DE LA RECHERCHE À L'INDUSTRIE

cea

ÉCOLE DES MINES NATIONALE SUPÉRIEURE DE SAINT-ÉTIENNE

# ATTAQUES PAR INJECTION DE FAUTES SUR MICROCONTRÔLEUR ET CONTRE-MESURES AU NIVEAU DU CODE EMBARQUÉ

**Nicolas MORO** (CEA)

Thèse encadrée par **Bruno ROBISSON** (CEA)
Sous la direction d'**Emmanuelle ENCRENAZ** (LIP6)

*Des parties du travail présenté ont été effectuées en collaboration avec Amine Dehbaoui (ENSM.SE) et Karine Heydemann (LIP6)*

30 MAI 2013

# CONTEXTE ET MOTIVATIONS

- **Sécurité** des systèmes à microcontrôleur face aux **attaques en faute**

- **Objectif** : améliorer la **sécurité** de ces systèmes

- Approche générale :
  - Elaboration d'un **modèle réaliste d'attaquant**
    - Mesures expérimentales et **injection électromagnétique**
    - Définition d'un **modèle de fautes** au niveau du **code assembleur**

  - Définition de **contre-mesures** relatives au modèle défini
    - Conception de structures de code tolérantes aux fautes du modèle
    - Preuve formelle visant à garantir une exécution correcte du code

# SOMMAIRE

Deux principales **classes d'attaques physiques** :

## ➢ Attaques par observation

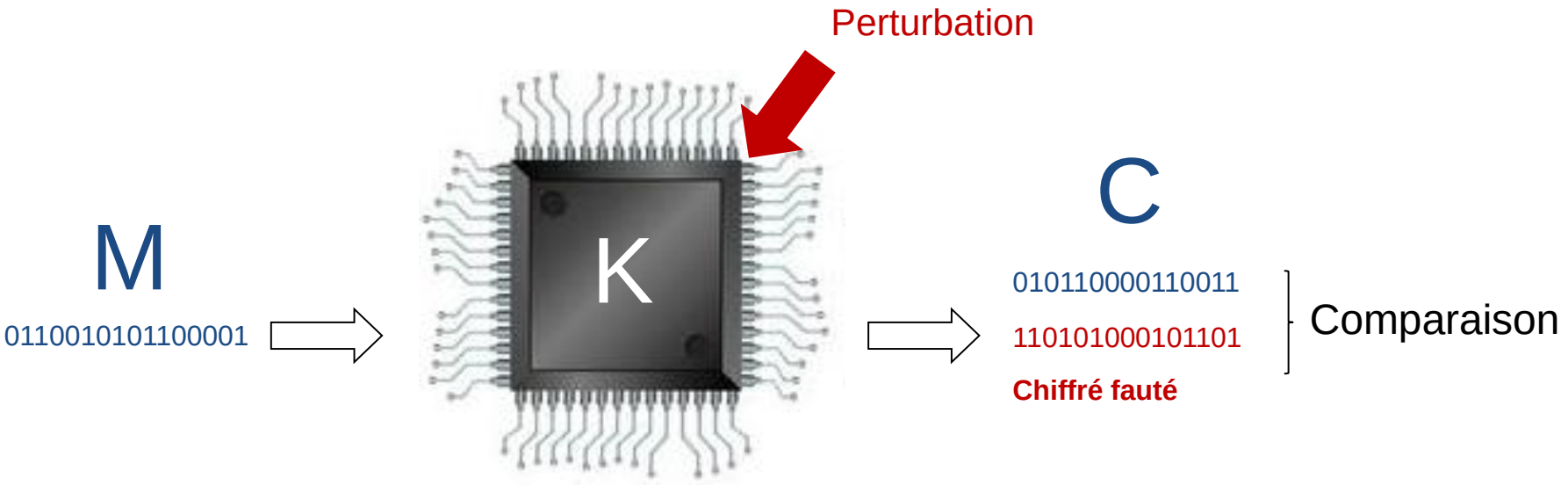
- Attaques **passives**
- Mesure de **courant**, de **rayonnement EM**, d'**émission de photons**…
- Exploitent le fait que les grandeurs physiques mesurées possèdent un certain degré de **corrélation statistique** par rapport aux données internes

## ➢ Attaques par injection de fautes

- Attaques **actives**
- Perturbation de la tension d'alimentation, **injection EM**, utilisation de lumière concentrée
- Peuvent permettre de modifier le flot d'exécution d'un programme embarqué
- Peuvent permettre de retrouver rapidement des clés d'algorithmes cryptographiques

# ATTAQUES PAR INJECTION DE FAUTE



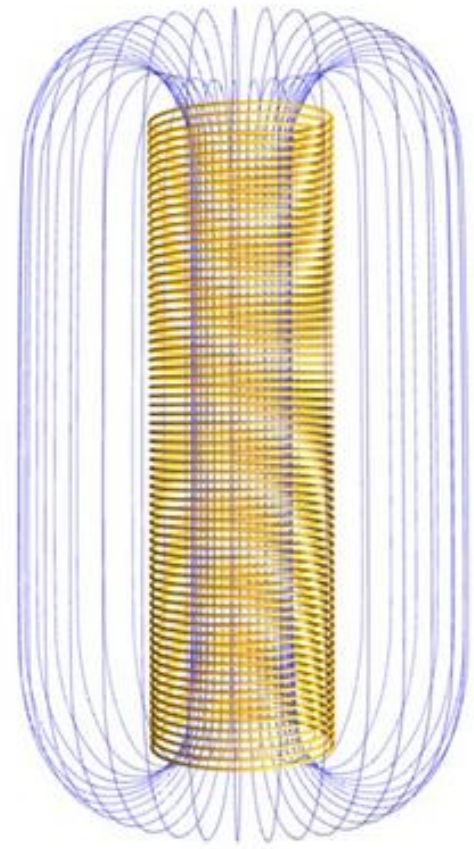



- Plusieurs **moyens physiques** d'injecter des fautes dans un circuit
- Nécessaire pour un attaquant de **connaître le type de fautes injectées**

| | |
|---|---|
| **Type de données affecté** | Données, instructions |
| **Type de faute** | Inversion, mise à zéro, mise à un, collage |
| **Granularité** | Bit, octet, mot |
| **Déterminisme** | Déterministe, métastable, aléatoire |

# INJECTION DE FAUTES PAR COUPLAGE EM





**Lignes de champ magnétique à l'intérieur d'un solénoïde**
*Source : Wikipedia*

- **Solénoïde** utilisé comme antenne d'injection
- Champ **fort à l'intérieur** du solénoïde, faible autour
- Utilisation comme moyen d'injection **en champ proche**

**Champ magnétique à l'intérieur d'une bobine**

$$\vec{B} = \mu_0 n i(t) \vec{u}_z$$

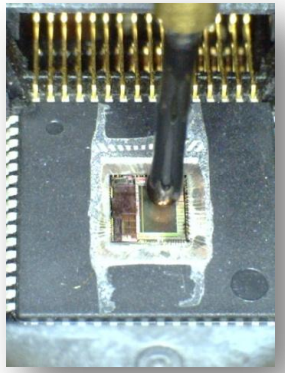
**Flux traversant une boucle magnétique de courant**

$$\Phi_B(t) = \int_S \vec{B}(M,t).d\vec{s}$$

**Force électromotrice engendrée**

$$e(t) = -\frac{\partial \Phi_B(t)}{\partial t}$$

## Injection par pulse électromagnétique



- **Effet local** et transitoire de l'injection
- **Circuits standards non protégés** contre cette technique
- **Solénoïde** utilisé comme antenne d'injection
- Réalise des **fautes par violation des contraintes temporelles** des circuits
- Jusqu'à **200V** envoyés dans le solénoïde d'injection
- Pulses de durée **supérieure à 10ns**

## Précédentes réalisations

- Injection de fautes de délai sur **FPGA 130nm**
- Saut d'instruction assembleur sur **µC Atmega 180nm**

# POSITION DU PROBLÈME

## Cas particulier d'une faute sur le dernier AddRoundKey de l'algorithme AES

AddRoundKey opcodes

```
1  LDD   R24 , Y+ i   load subkey
2  LD    R25 , X      load state
3  EOR   R24 , R25    exclusive OR
4  STD   Z+i  , R24   store result
```

+ **Fautes sur la sortie d'un AES**
*Correspondant à la valeur de la dernière clé de ronde*

- **Plusieurs modèles de fautes** peuvent expliquer ce résultat
- Comment déterminer **quelle instruction** a été perturbée ?
- Est-ce une **instruction** ou bien une **donnée** qui a été fautée ?

➔ Comment peut-on arriver à **définir un modèle des fautes réalisées** ?

# SOMMAIRE

# MICROCONTRÔLEUR UTILISÉ

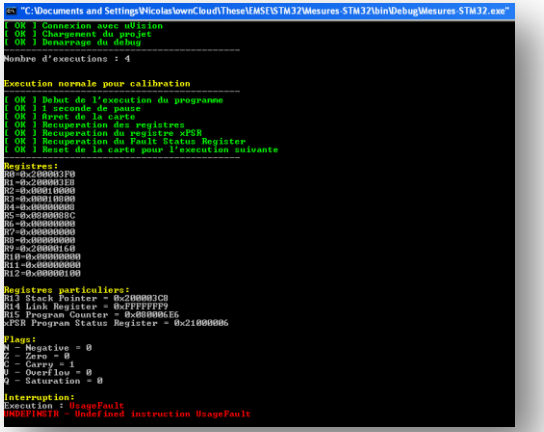
## Microcontrôleur à base de cœur ARM Cortex-M3

- Cœur **ARM Cortex-M3**
- Alimentation par port USB
- 128 Kio de Flash et 8 Kio de RAM
- Fréquence 8MHz, réglable jusqu'à 72MHz
- Jeu d'instruction RISC 16/32 bits **THUMB2**
- Architecture Harvard modifiée ARMv7-M
- Liaison SWD pour le **debug du microcontrôleur**

| Fondeurs | Modèles |
|---|---|
| **Atmel** | AT91 Smart ARM Microcontrollers |
| **NXP Semiconductors** | LPC1300, LPC1700, LPC1800 |
| **STMicroelectronics** | STM32 |
| **Silicon Laboratories** | SiM3C1xx |

# DÉTAIL DU PROTOCOLE EXPÉRIMENTAL

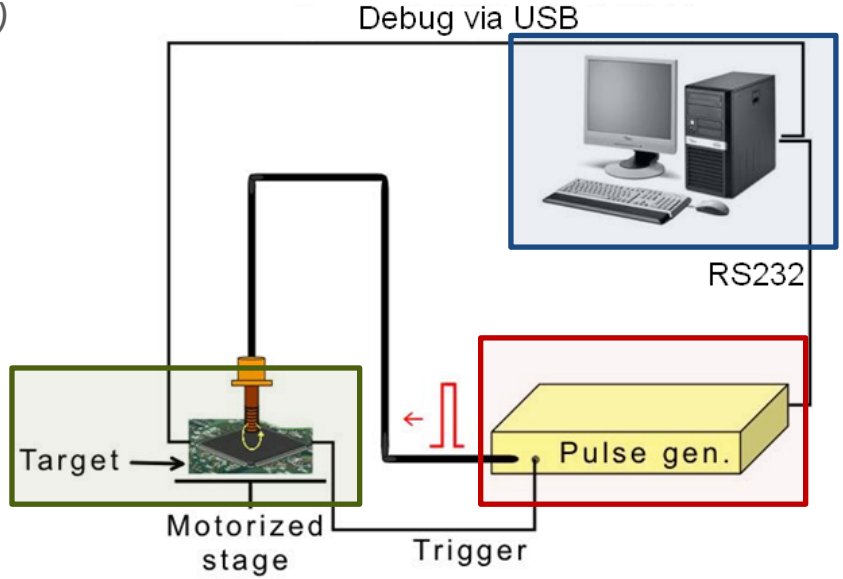
## Mise en place d'un nouveau protocole expérimental




- Expérimentation **pilotée par le PC**
- Exécution d'une **opération** sur la carte
- Envoi d'un **pulse de tension**
  *(signal de déclenchement envoyé par le µC,*
  *pulse envoyé à travers une antenne d'injection au-dessus du µC)*
- **Arrêt** de la carte
- **Récupération** de l'état du microcontrôleur par le PC
  *(registres, compteur de programme, résultat, interruptions)*
- Analyse des résultats obtenus

## Principaux paramètres expérimentaux

- ***Position*** de l'antenne
- ***Durée*** du pulse
- ***Instant*** d'injection du pulse
- ***Code exécuté*** sur le microcontrôleur



# SOMMAIRE

# ANALYSE DE L'ÉTAT INTERNE DU µC



## Récupération de grandeurs internes du microcontrôleur à la suite d'une injection de faute visant à comprendre l'effet des fautes

| Donnée récupérée | Détail |
|---|---|
| **R0 à R12** | Registres généraux |
| **R13 (SP)** | Pointeur de pile |
| **R14 (LR)** | Pointeur de retour de fonction |
| **R15 (PC)** | Compteur de programme |
| **XPSR** | Program Status Register<br>- Flags<br>- Détails sur les interruptions<br>- Détails sur le mode d'exécution |
| **Résultat** | Case mémoire qui contient le résultat du calcul |
| **Nombre de cycles** | Cycles effectués par le microcontrôleur pour le calcul |

# SIMULATION D'UN MODÈLE DE FAUTES



## Simulation (par modèle de saut d'instruction)

| IT | Ligne_assembleur | Detail_IT | Adresse | R0 | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|---|---|---|---|
| Thread_mode | | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x3 | 0x2 | 0x400 |
| Thread_mode | 0x08000224 6803 LDR r3,[r0,#0x00] | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x7[FAUTE] | 0x2 | 0x400 |
| Thread_mode | 0x08000226 4423 ADD r3,r3,r4 | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x2[FAUTE] | 0x2 | 0x400 |
| Thread_mode | 0x08000228 6003 STR r3,[r0,#0x00] | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x2[FAUTE] | 0x2 | 0x400 |
| Thread_mode | 0x0800022A F1010101 ADD r1,r1,#0x01 | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x4[FAUTE] | 0x2 | 0x400 |
| Thread_mode | 0x0800022E 2902 CMP r1,#0x02 | None | None | 0x2000040c | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x400 |
| Thread_mode | 0x08000230 DBF6 BLT 0x08000220 | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x3 | 0x2 | 0x400 |
| Thread_mode | 0x08000220 F8524021 LDR r4,[r2,r1,LSL #2] | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x2[FAUTE] | 0x1[FAUTE] | 0x400 |
| Thread_mode | 0x08000224 6803 LDR r3,[r0,#0x00] | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x3 | 0x2 | 0x400 |
| Thread_mode | 0x08000226 4423 ADD r3,r3,r4 | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x1[FAUTE] | 0x2 | 0x400 |

## Mesures expérimentales

| Instant | Exec | IT | Ligne_assembleur | Detail_IT | Adres | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 37600 | 1 | Thread_mode | None | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x3 | 0x2 | 0x40010c10 | 0x40010c14 | 0x100 |
| 37600 | 2 | Thread_mode | None | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x3 | 0x2 | 0x40010c10 | 0x40010c14 | 0x100 |
| 37600 | 3 | Thread_mode | None | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x3 | 0x2 | 0x40010c10 | 0x40010c14 | 0x100 |
| 37600 | 4 | Thread_mode | None | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x3 | 0x2 | 0x40010c10 | 0x40010c14 | 0x100 |
| 37600 | 5 | Thread_mode | None | None | None | 0x2000040c | 0x2 | 0x20000421 | 0x2[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x40010c14 | 0x100 |

- Possibilité de **tester très rapidement sur tout le code**
- Deux lignes **égales** ⇔ **R0 à R12 + XPSR + PC + SP + résultat** égaux

# SIMULATION D'UN MODÈLE DE FAUTES



## Exemple de simulation de remplacement d'instruction 16 bits

| Execution | IT | Ligne_assembleur | Detail_IT | Adresse | R0 | R1 | R2 | R3 | R4 | R5 | R6 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Thread_mode | | None | None | 0x200004f0 | 0x2 | 0x20000421 | 0x3 | 0x2 | 0x40010c10 | 0x40010c14 |
| 0 | Thread_mod | 0x20000B0F 0000 MOVS r0,r0 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x40010c14 |
| 1 | Thread_mod | 0x20000B0F 0001 MOVS r1,r0 | None | None | 0x200004f0 | 0x200004f0[F | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x40010c14 |
| 2 | Thread_mod | 0x20000B0F 0002 MOVS r2,r0 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x200004f0[F | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x40010c14 |
| 3 | Thread_mod | 0x20000B0F 0003 MOVS r3,r0 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x20000421 | 0x200004f0[F | 0x1[FAUTE] | 0x40010c10 | 0x40010c14 |
| 4 | Thread_mod | 0x20000B0F 0004 MOVS r4,r0 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x200004f0[F | 0x40010c10 | 0x40010c14 |
| 5 | Thread_mod | 0x20000B0F 0005 MOVS r5,r0 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x200004f0[F | 0x40010c14 |
| 6 | Thread_mod | 0x20000B0F 0006 MOVS r6,r0 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x200004f0[F |
| 7 | Thread_mod | 0x20000B0F 0007 MOVS r7,r0 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x40010c14 |
| 8 | Thread_mod | 0x20000B0F 0008 MOVS r0,r1 | None | None | 0x1[FAUTE] | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x40010c14 |
| 9 | Thread_mod | 0x20000B0F 0009 MOVS r1,r1 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x40010c14 |
| 10 | Thread_mod | 0x20000B0F 000A MOVS r2,r1 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x1[FAUTE] | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x40010c14 |
| 11 | Thread_mod | 0x20000B0F 000B MOVS r3,r1 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x40010c14 |
| 12 | Thread_mod | 0x20000B0F 000C MOVS r4,r1 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x40010c14 |
| 13 | Thread_mod | 0x20000B0F 000D MOVS r5,r1 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c14 |
| 14 | Thread_mod | 0x20000B0F 000E MOVS r6,r1 | None | None | 0x200004f0 | 0x1[FAUTE] | 0x20000421 | 0x1[FAUTE] | 0x1[FAUTE] | 0x40010c10 | 0x1[FAUTE] |

- Bien plus long pour un test exhaustif sur toutes les instructions
- Deux lignes **égales** ⇔ **R0 à R12 + XPSR + résultat** égaux

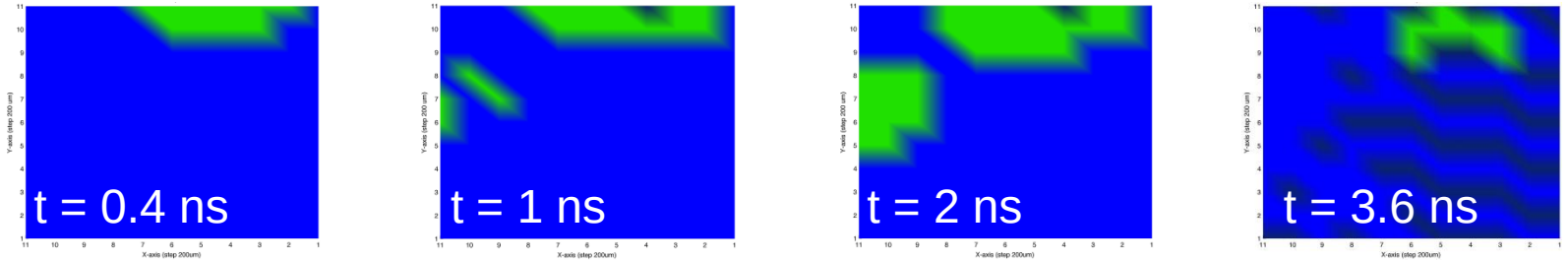
# SOMMAIRE

# INFLUENCE DE LA POSITION D'ANTENNE

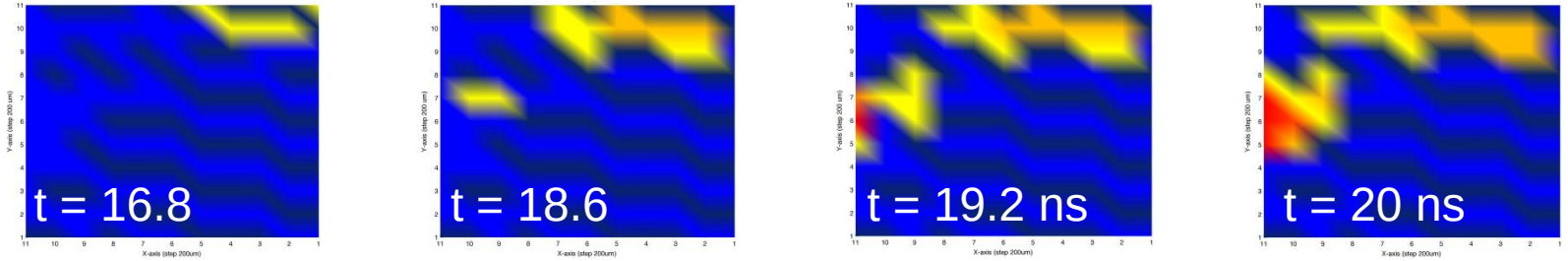


## Exemple de cartographie spatiale et temporelle

- Vert : déclenchement d'interruptions
- Rouge : fautes sur le résultat




**Fréquence 56 MHz – Période 17ns**




Interrupt triggered

Fault on the output value

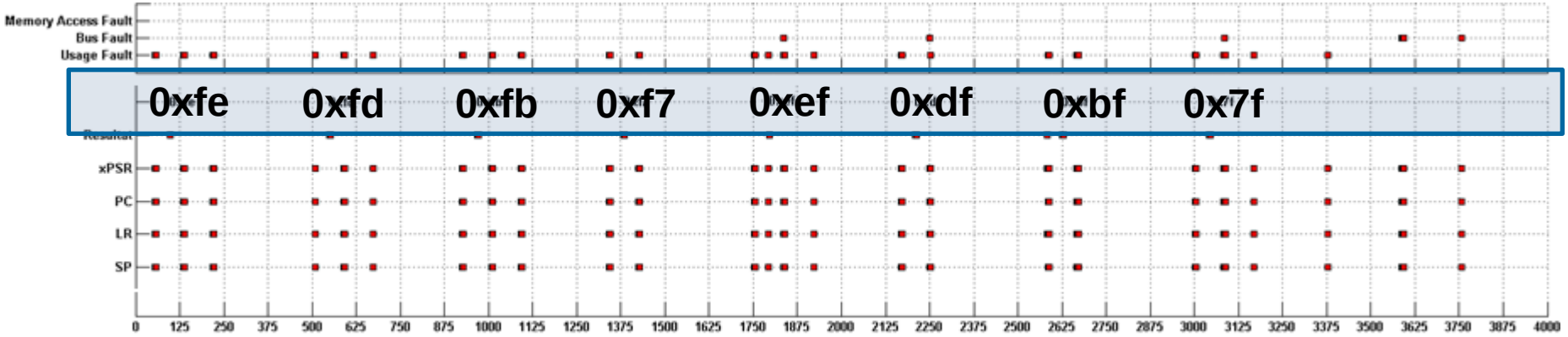
Crash of the microcontroller

No fault on the output value

# INFLUENCE DE L'INSTANT D'INJECTION

## Exemple de cartographie temporelle de l'injection EM

*(pour une addition de puissances de deux dont la somme vaut normalement 0xFF)*




**Saut d'instruction avec une assez forte probabilité de réalisation**

```
08000770 <fonctionTest>:
 8000770:       b570            push    {r4, r5, r6, lr}
 8000772:       4604            mov     r4, r0
 8000774:       460e            mov     r6, r1
 8000776:       2201            movs    r2, #1
 8000778:       0211            lsls    r1, r2, #8
 800077a:       480a            ldr     r0, [pc, #40]   ; (80007a4 <fonctionTest+0x34>)
 800077c:       f7ff fdf5       bl      800036a <GPIO_WriteBit>
 8000780:       2500            movs    r5, #0
 8000782:       e005            b.n     8000790 <fonctionTest+0x20>
 8000784:       7820            ldrb    r0, [r4, #0]
 8000786:       5d71            ldrb    r1, [r6, r5]
 8000788:       4408            add     r0, r1
 800078a:       b240            sxtb    r0, r0
 800078c:       7020            strb    r0, [r4, #0]
 800078e:       1c6d            adds    r5, r5, #1
 8000790:       2d08            cmp     r5, #8
 8000792:       dbf7            blt.n   8000784 <fonctionTest+0x14>
 8000794:       2200            movs    r2, #0
 8000796:       f44f 7180       mov.w   r1, #256        ; 0x100
 800079a:       4802            ldr     r0, [pc, #8]    ; (80007a4 <fonctionTest+0x34>)
 800079c:       f7ff fde5       bl      800036a <GPIO_WriteBit>
 80007a0:       bd70            pop     {r4, r5, r6, pc}
 80007a2:       0000            .short  0x0000
 80007a4:       40010c00        .word   0x40010c00
```

### Différentes observations

- **Non-exécution d'une instruction assembleur**
- Déclenchement d'exceptions matérielles
- Effet **déterministe** pour une configuration donnée

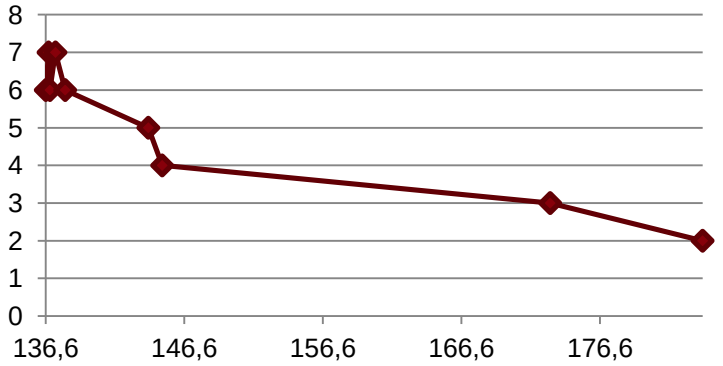
# FAUTES SUR LE FLOT DE DONNÉES



**LDR R3, PC#40** à 40.66ns avec 0x33330000 à l'adresse PC#40

| Amplitude du pulse | Valeur de sortie |
|---|---|
| 136.6V | **7B33** 0000 |
| 136.8V | **7333** 0000 |
| 136.9V | **7B33** 0000 |
| 137.3V | **3B33** 0000 |
| 138V | **7B33** 0000 |
| 144V | **7F33** 0000 |
| 145V | **FF33** 0000 |
| 173V | **FFB3** 0000 |
| 184V | **FFF3** 0000 |

Distance de Hamming entre la valeur fautée et 0xFFFF




- Simulation : ne correspond à **aucun remplacement d'instruction**
- Semble s'orienter vers une **mise à 1** lors des **transferts de données**
- Probable **précharge du bus de données à 1** sur cette architecture

# FAUTES SUR LE FLOT DE CONTRÔLE



- Test avec une suite de **NOP** (BF 00)

- **Trois types de faute**
  - Faute sur **R7**
  - Non-arrêt du programme
  - UsageFault – Undefined Instruction

- **R7 et R15 registres particuliers**
  - **R7** codé en **111** dans un opcode 16 bits
  - **R15** codé en **1111** dans un opcode 32 bits
  - **R15 program counter**

- Parfois modification du **nombre de cycles exécutés** (augmentation ou diminution)

| NOP - BF 00 | NOP - BF 00 |
|---|---|
| 1011 1111 0000 0000 | 1011 1111 0000 0000 |

```
0x0800021E 6037      STR      r7,[r6,#0x00]
   206:         POP           {R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12}
0x08000220 E8BD1FFE  POP      {r1-r12}
   207:         BX            LR
0x08000224 4770      BX       lr
0x08000226 0000      DCW      0x0000
0x08000228 0004      DCW      0x0004
0x0800022A 2000      DCW      0x2000
0x0800022C 0C10      DCW      0x0C10
0x0800022E 4001      DCW      0x4001
0x08000230 0C14      DCW      0x0C14
0x08000232 4001      DCW      0x4001
0x08000234 02A0      DCW      0x02A0
0x08000236 2000      DCW      0x2000
0x08000238 0304      DCW      0x0304
0x0800023A 0102      DCW      0x0102
0x0800023C 5678      DCW      0x5678
0x0800023E 1234      DCW      0x1234
0x08000240 4321      DCW      0x4321
0x08000242 8765      DCW      0x8765
    90:   while (1)
0x08000244 BF00      NOP
0x08000246 E7FE      B        0x08000246
   128: }
   129:
   130: /**
   131:   * @brief  This function handles PendSV_Handler exception.
   132:   * @param  None
   133:   * @retval None
   134:   */
   135: void PendSV_Handler(void)
   136: {
0x08000248 4770      BX       lr
   334:   for(; nCount != 0; nCount--);
```

# PLUSIEURS POSSIBILITÉS PHYSIQUES



- **Couplage directement sur les lignes de bus**
  → ***Peu probable**, sinon on arriverait également à fauter le bus d'adresse*

- **Couplage sur les rails d'alimentation ou le plan de masse**
  → ***Plausible** : ralentirait le transfert des données sur le bus*

- **Couplage sur l'arbre d'horloge**
  → ***Plausible** : provoquerait un coup d'horloge plus court*

**Référence bibliographique sur les fautes de délai**

**Investigation of timing constraints violation as a fault injection means**
**2012 -** Loïc Zussa, Jean-Max Dutertre, Jessy Clédière, Bruno Robisson, Assia Tria
*27th Conference on Design of Circuits and Integrated Systems (DCIS*). Avignon

# BILAN SUR LE MODÈLE DE FAUTES DÉFINI



## ➔ Possibilité de fauter les transferts depuis la mémoire Flash

**Conséquences au niveau des instructions**

- **Remplacement** d'instructions
- **Saut** d'instructions assembleur
- Certaines instructions possiblement **plus sensibles que d'autres**
- Certains registres qui semblent **plus sensibles que d'autres**

**Conséquences au niveau des données**

- Corruption des `LOAD` depuis la mémoire Flash (clé de chiffrement, …)
- Valeurs à poids de Hamming élevé plus faciles à obtenir (sur cette architecture)

Article accepté à FDTC 2013 sur l'élaboration du modèle de fautes

# SOMMAIRE

I. Généralités sur l'injection électromagnétique

II. Présentation du montage expérimental

III. Approche pour évaluer les effets obtenus

IV. Résultats expérimentaux d'injection EM

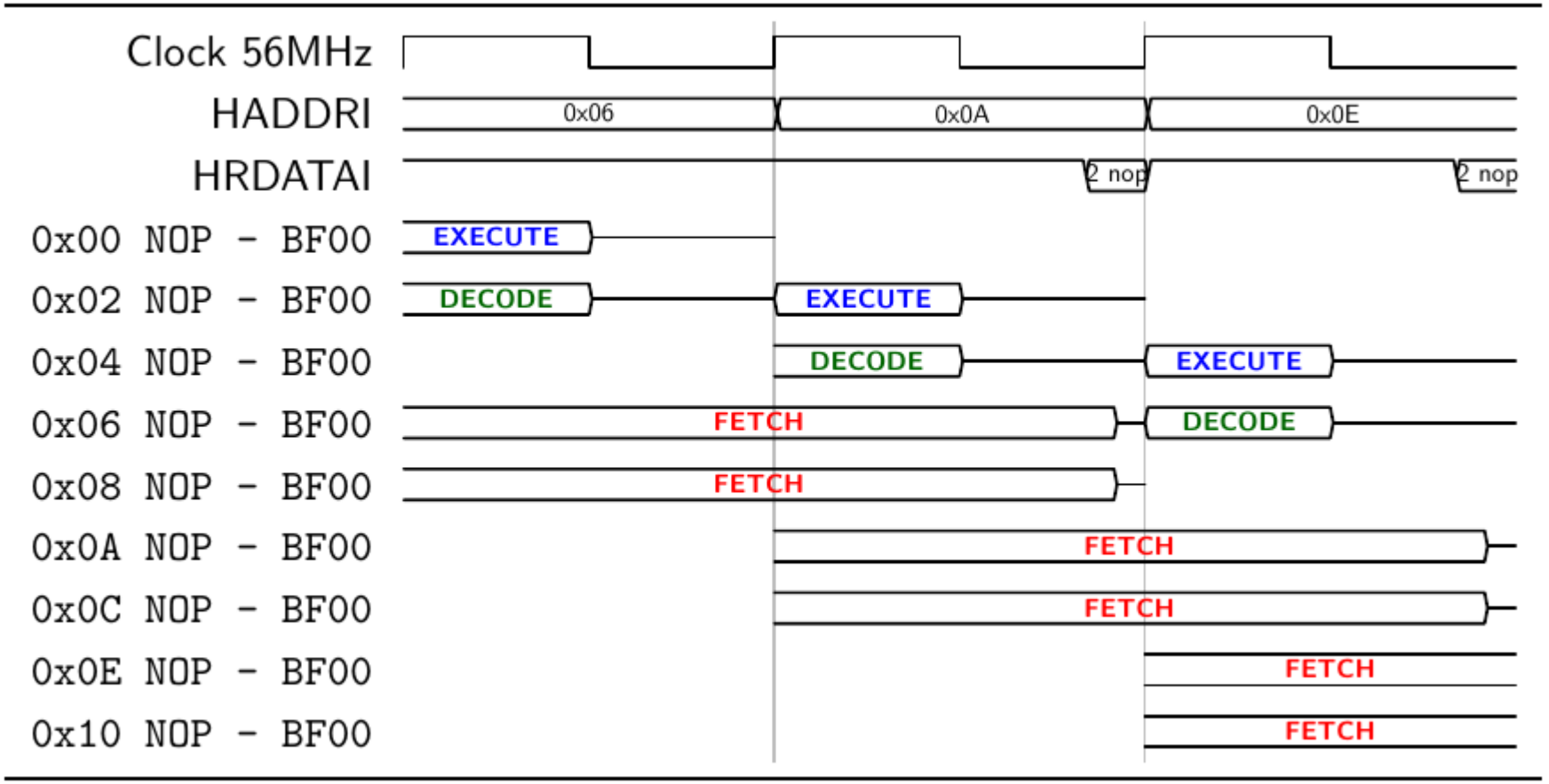
V. Modèle de fautes au niveau RTL

VI. Contre-mesures et preuve formelle sur le code

# FETCH D'INSTRUCTION



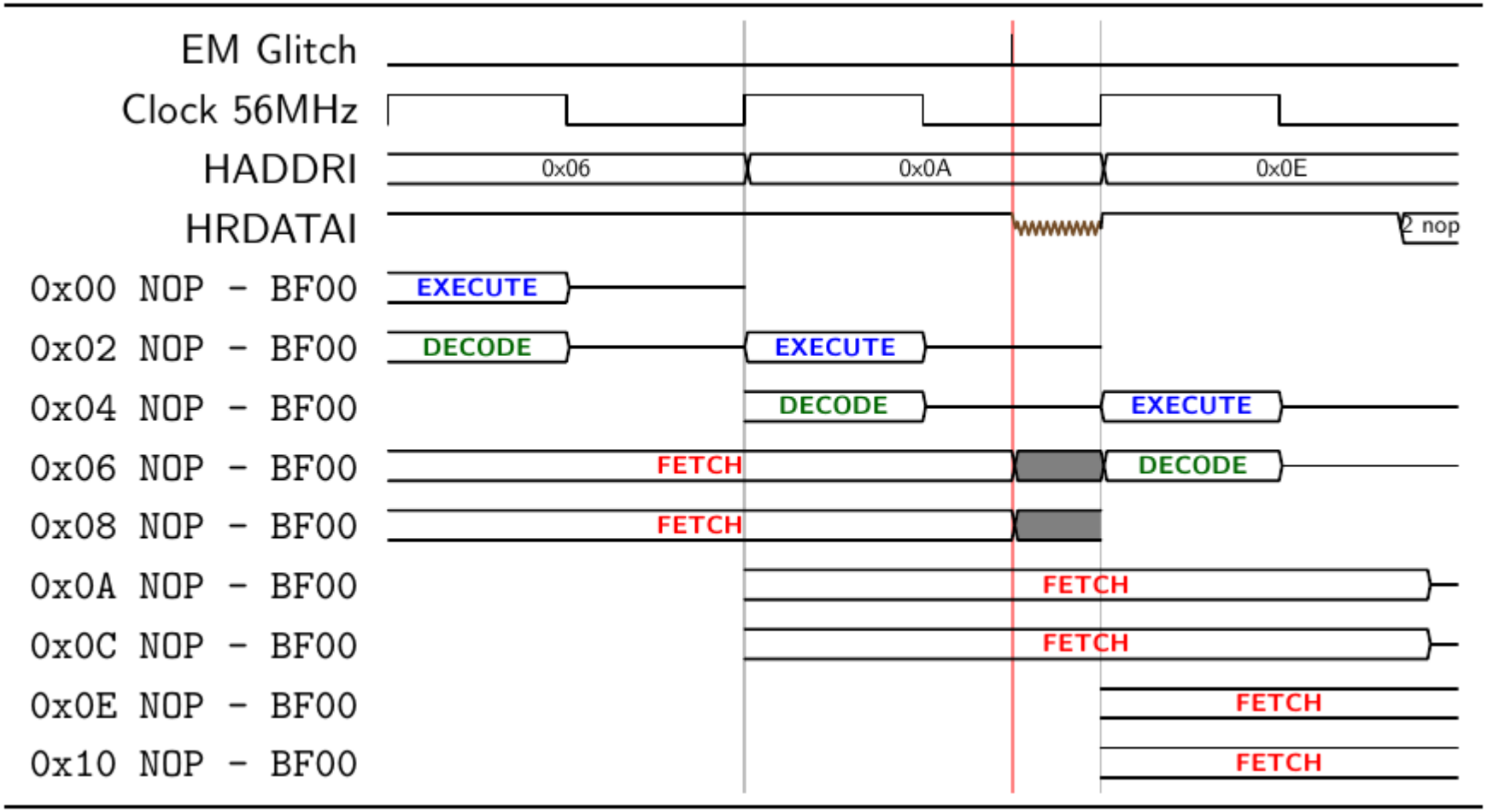
## Fonctionnement normal



# FETCH D'INSTRUCTION

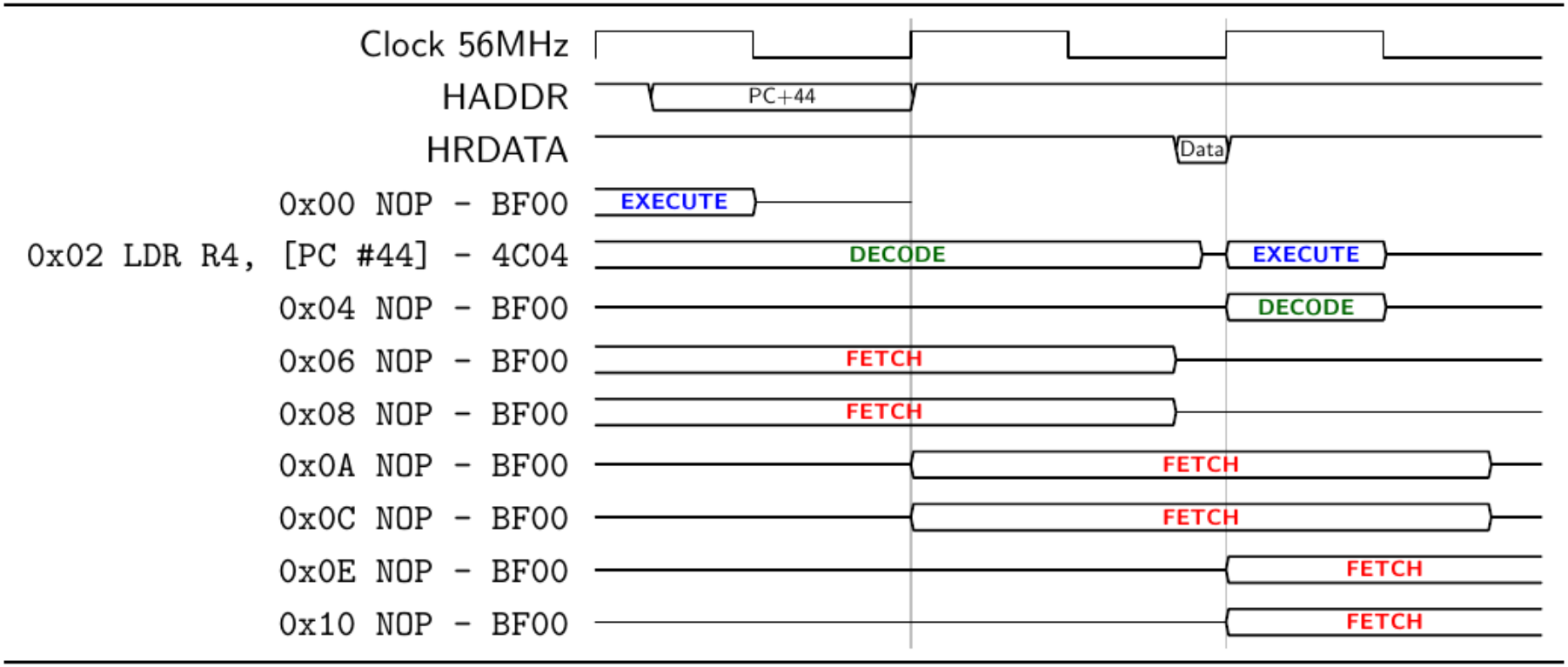


## Avec glitch électromagnétique



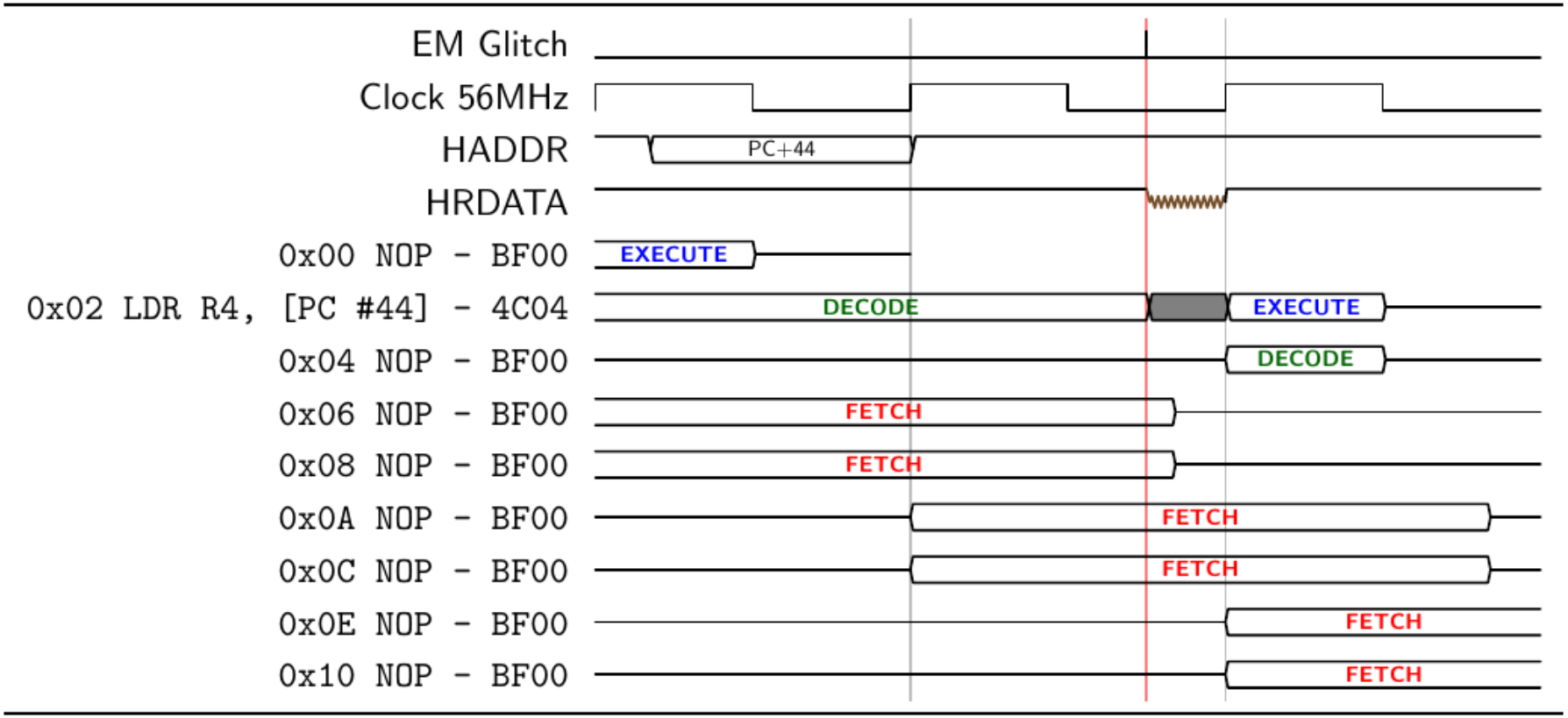
# DONNÉE DEPUIS LA MÉMOIRE FLASH

## Fonctionnement normal



# DONNÉE DEPUIS LA MÉMOIRE FLASH

## Avec glitch électromagnétique



# SOMMAIRE

I. Généralités sur l'injection électromagnétique

II. Présentation du montage expérimental

III. Approche pour évaluer les effets obtenus

IV. Résultats expérimentaux d'injection EM

V. Modèle de fautes au niveau RTL

VI. Contre-mesures et preuve formelle sur le code

# ADDITION DE TABLEAU

- Addition de **puissances de deux** qui utilise une **structure de boucle**
- Injection électromagnétique sur **toute la longueur de la boucle** par pas de 100ps
- **Simulation** du saut de chaque instruction et comparaison par rapport aux mesures

## Code source C

```
void addition_tableau(char tab[], int *resultat)
{
  int i;

  for(i=0; i<2; i++)
  {
    *resultat += tab[i];
  }

}
```

*resultat = 0
tab[0] = 1
tab[1] = 2
tab[2] = 4

**Résultat normal : 3**

## Assembleur

```
; opération d'addition des éléments
boucle_addition

    LDR     R4, [R2,R1, lsl #2] ; r4 = tab[i] (exactement *(&tab[0]+i)
    LDR     R3, [R0,#0]         ; r3 = res
    ADD     R3, R4              ; r3 = r3 + r4
    STR     R3, [R0,#0]         ; *(r0) = r3
    ADD     R1, R1, #1          ; r1 = r1 + 1
    CMP     R1, #2              ; r1 == 2 ?
    BLT     boucle_addition     ; si non, reboucle
    NOP

; trigger à 0
    STR     R7, [R6, #0]        ; trigger à 0

; pour la manip
    LDR     R3, [R0,#0]

; fin du programme
    POP     {R3,R4,R5,R6,R7,R8}
    BX      LR
```

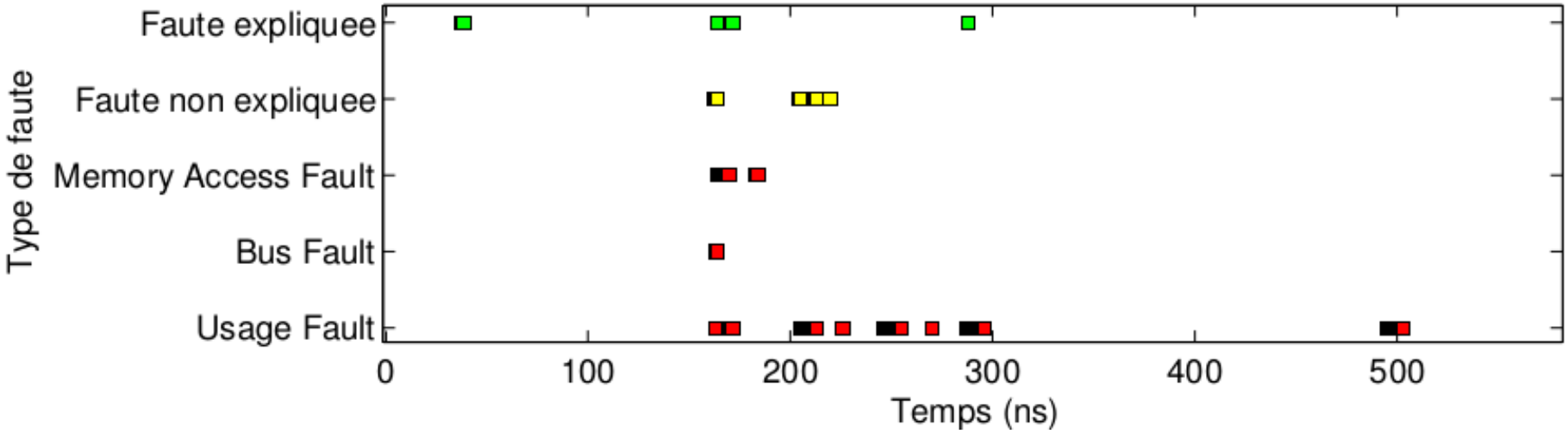
# MODÈLE SIMPLE DE SAUT D'INSTRUCTION



| | |
|---|---|
| Lignes avec un résultat fauté | **127** |
| Etats de sortie différents | **16** |
| Valeurs de sortie différentes | **7** |
| Lignes expliquées | **46** |
| Lignes de simulation qui expliquent un résultat de mesure | **2** |

**Sorties différentes :**

'0x2' '0xd1080002' '0x0' '0x5710800' '0x1' '0x3' '0x7'



# CHEMINS D'ATTAQUE PERMIS PAR LE MODÈLE

- Beaucoup de **remplacements d'instruction** ont un effet équivalent à celui d'un **saut d'instruction**

- **Large ensemble** de possibilités d'attaque selon l'algorithme

  (non-exécution d'un appel de fonction, saut d'une opération ALU, saut du stockage d'un résultat, …)

Modèle de plus haut niveau choisi pour les contre-mesures :
➔ **remplacement d'une instruction par un NOP**

- **Problématique** : comment **garantir une exécution correcte** avec un potentiel saut d'instruction par un attaquant ?

Article présenté à COSADE 2013 sur l'exploitation d'un de ces chemins d'attaque

# CODE TOLÉRANT AUX FAUTES DU MODÈLE

- Structure de code à base de **duplication d'instruction**
- Trois principales **classes d'instructions** dans notre ISA
- Proposer une **séquence de remplacement** par instruction

1. **Instructions idempotentes** (`MOV, CMP, B, ...`)
   ➔ Simple duplication de l'instruction
2. **Instructions séparables** (`ADD R1, R1, R2, ...`)
   ➔ Séparation en plusieurs instructions, utilisation de registres disponibles
3. **Instructions particulières** (`PUSH, POP, BL, ...`)
   ➔ Séquence de remplacement spécifique à chaque instruction

# CODE TOLÉRANT AUX FAUTES DU MODÈLE



- Surcoût important mais application à **tout code assembleur**
- Prouver l'**équivalence en sortie** entre :

Un **code standard** non sécurisé
*sans injection de faute*

```
ADD   R1, R1, #1
CMP   R1, #9
B     <label>
```

Un **code sécurisé avec duplication**
*en présence d'une injection de faute*

```
ADD   R3, R1, R1
ADD   R3, R1, R1
MOV   R1, R3
MOV   R1, R3
CMP   R1, #9
CMP   R1, #9
B     <label>
B     <label>
```

Preuve par **model-checking** à l'aide de l'outil **Vis**
visant à **garantir l'équivalence des deux machines d'états**

# CONCLUSION ET PERSPECTIVES

- Premier **modèle de fautes** pour l'injection EM sur microcontrôleur
- Perturbation des **transferts depuis la mémoire Flash** sur les bus
- **Effets similaires** obtenus précédemment sur une autre architecture
  (microcontrôleur Atmel AVR 8 bits ATmega128)
- Possibilité de réaliser des **sauts d'instruction** sous certaines conditions
- Première approche de **contre-mesures par duplication d'instructions**

**Perspectives**
- Caractériser les remplacements d'instruction à l'aide d'une **sonde de trace**
- Proposer des bouts de code **prouvés sûrs** contre le modèle
- Essayer **expérimentalement** d'attaquer ces bouts de code
- Proposer une approche plus fine pour la **duplication**