Hardware Trojan Horses and Microarchitectural Side-Channel Attacks: Detection and Mitigation via Hardware-based Methodologies

Alessandro Palumbo

Associate Professor at CentraleSupélec, Paris-Saclay University, Inria SUSHI Team, Rennes Campus

alessandro.palumbo@inria.fr *https://palessumbo.github.io/*

HARDWARE SECURITY

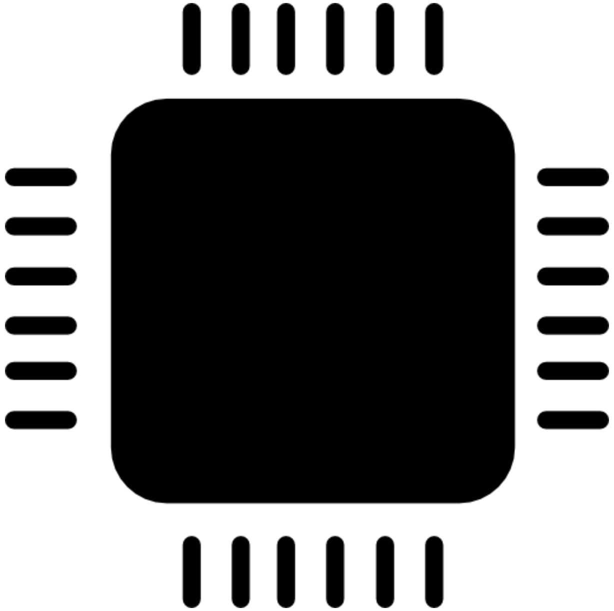
“Cybersecurity experts have traditionally assumed that the hardware underlying information systems is secure and trusted. It has been demonstrated that such assumption is no longer true.”

Prof. Mark M. Tehranipoor, PhD, Fellow of IEEE, ACM, NAI

Hardware Security

The Idea

- **Exploring methodologies to analyze and detect potential malicious activity in microprocessors**



Hardware Vulnerabilities

Introduction

- **Hardware Trojan Horses**
- **Reverse Engineering**
- **IP Piracy**
  - IP cloning
- **Side-Channel Attacks**
  - Microarchitectural SCAs
  - Physical attacks

- **Counterfeiting**
  - Overproduction
  - IC cloning
- **Backdoors**
  - Circuit modifications leaking secrets
- **Tampering**

Hardware Vulnerabilities

Introduction

- **Hardware Trojan Horses**
- **Reverse Engineering**
- **IP Piracy**
  - IP cloning
- **Side-Channel Attacks**
  - **Microarchitectural SCAs**
  - Physical attacks

- **Counterfeiting**
  - Overproduction
  - IC cloning
- **Backdoors**
  - Circuit modifications leaking secrets
- **Tampering**
  - **FPGA bitstream modifications**

Hardware Trojan Horses

Background

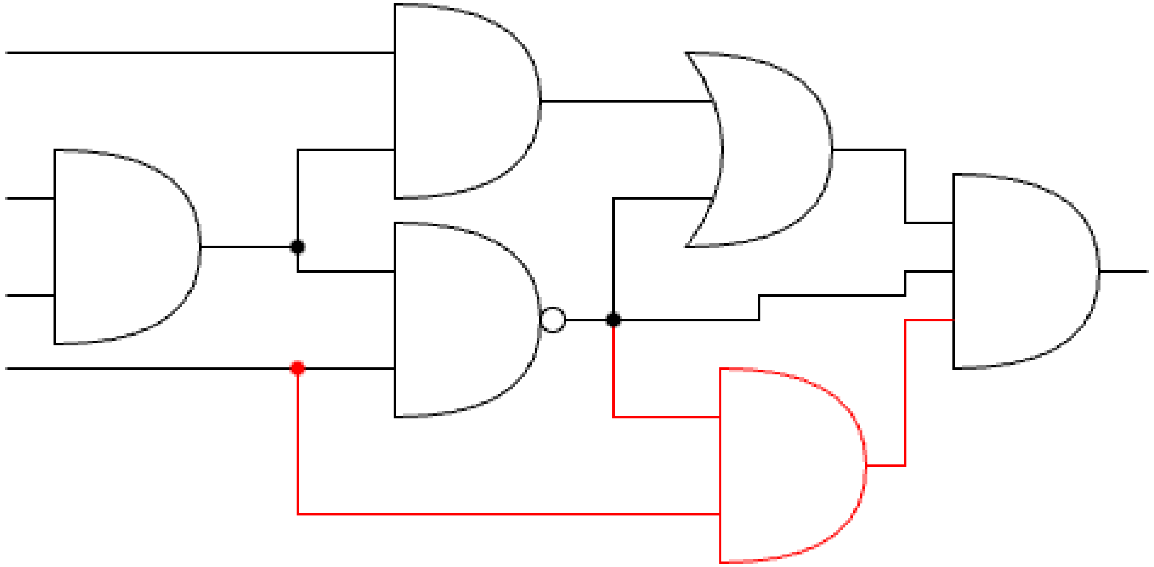
- **What is an Hardware Trojan Horse?**
  - A malicious addition or modification to the existing circuit elements

- **What an Hardware Trojan Horse can do?**
  - Change the functionality
  - Reduce the reliability
  - Leak valuable information

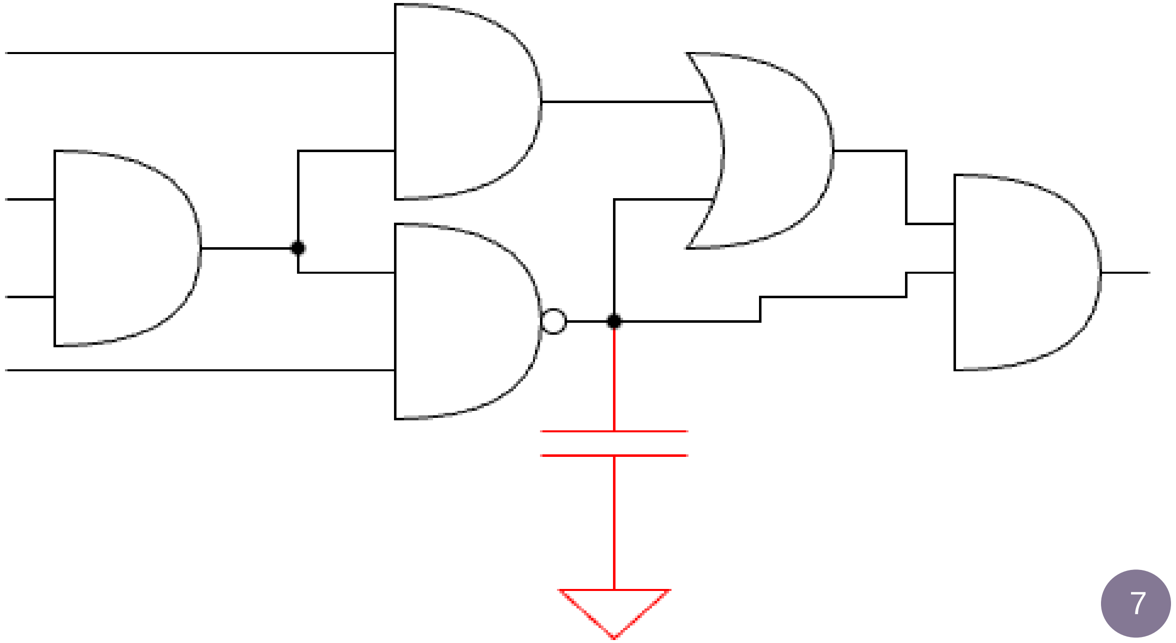
Hardware Trojan Horses

Background

- **Modify a Function**

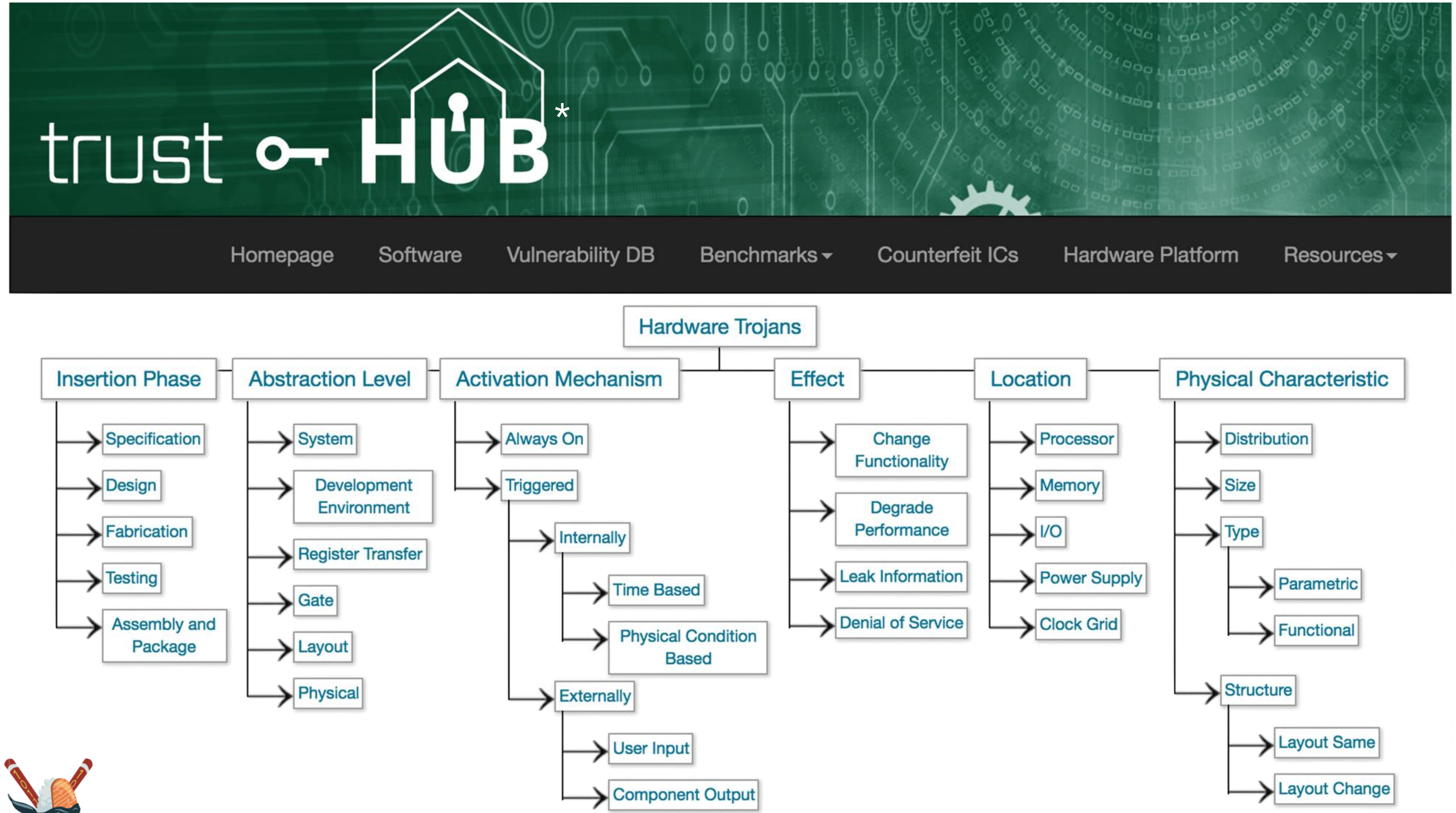


- **Modify the Specification**
  - Noise
  - Delay



Hardware Trojan Horses

Introduction – Taxonomy



Hardware Trojans

- Insertion Phase
  - Specification
  - Design
  - Fabrication
  - Testing
  - Assembly and Package
- Abstraction Level
  - System
  - Development Environment
  - Register Transfer
  - Gate
  - Layout
  - Physical
- Activation Mechanism
  - Always On
  - Triggered
    - Internally
      - Time Based
      - Physical Condition Based
    - Externally
      - User Input
      - Component Output
- Effect
  - Change Functionality
  - Degrade Performance
  - Leak Information
  - Denial of Service
- Location
  - Processor
  - Memory
  - I/O
  - Power Supply
  - Clock Grid
- Physical Characteristic
  - Distribution
  - Size
  - Type
    - Parametric
    - Functional
  - Structure
    - Layout Same
    - Layout Change

*https://trust-hub.org/#/home

Hardware Trojan Horses: Just Research?

Introduction – The motivation

- **The *Rosenbridge backdoor** has been found in a commercial Via Technologies C3 processor**
  - A specific sequence of instructions allowed the attacker to activate the *Rosenbridge* backdoor and enter the supervisor mode
- **Via Technologies officially commented that this behavior was due to an undocumented feature meant for debugging**

**C. Domas, “Hardware backdoors in x86 cpus.” https://i.blackhat.com/us-18/Thu-August-9/us-18-Domas-God-Mode-Unlocked-Hardware-Backdoors-In-x86-CPUs-wp.pdf,*

Hardware Trojan Horses

Background

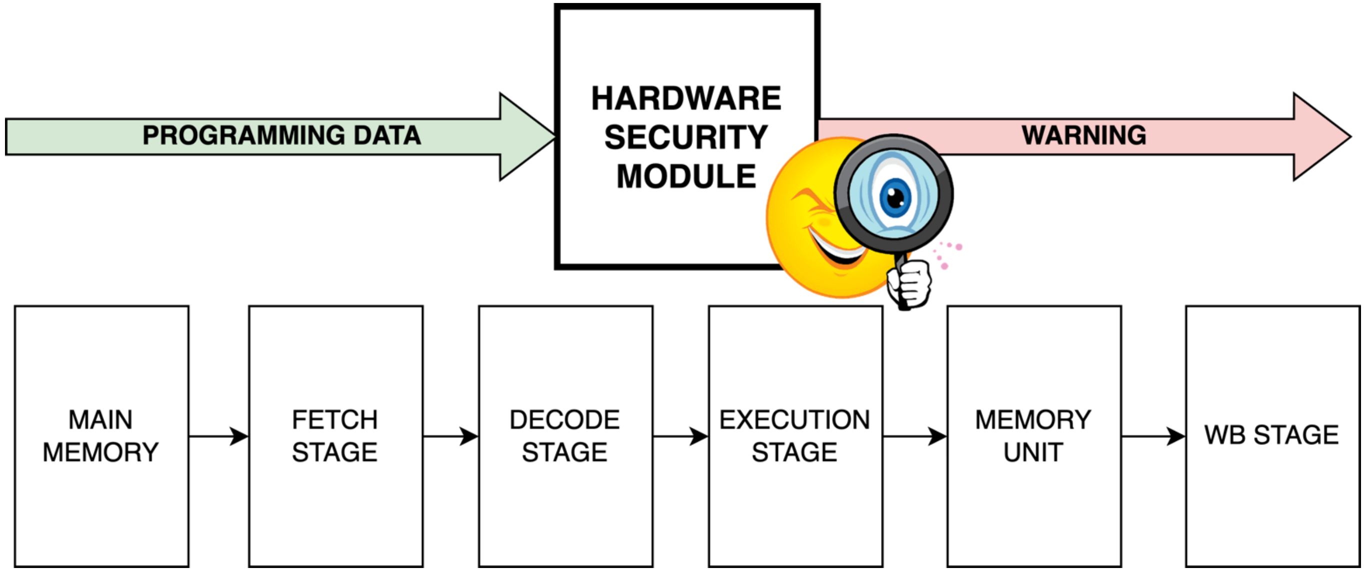
- **What is an Hardware Trojan Horse?**
  - A malicious addition or modification to the existing circuit elements

- **What an Hardware Trojan Horse can do?**
  - **Change the functionality**
    - **Interfering with Fetch instruction activity**
  - Reduce the reliability
  - **Leak valuable information**

Architectural Countermeasure 1/2 Approach

Detecting Hardware Trojans Interfering with Fetching Instruction Activity

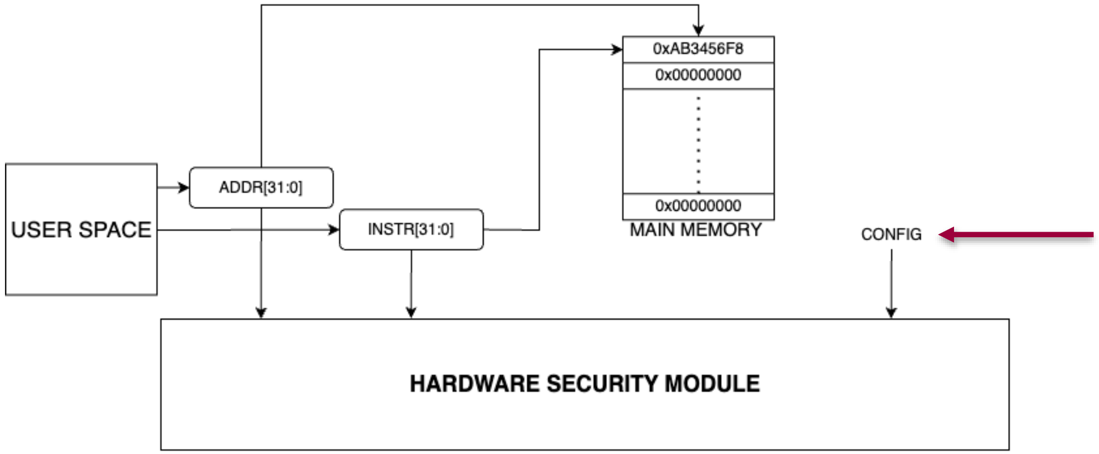
- **Add an online Hardware Security Module to analyze and detect potential malicious fetching instruction activity interferences**
  - The programmable is useful to specify what is « *legit* »



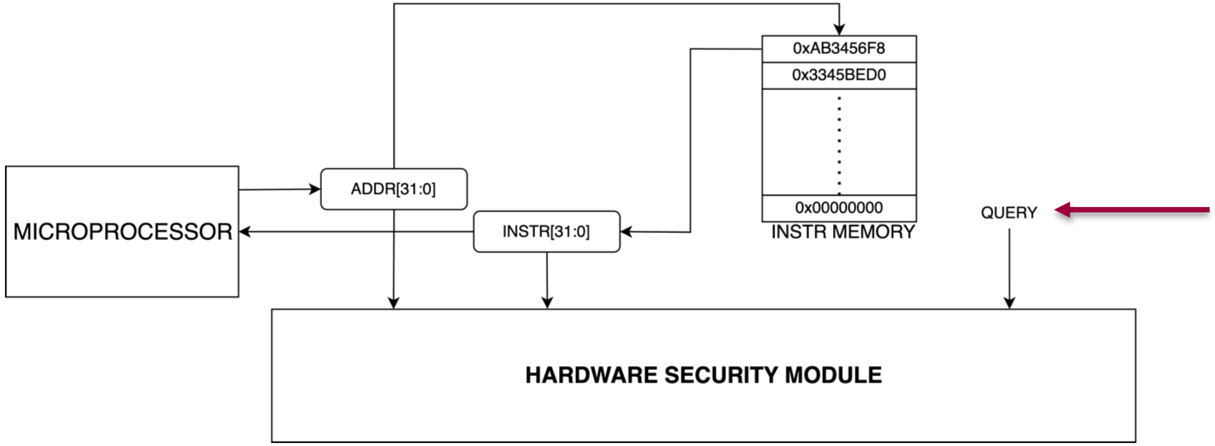
Architectural Countermeasure 1/2 Idea

Detecting Hardware Trojans Interfering with Fetching Instruction Activity

- **Configuration phase**
  - The HSM stores the information about legit address-instruction pairs



- **Query Phase**
  - The HSM checks at runtime if the fetched instructions are legit

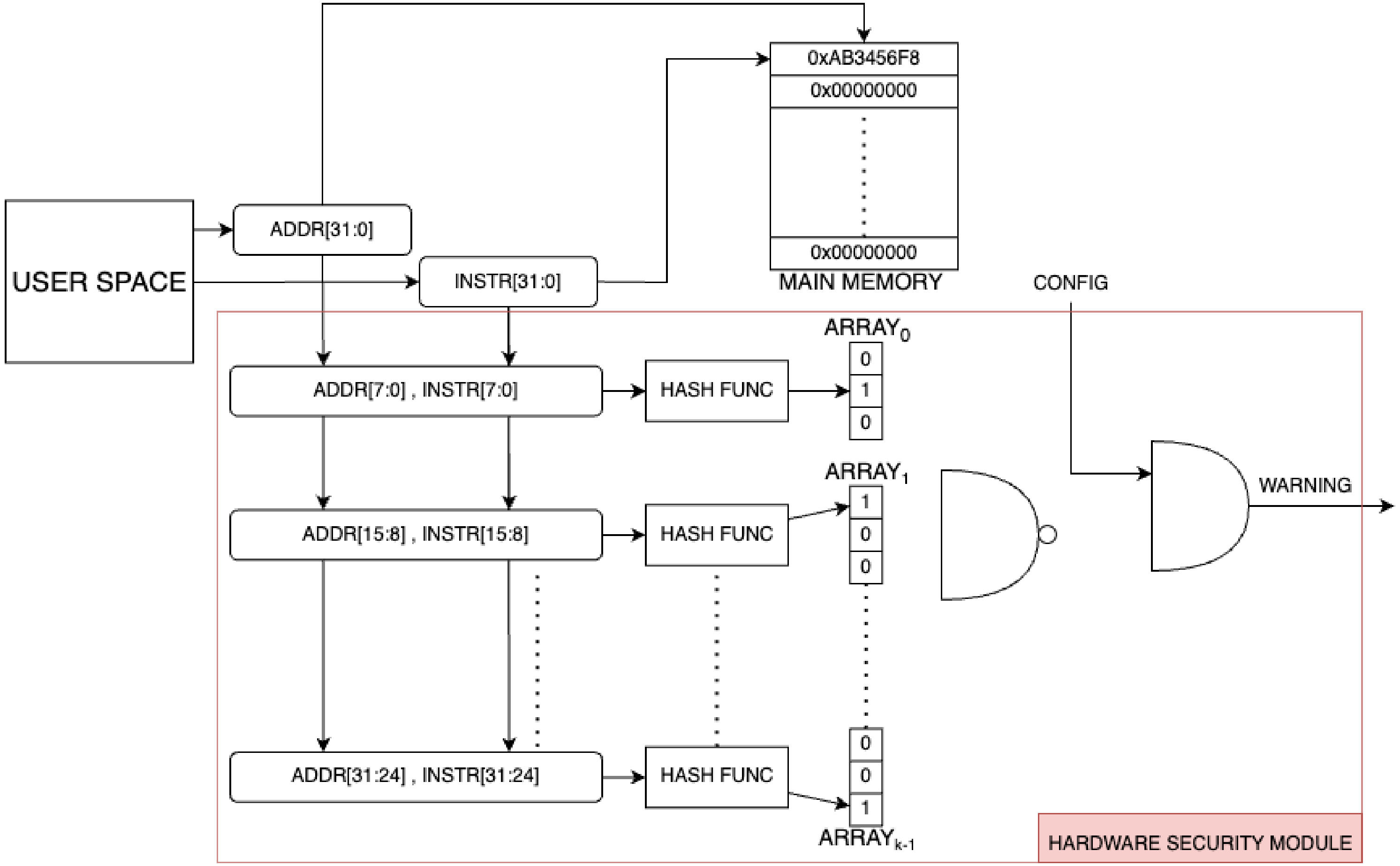


[1] *A. Palumbo, et al. "A lightweight security checking module to protect microprocessors against hardware trojan horses," in 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1– 6, 2021*

Architectural Countermeasure 1/2 Idea

Detecting Hardware Trojans Interfering with Fetching Instruction Activity

- **Configuration phase**
  - The HSM stores the information about legit address-instruction pairs

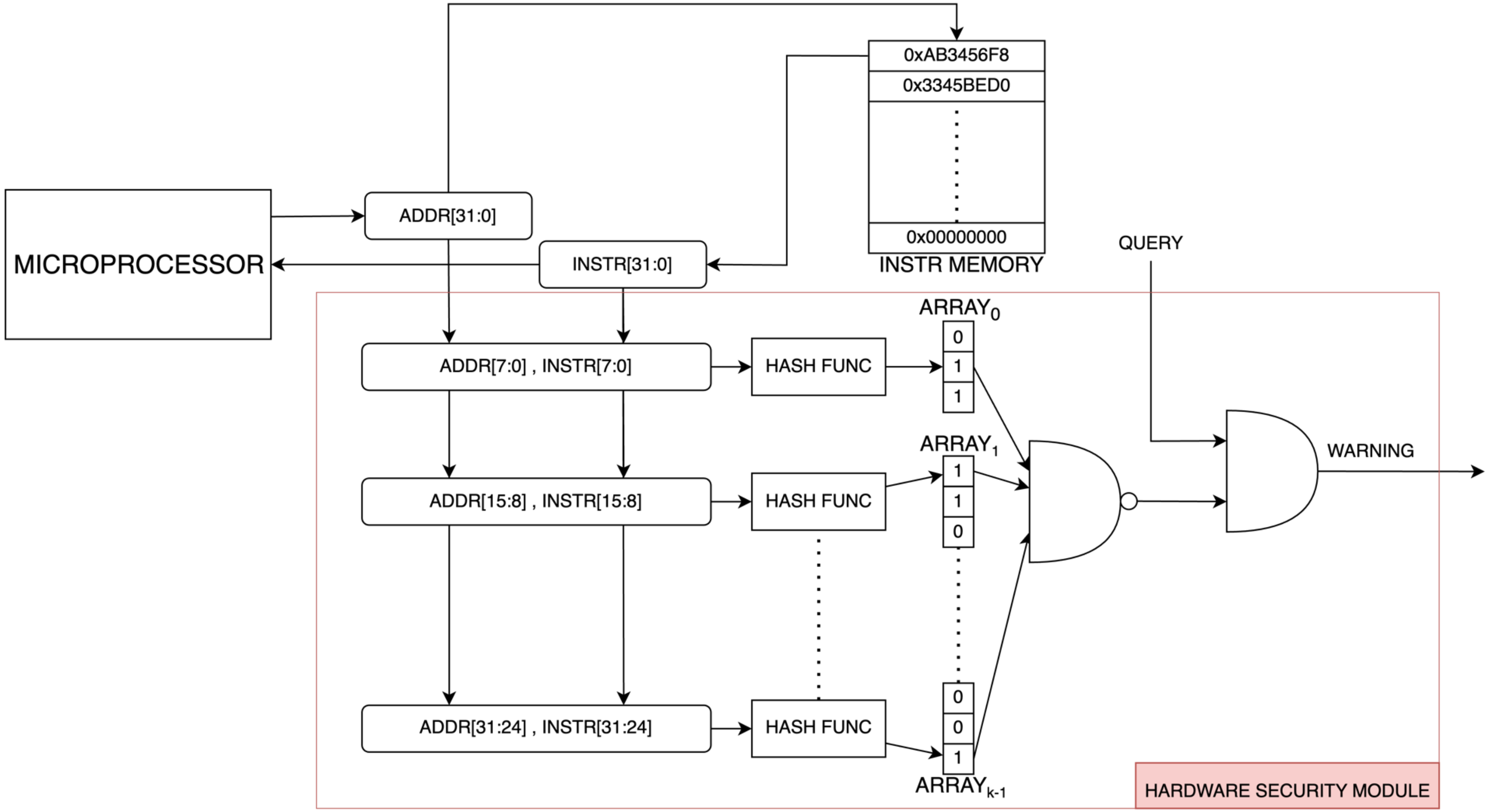


[1] A. Palumbo, et al. “A lightweight security checking module to protect microprocessors against hardware trojan horses,” in 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1– 6, 2021

Architectural Countermeasure 1/2 Idea

Detecting Hardware Trojans Interfering with Fetching Instruction Activity

- **Query Phase**
  - The HSM checks at runtime if the fetched instructions are legit

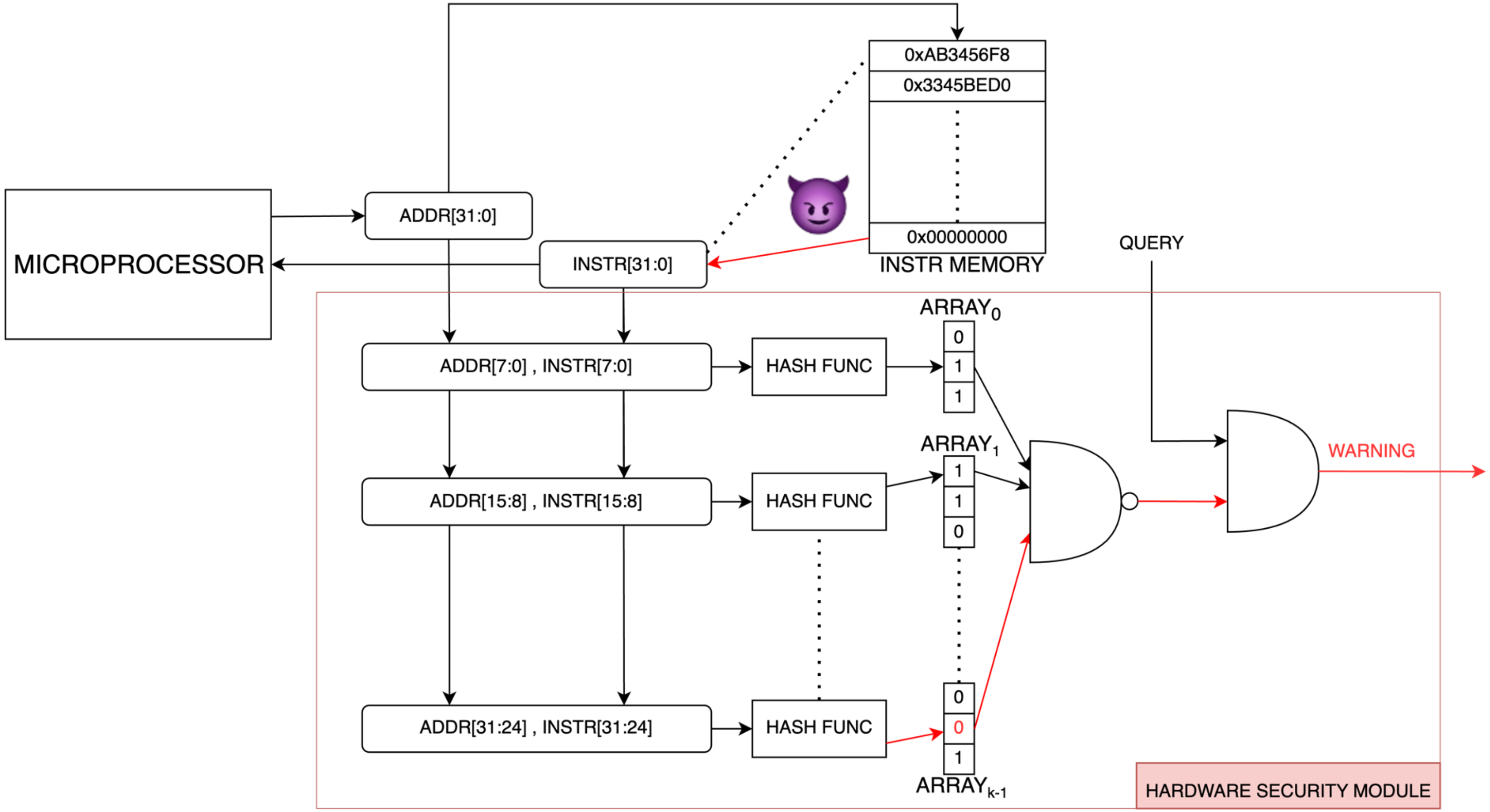


[1] A. Palumbo, et al. "A lightweight security checking module to protect microprocessors against hardware trojan horses," in 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1– 6, 2021

Architectural Countermeasure 1/2 Idea

Detecting Hardware Trojans Interfering with Fetching Instruction Activity

- **Query Phase**
  - The HSM checks at runtime if the fetched instructions are legit

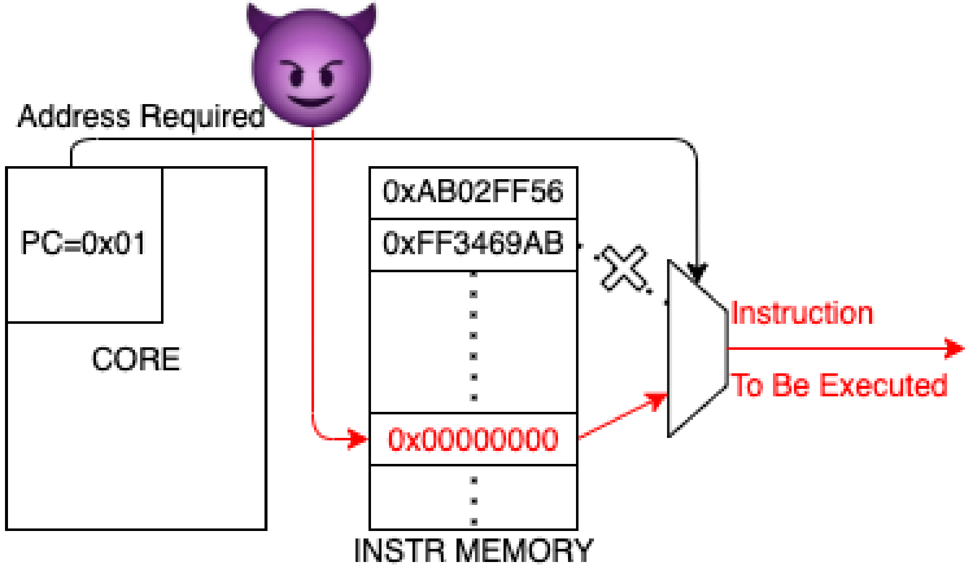


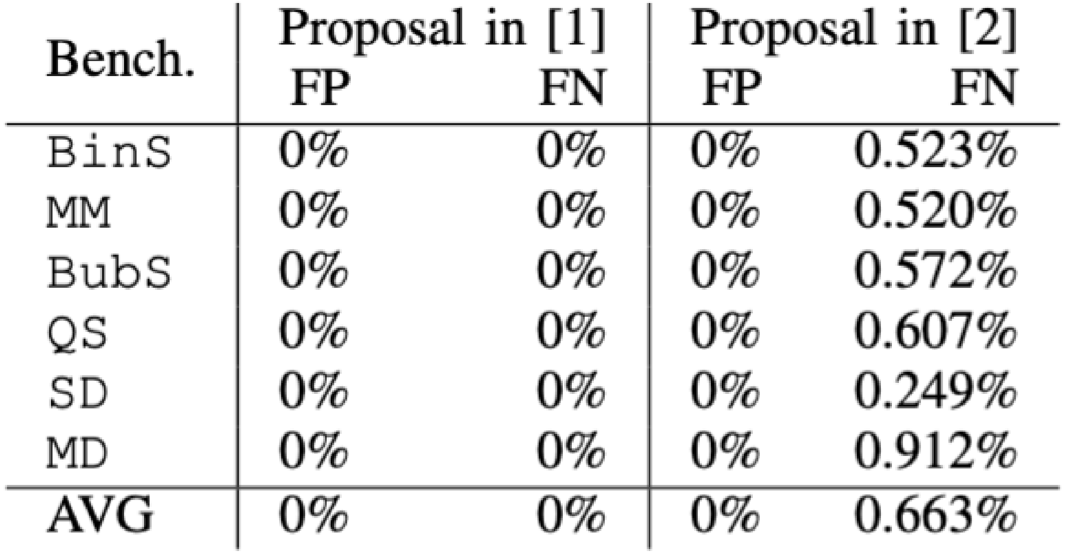
[1] *A. Palumbo, et al. “A lightweight security checking module to protect microprocessors against hardware trojan horses,” in 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1– 6, 2021*

Architectural Countermeasure 1/2 Idea

Detecting Hardware Trojans Interfering with Fetching Instruction Activity

- **Threat Model 1**
  - Injecting the fetch of a malicious instruction not part of the installed program





Bench.	Proposal in [1]		Proposal in [2]	
	FP	FN	FP	FN
BinS	0%	0%	0%	0.523%
MM	0%	0%	0%	0.520%
BubS	0%	0%	0%	0.572%
QS	0%	0%	0%	0.607%
SD	0%	0%	0%	0.249%
MD	0%	0%	0%	0.912%
AVG	0%	0%	0%	0.663%

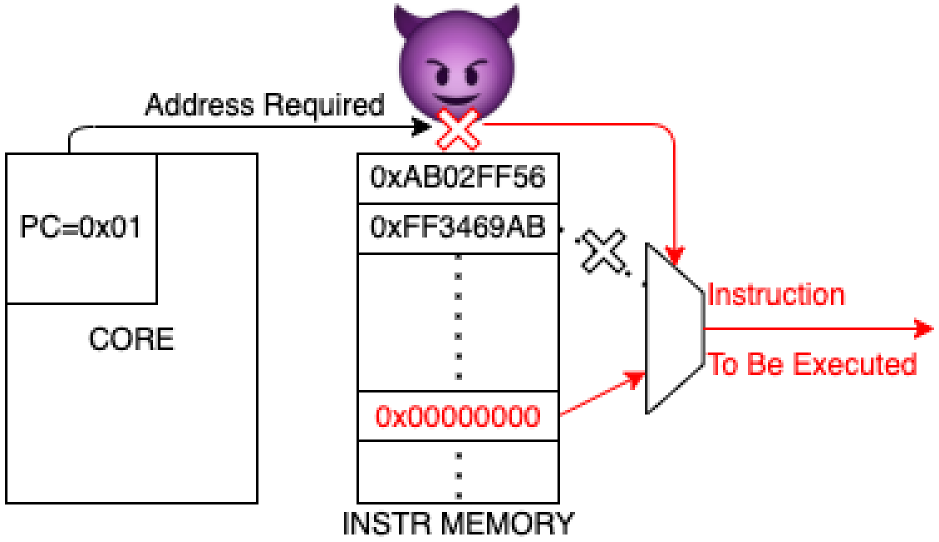
[1] A. Palumbo, et al. “A lightweight security checking module to protect microprocessors against hardware trojan horses,” in 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1– 6, 2021

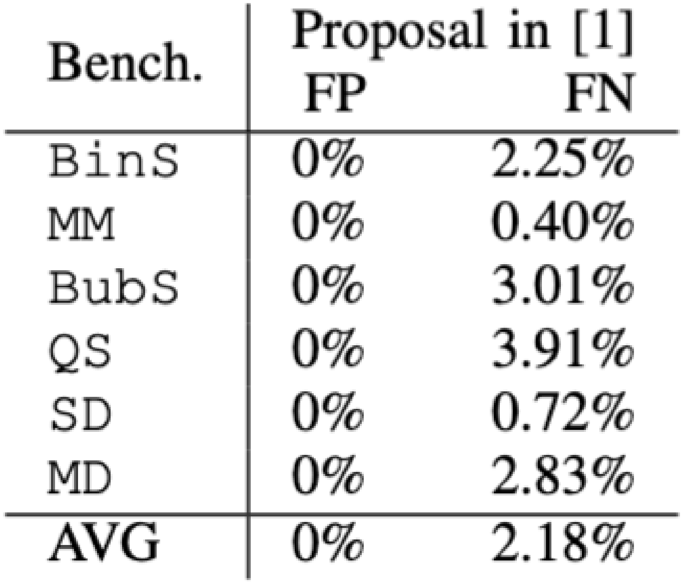
[2] A. Bolat, et al. “A microprocessor protection architecture against hardware trojans in memories,” in 2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS), pp. 1–6, 2020.

Architectural Countermeasure 1/2 Idea

Detecting Hardware Trojans Interfering with Fetching Instruction Activity

- **Threat Model 2**
  - Injecting the fetch of an instruction part of the installed program, but in a « wrong moment »





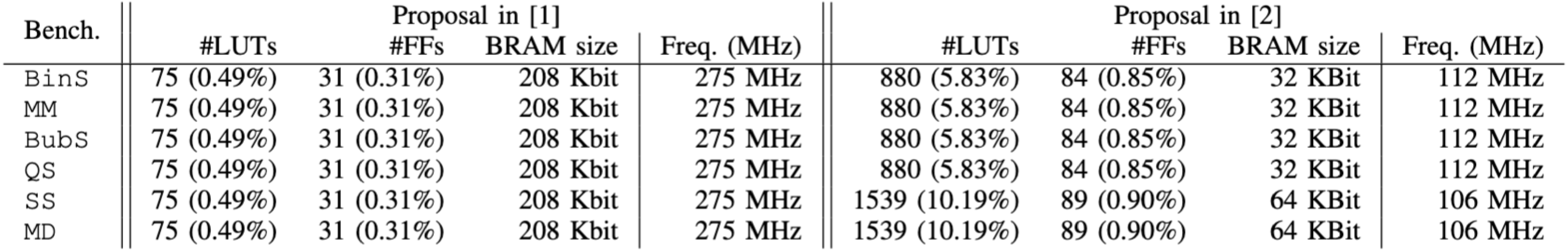
Bench.	Proposal in [1]	
	FP	FN
BinS	0%	2.25%
MM	0%	0.40%
BubS	0%	3.01%
QS	0%	3.91%
SD	0%	0.72%
MD	0%	2.83%
AVG	0%	2.18%

[1] A. Palumbo, et al. “A lightweight security checking module to protect microprocessors against hardware trojan horses,” in 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1– 6, 2021

Architectural Countermeasure 1/2 Idea

Detecting Hardware Trojans Interfering with Fetching Instruction Activity

- **FPGA Emulation**
  - Resources usage compared with RI5CY-V PULPINO core



Bench.	Proposal in [1]				Proposal in [2]			
	#LUTs	#FFs	BRAM size	Freq. (MHz)	#LUTs	#FFs	BRAM size	Freq. (MHz)
BinS	75 (0.49%)	31 (0.31%)	208 Kbit	275 MHz	880 (5.83%)	84 (0.85%)	32 KBit	112 MHz
MM	75 (0.49%)	31 (0.31%)	208 Kbit	275 MHz	880 (5.83%)	84 (0.85%)	32 KBit	112 MHz
BubS	75 (0.49%)	31 (0.31%)	208 Kbit	275 MHz	880 (5.83%)	84 (0.85%)	32 KBit	112 MHz
QS	75 (0.49%)	31 (0.31%)	208 Kbit	275 MHz	880 (5.83%)	84 (0.85%)	32 KBit	112 MHz
SS	75 (0.49%)	31 (0.31%)	208 Kbit	275 MHz	1539 (10.19%)	89 (0.90%)	64 KBit	106 MHz
MD	75 (0.49%)	31 (0.31%)	208 Kbit	275 MHz	1539 (10.19%)	89 (0.90%)	64 KBit	106 MHz

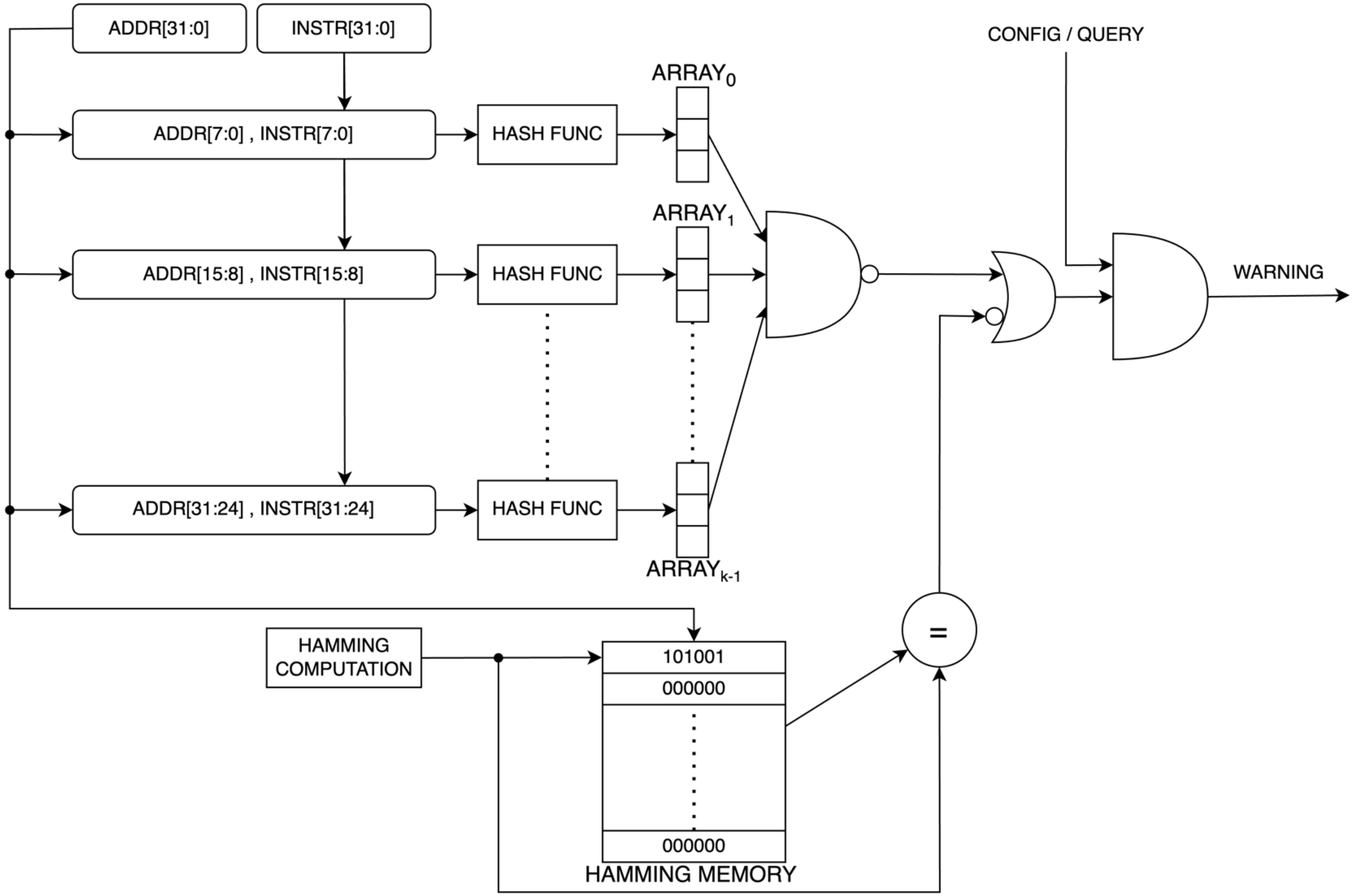
[1] A. Palumbo, et al. “A lightweight security checking module to protect microprocessors against hardware trojan horses,” in 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1– 6, 2021

[2] A. Bolat, et al. “A microprocessor protection architecture against hardware trojans in memories,” in 2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS), pp. 1–6, 2020.

Architectural Countermeasure 1/2 Idea Evolution

Improving the Detection Hardware Trojans Interfering with Fetching Instruction Activity

- **Two goals at the same time:**
  - Protecting from HTHs
  - Correcting Bit Flips

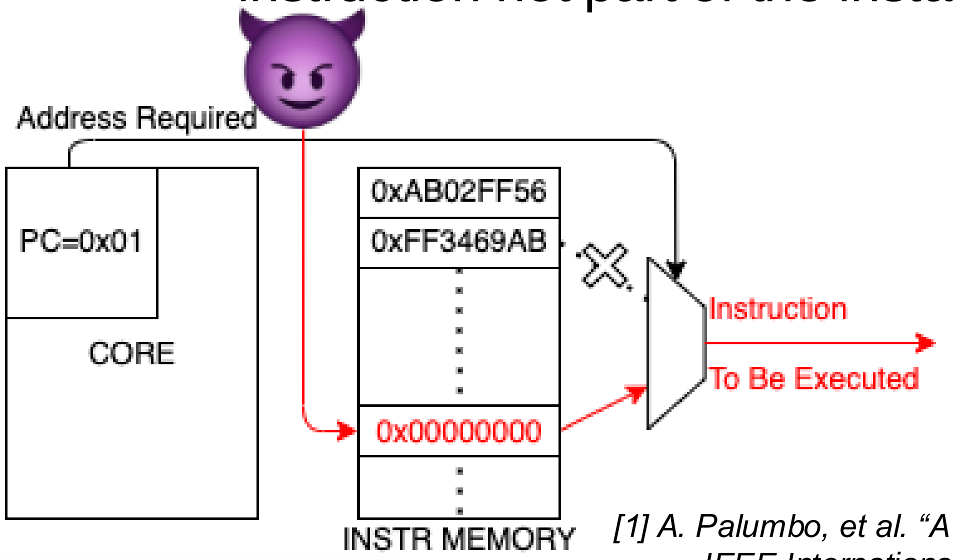


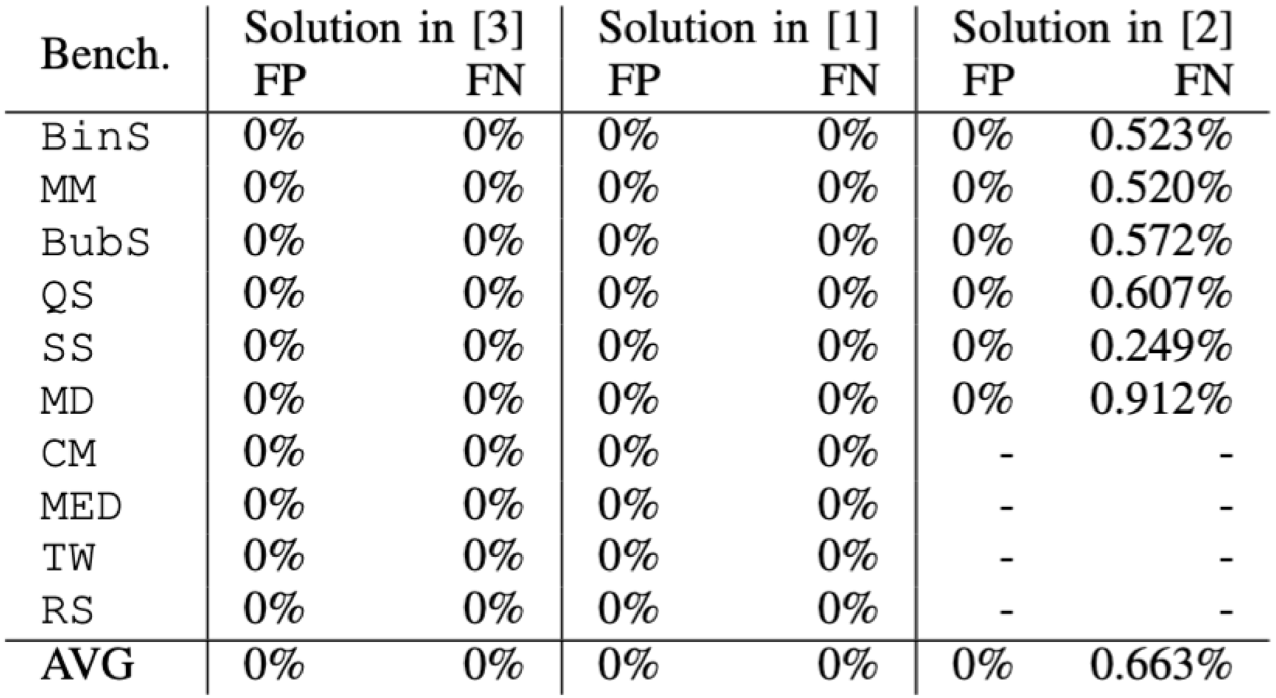
[3] A. Palumbo, et al. "Improving the detection of hardware trojan horses in microprocessors via hamming codes," in 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1–6, 2023.

Architectural Countermeasure 1/2 Idea

Improving the Detection Hardware Trojans Interfering with Fetching Instruction Activity

- **Threat Model 1**
  - Injecting the fetching of a malicious instruction not part of the installed program





Bench.	Solution in [3]		Solution in [1]		Solution in [2]	
	FP	FN	FP	FN	FP	FN
BinS	0%	0%	0%	0%	0%	0.523%
MM	0%	0%	0%	0%	0%	0.520%
BubS	0%	0%	0%	0%	0%	0.572%
QS	0%	0%	0%	0%	0%	0.607%
SS	0%	0%	0%	0%	0%	0.249%
MD	0%	0%	0%	0%	0%	0.912%
CM	0%	0%	0%	0%	-	-
MED	0%	0%	0%	0%	-	-
TW	0%	0%	0%	0%	-	-
RS	0%	0%	0%	0%	-	-
AVG	0%	0%	0%	0%	0%	0.663%

[1] A. Palumbo, et al. "A lightweight security checking module to protect microprocessors against hardware trojan horses," in 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1– 6, 2021.

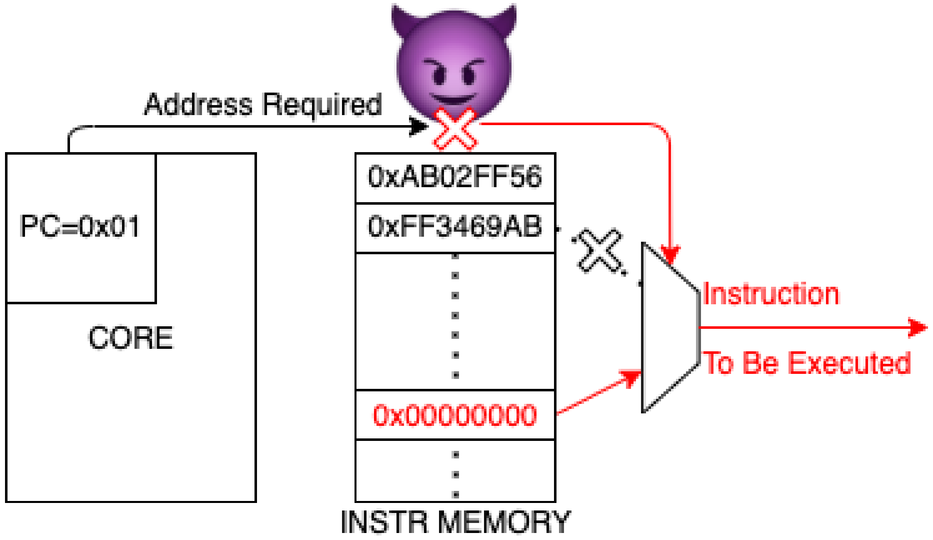
[2] A. Bolat, et al. "A microprocessor protection architecture against hardware trojans in memories," in 2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS), pp. 1–6, 2020.

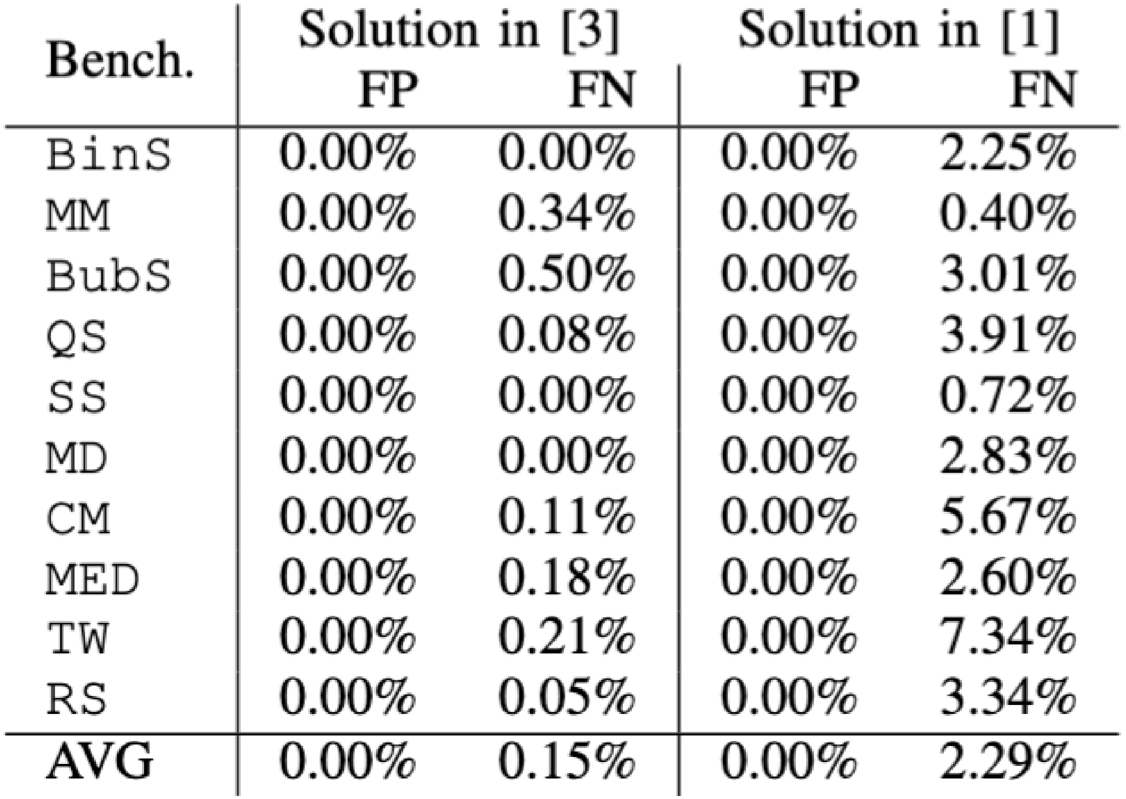
[3] A. Palumbo, et al. "Improving the detection of hardware trojan horses in microprocessors via hamming codes," in 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1–6, 2023.

Architectural Countermeasure 1/2 Idea

Improving the Detection Hardware Trojans Interfering with Fetching Instruction Activity

- **Threat Model 2**
  - Injecting the fetching of an instruction part of the installed program, but in a « wrong moment »





Bench.	Solution in [3]		Solution in [1]	
	FP	FN	FP	FN
BinS	0.00%	0.00%	0.00%	2.25%
MM	0.00%	0.34%	0.00%	0.40%
BubS	0.00%	0.50%	0.00%	3.01%
QS	0.00%	0.08%	0.00%	3.91%
SS	0.00%	0.00%	0.00%	0.72%
MD	0.00%	0.00%	0.00%	2.83%
CM	0.00%	0.11%	0.00%	5.67%
MED	0.00%	0.18%	0.00%	2.60%
TW	0.00%	0.21%	0.00%	7.34%
RS	0.00%	0.05%	0.00%	3.34%
AVG	0.00%	0.15%	0.00%	2.29%

[1] A. Palumbo, et al. "A lightweight security checking module to protect microprocessors against hardware trojan horses," in 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1– 6, 2021.

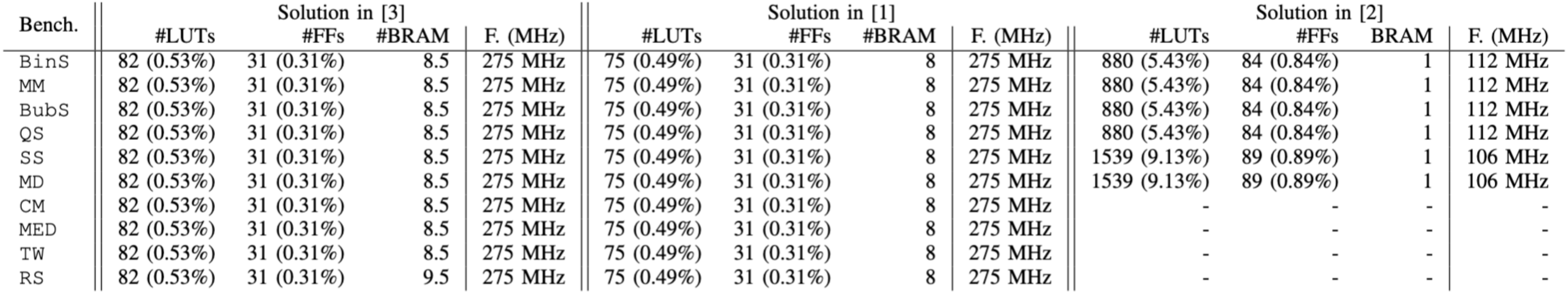
[3] A. Palumbo, et al. "Improving the detection of hardware trojan horses in microprocessors via hamming codes," in 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1–6, 2023.

Architectural Countermeasure 1/2 Idea

Improving the Detection Hardware Trojans Interfering with Fetching Instruction Activity

- **FPGA Emulation**

Resource usage compared with RI5CY-V PULPINO core



Bench.	Solution in [3]				Solution in [1]				Solution in [2]			
	#LUTs	#FFs	#BRAM	F. (MHz)	#LUTs	#FFs	#BRAM	F. (MHz)	#LUTs	#FFs	BRAM	F. (MHz)
BinS	82 (0.53%)	31 (0.31%)	8.5	275 MHz	75 (0.49%)	31 (0.31%)	8	275 MHz	880 (5.43%)	84 (0.84%)	1	112 MHz
MM	82 (0.53%)	31 (0.31%)	8.5	275 MHz	75 (0.49%)	31 (0.31%)	8	275 MHz	880 (5.43%)	84 (0.84%)	1	112 MHz
BubS	82 (0.53%)	31 (0.31%)	8.5	275 MHz	75 (0.49%)	31 (0.31%)	8	275 MHz	880 (5.43%)	84 (0.84%)	1	112 MHz
QS	82 (0.53%)	31 (0.31%)	8.5	275 MHz	75 (0.49%)	31 (0.31%)	8	275 MHz	880 (5.43%)	84 (0.84%)	1	112 MHz
SS	82 (0.53%)	31 (0.31%)	8.5	275 MHz	75 (0.49%)	31 (0.31%)	8	275 MHz	1539 (9.13%)	89 (0.89%)	1	106 MHz
MD	82 (0.53%)	31 (0.31%)	8.5	275 MHz	75 (0.49%)	31 (0.31%)	8	275 MHz	1539 (9.13%)	89 (0.89%)	1	106 MHz
CM	82 (0.53%)	31 (0.31%)	8.5	275 MHz	75 (0.49%)	31 (0.31%)	8	275 MHz	-	-	-	-
MED	82 (0.53%)	31 (0.31%)	8.5	275 MHz	75 (0.49%)	31 (0.31%)	8	275 MHz	-	-	-	-
TW	82 (0.53%)	31 (0.31%)	8.5	275 MHz	75 (0.49%)	31 (0.31%)	8	275 MHz	-	-	-	-
RS	82 (0.53%)	31 (0.31%)	9.5	275 MHz	75 (0.49%)	31 (0.31%)	8	275 MHz	-	-	-	-

[1] A. Palumbo, et al. “A lightweight security checking module to protect microprocessors against hardware trojan horses,” in 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1– 6, 2021.

[2] A. Bolat, et al. “A microprocessor protection architecture against hardware trojans in memories,” in 2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS), pp. 1–6, 2020.

[3] A. Palumbo, et al. “Improving the detection of hardware trojan horses in microprocessors via hamming codes,” in 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1–6, 2023.

Hardware Trojan Horses: Just Research?

Introduction – The motivation

- The *Rosenbridge backdoor** has been found in a commercial Via Technologies C3 processor
  - **A specific sequence of instructions allowed the attacker to activate the *Rosenbridge* backdoor and enter the supervisor mode**
- Via Technologies officially commented that this behavior was due to an undocumented feature meant for debugging

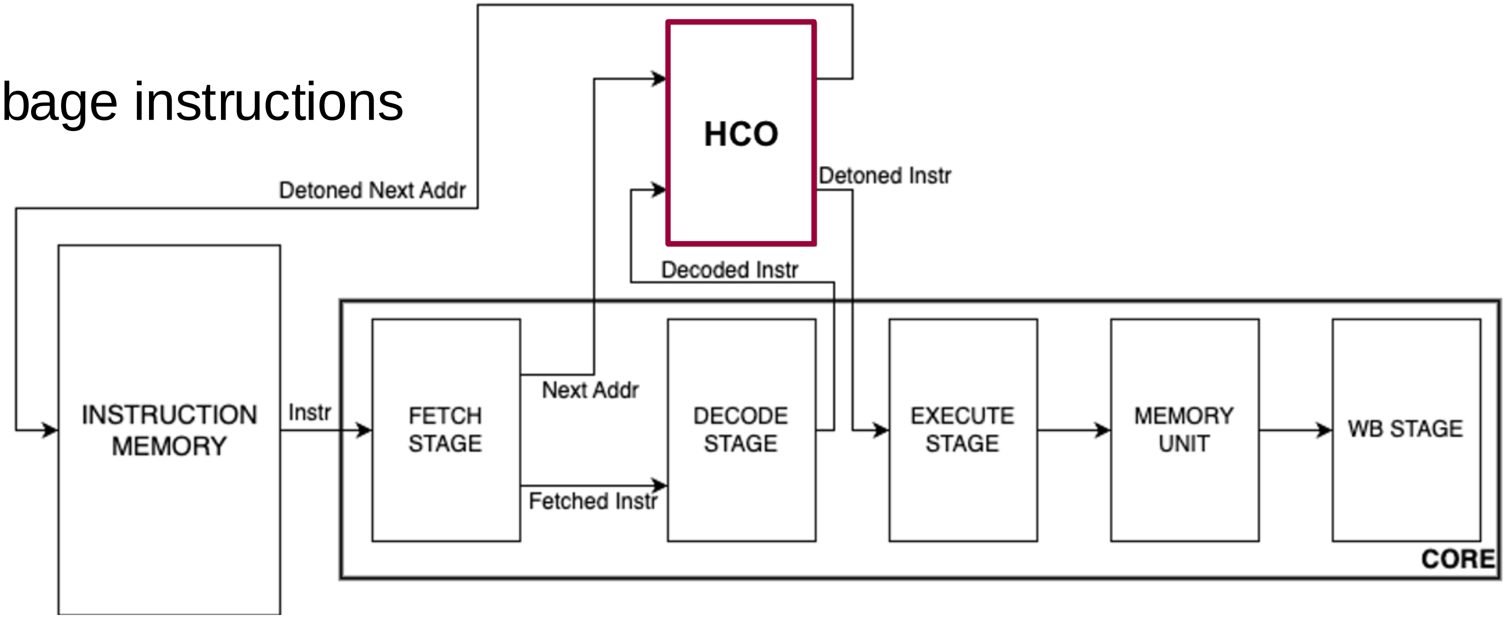
How can we avoid Software Exploitable Hardware Trojan Horse activations?

*C. Domas, "Hardware backdoors in x86 cpus." https://i.blackhat.com/us-18/Thu-August-9/us-18-Domas-God-Mode-Unlocked-Hardware-Backdoors-In-x86-CPUs-wp.pdf,

Architectural Countermeasure 2/2 Idea

Preventing the Activation of Software-Exploitable Hardware Trojan Horses

- **Add an online Hardware Code Obfuscator (HCO) in a microprocessor: injecting confusion**
  - Modify the instructions of the program → **an Hardware Compiler at runtime!**
    - Adding register scrambling instructions
    - Adding xoring instructions data after writes and the dexoring data instructions before reads
    - Adding garbage instructions



[4] A. Palumbo et al. "Built-in Software Obfuscation for Protecting Microprocessors against Hardware Trojan Horses." 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). IEEE, 2023

Architectural Countermeasure 2/2 Idea

Preventing the Activation of Software-Exploitable Hardware Trojan Horses

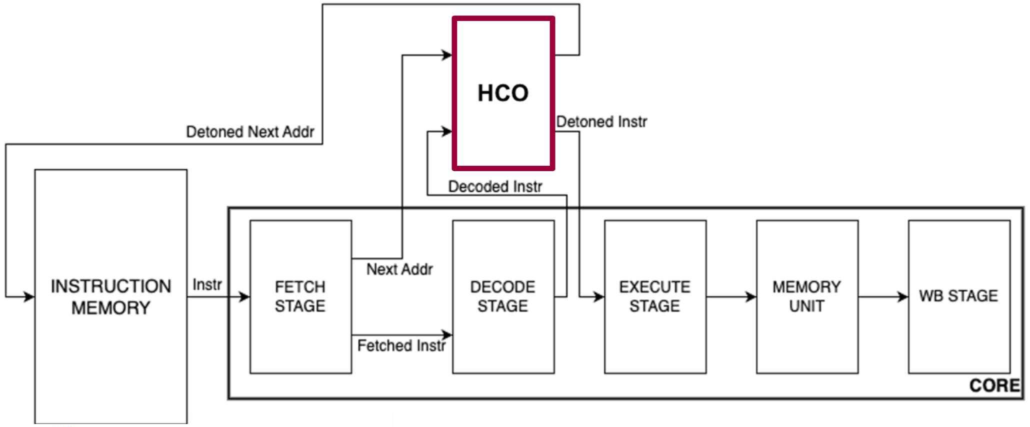
- **Add an online Hardware Code Obfuscator (HCO) in a microprocessor**

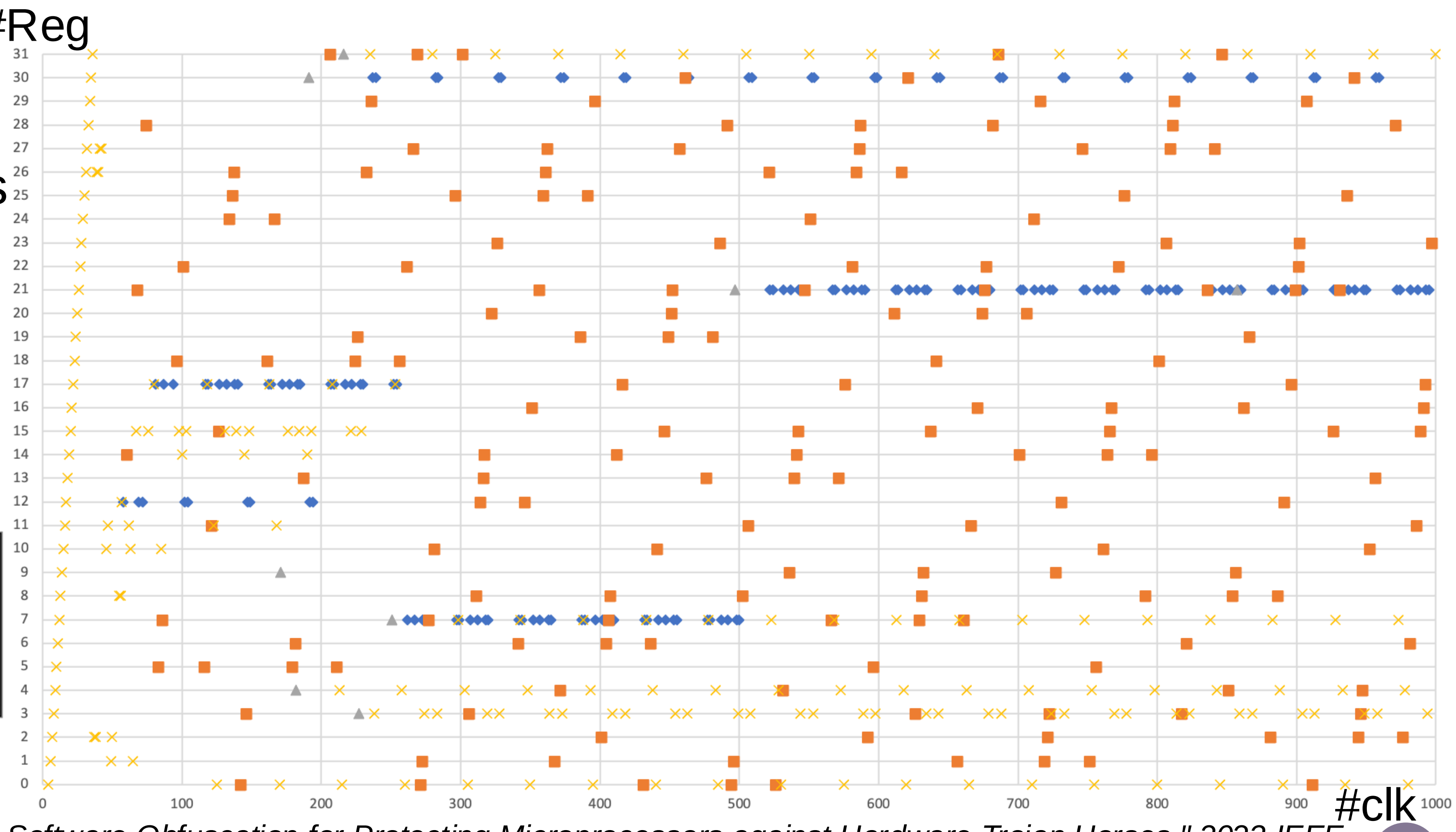
× No modified Instructions

▲ Register scrambling instructions

◆ Xoring/dexoring data instructions

■ Garbage instructions



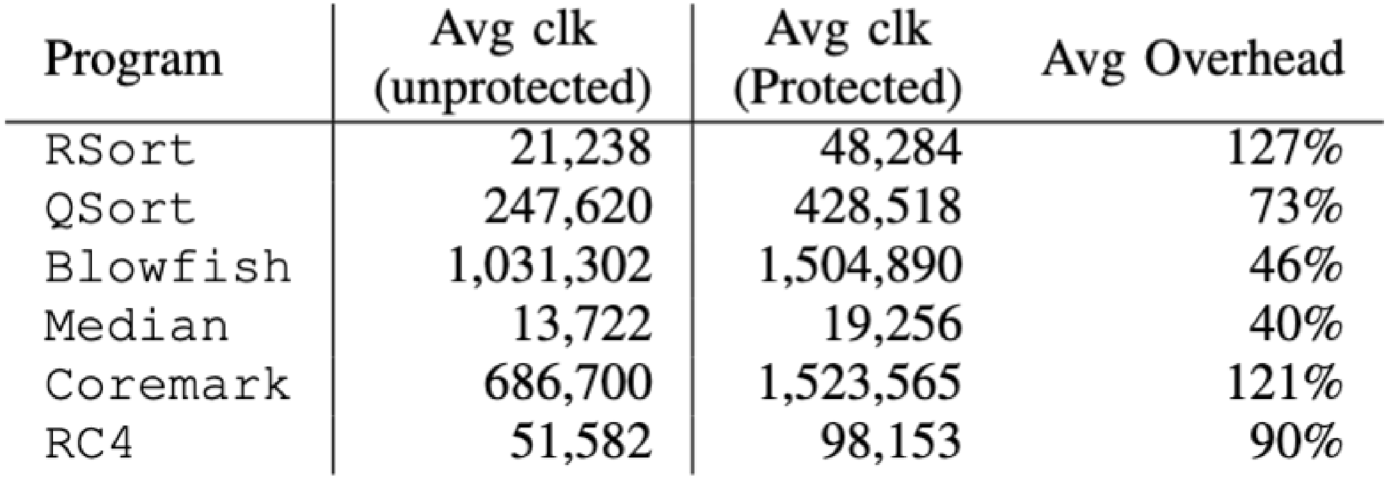


[4] A. Palumbo et al. "Built-in Software Obfuscation for Protecting Microprocessors against Hardware Trojan Horses." 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). IEEE, 2023

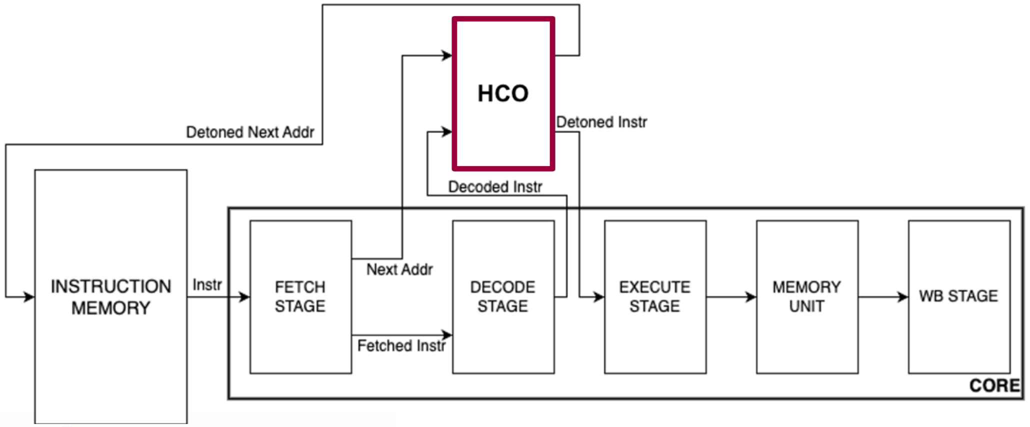
Architectural Countermeasure 2/2 Idea

Preventing the Activation of Software-Exploitable Hardware Trojan Horses

- **Add an online Hardware Code Obfuscator (HCO) in a microprocessor**
  - Register scrambling instructions
  - Xoring/dexoring data instructions
  - Garbage instructions



Program	Avg clk (unprotected)	Avg clk (Protected)	Avg Overhead
RSort	21,238	48,284	127%
QSort	247,620	428,518	73%
Blowfish	1,031,302	1,504,890	46%
Median	13,722	19,256	40%
Coremark	686,700	1,523,565	121%
RC4	51,582	98,153	90%

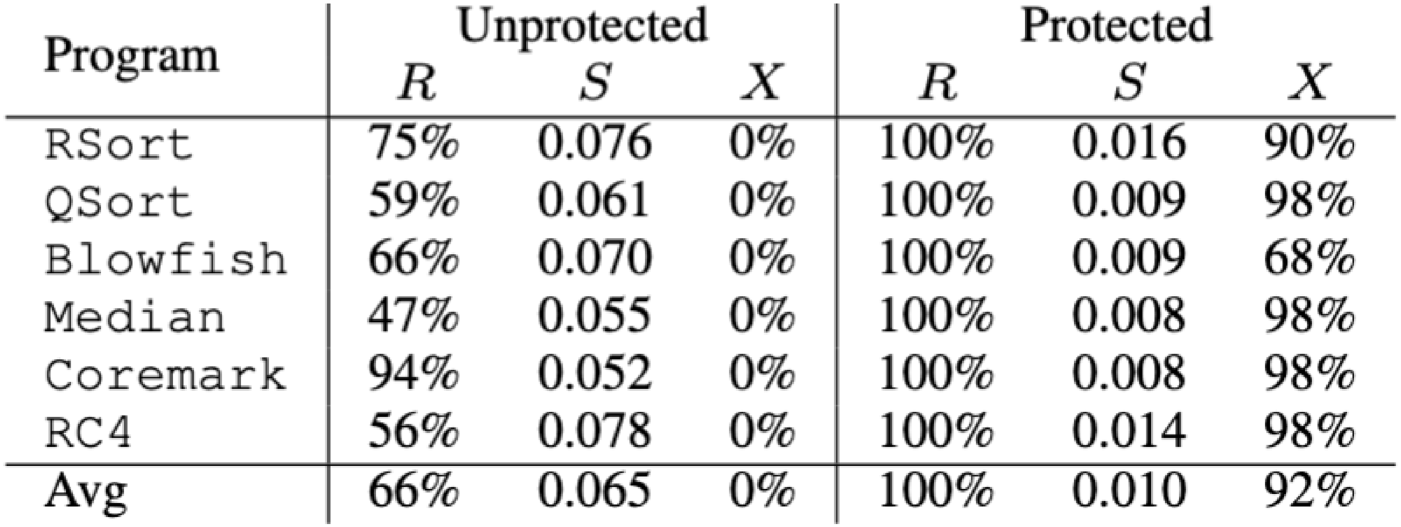


[4] A. Palumbo et al. "Built-in Software Obfuscation for Protecting Microprocessors against Hardware Trojan Horses." 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). IEEE, 2023

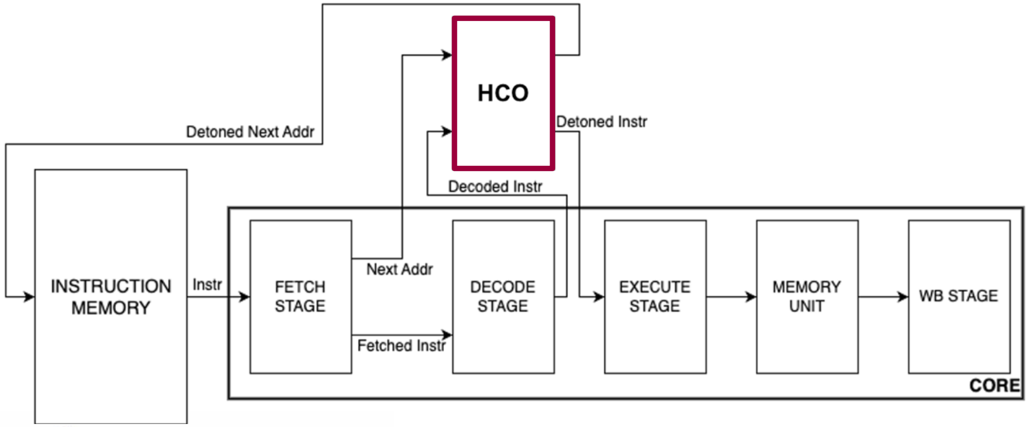
Architectural Countermeasure 2/2 Idea

Preventing the Activation of Software-Exploitable Hardware Trojan Horses

- **Add an online Hardware Code Obfuscator (HCO) in a microprocessor**
  - Register scrambling instructions
  - Xoring/dexoring data instructions
  - Garbage instructions



Program	Unprotected			Protected		
	R	S	X	R	S	X
RSort	75%	0.076	0%	100%	0.016	90%
QSort	59%	0.061	0%	100%	0.009	98%
Blowfish	66%	0.070	0%	100%	0.009	68%
Median	47%	0.055	0%	100%	0.008	98%
Coremark	94%	0.052	0%	100%	0.008	98%
RC4	56%	0.078	0%	100%	0.014	98%
Avg	66%	0.065	0%	100%	0.010	92%



- ***R*** → Registers written at least once
- ***S*** → Standard Deviation of Registers write operations
- ***X*** → Time of the data encrypted in registers

[4] A. Palumbo et al. "Built-in Software Obfuscation for Protecting Microprocessors against Hardware Trojan Horses." 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). IEEE, 2023

Hardware Vulnerabilities (again)

Introduction

- **Hardware Trojan Horses**
- **Reverse Engineering**
- **IP Piracy**
  - IP cloning
- **Side-Channel Attacks**
  - **Microarchitectural SCAs**
  - Physical Attacks
- **Counterfeiting**
  - Overproduction
  - IC cloning
- **Backdoors**
  - Circuit modifications leaking secrets
- **Tampering**
  - **FPGA bitstream modifications**

Side-Channel Attacks

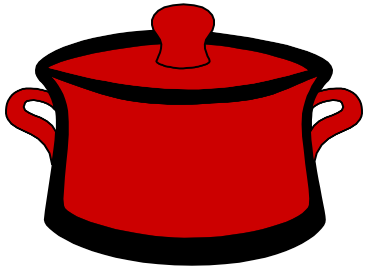
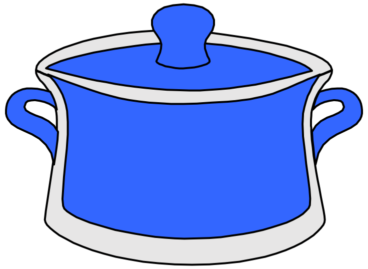
Background

- **What is a Side-Channel Attack?**
  - Exploitation (unintended) for information leakage of computing devices or implementations to infer sensitive information
    - **Microarchitectural Side-Channel Attacks don’t require to have physical access to the attacked system**
- **What a Side-Channel Attack can do?**
  - Leak information
  - Inject a Fault

A Simple Game to Understand SCA

Background

1. **You put 28 in one of the pots and 10 in the other**
2. Multiply the contents of the red pot by 7 and the contents of the blu pot by 10

7 x  +  x 10 =

3. Add the two results

Is the sum odd or even?

A Simple Game to Understand SCA

Background

1. **You put 28 in one of the pots and 10 in the other**
2. Multiply the contents of the red pot by 7 and the contents of the blu pot by 10

7 x 28 + 10 x 10 = 296

3. Add the two results

The sum is even

A Simple Game to Understand SCA

Background

1. **You put 28 in one of the pots and 10 in the other**
2. Multiply the contents of the red pot by 7 and the contents of the blu pot by 10

7 x 10 + 28 x 10 = 350

3. Add the two results

The sum is even too

Is This Really a Game?

Background

- **Is the answer enough to reveal what's in each pot?**

7 x 10 + 28 x 10 = 350

7 x 28 + 10 x 10 = 296

In both cases, we have even numbers…

However, **just by monitoring the time it takes to answer, we can discover where each amount is**

(the mental calculation leading to 296 is a bit more complicated than the one leading to 350)

TIMING ATTACK!

Flush + Reload Attack

How can an attacker know if someone is using a particular line of cache?

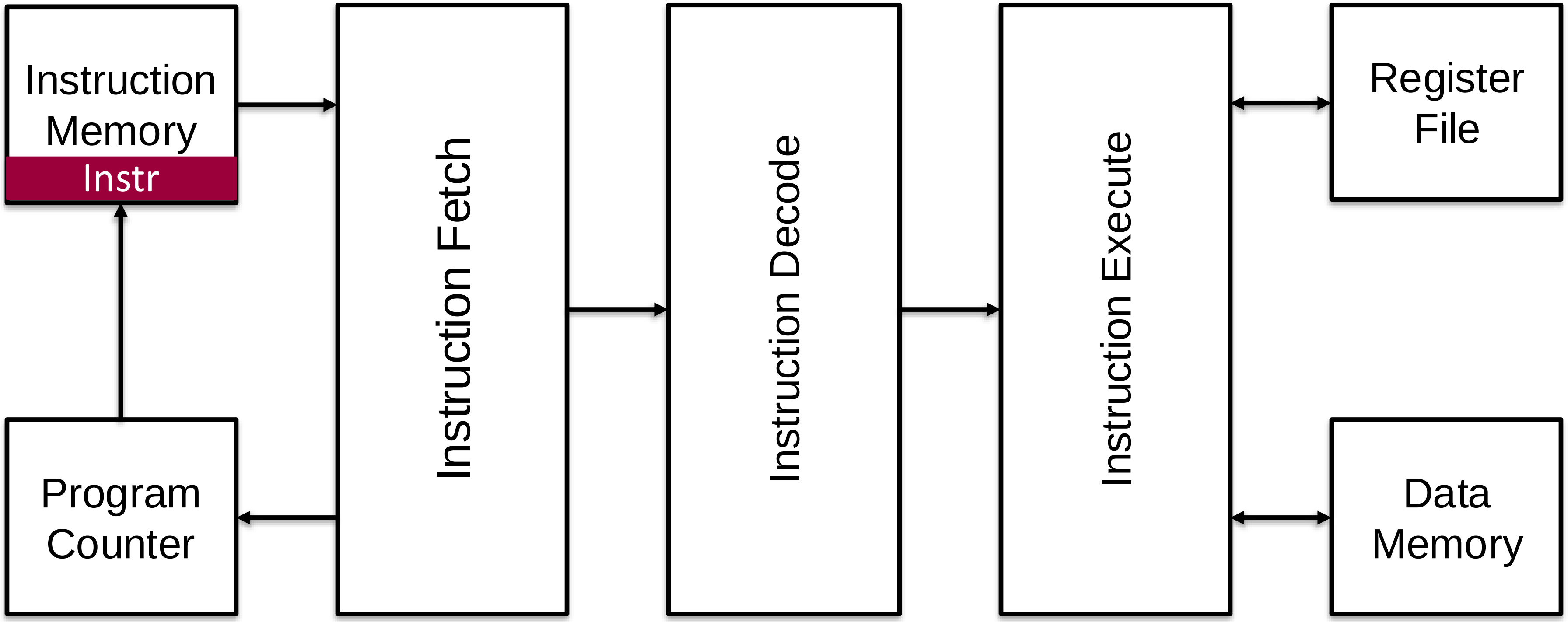
- **Attack iteration**
  - **Phase 1:** The monitored memory line is flushed from the cache
  - **Phase 2:** The attacker waits to allow the victim to access that memory line
  - **Phase 3:** The spy reloads the memory line, measuring the time to load it

If during the wait phase the victim accesses the memory line, the line will be available in the cache and the reload operation will take a short time.

If, on the other hand, the victim has not accessed the memory line, the line will need to be brought from the memory and the reload will take longer

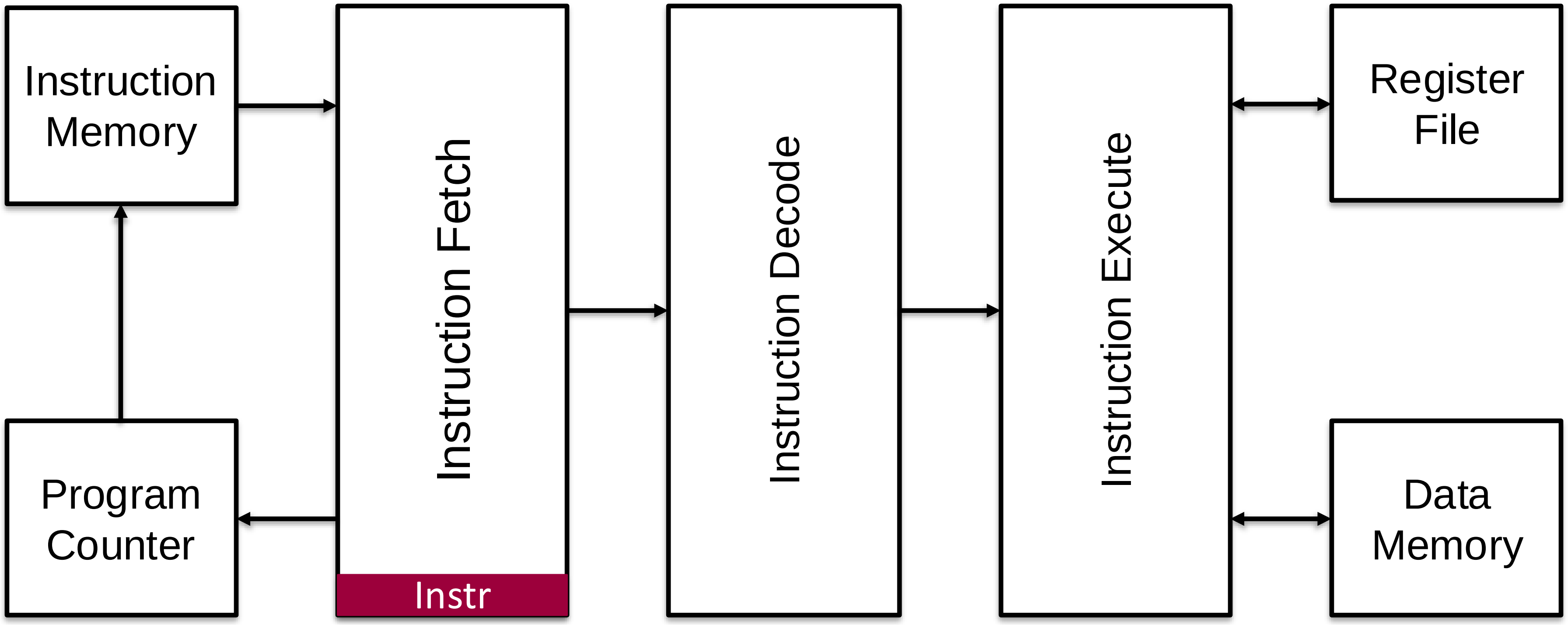
CPU: The Basic Idea

Background



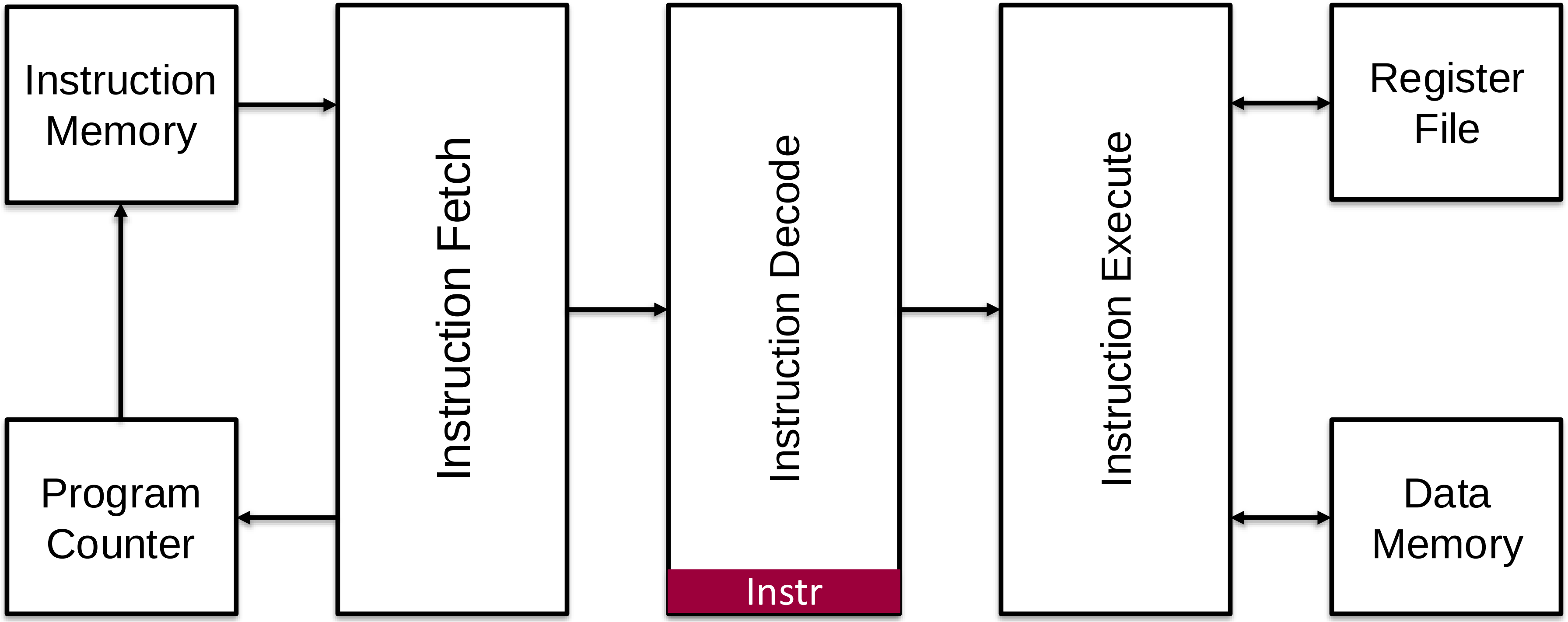
CPU: The Basic Idea

Background



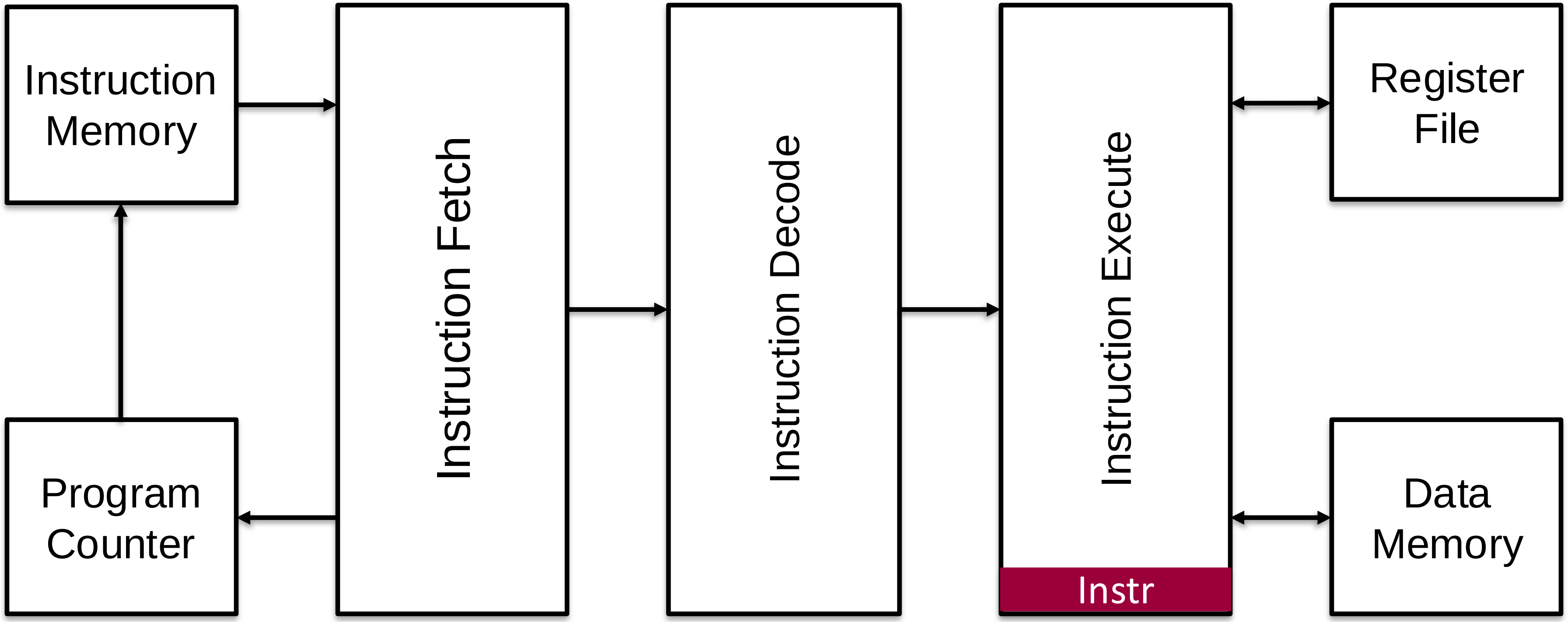
CPU: The Basic Idea

Background



CPU: The Basic Idea

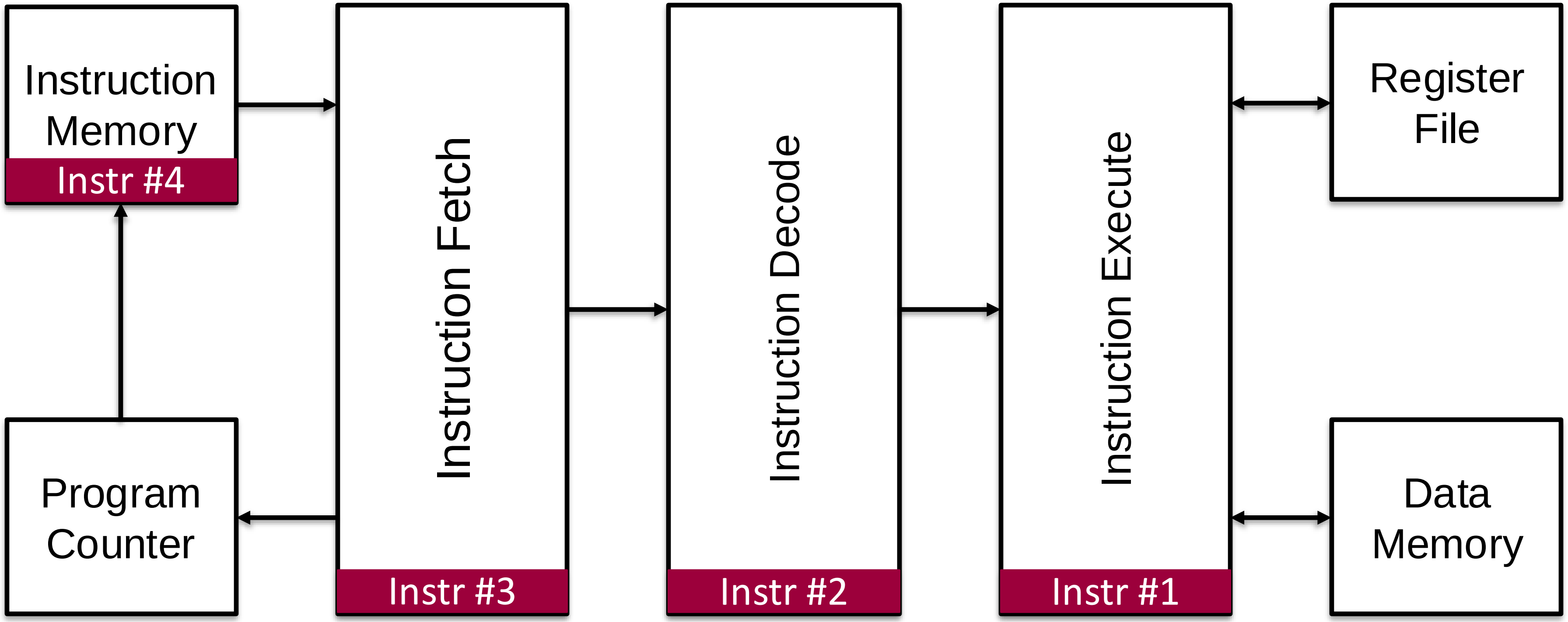
Background



While the instruction is in one stage, other stages are idle. Need to pipeline instructions to increase throughput

CPU: Pipelined Architecture

Background



Throughput improved, but what about branches instructions? Jump addresses are calculated in IE stage, which instructions are loaded in ID and IF stage?

Managing Branches

Background

- **Stall the pipeline**
  - Do not put anything in IF and ID and wait for the IE to determine what the next instruction to be fetched (**poor performances**)

- **Branch prediction**
  - Use hardware blocks to “learn” from code which branches are most likely to be taken to increase the rate of correct predictions

Speculative Execution

Background

- **Branch prediction uses hardware blocks to “learn” from code which branches are most likely to be taken to increase the rate of correct predictions**
    - Speculating on what is going to be the next instruction to be executed

But what happens if the prediction is wrong?

Handling Mispredictions

Background

- **The CPU saves his state to be able to roll back if a misprediction occurs**
  - Results of transient instructions are not committed to memory or registers until the CPU knows that the prediction is correct

But what if a transient instruction reads data from RAM?

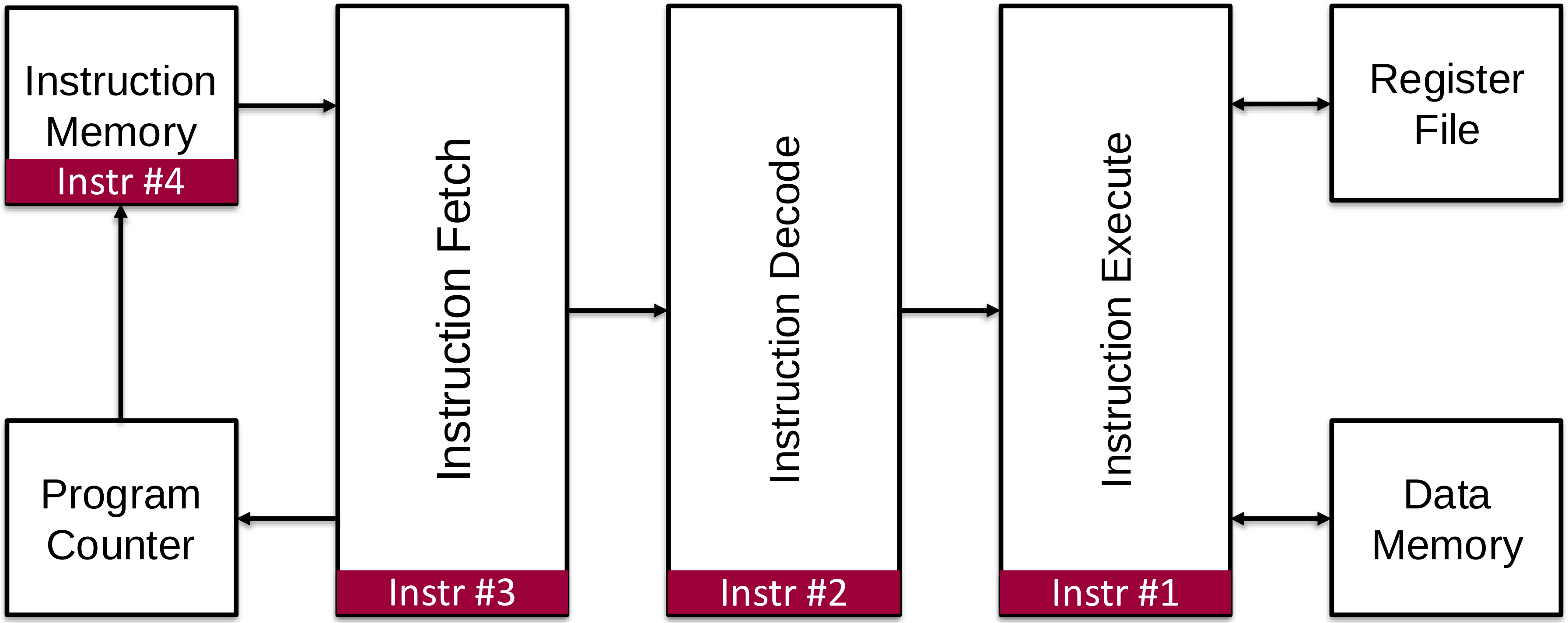
Data is fetched from RAM and copied inside the cache. **The CPU** will abort the execution due to misprediction and will **roll back its state.**

Its state, not the cache! Transient instructions may leave footprints even after CPU roll back

SPECTRE ATTACK!

CPU: Pipelined Architecture (again)

Background



What if Instr #2 depends on Instr #1 result?

Read After Write

What about if Instr #2 depends on the results of Instr #1?

Instr #1: ldw $r1, 0x67 // load in $r1 the content of 0x67

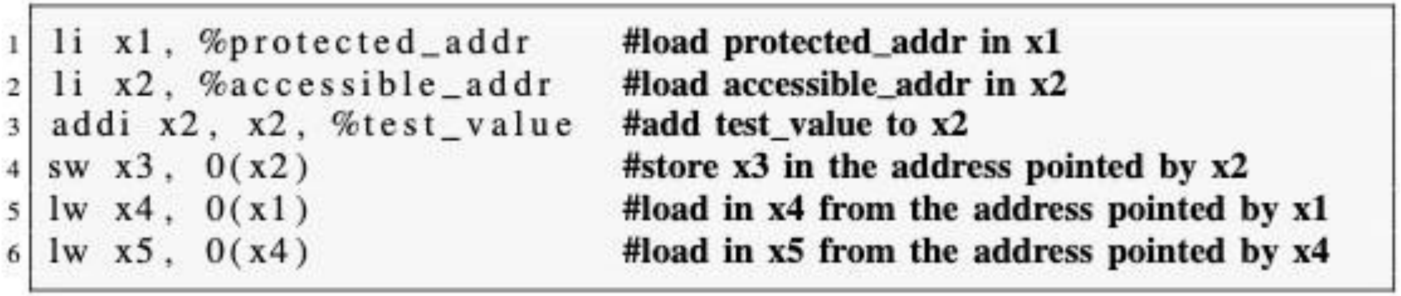
Instr #2: add $r2, $r1 // add to $r2 $r1

- **When Instr #1 is writing the result of execution in the register file, Instr #2 is in the execute stage**
  - It may take the old value of $r1
- **This may be solved by waiting for the writeback of Instr #1:**

READ AFTER WRITE: May be a problem?

Intentional Read After Write

May RAW be a problem?



```
1  li x1, %protected_addr      #load protected_addr in x1
2  li x2, %accessible_addr     #load accessible_addr in x2
3  addi x2, x2, %test_value    #add test_value to x2
4  sw x3, 0(x2)                #store x3 in the address pointed by x2
5  lw x4, 0(x1)                #load in x4 from the address pointed by x1
6  lw x5, 0(x4)                #load in x5 from the address pointed by x4
```

- **The attacker tries to guess x1 value, by iteratively increasing x2;**
  - x1 is not accessible by the attacker

- **Instr #4 is the first instruction of the intentional RAW;**
- **Instr #5 use the protected data in x1 as memory address;**
- **Instr #6 is the second instruction of the intentional RAW.**

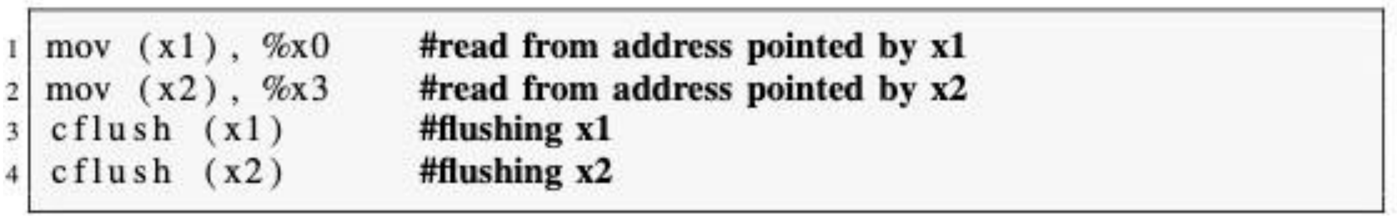
If the address x2 and the address x4 have the same value, the pipeline will stall

if x2 and x4 have different values the execution will be faster

ORCHESTRATION ATTACK!

RowHammer

A Side-Channel injection attack



```
mov  (x1), %x0      #read from address pointed by x1
mov  (x2), %x3      #read from address pointed by x2
cflush (x1)         #flushing x1
cflush (x2)         #flushing x2
```

- **DRAM technology has contiguous cells electrically interact between themselves causing a charge leak** (x1 and x2 in different memory rows, but in the same bank)
  - This unintended charge transfer may cause an unwanted change in the content of memory rows that are near the accessed row

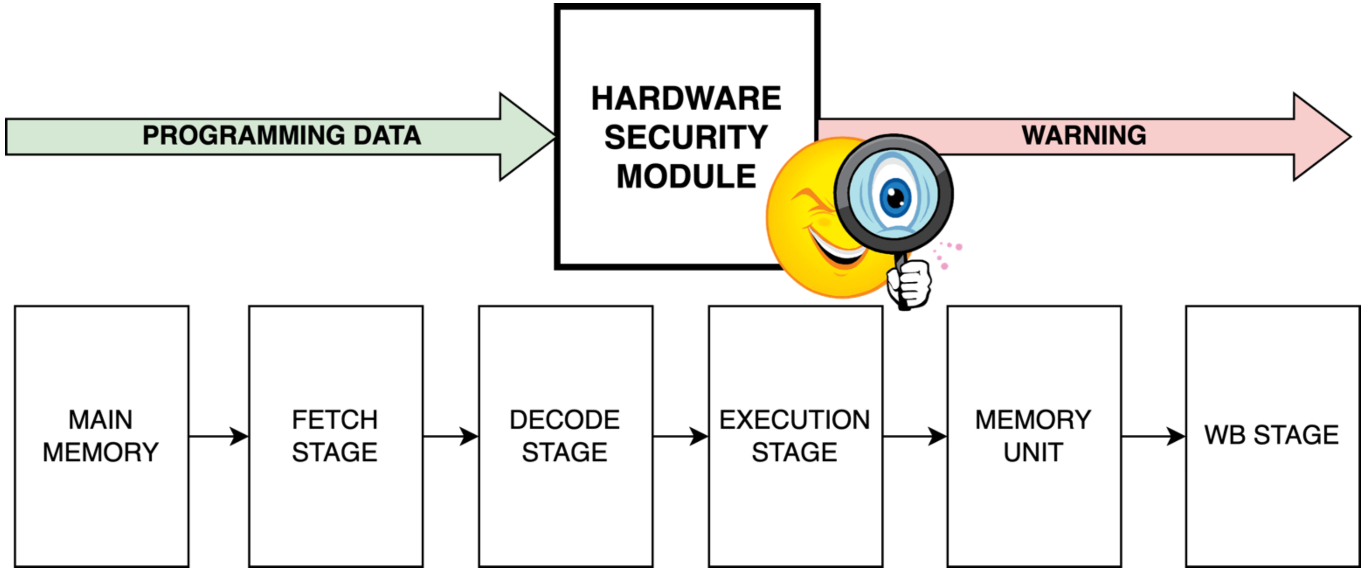
By iteratively accessing and flushing (hammering) memory locations, an attacker will be able to flip the content of the adjacent cell.

<u>ROWHAMMER ATTACK!</u>

Architectural Countermeasure Approach (again)

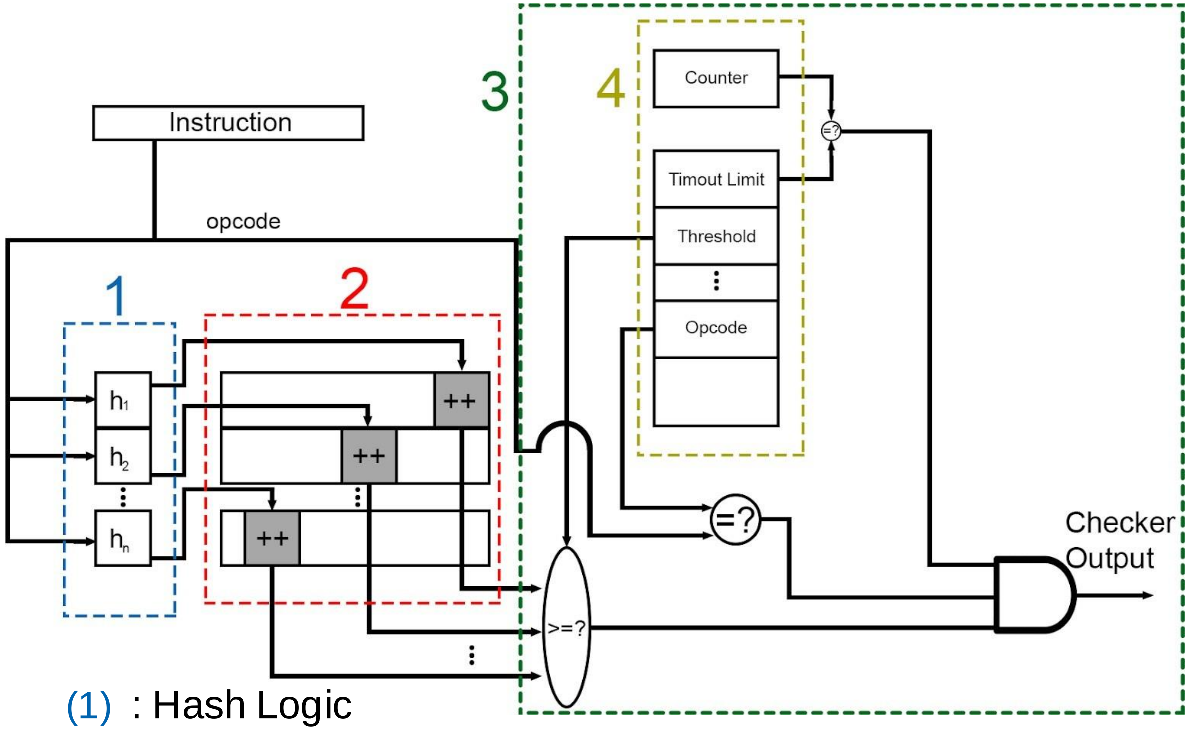
Side Channel Attacks & Microarchitectural Vulnerabilities

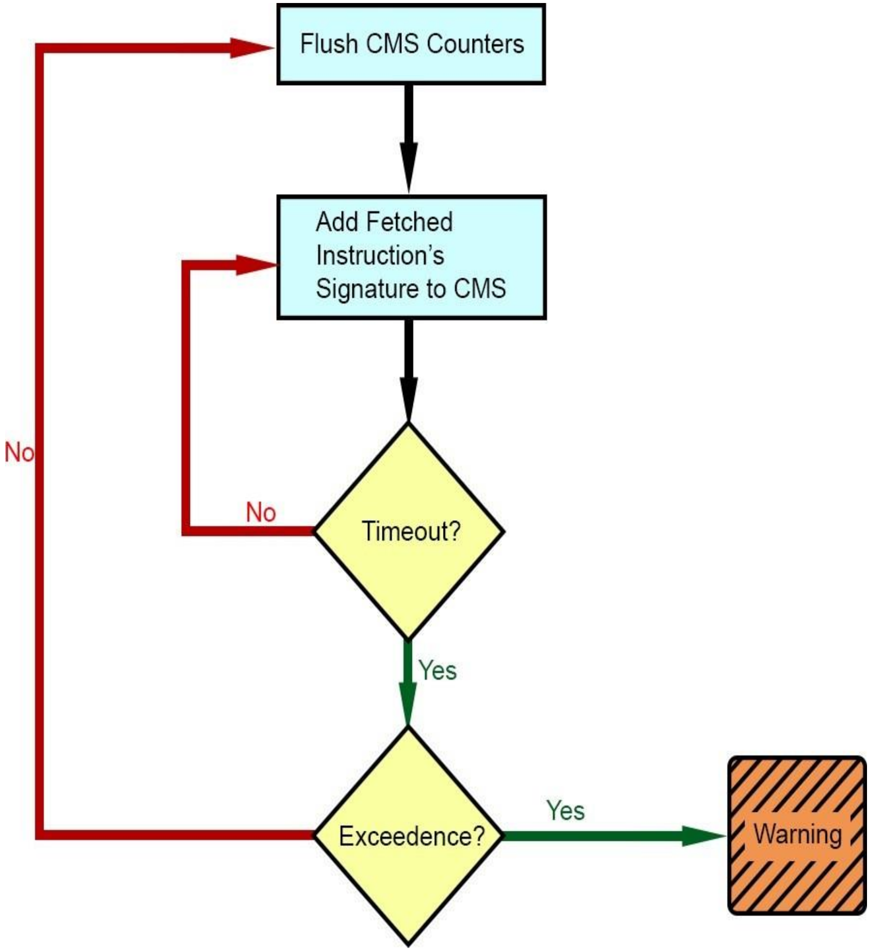
- **Add an online checker to analyze and detect potential malicious software running**
  - The programmability is useful to specify what attacks we want to detect



Architectural Countermeasure 1/2 – Hash-based

Side Channel Attacks & Microarchitectural Vulnerabilities – Workflow





(1) : Hash Logic

(2) : Memories

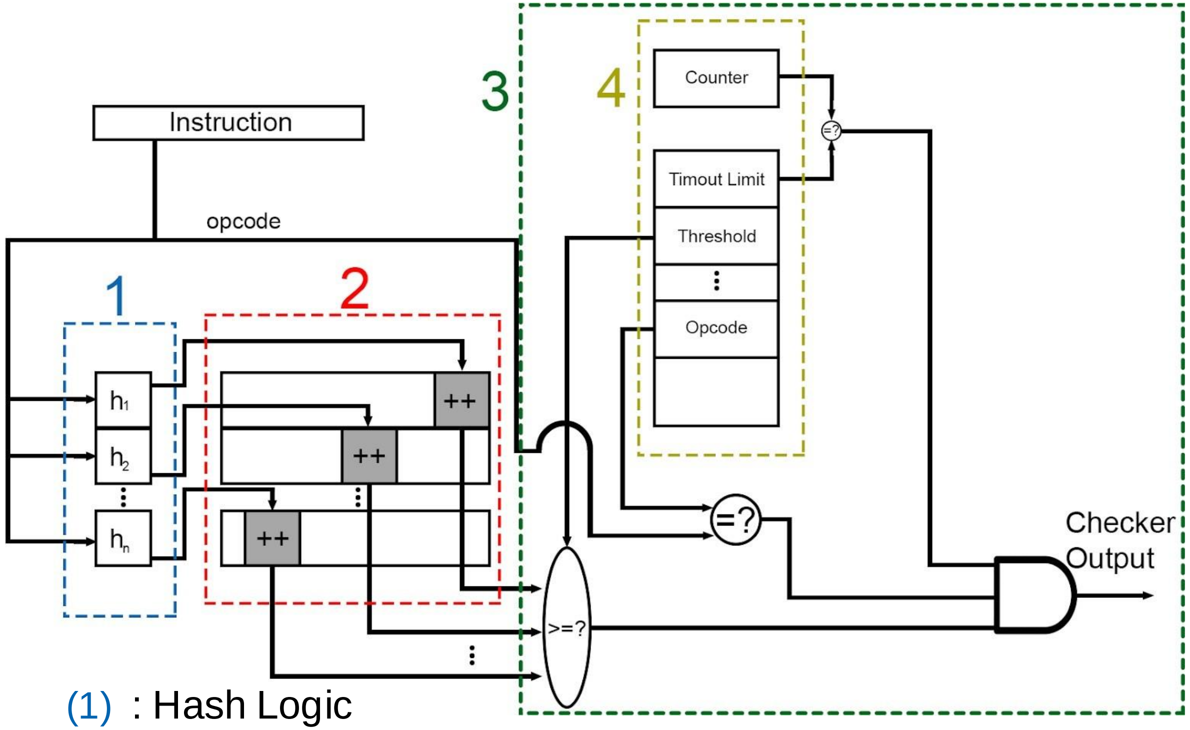
(3) : Checking Module

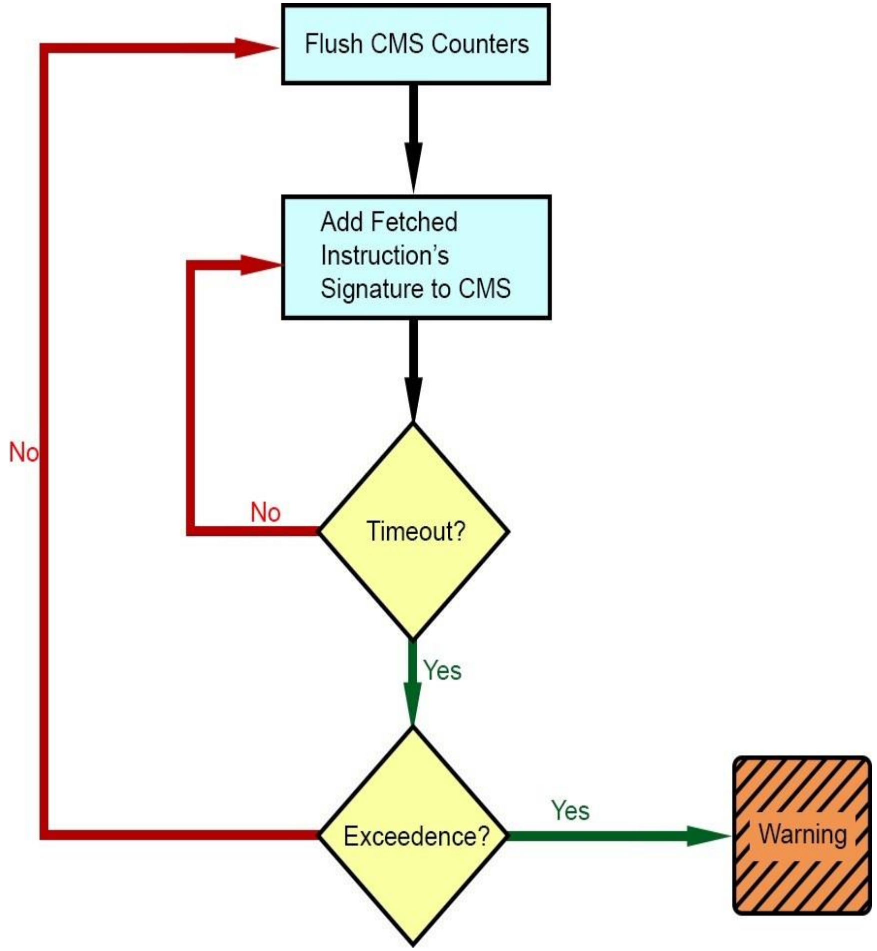
(4) : Programmable Attack Model Description Module

[5] K. Arıkan, A. Palumbo et al., "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 7, pp. 938-951, July 2022.

Architectural Countermeasure 1/2 – Hash-based

Side Channel Attacks & Microarchitectural Vulnerabilities – Workflow





(1) : Hash Logic

(2) : Memories

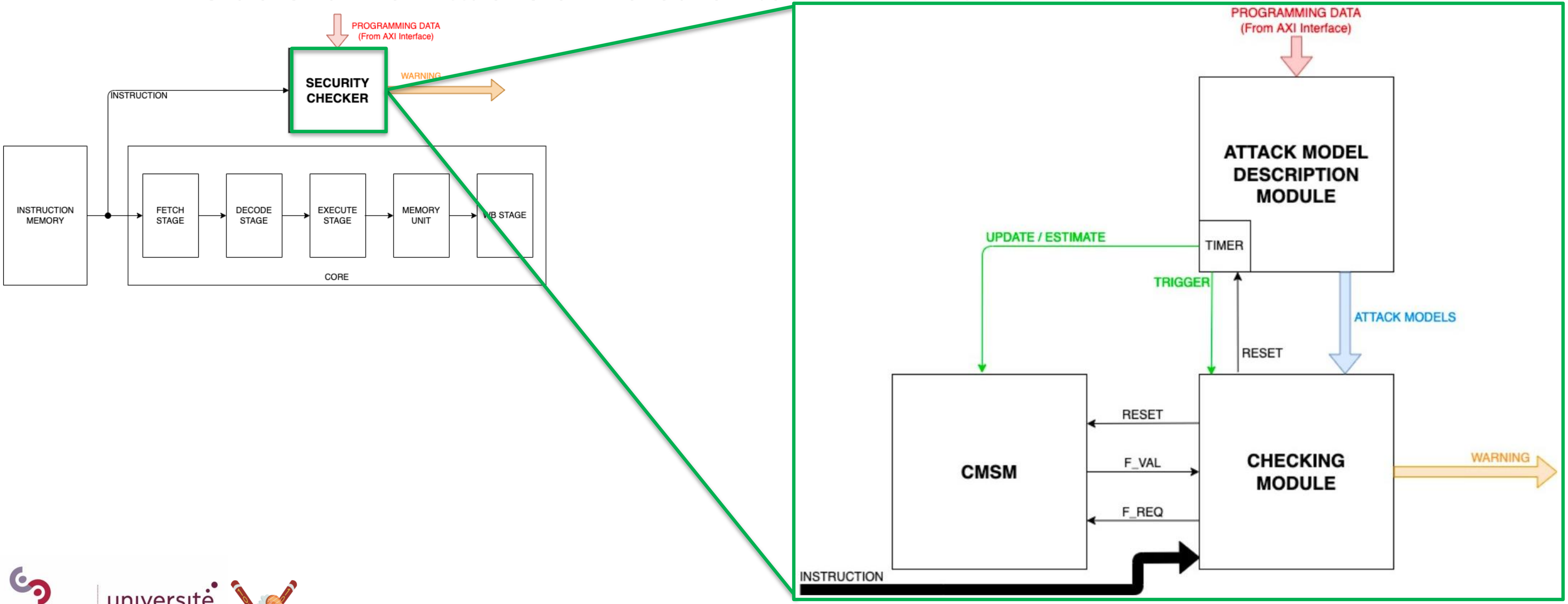
(3) : Checking Module

(4) : Programmable Attack Model Description Module

[5] K. Arıkan, A. Palumbo et al., "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 7, pp. 938-951, July 2022.

Architectural Countermeasure 1/2 – Hash-based

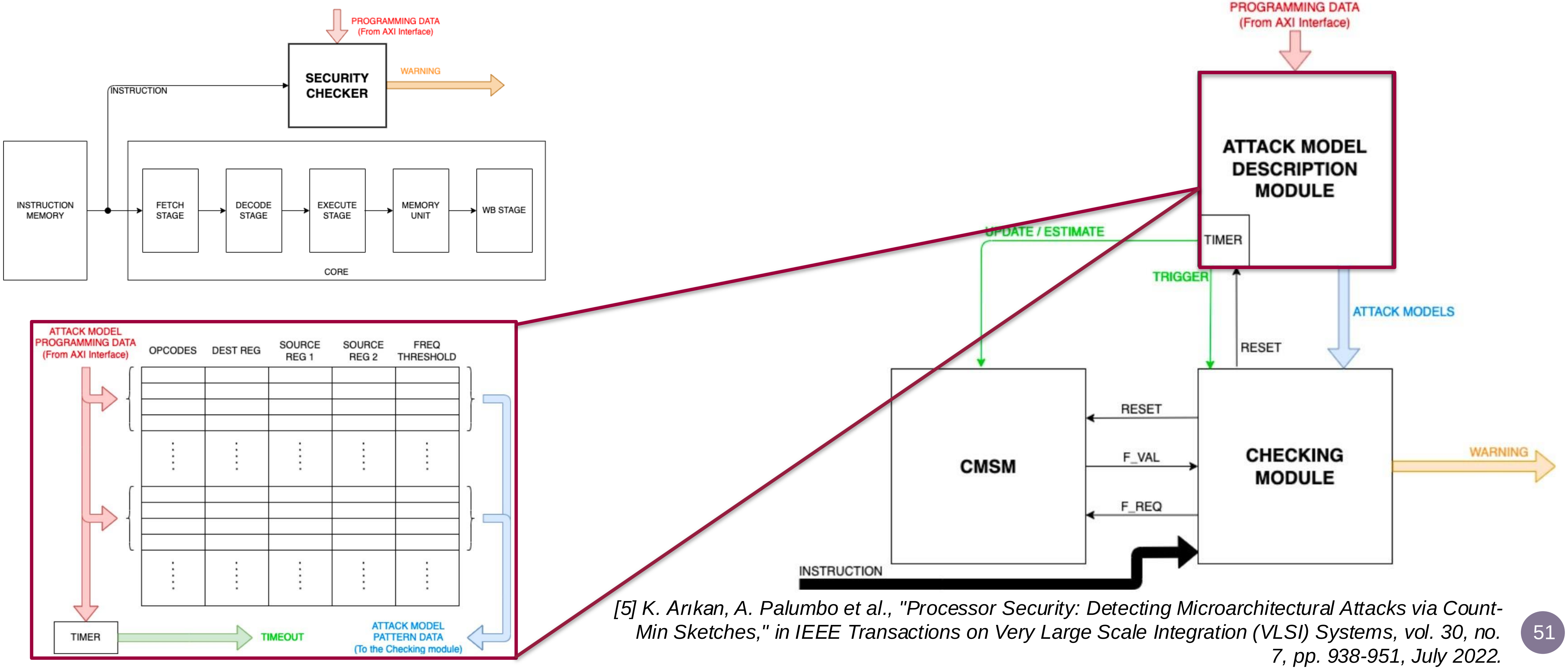
Side Channel Attacks & Microarchitectural Vulnerabilities – Workflow



[5] *K. Arıkan, A. Palumbo et al., "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 7, pp. 938-951, July 2022.*

Architectural Countermeasure 1/2 – Hash-based

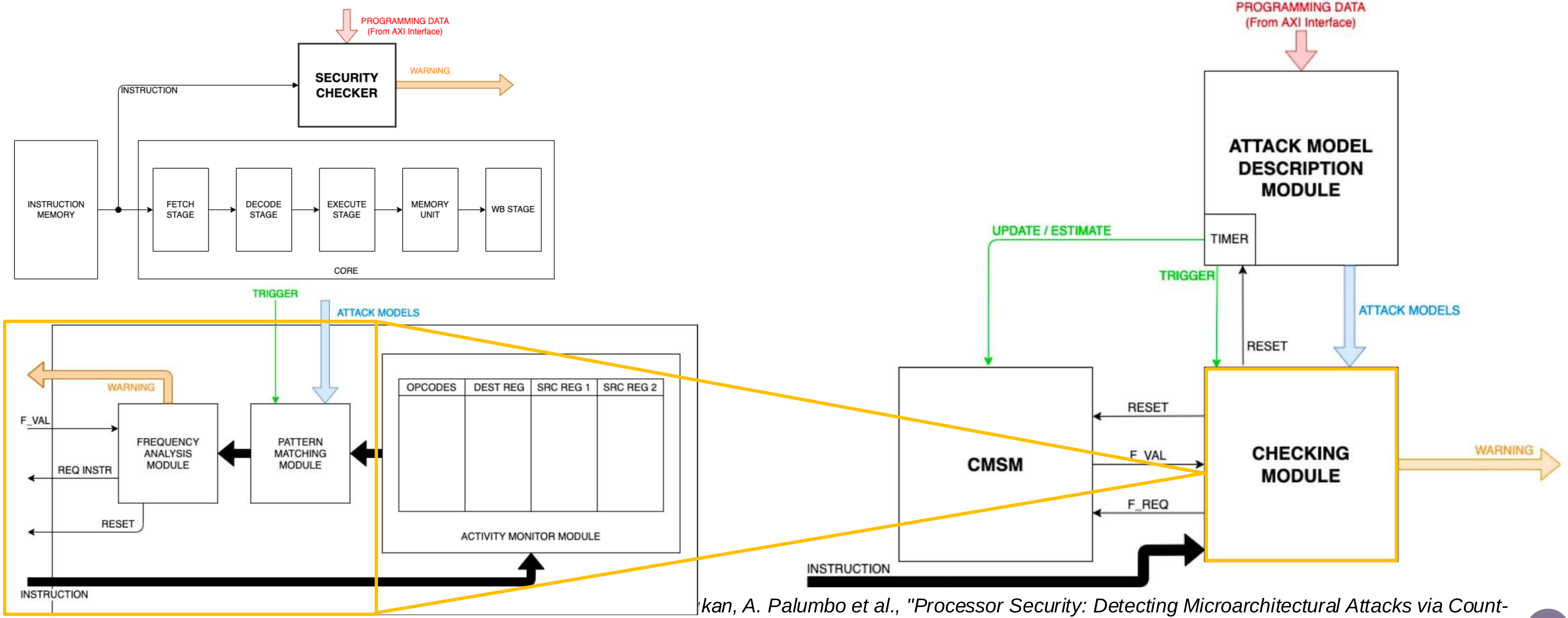
Side Channel Attacks & Microarchitectural Vulnerabilities – Workflow



[5] K. Arıkan, A. Palumbo et al., "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 7, pp. 938-951, July 2022.

Architectural Countermeasure 1/2 – Hash-based

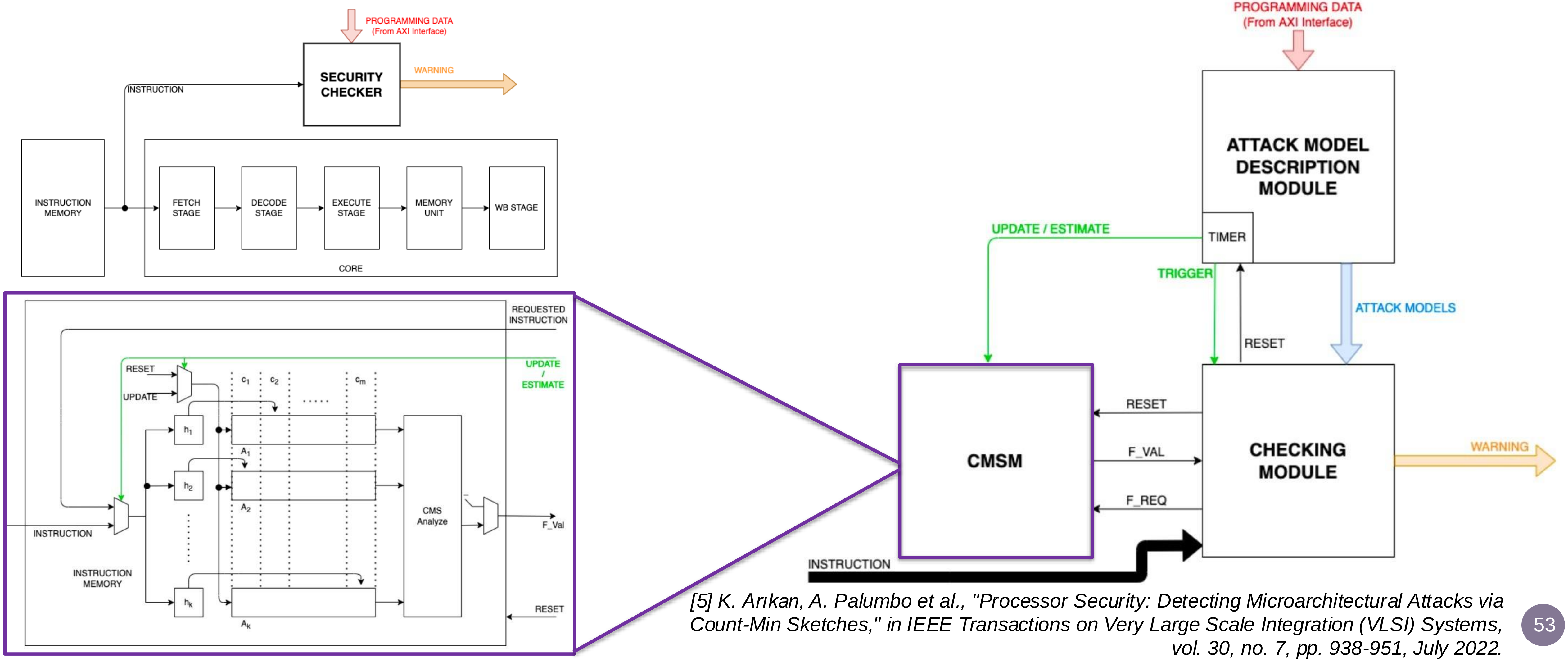
Side Channel Attacks & Microarchitectural Vulnerabilities – Workflow



kan, A. Palumbo et al., "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 7, pp. 938-951, July 2022.

Architectural Countermeasure 1/2 – Hash-based

Side Channel Attacks & Microarchitectural Vulnerabilities – Workflow

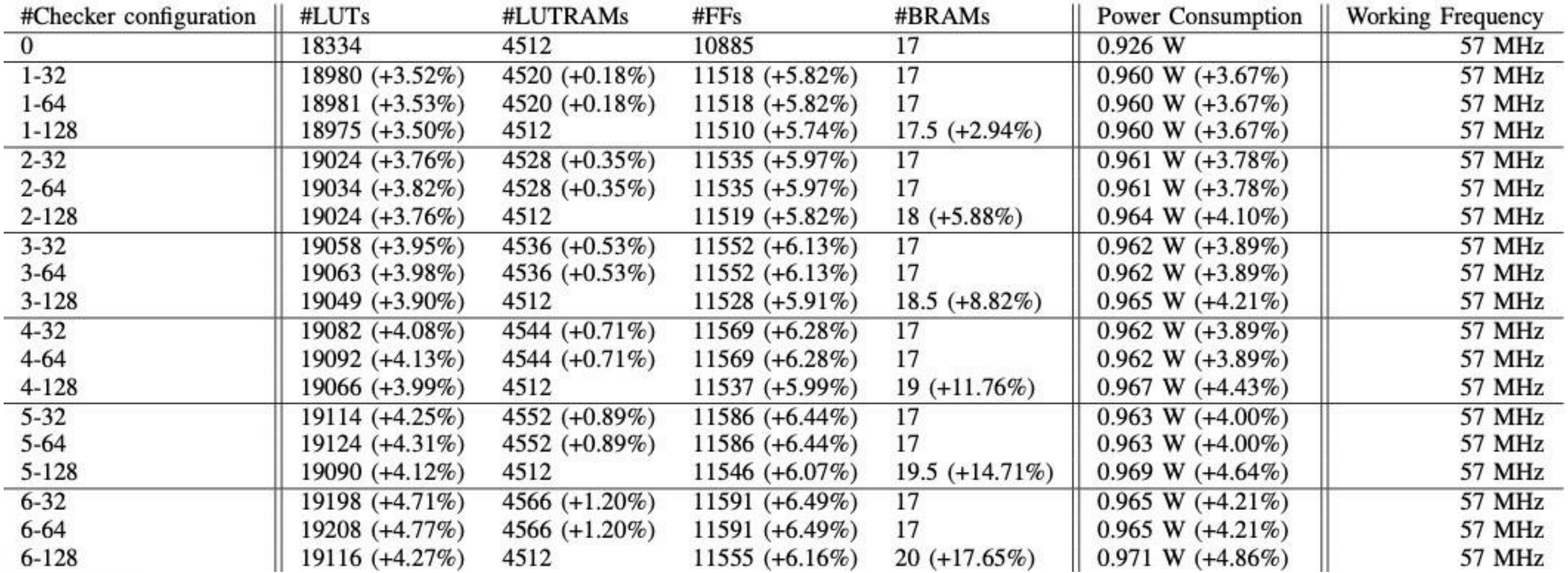


[5] K. Arıkan, A. Palumbo et al., "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 7, pp. 938-951, July 2022.

Architectural Countermeasure 1/2 – Hash-based

Hardware Trojans Interfering with Fetching Instruction Activity

- **FPGA Emulation: Resources usage compared with RISC-V Out Of Order RSD core**



#Checker configuration	#LUTs	#LUTRAMs	#FFs	#BRAMs	Power Consumption	Working Frequency
0	18334	4512	10885	17	0.926 W	57 MHz
1-32	18980 (+3.52%)	4520 (+0.18%)	11518 (+5.82%)	17	0.960 W (+3.67%)	57 MHz
1-64	18981 (+3.53%)	4520 (+0.18%)	11518 (+5.82%)	17	0.960 W (+3.67%)	57 MHz
1-128	18975 (+3.50%)	4512	11510 (+5.74%)	17.5 (+2.94%)	0.960 W (+3.67%)	57 MHz
2-32	19024 (+3.76%)	4528 (+0.35%)	11535 (+5.97%)	17	0.961 W (+3.78%)	57 MHz
2-64	19034 (+3.82%)	4528 (+0.35%)	11535 (+5.97%)	17	0.961 W (+3.78%)	57 MHz
2-128	19024 (+3.76%)	4512	11519 (+5.82%)	18 (+5.88%)	0.964 W (+4.10%)	57 MHz
3-32	19058 (+3.95%)	4536 (+0.53%)	11552 (+6.13%)	17	0.962 W (+3.89%)	57 MHz
3-64	19063 (+3.98%)	4536 (+0.53%)	11552 (+6.13%)	17	0.962 W (+3.89%)	57 MHz
3-128	19049 (+3.90%)	4512	11528 (+5.91%)	18.5 (+8.82%)	0.965 W (+4.21%)	57 MHz
4-32	19082 (+4.08%)	4544 (+0.71%)	11569 (+6.28%)	17	0.962 W (+3.89%)	57 MHz
4-64	19092 (+4.13%)	4544 (+0.71%)	11569 (+6.28%)	17	0.962 W (+3.89%)	57 MHz
4-128	19066 (+3.99%)	4512	11537 (+5.99%)	19 (+11.76%)	0.967 W (+4.43%)	57 MHz
5-32	19114 (+4.25%)	4552 (+0.89%)	11586 (+6.44%)	17	0.963 W (+4.00%)	57 MHz
5-64	19124 (+4.31%)	4552 (+0.89%)	11586 (+6.44%)	17	0.963 W (+4.00%)	57 MHz
5-128	19090 (+4.12%)	4512	11546 (+6.07%)	19.5 (+14.71%)	0.969 W (+4.64%)	57 MHz
6-32	19198 (+4.71%)	4566 (+1.20%)	11591 (+6.49%)	17	0.965 W (+4.21%)	57 MHz
6-64	19208 (+4.77%)	4566 (+1.20%)	11591 (+6.49%)	17	0.965 W (+4.21%)	57 MHz
6-128	19116 (+4.27%)	4512	11555 (+6.16%)	20 (+17.65%)	0.971 W (+4.86%)	57 MHz

[5] K. Arıkan, A. Palumbo et al., "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 7, pp. 938-951, July 2022.

Architectural Countermeasure 1/2 – Hash-based

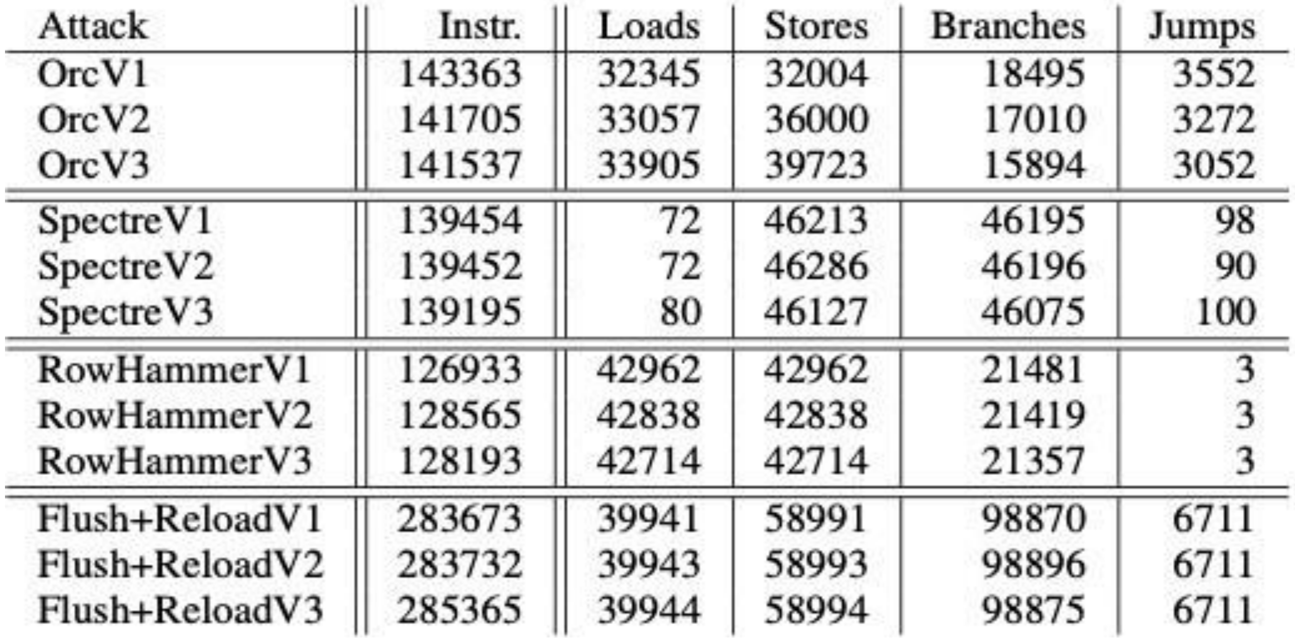
Hardware Trojans Interfering with Fetching Instruction Activity

- **Malicious Codes, three version each → 100% Accuracy, No False Negative**
  - Orchestration

  - Spectre

  - RowHammer

  - Fulsh+Reload



Attack	Instr.	Loads	Stores	Branches	Jumps
OrcV1	143363	32345	32004	18495	3552
OrcV2	141705	33057	36000	17010	3272
OrcV3	141537	33905	39723	15894	3052
SpectreV1	139454	72	46213	46195	98
SpectreV2	139452	72	46286	46196	90
SpectreV3	139195	80	46127	46075	100
RowHammerV1	126933	42962	42962	21481	3
RowHammerV2	128565	42838	42838	21419	3
RowHammerV3	128193	42714	42714	21357	3
Flush+ReloadV1	283673	39941	58991	98870	6711
Flush+ReloadV2	283732	39943	58993	98896	6711
Flush+ReloadV3	285365	39944	58994	98875	6711

[5] K. Arıkan, A. Palumbo et al., "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 7, pp. 938-951, July 2022.

Architectural Countermeasure 1/2 – Hash-based

Hardware Trojans Interfering with Fetching Instruction Activity

- **Malicious Codes, three version each → 100% Accuracy, No False Negative**
- $FP \leq e^{-k}$
- (k, m):
  - #Memories, #data memory bit

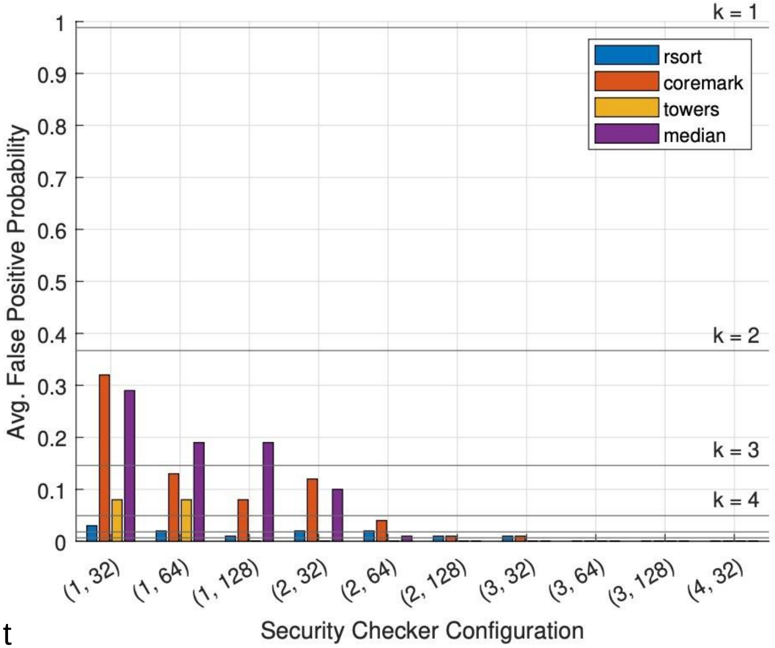


Fig. 11. Average False Positive Probability (FP_p) when attacking several configurations of the SC with the Orchestration Attack

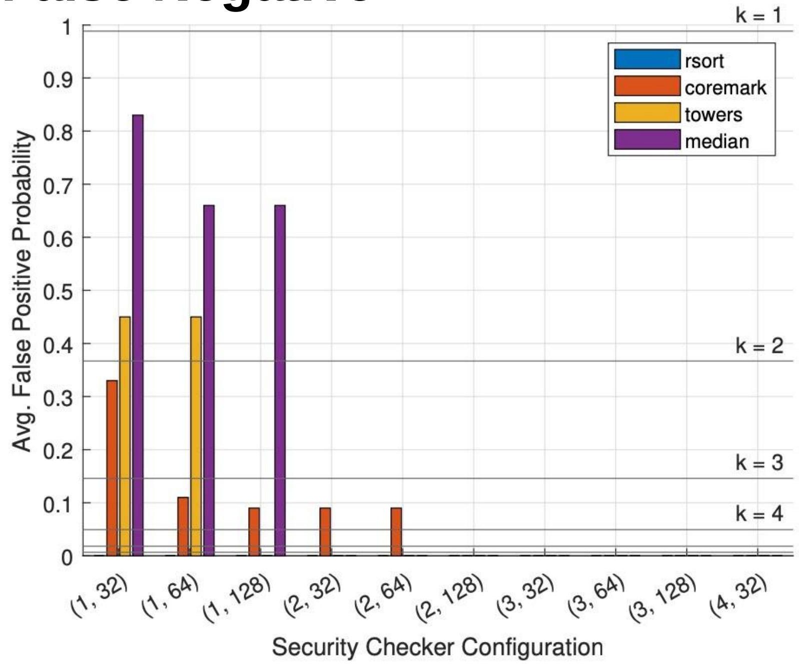


Fig. 12. Average False Positive Probability (FP_p) when attacking several configurations of the SC with the Spectre Attack

[5] K. Arıkan, A. Palumbo et al., "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 7, pp. 938-951, July 2022.

Architectural Countermeasure 1/2 – Hash-based

Hardware Trojans Interfering with Fetching Instruction Activity

- **Malicious Codes, three version each → 100% Accuracy, No False Negative**
- $FP \leq e^{-k}$
- (k, m):
  - #Memories, #data memory bit

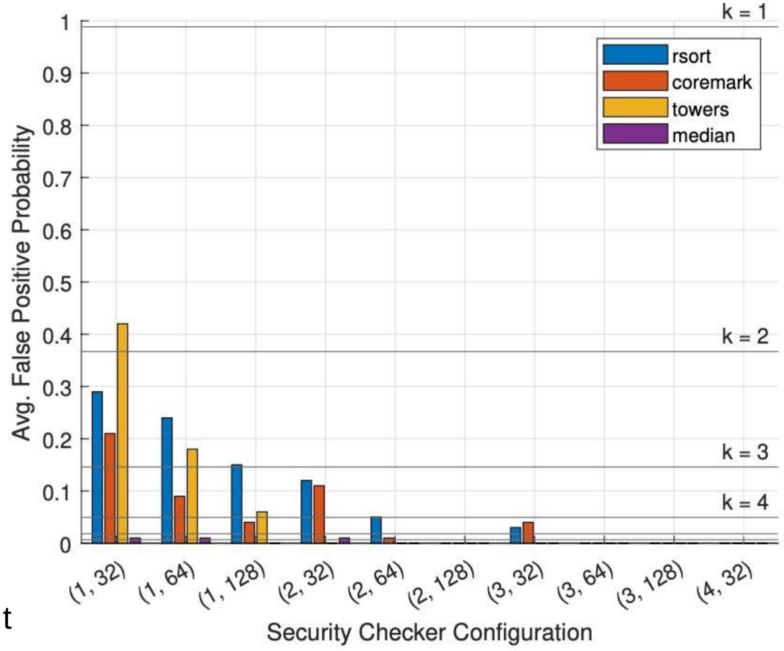


Fig. 13. Average False Positive Probability (FP_p) when attacking several configurations of the SC with the Rowhammer Attack

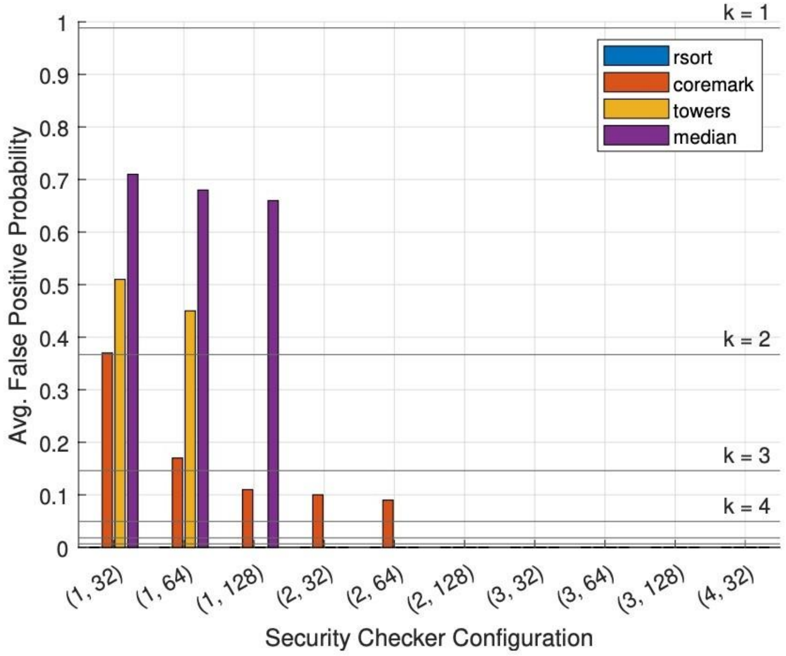


Fig. 14. Average False Positive Probability (FP_p) when attacking several configurations of the SC with the Flush+Reload Attack

[5] K. Arıkan, A. Palumbo et al., "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 7, pp. 938-951, July 2022.

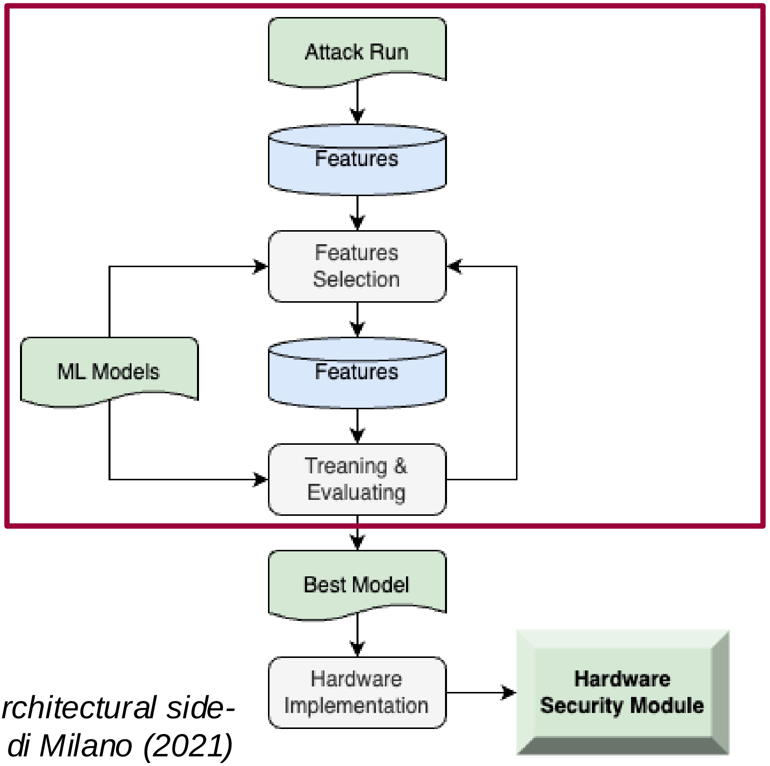
Architectural Countermeasure 2/2 Idea – ML-based

Side Channel Attacks & Microarchitectural Vulnerabilities – Workflow

1. Run the malicious software(s) on the CPU. Target ISA is RISC-V

- Features extracted via tools (gem5, verilator) or FPGA emulation:
  - Performance Counters
  - Computation Time
  - Temperature Traces
  - Power Consumption
  - …

2. **Design the HSM architecture** based on the best ML algo



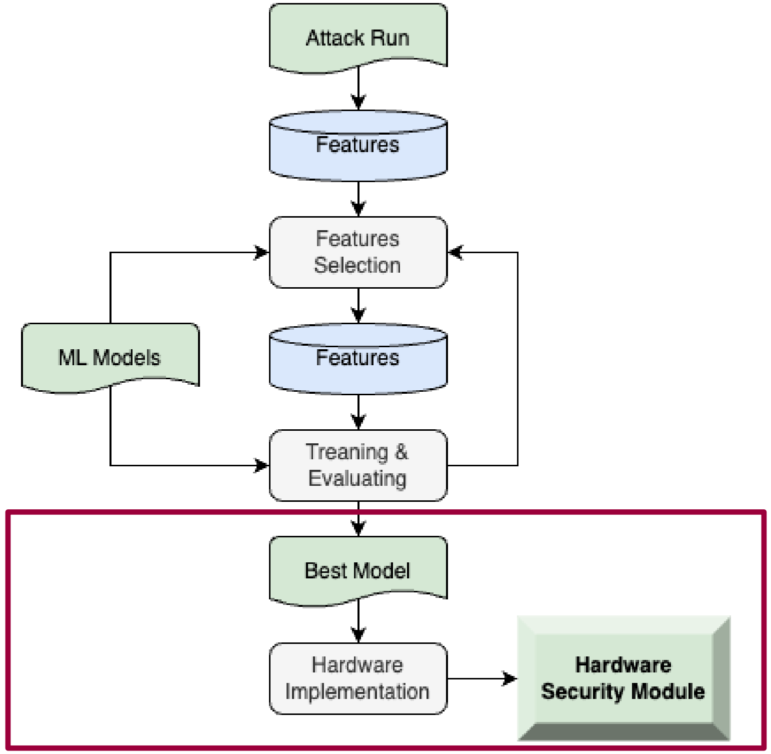
[6] M.Iamundo, "A machine learning-based security architecture to detect microarchitectural side-channel attacks in microprocessors ", Master Thesis, Politecnico di Milano (2021)

Architectural Countermeasure 2/2 Idea – ML-based

Side Channel Attacks & Microarchitectural Vulnerabilities – Workflow

1. Run the malicious software(s) on the CPU. Target ISA is **RISC-V**

- **Features** extracted via tools (gem5, verilator) or FPGA emulation:
  - Performance Counters
  - Computation Time
  - Temperature Traces
  - Power Consumption
  - …



[6] M.Iamundo, "A machine learning-based security architecture to detect microarchitectural side-channel attacks in microprocessors ", Master Thesis, Politecnico di Milano (2021)

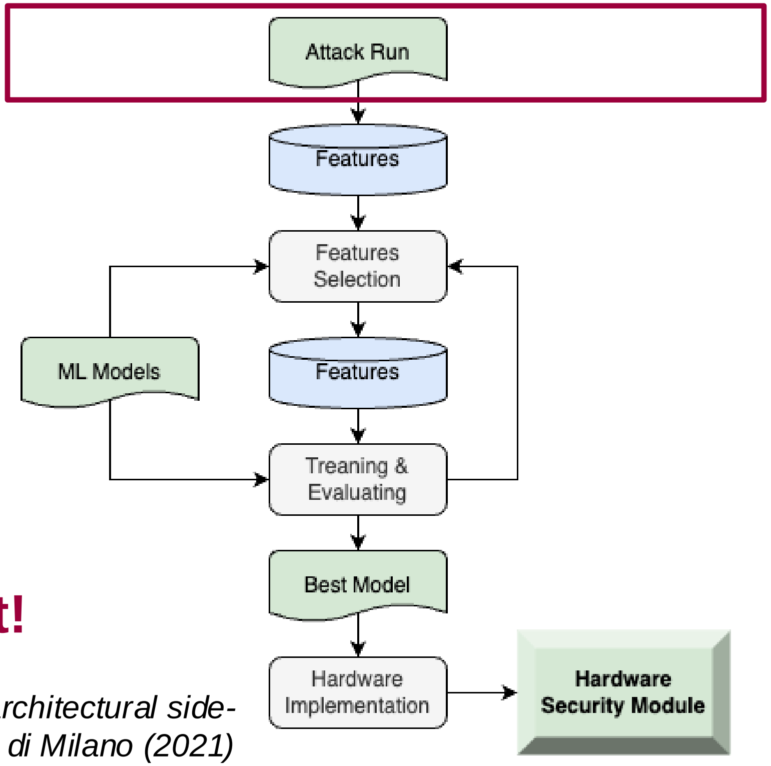
Architectural Countermeasure 2/2 Idea – ML-based

Side Channel Attacks & Microarchitectural Vulnerabilities – Workflow

1. Run the malicious software(s) on the CPU. Target ISA is RISC-V

- Features extracted via tools (gem5, verilator) or FPGA emulation:
  - Performance Counters
  - Computation Time
  - Temperature Traces
  - Power Consumption
  - …

2. Design the HSM architecture based on the best ML algo

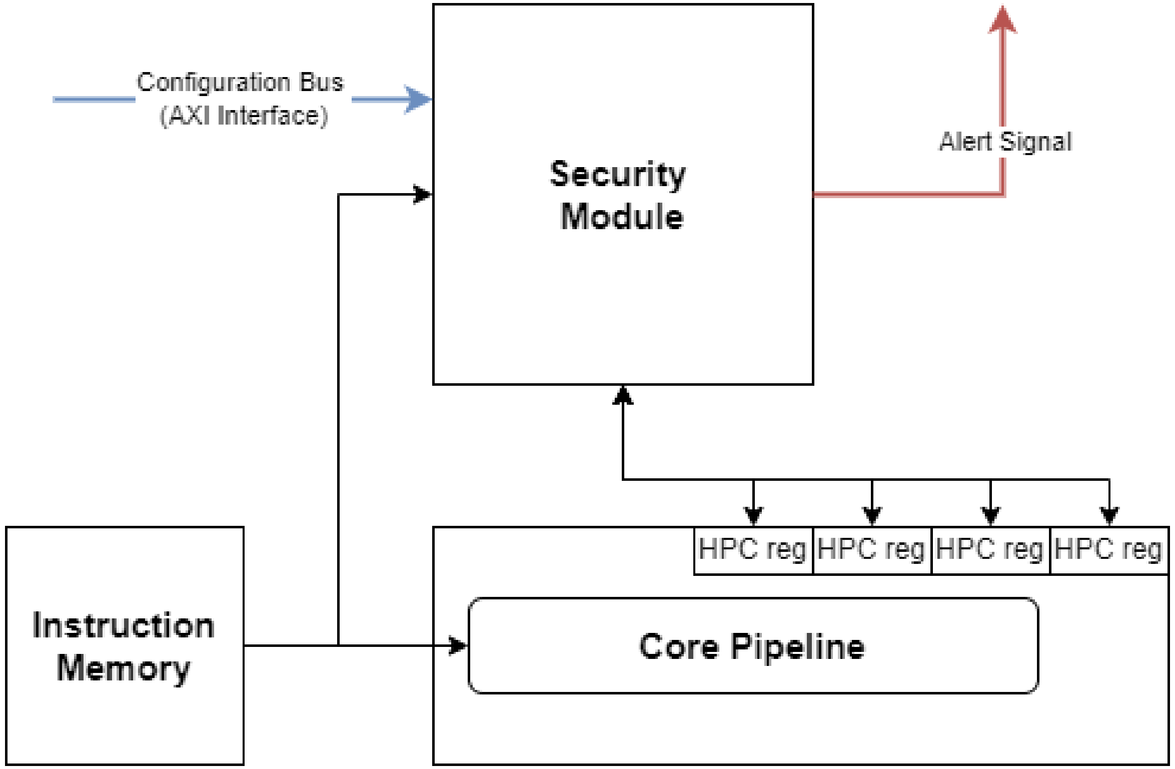


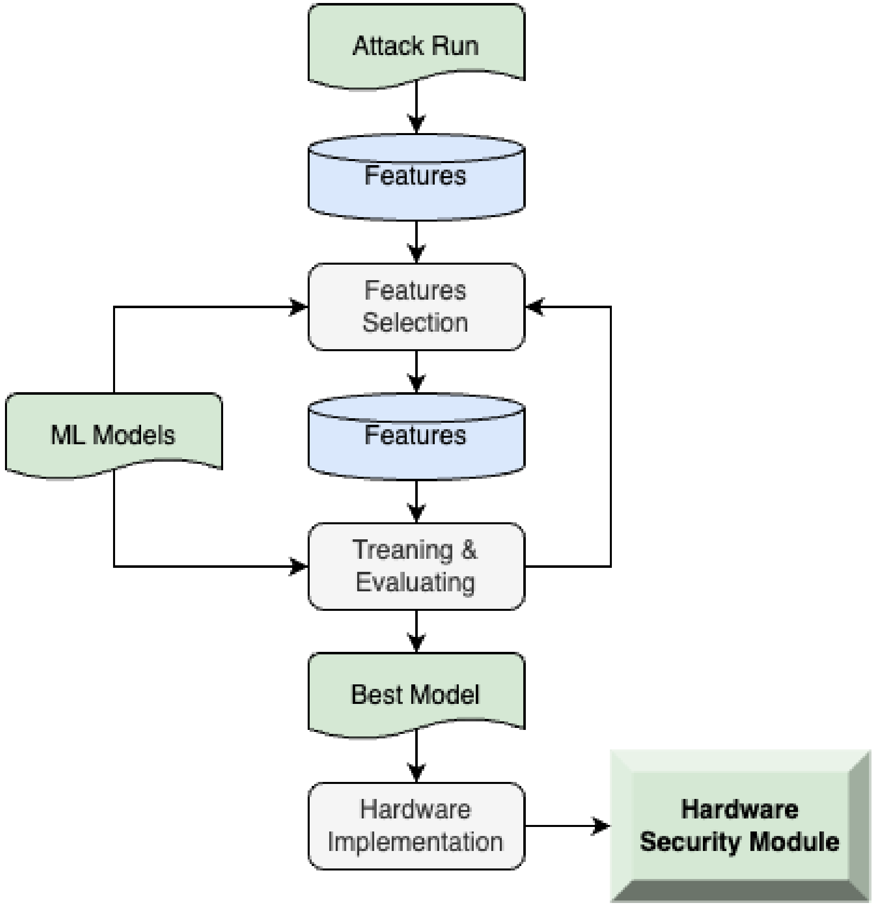
What if a new attack comes? Just restart!

[6] M.Iamundo, "A machine learning-based security architecture to detect microarchitectural side-channel attacks in microprocessors ", Master Thesis, Politecnico di Milano (2021)

Architectural Countermeasure 2/2 Idea – ML-based

Side Channel Attacks & Microarchitectural Vulnerabilities – Workflow

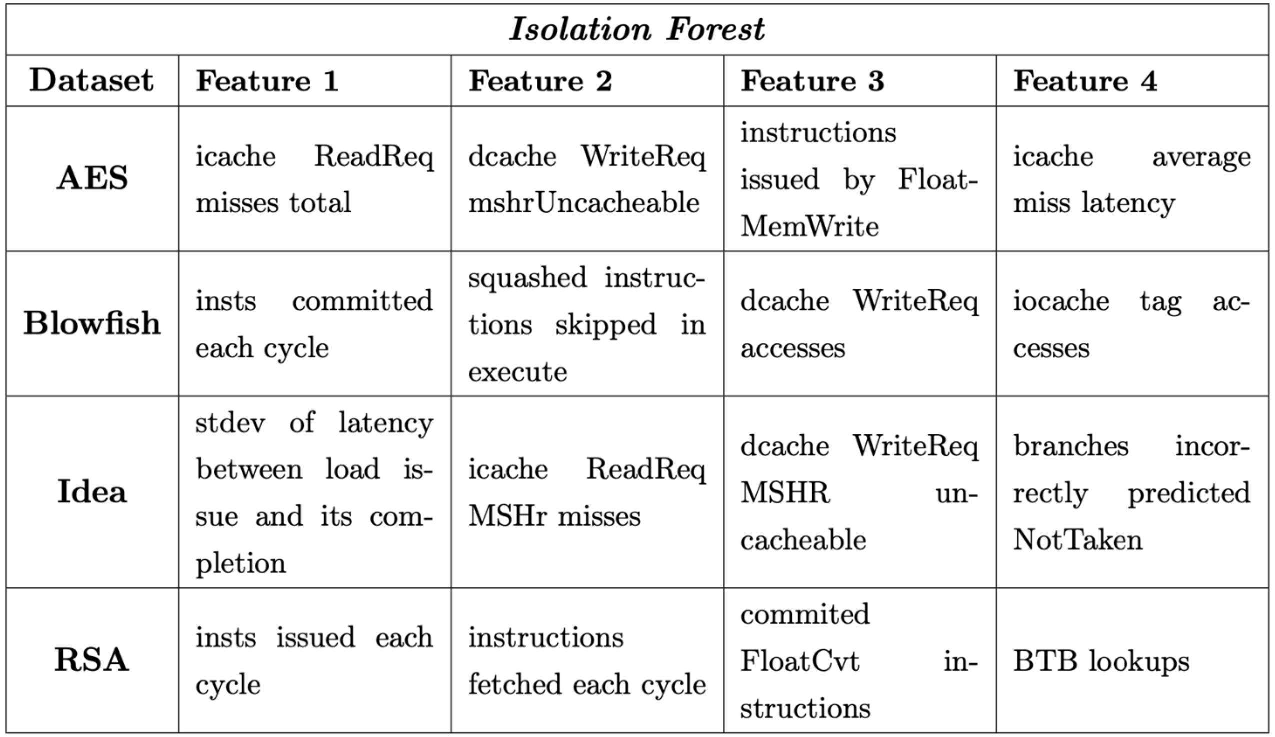




[6] *M.Iamundo, "A machine learning-based security architecture to detect microarchitectural side-channel attacks in microprocessors ", Master Thesis, Politecnico di Milano (2021)*

Architectural Countermeasure 2/2 Idea – ML-based

Side Channel Attacks & Microarchitectural Vulnerabilities – Workflow



Isolation Forest				
Dataset	**Feature 1**	**Feature 2**	**Feature 3**	**Feature 4**
AES	icache ReadReq misses total	dcache WriteReq mshrUncacheable	instructions issued by Float-MemWrite	icache average miss latency
Blowfish	insts committed each cycle	squashed instructions skipped in execute	dcache WriteReq accesses	iocache tag accesses
Idea	stdev of latency between load issue and its completion	icache ReadReq MSHr misses	dcache WriteReq MSHR uncacheable	branches incorrectly predicted NotTaken
RSA	insts issued each cycle	instructions fetched each cycle	commited FloatCvt instructions	BTB lookups

Attack	Dataset	TP %	TN %	FP %	FN %
Spectre	AES	63,35%	35,94%	0,71%	0%
	Blowfish	70,97%	28,82%	0,21%	0%
	Idea	70,8%	28,63%	0,57%	0%
	RSA	65,94%	33,7%	0,36%	0%
Meltdown	AES	67,5%	31,94%	0,56%	0%
	Blowfish	69,69%	30,01%	0,30%	0%
	Idea	67,09%	32,6%	0,31%	0%
	RSA	63,67%	36,25%	0,21%	0%

- **Hardware Overhead (#LUTs + #FFs):**
  - 6,75% in x86 Intel Nehalem (stand alone implementation)
  - RISC-V → ongoing (paper under review @ an IEE Transaction)

[6] *M.Iamundo, "A machine learning-based security architecture to detect microarchitectural side-channel attacks in microprocessors ", Master Thesis, Politecnico di Milano (2021)*

Hardware Vulnerabilities

Introduction

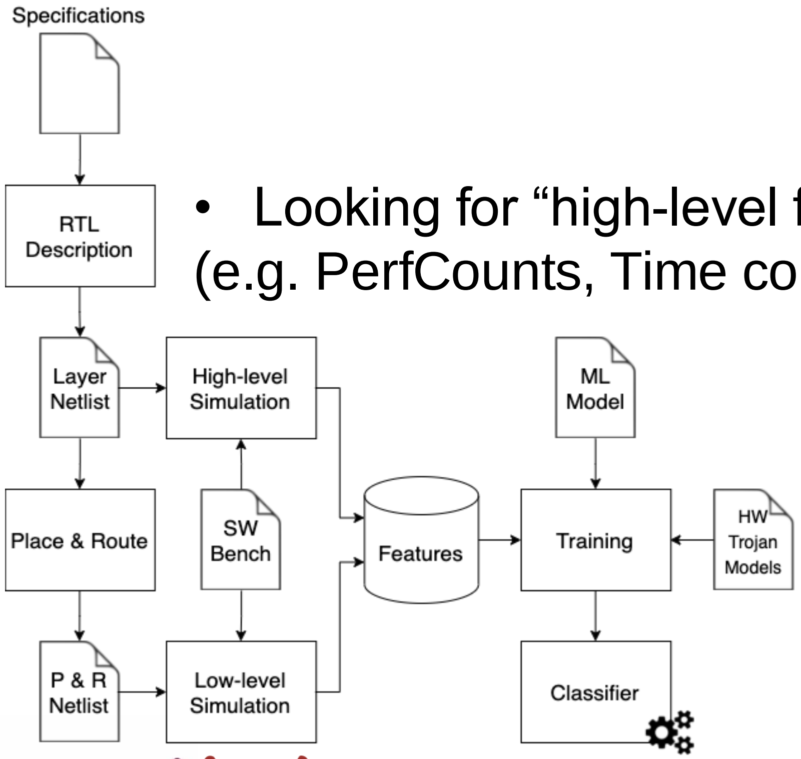
- **Trojan Horses**
- **Reverse Engineering**
- **IP Piracy**
  - IP cloning
- **Side-Channel Attacks**
  - **Microarchitectural SCAs**
  - Physical Attacks

- **Counterfeiting**
  - Overproduction
  - IC cloning
- **Backdoors**
  - Circuit modifications leaking secrets
- **Tampering**
  - **FPGA bitstream modifications**

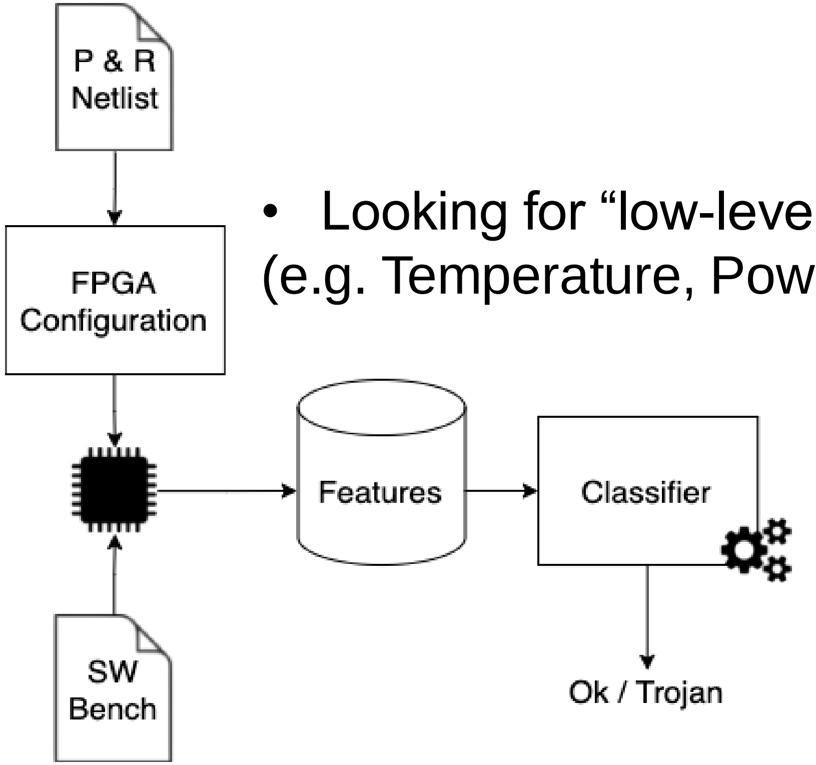
Methodology Countermeasure Idea

Tampering: FPGA Bitstream modifications

- **Are FPGAs implementing soft cores, Trojan-free? Machine Learning methodology will give the answer**



- Looking for “high-level features” (e.g. PerfCounts, Time comps)



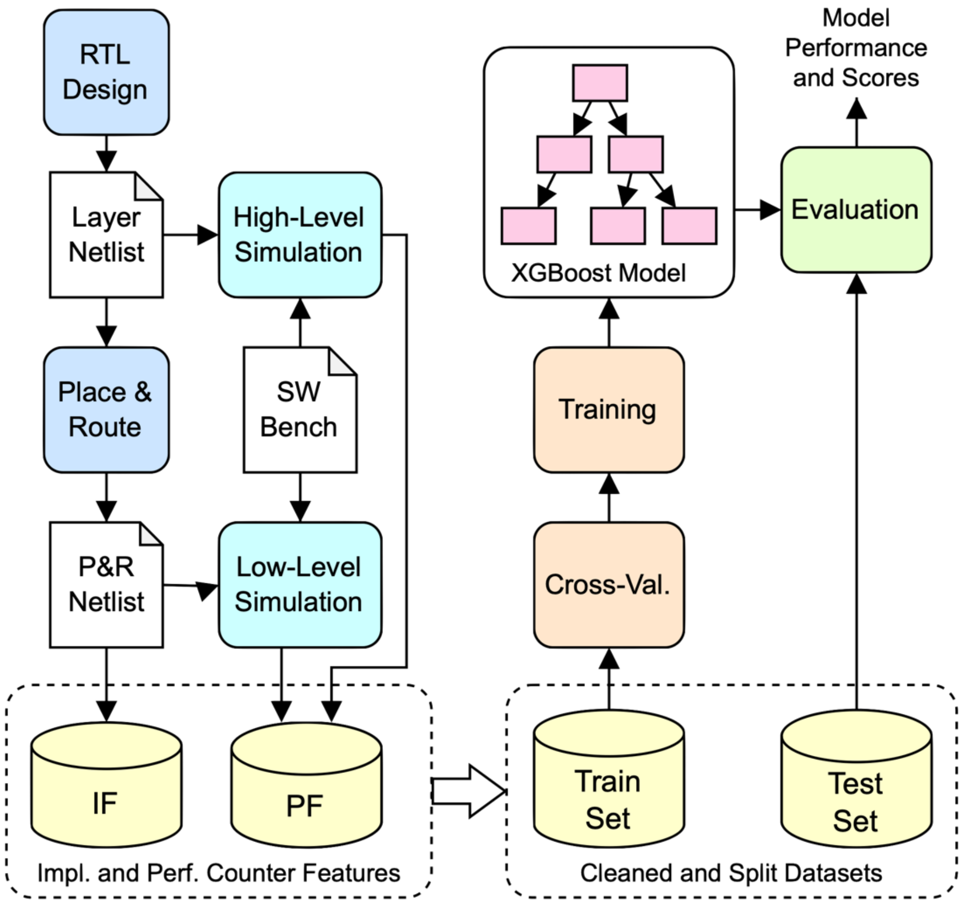
- Looking for “low-level features” (e.g. Temperature, Power)

[7] A. Palumbo, et al. "Is your FPGA bitstream Hardware Trojan-free? Machine learning can provide an answer", Journal of Systems Architecture, 128, 2022.

Methodology Countermeasure Idea

Tampering: FPGA Bitstream modifications

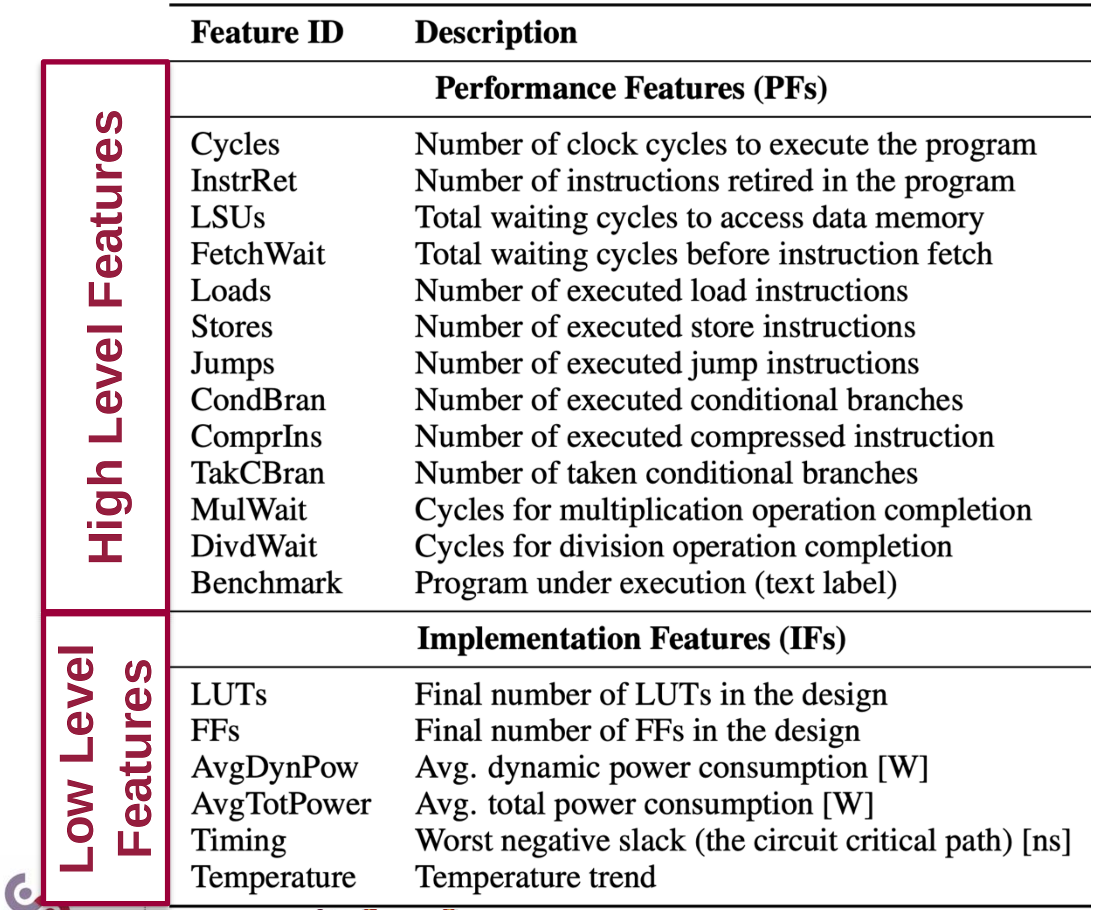
Feature ID	Description
Performance Features (PFs)	
Cycles	High Level Features
InstrRet	
LSUs	
FetchWait	
Loads	
Stores	
Jumps	
CondBran	
ComprIns	
TakCBran	
MulWait	
DivdWait	
Benchmark	
Implementation Features (IFs)	
LUTs	Low Level Features
FFs	
AvgDynPow	
AvgTotPower	
Timing	
Temperature	



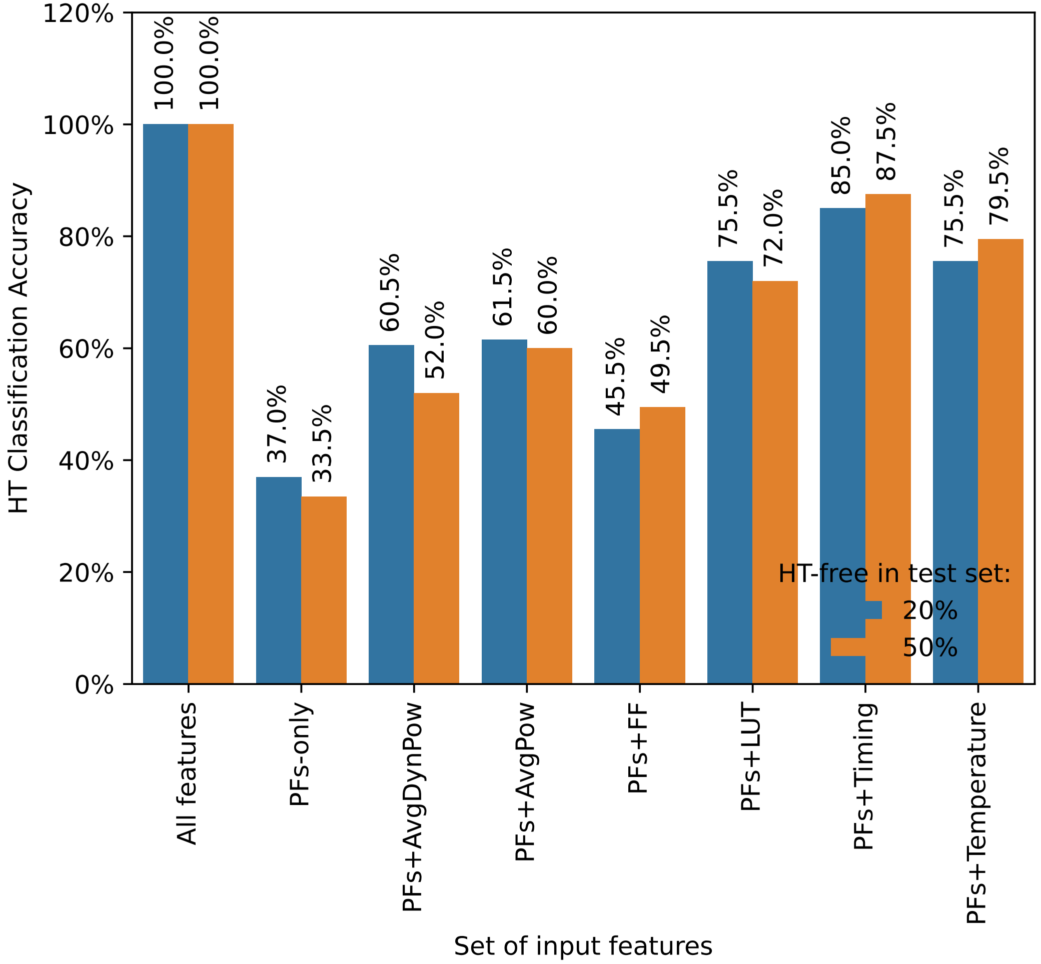
[8] *S. Ribes, et al. "Machine Learning-Based Classification of Hardware Trojans in FPGAs Implementing RISC-V Cores», International Conference on Information Systems Security and Privacy, 1: 717-724, 2024*

Methodology Countermeasure Idea

Tampering: FPGA Bitstream modifications



	Feature ID	Description
	Performance Features (PFs)	
High Level Features	Cycles	Number of clock cycles to execute the program
	InstrRet	Number of instructions retired in the program
	LSUs	Total waiting cycles to access data memory
	FetchWait	Total waiting cycles before instruction fetch
	Loads	Number of executed load instructions
	Stores	Number of executed store instructions
	Jumps	Number of executed jump instructions
	CondBran	Number of executed conditional branches
	ComprIns	Number of executed compressed instruction
	TakCBran	Number of taken conditional branches
	MulWait	Cycles for multiplication operation completion
	DivdWait	Cycles for division operation completion
	Benchmark	Program under execution (text label)
	Implementation Features (IFs)	
Low Level Features	LUTs	Final number of LUTs in the design
	FFs	Final number of FFs in the design
	AvgDynPow	Avg. dynamic power consumption [W]
	AvgTotPower	Avg. total power consumption [W]
	Timing	Worst negative slack (the circuit critical path) [ns]
	Temperature	Temperature trend

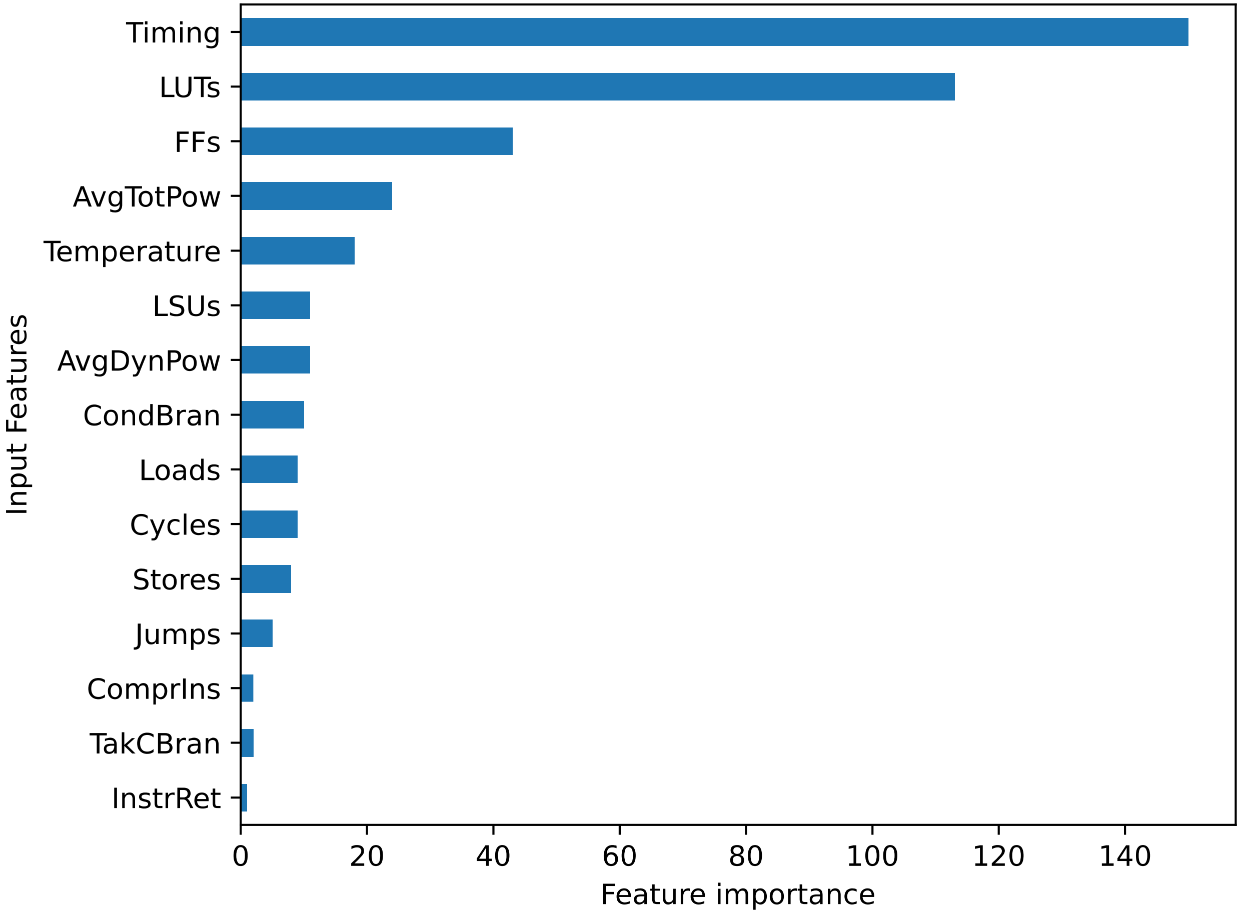


[8] S. Ribes, et al. "Machine Learning-Based Classification of Hardware Trojans in FPGAs Implementing RISC-V Cores», International Conference on Information Systems Security and Privacy, 1: 717-724, 2024

Methodology Countermeasure Idea

Tampering: FPGA Bitstream modifications

	Feature ID	Description
		Performance Features (PFs)
High Level Features	Cycles	Number of clock cycles to execute the program
	InstrRet	Number of instructions retired in the program
	LSUs	Total waiting cycles to access data memory
	FetchWait	Total waiting cycles before instruction fetch
	Loads	Number of executed load instructions
	Stores	Number of executed store instructions
	Jumps	Number of executed jump instructions
	CondBran	Number of executed conditional branches
	ComprIns	Number of executed compressed instruction
	TakCBran	Number of taken conditional branches
	MulWait	Cycles for multiplication operation completion
	DivdWait	Cycles for division operation completion
	Benchmark	Program under execution (text label)
		Implementation Features (IFs)
Low Level Features	LUTs	Final number of LUTs in the design
	FFs	Final number of FFs in the design
	AvgDynPow	Avg. dynamic power consumption [W]
	AvgTotPower	Avg. total power consumption [W]
	Timing	Worst negative slack (the circuit critical path) [ns]
	Temperature	Temperature trend



[8] S. Ribes, et al. "Machine Learning-Based Classification of Hardware Trojans in FPGAs Implementing RISC-V Cores», International Conference on Information Systems Security and Privacy, 1: 717-724, 2024

Microprocessors Vulnerability and Countermeasures

Challenges & Open Problems in the Hardware Security – Further readings

[1] A. Palumbo et al. “A lightweight security checking module to protect microprocessors against hardware trojan horses,” in 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1– 6, 2021.

[2] A. Bolat et al. “A microprocessor protection architecture against hardware trojans in memories,” in 2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS), pp. 1–6, 2020.

[3] A. Palumbo, et al. “Improving the detection of hardware trojan horses in microprocessors via hamming codes,” in 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1–6, 2023.

[4] A. Palumbo, et al. "Built-in Software Obfuscation for Protecting Microprocessors against Hardware Trojan Horses." 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). IEEE, 2023.

[5] K. Arıkan, A. Palumbo et al., "Processor Security: Detecting Microarchitectural Attacks via Count-Min Sketches," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 7, pp. 938-951, July 2022.

[6] M.Iamundo, "A machine learning-based security architecture to detect microarchitectural side-channel attacks in microprocessors ", Master Thesis, Politecnico di Milano (2021)

[7] A. Palumbo et al. "Is your FPGA bitstream Hardware Trojan-free? Machine learning can provide an answer", Journal of Systems Architecture, 128, 2022.

[8] S. Ribes, et al. "Machine Learning-Based Classification of Hardware Trojans in FPGAs Implementing RISC-V Cores», International Conference on Information Systems Security and Privacy, 1: 717-724, 2024

[9] L. Cassano et al. "Is RISC-V ready for Space? A Security Perspective", 2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)

[10] P. R. Nikiema et al. "Towards Dependable RISC-V Cores for Edge Computing Devices", 2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)

Hardware Trojan Horses and Microarchitectural Side-Channel Attacks: Detection and Mitigation via Hardware-based Methodologies

Q&A?

Alessandro Palumbo

Associate Professor at CentraleSupélec, Paris-Saclay University, Inria SUSHI Team, Rennes Campus

alessandro.palumbo@inria.fr

https://palessumbo.github.io/