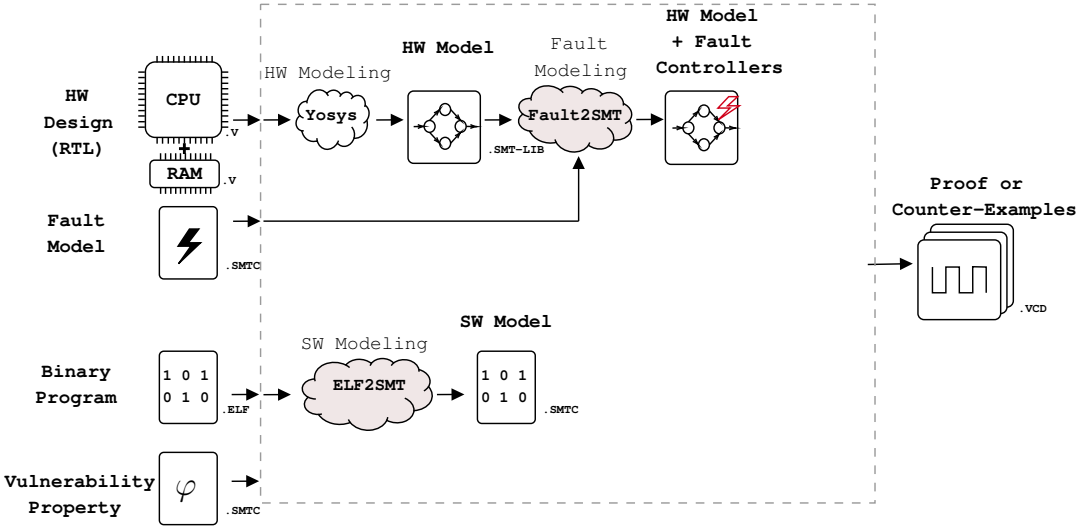
Workflow: Modeling Steps

Fault Modeling



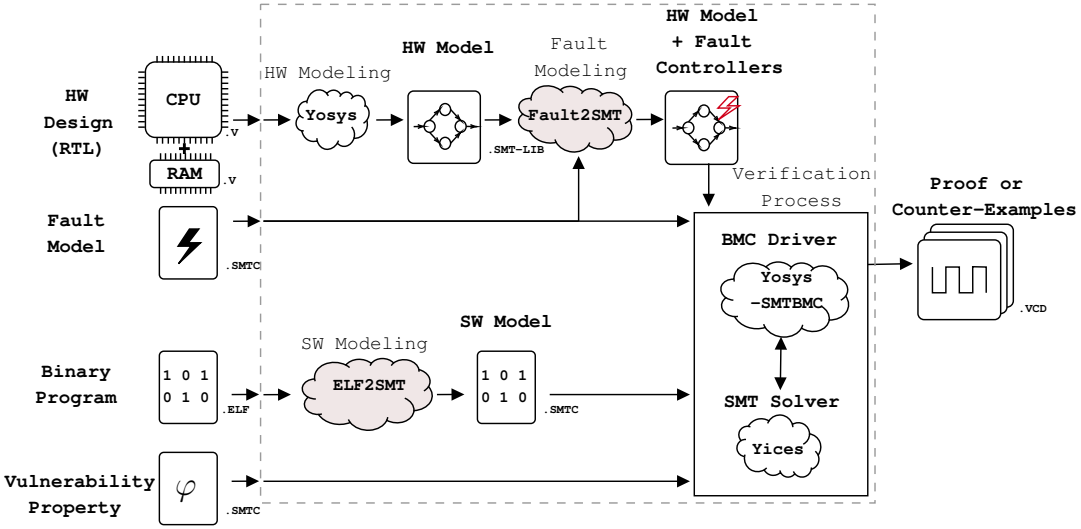
Workflow: Modeling Steps

Software Modeling



Workflow: Modeling Steps

Property Specification



1 Background on Fault Injection (FI) Attacks

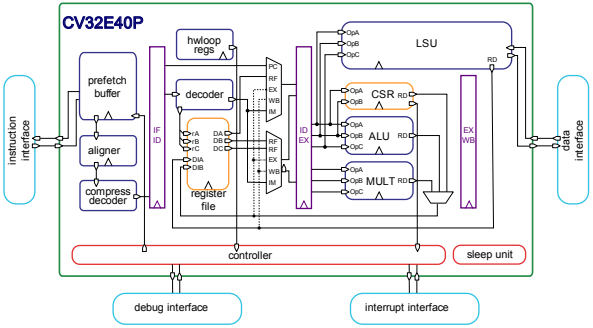
2 Motivating Example and Goals

3 Contributions: Formal Verification Workflow

4 Use Case: CV32E40P and VerifyPIN

Hardware Part

CV32E40P (RISCY)



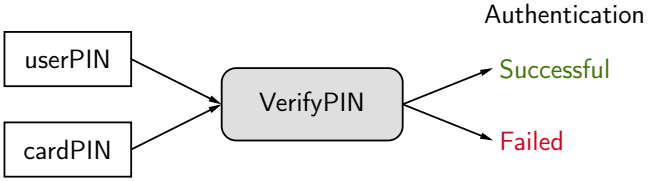
- Standard version [CV32E40P]
- Hardened version [Chamelot, 2022]
  - Control flow integrity
  - Code integrity
  - Execution integrity

Microarchitectural Fault Model

- Single fault injection
- During the whole program
- Everywhere in the circuit
- Symbolic fault effect

Software Part

VerifyPIN



- Standard version [Dureuil (FISSC), 2016]
- Versions implementing SW countermeasures
  - Constant iteration number loop
  - Inline function calls
  - Duplication of critical tests

```
Compare {
  for (i = 0; i < 4; i++) {
    if (userPIN[i] != cardPIN[i])
      return false;
  return true;
}

VerifyPIN {
  authentification = false;
  if (tries > 0) {
    if (Compare()) {
      tries = 3;
      authentification = true;
    } else {
      tries--;
    }
  }
}
```

Security Property

- $\texttt{userPIN} \neq \texttt{cardPIN} \implies \neg\, \texttt{authenticated}\ (\vee\ \texttt{detected\_attack})$

Overview of the Results

Baseline CV32E40P + VerifyPIN with various countermeasures

- Many FI vulnerabilities have been found ($\sim$ 59)
- Some of them already exist in the literature (exploiting the forwarding mechanism)
- Others highlight new effects (e.g., the Prefetch Buffer)

Baseline CV32E40P + VerifyPIN with the most countermeasures

- No fault injection permits bypassing the secure authentication was detected

Hardened CV32E40P + unprotected VerifyPIN

- No fault injection permits bypassing the secure authentication
- The hardware countermeasure is effective

Perspective and Conclusion

Performances

Use Case	HW Size	SW Length	# FI locations	Fault Effects	userPIN & cardPIN (32 bits)	Overall Run Time
`Baseline CV32E40P`	2.8 kGates	70 instr	15240	Symbolic	Symbolic	**12.9 h**
`Hardened CV32E40P`	4.6 kGates	120 instr	22640	Symbolic	Symbolic	**25.0 h**

Scaling up on more complex designs

- *CV32E40P (RISCY)* $\sim$ 3 KGates – 4-stage pipeline
- *CVA6 (ARIANE)* $\sim$ 10 KGates – 6-stage pipeline

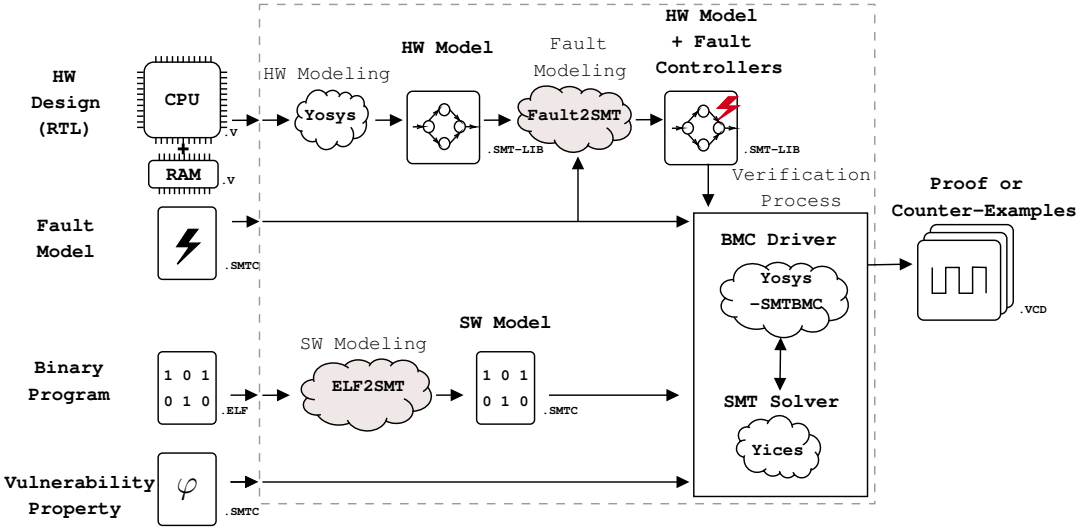
Optimizations

- *Modularity:* Compose with fault effects is not easy
- *Abstraction:* Attacks in unused modules (e.g., Multiplication) may result in vulnerabilities.

Conclusion

- Need to consider the HW and the SW together
- Propose a workflow: model + verification

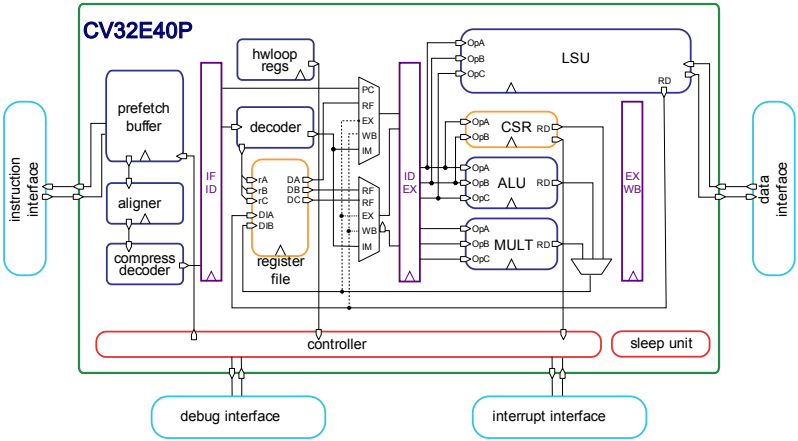
Questions ?


HW Design (RTL)
CPU
.v
RAM
.v
HW Modeling
Yosys
HW Model
.SMT-LIB
Fault Modeling
Fault2SMT
HW Model + Fault Controllers
.SMT-LIB
Verification Process
Proof or Counter-Examples
.VCD
Fault Model
.SMTC
BMC Driver
Yosys -SMTBMC
SMT Solver
Yices
Binary Program
1 0 1
0 1 0
.ELF
SW Modeling
ELF2SMT
SW Model
1 0 1
0 1 0
.SMTC
Vulnerability Property
φ
.SMTC

Fault Effects Exploration Results

The forwarding mechanism (known attack [Laurent, 2019])

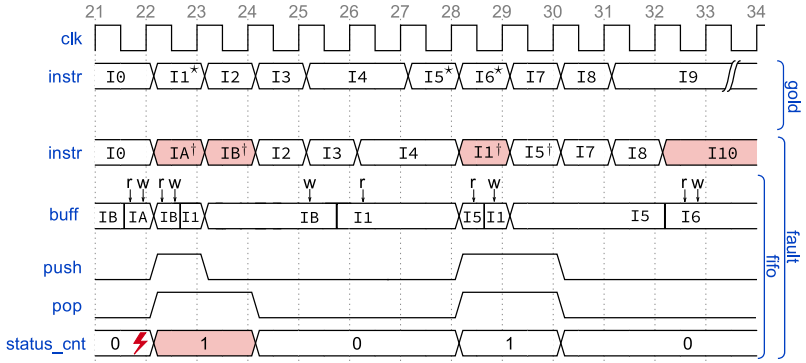
- Retrieve sensitive last-read data from the memory
- Invert conditional branches



Fault Effects Exploration Results

Fault in the Prefetch Buffer

- **Immediate one-time effect**, e.g., replay the Prefetch Buffer instructions
- **Immediate recurring effect**, e.g., incorrect order of the (replayed) instructions
- **Long-term effect**, e.g., corruption of the next branch target



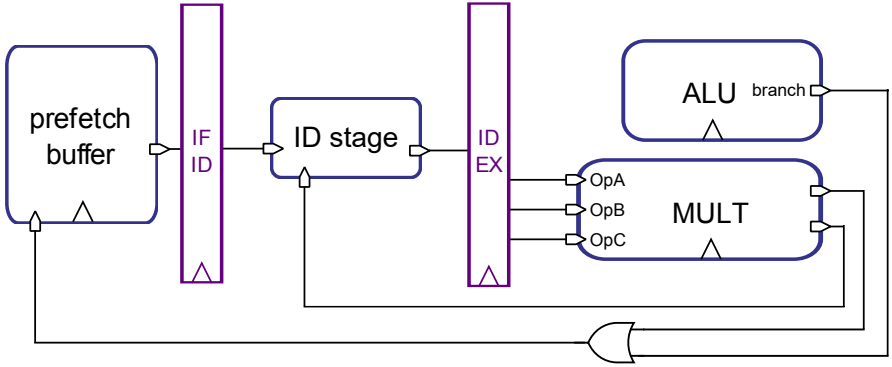
➔ *Fault effects depend on the microarchitectural details and the execution context*

Fault Effects Exploration Results

Fault in the Multiplier

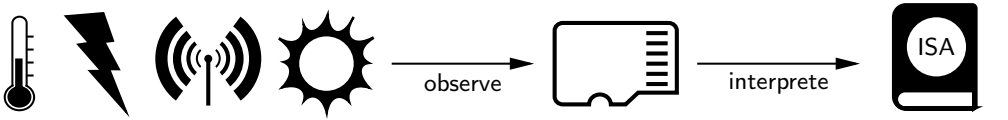
- When a multi-cycle multiplication is in progress, other stages are stalled
- When a branch address is calculated in the ALU, the IF stage cannot be stalled by the EX stage

➔ Activating the ALU and MULT at the same time will result in instructions being ignored



Experimental Verification Techniques

Experimental characterization process



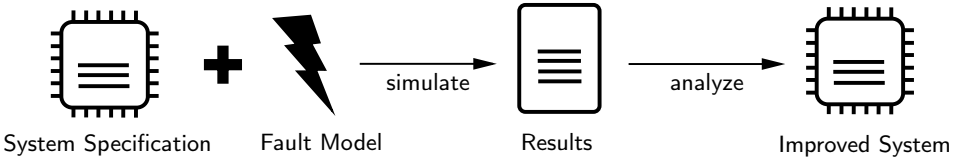
FI attacks on the targeted platform — Memory, Register File — ISA fault model

Experimental works and their observations

- EMFI on ARMv7-M architecture [Riviere, 2015] → Instruction skip and Instruction replay
- EMFI on an 8-bit ATmega328P microcontroller [Menu, 2020] → Multiple Instruction skip
- EMFI on ARM Cortex-M3 [Moro, 2013] → Instruction replacement

Simulation Verification Techniques

Simulation Process



Simulation-based related works

- ARMORY: ARM-M binaries emulator for FI [Hoffmann, (ARMORY) 2021]
  - Fault Model: Instruction skip ; Memory corruption ; Instruction replacement
- SimpliFI: gate-level simulation under FI (processor + software) [Grycel (SimpliFI), 2021]
  - Fault Model: clock glitch (delayed clock signal)

References I

[Tollec, FDTC 2022] S. Tollec et al. (2022)
Exploration of Fault Effects on Formal RISC-V Microarchitecture Models
Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)

[Riviere, 2015] L. Riviere et al. (2015)
High precision fault injections on the instruction cache of ARMv7-M architectures
IEEE International Symposium on Hardware Oriented Security and Trust (HOST)

[Moro, 2013] N. Moro et al. (2013)
Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller
Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)

[Menu, 2020] A. Menu et al. (2020)
Experimental analysis of the electromagnetic instruction skip fault model
15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS)

[Trouchkine, 2020] T. Trouchkine et al. (2020)
Fault Injection Characterization on Modern CPUs
Information Security Theory and Practice

[Laurent, 2018] J. Laurent et al. (2018)
On the importance of analysing microarchitecture for accurate software fault models
21st Euromicro Conference on Digital System Design (DSD)

References II

[Laurent, 2019] J. Laurent et al. (2019)
Fault Injection on Hidden Registers in a RISC-V Rocket Processor and Software Countermeasures
Design, Automation & Test in Europe Conference & Exhibition (DATE)

[Proy, 2019] J. Proy et al. (2019)
A first ISA-level characterization of EM pulse effects on superscalar microarchitectures: a secure software perspective
Proceedings of the 14th International Conference on Availability, Reliability and Security

[CV32E40P] OpenHW group
CORE-V CV32E40P User Manual `https://cv32e40p.readthedocs.io/en/latest/intro/`

[Chamelot, 2022] T. Chamelot et al. (2022)
SCI-FI: control signal, code, and control flow integrity against fault injection attacks
Design, Automation & Test in Europe Conference & Exhibition (DATE)

[Dureuil (FISSC), 2016] L. Dureuil et al. (2016)
FISSC: A fault injection and simulation secure collection
International Conference on Computer Safety, Reliability, and Security

[Hoffmann, (ARMORY) 2021] M. Hoffmann et al. (2021)
ARMORY: Fully Automated and Exhaustive Fault Simulation on ARM-M Binaries
IEEE Transactions on Information Forensics and Security

References III

[Nasahl (SYNFI), 2022] P. Nasahl et al. (2022)
SYNFI: Pre-Silicon Fault Analysis of an Open-Source Secure Element
IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES)

[Grycel (SimpliFI), 2021] J. Grycel et al. (2021)
SimpliFI: Hardware Simulation of Embedded Software Fault Attacks
Cryptography