Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller

Brice Colombier[1], Alexandre Menu[2], Jean-Max Dutertre[2], Pierre-Alain Moëllic[3], Jean-Baptiste Rigaud[2] and Jean-Luc Danger[4]

[1]Univ Lyon, UJM-Saint-Etienne, CNRS, Laboratoire Hubert Curien UMR 5516
[2]IMT, Mines Saint-Etienne, Centre CMP, Equipe Commune CEA Tech - Mines Saint-Etienne
[3]CEA Tech, Centre CMP, Equipe Commune CEA Tech - Mines Saint-Etienne
[4]LTCI, Télécom ParisTech , Institut Mines-télécom, Université Paris Saclay

Cryptarchi workshop
June 25th 2019

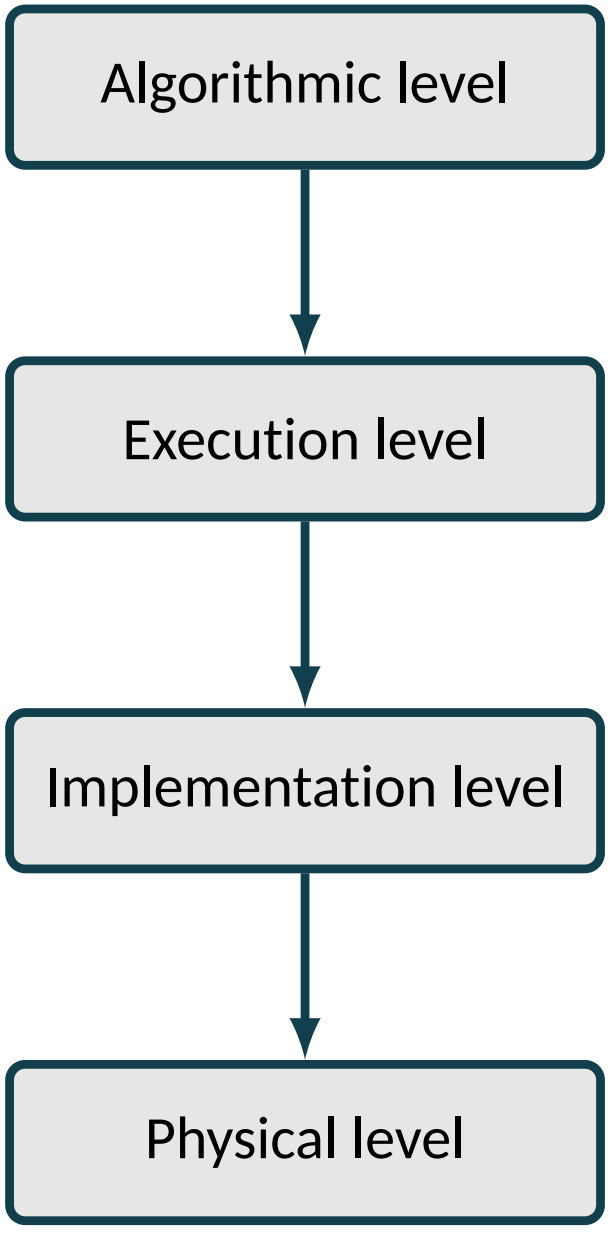
Fault attacks on 32-bit microcontrollers

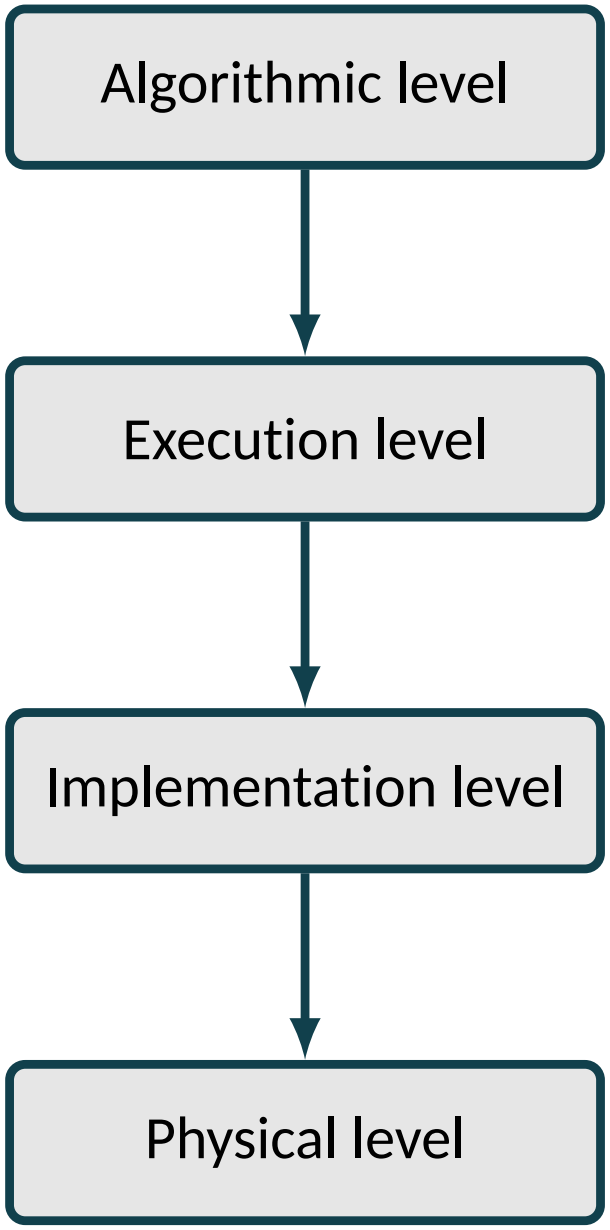
Definition: fault attacks

A fault attack consists in **disturbing the operating conditions** of a device to gain **privileged access** or **knowledge about the secret data** it handles.

Fault injection techniques

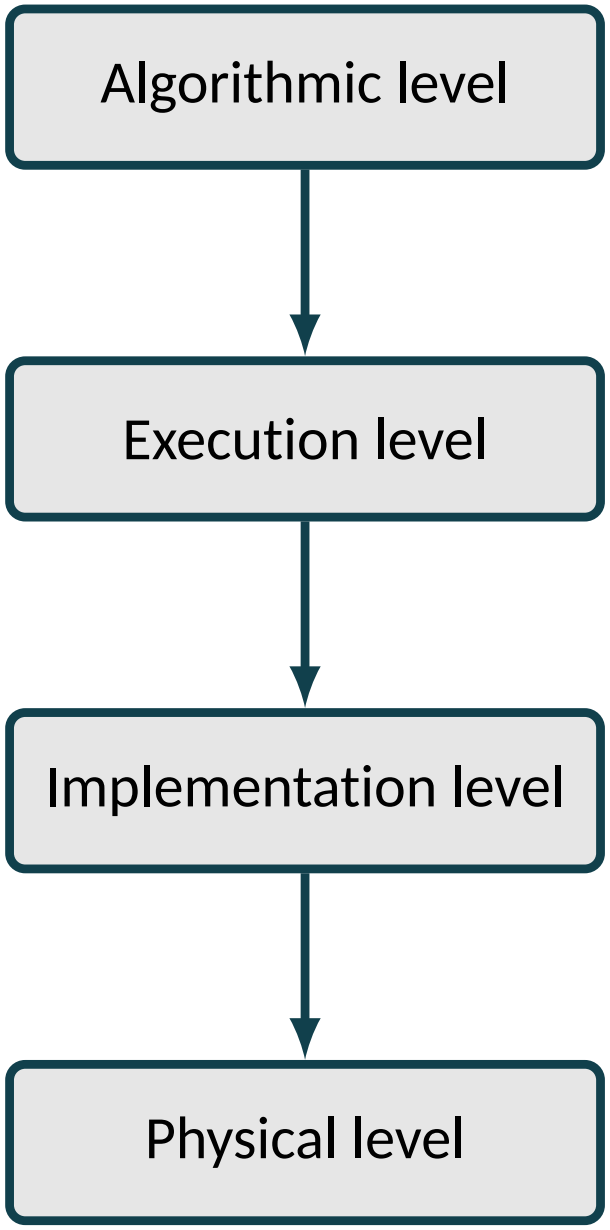
- Global
  - Clock glitches,
  - Supply voltage glitches,
  - Underpowering,
  - ...
- Local
  - Electromagnetic,
  - **Optical**,
  - ...


Algorithmic level
Execution level
Implementation level
Physical level



8-bit understanding:

- attacks on cryptographic algorithms,
- register corruption and instruction skip,
- timing constraints violation.

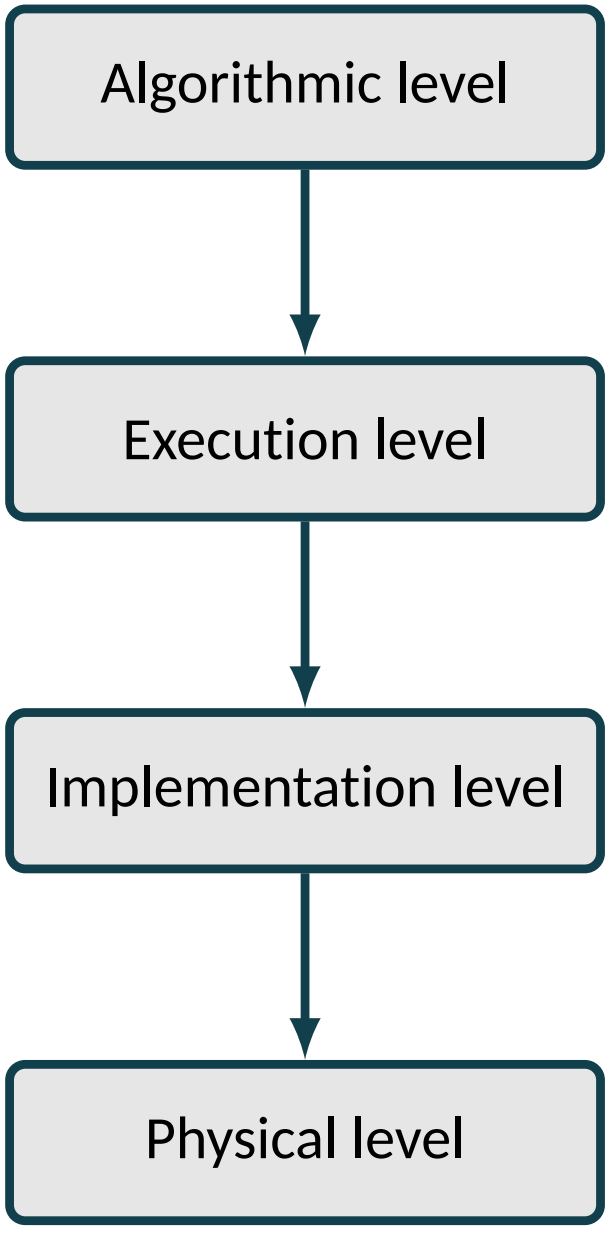


8-bit understanding:

- attacks on cryptographic algorithms,
- register corruption and instruction skip,
- timing constraints violation.

32-bit understanding:

- **Currently**: mostly algorithmic and execution level.



8-bit understanding:

- attacks on cryptographic algorithms,
- register corruption and instruction skip,
- timing constraints violation.

32-bit understanding:

- **Currently**: mostly algorithmic and execution level.

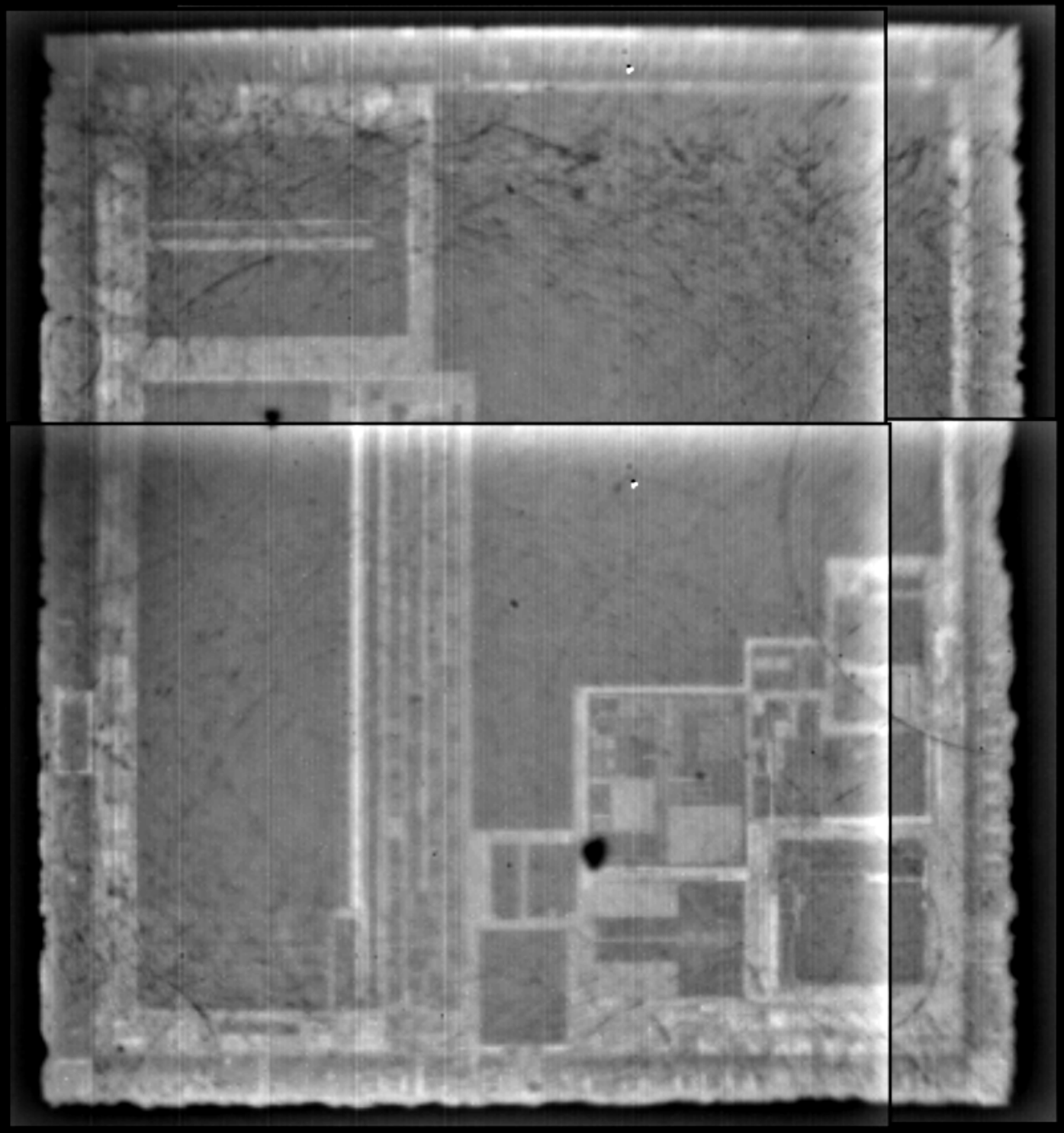
32-bit challenges

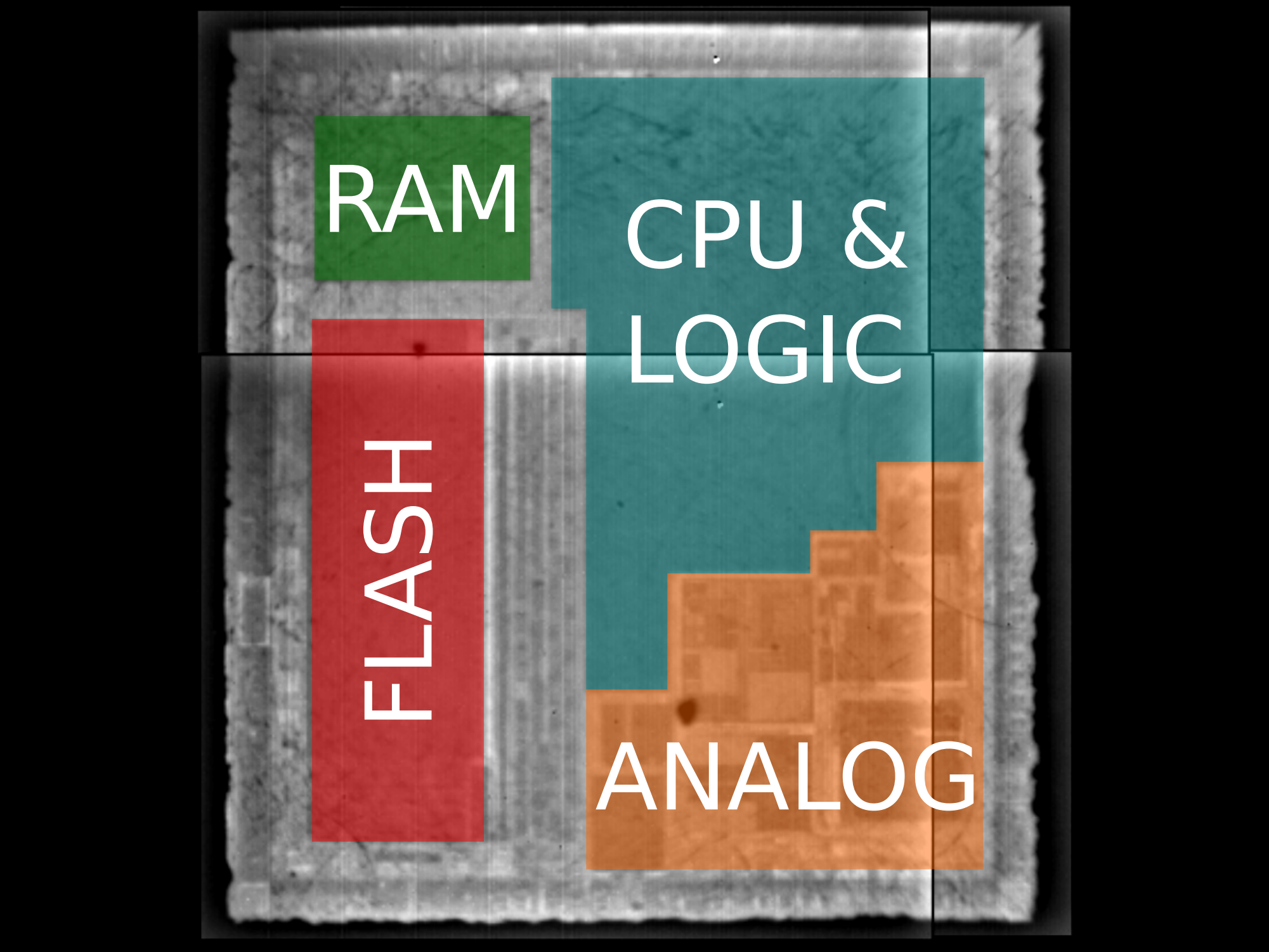
- **Bigger**, more **complex** chips,
- **Micro-architecture**: pipeline, pre-fetch...
- Execution timing **variability**.

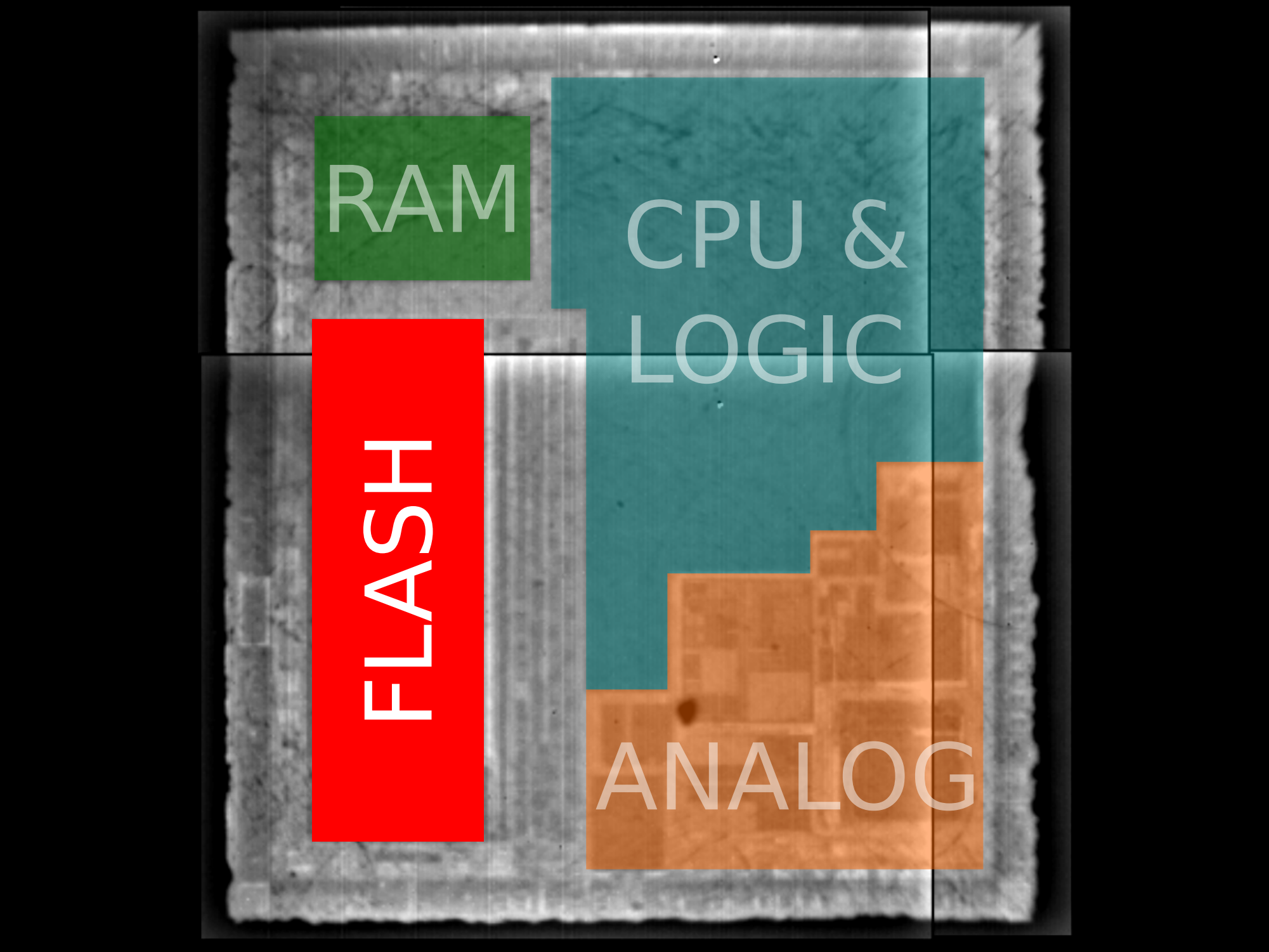
Experimental setup and preparatory work

A **32-bit** microcontroller:

- 2.5 x 2.5 mm chip,
- ARM Cortex-M3 core,
- 90 nm technology node,
- 128 kB of Flash memory.

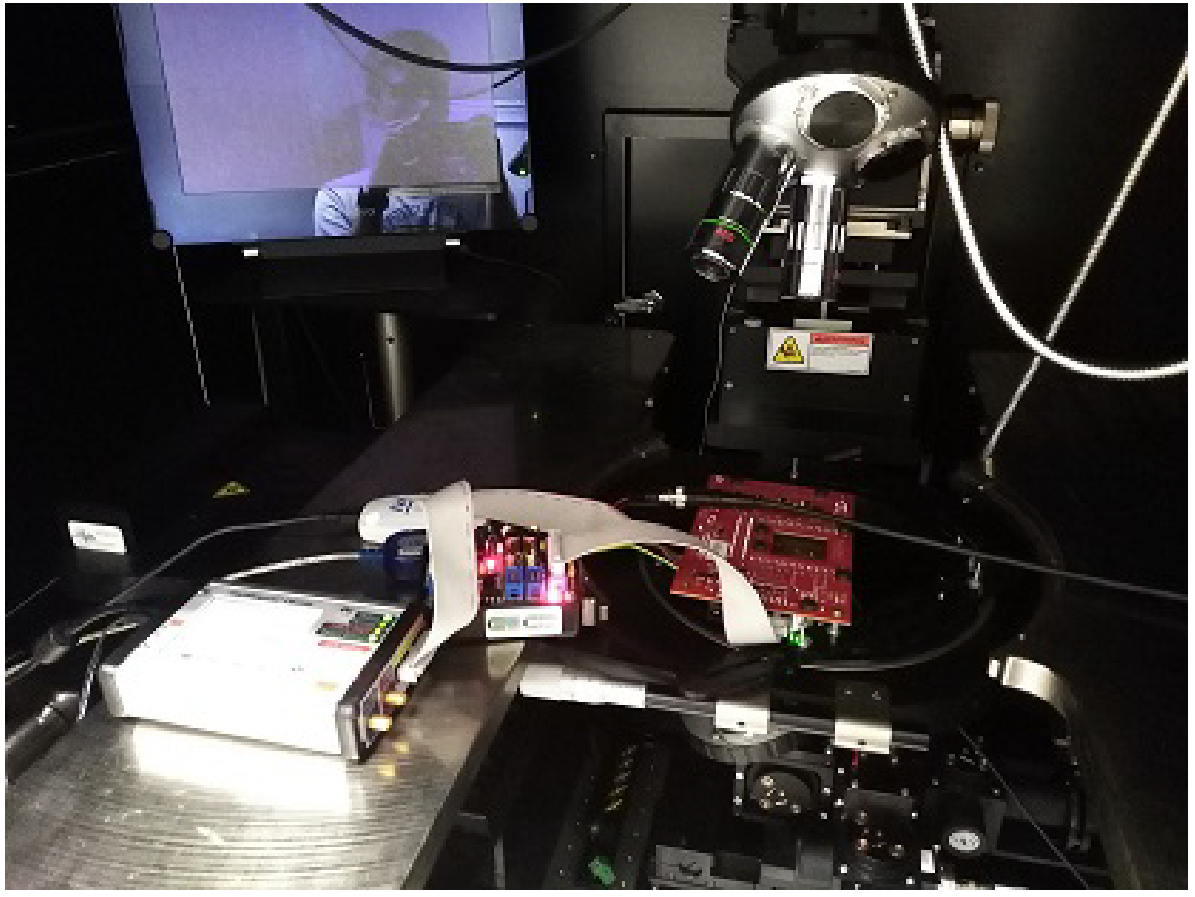



RAM
CPU &
LOGIC
FLASH
ANALOG


RAM
CPU &
LOGIC
FLASH
ANALOG

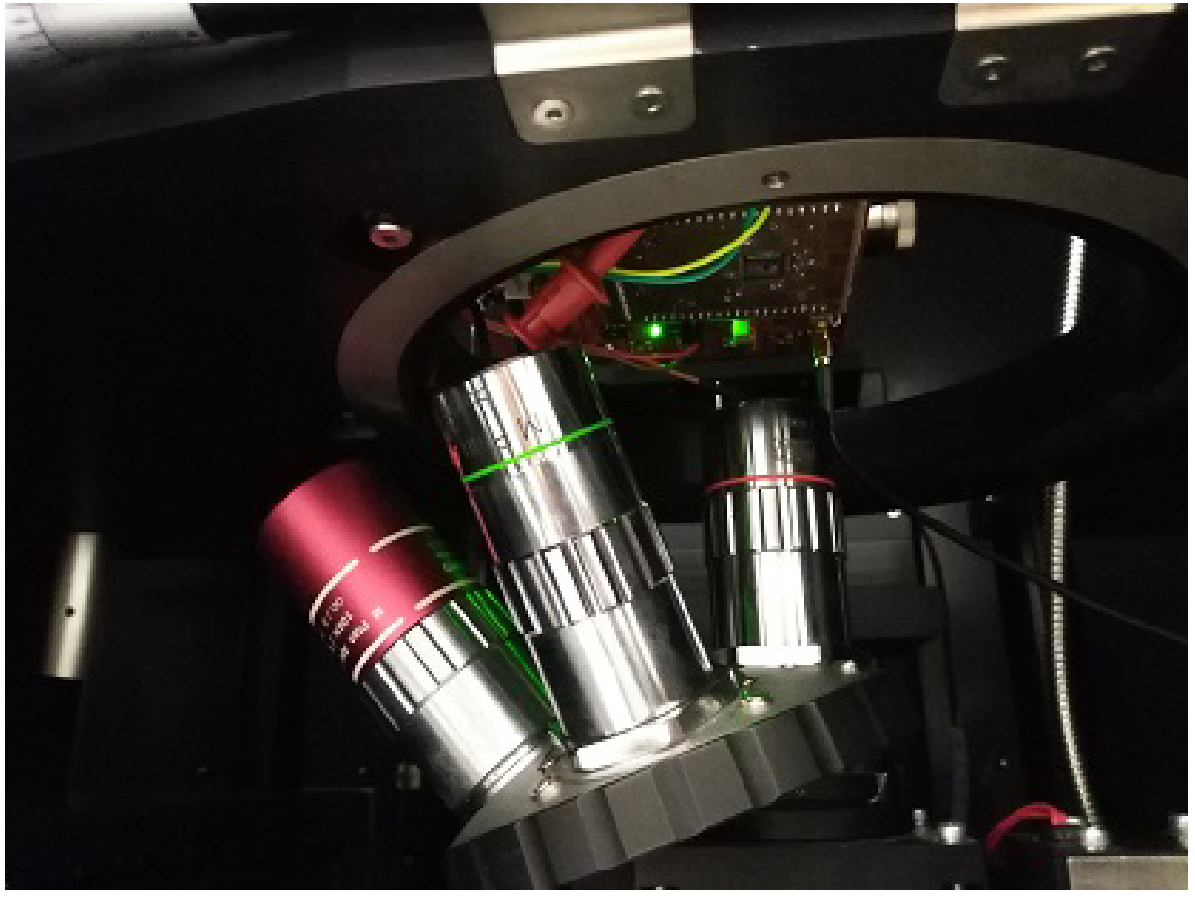
Laser bench characteristics

- Infrared (1064 nm) for **back-side** injection,
- >30 ps,
- 0-3 W,
- 3 objective lenses:
  - x5 (20 µm),
  - **x20 (5 µm),**
  - x100 (1 µm).



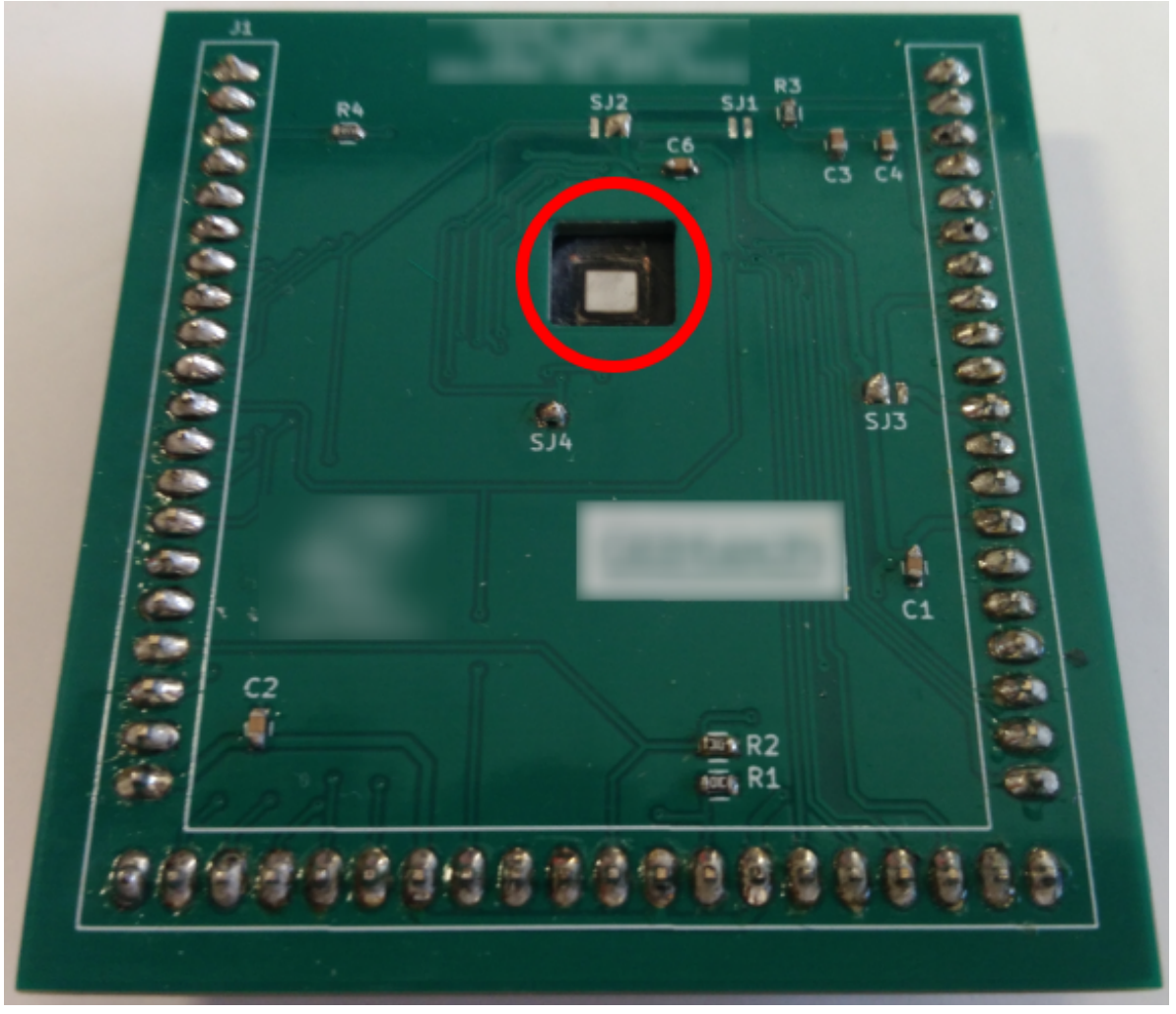
Laser bench characteristics

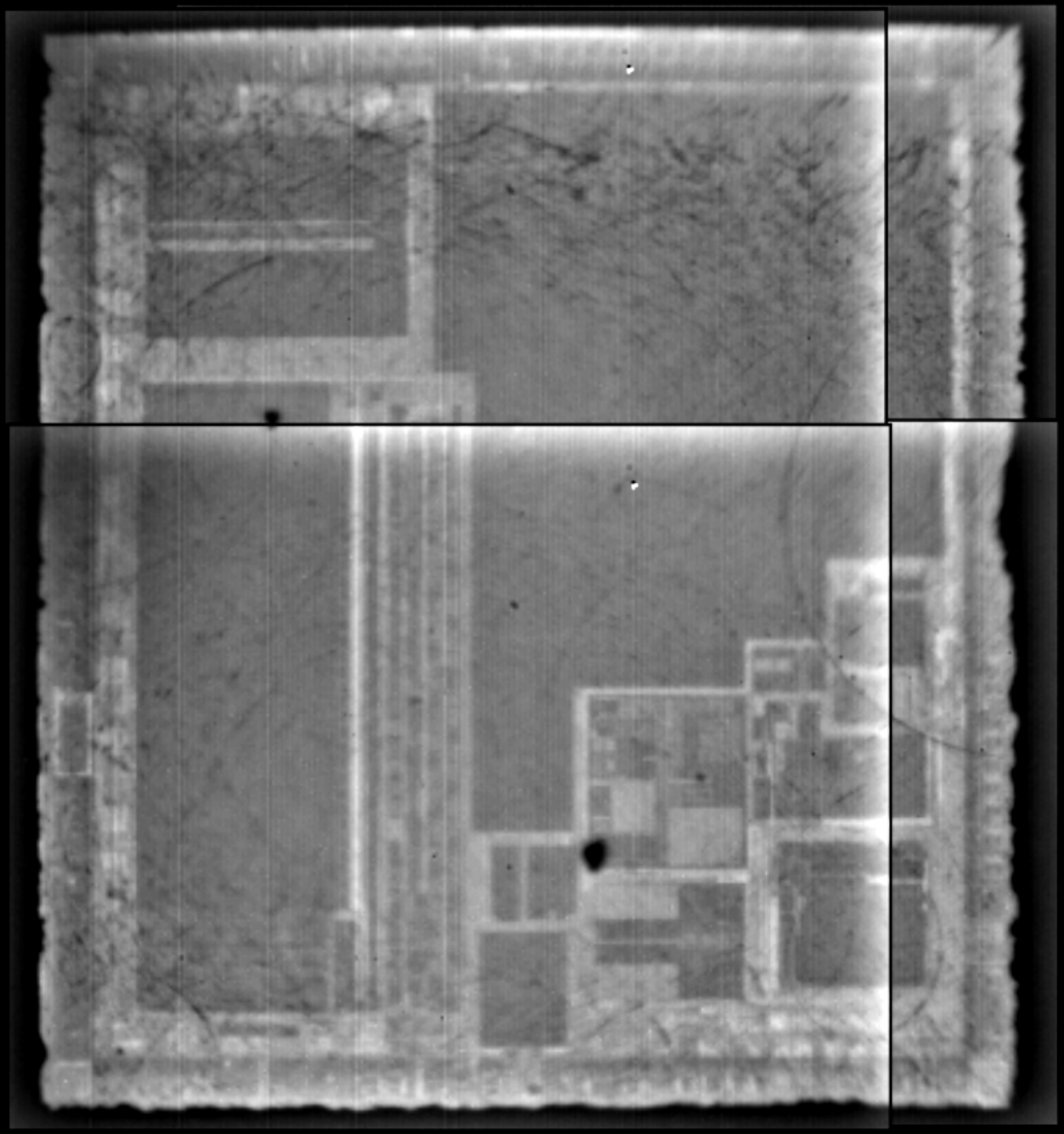
- Infrared (1064 nm) for **back-side** injection,
- >30 ps,
- 0-3 W,
- 3 objective lenses:
  - x5 (20 µm),
  - **x20 (5 µm),**
  - x100 (1 µm).

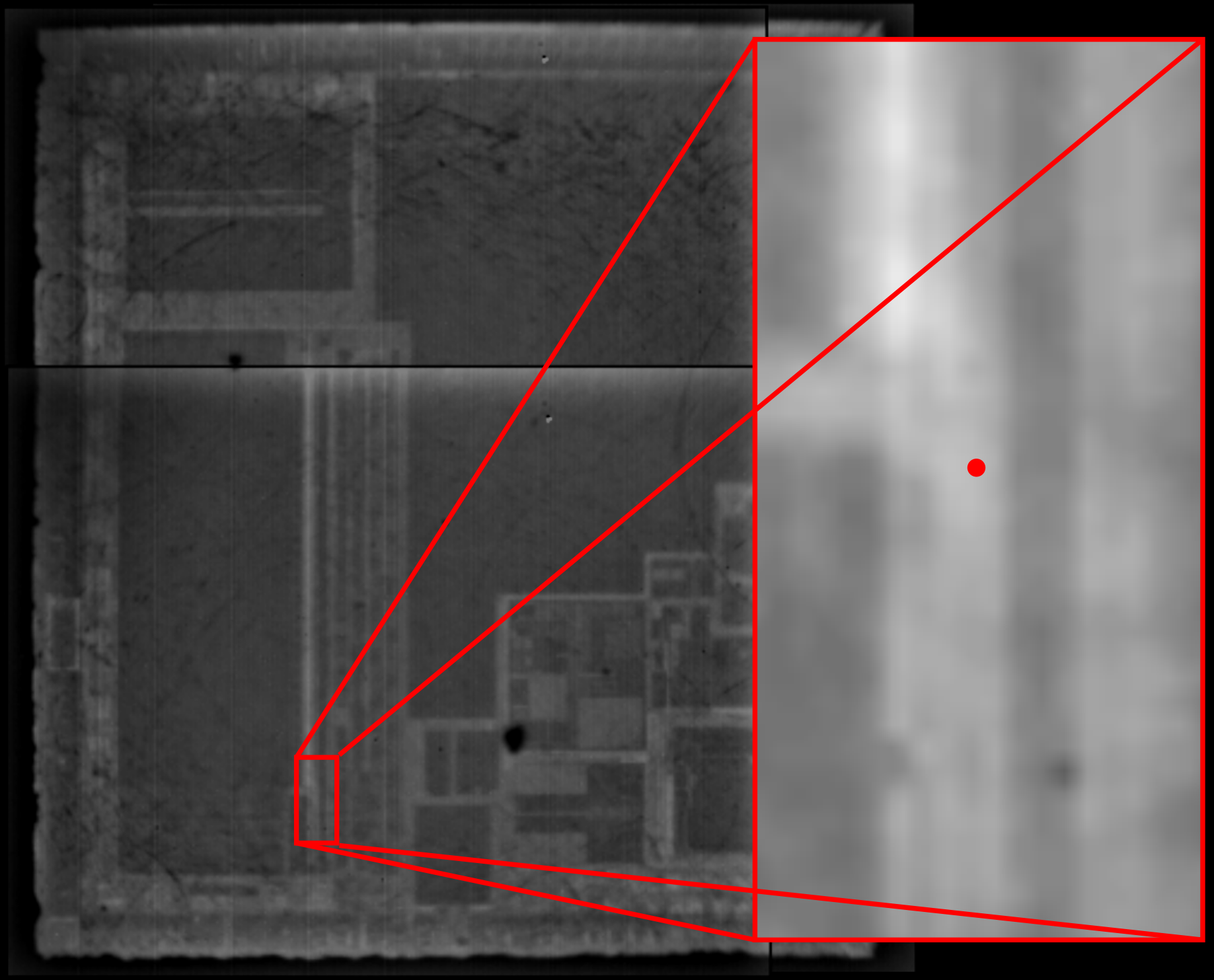


Laser bench characteristics

- Infrared (1064 nm) for **back-side** injection,
- >30 ps,
- 0-3 W,
- 3 objective lenses:
  - x5 (20 µm),
  - **x20 (5 µm),**
  - x100 (1 µm).





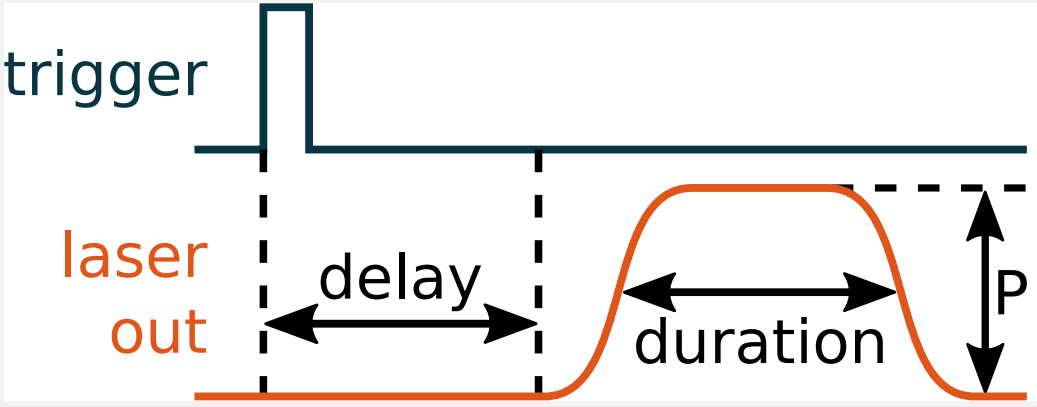


Preparatory work (4-5 months)

- ✔ Design of a **custom** ChipWhisperer target board:
  - ✔ **Front-side** access,
  - ✔ **Back-side** access.
- ✔ Target **preparation**: **decapsulate** the chip to see the die,
- ✔ **Mechanical setup** on the laser injection bench,
- ✔ Faults **mapping**:
  - ✔ x-position,
  - ✔ y-position,
  - ✔ power,
  - ✔ duration,
  - ✔ delay,
  - ✔ **type** of fault: instruction skip, bit-set, bit-reset, bit-flip...

Preparatory work (4-5 months)

- ✔ Design of a **custom** ChipWhisperer target board:
  - ✔ **Front-side** access,
  - ✔ **Back-side** access.
- ✔ Target **preparation**: **decapsulate** the chip to see the die,
- ✔ **Mechanical setup** on the laser injection bench,
- ✔ Faults **mapping**:
  - ✔ x-position,
  - ✔ y-position,
  - ✔ power,
  - ✔ duration,
  - ✔ delay,
  - ✔ **type** of fault: instruction skip, bit-set, bit-reset, bit-flip...



Characterisation results

```
1  test_data:
2  .word 0x00000000
3  NOP
4  NOP
5  NOP
6  NOP
7  NOP
8  NOP
9  LDR R0, test_data
10 NOP
11 NOP
12 NOP
13 NOP
14 NOP
15 NOP
16 # Reading back R0
```

- Write a **test data** at a specific address in Flash memory,
- **Store** this value in a **known register**,
- **Read back** the register.

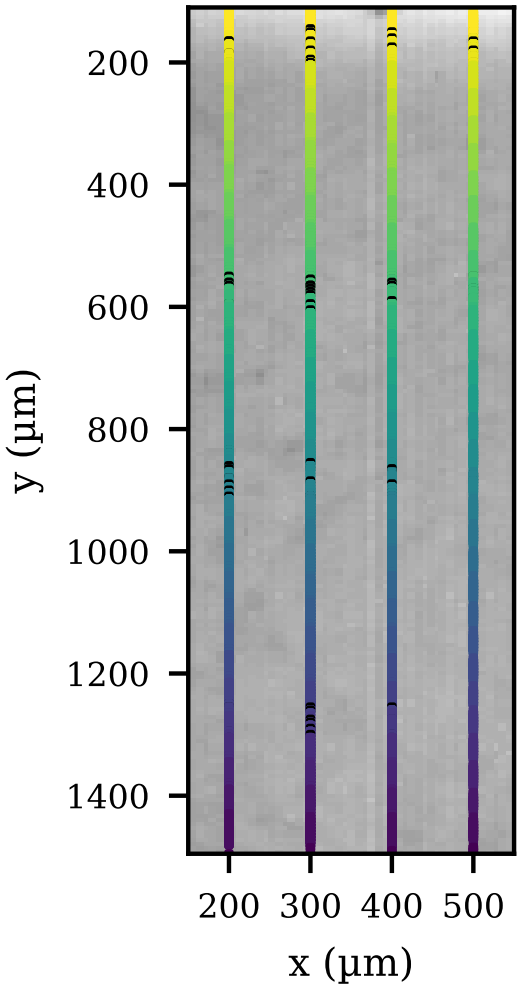
```
1  test_data:
2  .word 0x00000000
3  NOP
4  NOP
5  NOP
6  NOP
7  NOP
8  NOP
9  LDR R0, test_data
10 NOP
11 NOP
12 NOP
13 NOP
14 NOP
15 NOP
16 # Reading back R0
```

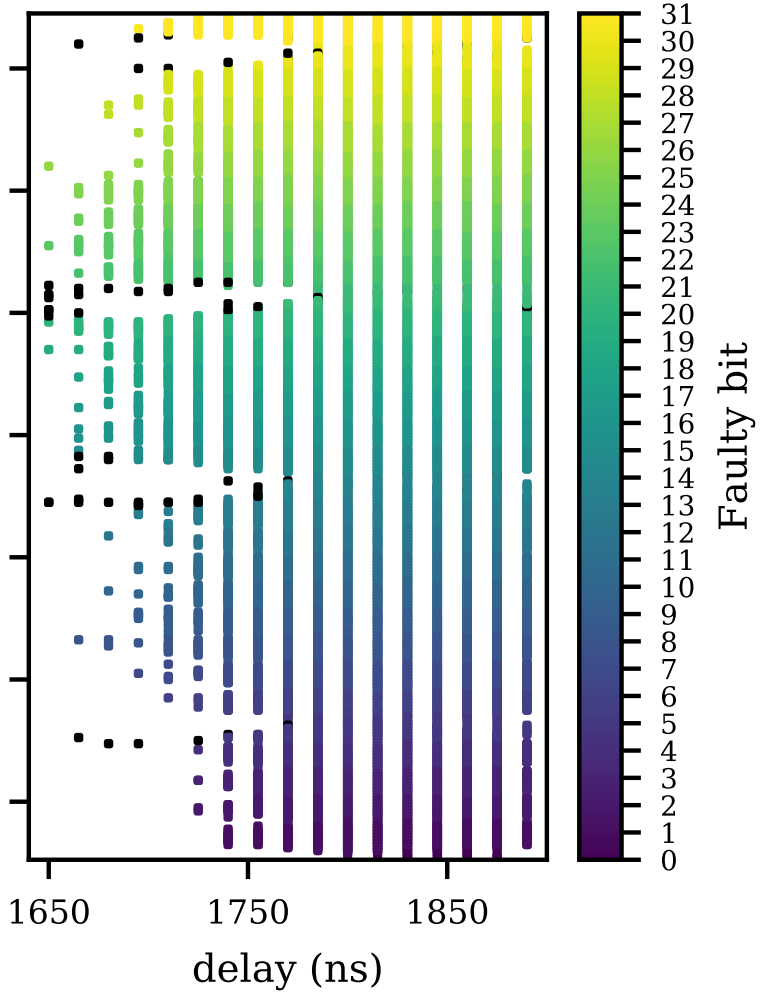
- Write a **test data** at a specific address in Flash memory,
- **Store** this value in a **known register**,
- **Read back** the register.

Choice of test data

- `0x00000000`: bit-sets,
- `0xFFFFFFFF`: bit-resets,
- `0x55555555`
  `0xAAAAAAAA`: bit-flips.

Faulty bit wrt. x/y position and delay





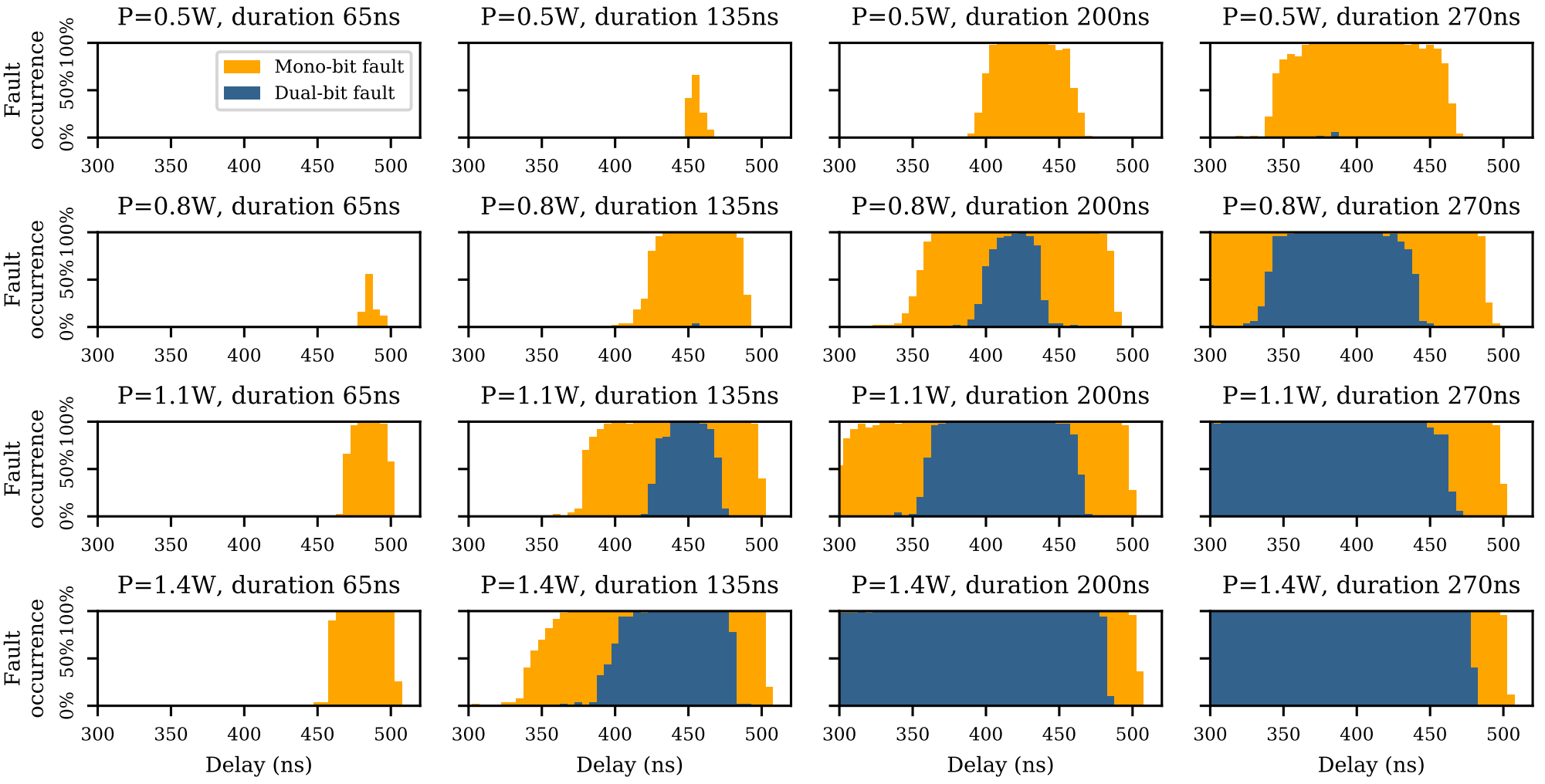
Fault model

Monobit-set on fetched data.

Parameters dependency

Faulty bit depends on **y** position.

Faulty bit wrt. power and duration



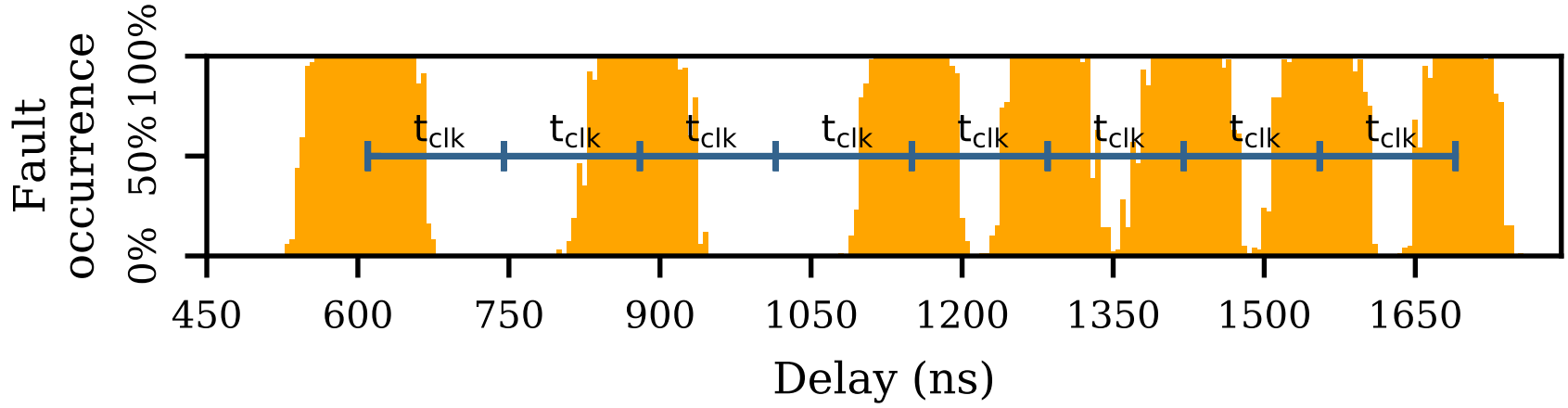
Observation

Increasing the **energy** allows to fault more bits.

Characterisation code 2

```
1  # Initialising registers
2  # R0, R1, R4, R5, R6, R8
3  # and R9 to 0xFFFFFFFF
4  NOP
5  NOP
6  MOVW R0, 0x0000 ←
7  MOVW R1, 0x0000 ←
8  MOVW R4, 0x0000 ←
9  MOVW R5, 0x0000 ←
10 MOVW R6, 0x0000 ←
11 MOVW R8, 0x0000 ←
12 MOVW R9, 0x0000 ←
13 NOP
14 NOP
15 # Reading back the registers
```

Fault periodicity (faulty bit wrt. delay)



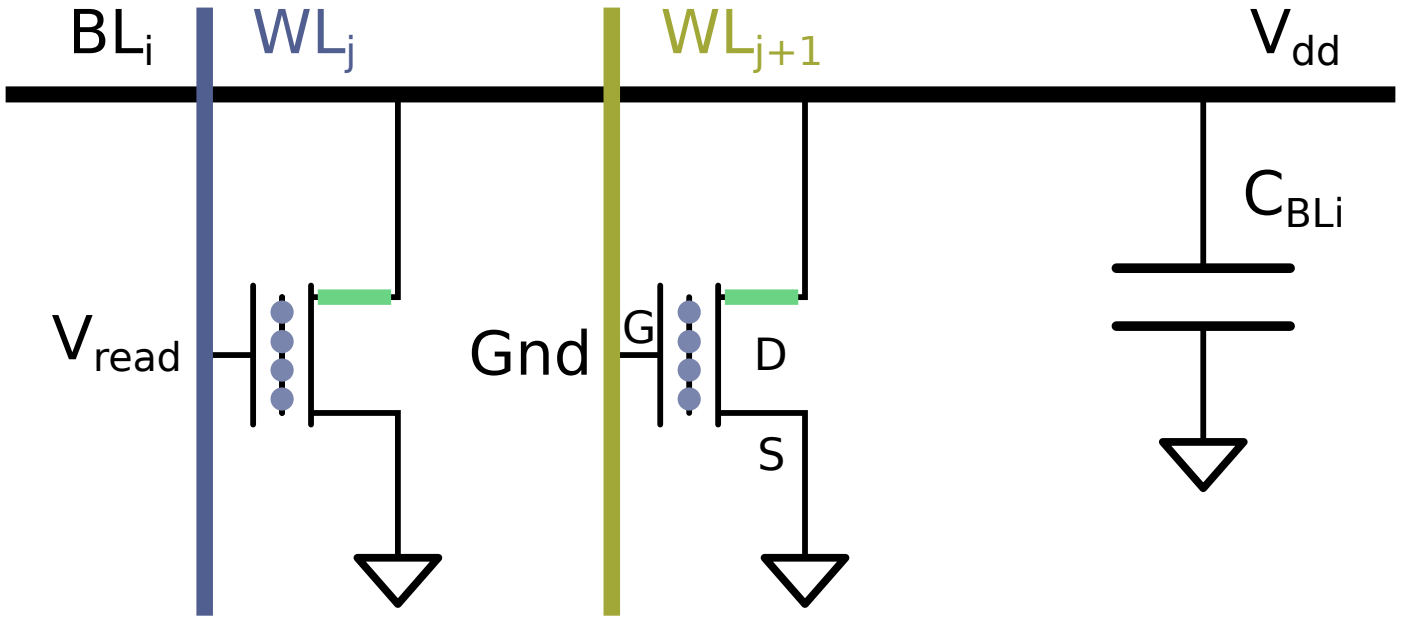

```
1  MOVW R0, 0x0000
2  MOVW R1, 0x0000
3  MOVW R4, 0x0000
4  MOVW R5, 0x0000
5  MOVW R6, 0x0000
6  MOVW R8, 0x0000
7  MOVW R9, 0x0000
```

Observations

- **Each** instruction can be faulty,
- The occurence **always** reaches **100%**,
- The delay between two optimal injection timings is always a **multiple of the clock period**
- The delay between two optimal injection timings is **not constant**.

Physical explanation

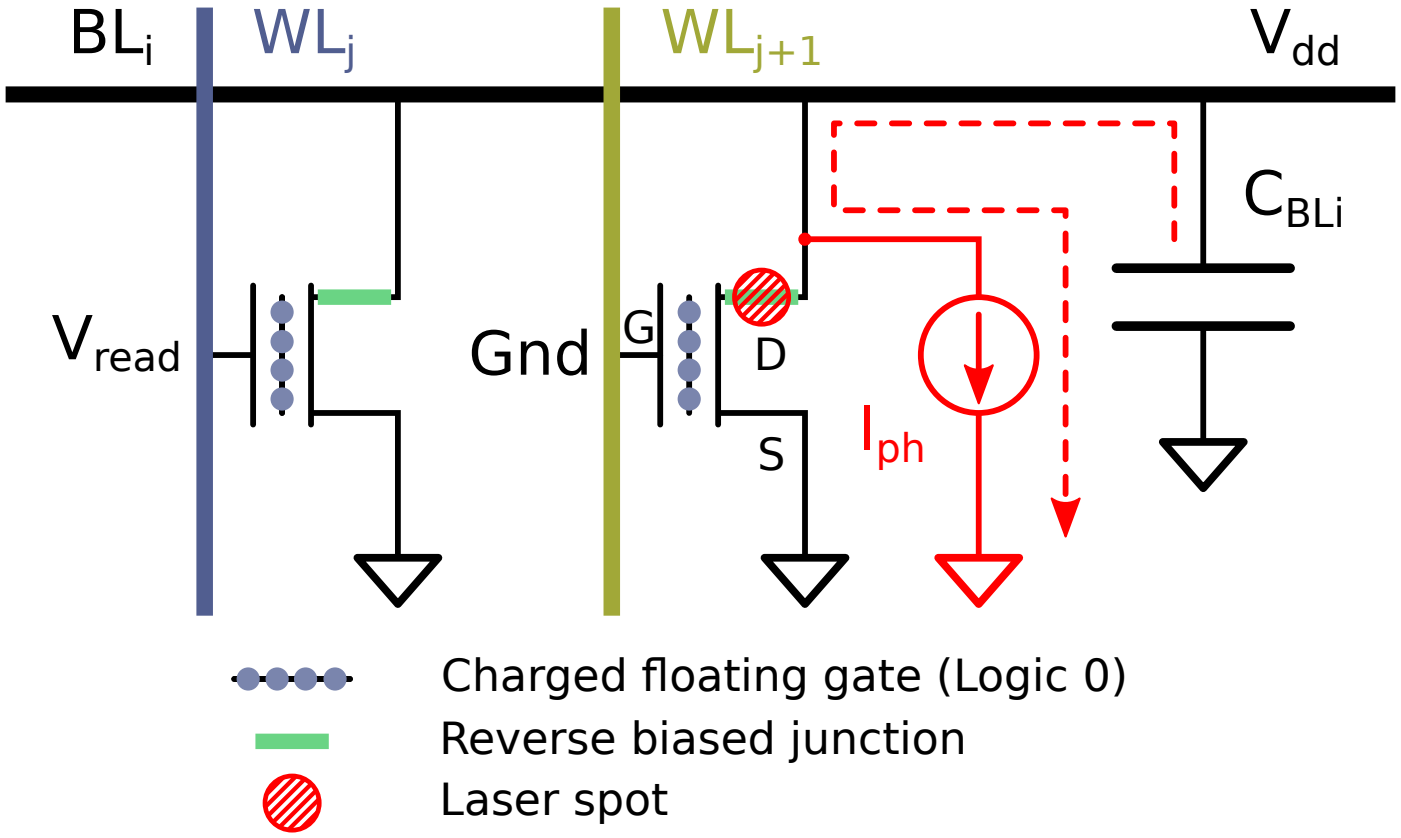
Physical explanation for the y-dependency



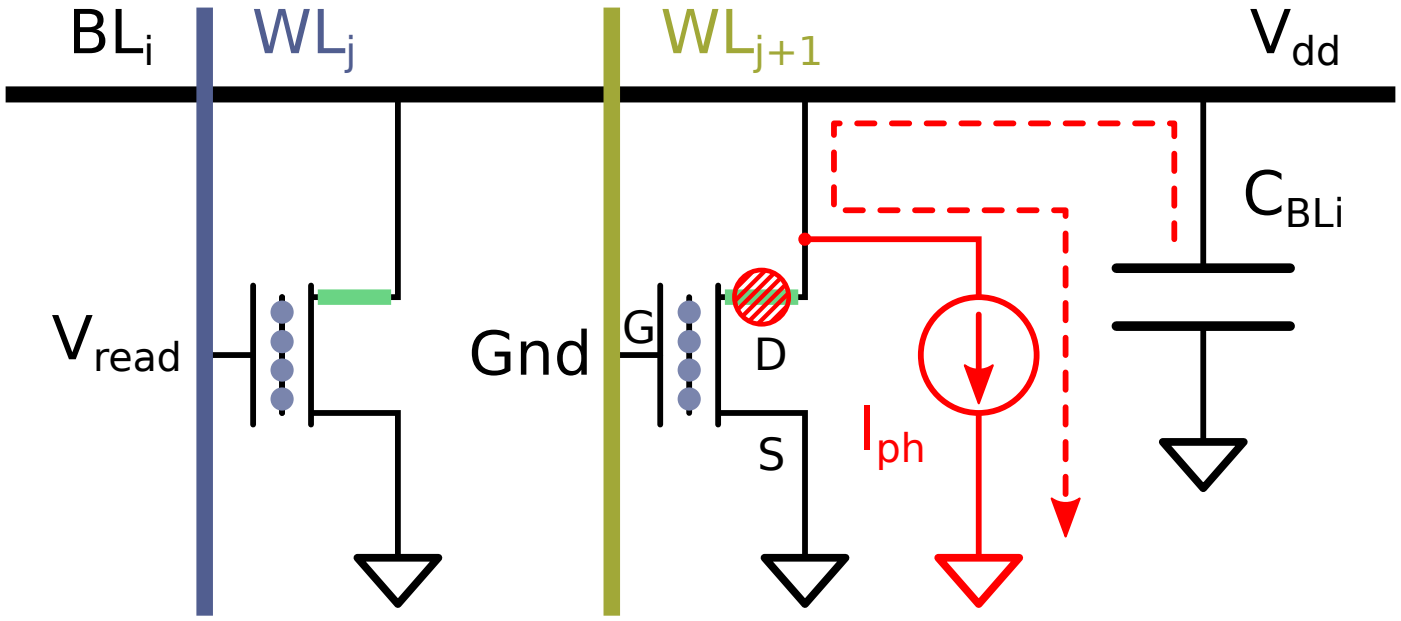
Charged floating gate (Logic 0)

Reverse biased junction

Physical explanation for the y-dependency



Physical explanation for the y-dependency



Charged floating gate (Logic 0)

Reverse biased junction

Laser spot

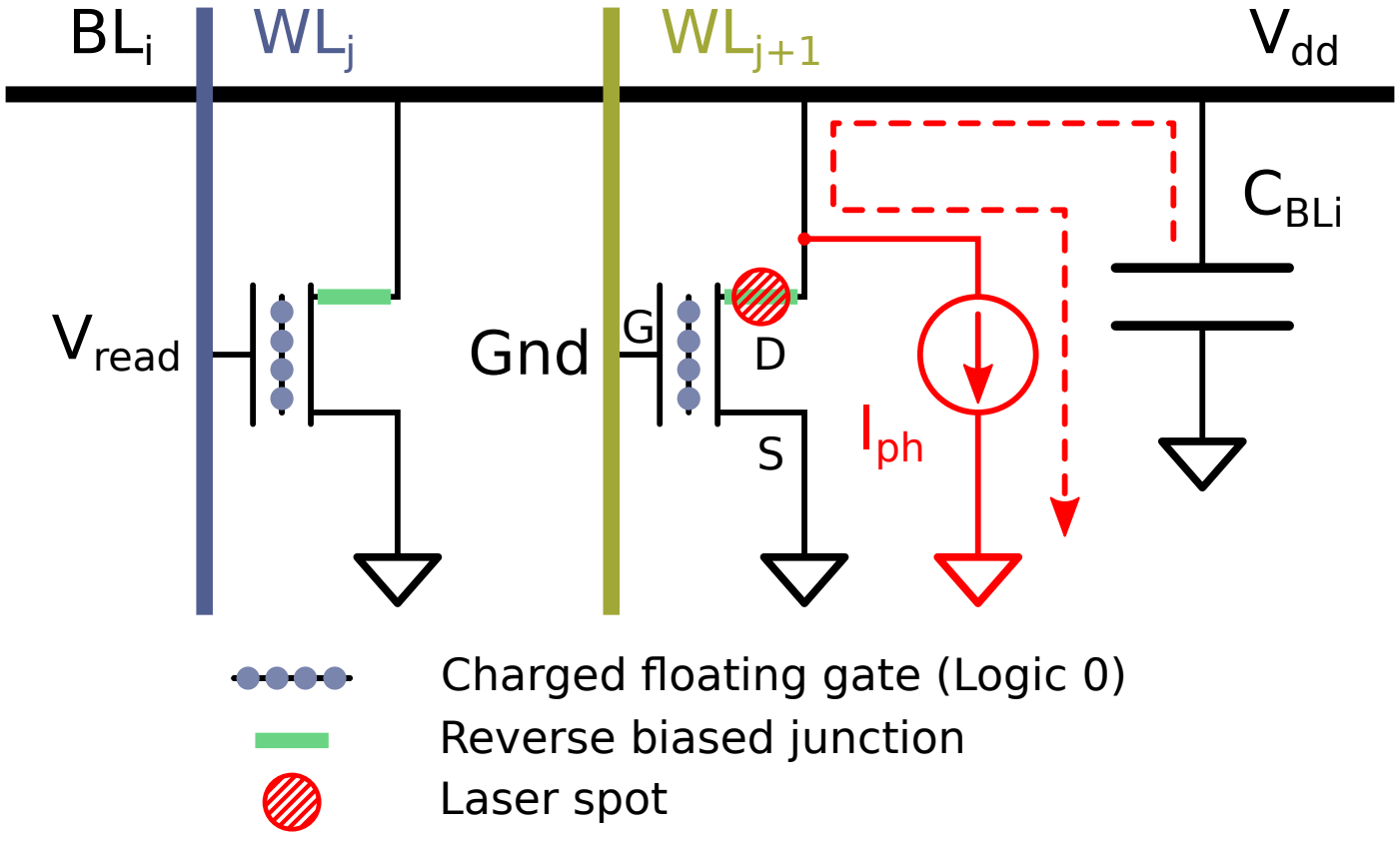
Moving along the x-axis

- Transistors of the same **BL**.
- **Same** faulty bit.

Moving along the y-axis

- Transistors of the same **WL**.
- **Successive** faulty bits.

Physical explanation for the asymmetry



Without laser shot

- **with** charges: BL to V_{dd}
- **without** charges: BL to **GND**

With laser shot

- **with** charges: BL to **GND**
- **without** charges: BL to **GND**

Applications

Example of instruction corruption: 32-bit `MOVW`

`MOVW`: store a 16-bit value in the lower half of a 32-bit register.

bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	1 2	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions:

`MOVW`	1	1	1	1	0	i	1	0	0	1	0	0	imm4				0	imm3			Rd				imm8							
`MOVW, R0, 0`	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Example of instruction corruption: 32-bit `MOVW`

`MOVW`: store a 16-bit value in the lower half of a 32-bit register.

bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	1 2	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions:

`MOVW`	1	1	1	1	0	i	1	0	0	1	0	0	imm4				0	imm3			Rd				imm8							
`MOVW, R0, 0`	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Data corruption:

`MOVW, R0, 4`	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0

Example of instruction corruption: 32-bit MOVW

MOVW: store a 16-bit value in the lower half of a 32-bit register.

bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	1 2	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions:

MOVW	1	1	1	1	0	i	1	0	0	1	0	0	imm4				0	imm3			Rd				imm8							
MOVW, R0, 0	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Data corruption:

MOVW, R0, 4	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0

Register corruption:

MOVW, R1, 0	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0

Example of instruction corruption: 32-bit MOVW

MOVW: store a 16-bit value in the lower half of a 32-bit register.

bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	1 2	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions:

MOVW	1	1	1	1	0	i	1	0	0	1	0	0	imm4				0	imm3			Rd				imm8							
MOVW, R0, 0	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Data corruption:

MOVW, R0, 4	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0

Register corruption:

MOVW, R1, 0	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0

Opcode corruption:

MOVT, R0, 0	1	1	1	1	0	0	1	0	**1**	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PIN verification algorithm

Constant-time implementation with **hardened booleans**:
No **simple** side-channel attack and `TRUE=0x5555`, `FALSE=0xAAAA`.

```
 1: trials = 3
 2: ref_PIN[4] = {1, 2, 3, 4}
 3: procedure VerifyPIN(user_PIN[4])
 4:     authenticated = FALSE
 5:     diff = FALSE
 6:     dummy = TRUE
 7:     if trials > 0 then
 8:         for i ← 0 to 3 do
 9:             if user_PIN[i] != ref_PIN[i] then
10:                 diff = TRUE
11:             else
12:                 dummy = FALSE
13:             end if
14:         end for
15:         if diff == TRUE then
16:             trials = trials - 1
17:         else
18:             authenticated = TRUE
19:         end if
20:     end if
21:     return authenticated
22: end procedure
```

PIN verification algorithm

Constant-time implementation with **hardened booleans**:
No **simple** side-channel attack and `TRUE=0x5555`, `FALSE=0xAAAA`.

```
 1: trials = 3
 2: ref_PIN[4] = {1, 2, 3, 4}
 3: procedure VerifyPIN(user_PIN[4])
 4:     authenticated = FALSE
 5:     diff = FALSE
 6:     dummy = TRUE
 7:     if trials > 0 then
 8:         for i ← 0 to 3 do
 9:             if user_PIN[i] != ref_PIN[i] then
10:                 diff = TRUE
11:             else
12:                 dummy = FALSE
13:             end if
14:         end for
15:         if diff == TRUE then
16:             trials = trials - 1
17:         else
18:             authenticated = TRUE
19:         end if
20:     end if
21:     return authenticated
22: end procedure
```

```
if (trials > 0)
{
   ...
}
```

```
CMP R3, 0
BLE address
```

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic CMP	0	0	1	0	1	Rd			imm8							
CMP R3, 0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic CMP	0	0	1	0	1	Rd			imm8							
CMP R3, 0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0

Perform a bit-set on the **10th** bit of the instruction: R3 ➔ R7.
By design, R7 stores the *frame-pointer*, **always positive**.

Register corruption ⬇

CMP R7, 0	0	0	1	0	1	**1**	1	1	0	0	0	0	0	0	0	0

Fault injection on the `CMP` instruction

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic `CMP`	0	0	1	0	1	Rd			imm8							
`CMP R3, 0`	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0

Perform a bit-set on the **10th** bit of the instruction: `R3` ➔ `R7`.
By design, `R7` stores the *frame-pointer*, **always positive**.

Register corruption ↓

`CMP R7, 0`	0	0	1	0	1	**1**	1	1	0	0	0	0	0	0	0	0

Outcome

trials is **never compared** ➔ **unlimited** number of trials.

1: **procedure** AddRoundKey
2: **for** i $\leftarrow$ 0 to 3 **do**
3: **for** j $\leftarrow$ 0 to 3 **do**
4: $S_{i,j} = S_{i,j} \oplus K_{i,j}^{10}$
5: **end for**
6: **end for**
7: **end procedure**

1: **procedure** AddRoundKey
2: **for** i $\leftarrow$ 0 to 3 **do**
3: **for** j $\leftarrow$ 0 to 3 **do**
4: $S_{i,j} = S_{i,j} \oplus K_{i,j}^{10}$
5: **end for**
6: **end for**
7: **end procedure**

```
for (int i=0; i<4; i++)
{
  for (int j=0; j<4; j++)
  {
    ...
  }
}
```

```
MOV R0, 0
addr_i:
MOV R1, 0
addr_j:
...
ADD R1, 1
CMP R1, 3
BLE addr_j
ADD R0, 1
CMP R0, 3
BLE addr_i
```

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic ADD	0	0	1	1	0	Rd			imm8							
ADD R0, 1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1

Fault injection on the `ADD` instruction

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic `ADD`	0	0	1	1	0	Rd			imm8							
`ADD R0, 1`	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1

Perform a bit-set on the **2nd** bit of the instruction.
Add **5** instead of **1** to the **loop variable**.

Data corruption

`ADD R0, 5`	0	0	1	1	0	0	0	0	0	0	0	0	0	**1**	0	1

Fault injection on the `ADD` instruction

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic `ADD`	0	0	1	1	0	Rd			imm8							
`ADD R0, 1`	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1

Perform a bit-set on the **2nd** bit of the instruction.
Add **5** instead of **1** to the **loop variable**.

Data corruption

`ADD R0, 5`	0	0	1	1	0	0	0	0	0	0	0	0	0	**1**	0	1

Outcome

For loop exit after **one** execution only.

Fault on the *for* loops

Faulty ciphertext byte: $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$

Fault on the **inner** for loop on its **first** execution.

$C_{0,0}$	$C_{1,0}$	$C_{2,0}$	$C_{3,0}$
$\tilde{C}_{0,1}$	$C_{1,1}$	$C_{2,1}$	$C_{3,1}$
$\tilde{C}_{0,2}$	$C_{1,2}$	$C_{2,2}$	$C_{3,2}$
$\tilde{C}_{0,3}$	$C_{1,3}$	$C_{2,3}$	$C_{3,3}$

Fault on the *for* loops

Faulty ciphertext byte: $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$

Fault on the **inner** for loop on its **first** execution.

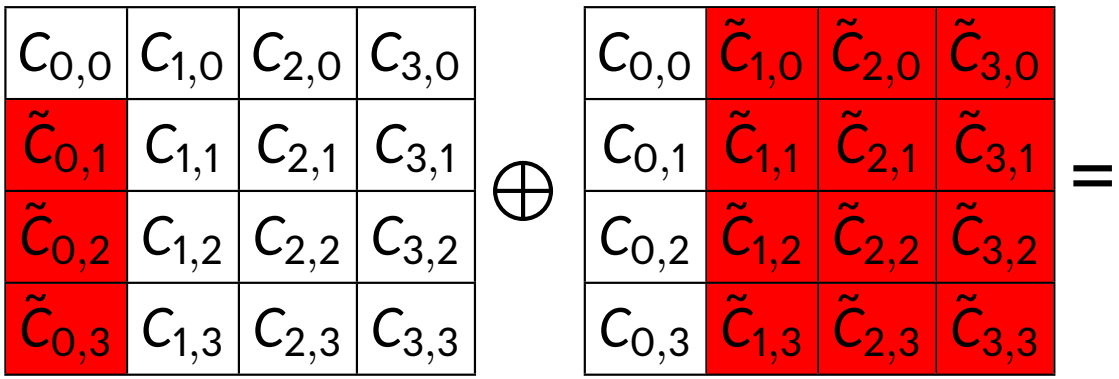
$C_{0,0}$	$C_{1,0}$	$C_{2,0}$	$C_{3,0}$
$\tilde{C}_{0,1}$	$C_{1,1}$	$C_{2,1}$	$C_{3,1}$
$\tilde{C}_{0,2}$	$C_{1,2}$	$C_{2,2}$	$C_{3,2}$
$\tilde{C}_{0,3}$	$C_{1,3}$	$C_{2,3}$	$C_{3,3}$

Fault on the **outer** for loop.

$C_{0,0}$	$\tilde{C}_{1,0}$	$\tilde{C}_{2,0}$	$\tilde{C}_{3,0}$
$C_{0,1}$	$\tilde{C}_{1,1}$	$\tilde{C}_{2,1}$	$\tilde{C}_{3,1}$
$C_{0,2}$	$\tilde{C}_{1,2}$	$\tilde{C}_{2,2}$	$\tilde{C}_{3,2}$
$C_{0,3}$	$\tilde{C}_{1,3}$	$\tilde{C}_{2,3}$	$\tilde{C}_{3,3}$

10^{th} round-key recovery

Faulty ciphertext byte: $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$



10^{th} round-key recovery

Faulty ciphertext byte: $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$

$C_{0,0}$	$C_{1,0}$	$C_{2,0}$	$C_{3,0}$
$\tilde{C}_{0,1}$	$C_{1,1}$	$C_{2,1}$	$C_{3,1}$
$\tilde{C}_{0,2}$	$C_{1,2}$	$C_{2,2}$	$C_{3,2}$
$\tilde{C}_{0,3}$	$C_{1,3}$	$C_{2,3}$	$C_{3,3}$

$\oplus$

$C_{0,0}$	$\tilde{C}_{1,0}$	$\tilde{C}_{2,0}$	$\tilde{C}_{3,0}$
$C_{0,1}$	$\tilde{C}_{1,1}$	$\tilde{C}_{2,1}$	$\tilde{C}_{3,1}$
$C_{0,2}$	$\tilde{C}_{1,2}$	$\tilde{C}_{2,2}$	$\tilde{C}_{3,2}$
$C_{0,3}$	$\tilde{C}_{1,3}$	$\tilde{C}_{2,3}$	$\tilde{C}_{3,3}$

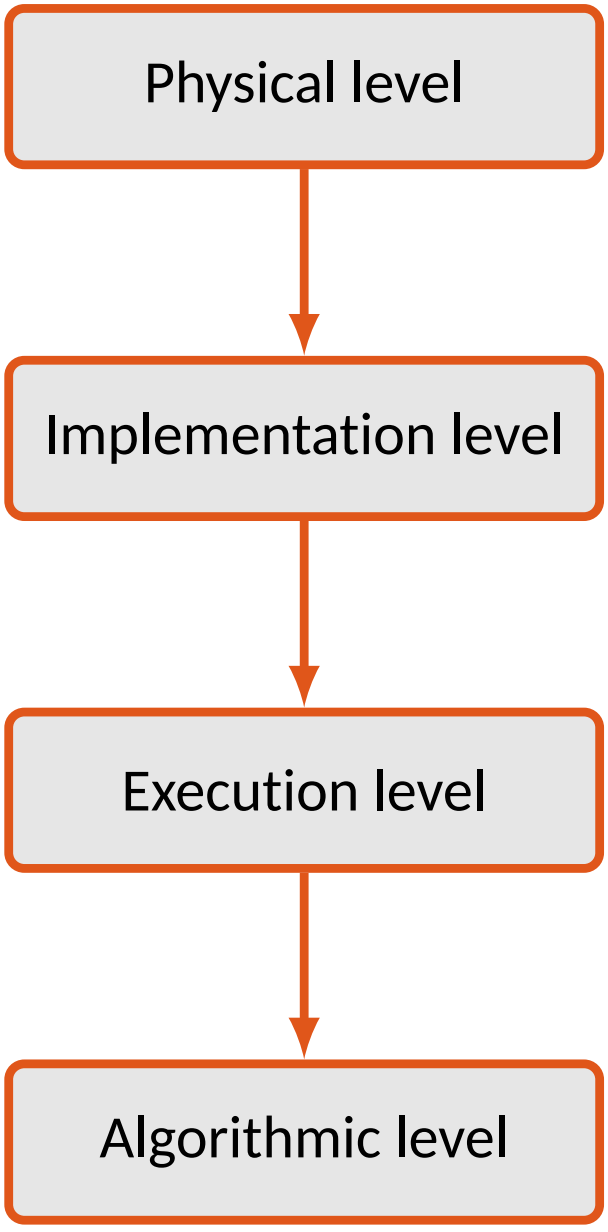
$=$

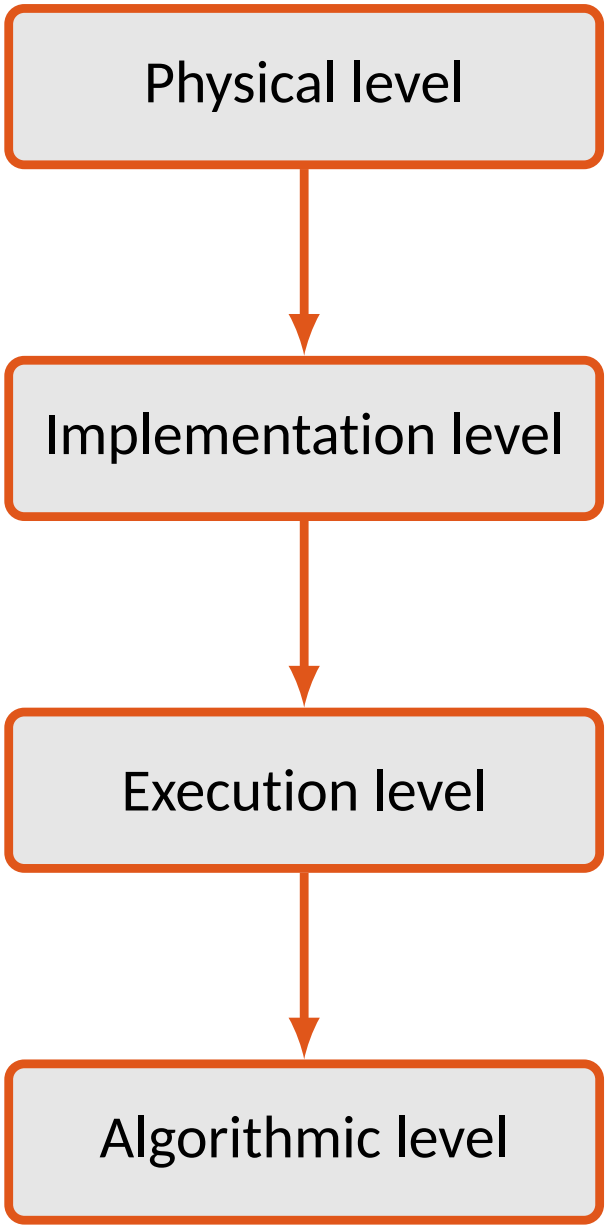
0	$K^{10}_{1,0}$	$K^{10}_{2,0}$	$K^{10}_{3,0}$
$K^{10}_{0,1}$	$K^{10}_{1,1}$	$K^{10}_{2,1}$	$K^{10}_{3,1}$
$K^{10}_{0,2}$	$K^{10}_{1,2}$	$K^{10}_{2,2}$	$K^{10}_{3,2}$
$K^{10}_{0,3}$	$K^{10}_{1,3}$	$K^{10}_{2,3}$	$K^{10}_{3,3}$

What then?

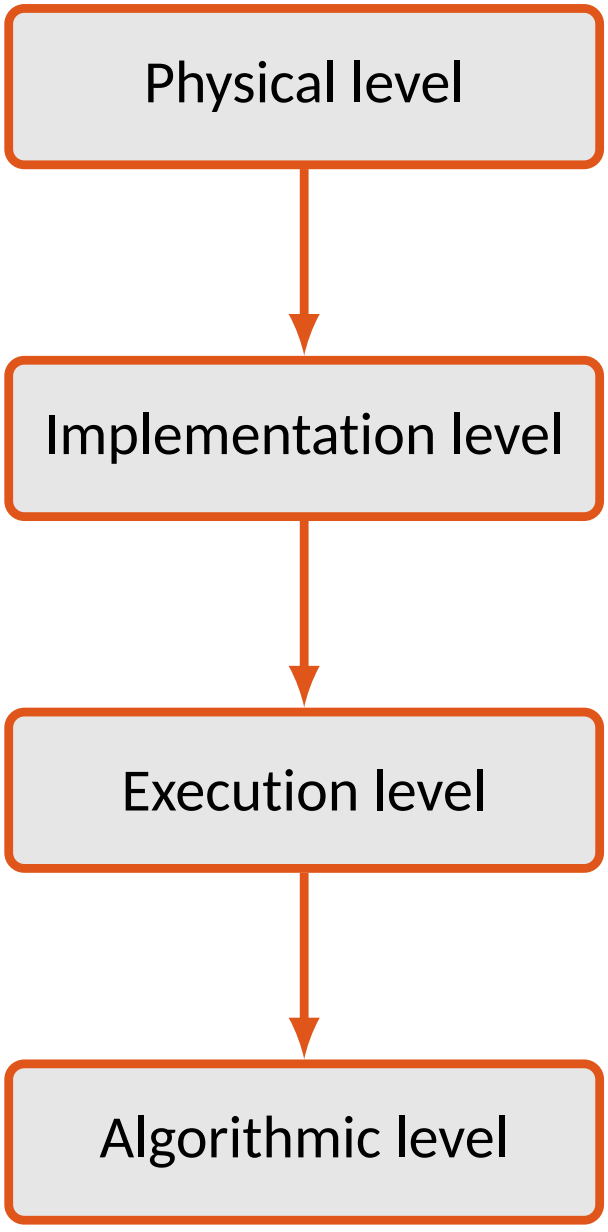
Only **one byte** of the 10^{th} round-key, must be **brute-forced**.

Conclusion


Physical level
Implementation level
Execution level
Algorithmic level

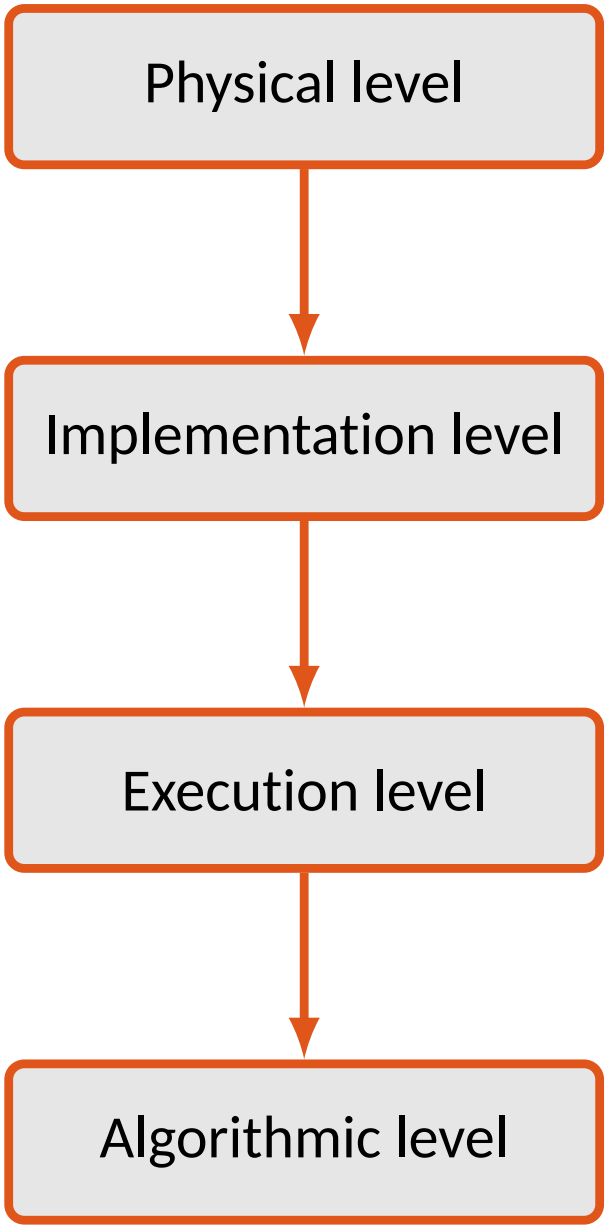


Force storage transistors to **conduct** in Flash memory.



Force storage transistors to **conduct** in Flash memory.

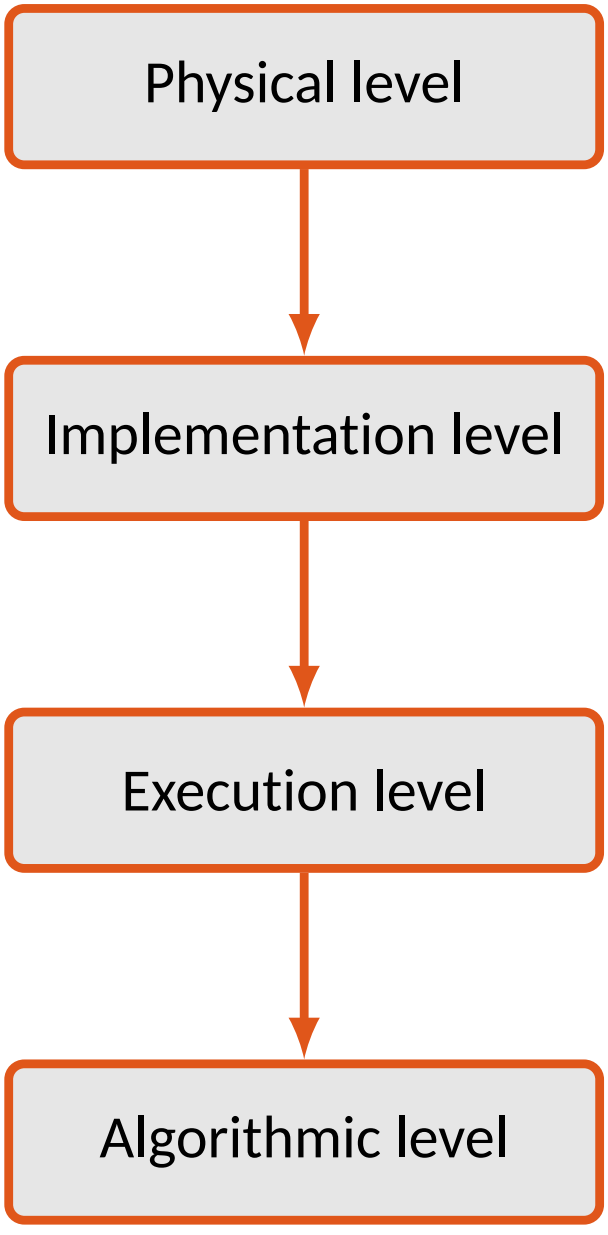
Perform a **bit-set** on a **chosen single** bit of the instruction/data.



Force storage transistors to **conduct** in Flash memory.

Perform a **bit-set** on a **chosen single** bit of the instruction/data.

Always take the first *if* branch.
Prematurely exit the *for* loops.



Force storage transistors to **conduct** in Flash memory.

Perform a **bit-set** on a **chosen single** bit of the instruction/data.

Always take the first *if* branch.
Prematurely exit the *for* loops.

Unlimited trials on the VerifyPIN.
AES last `AddRoundKey` alteration.

Possibilities

- **Bit-set** on Flash data,
- Security level **lowering**.

Limitations

- **Contiguous** bits only,
- **Control-flow** alteration mostly.

Possibilities

- **Bit-set** on Flash data,
- Security level **lowering**.

Limitations

- **Contiguous** bits only,
- **Control-flow** alteration mostly.

Perspectives:

- Try on:
  - **other application** codes,
  - **protected** codes,
  - **other microcontrollers**,
- **Multispot** laser:
  - More **possibilities** of corruption,
  - Disable **error-detection/correction** capabilities.
- Design **countermeasures.**

Possibilities

- **Bit-set** on Flash data,
- Security level **lowering**.

Limitations

- **Contiguous** bits only,
- **Control-flow** alteration mostly.

Perspectives:

- Try on:
  - **other application** codes,
  - **protected** codes,
  - **other microcontrollers**,
- **Multispot** laser:
  - More **possibilities** of corruption,
  - Disable **error-detection/correction** capabilities.
- Design **countermeasures.**

— Questions? —