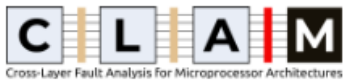
Cross-Layer Fault Analysis for Microprocessor Architectures
Séminaire SemSecuElec

Ihab ALSHAER

Université Grenoble Alpes, LCIS & TIMA labs, France

February 23rd, 2024
Rennes, France



Introduction

Introduction

- Embedded systems and IoT devices are everywhere.
- Security issues must be considered.

- Physical attacks are real threats.
- Fault injection attacks are major physical attacks.

Fault injection attack

- Active physical attack
- Introduce fault(s) to change normal behavior
- Possible vulnerability

- Main techniques:

Clock glitch

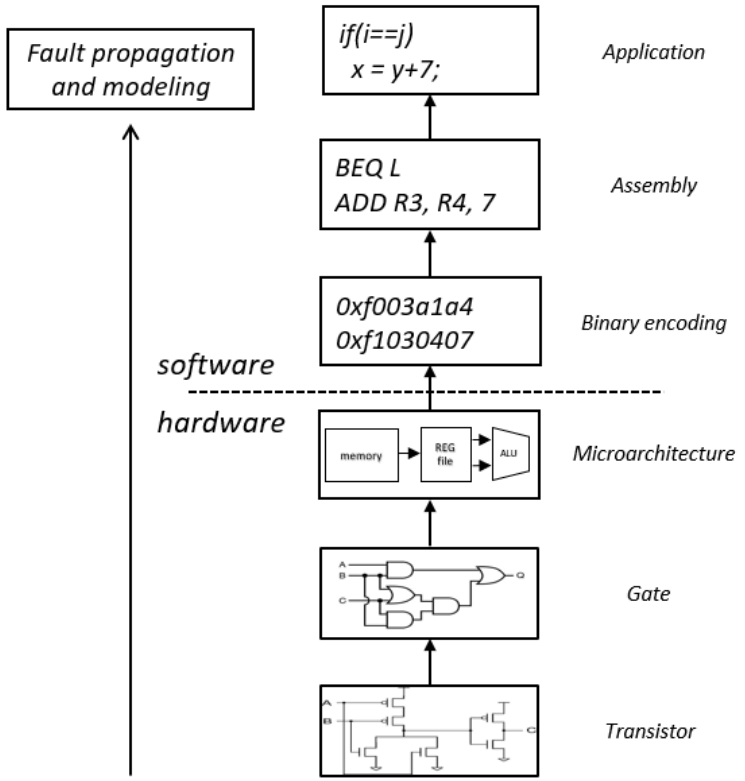
Voltage glitch

EM pulses

Laser

Problematic and objective

- Securing digital systems against fault attacks requires proper characterizations to build fault models.
- **Fault models** are necessary:
  - perform vulnerability analysis
  - design countermeasures
- Improper fault effect characterization:
  - inaccurate and incomplete fault models, thus non-optimal countermeasures:
    - **over-protections** (cost & performance)
    - **under-protections** (security)

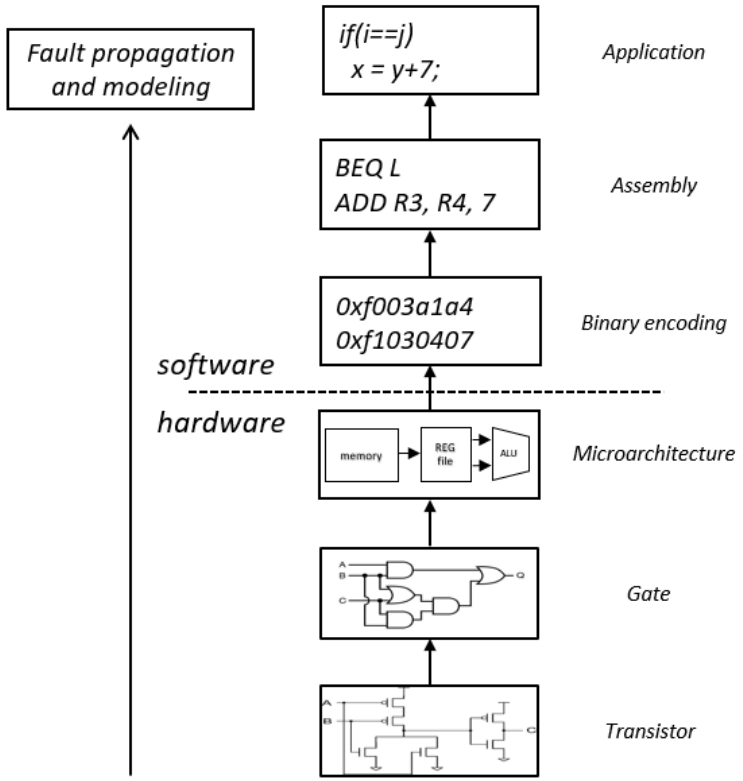


Problematic and objective

- Securing digital systems against fault attacks requires proper characterizations to build fault models.

- **Fault models** are necessary:
  - perform vulnerability analysis
  - design countermeasures

- Improper fault effect characterization:
  - inaccurate and incomplete fault models, thus non-optimal countermeasures:
    - **over-protections** (cost & performance)
    - **under-protections** (security)



- **Main objective**:
  - realistic and reliable fault models at software and hardware levels.
    - analyzing and understanding the effects of the fault injection over different levels of abstractions.

State-of-the-art

Fault effect characterization and modelling

- **Random** fault effect:
  - e.g., Spensky et al.[1] Khelil et al.[2], Tang et al.[3], Buhran et al.[4].
  - random bit/byte faults.

State-of-the-art

Fault effect characterization and modelling

- **Random** fault effect:
  - e.g., Spensky et al.[1] Khelil et al.[2], Tang et al.[3], Buhran et al.[4].
  - random bit/byte faults.

- Fault effect characterization at **ISA**:
  - e.g, Moro et al.[5], Trabelsi et al.[6], Werner et al.[7], Timmers et al.[8], Proy et al.[9], Trouchkine et al.[10], Khaut et al.[11].
  - Instruction(s) **skip** [7], [9], [11].
  - Instruction(s) **skip** and **replay** [9], [11].
  - Instruction **corruption/replacement** [5], [8], [10].
  - Register **corruption** [6], [8], [10].

State-of-the-art

Fault effect characterization and modelling

- **Random** fault effect:
  - e.g., Spensky et al.[1] Khelil et al.[2], Tang et al.[3], Buhran et al.[4].
  - random bit/byte faults.

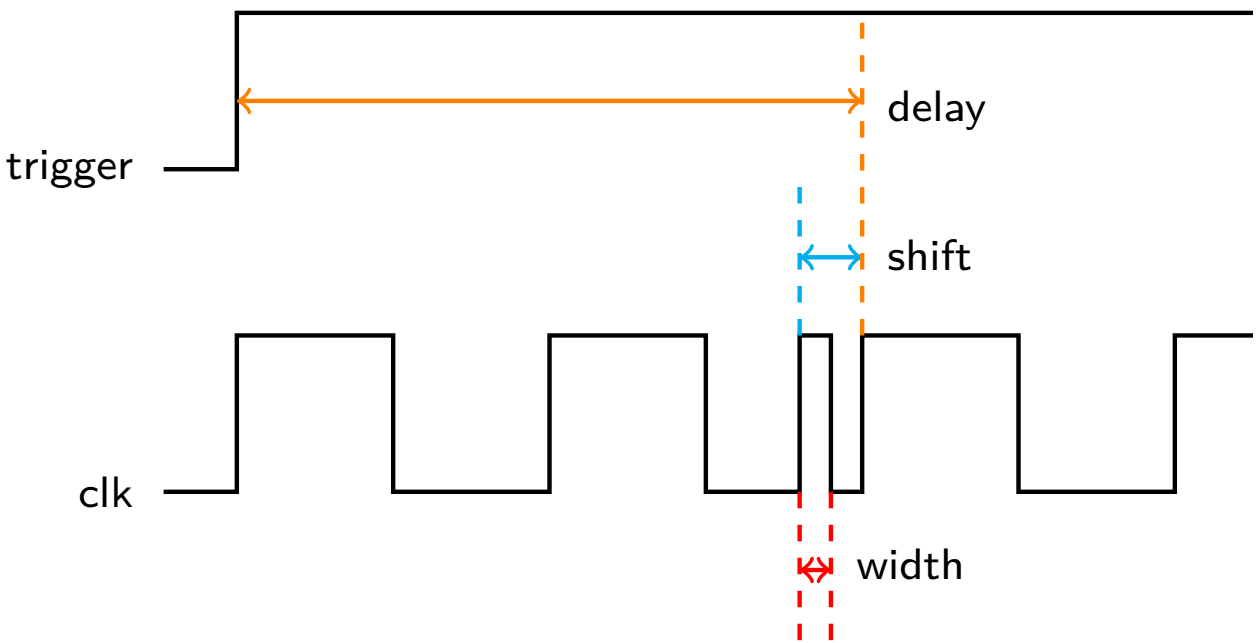
- Fault effect characterization at **ISA**:
  - e.g, Moro et al.[5], Trabelsi et al.[6], Werner et al.[7], Timmers et al.[8], Proy et al.[9], Trouchkine et al.[10], Khaut et al.[11].
  - Instruction(s) **skip** [7], [9], [11].
  - Instruction(s) **skip** and **replay** [9], [11].
  - Instruction **corruption/replacement** [5], [8], [10].
  - Register **corruption** [6], [8], [10].

- Fault effect characterization at **ISA** and **RTL** levels
  - e.g., Laurent et al.[12], Tollec et al.[13].
  - only **simulations**, RISC-V, bit(s) **flip/set/reset**.

Experimental evidence I

Experimental setup: Fault injection technique and Target devices

Clock glitch fault injection



- **Target devices**
  - STM32F0:
    - Cortex-M0
  - STM32F1:
    - Cortex-M3
  - STM32F3:
    - Cortex-M4
  - Widely used in embedded systems

- ChipWhisperer[14] environment is used to perform the clock glitch fault injection.

I. ALSHAER et al., *Cross-layer inference methodology for microarchitecture-aware fault models*, Microelectronics Reliability, volume 139, 2022

Experimental evidence II

Experimental setup: Target programs

```
1 CMP R4, R6
2 BNE labelx
3 ADD R2, R4, R6
4 labelx:
5 ADD R5, R4, R6
```

Control flow program.

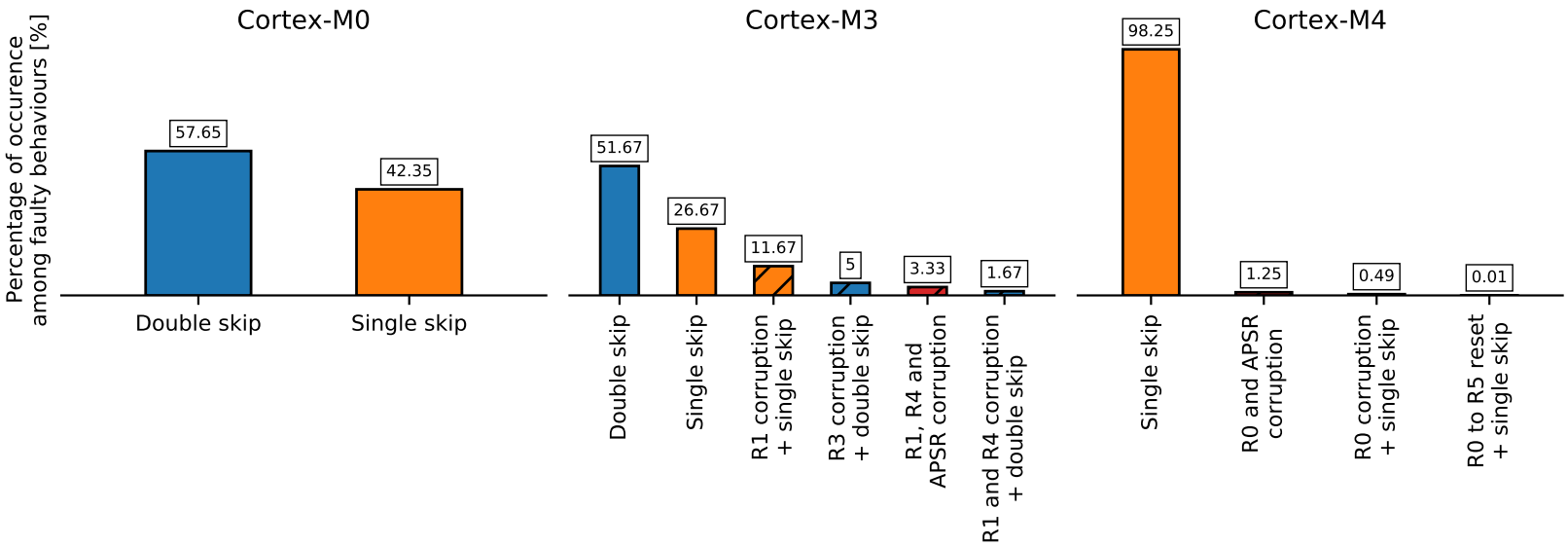
```
1 ADD R1, R1, 0x6
2 ADD R3, R3, 0xA
3 ADD R4, R4, 0xB
4 ADD R5, R6, R3
5 ADD R3, R3, 0xF
```

Data flow program.

I. ALSHAER et al., *Cross-layer inference methodology for microarchitecture-aware fault models*, Microelectronics Reliability, volume 139, 2022

Experimental evidence III: Target device dependency

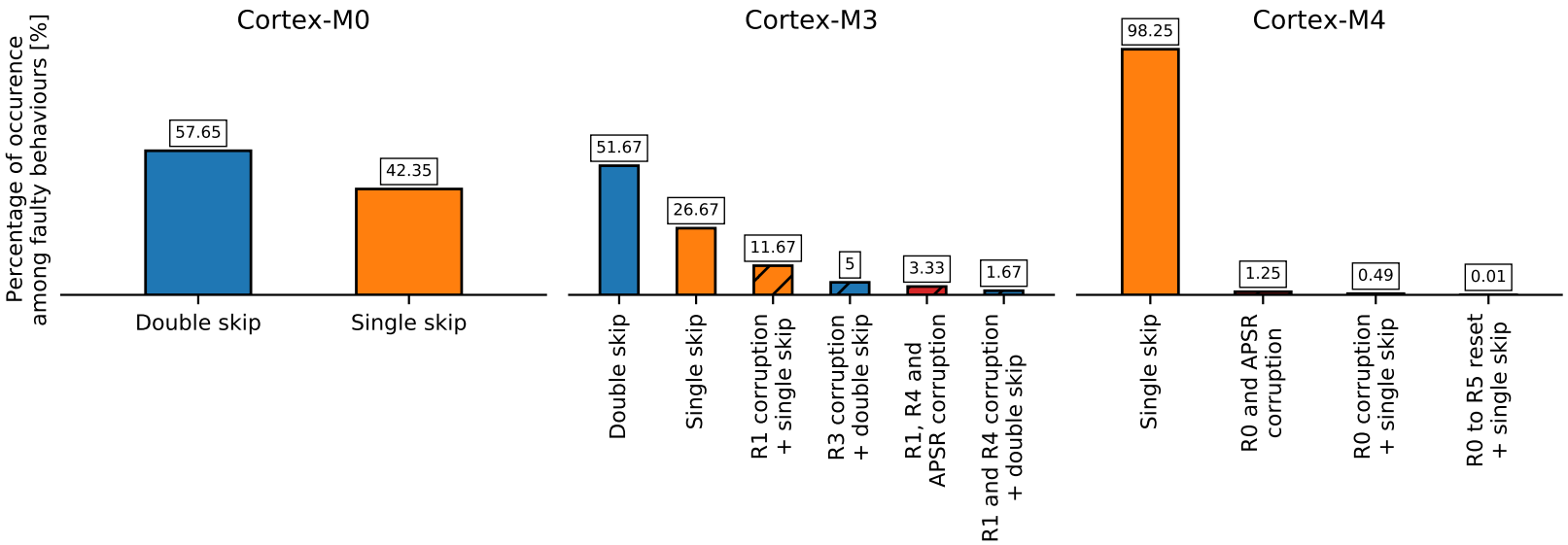
Control-flow target



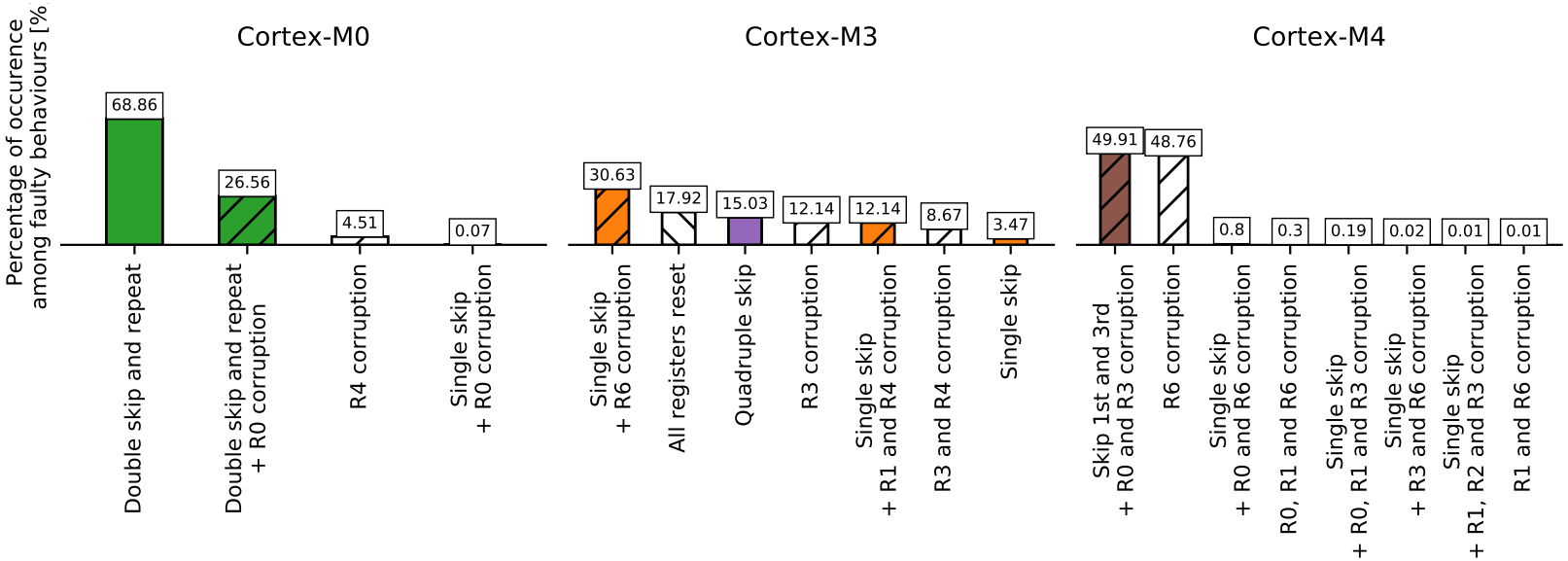
Data-flow target

Experimental evidence III: Target device dependency

Control-flow target

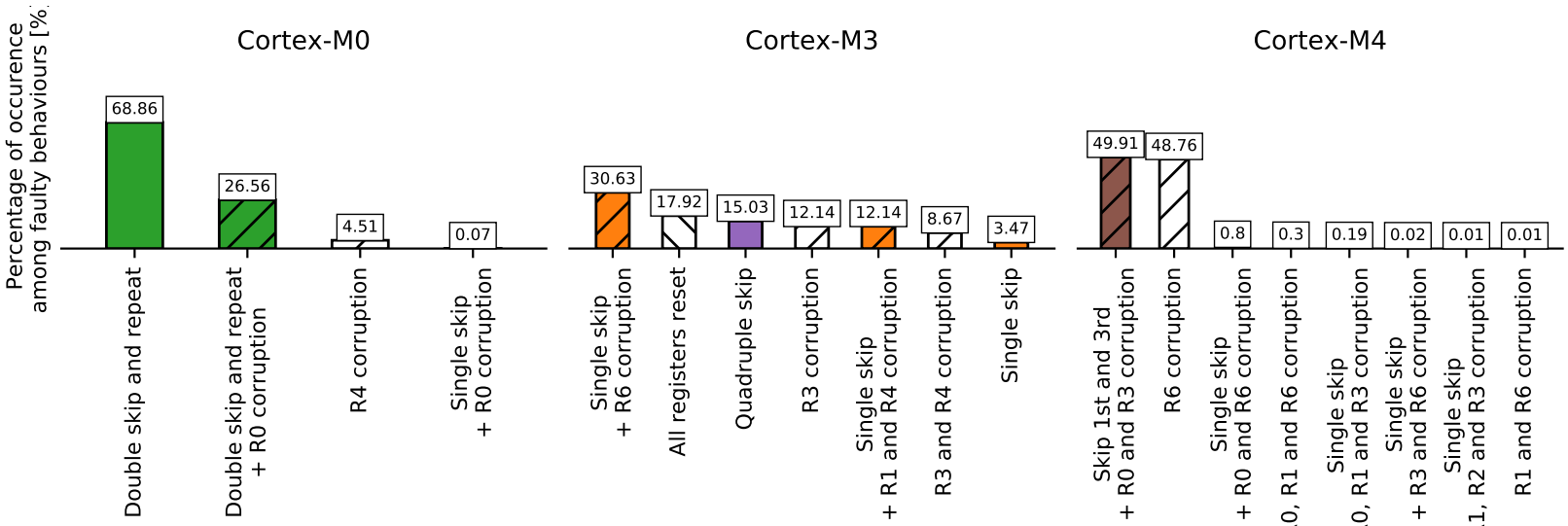


Data-flow target

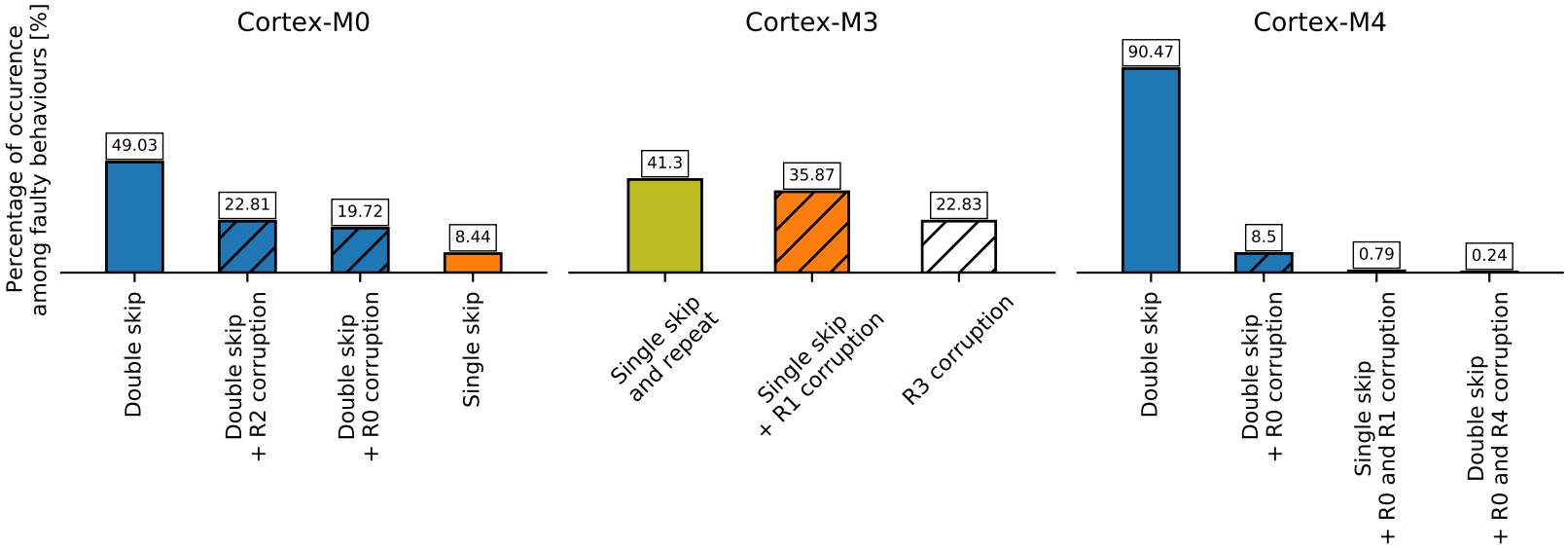


Experimental evidence IV: Target program dependency

Data-flow target



additional NOP



Experimental evidence V

Remarks and conclusions

The **difficulty** of fault effect characterization when considering only ISA

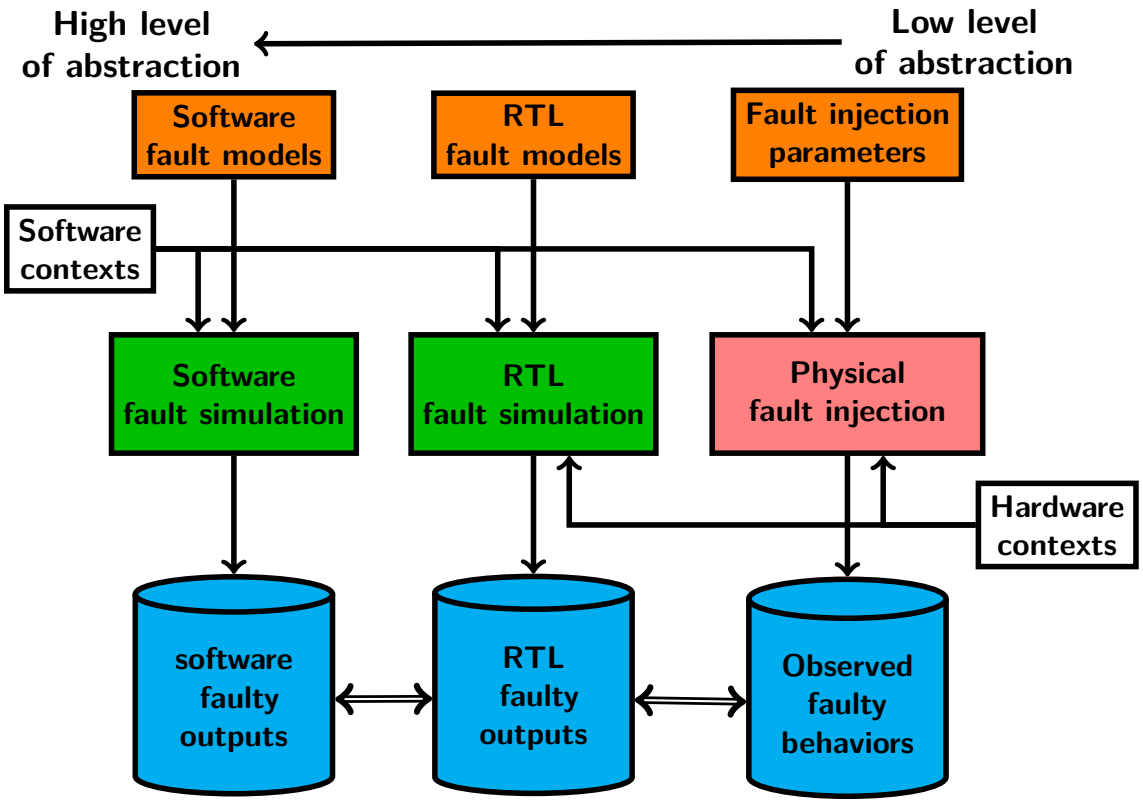
- Target **device** dependency
- Target **program** dependency
- **Small changes** in the target program have **large consequences** on the observed faults.
  - e.g., adding only a single NOP instruction
- **Complex** faulty behaviors are observed
  - **Combination** of **classical** fault models were needed to describe the effect of a single glitch.
    - e.g., R6 corruption + single instruction skip.
  - Some **registers** that are **not used** in the program end up being **corrupted**.
  - No **explanation** for the **corrupted** values

How can we address these issues?

I. ALSHAER et al., *Cross-layer inference methodology for microarchitecture-aware fault models*, Microelectronics Reliability, volume 139, 2022

Proposed methodology

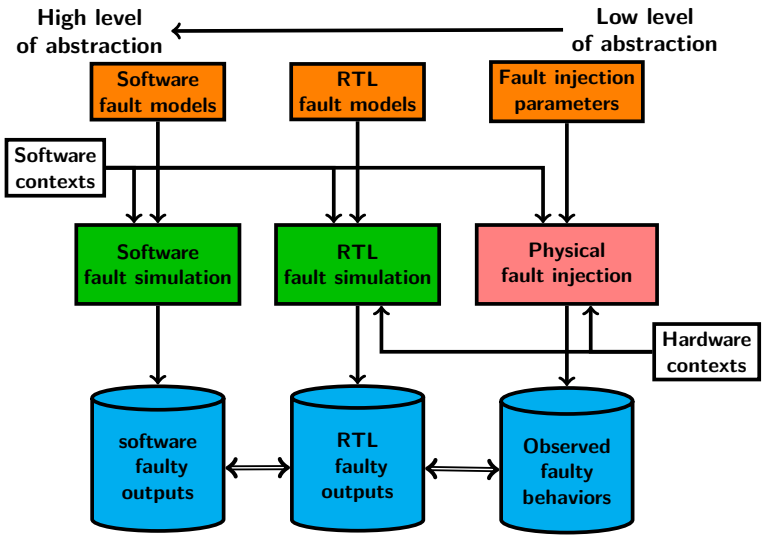
Cross-layer inference methodology I

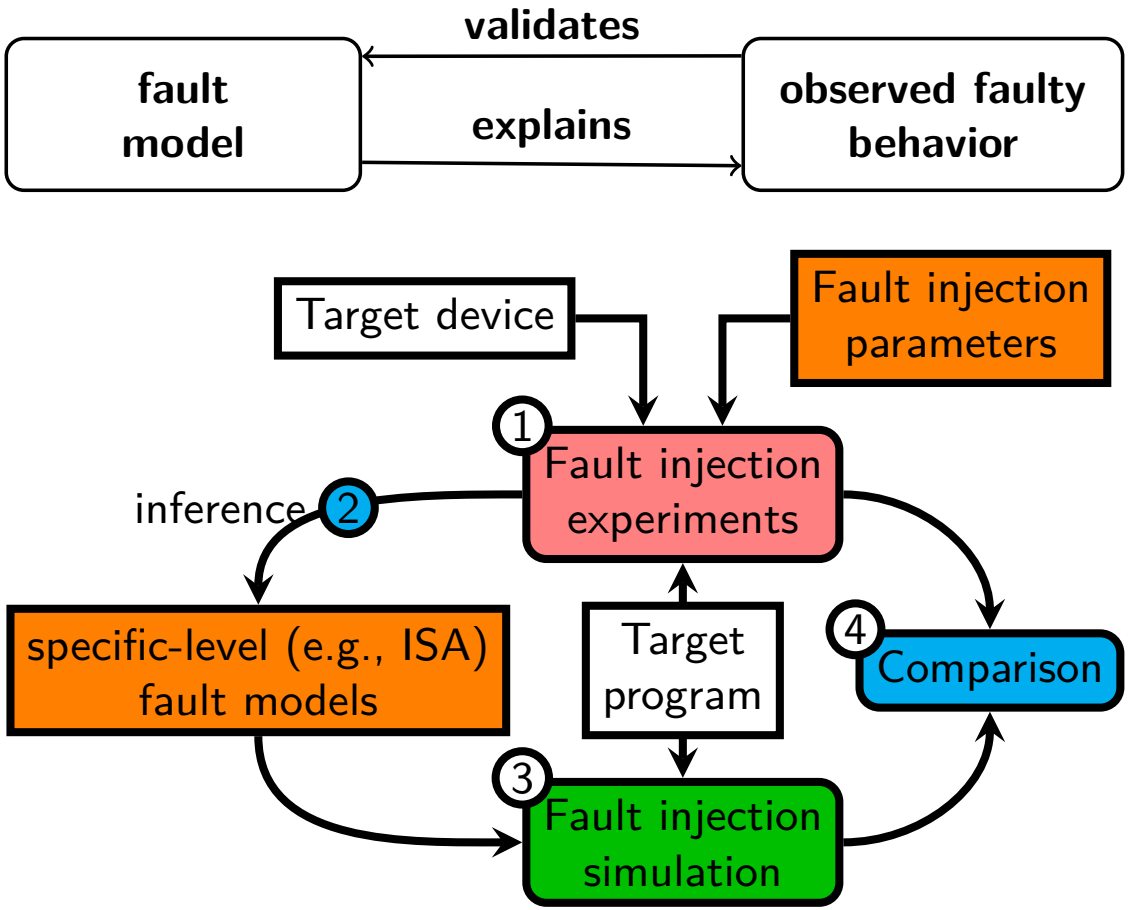


- Academic Access Agreement (AAA) with Arm

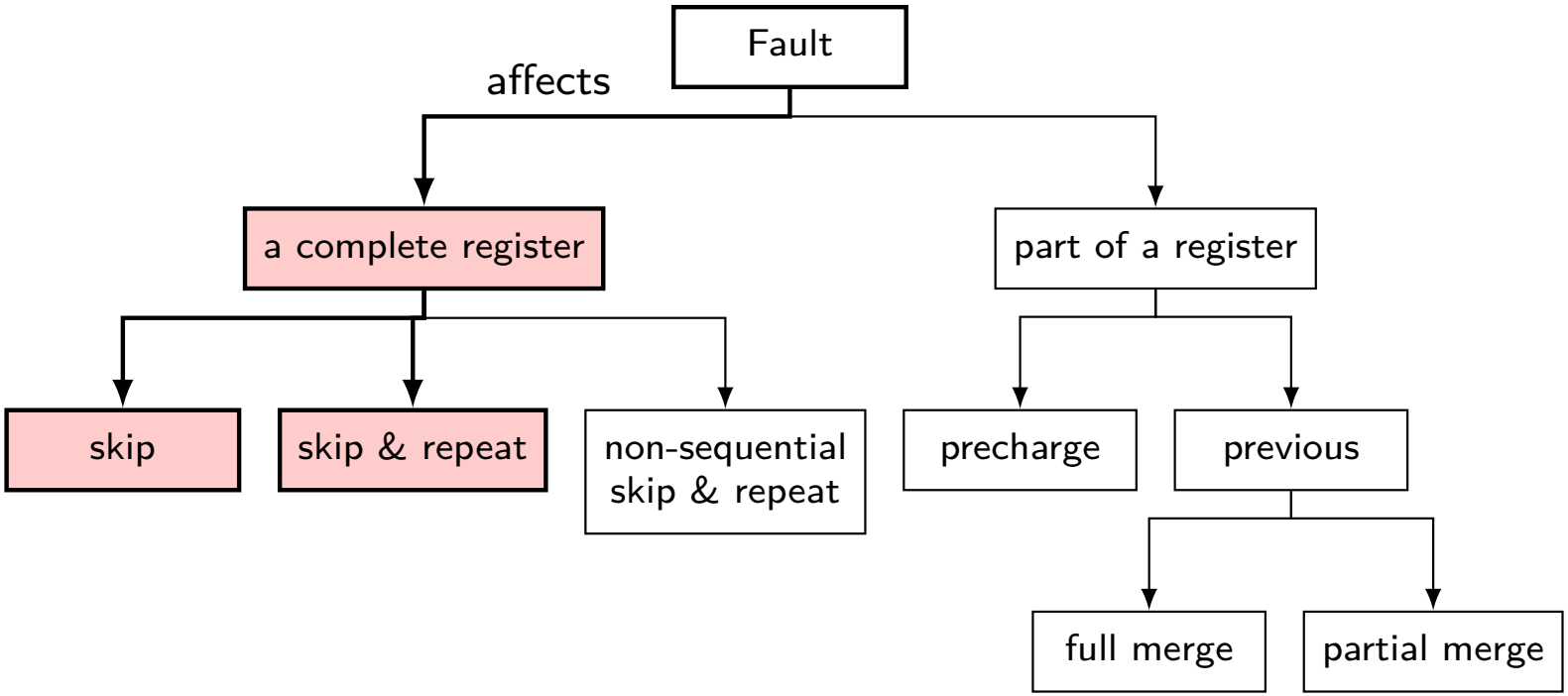
I. ALSHAER et al., *Cross-layer inference methodology for microarchitecture-aware fault models*, Microelectronics Reliability, volume 139, 2022

Cross-layer inference methodology II



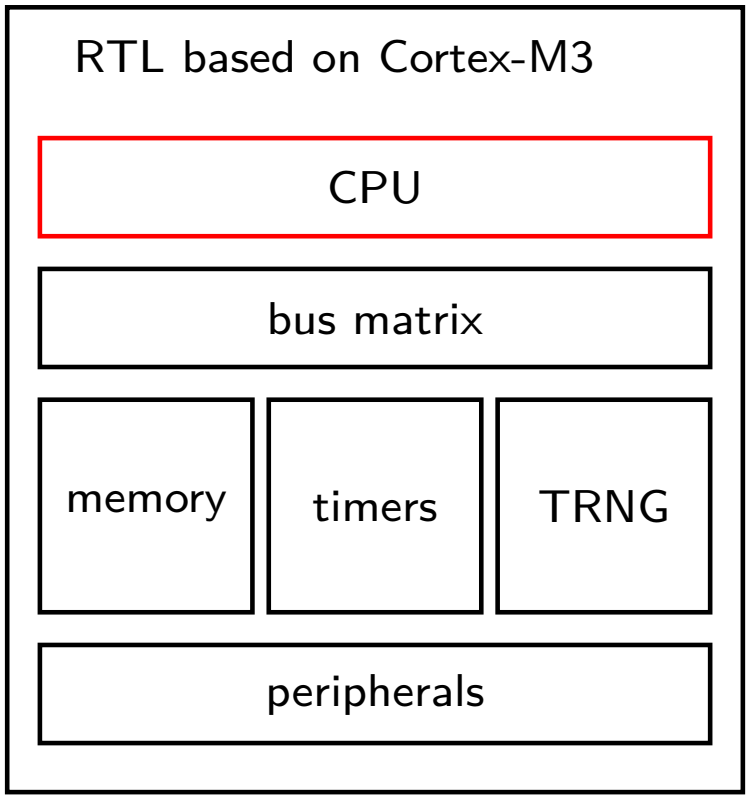


- Reliable and realistic fault models to be proposed

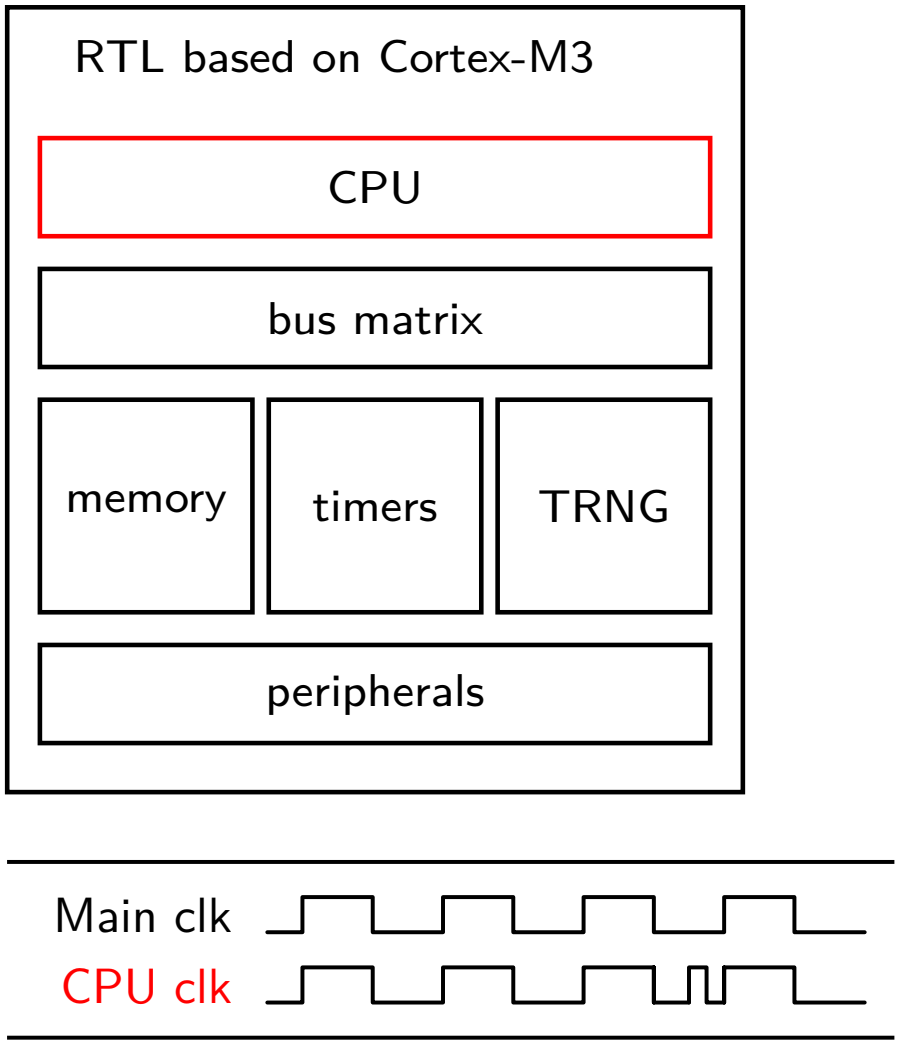

Fault
affects
a complete register
part of a register
skip
skip & repeat
non-sequential
skip & repeat
precharge
previous
full merge
partial merge

Preliminary HW simulation and binary encoding fault models

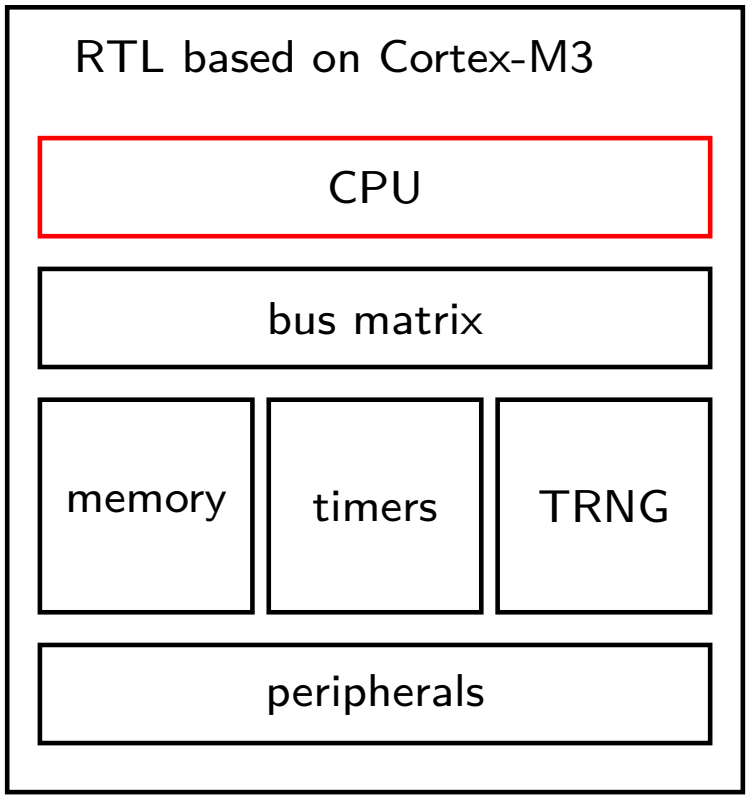
Preliminary RTL fault simulation

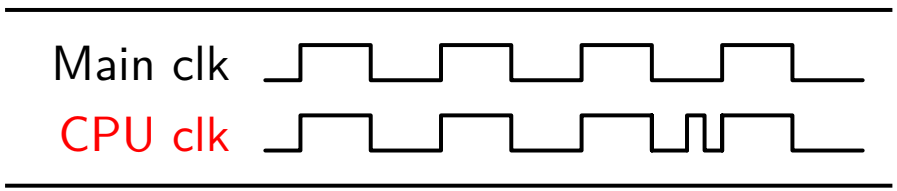


Preliminary RTL fault simulation



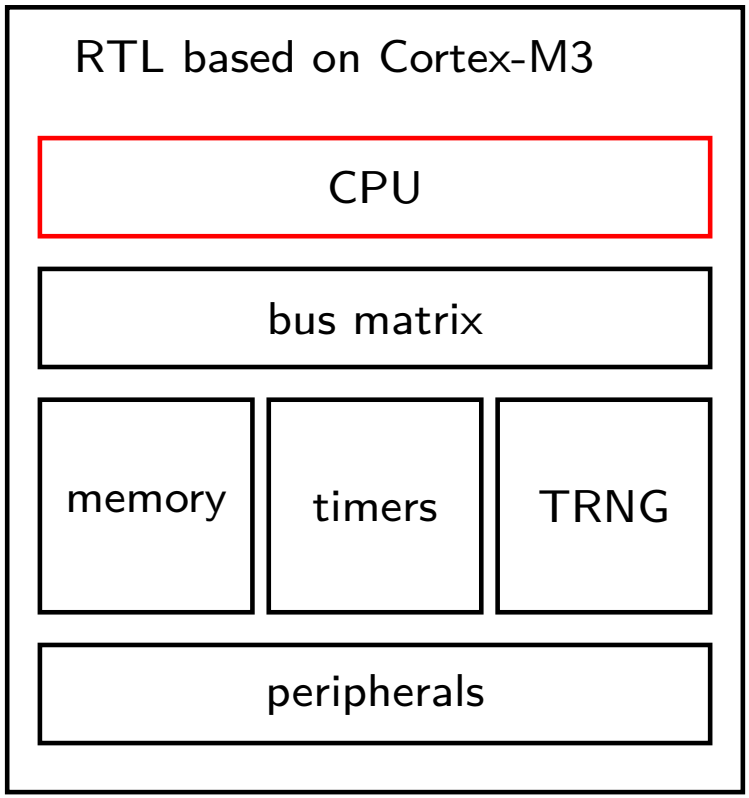
Preliminary RTL fault simulation

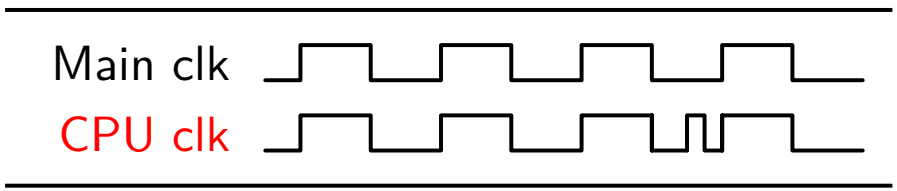




- Skip an instruction and repeat the previous one

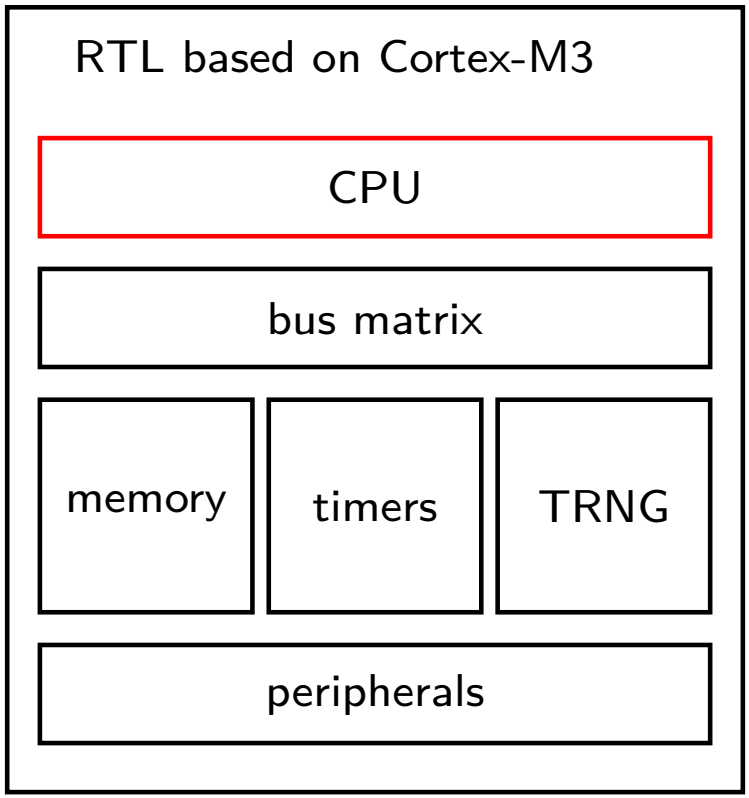
Preliminary RTL fault simulation



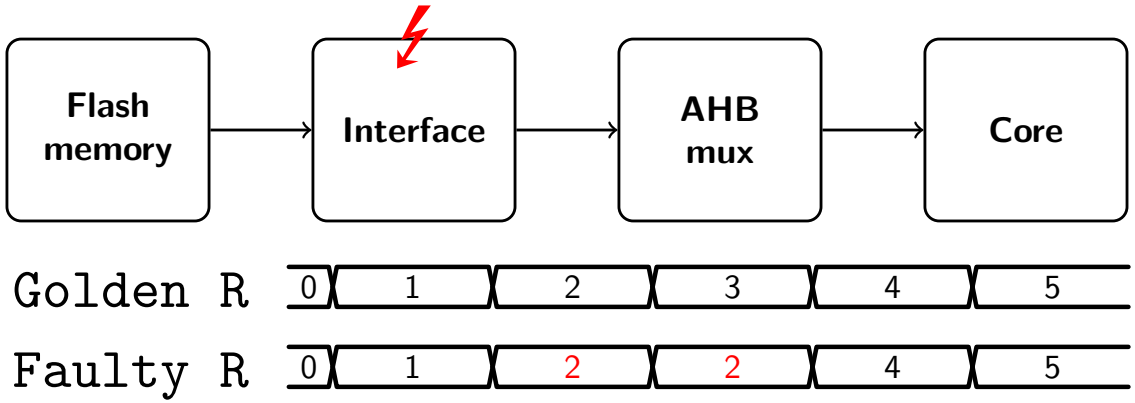


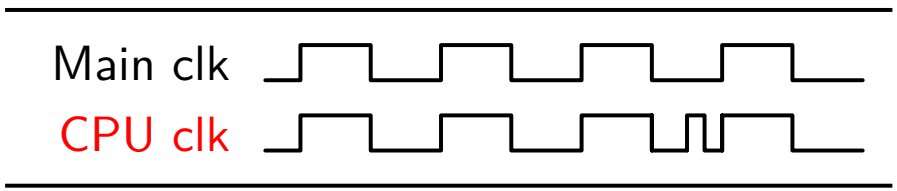
- Skip an instruction and repeat the previous one
- Adding a single NOP to the target program and repeating the simulation led to observe:
  - Double instruction corruption

Preliminary RTL fault simulation

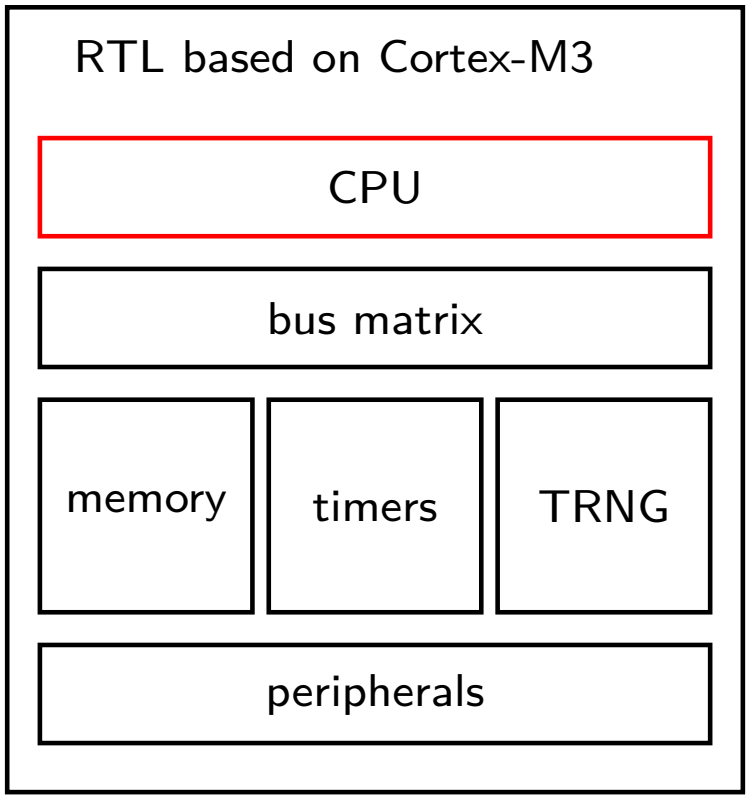


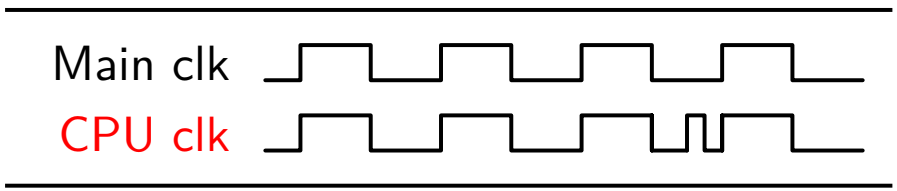
- Skip an instruction and repeat the previous one
- Adding a single NOP to the target program and repeating the simulation led to observe:
  - Double instruction corruption



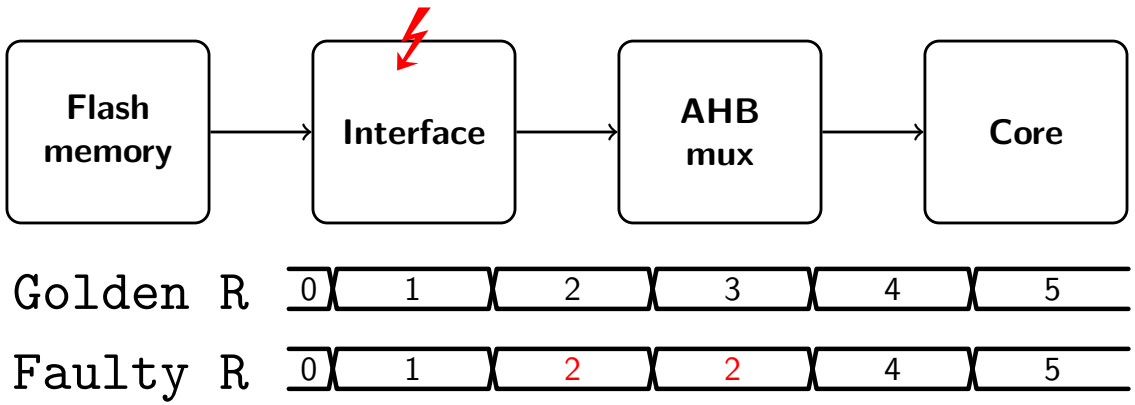


Preliminary RTL fault simulation





- Skip an instruction and repeat the previous one
- Adding a single NOP to the target program and repeating the simulation led to observe:
  - Double instruction corruption



- Same fault at HW => different faulty behaviors at SW:
  - Alignment in memory was different
    - Variable-length instruction set

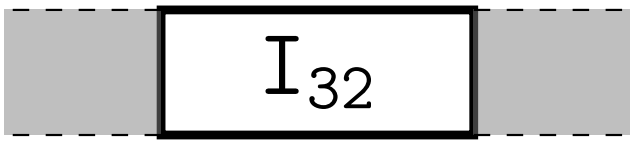
Variable-length instruction set

- Has not the same length of encoding for all the instructions.
- Main advantage: reducing the code size to achieve higher code density.
- Example: 16- and 32-bit encoding
  - microMIPS
  - RISC-V compressed (RVC)
  - Arm **Thumb2**

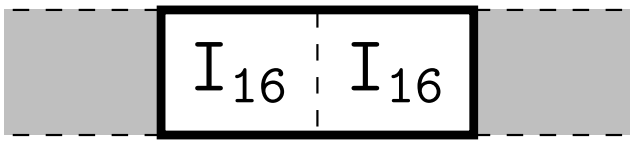
I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Fetch cases of 32 bits for 16- and 32-bit encoding

Fetching aligned instructions



(a)

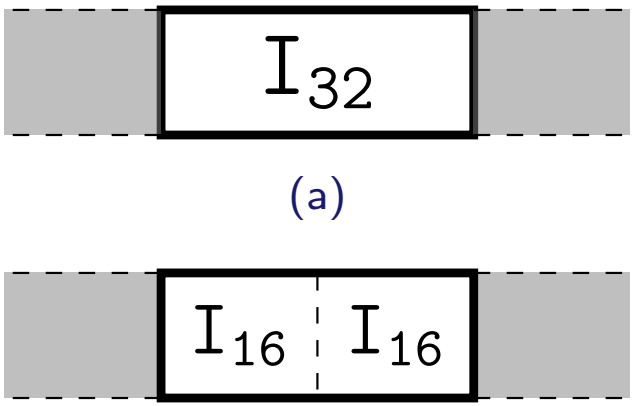


(b)

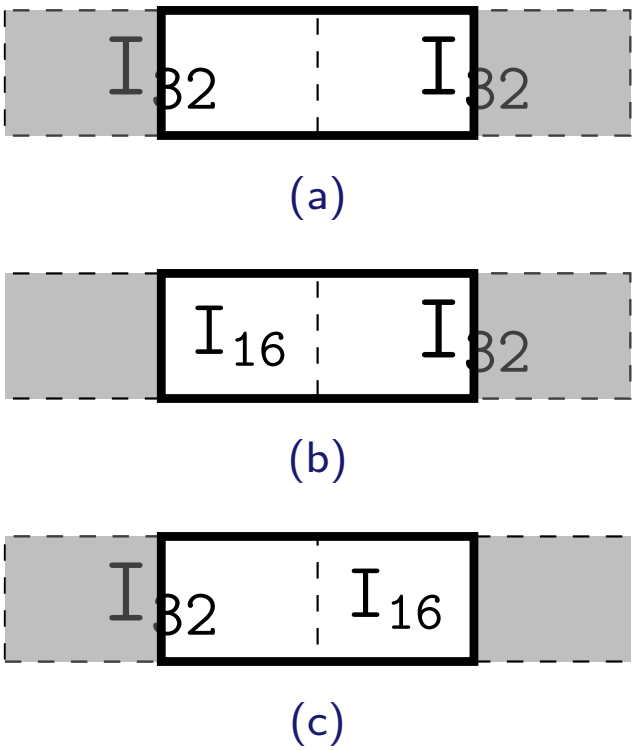
I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Fetch cases of 32 bits for 16- and 32-bit encoding

Fetching aligned instructions



Fetching misaligned instructions



I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Inferred binary encoding fault models

- Skip a specific number of bits
- Skip and repeat a specific number of bits
- This specific number of bits is related to:
  - flash memory access size,
  - cache line size,
  - instruction buffer size, or
  - internal register size.
  - e.g., 32 bits, and 64 bits.

I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Aligned code scenario I

```
MOV R8, R4
LSLS R2, R0, 0x10
ADD R1, R1, 0x6
ADD R3, R3, 0xa
ADD R4, R4, 0xb
ADD R5, R6, R3
ADD R3, R3, 0xf
```

Aligned code target program.

```
1 46a00402
2 f1010106
3 f103030a
4 f104040b
5 eb060503
6 f103030f
```

Binary encoding of the aligned code in hex. format.

- Skip 32 bits: the 32 bits at line *i* are skipped, and the execution resumes from line *i+1*.
- Skip & repeat 32 bits: the 32 bits at line *i+1* are skipped and the 32 bits at line *i* are repeated.

I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Aligned code scenario II

Skip 32 bits

```
1 46a00402
2 f1010106
3 f103030a
4 f104040b
5 eb060503
6 f103030f
```

Original encoding.

I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Aligned code scenario II

Skip 32 bits

1	46a00402
2	f1010106
3	f103030a
4	f104040b
5	eb060503
6	f103030f

Original encoding.

1	~~46a00402~~
2	f1010106
3	f103030a
4	f104040b
5	eb060503
6	f103030f

Skip line 1.

I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Aligned code scenario II

Skip 32 bits

```
1 46a00402
2 f1010106
3 f103030a
4 f104040b
5 eb060503
6 f103030f
```

Original encoding.

```
1 ~~46a00402~~
2 f1010106
3 f103030a
4 f104040b
5 eb060503
6 f103030f
```

Skip line 1.

```
1 ADD R1, R1, 0x6
2 ADD R3, R3, 0xa
3 ADD R4, R4, 0xb
4 ADD R5, R6, R3
5 ADD R3, R3, 0xf
```

Double instruction skip.

```
1 MOV R8, R4
2 LSLS R2, R0, 0x10
3 ADD R1, R1, 0x6
4 ADD R3, R3, 0xa
5 ADD R4, R4, 0xb
6 ADD R5, R6, R3
7 ADD R3, R3, 0xf
```

Original code.

I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Misaligned code scenario I

```
~~MOV R0, R4~~
LSLS R2, R0, 0x10
ADD R1, R1, 0x6
ADD R3, R3, 0xa
ADD R4, R4, 0xb
ADD R5, R6, R3
ADD R3, R3, 0xf
```

Misaligned code target program.

```
1 0402f101
2 0106f103
3 030af104
4 040beb06
5 0503f103
6 030fbf00 // bf00: NOP.
```

Binary encoding of the misaligned code in hex. format.

- Skip 32 bits: the 32 bits at line *i* are skipped, and the execution resumes from line *i+1*.
- Skip & repeat 32 bits: the 32 bits at line *i+1* are skipped and the 32 bits at line *i* are repeated.

I. ALSHAER et al.,*Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Misaligned code scenario II

Skip 32 bits

```
1 0402f101
2 0106f103
3 030af104
4 040beb06
5 0503f103
6 030fbf00
```

Original encoding.

I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Misaligned code scenario II

Skip 32 bits

```
1 0402f101
2 0106f103
3 030af104
4 040beb06
5 0503f103
6 030fbf00
```

Original encoding.

```
1 0402f101
2 0106f103
3 030af104
4 040beb06
5 0503f103
6 030fbf00
```

Skip line 1.

I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Misaligned code scenario II

Skip 32 bits

```
1 0402f101
2 0106f103
3 030af104
4 040beb06
5 0503f103
6 030fbf00
```

Original encoding.

```
1 0402f101
2 0106 f103
3 030af104
4 040beb06
5 0503f103
6 030fbf00
```

Skip line 1.

```
1 LSLS R6, R0, 0x4
2 ADD R3, R3, 0xa
3 ADD R4, R4, 0xb
4 ADD R5, R6, R3
5 ADD R3, R3, 0xf
```

Double instruction skip and new instruction execution.

```
1 LSLS R2, R0, 0x10
2 ADD R1, R1, 0x6
3 ADD R3, R3, 0xa
4 ADD R4, R4, 0xb
5 ADD R5, R6, R3
6 ADD R3, R3, 0xf
```

Original code.

I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Misaligned code scenario II

Skip 32 bits

```
1 0402f101
2 0106f103
3 030af104
4 040beb06
5 0503f103
6 030fbf00
```

Original encoding.

```
1 ~~0402f101~~
2 0106 f103
3 030af104
4 040beb06
5 0503f103
6 030fbf00
```

Skip line 1.

```
1 LSLS R6, R0, 0x4
2 ADD R3, R3, 0xa
3 ADD R4, R4, 0xb
4 ADD R5, R6, R3
5 ADD R3, R3, 0xf
```

Double instruction skip and new instruction execution.

```
1 LSLS R2, R0, 0x10
2 ADD R1, R1, 0x6
3 ADD R3, R3, 0xa
4 ADD R4, R4, 0xb
5 ADD R5, R6, R3
6 ADD R3, R3, 0xf
```

Original code.

- Changing the destination register (R1) and/or the immediate value (0x6) allows observing the execution of other new instructions.

I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Misaligned code scenario III

More on the ability of executing a new instruction

- Effect of last observed behavior with different destination register and/or immediate value.

Original instruction	Least-significant 16 bits	New instruction
`ADD R4, R1, 0x9`	`0x0409`	`LSLS R1, R1, 0x10`
`ADD R0, R1, 0x46c`	`0x406c`	`EORS R4, R5`
`ADD R12, R1, 0x60c`	`0x6c0c`	`LDR R4, [R1, 0x40]`
`ADD R0, R1, 0x161`	`0x1061`	`ASRS R1, R4, 0x1`
`ADD R0, R1, 0x205`	`0x2005`	`MOVS R0, 0x5`
`ADD R3, R1, 0x416`	`0x4316`	`ORRS R6, R2`

- The **S** in 16-bit instructions updates the `APSR flags` => `APSR` corruption

I. ALSHAER et al.,*Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Exploit the ability of executing a new instruction

- The security property is to not modify the program counter to a specific address stored in **R8**.
- This security property is violated if we manage to execute the **MOV PC, R8** instruction (**0x46c7**).

```
1  //R8 = address of line 10
2  //series of NOPs
3  LSLS R2, R0, 0x10
4  //any instruction from the Table
5  ADD R3, R3, 0xa
6  ADD R4, R4, 0xb
7  ADD R5, R6, R3
8  ADD R3, R3, 0xf
9  //series of NOPs
10 LDR R1, [R1,0xf00]
11 MOV R9, R6
```

Target program for exploitation example.

Original instruction	Least-significant 16 bits
`ADD R6, R1, 0x4c7`	0x46c7
`SUB R6, R1, 0x4c7`	0x46c7
`MOVW R6, 0x4c7`	0x46c7
`LDR R4, [r0, 0x6c7]`	0x46c7
`ORR R6, R6, 0x63800000`	0x46c7

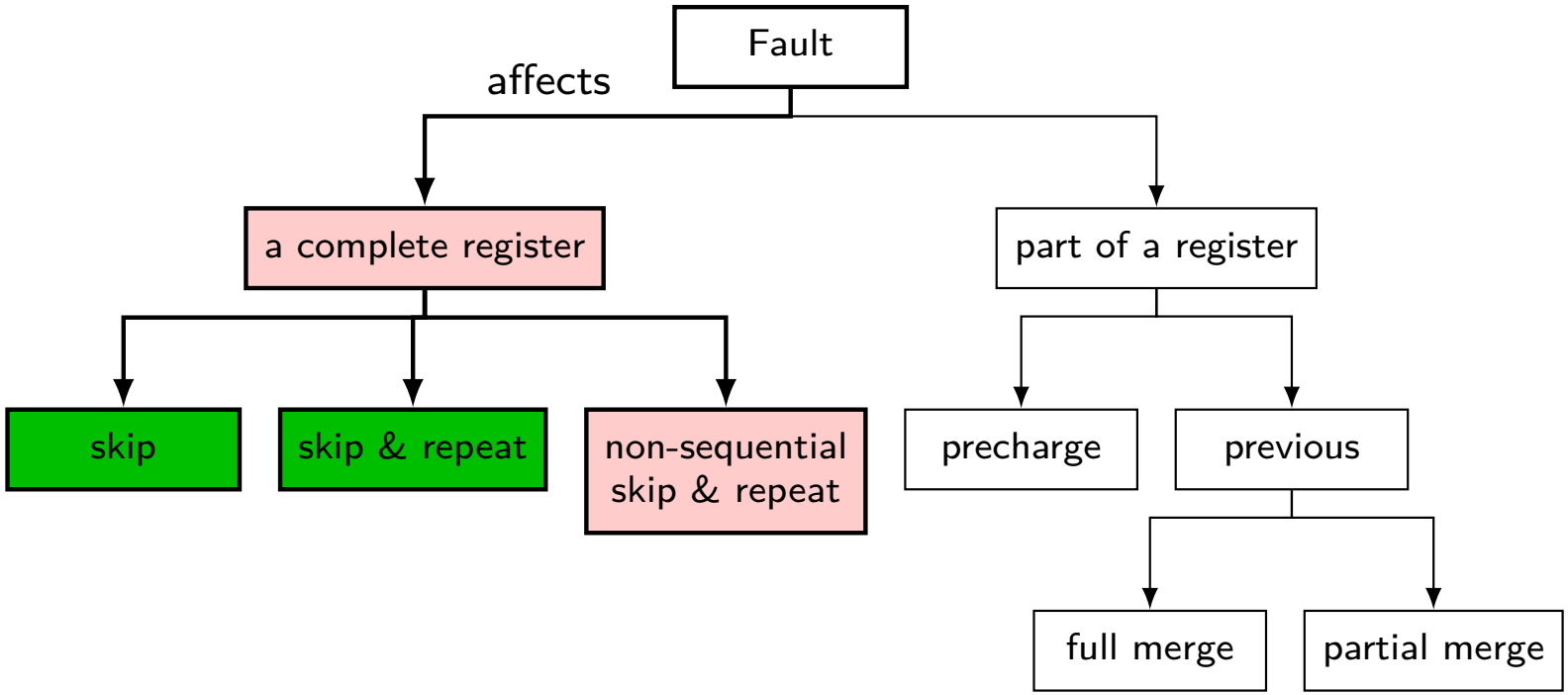
Table: Instructions that lead to modify the PC to the value in R8 when performing clock glitch fault injection.

I. ALSHAER et al., *Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Vulnerability analysis on 3 implementations of AES

- Software fault simulation that considers only the 32-bit faults of misaligned codes:
  - *BroAES:* retrieve the key
  - *TinyAES128*: retrieve the key
  - *MbedTLS-AES*: simplify performing DFA
- Physical clock glitch on BroAES validated the analysis on three different MCUs (STM32F1, STM32F3 & STM32L4).

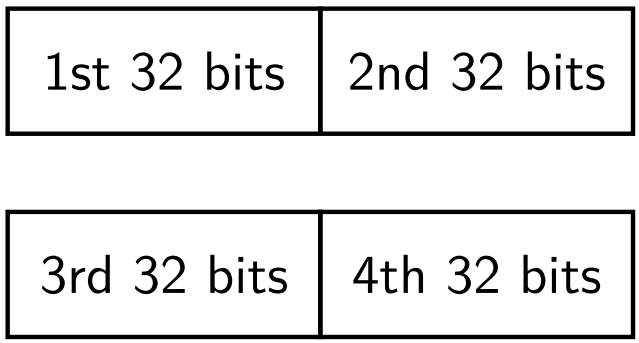
I. ALSHAER et al., submitted to JCEN


Fault
affects
a complete register
part of a register
skip
skip & repeat
non-sequential
skip & repeat
precharge
previous
full merge
partial merge

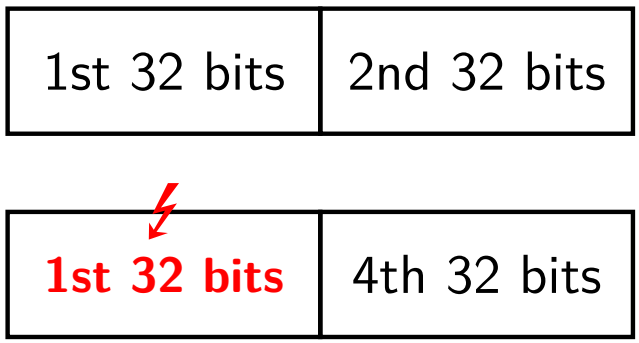
Additional faulty behaviors

Another fault model at Binary encoding level

- Non-sequential skip and repeat 32 bits
  - the skipped 32 bits are not the next bits with respect to the repeated bits.
- Example of observed faulty behaviors in STM32L4 (it has 64-bit cache lines):



(a) Golden execution.



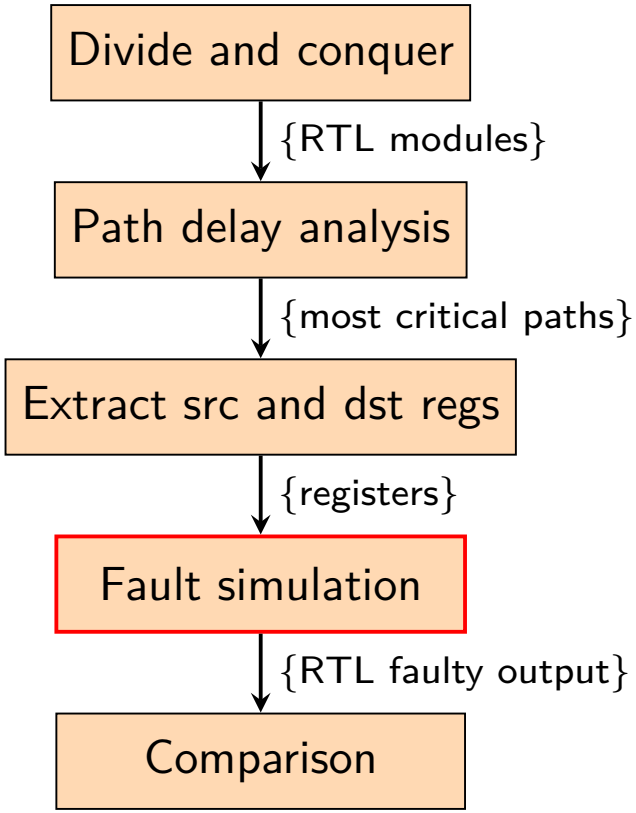
(b) Faulty execution.

- Observed for both aligned and misaligned codes

HW fault simulation and Partial update fault model

RTL fault simulation

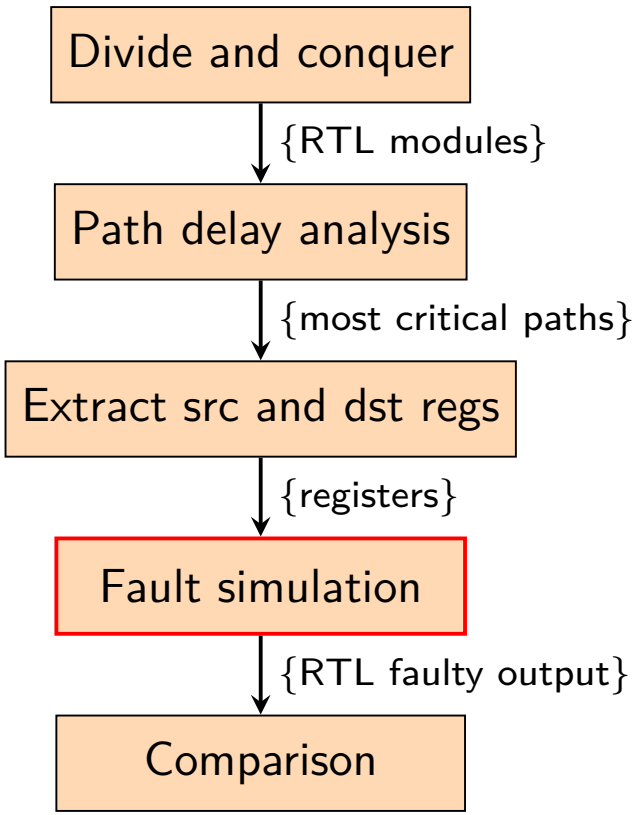
- Approach To facilitate and accelerate the fault simulation and the analysis:



I. ALSHAER et al., submitted to JCEN

RTL fault simulation

- Approach To facilitate and accelerate the fault simulation and the analysis:

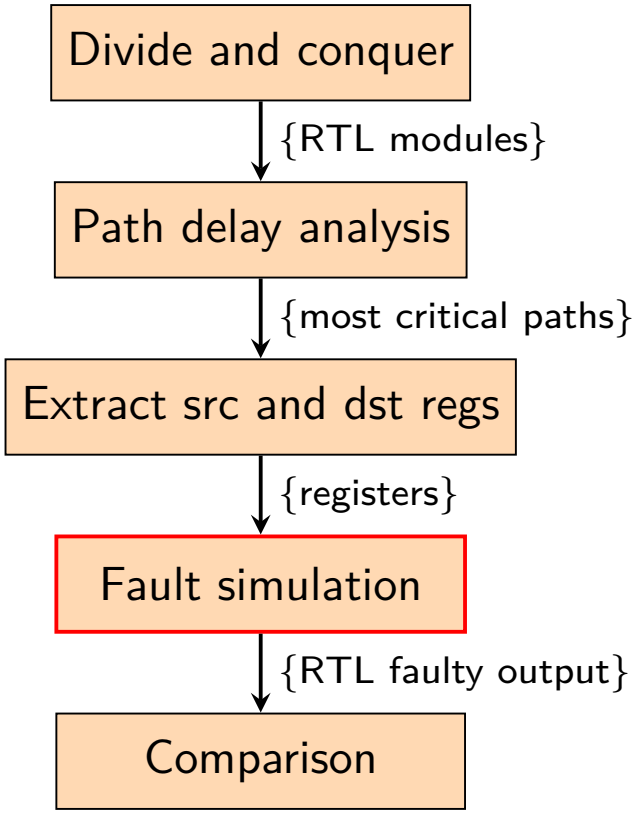


- New inferred RTL fault models are used to perform the fault simulation:
  - *prevent the update*: the register keeps its previous value at a given clock cycle.
  - *anticipate the update*: at clock cycle i, the value that the register would actually store at clock cycle $i+1$ is loaded.

I. ALSHAER et al., submitted to JCEN

RTL fault simulation

- Approach To facilitate and accelerate the fault simulation and the analysis:

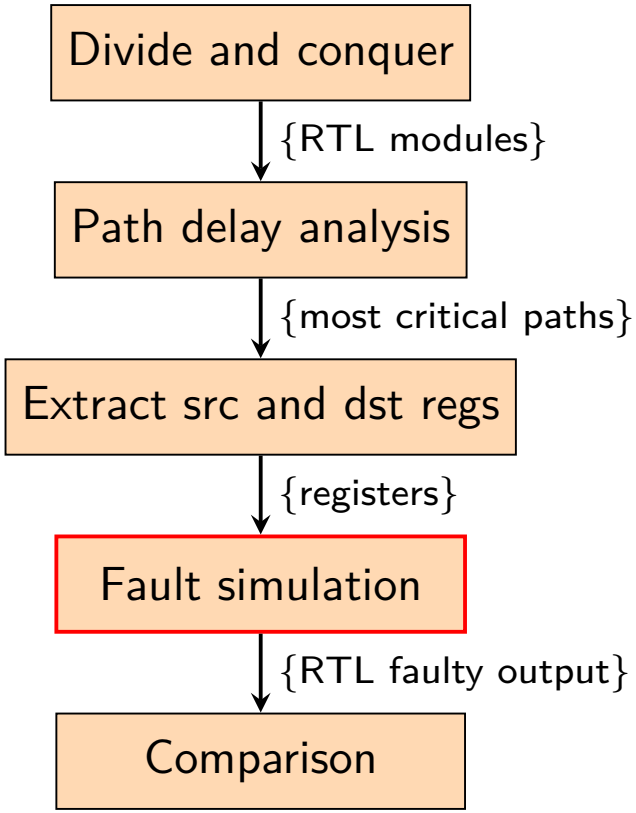


- New inferred RTL fault models are used to perform the fault simulation:
  - *prevent the update*: the register keeps its previous value at a given clock cycle.
  - *anticipate the update*: at clock cycle i, the value that the register would actually store at clock cycle $i+1$ is loaded.
- Identical results to Skip and Skip & repeat are obtained.

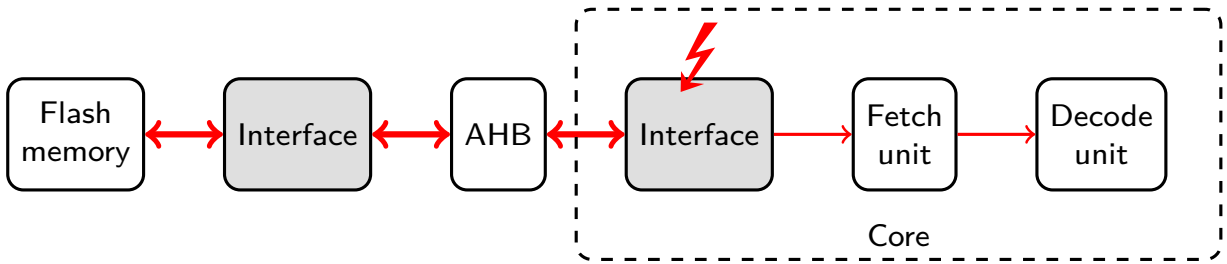
I. ALSHAER et al., submitted to JCEN

RTL fault simulation

- Approach To facilitate and accelerate the fault simulation and the analysis:

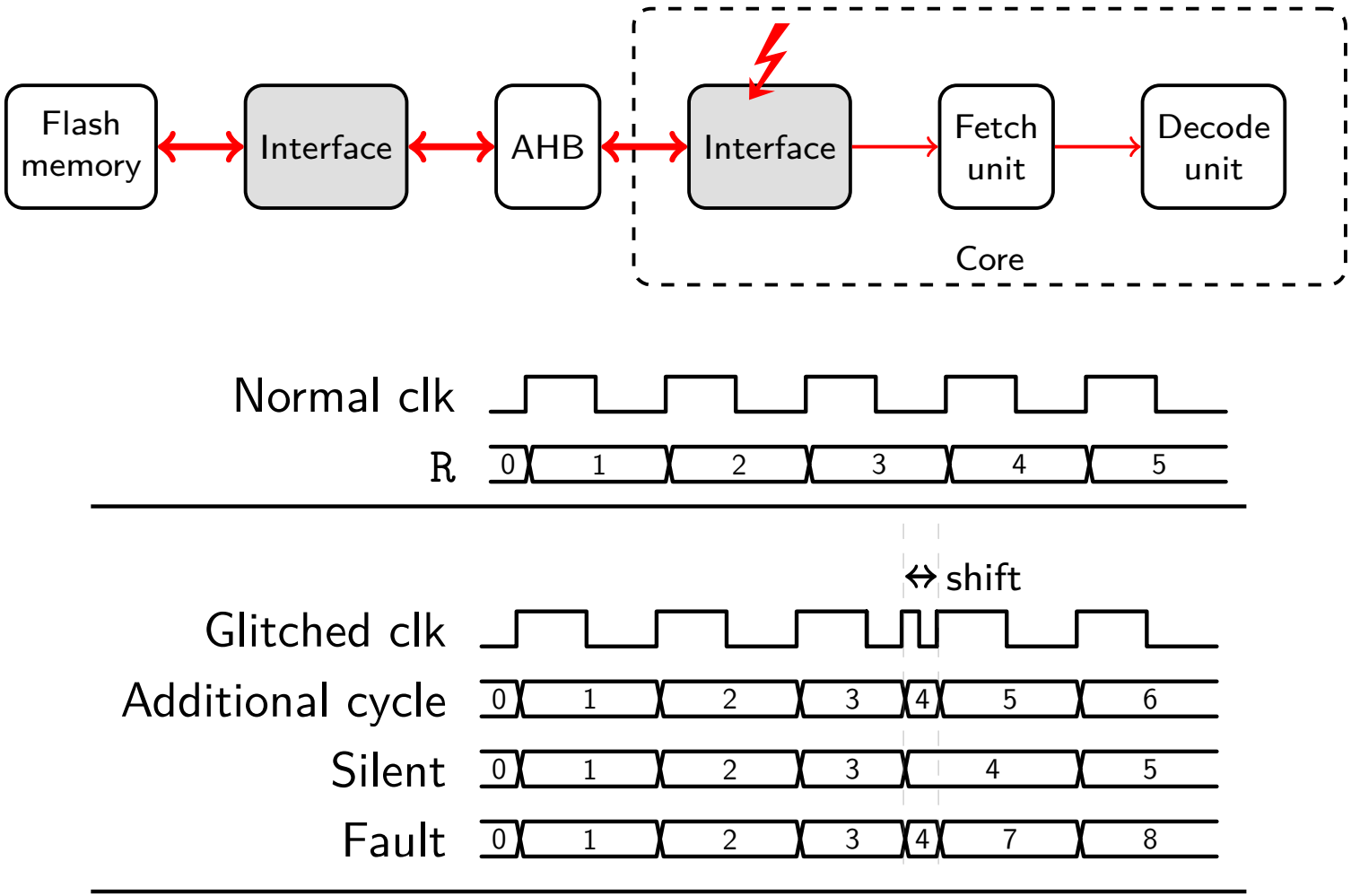


- New inferred RTL fault models are used to perform the fault simulation:
  - *prevent the update*: the register keeps its previous value at a given clock cycle.
  - *anticipate the update*: at clock cycle i, the value that the register would actually store at clock cycle $i+1$ is loaded.
- Identical results to Skip and Skip & repeat are obtained.



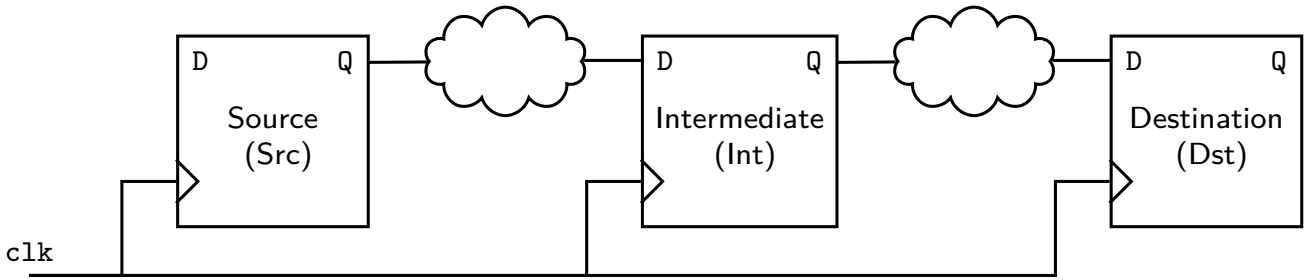
I. ALSHAER et al., submitted to JCEN

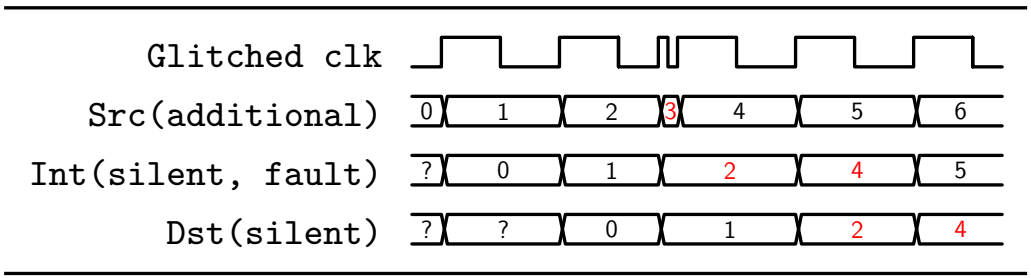
Post-synthesis clock glitch timing simulation I



I. ALSHAER et al., submitted to JCEN

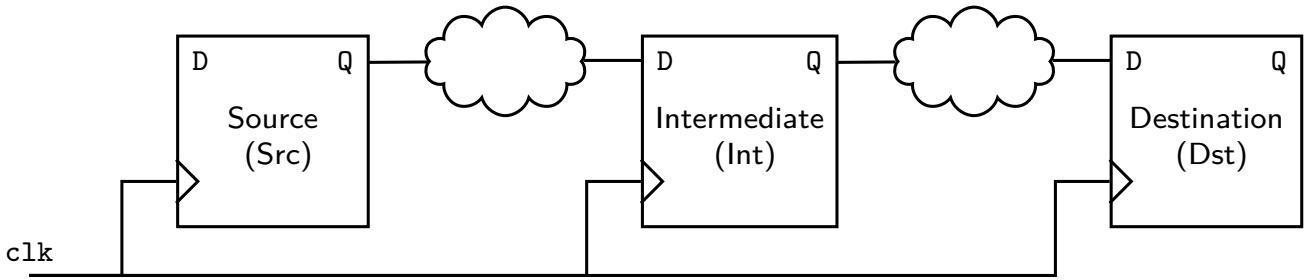
Post-synthesis clock glitch timing simulation II

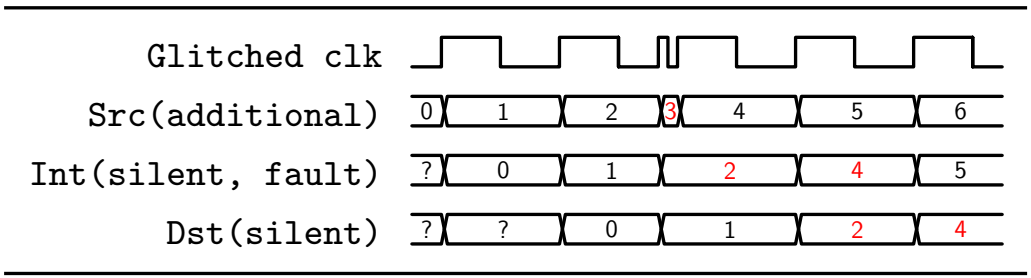




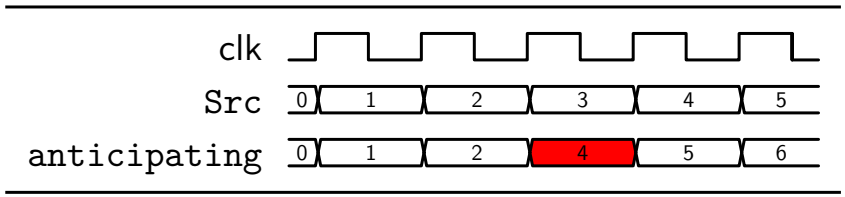
I. ALSHAER et al., submitted to JCEN

Post-synthesis clock glitch timing simulation II



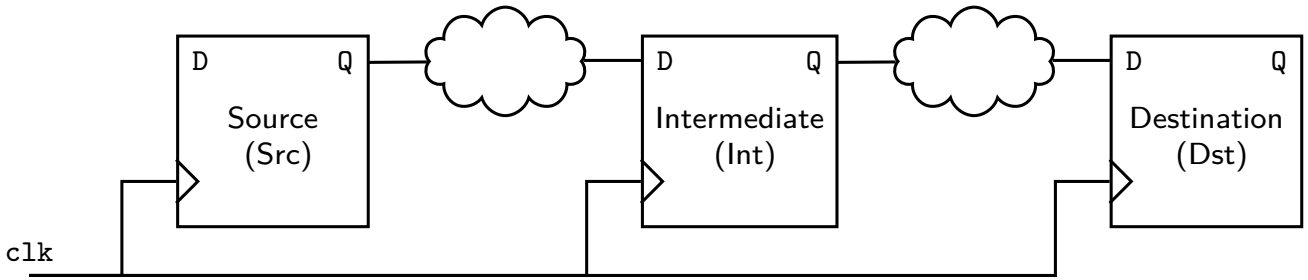


- Fault simulation at RTL
  - skip 3

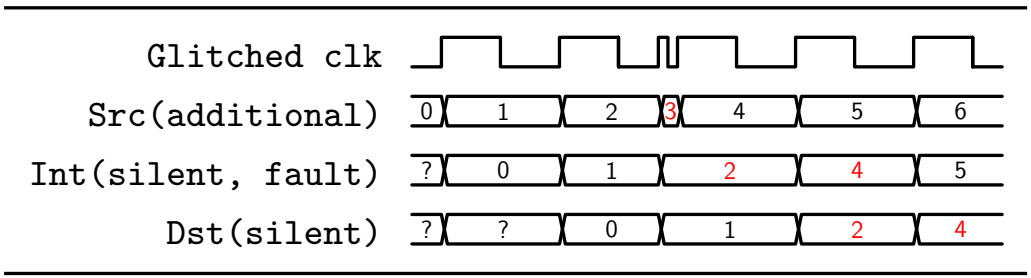


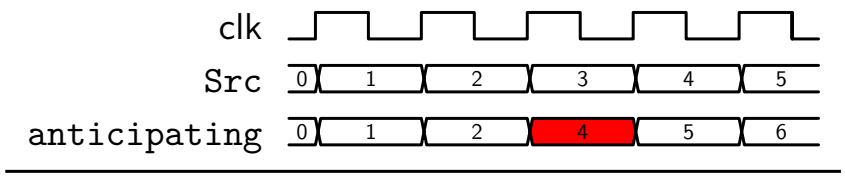
I. ALSHAER et al., submitted to JCEN

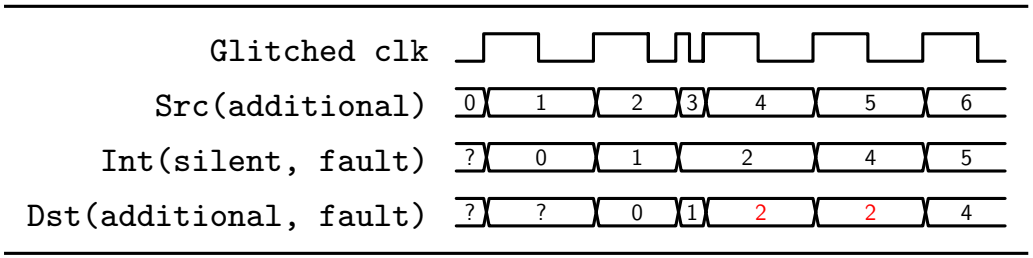
Post-synthesis clock glitch timing simulation II



- Fault simulation at RTL
  - skip 3

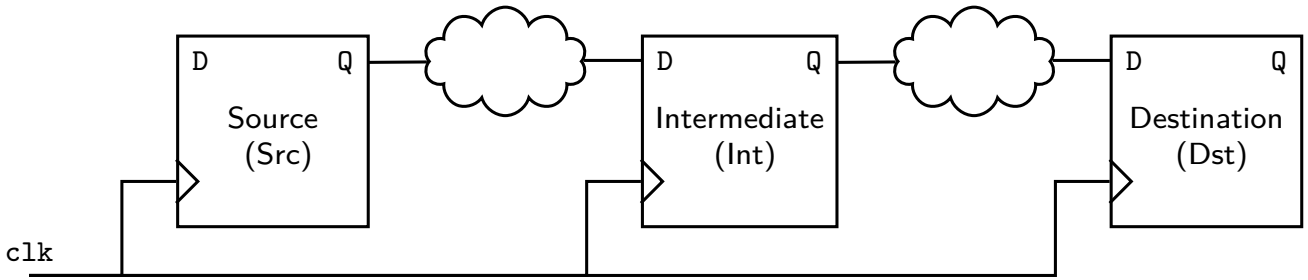


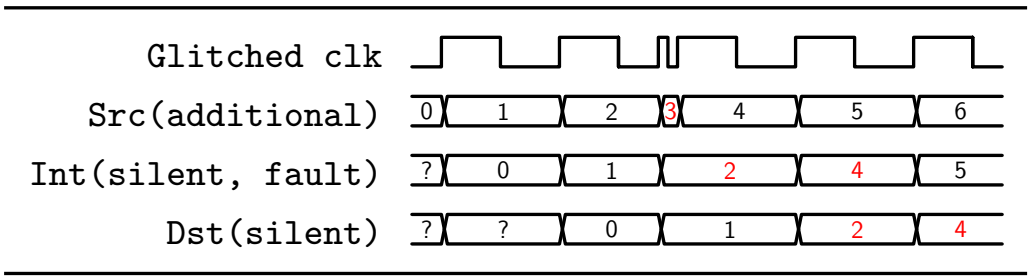




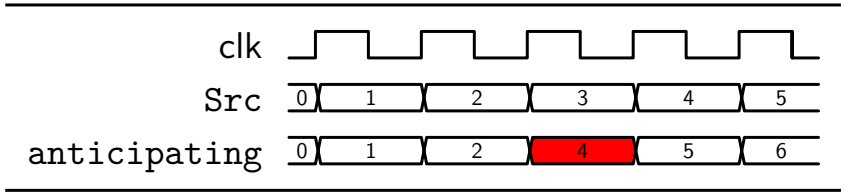
I. ALSHAER et al., submitted to JCEN

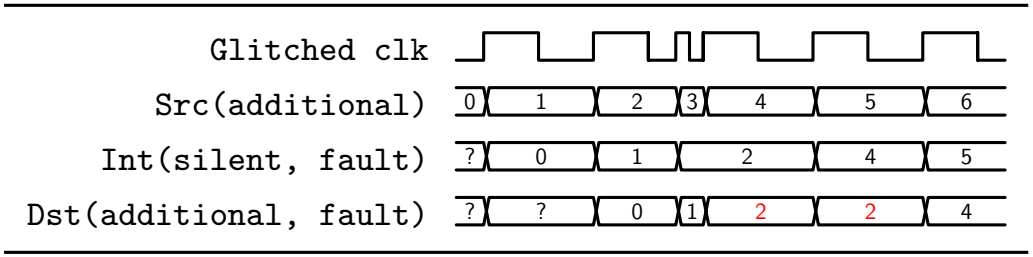
Post-synthesis clock glitch timing simulation II



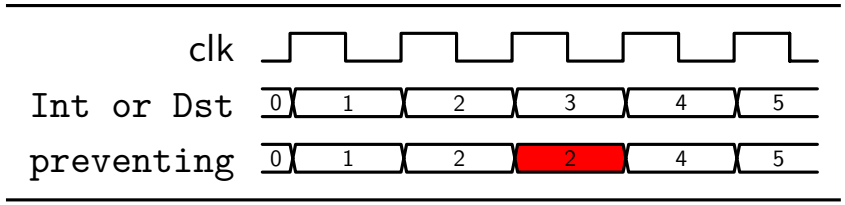


- Fault simulation at RTL
  - skip 3

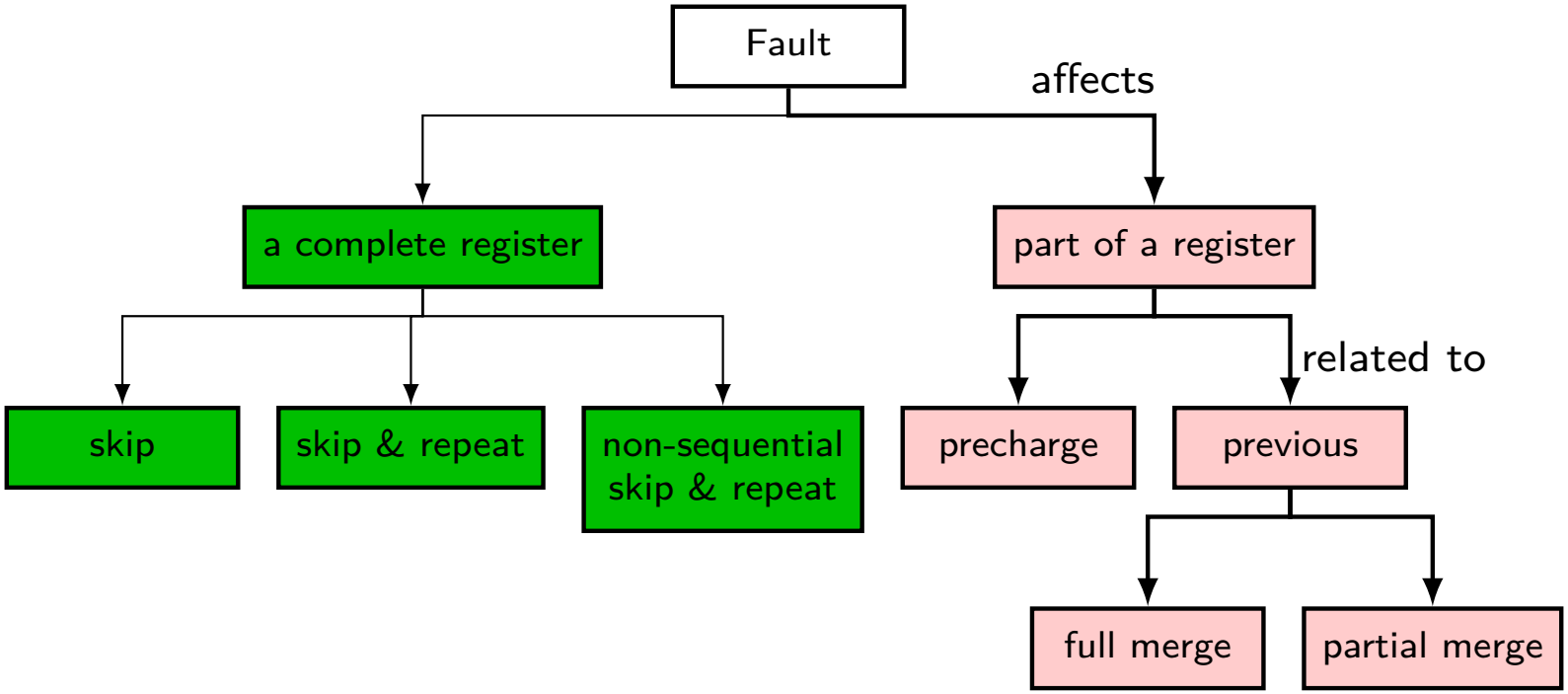




  - skip 3 & repeat 2

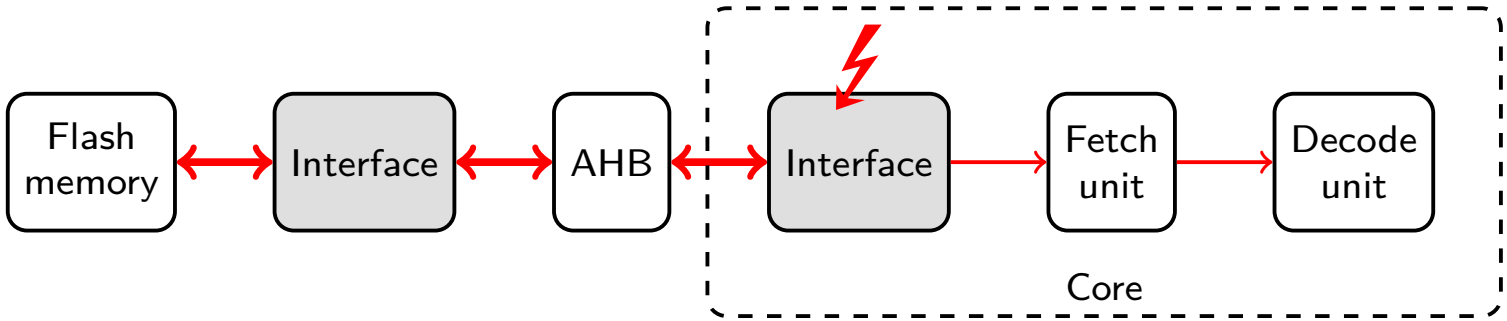


I. ALSHAER et al., submitted to JCEN


Fault
affects
a complete register
part of a register
skip
skip & repeat
non-sequential
skip & repeat
related to
precharge
previous
full merge
partial merge

Partial update fault model

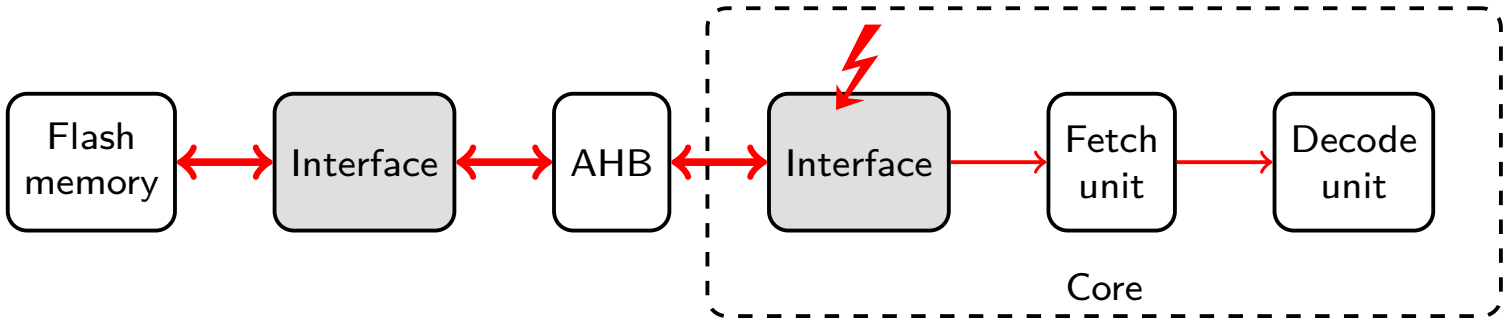
- Not all flip-flops within a register in the path end up faulty.
  $\Rightarrow$ Partially correct.



I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

Partial update fault model

- Not all flip-flops within a register in the path end up faulty.
  $\Rightarrow$ Partially correct.



- Based on the faulty register and/or the glitch parameters, the remaining part gets its value either:
  - from the **precharge** value:
    - explained as **multi-bit reset**, assuming the precharge values are zeros.
  - from the **previous** value:
    - explained as **bit-wise OR** between the previous value and the correct value (**full** or **partial** merge)

I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

Further results and details

Program Counter (PC) modification I

```
1  R8 = address of line 11
2  // series of 0x0000
3  ADD R6, R1, 0x4c7
4  ADD R3, R3, 0xa
5  ADD R4, R4, 0xb
6  ADD R5, R6, R3
7  ADD R3, R3, 0xf
8  // series of 0x0000
9  ADD R5, R5, 0x5
10 // series of 0x0000
11 ADD R1, R1, 0x3
12 ADD R9, R0, R6
```

- The encoding of `ADD R6, R1, 0x4c7` is `0xf201 46c7`.
- `0x46c7` is the encoding of **MOV PC, R8**.
- Scenarios:

I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

Program Counter (PC) modification I

```
1  R8 = address of line 11
2  // series of 0x0000
3  ADD R6, R1, 0x4c7
4  ADD R3, R3, 0xa
5  ADD R4, R4, 0xb
6  ADD R5, R6, R3
7  ADD R3, R3, 0xf
8  // series of 0x0000
9  ADD R5, R5, 0x5
10 // series of 0x0000
11 ADD R1, R1, 0x3
12 ADD R9, R0, R6
```

- The encoding of `ADD R6, R1, 0x4c7` is `0xf201` `46c7`.
- `0x46c7` is the encoding of **MOV PC, R8**.
- Scenarios:
  - Misaligned code: `0xf201` `0x46c7`

I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

Program Counter (PC) modification I

```
1  R8 = address of line 11
2  // series of 0x0000
3  ADD R6, R1, 0x4c7
4  ADD R3, R3, 0xa
5  ADD R4, R4, 0xb
6  ADD R5, R6, R3
7  ADD R3, R3, 0xf
8  // series of 0x0000
9  ADD R5, R5, 0x5
10 // series of 0x0000
11 ADD R1, R1, 0x3
12 ADD R9, R0, R6
```

- The encoding of `ADD R6, R1, 0x4c7` is `0xf201` `46c7`.
- `0x46c7` is the encoding of **MOV PC, R8**.
- Scenarios:
  - Misaligned code: `0xf201` `0x46c7`
    - Skip fault model
    - Success rate: 100 %
    - {shift = -12, width =3}

I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

Program Counter (PC) modification I

```
1  R8 = address of line 11
2  // series of 0x0000
3  ADD R6, R1, 0x4c7
4  ADD R3, R3, 0xa
5  ADD R4, R4, 0xb
6  ADD R5, R6, R3
7  ADD R3, R3, 0xf
8  // series of 0x0000
9  ADD R5, R5, 0x5
10 // series of 0x0000
11 ADD R1, R1, 0x3
12 ADD R9, R0, R6
```

- The encoding of `ADD R6, R1, 0x4c7` is `0xf201` `46c7`.
- `0x46c7` is the encoding of **MOV PC, R8**.
- Scenarios:
  - Misaligned code: `0xf201` `0x46c7`
    - Skip fault model
    - Success rate: 100 %
    - {shift = -12, width =3}
  - Aligned code by adding 0x0000 `0xf20146c7`

I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

Program Counter (PC) modification I

```
1  R8 = address of line 11
2  // series of 0x0000
3  ADD R6, R1, 0x4c7
4  ADD R3, R3, 0xa
5  ADD R4, R4, 0xb
6  ADD R5, R6, R3
7  ADD R3, R3, 0xf
8  // series of 0x0000
9  ADD R5, R5, 0x5
10 // series of 0x0000
11 ADD R1, R1, 0x3
12 ADD R9, R0, R6
```

- The encoding of `ADD R6, R1, 0x4c7` is `0xf201` `46c7`.
- `0x46c7` is the encoding of **MOV PC, R8**.
- Scenarios:
  - Misaligned code: `0xf201` `0x46c7`
    - Skip fault model
    - Success rate: 100 %
    - {shift = -12, width =3}
  - Aligned code by adding 0x0000 `0xf20146c7`
    - Partial update from precharge value
    - Two 16-bit instructions will be executed, e.g., `0x4200 (TST R0, R0)` and then `0x46c7`.
    - Success rate = 0.71 %
    - {shift = -13, width =10}

I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

Program Counter (PC) modification I

```
1  R8 = address of line 11
2  // series of 0x0000
3  ADD R6, R1, 0x4c7
4  ADD R3, R3, 0xa
5  ADD R4, R4, 0xb
6  ADD R5, R6, R3
7  ADD R3, R3, 0xf
8  // series of 0x0000
9  ADD R5, R5, 0x5
10 // series of 0x0000
11 ADD R1, R1, 0x3
12 ADD R9, R0, R6
```

- The encoding of `ADD R6, R1, 0x4c7` is `0xf201 46c7`.
- `0x46c7` is the encoding of **MOV PC, R8**.
- Scenarios:
  - Misaligned code: `0xf201` `0x46c7`
    - Skip fault model
    - Success rate: 100 %
    - {shift = -12, width =3}
  - Aligned code by adding 0x0000 `0xf20146c7`
    - Partial update from precharge value
    - Two 16-bit instructions will be executed, e.g., `0x4200 (TST R0, R0)` and then `0x46c7`.
    - Success rate = 0.71 %
    - {shift = -13, width =10}

- Countermeasure: **register substitution**
  - misaligned
  - replace R6 with R2 (0x46c7 => 0x42c7)
  - Success rate = 0 %

I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

Program Counter (PC) modification II

Continue on scenarios: Trojan

- Dummy code has no effect on the original code, but it implements a Trojan that can be activated by a fault injection.

```
1 CMP  R1, R0       // 0x4281
2 MOVS R0, R0       // 0x0000
3 MOVS R0, R0       // 0x0000
4 MOVS R0, R0       // 0x0000
5 LSLS R6, R0, 0x11// 0x0446
6 MOVS R0, R0       // 0x0000
7 MOVS R0, R0       // 0x0000
8 MOVS R0, R0       // 0x0000
```

I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

Program Counter (PC) modification II

Continue on scenarios: Trojan

- Dummy code has no effect on the original code, but it implements a Trojan that can be activated by a fault injection.

```
1 CMP  R1, R0          // 0x4281
2 MOVS R0, R0          // 0x0000
3 MOVS R0, R0          // 0x0000
4 MOVS R0, R0          // 0x0000
5 LSLS R6, R0, 0x11// 0x0446
6 MOVS R0, R0          // 0x0000
7 MOVS R0, R0          // 0x0000
8 MOVS R0, R0          // 0x0000
```

- Partial update from previous value with Full merge:
- 0x4281 | 0x0446 = 0x46c7

I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

Program Counter (PC) modification II

Continue on scenarios: Trojan

- Dummy code has no effect on the original code, but it implements a Trojan that can be activated by a fault injection.

```
1 CMP  R1, R0          // 0x4281
2 MOVS R0, R0          // 0x0000
3 MOVS R0, R0          // 0x0000
4 MOVS R0, R0          // 0x0000
5 LSLS R6, R0, 0x11// 0x0446
6 MOVS R0, R0          // 0x0000
7 MOVS R0, R0          // 0x0000
8 MOVS R0, R0          // 0x0000
```

- Partial update from previous value with Full merge:
- 0x4281 | 0x0446 = 0x46c7
- Success rate: 95.11 %
- {shift = -9, width = 4}

I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

Program Counter (PC) modification II

Continue on scenarios: Trojan

- Dummy code has no effect on the original code, but it implements a Trojan that can be activated by a fault injection.

```
1 CMP  R1, R0          // 0x4281
2 MOVS R0, R0          // 0x0000
3 MOVS R0, R0          // 0x0000
4 MOVS R0, R0          // 0x0000
5 LSLS R6, R0, 0x11// 0x0446
6 MOVS R0, R0          // 0x0000
7 MOVS R0, R0          // 0x0000
8 MOVS R0, R0          // 0x0000
```

- Partial update from previous value with Full merge:
- 0x4281 | 0x0446 = 0x46c7
- Success rate: 95.11 %
- {shift = -9, width = 4}

- The malicious user or tool can be either:
  - the software developer himself,
  - the compiler is an untrusted compiler.

I. ALSHAER et al., *Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection*, CARDIS 2023

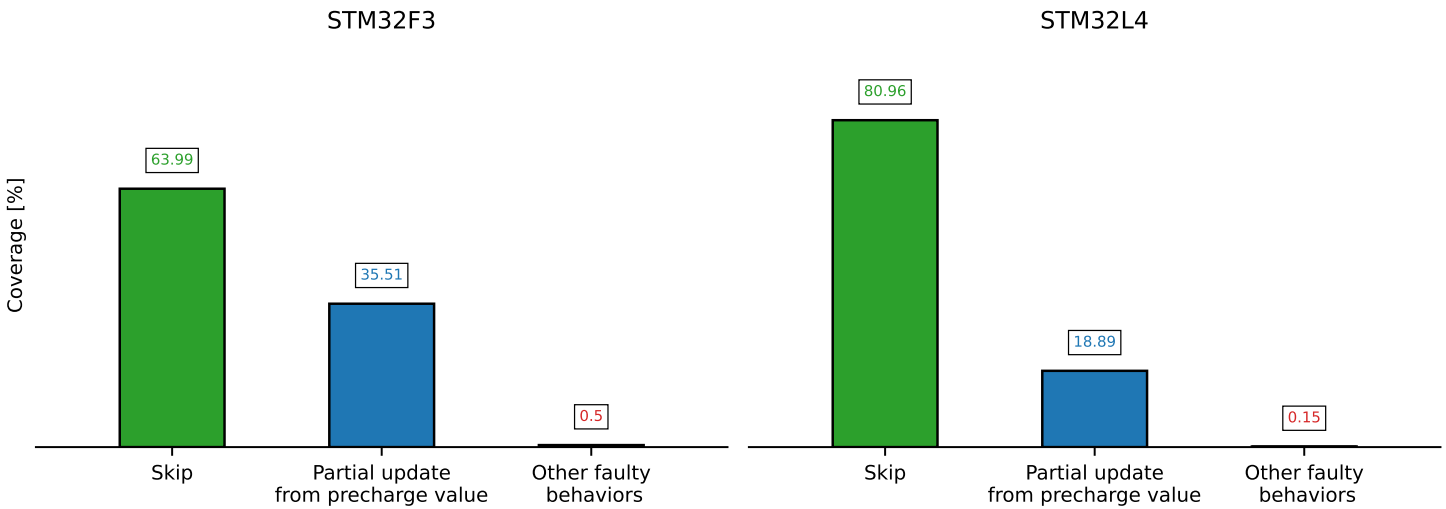
Fault models evaluation: Clock glitch

- Coverage: measures the portion of the observed faulty behaviors that have corresponding fault models.

Fault models evaluation: Clock glitch

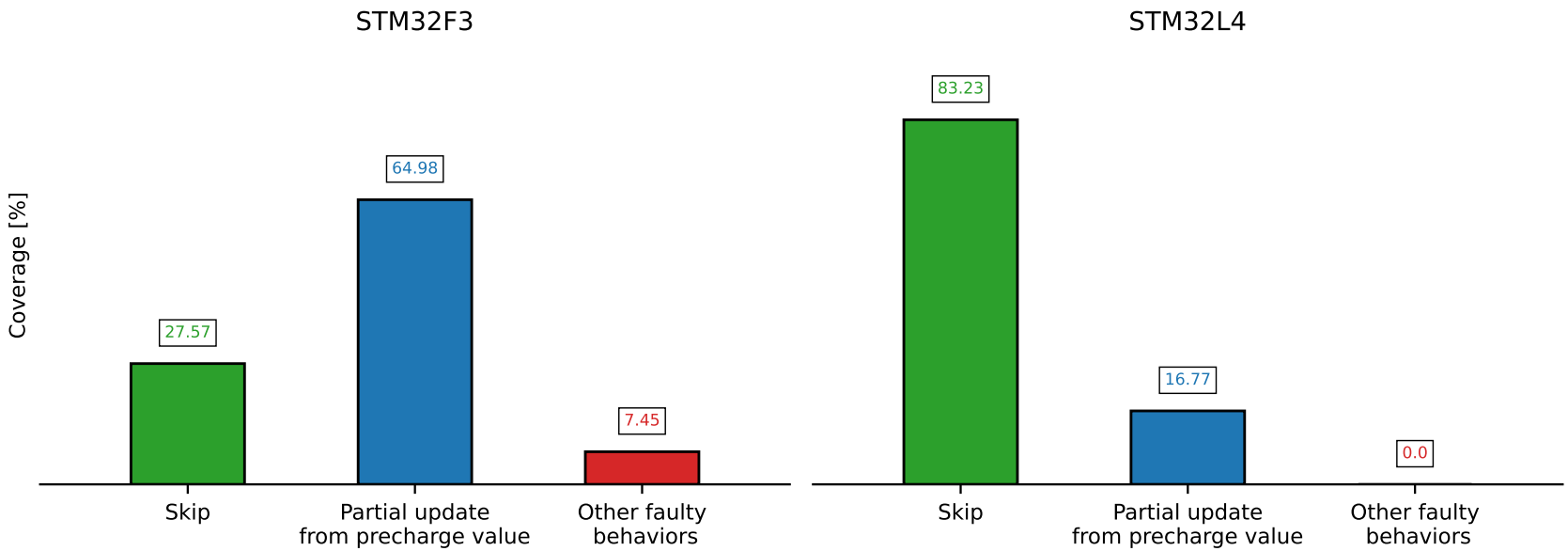
- Coverage: measures the portion of the observed faulty behaviors that have corresponding fault models.

- Coverage $>$ 99 % when targeting a series of `0x332b (ADDS R3, 0x2b)`.



Fault models evaluation: Voltage glitch

- Coverage $>$ 92% (92.55% for STM32F3, 100% for STM32L4)



Conclusion and perspectives

Conclusion I

- Fault effect analysis, at a **single level** of abstraction, **limits the understanding** of the fault effect *[ALSHAER et al., DTIS 2021]*
  - non-realistic and incomplete fault models would lead to over- or under-protections.

Conclusion I

- Fault effect analysis, at a **single level** of abstraction, **limits the understanding** of the fault effect *[ALSHAER et al., DTIS 2021]*
  - non-realistic and incomplete fault models would lead to over- or under-protections.

- **Cross-layer** analysis led to **better understanding** *[ALSHAER et al., MICROREL, vol139, 2022], [ALSHAER et al., submitted to JCEN]*
  - novel fault models, at SW and HW, have been proposed.
    - **skip** *[ALSHAER et al., DSD 2022]*
    - **skip & repeat** *[ALSHAER et al., DSD 2022]*
    - **non-sequential skip & repeat**
    - partial update from **precharge** value *[ALSHAER et al., CARDIS 2023]*
    - partial update from **previous** value *[ALSHAER et al., CARDIS 2023]*
    - **anticipate** the update *[ALSHAER et al., submitted to JCEN]*
    - **prevent** the update *[ALSHAER et al., submitted to JCEN]*

Conclusion I

- Fault effect analysis, at a **single level** of abstraction, **limits the understanding** of the fault effect *[ALSHAER et al., DTIS 2021]*
  - non-realistic and incomplete fault models would lead to over- or under-protections.

- **Cross-layer** analysis led to **better understanding** *[ALSHAER et al., MICROREL, vol139, 2022], [ALSHAER et al., submitted to JCEN]*
  - novel fault models, at SW and HW, have been proposed.
    - **skip** *[ALSHAER et al., DSD 2022]*
    - **skip & repeat** *[ALSHAER et al., DSD 2022]*
    - **non-sequential skip & repeat**
    - partial update from **precharge** value *[ALSHAER et al., CARDIS 2023]*
    - partial update from **previous** value *[ALSHAER et al., CARDIS 2023]*
    - **anticipate** the update *[ALSHAER et al., submitted to JCEN]*
    - **prevent** the update *[ALSHAER et al., submitted to JCEN]*
  - **high coverage, high fidelity, and low complexity**

Conclusion II

- Fault models' **reliability** is **validated** by **predicting** the possible faulty behaviors before conducting the injection, e.g.,:
  - PC modification *[ALSHAER et al., DSD 2022]*
  - AES implementations *[ALSHAER et al., submitted to JCEN]*
  - Trojan *[ALSHAER et al., CARDIS 2023]*

Conclusion II

- Fault models' **reliability** is **validated** by **predicting** the possible faulty behaviors before conducting the injection, e.g.,:
  - PC modification *[ALSHAER et al., DSD 2022]*
  - AES implementations *[ALSHAER et al., submitted to JCEN]*
  - Trojan *[ALSHAER et al., CARDIS 2023]*

- **Combinations** of the observed behaviors are obtained by **multiple-glitch** fault injection.

Conclusion II

- Fault models' **reliability** is **validated** by **predicting** the possible faulty behaviors before conducting the injection, e.g.,:
  - PC modification *[ALSHAER et al., DSD 2022]*
  - AES implementations *[ALSHAER et al., submitted to JCEN]*
  - Trojan *[ALSHAER et al., CARDIS 2023]*
- **Combinations** of the observed behaviors are obtained by **multiple-glitch** fault injection.
- The fault models were able to **explain** faulty behaviors observed by **clock & voltage** glitch fault injection.

Perspective future works

- Different **architectures** from Arm Cortex-M (RISC-V)
- Different fault injection **techniques** (EM & laser)
- **Countermeasure** design at SW and/or HW levels
- **Vulnerability** analysis on **real-life** security codes (beyond AES)

Thank you! Questions?

Ihab ALSHAER

ihab.alshaer@univ-grenoble-alpes.fr

References I

[1] C. Spensky, A. Machiry, N. Burow, et al., "Glitching demystified: Analyzing control-flow-based glitching attacks and defenses," in 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems IEEE, 2021, pp. 400–412.

[2] F. Khelil, M. Hamdi, S. Guilley, J. Danger, and N. Selmane, "Fault analysis attack on an FPGA AES implementation," in NTMS 2008, 2nd International Conference on New Technologies, Mobility and IEEE, 2008, pp. 1–5.

[3] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "CLKSCREW: exposing the perils of security-oblivious energy management," in 26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, C E. Kirda and T. Ristenpart, Eds., USENIX Association, 2017, pp. 1057–1074.

References II

[4] R. Buhren, H. N. Jacob, T. Krachenfels, and J. Seifert, "One glitch to rule them all: Fault injection attacks against amd's secure encrypted virtualization," in CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications S Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds., ACM, 2021, pp. 2875–2889.

[5] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz, "Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller," in 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, 2013, pp. 77–88.

[6] O. Trabelsi, L. Sauvage, and J.-L. Danger, "Characterization of electromagnetic fault injection on a 32-bit microcontroller instruction buffer," in 2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), 2020, pp. 1–6.

References III

[7] V. Werner, L. Maingault, and M. Potet, "An end-to-end approach for multi-fault attack vulnerability assessment," in Workshop on Fault Detection and Tolerance in Cryptography, Milan, Italy: IEEE, 2020, pp. 10–17.

[8] N. Timmers, A. Spruyt, and M. Witteman, "Controlling PC on ARM Using Fault Injection," in 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016, pp. 25–35.

[9] J. Proy, K. Heydemann, A. Berzati, F. Majéric, and A. Cohen, "A first ISA-level characterization of EM pulse effects on superscalar microarchitectures: A secure software perspective," in Proceedings of the 14th International Conference on Availability, Reliability an ACM, 2019, 7:1–7:10.

References IV

[10] T. Trouchkine, G. Bouffard, and J. Clédière, "EM fault model characterization on socs: From different architectures to the same fault model," in 2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC), IEEE, 2021, pp. 31–38.

[11] V. Khuat, J.-L. Danger, and J.-M. Dutertre, "Laser fault injection in a 32-bit microcontroller: From the flash interface to the execution pipeline," in 2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC), 2021, pp. 74–85.

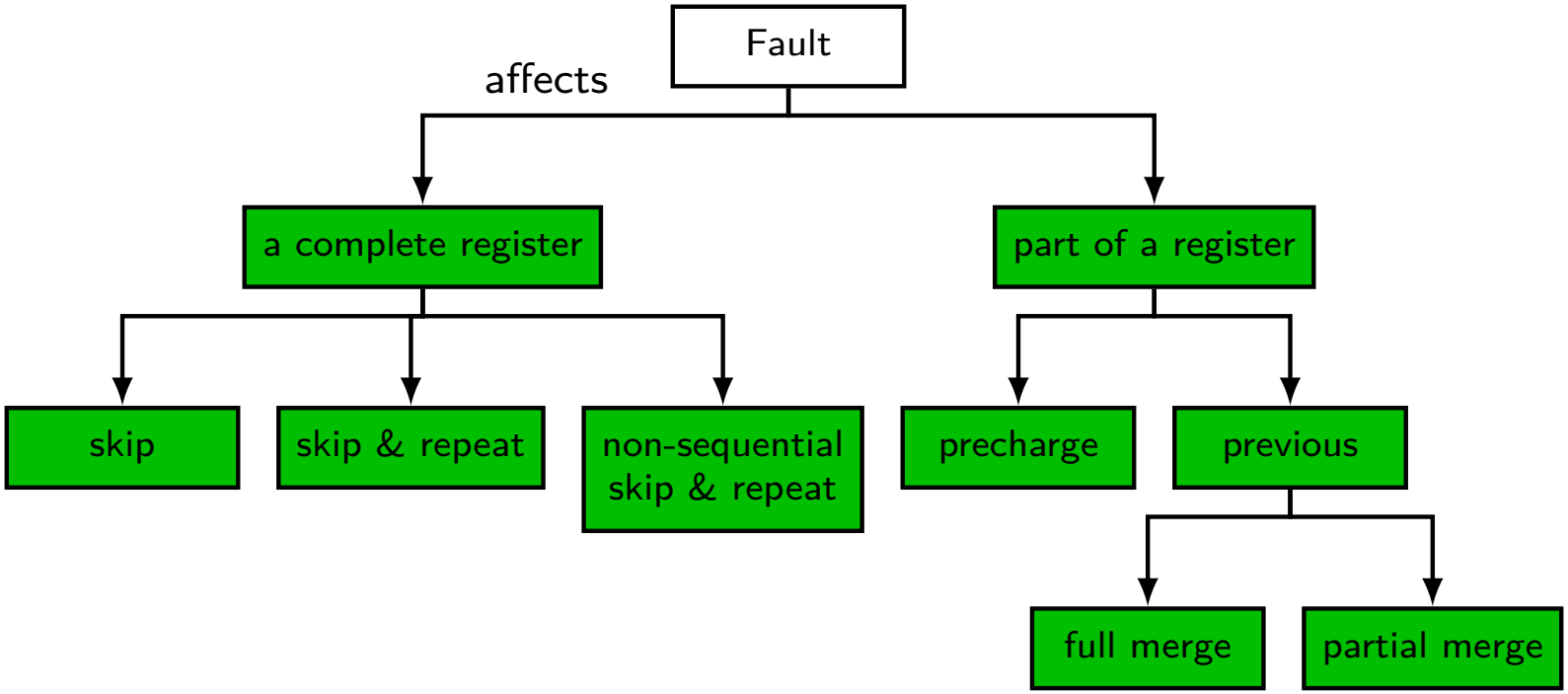
[12] J. Laurent, V. Beroulle, C. Deleuze, F. Pebay-Peyroula, and A. Papadimitriou, "Cross-layer analysis of software fault models and countermeasures against hardware fault attacks in a RISC-V processor," Microprocessors and Microsystems, vol. 71, 2019.

[13] S. Tollec, M. Asavoae, D. Couroussé, K. Heydemann, and M. Jan, "Exploration of fault effects on formal RISC-V microarchitecture models," in Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2022, Vir IEEE, 2022, pp. 73–83.

References V

[14] C. O'Flynn and Z. (Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in International Workshop on Constructive Side-Channel Analysis and Secure Des E. Prouff, Ed., ser. Lecture Notes in Computer Science, vol. 8622, Paris, France: Springer, 2014, pp. 243–260.

[16] First slide image [Accessed: September 14, 2023]. [Online]. Available: https://www.pinterest.com/pin/554716879105762154/.


Fault
affects
a complete register
part of a register
skip
skip & repeat
non-sequential
skip & repeat
precharge
previous
full merge
partial merge

Multiple glitch fault injection

- Faulty behaviors, that can be explained as a combination of the previous fault models, are observed.
- Example: combination of skip & repeat faults as a result of 2 glitches on STM32L4.

```
1 46a00602
2 f2030109
3 f2010308
4 f104040b
5 eb060503
6 f103030f
7 f1060607
8 f1050505
```

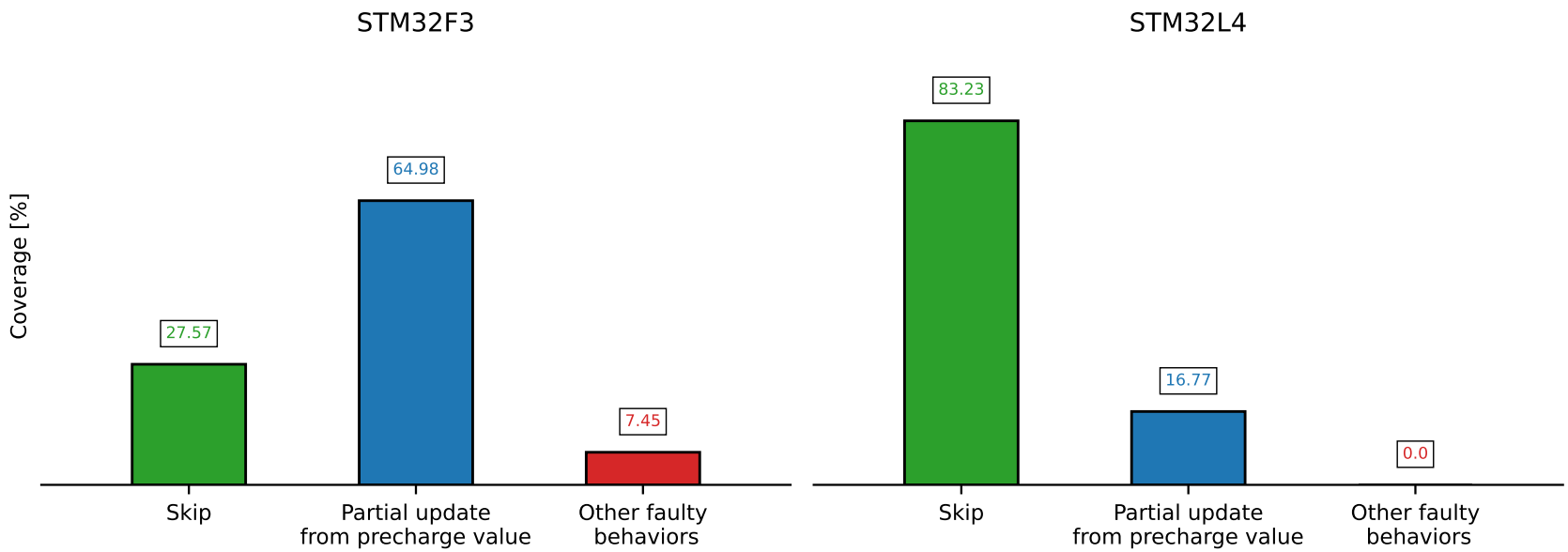
1
2
3
2
3
3
7
8

1
2
3
2
3
2
7
8

1
2
3
2
3
2
3
8

Voltage glitch fault injection

- Faulty behaviors, that can be explained using the previous fault models, are observed.
  - Skip, Skip & repeat, non-sequential skip & repeat , and Partial update (both cases).
- Targeting a series of `ADDS R3, 0x2b` (`0x332b`) => (F3: 92.55 %, L4: 100 %).



- Success rate of executing `MOV PC, R8` using the Trojan scenario was 92.1 % (clock: 95.11 %).