Corruption temporaire par injection laser du firmware stocké en mémoire Flash d'un micro-contrôleur 32 bits

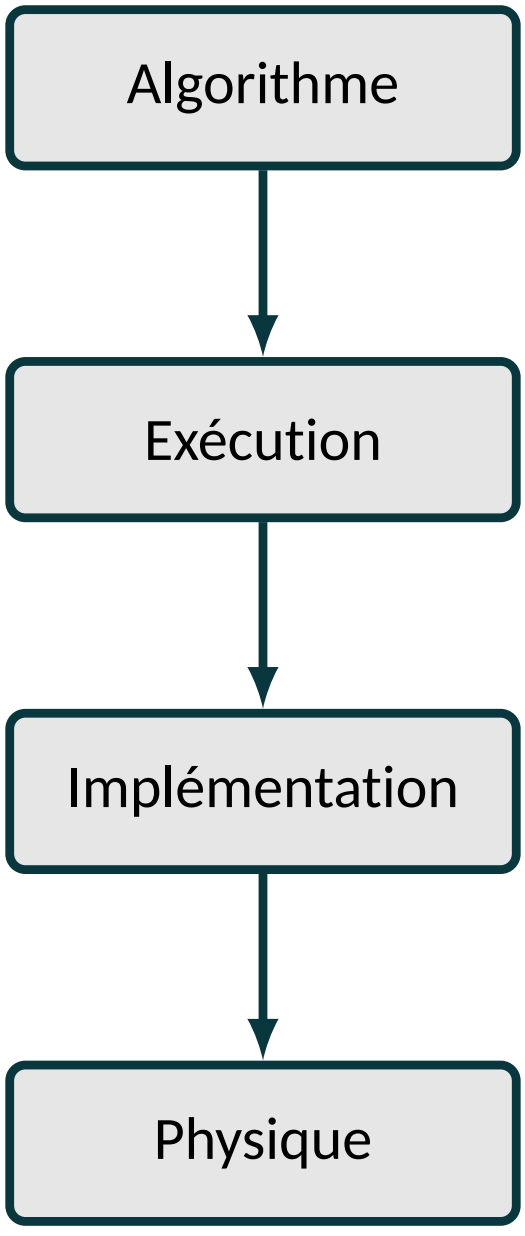
Brice Colombier[1], Alexandre Menu[2], Jean-Max Dutertre[2], Pierre-Alain Moëllic[3], Jean-Baptiste Rigaud[2] and Jean-Luc Danger[4]

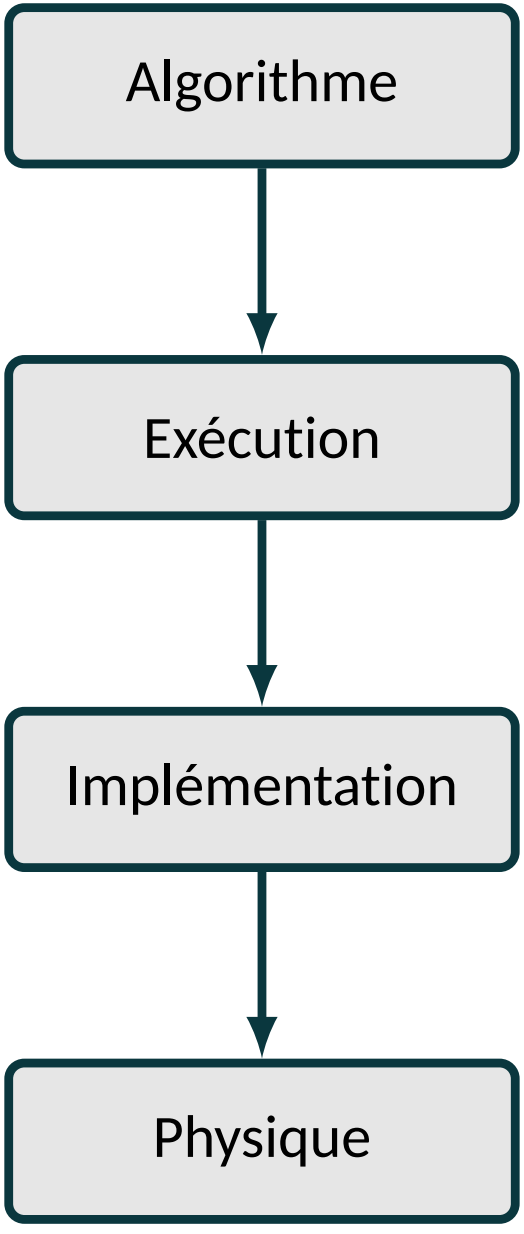
[1]Univ Lyon, UJM-Saint-Etienne, CNRS, Laboratoire Hubert Curien
[2]IMT, Mines Saint-Etienne, Centre CMP, Equipe Commune CEA Tech - Mines Saint-Etienne
[3]CEA Tech, Centre CMP, Equipe Commune CEA Tech - Mines Saint-Etienne
[4]LTCI, Télécom ParisTech , Institut Mines-télécom, Université Paris Saclay

Journée thématique sur les attaques par injection de fautes

23 mai 2019

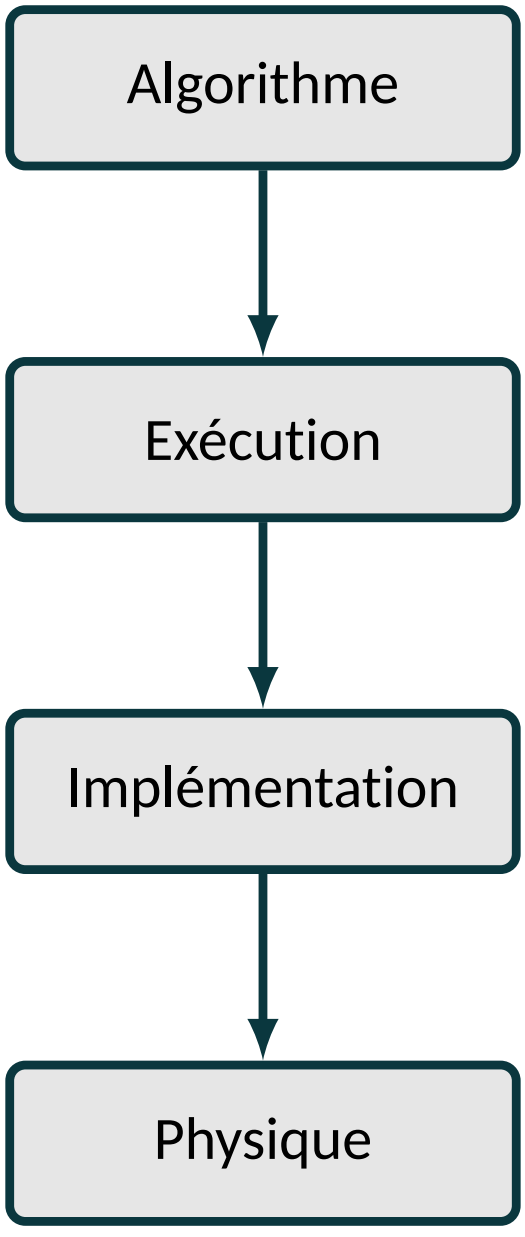
Attaques en faute sur les micro-contrôleurs 32 bits


Algorithme
Exécution
Implémentation
Physique



Compréhension sur cible **8 bits** :

- attaques sur algorithmes cryptographiques,
- corruption de registres et saut d'instruction,
- violation des contraintes temporelles.

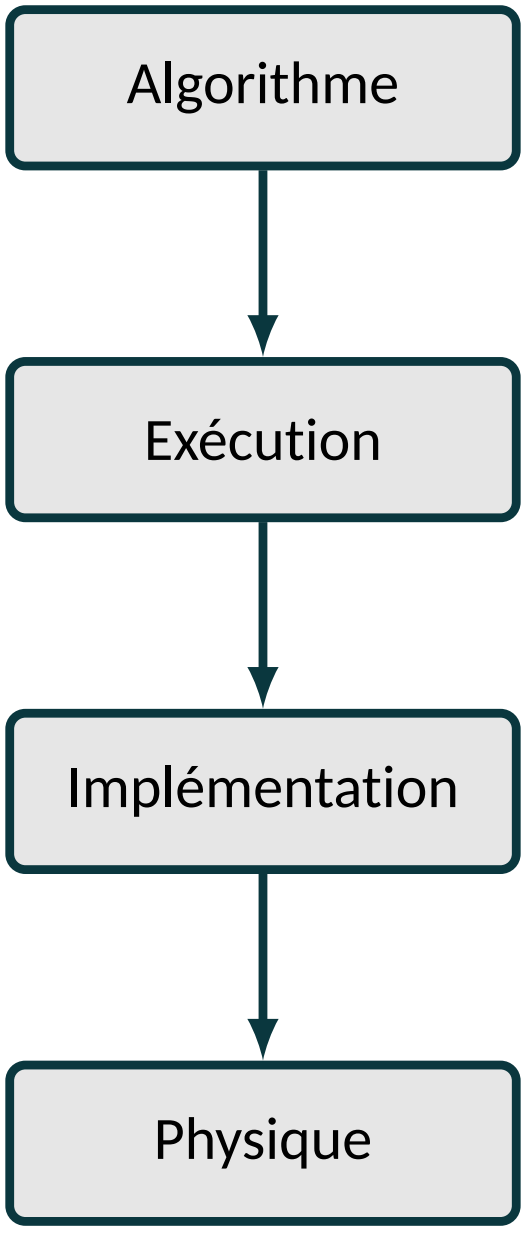


Compréhension sur cible **8 bits** :

- attaques sur algorithmes cryptographiques,
- corruption de registres et saut d'instruction,
- violation des contraintes temporelles.

Compréhension sur cible **32 bits** :

- **Actuellement** : majoritairement au niveau de l'algorithme et de l'exécution.



Compréhension sur cible **8 bits** :

- attaques sur algorithmes cryptographiques,
- corruption de registres et saut d'instruction,
- violation des contraintes temporelles.

Compréhension sur cible **32 bits** :

- **Actuellement** : majoritairement au niveau de l'algorithme et de l'exécution.

Défis spécifiques aux cibles 32 bits

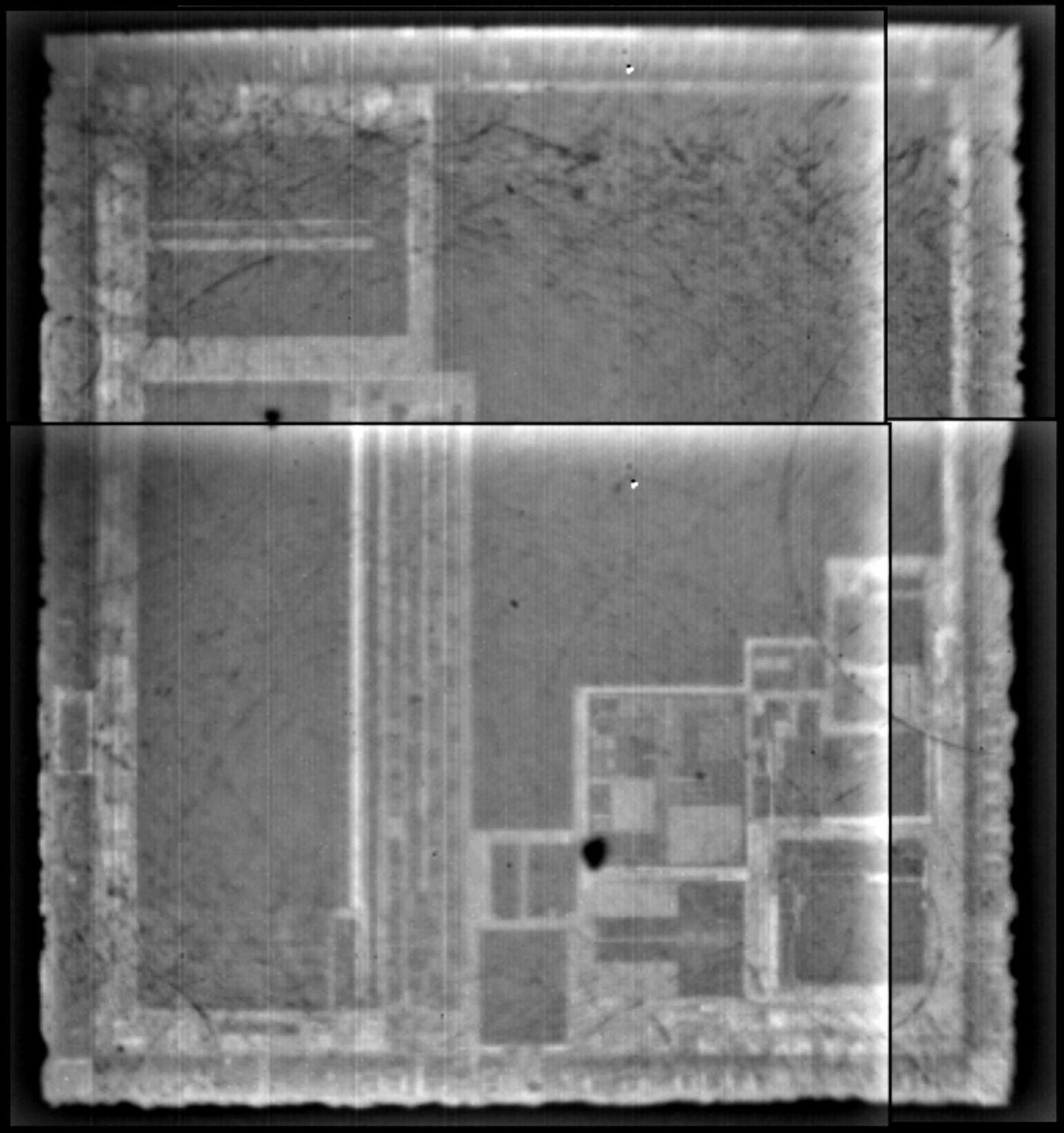
- Circuits **plus gros** et **plus complexes**,
- **Micro-architecture** : pipeline, pre-fetch...
- **Variabilité** de la durée d'exécution.

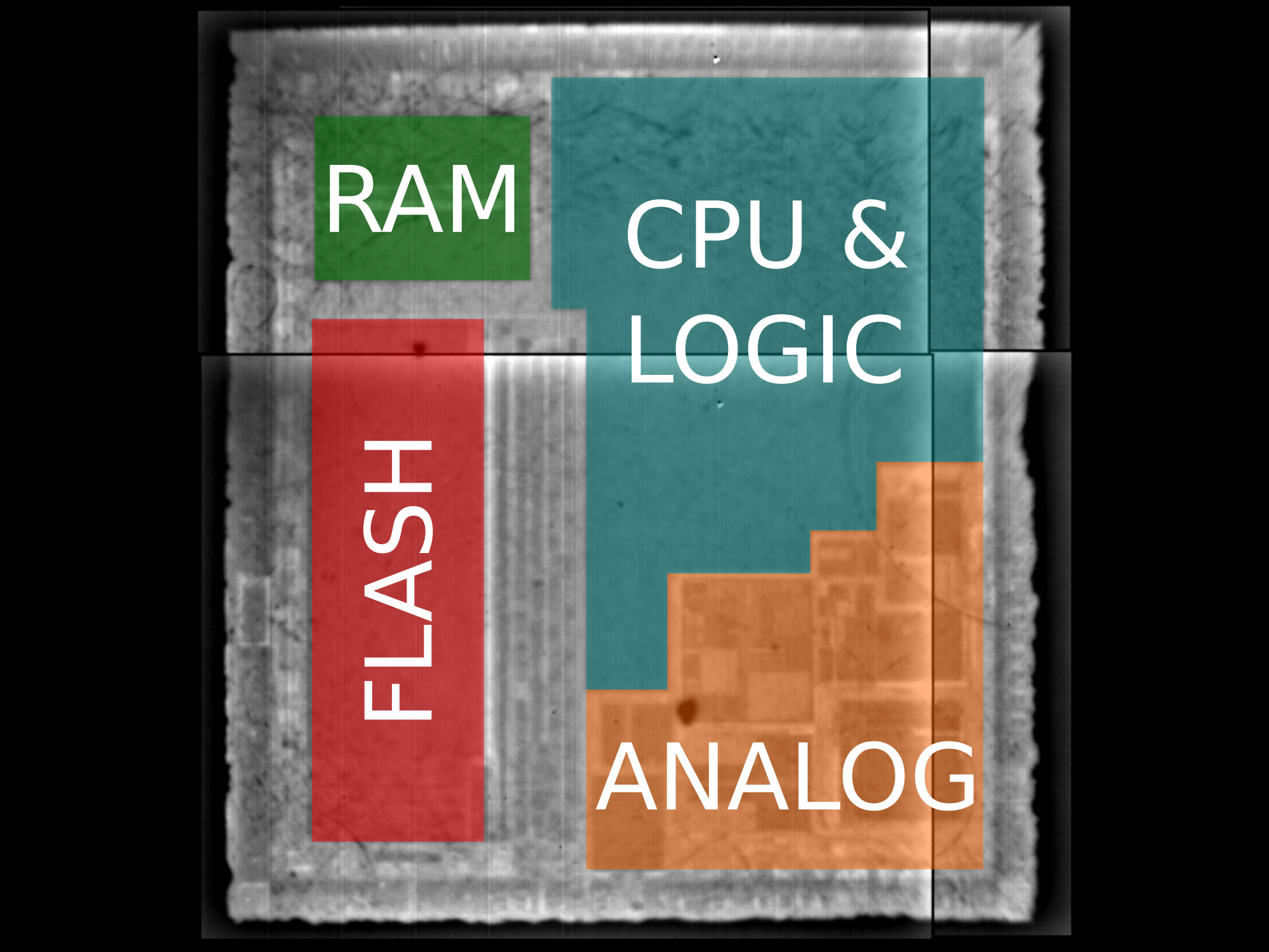
Dispositif expérimental et travail préparatoire

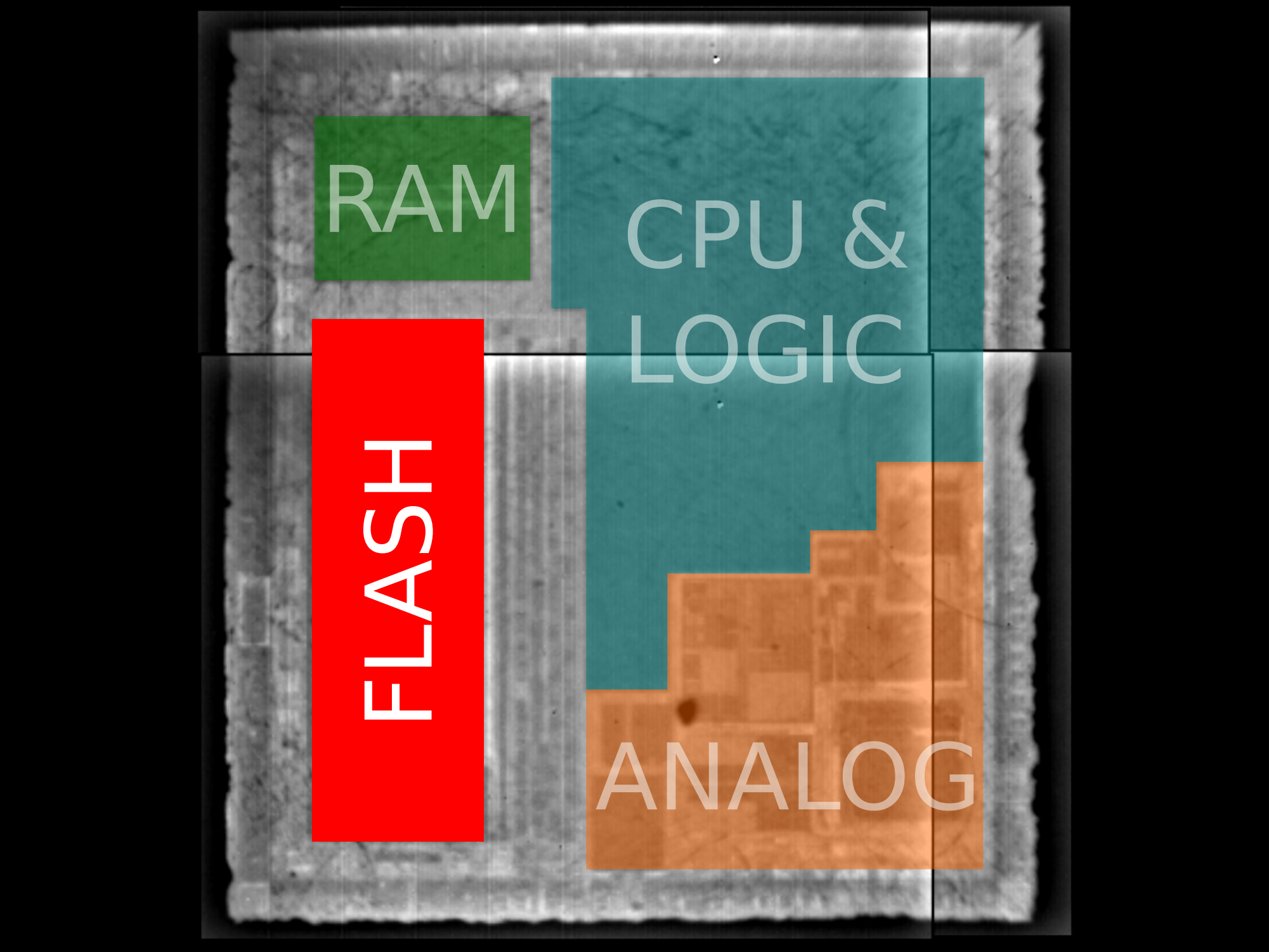
Micro-contrôleur **32 bits** :

- 2.5 x 2.5 mm,
- Cœur ARM Cortex-M3,
- Nœud technologique 90 nm,
- 128 kB de mémoire Flash.

Le code source C est compilé en jeu d'instruction **Thumb-2**.

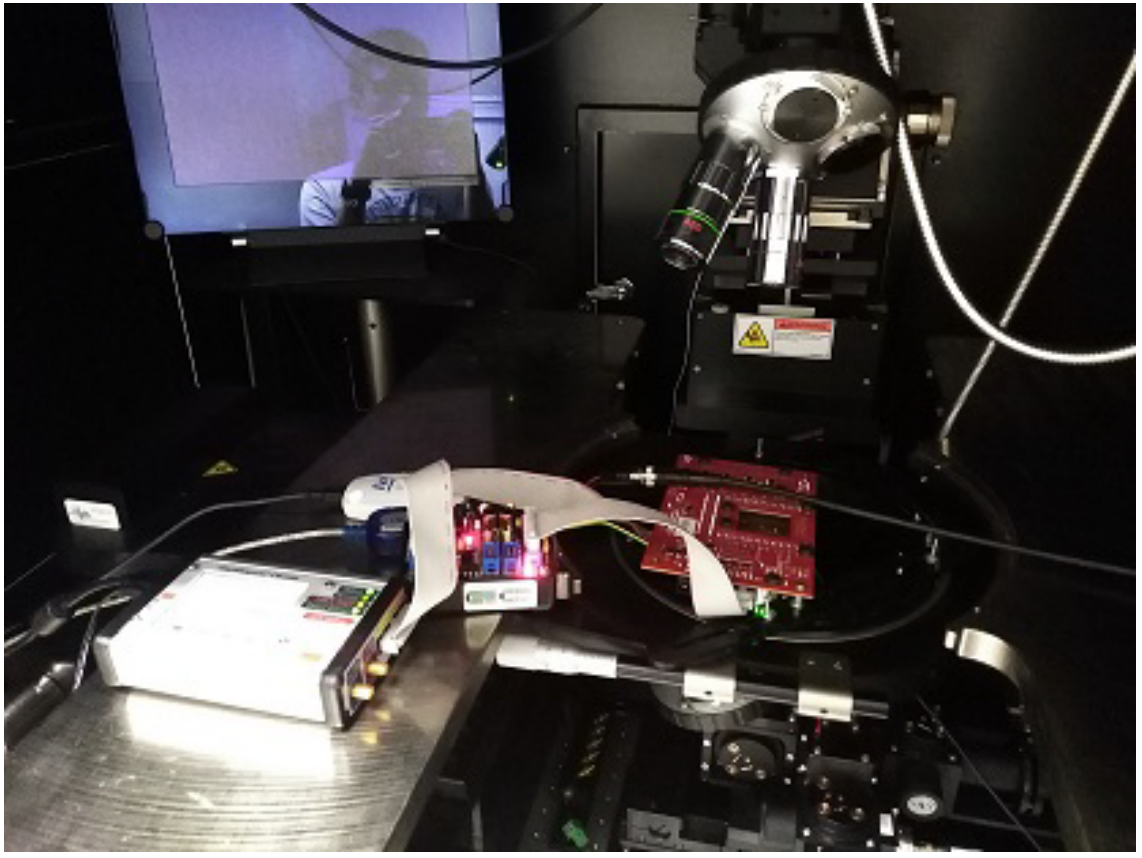



RAM
CPU &
LOGIC
FLASH
ANALOG


RAM
CPU &
LOGIC
FLASH
ANALOG

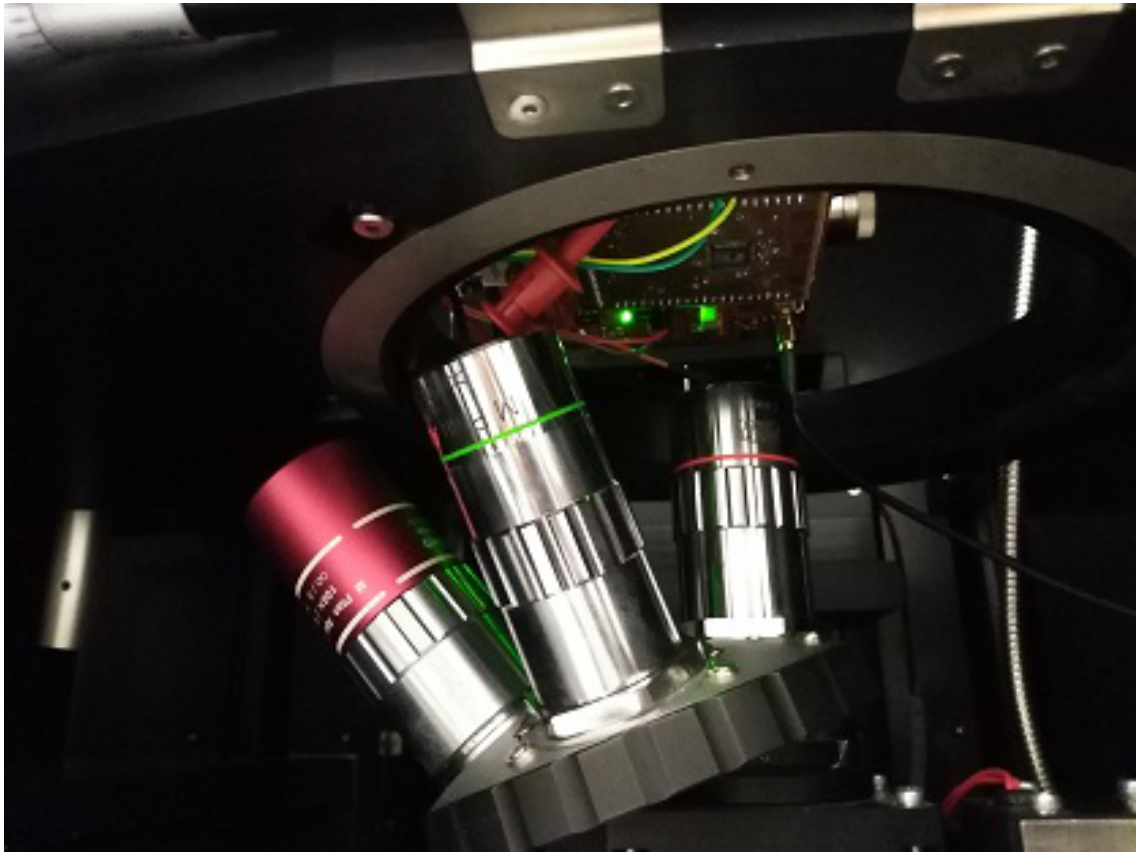
Caractéristiques du banc laser

- **Infrarouge** (1064 nm) : injection en **face arrière**,
- >30 ps,
- 0-3 W,
- 3 objectifs :
  - x5 (20 µm),
  - **x20 (5 µm),**
  - x100 (1 µm).



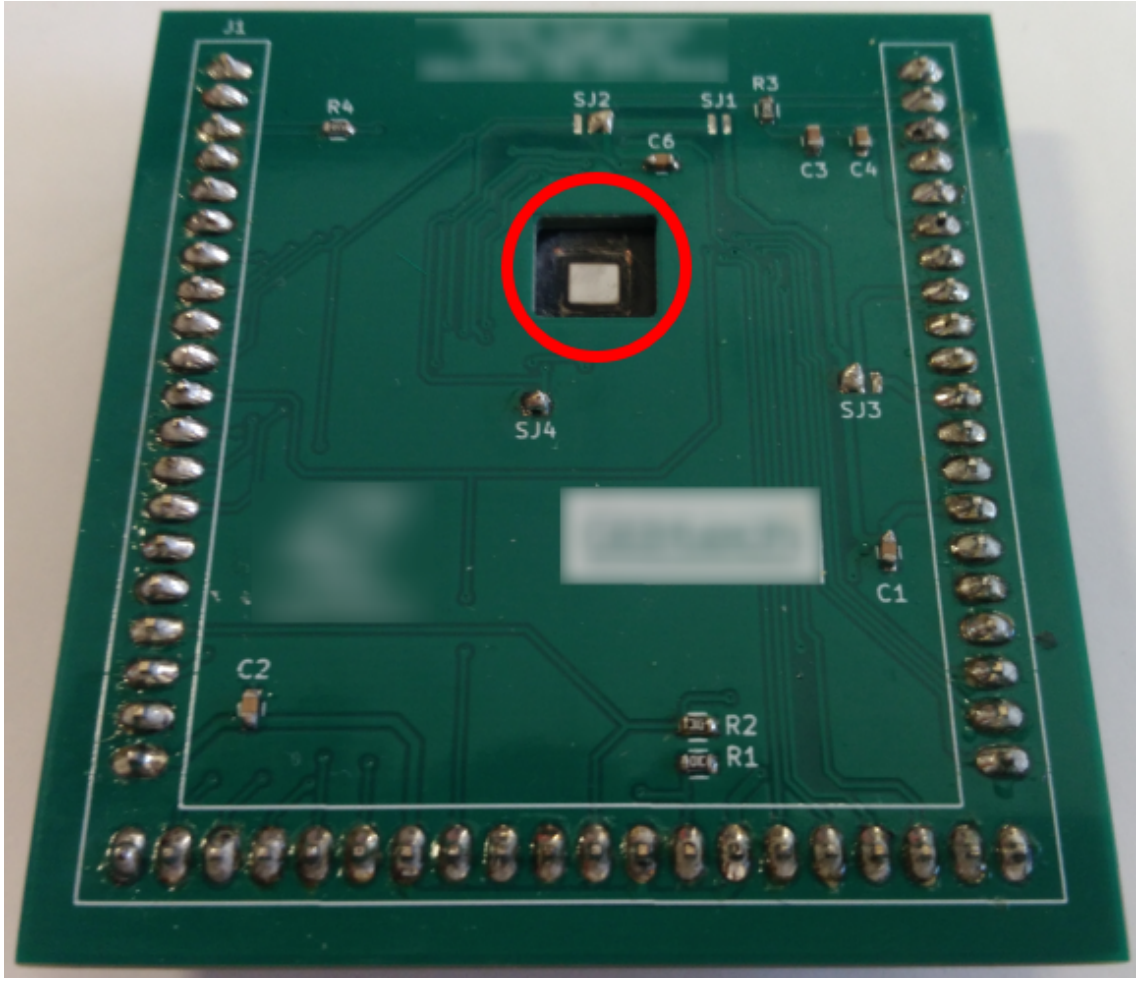
Caractéristiques du banc laser

- **Infrarouge** (1064 nm) : injection en **face arrière**,
- >30 ps,
- 0-3 W,
- 3 objectifs :
  - x5 (20 µm),
  - **x20 (5 µm),**
  - x100 (1 µm).



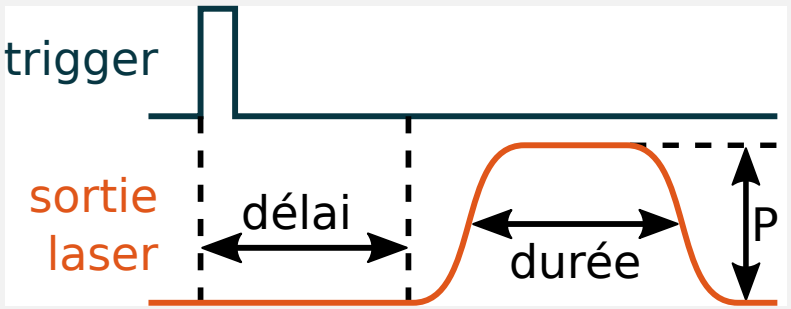
Caractéristiques du banc laser

- **Infrarouge** (1064 nm) : injection en **face arrière**,
- >30 ps,
- 0-3 W,
- 3 objectifs :
  - x5 (20 µm),
  - **x20 (5 µm),**
  - x100 (1 µm).



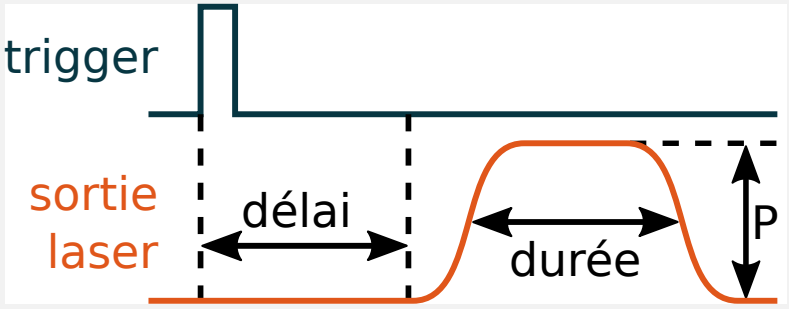
Travail préparatoire (4-5 mois)

- Conception d'une carte-cible ChipWhisperer spécifique :
  - Accès en **face avant**,
  - Accès en **face arrière**.
- **Préparation** : **Décapsulation** pour accéder à la puce,
- **Mise en place mécanique** sur le banc d'injection,
- **Recensement** des fautes:
  - position en x,
  - position en y,
  - puissance,
  - durée,
  - délai,



Travail préparatoire (4-5 mois)

- Conception d'une carte-cible ChipWhisperer spécifique :
  - Accès en **face avant**,
  - Accès en **face arrière**.
- **Préparation** : **Décapsulation** pour accéder à la puce,
- **Mise en place mécanique** sur le banc d'injection,
- **Recensement** des fautes:
  - position en x,
  - position en y,
  - puissance,
  - durée,
  - délai,
  - **type** de faute : saut d'instruction, mise à 0/1, inversion...



Résultats de caractérisation

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

- Écrire une **donnée de test spécifique** à une adresse donnée en mémoire Flash.
- **Stocker** cette donnée dans un **registre connu**,
- **Lire** ce registre.

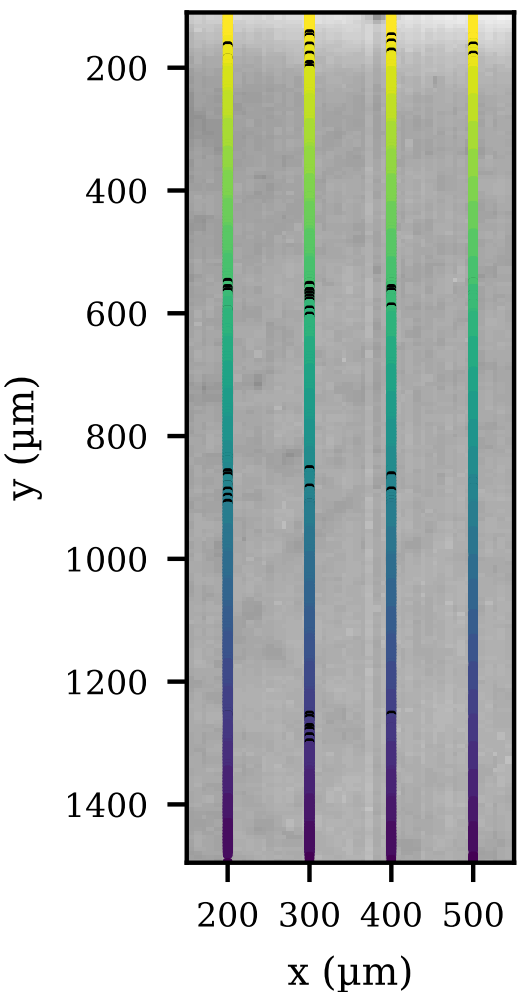
```
1  donnee_test:
2  .word 0x00000000
3  NOP
4  NOP
5  NOP
6  NOP
7  NOP
8  NOP
9  LDR R0, donnee_test
10 NOP
11 NOP
12 NOP
13 NOP
14 NOP
15 NOP
16 # Lecture de R0
```

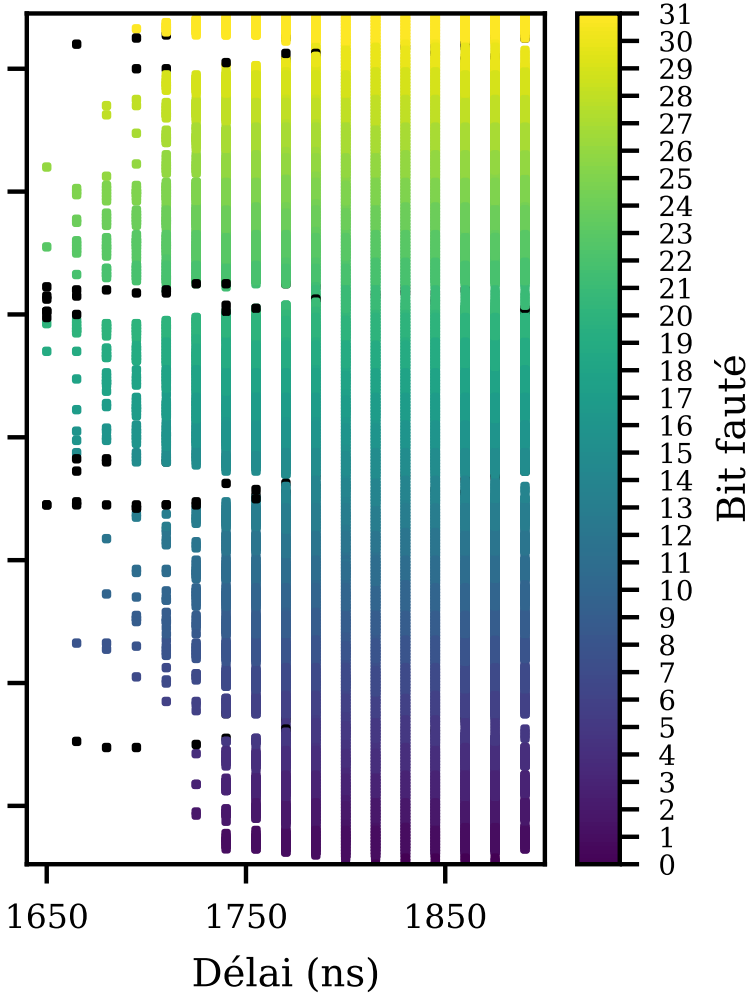
- Écrire une **donnée de test spécifique** à une adresse donnée en mémoire Flash.
- **Stocker** cette donnée dans un **registre connu**,
- **Lire** ce registre.

Choix de la donnée de test

- `0x00000000`: mise à 1,
- `0xFFFFFFFF`: mise à 0,
- `0x55555555`
  `0xAAAAAAAA`: inversion.

Bit fauté en fonction de la position et du délai





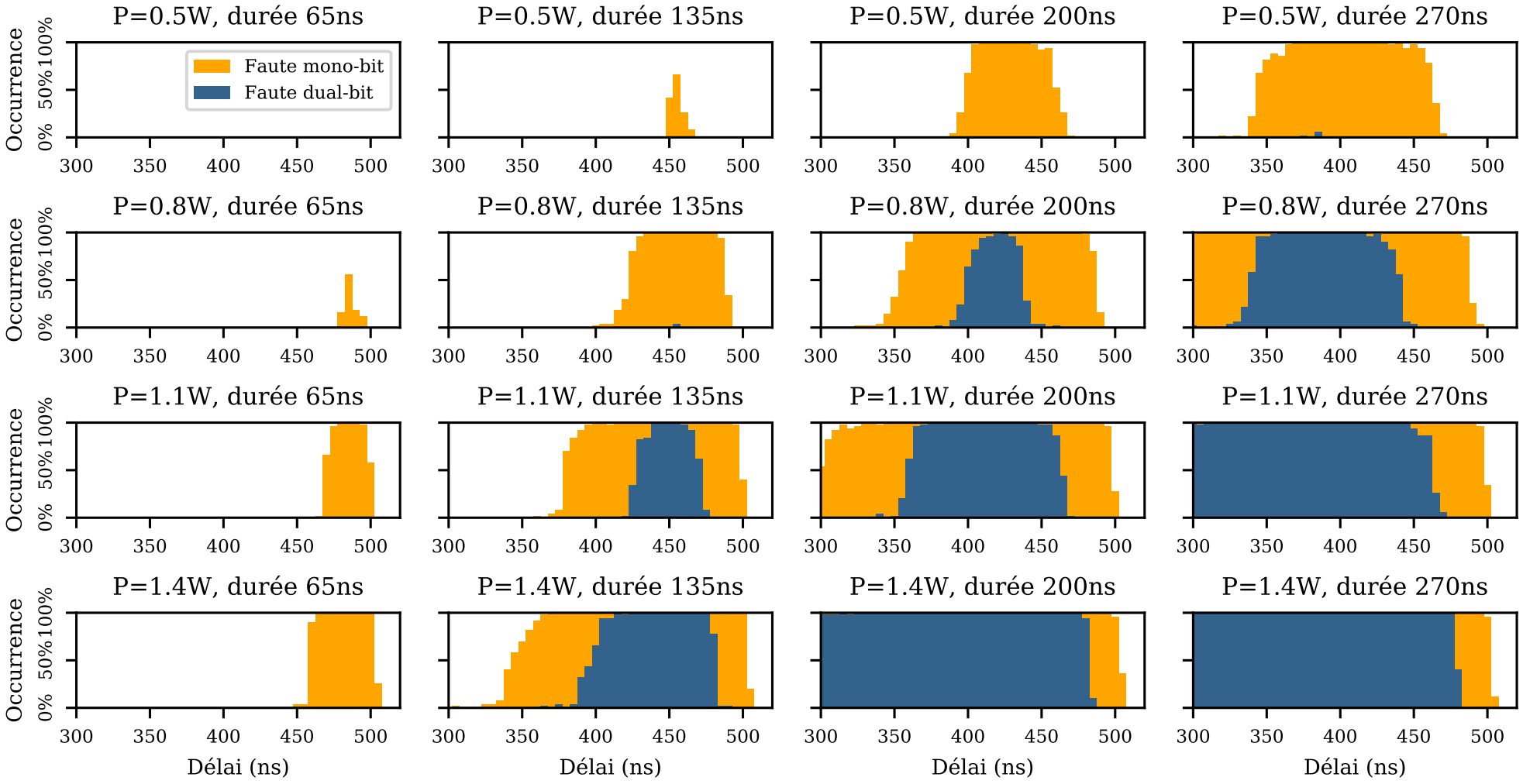
Modèle de faute asymétrique

Mise à 1 monobit de la donnée.

Dépendance aux paramètres

Le bit fauté dépend de la position **y**.

Bit fauté en fonction de la puissance et de la durée

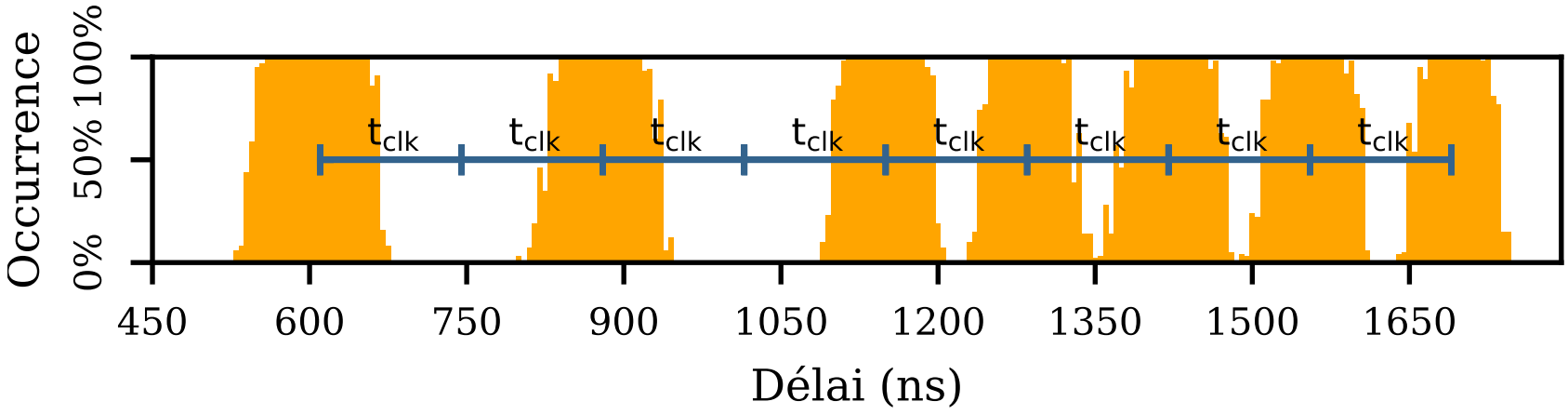


Observation

Augmenter **l'énergie** permet de fauter plus de bits.

```
1  # Initialisation des registres
2  # R0, R1, R4, R5, R6, R8
3  # et R9 à 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 # Lecture des registres
```

Périodicité des fautes (bit fauté en fonction du délai)



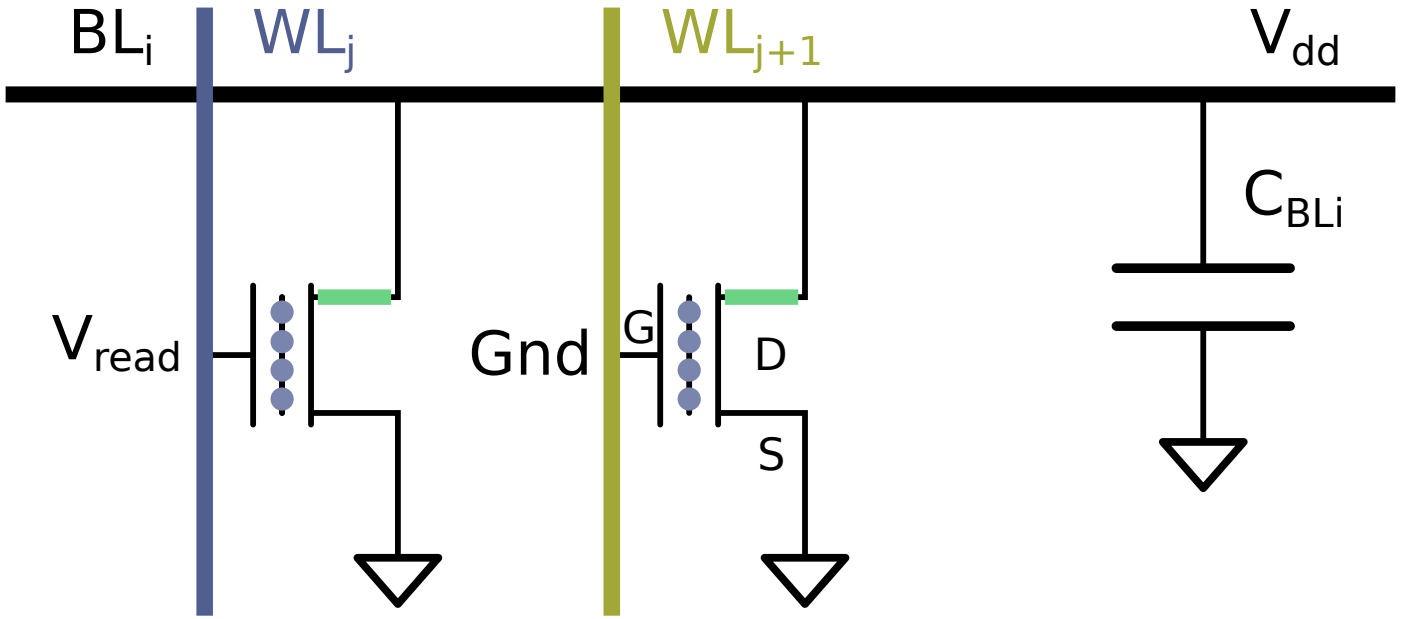

```
1  MOVW R0, 0x0000
2  MOVW R1, 0x0000
3  MOVW R4, 0x0000
4  MOVW R5, 0x0000
5  MOVW R6, 0x0000
6  MOVW R8, 0x0000
7  MOVW R9, 0x0000
```

Observations

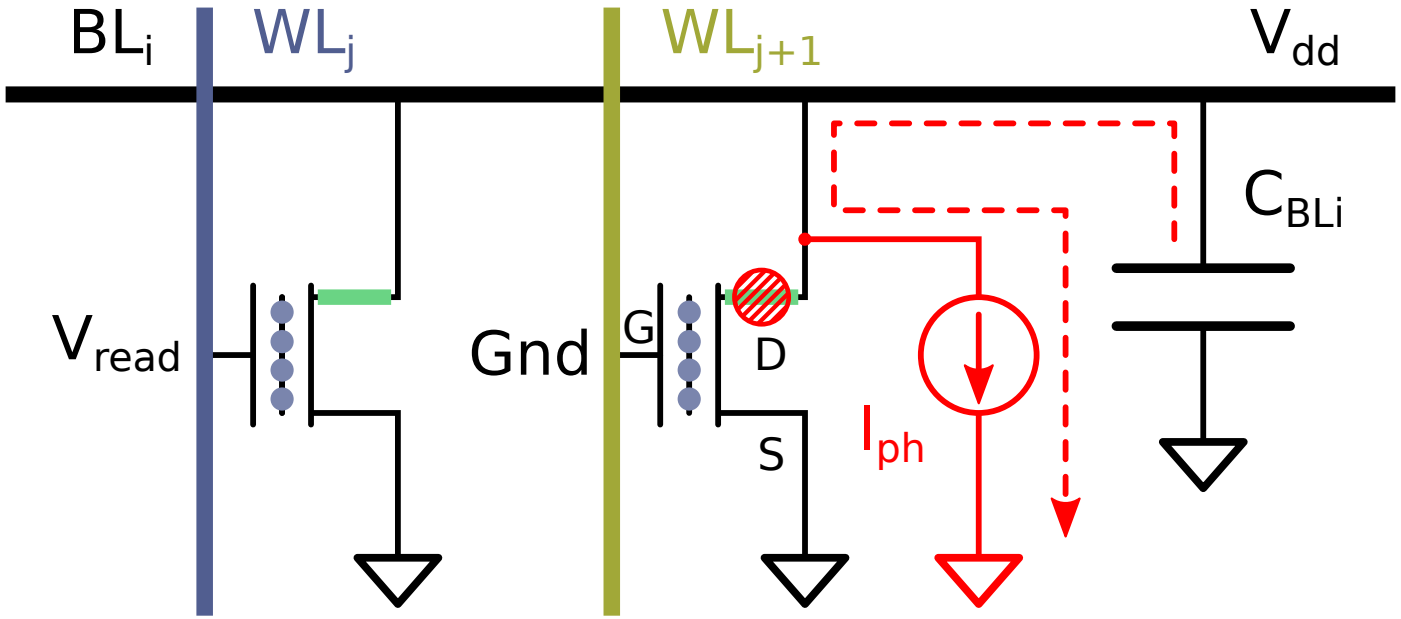
- **Toute** instruction peut être fautée,
- L'occurrence atteint **toujours 100%**,
- Le délai entre deux temps d'injection optimaux est toujours un **multiple de la période d'horloge**.
- Le délai entre deux temps d'injection optimaux n'est **pas constant**.

Proposition d’explication physique

Proposition d'explication physique de la dépendance en y



Proposition d'explication physique de la dépendance en y

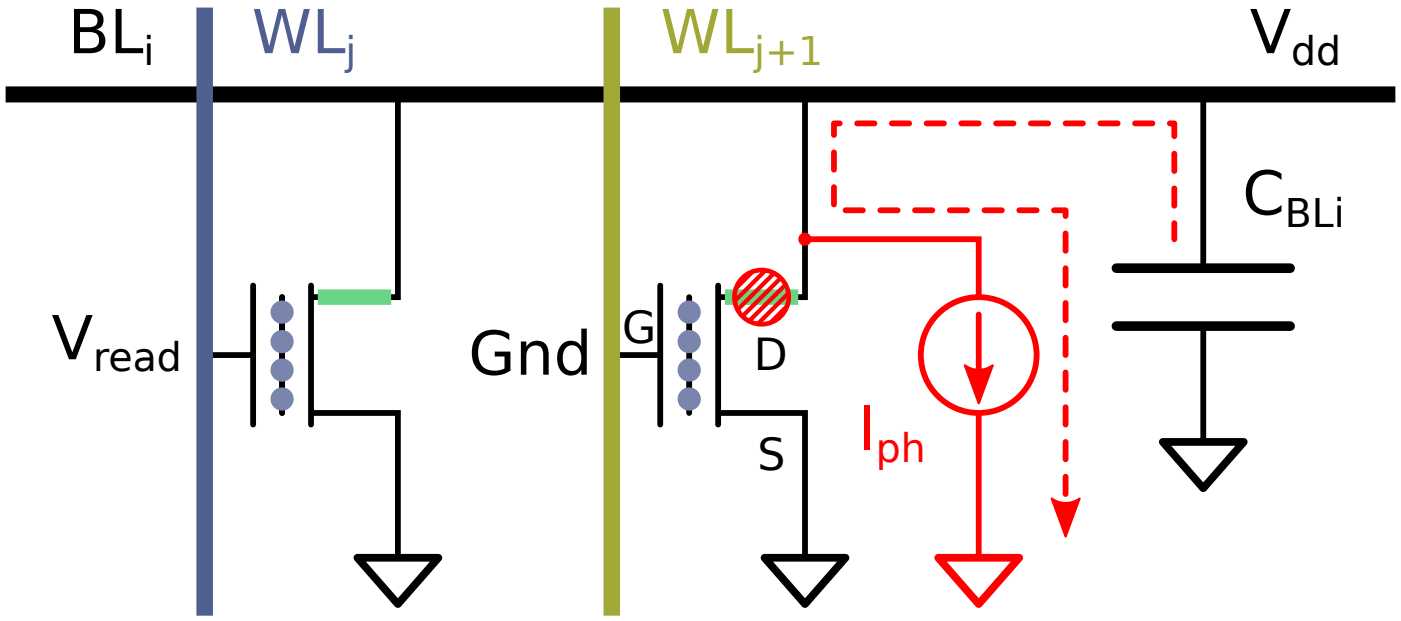


Grille flottante chargée ('0' logique)

Jonction polarisée en inverse

Spot laser

Proposition d'explication physique de la dépendance en y



Grille flottante chargée ('0' logique)

Jonction polarisée en inverse

Spot laser

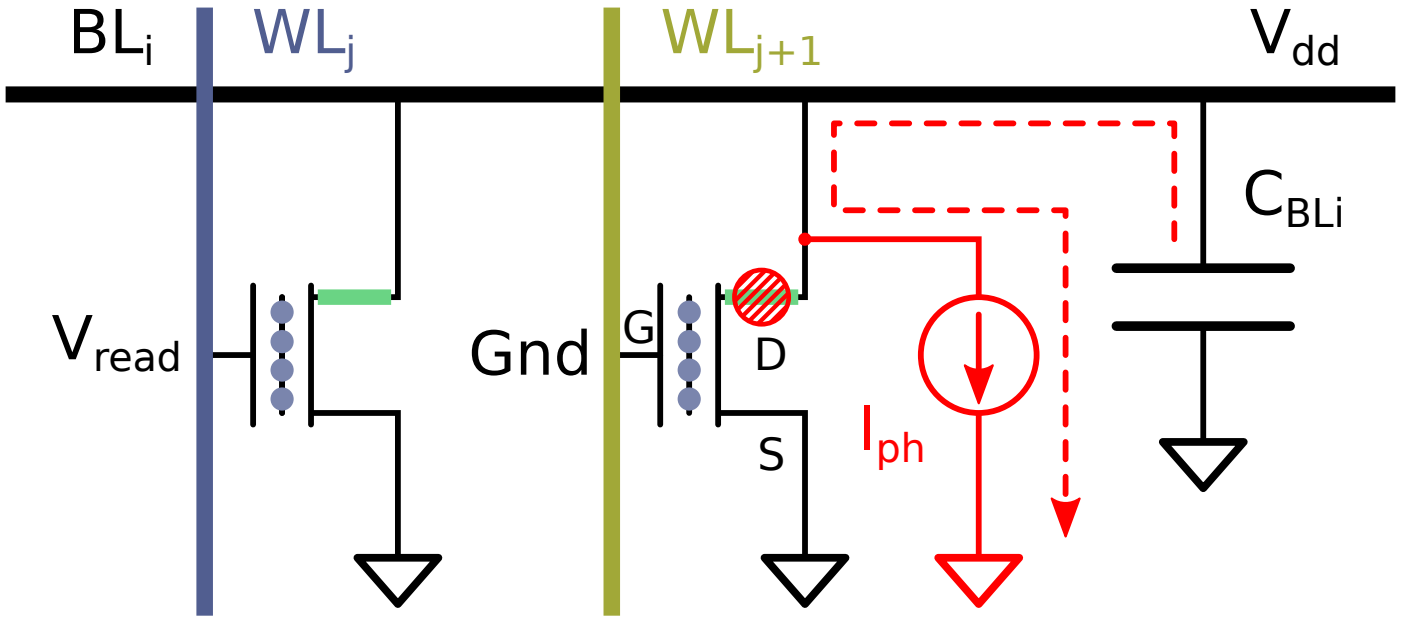
Déplacement selon l'axe x

- Transistors de la même **BL**.
- **Même** bit fauté.

Déplacement selon l'axe y

- Transistors de la même **WL**.
- Bits fautés **successifs**.

Proposition d'explication physique de l'asymétrie



Grille flottante chargée ('0' logique)

Jonction polarisée en inverse

Spot laser

Sans tir laser

- **avec** charges : BL à $\mathbf{V_{dd}}$
- **sans** charges : BL à **GND**

Lors d'un tir laser

- **avec** charges : BL à **GND**
- **sans** charges : BL à **GND**

Applications

Exemple de corruption d'instruction: `MOVW` 32 bits

`MOVW`: stocke une valeur dans la moitié basse d'un registre 32 bits.

bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`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

Instructions de référence :

Exemple de corruption d'instruction: MOVW 32 bits

MOVW: stocke une valeur dans la moitié basse d'un registre 32 bits.

bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Instructions de référence :

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

Corruption de **donnée** :

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

Exemple de corruption d'instruction: `MOVW` 32 bits

`MOVW`: stocke une valeur dans la moitié basse d'un registre 32 bits.

bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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

Instructions de référence :

Corruption de **donnée** :

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

Corruption de **registre source/destination** :

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

Exemple de corruption d'instruction: MOVW 32 bits

MOVW: stocke une valeur dans la moitié basse d'un registre 32 bits.

bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Instructions de référence :

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

Corruption de **donnée** :

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

Corruption de **registre source/destination** :

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

Corruption de l'**opcode** :

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

Algorithme de vérification de PIN

Implémentation en **temps constant** avec booléens durcis :
Pas d'attaque *timing* simple, `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
```

Algorithme de vérification de PIN

Implémentation en **temps constant** avec booléens durcis :
Pas d'attaque *timing* simple, `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

Instructions de référence

CMP générique	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

Instructions de référence

CMP générique	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

Réaliser une mise à 1 sur le **10ème** bit de l'instruction : R3 ➔ R7.
Par conception, R7 stocke le *frame pointer*, **toujours positif**.

Corruption du registre source ⬇

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

Instructions de référence

CMP générique	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

Réaliser une mise à 1 sur le **10ème** bit de l'instruction : R3 ➔ R7.
Par conception, R7 stocke le *frame pointer*, **toujours positif**.

Corruption du registre source ↓

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

Conséquence

trials n'est **jamais comparé** ➔ nombre **illimité** d'essais.

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: $S_{i,j} = S_{i,j} \oplus K^{10}_{i,j}$
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

Instructions de référence

ADD générique	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

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

Instructions de référence

ADD générique	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

Réaliser une mise à 1 sur le **2ème** bit de l'instruction.
Ajouter **5** au lieu de **1** à la variable de boucle.

Corruption de donnée

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

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

Instructions de référence

ADD générique	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

Réaliser une mise à 1 sur le **2ème** bit de l'instruction.
Ajouter **5** au lieu de **1** à la variable de boucle.

Corruption de donnée

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

Conséquence

On sort de la boucle *for* après **une exécution** seulement.

Octet du chiffré fauté : $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$

Faute sur la boucle for **interne**,
lors de sa **première** exécution.

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

Faute sur les boucles *for*

Octet du chiffré fauté : $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$

Faute sur la boucle for **interne**, lors de sa **première** exécution.

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

Faute sur la boucle *for* externe.

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

Récupération de la 10ème clé de ronde

Octet du chiffré fauté : $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$

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

$\oplus$

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

=

Octet du chiffré fauté : $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$

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

$\oplus$

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

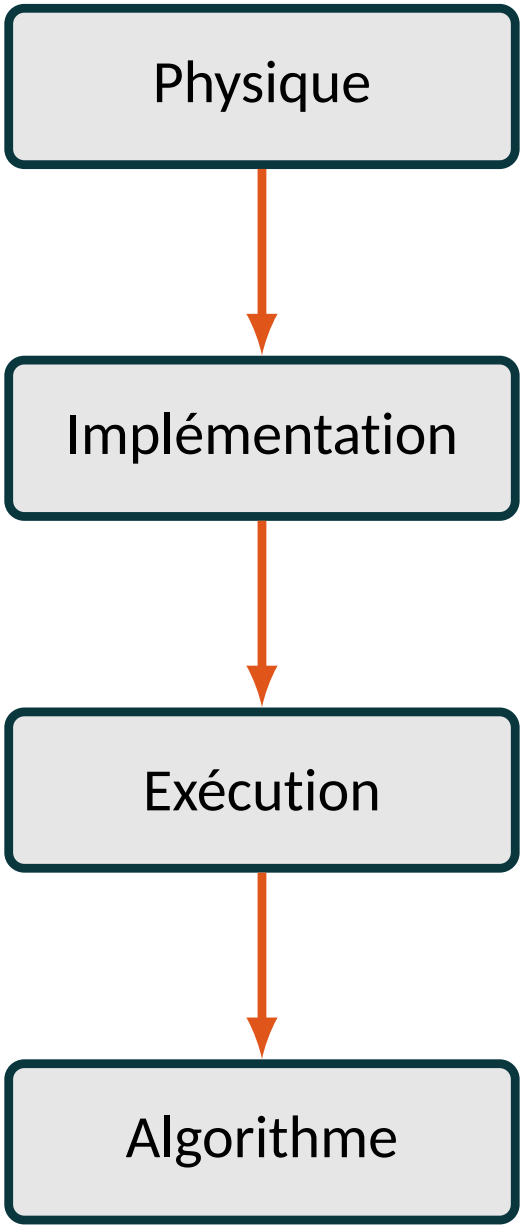
$=$

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

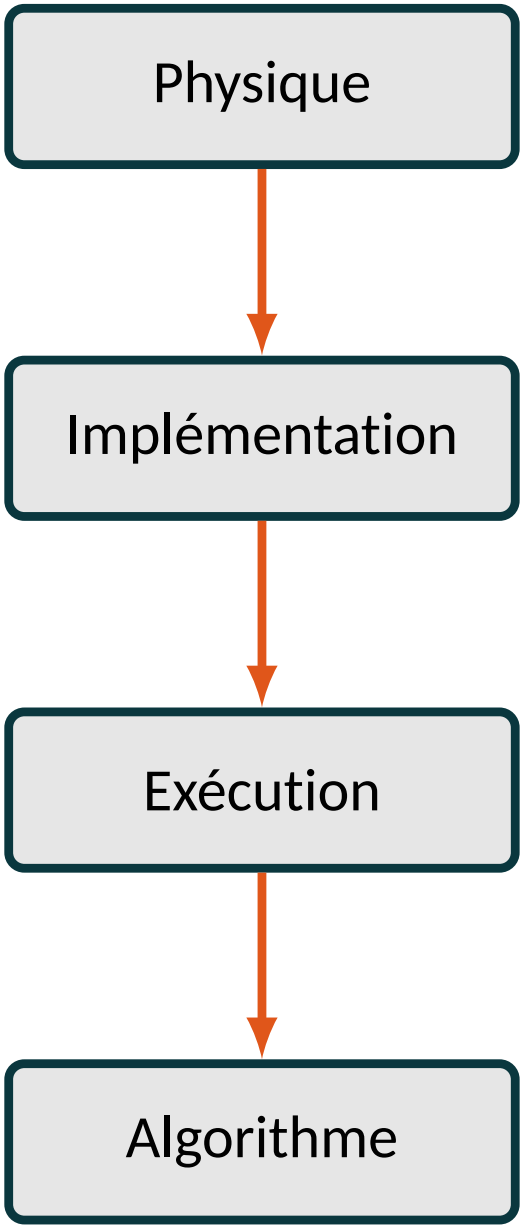
Ensuite ?

Recherche exhaustive du premier octet $K^{10}_{0,0}$.

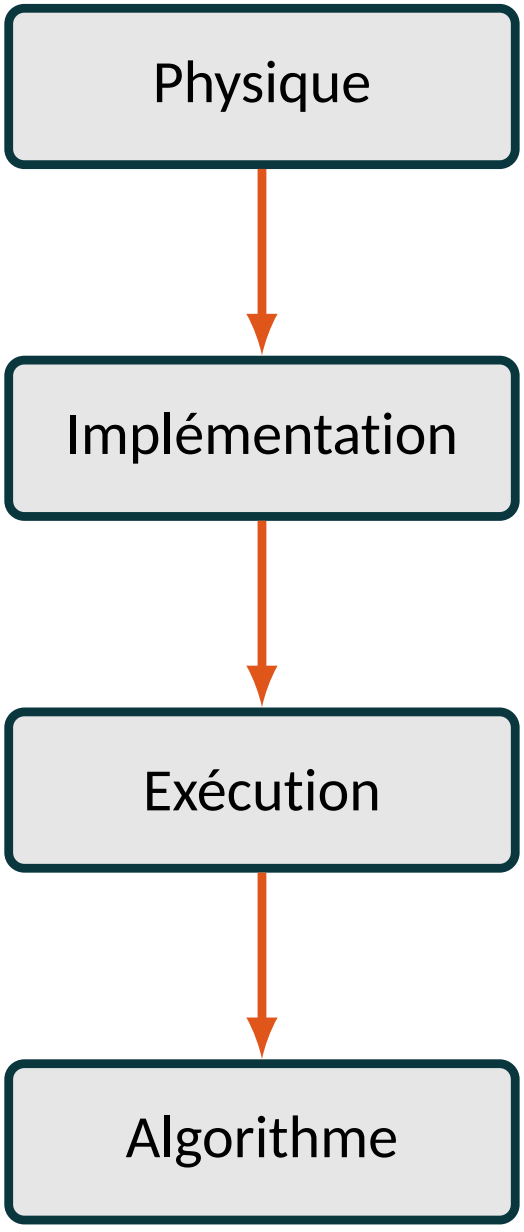
Conclusion


Physique
Implémentation
Exécution
Algorithme

Résumé

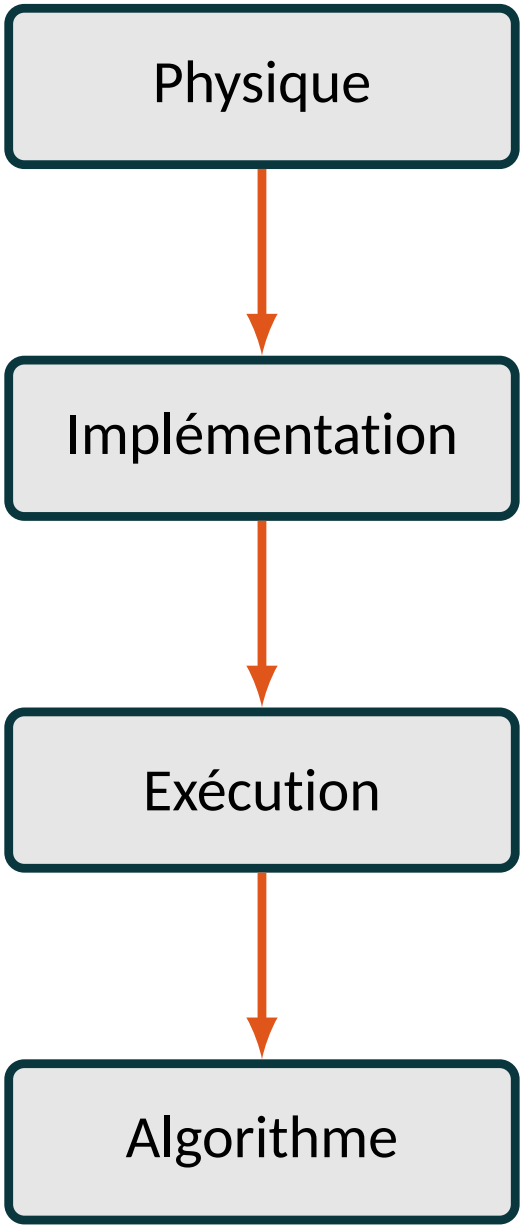


Forcer les transistors de stockage de la mémoire Flash à **conduire**



Forcer les transistors de stockage de la mémoire Flash à **conduire**

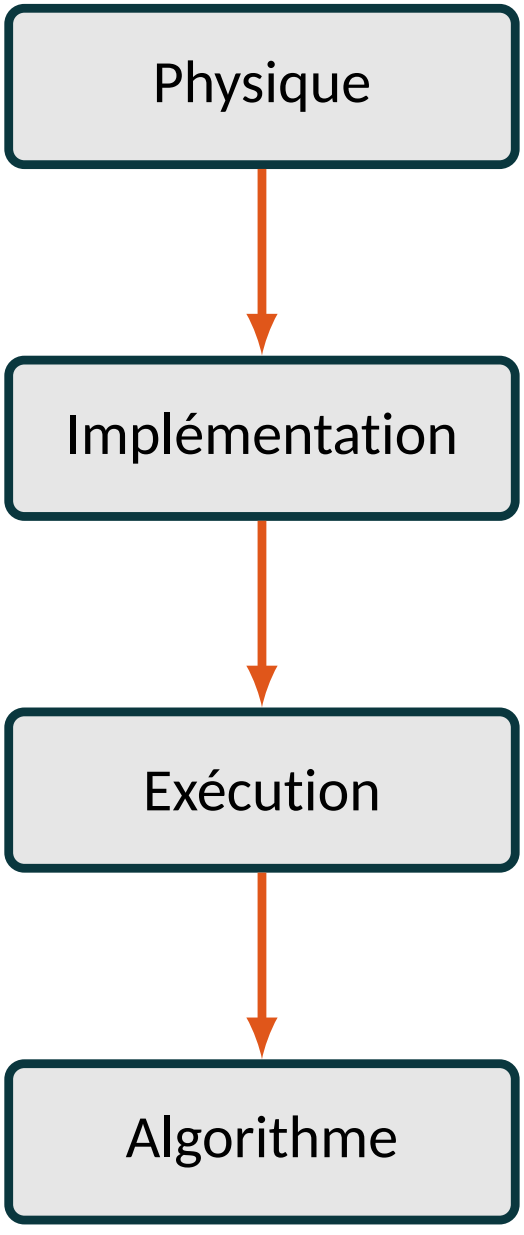
Réaliser une **mise à 1** d'un **bit unique et choisi** de l'instruction.



Forcer les transistors de stockage de la mémoire Flash à **conduire**

Réaliser une **mise à 1** d'un **bit unique et choisi** de l'instruction.

Toujours prendre la première branche d'un **if**.
Sortir prématurément d'une boucle *for*.



Forcer les transistors de stockage de la mémoire Flash à **conduire**

Réaliser une **mise à 1** d'un **bit unique et choisi** de l'instruction.

Toujours prendre la première branche d'un **if**.
Sortir prématurément d'une boucle *for*.

Nombre **illimité** d'essais pour un VerifyPIN.
Fauter le dernier `AddRoundKey` de l'AES-128.

Possibilités

- **Mise à 1** de la donnée lue en mémoire Flash,
- **Réduction** de sécurité.

Limitations

- Bits **contigus** seulement,
- Ciblant majoritairement le **flot de contrôle**.

Possibilités

- **Mise à 1** de la donnée lue en mémoire Flash,
- **Réduction** de sécurité.

Limitations

- Bits **contigus** seulement,
- Ciblant majoritairement le **flot de contrôle**.

Perspectives :

- Essayer sur d'**autres codes applicatifs**.
- Essayer sur des **codes protégés**.
- Essayer sur d'**autres micro-contrôleurs** et **mémoires Flash**.
- Laser **multispot** :
  - Plus de **possibilités** de corruption,
  - Désactivation des fonctions de **détection/correction d'erreurs**.

Possibilités

- **Mise à 1** de la donnée lue en mémoire Flash,
- **Réduction** de sécurité.

Limitations

- Bits **contigus** seulement,
- Ciblant majoritairement le **flot de contrôle**.

Perspectives :

- Essayer sur d'**autres codes applicatifs**.
- Essayer sur des **codes protégés**.
- Essayer sur d'**autres micro-contrôleurs** et **mémoires Flash**.
- Laser **multispot** :
  - Plus de **possibilités** de corruption,
  - Désactivation des fonctions de **détection/correction d'erreurs**.

— Questions ? —