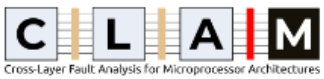
Microarchitectural Insights into Unexplained Behaviors under Clock Glitch Fault Injection

Ihab Alshaer, Brice Colombier, Christophe Deleuze, Vincent Beroulle & Paolo Maistri

CARDIS 2023
Amsterdam, Netherlands

November 15th, 2023



Introduction

Fault injection attack

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

- Main techniques:

Clock glitch

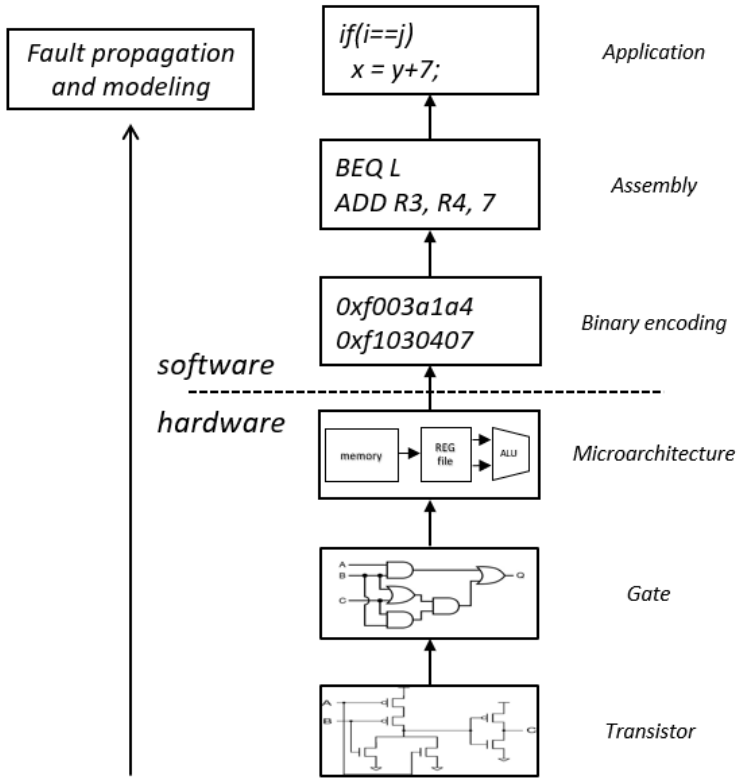
Voltage glitch

EM pulses

Laser

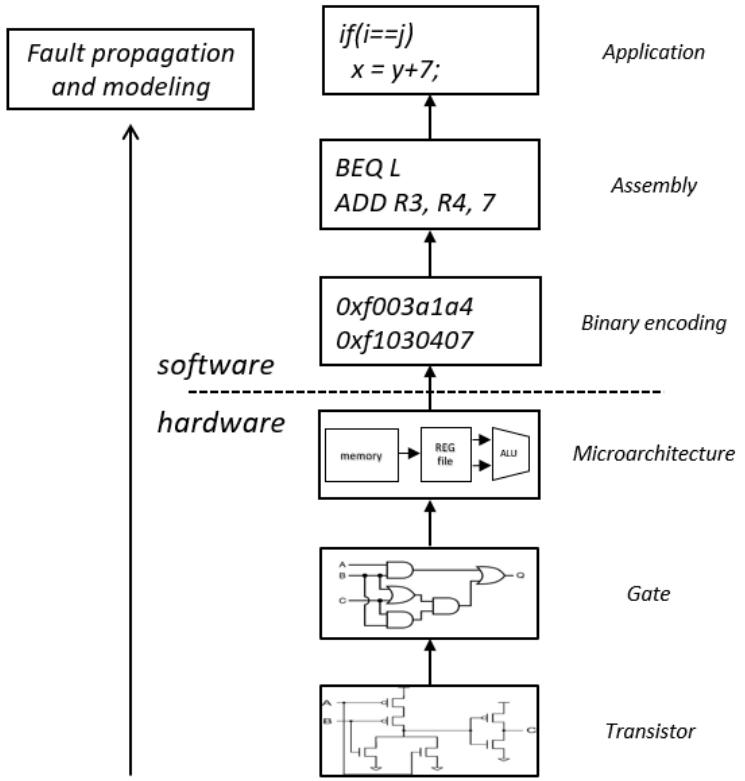
Problematic and objective

- 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 different levels of abstractions.

Fault models at binary encoding level

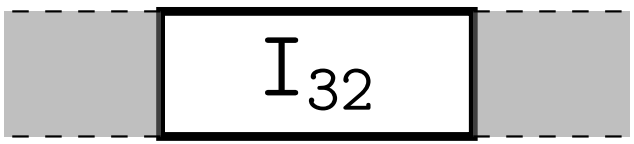
- *Skip* a specific number of bits
- *Skip and Repeat* a specific number of bits

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

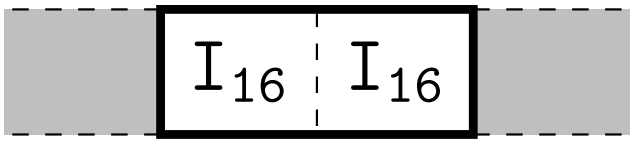
Fault models at binary encoding level

- *Skip* a specific number of bits
- *Skip and Repeat* a specific number of bits
- The effect is different based on the alignment in memory.

Fetching aligned instructions



(a)



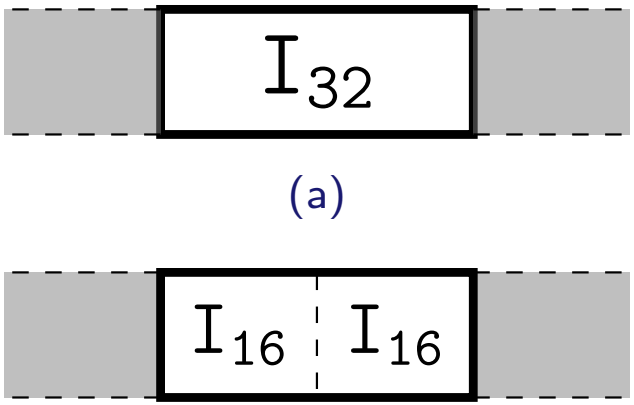
(b)

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

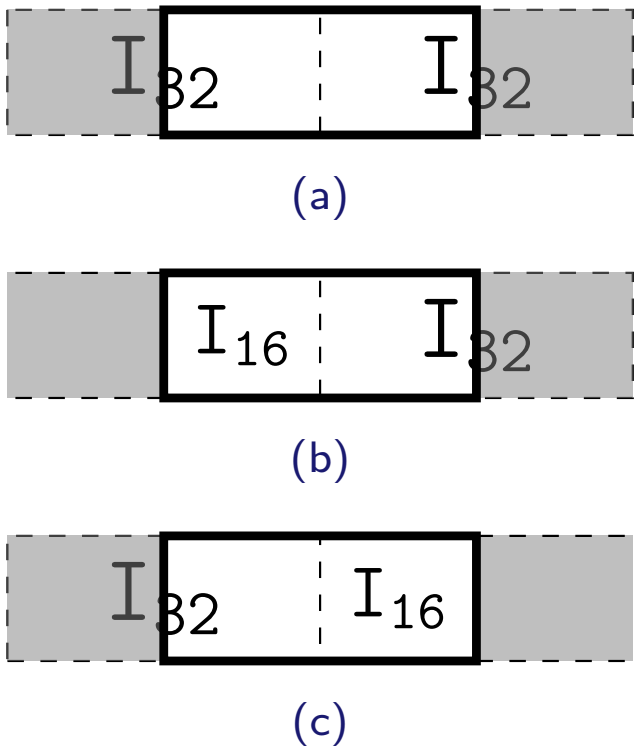
Fault models at binary encoding level

- *Skip* a specific number of bits
- *Skip and Repeat* a specific number of bits
- The effect is different based on the alignment in memory.

Fetching aligned instructions



Fetching misaligned instructions



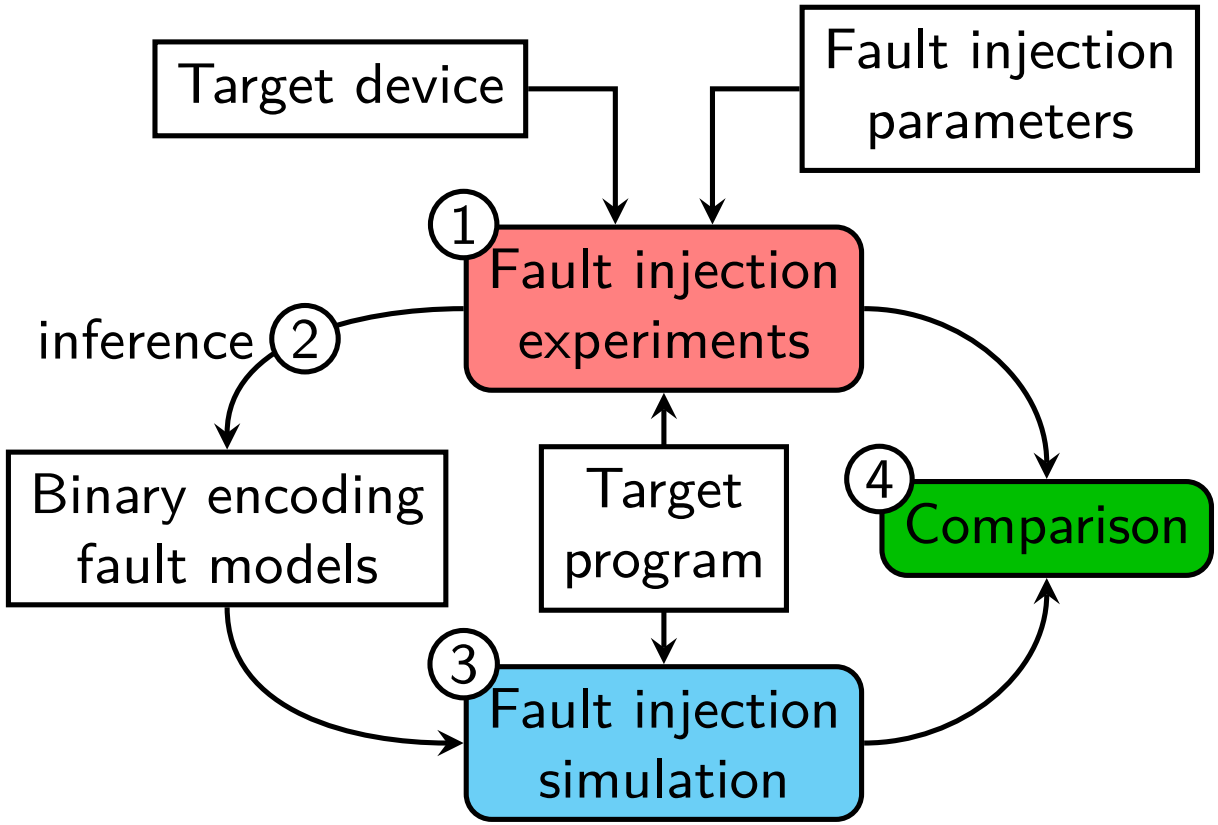
I. ALSHAER et al.,*Variable-length Instruction Set: Feature or Bug?*, DSD 2022

Main contributions

- Present a new inferred fault model at the binary encoding level of instructions:
  - Partial update fault model:
    - from precharge value
    - from previous value
- Explain a wide range of the obtained faulty behaviors.
- Show that *partial update from precharge value*:
  - instruction-independent with high probability
  - highly device-dependent (and it physical implementation)
- Show the ability to execute new instructions even with a different length of encoding.
- Make use of the presented fault models to modify the program counter (PC).

Fault models inference

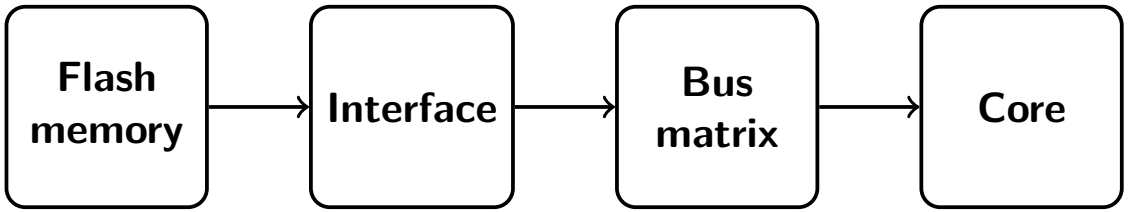
Overview



- The objectives of the comparison are to:
  - explain the observed faulty behaviors,
  - provide proper description of the fault effects.
  - thus, to propose realistic fault models.

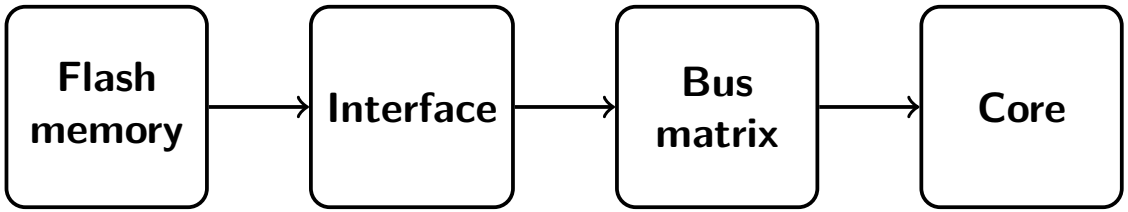
Partial update fault model

- Not all flip-flops within a register or buffer in the fetch path end up faulty.
  $=>$ Partially correct



Partial update fault model

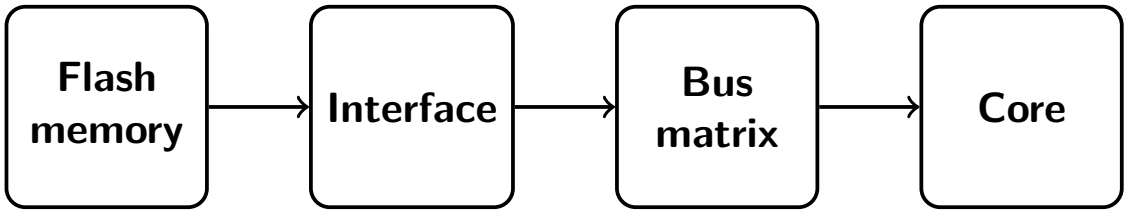
- Not all flip-flops within a register or buffer in the fetch path end up faulty.
  $=>$ Partially correct



- 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.

Partial update fault model

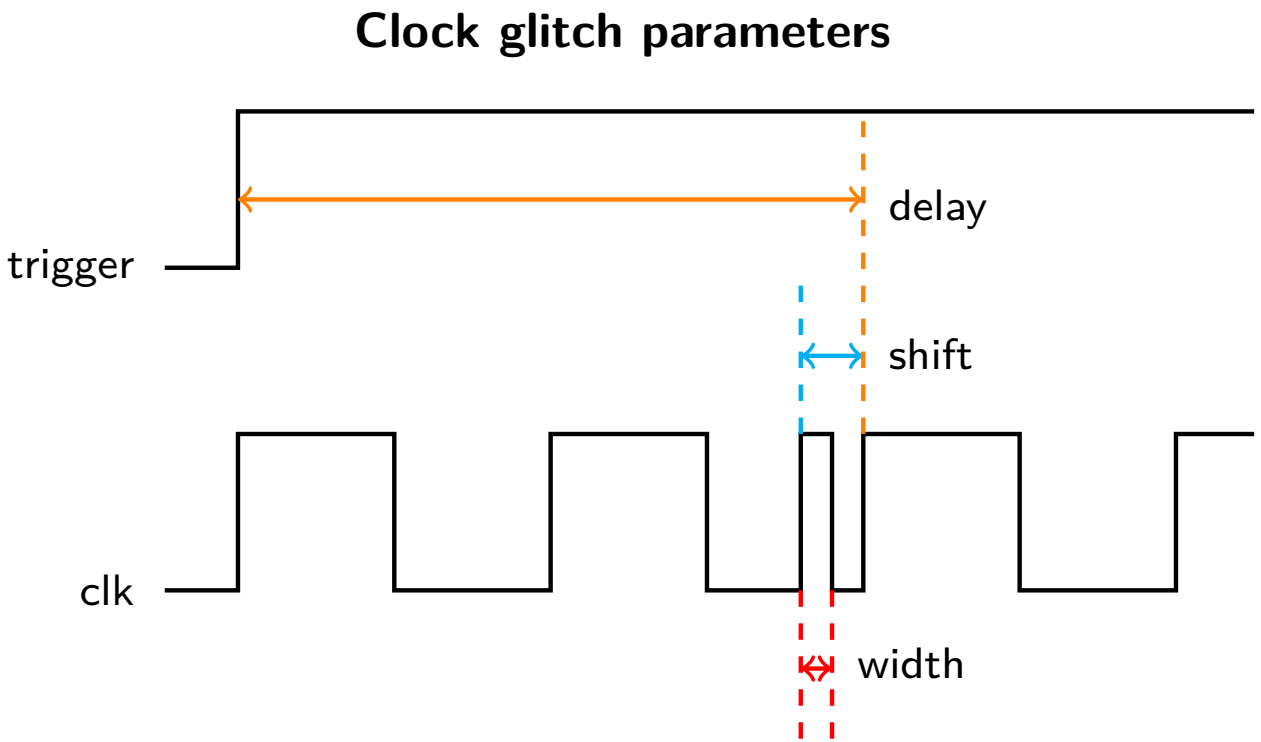
- Not all flip-flops within a register or buffer in the fetch path end up faulty.
  $=>$ Partially correct



- 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).

Experimental setup

Clock glitch fault injection

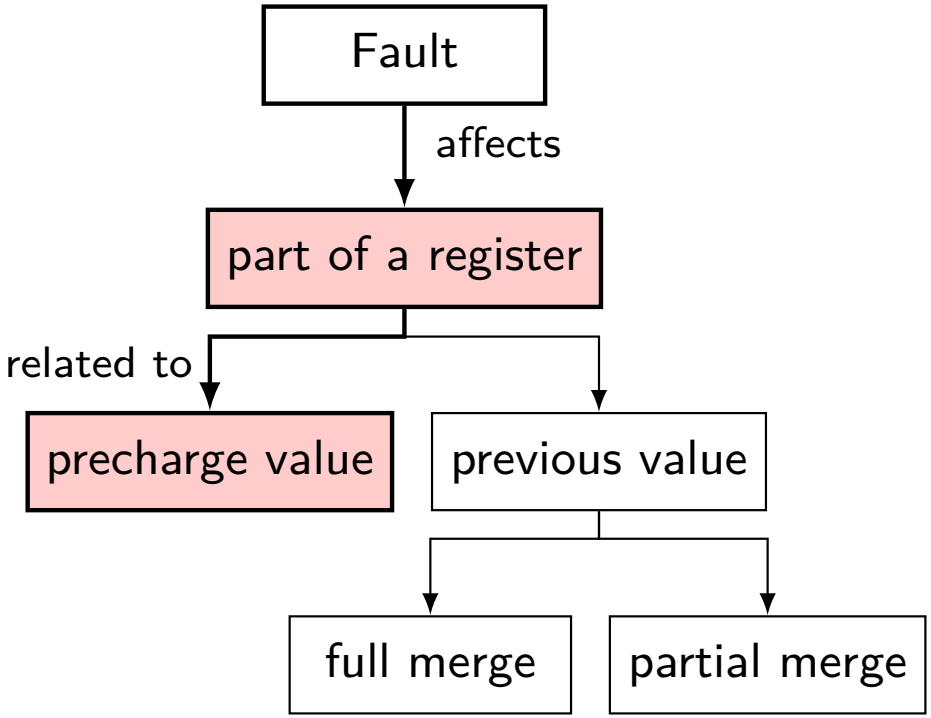


- ChipWhisperer environment is used to perform the clock glitch fault injection.

Target device

- 32-bit micro-controller.
- Embeds Arm Cortex-M4 processor.
- Supports Thumb2 instruction set:
  - 16- and 32-bit instructions.
- The flash memory access size is fixed and equals 64 bits.
- The instructions are either aligned or misaligned in the flash memory.

Experimental results


Fault
affects
part of a register
related to
precharge value
previous value
full merge
partial merge

Partial update from precharge value I

- Target a high Hamming weight instruction: `SUBS R6, 0xff (0x3eff)`
- 4 injection campaigns (20 440 executions in each campaign)

Position	1^{st}	2^{nd}	3^{rd}	4^{th}
Target program	0x3eff	0x0000	0x0000	0x0000
	0x0000	0x3eff	0x0000	0x0000
	0x0000	0x0000	0x3eff	0x0000
	0x0000	0x0000	0x0000	0x3eff

Partial update from precharge value I

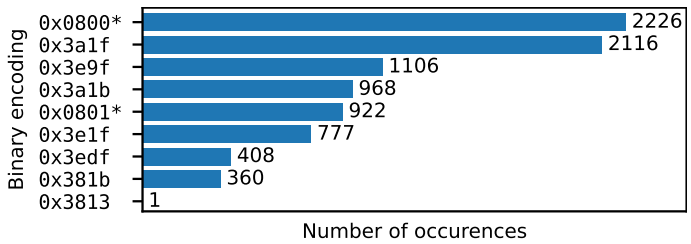
- Target a high Hamming weight instruction: `SUBS R6, 0xff (0x3eff)`
- 4 injection campaigns (20 440 executions in each campaign)

Position	1st	2nd	3rd	4th
Target program	0x3eff	0x0000	0x0000	0x0000
	0x0000	0x3eff	0x0000	0x0000
	0x0000	0x0000	0x3eff	0x0000
	0x0000	0x0000	0x0000	0x3eff

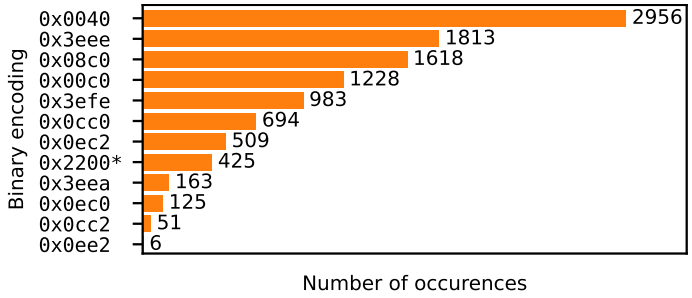
Class	Position 1st	2nd	3rd	4th
Crash	0	0	23	1
Silent	33	1574	2273	158
Fault	20 407	18 866	18 144	20 281
Skip	11 523	8295	11 107	7901
Partial update from the precharge value	8884	10 571	7037	12 380

Partial update from precharge value II: faulty executions

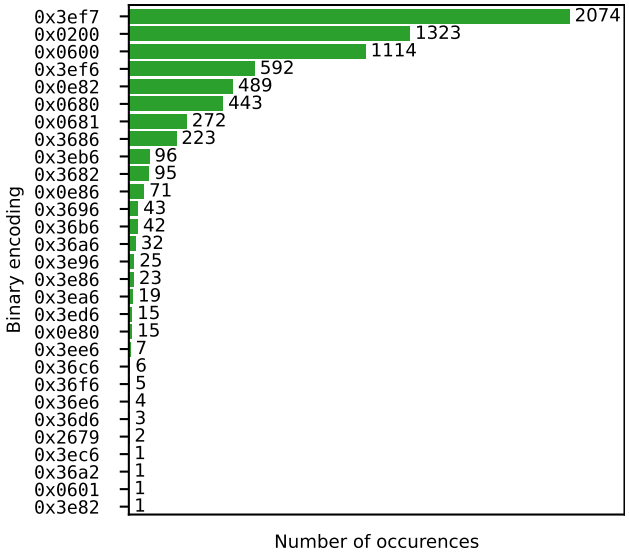
results of targeting 0x3eff at four different positions



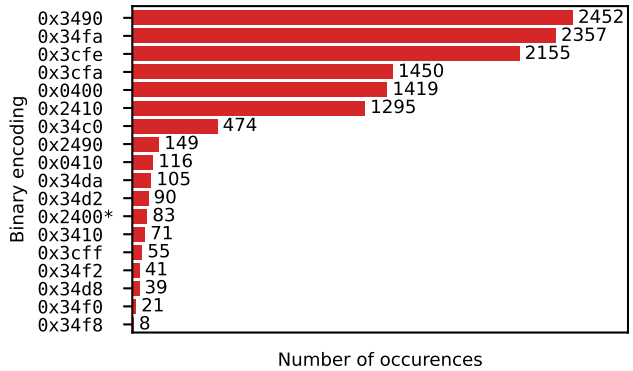
(a) 1st position



(b) 2nd position



(c) 3rd position



(d) 4th position

Partial update from precharge value III: Bit sensitivity

Bit sensitivity values when targeting `0x3eff` at four different positions

- Bit sensitivity $S_p(f, b)$: measures how much a bit is sensitive to be reset.

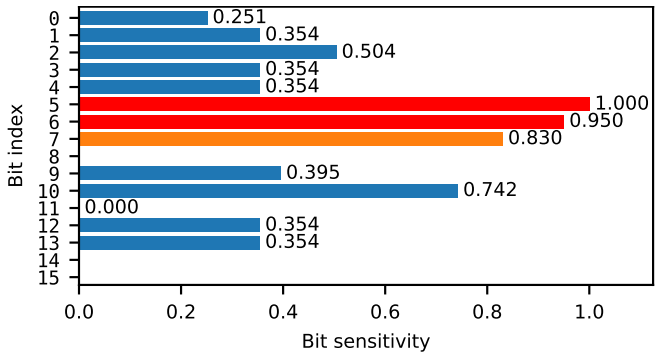
$$S_p(f, b) = 1 - \frac{P(b = 1 \mid p)}{P(\text{fault model} = f \mid p)} \tag{1}$$

Partial update from precharge value III: Bit sensitivity

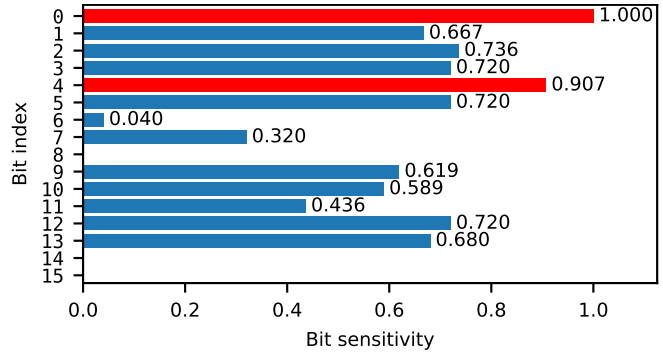
Bit sensitivity values when targeting `0x3eff` at four different positions

- Bit sensitivity $S_p(f, b)$: measures how much a bit is sensitive to be reset.

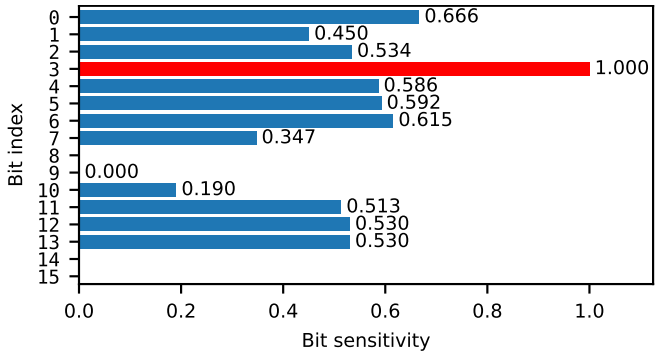
$$S_p(f, b) = 1 - \frac{P(b = 1 \mid p)}{P(\text{fault model} = f \mid p)} \tag{1}$$



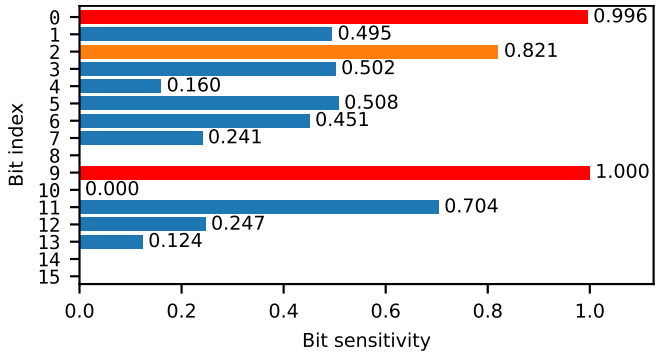
(a) 1st position



(b) 2nd position



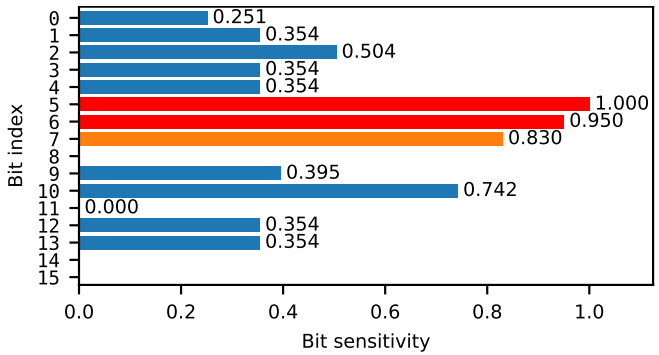
(c) 3rd position



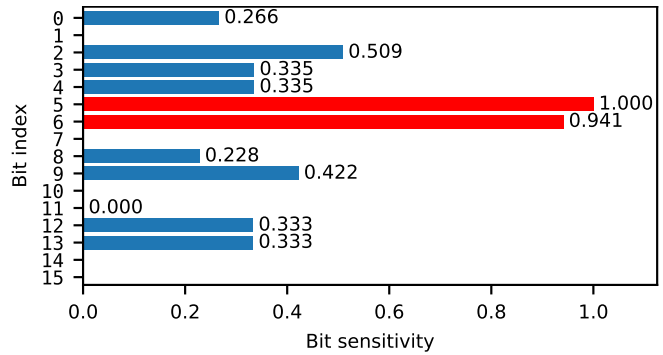
(d) 4th position

Partial update from precharge value IV

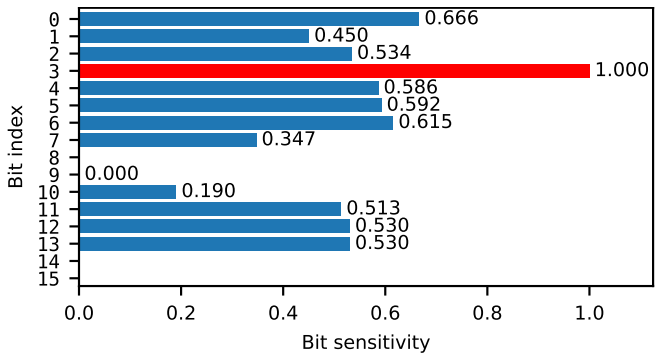
- Observing almost similar sensitivities when targeting different instructions (`0x3b7d: SUBS R3, 0x7d`).



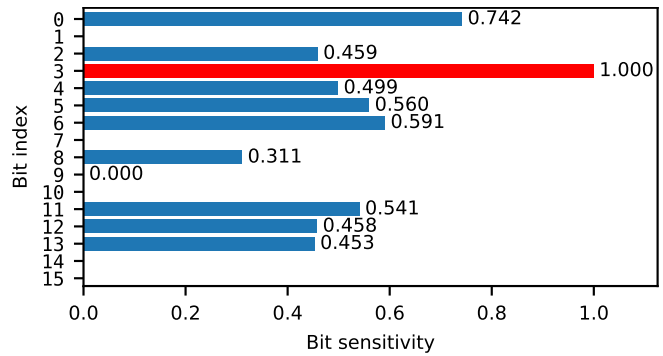
(a) 1st position 0x3eff



(b) 1st position 0x3b7d



(c) 3rd position 0x3eff

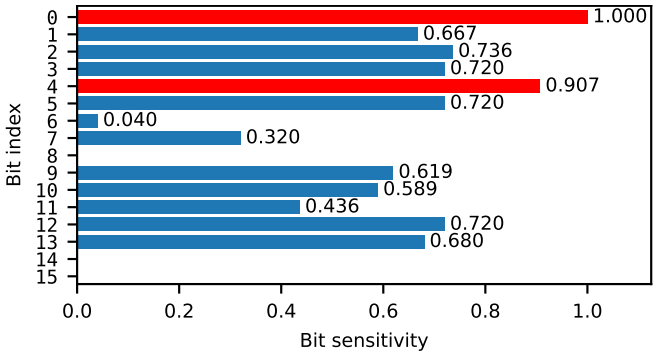


(d) 3rd position 0x3b7d

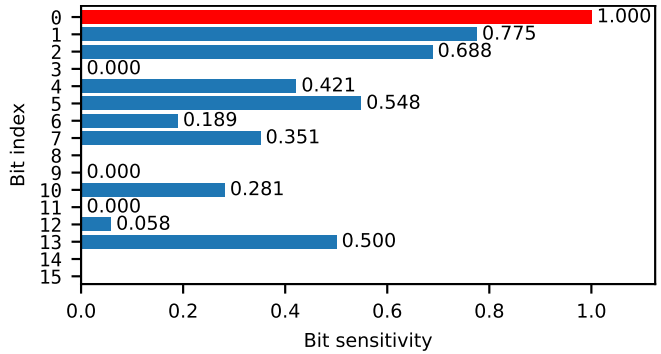
Partial update from precharge value V

Targeting 0x3eff using different devices

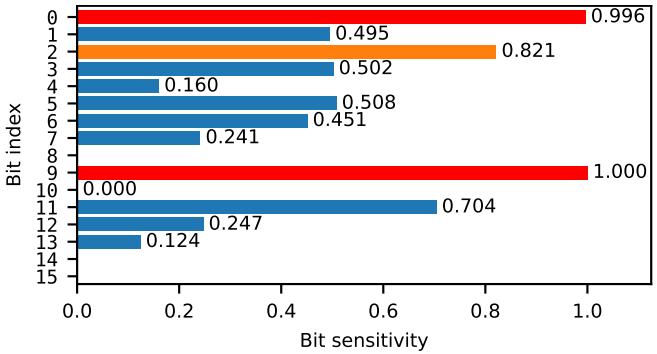
- Observing different sensitivities when targeting different devices.



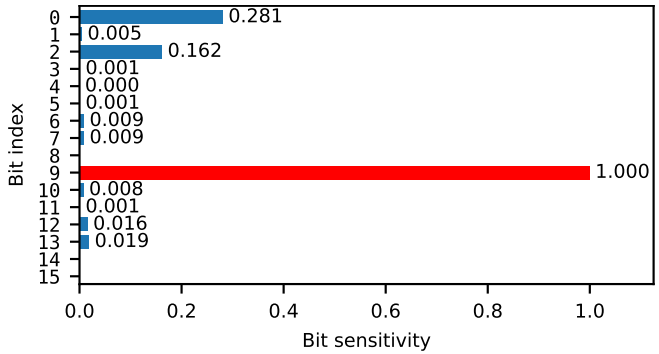
(a) 2nd position using old MCU



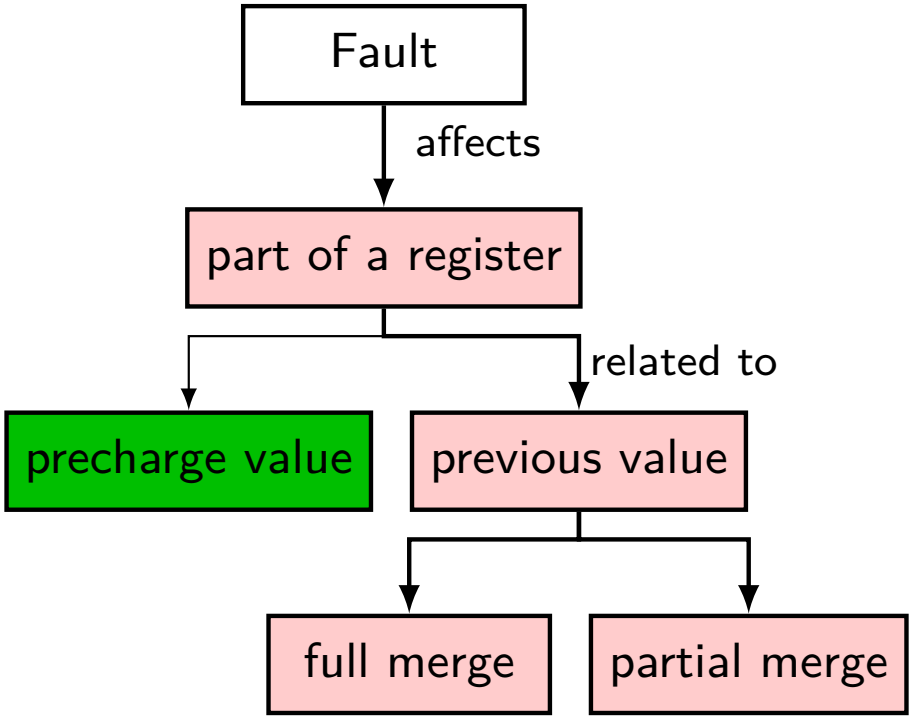
(b) 2nd position using new MCU



(c) 4th position using old MCU



(d) 4th position using new MCU


Fault
affects
part of a register
related to
precharge value
previous value
full merge
partial merge

Partial update from previous value I

Full merge

- Example of observed Full merge between two 64 bits:

```
1 ADD  R1, SP, 0x0     // 0xa900
2 MOVS R0, R0          // 0x0000
3 ADD  R1, SP, 0x0     // 0xa900
4 MOVS R0, R0          // 0x0000
5 TST  R0, R0          // 0x4200
6 LSLS R5, R0, 0xc     // 0x0305
7 TST  R2, R0          // 0x4202
8 LSLS R5, R0, 0x10    // 0x0405
```

Golden execution

```
1 ADD  R1, SP, 0x0     // 0xa900
2 MOVS R0, R0          // 0x0000
3 ADD  R1, SP, 0x0     // 0xa900
4 MOVS R0, R0          // 0x0000
5 ADD  R3, R0, R5   // 0xeb000305
6 ADD  R4, R2, R5   // 0xeb020405
```

Faulty execution

- Merging the 32 bits at lines 1 and 2 with the 32 bits at lines 5 and 6 respectively: 0xa9000000 | 0x42000305 = 0xeb000305 (ADD R3, R0, R5)
- Merging the 32 bits at lines 3 and 4 with the 32 bits at lines 7 and 8 respectively: 0xa9000000 | 0x42020405 = 0xeb020405 (ADD R4, R2, R5)

Partial update from previous value II

Partial merge

- Example of observed Partial merge between two 64 bits:

```
1 ADD  R1, R1, 0x4 // 0xf1010104
2 ANDS R2, R0      // 0x4002
3 MOVS R0, R0      // 0x0000
4 ADD  R2, R2, 0xa // 0xf102020a
5 MOVS R4, R0      // 0x0004
6 MOVS R0, R0      // 0x0000
```

Golden exection

```
1 ADD  R1, R1, 0x4 // 0xf1010104
2 ANDS R2, R0      // 0x4002
3 MOVS R0, R0      // 0x0000
4 ADD  R3, R3, 0xa // 0xf103030a
5 MOVS R6, R0      // 0x0006
6 MOVS R0, R0      // 0x0000
```

Faulty execution

- 0xf1 0101 04 | 0xf1 0202 0a = 0xf103030a
- 0x400 2 | 0x000 4 = 0x0006

Program counter modification

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:

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`

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}

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`

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}

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 %

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
```

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

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}

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.

Conclusion and perspectives

Conclusion

- Presented a new fault model: ***Partial update*** *fault model*
  - Partial update from **precharge** value
  - Partial update from **previous** value
  - Allowed **explaining** a wide range of the faulty behaviors that are obtained when performing clock glitch.
- Examined the **dependency** of partial update from the **precharge** value fault model with respect to the target **device** and **program**:
  - **program-independent**, but **device-dependent** with high probability
- Presented different scenarios to **modify** the **PC**
  - register substitution as a countermeasure

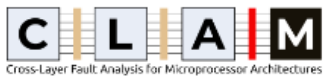
Perspective future works

- Proper formalization of **protections**
- Different **architectures** than Arm Cortex-M
- Alternative fault injection **techniques**
- More investigation on the target **device dependency** wrt partial update from **precharge** value:
  - Aging, process variation, environmental conditions, or power-supply noise?
    - **Bit sensitivity**

Thank you! Questions?

Ihab Alshaer, Brice Colombier, Christophe Deleuze, Vincent Beroulle & Paolo Maistri

ihab.alshaer@univ-grenoble-alpes.fr



State-of-the-art I

Fault effect characterization and modelling

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

State-of-the-art I

Fault effect characterization and modelling

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

- Fault effect characterization at **ISA**:
  - e.g., Moro et al.[4], Proy et al.[5], Trouchkine et al.[6], Khaut et al.[7].
  - Instruction(s) **skip** [5], [7].
  - Instruction(s) **skip** and **replay** [5], [7].
  - Instruction **corruption/replacement** [4], [6].
  - Register **corruption** [6].

State-of-the-art I

Fault effect characterization and modelling

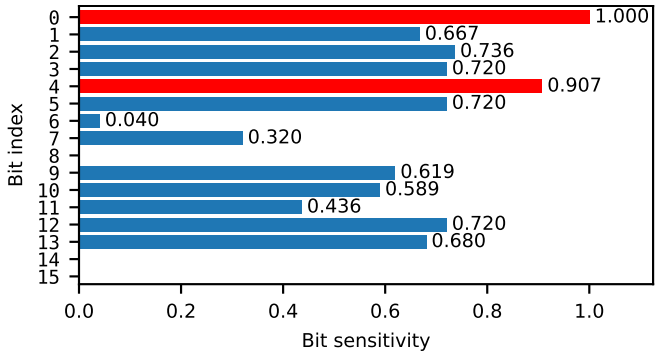
- **Random** fault effect:
  - e.g., Spensky et al.[1] Khelil et al.[2], Buhran et al.[3].
  - random bit/byte faults.

- Fault effect characterization at **ISA**:
  - e.g., Moro et al.[4], Proy et al.[5], Trouchkine et al.[6], Khaut et al.[7].
  - Instruction(s) **skip** [5], [7].
  - Instruction(s) **skip** and **replay** [5], [7].
  - Instruction **corruption/replacement** [4], [6].
  - Register **corruption** [6].

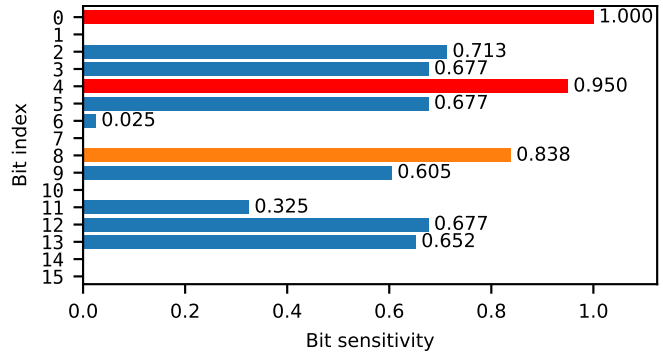
- Fault effect characterization at **ISA** and **RTL** levels
  - e.g., Laurent et al.[8], Tollec et al.[9].
  - only **simulations**, bit(s) **flip/set/reset**

Partial update from precharge value IV

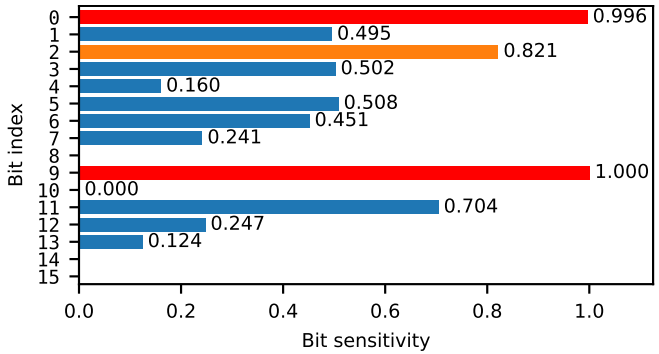
- Observing almost similar sensitivities when targeting different instructions (`0x3b7d: SUBS R3, 0x7d`).



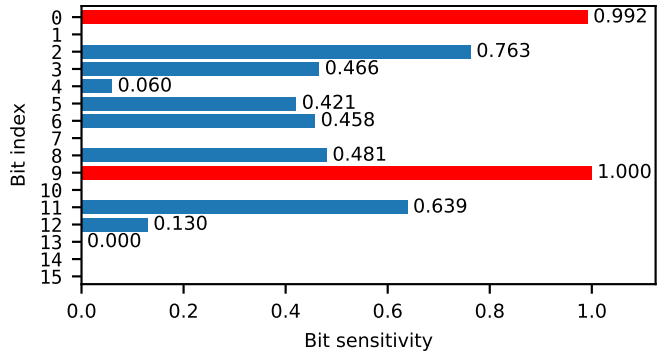
(a) 2nd position 0x3eff



(b) 2nd position 0x3b7d



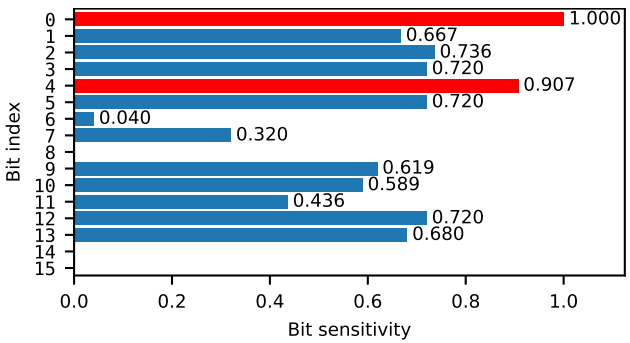
(c) 4th position 0x3eff



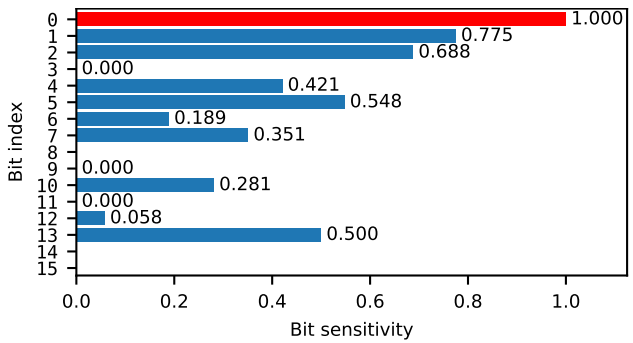
(d) 4th position 0x3b7d

Partial update from precharge value IV

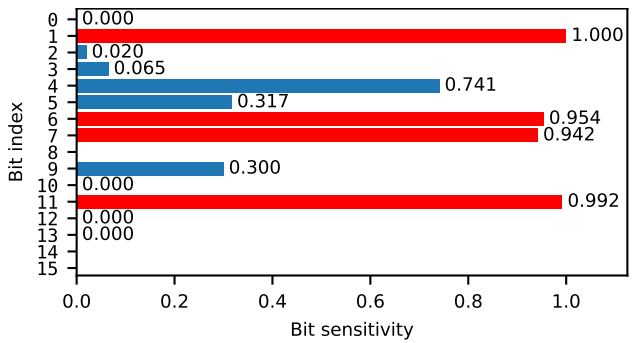
- Observing different sensitivities when targeting different devices.
- Example: targeting `0x3eff` at the 2nd position on three devices:



Old STM32F3



New STM32F3



Old STM32L4

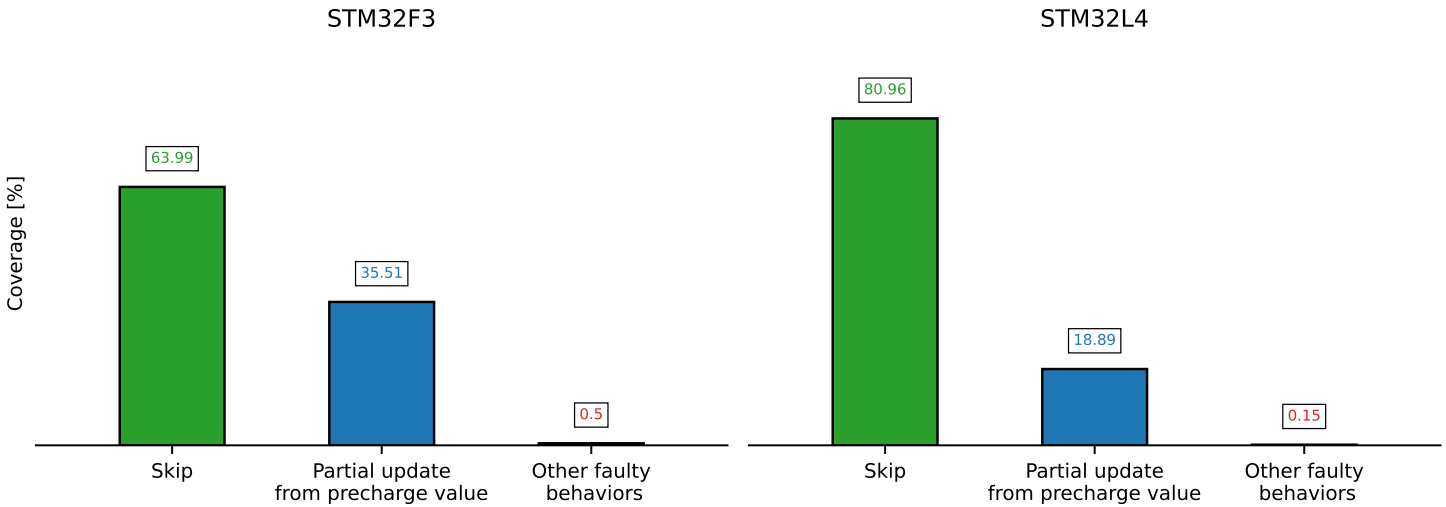
effect of partial update from the previous value on flip-flops that carry binary encoding of instructions

case	correct	previous	fault model effect
1st	1	1	1 (no fault)
2nd	1	0	1 (no fault)
3rd	0	0	0 (no fault)
4th	0	1	0 no fault, or 1 (fault)

- if **all** flip-flops that are under the 4th case have 1 => full merge
- if **some** flip-flops that are under the 4th case have 1 => partial merge.

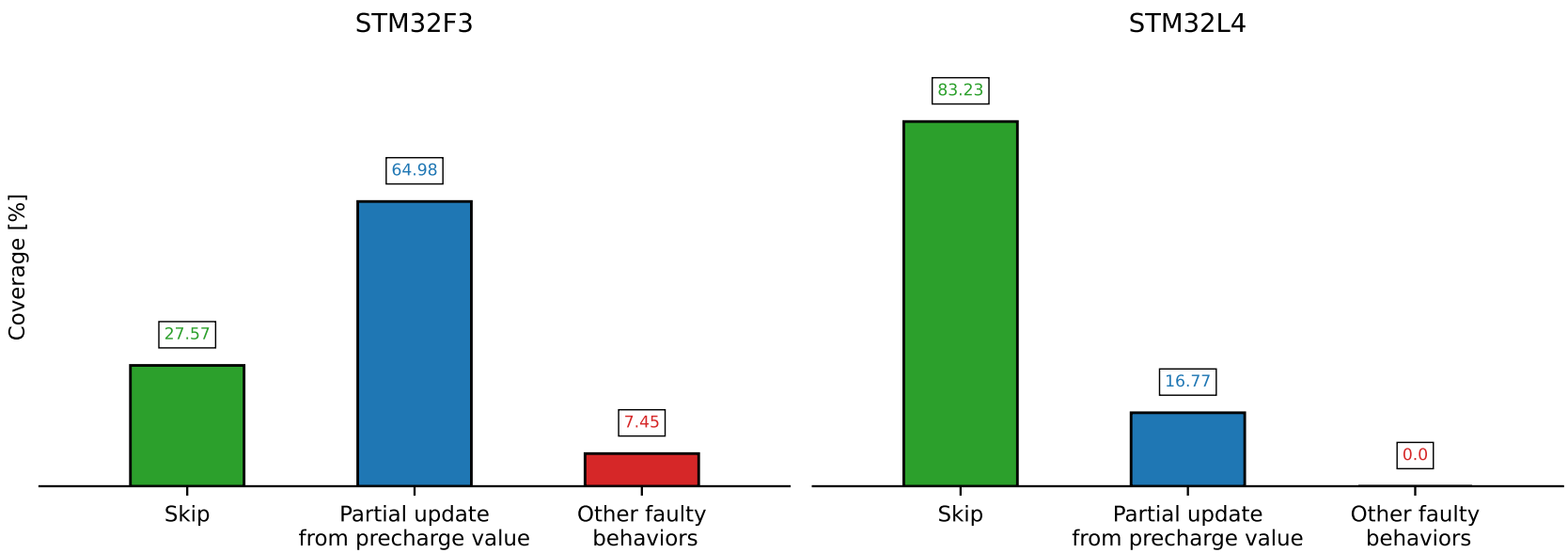
Clock glitch fault injection

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



Voltage glitch fault injection

- Targeting a series of `ADDS R3, 0x2b` (`0x332b`)



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

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 a 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 S IEEE, 2008, pp. 1–5.

[3] 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 Se Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds., ACM, 2021, pp. 2875–2889.

References II

[4] 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.

[5] 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 and ACM, 2019, 7:1–7:10.

[6] 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.

References III

[7] 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.

[8] 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.

[9] 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, Virt IEEE, 2022, pp. 73–83.