

Comment deviner les secrets de votre carte à puce ?

Application à la Java CardTM



Damien SAUVERON

sauveron@labri.u-bordeaux.fr

<http://dept-info.labri.u-bordeaux.fr/~sauveron>

Plan

La carte à microprocesseur et ses
sécurités

Les canaux cachés

La rétro-conception

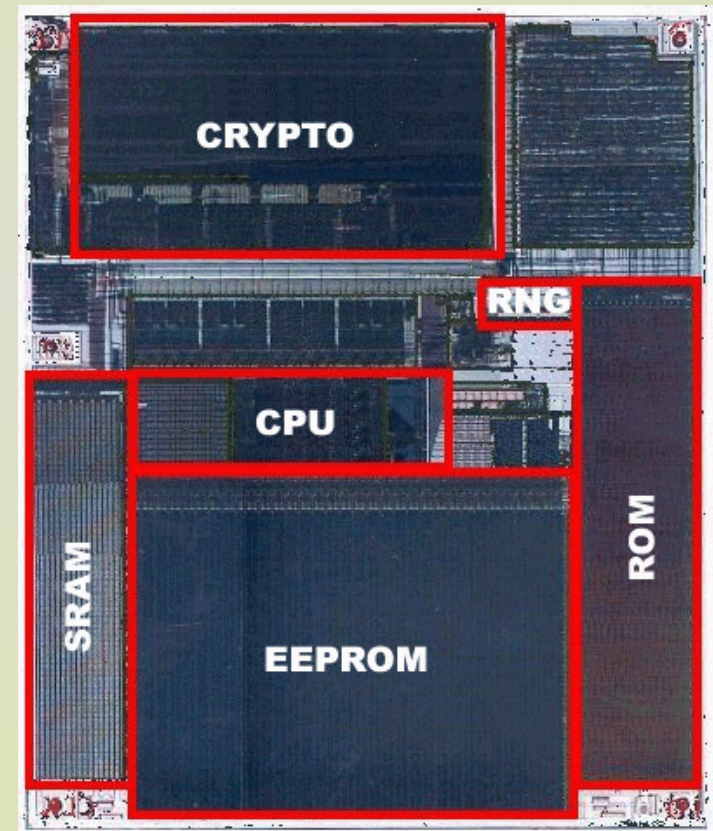
Application à la Java Card

Perspectives

La carte à microprocesseur

Taille de la puce : 25mm^2

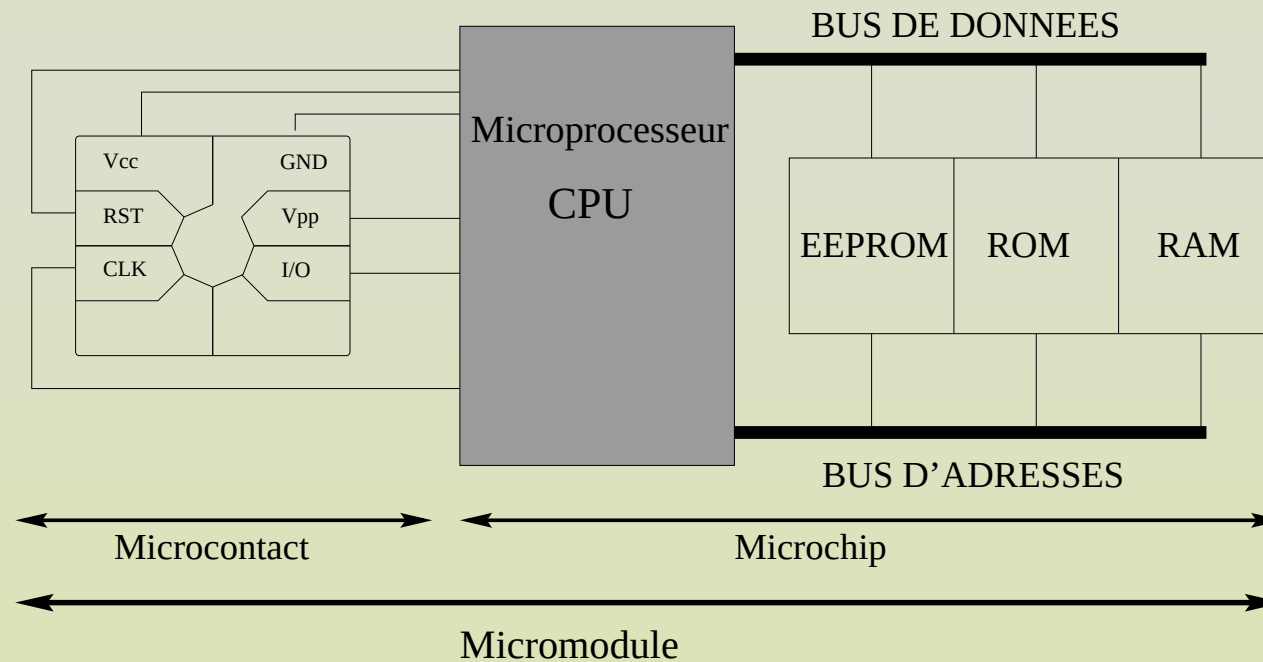
- ➡ **Microprocesseur** (CPU) : 8, 16 ou 32 bits (à architecture RISC ou pas)
- ➡ **ROM** : 16 à 24 Ko
- ➡ **EEPROM** : 8 à 64 Ko
- ➡ **RAM** : 1 Ko
- ➡ Coprocesseur cryptographique
- ➡ Générateur de nombres aléatoires (RNG)



La carte à contact

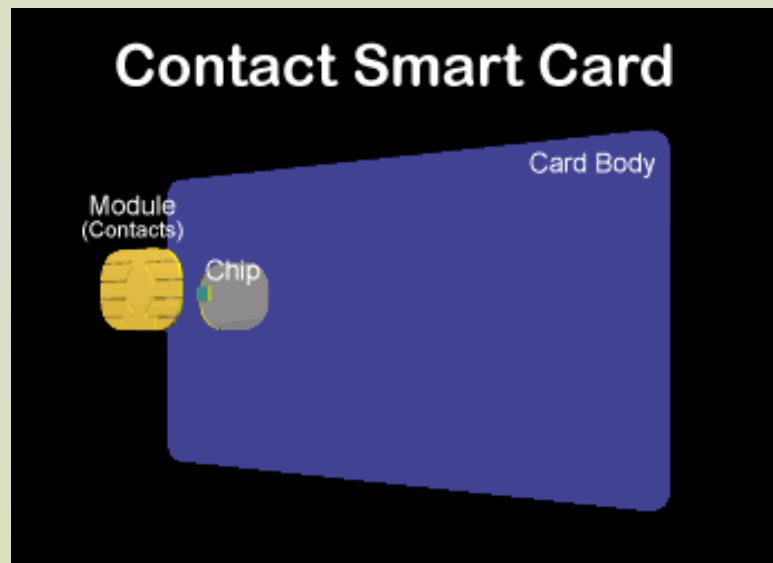
Son fonctionnement, l'oblige à être insérée dans un lecteur de carte.

Elle utilise une communication série via huit contacts.



La carte à contact

Suit le standard ISO 7816.



Problèmes :

- ☞ l'insertion et le retrait sont des facteurs d'usure de la carte
- ☞ il faut bien orienter la carte dans le lecteur.

Les sécurités de la carte à puce

➡ *au niveau physique* : techniques d'impression sophistiquées

➡ *au niveau hardware* :

- un numéro de série unique
- l'utilisation de mémoire de type PROM
- blindage physique du composant
- des détecteurs de conditions anormales (température,...)
- brouillage des informations dans le composant
- co-processeurs cryptographiques

Les sécurités de la carte à puce

☞ *au niveau software :*

- contrôles d'accès aux données
- maintien de l'intégrité des données
- entrées/sorties sécurisées

☞ *au niveau de l'environnement de production* qui doit être physiquement sécurisé

Les canaux cachés

👉 le temps d'exécution

⇒ nombre de cycle d'une instruction ou d'un algorithme.

👉 la consommation de courant

⇒ Les modifications rapides de la tension et de l'intensité du courant au sein du même composant sont à la base des émissions du circuit car ils conduisent des courants RF à l'intérieur et à l'extérieur du chip.

👉 les émissions électromagnétiques

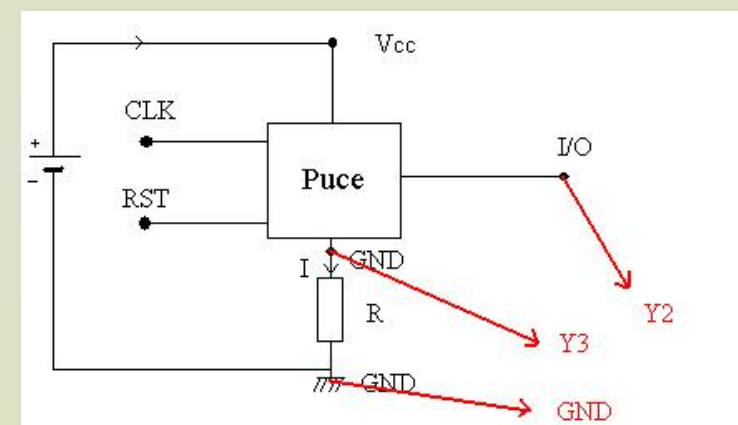
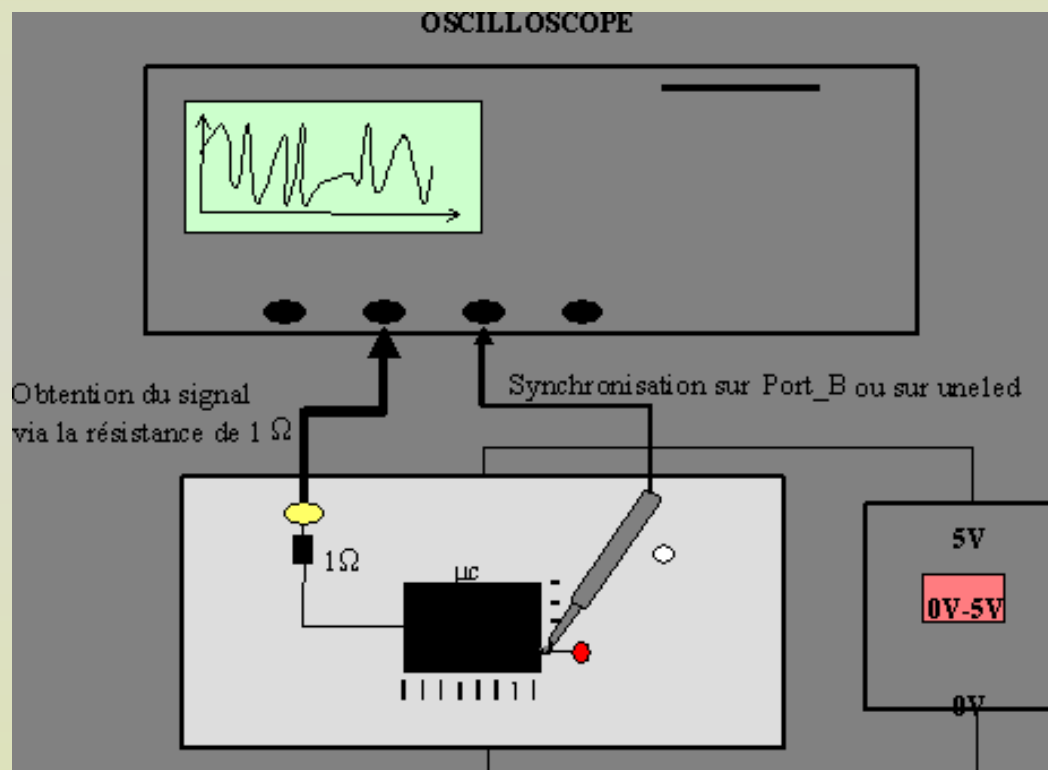
⇒ Les courants RF entraînent un rayonnement électromagnétique.

Le matériel

- ☞ un oscilloscope numérique,
- ☞ un lecteur de carte à puce,
- ☞ un pc équipé de cartes d'acquisition et de logiciels mathématique pour le traitement des données,
- ☞ une sonde CEM si on veut étudier les émissions électromagnétiques.



Le montage



La “timing attack”

Cette attaque consiste à mesurer le temps d’exécution d’un algorithme.
 ⇒ Révèle des informations sur les opérations et/ou les opérandes.

Nécessite souvent :

- ☞ un grand nombre d’exécution à messages choisis,
- ☞ un traitement statistique des résultats obtenus.

La “timing attack” sur le PIN

```
for (b = 0; i >= 7; i++)
    if (pinCarte[i] != pinPresente[i])
        return false;
return true;
```

- ➡ Présenter les n valeurs de `pinPresente[0]` (256 valeurs).
- ➡ Mesurer la durée d'exécution de la commande τ pour les n valeurs.
- ➡ Calculer $\tau[n_0]$ le maximum des τ
 - $\tau[n_0] = \max(\tau[n]), n = 0, \dots, 255$
 - n_0 est la solution pour `pinPresente[0]`
- ➡ Itérer sur tous les `pinPresente[i]`

Nombre d'essais : $8 * 256 = 2048$ (contre 256^8 en force brute)

La consommation de courant

Elle est surtout utilisée dans le domaine de la cryptographie.

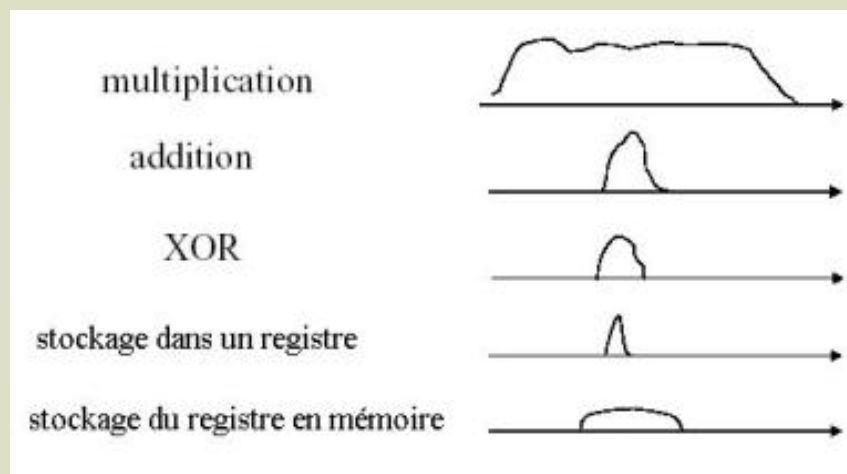
Il existe différentes attaques :

- ➡ la SPA (Simple Power Analysis)
- ➡ la DPA (Differential Power Analysis)
- ➡ la HODPA (High Order Differential Power Analysis)

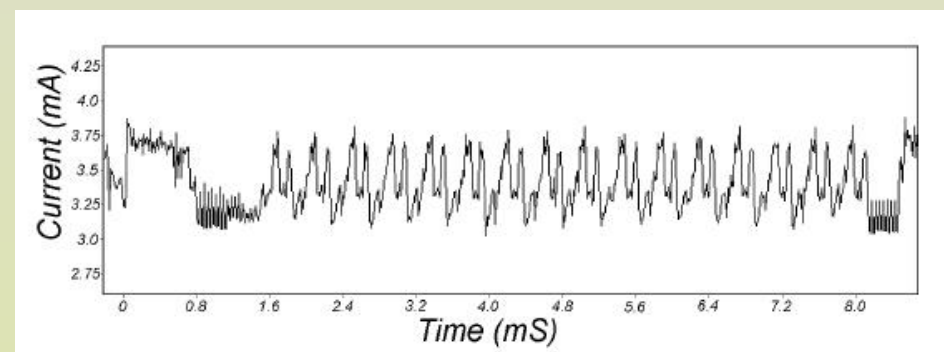
La consommation de courant

La SPA (Simple Power Analysis)

Principe : Des instructions différentes ont une trace différente.



Consommation en courant d'un DES. On peut voir la permutation initiale, suivie des 16 tours.



La consommation de courant

La SPA sur la signature RSA

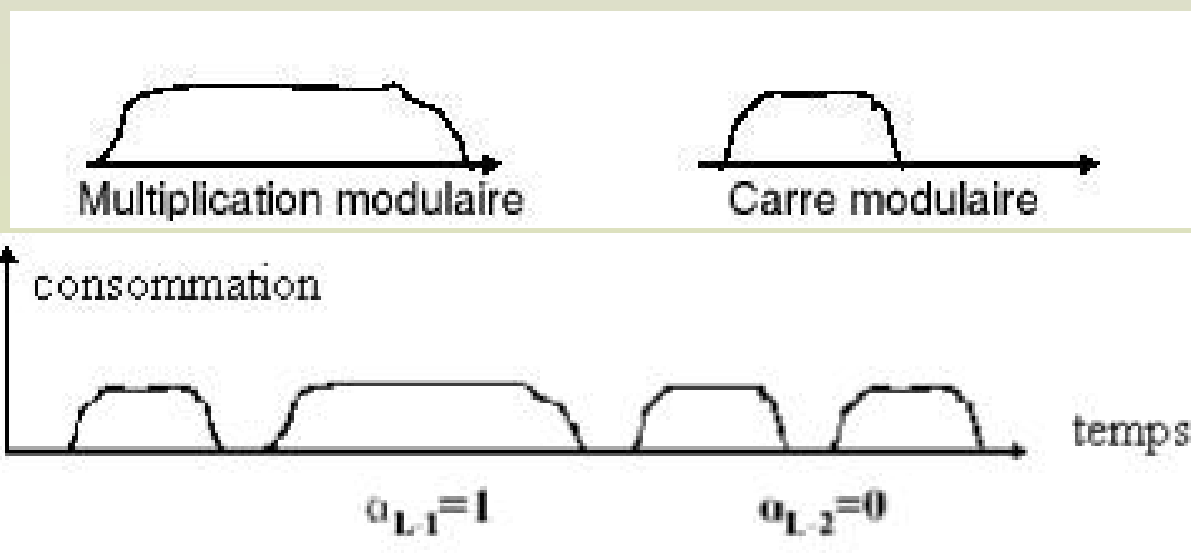
```
s = 1;
for(i = L - 1; i >= 0; i--) {
    s = s*s mod n;
    if (a[i] == 1)
        s = s*y mod n;
}
```

Signature RSA : $y^a \bmod n$

y est le message à signer,

n est public,

a , l'exposant peut être considéré comme la clé secrète.



La consommation de courant

La DPA (Differential Power Analysis)

Principe : La consommation dépend des opérations effectuées (les instructions) mais aussi des opérandes.

Elle utilise des **fonctions statistiques** adaptées à l'algorithme visé qui font ressortir des **corrélations entre un bit** intermédiaire a (ne dépendant que d'un fragment K_r de r bits de la clé et du message d'entrée M) **et la consommation de courant**.

La consommation de courant

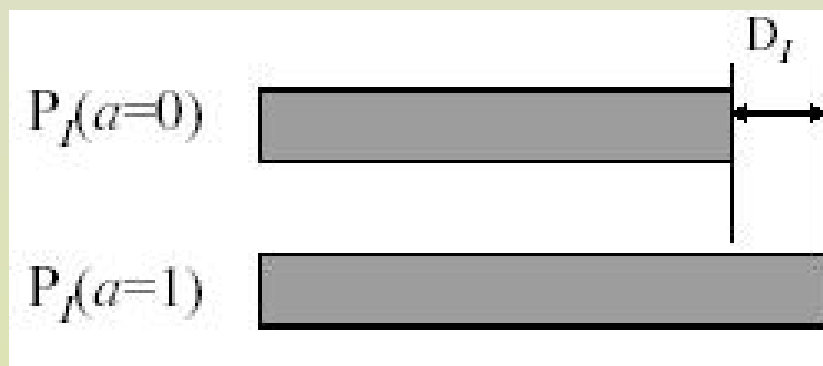
La DPA (Differential Power Analysis)

Préliminaire : $a = f(K_r, M)$ où

K_r est inconnu, M est connu et $f()$ est une fonction de sélection qui peut être déduite de l'algorithme cryptographique connu.

Le bit a apparaît à l'entrée d'une instruction I .

Plus la dépendance entre la consommation de courant pour cette instruction P_I et a sera forte, plus la DPA a de chances de réussir.



La DPA (Differential Power Analysis)

Collecte de données :

- ➡ Faire tourner l'algorithme sur la carte N fois avec N messages d'entrée aléatoires : $M_1, M_2, \dots, M_N$.
- ➡ Pendant les exécutions, on enregistre les consommations de courant $P(K_r, M_1), P(K_r, M_2), \dots, P(K_r, M_N)$.

Analyse de données :

- ➡ Pour chacun des 2^r fragments de clé K_r^* possible, faire tourner l'algorithme N fois avec les **mêmes messages** que précédemment sur une implémentation personnelle, de sorte à pouvoir “observer” le bit a .
- ➡ Séparer les courbes en 2 paquets :

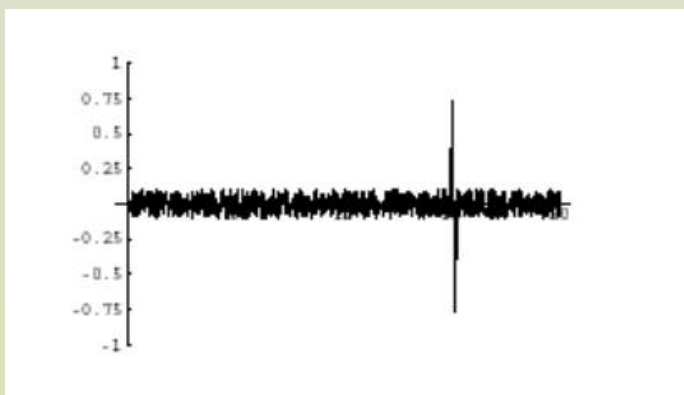
$$E_0 = P(K_r^*, M_i) \text{ tel que } f(K_r^*, M_i) = 0$$

$$E_1 = P(K_r^*, M_i) \text{ tel que } f(K_r^*, M_i) = 1$$

La DPA (Differential Power Analysis)

Si le fragment de clé K_r^* est correct ($K_r^* = K_r$), quand l'instruction I est effectuée, il y aura une différence notable D_I entre les $P(K_r^*, M_i)$ de E_0 et ceux de E_1 .

$$g(t) = \langle P(K_r^*, M_i)(t) \rangle_{P(K_r^*, M_i) \in E_0} - \langle P(K_r^*, M_i)(t) \rangle_{P(K_r^*, M_i) \in E_1}$$



Succès



Échec

La consommation de courant

La DPA (Differential Power Analysis)

Résumé :

On essaie toutes les possibilités pour le fragment de clé K_r^* en regardant à chaque fois le graphe de la fonction g .

Le fragment donnant une fonction g présentant un pic est normalement le bon.

La consommation de courant

La HODPA (High Order DPA)

La HODPA ou DPA d'ordre n est aussi basée sur une étude statistique de la consommation de courant de la carte.

Différence : elle utilise des corrélations entre la consommation de courant et n variables intermédiaires ne dépendant que d'un fragment de la clé et du message d'entrée.

Elle est beaucoup plus difficile à réaliser, mais beaucoup plus puissante.

La CEM (Conformité EM)

Principe : Les courants qui circulent dans la puce induisent des champs électromagnétiques qui sont susceptibles de donner le même type d'information que le courant.

Différence : L'information est plus locale. On peut déplacer la micro-sonde électromagnétique au dessus de la zone qui nous donnera le plus d'informations (exemple : co-processeur cryptographique).

Avantage : Insensible aux contre-mesures physiques tels que l'ajout de bruit en sortie ou le lissage de la consommation globale de courant.

Inconvénient : La reproductibilité des mesures est difficile.

La rétro-conception

Nous aborderons ces quelques travaux :

⇒ méthode du “dictionnaire” :

☞ avec la consommation en courant,

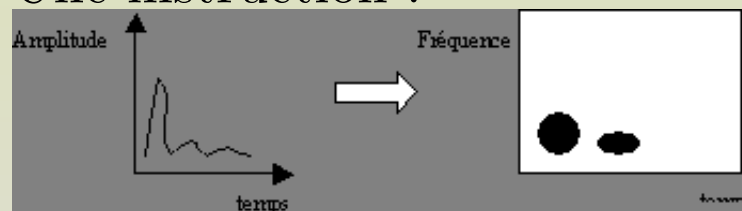
☞ avec la CEM.

⇒ traitement du signal en temps-fréquence et temps-échelle.

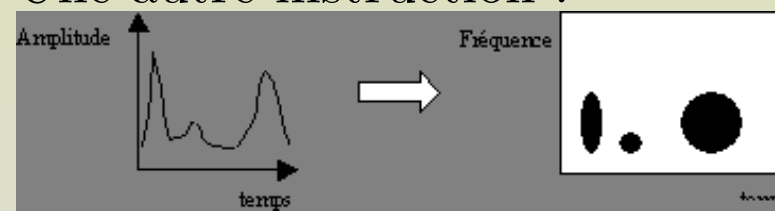
⇒ cartographie avec la CEM.

Le “dictionnaire”

Une instruction :



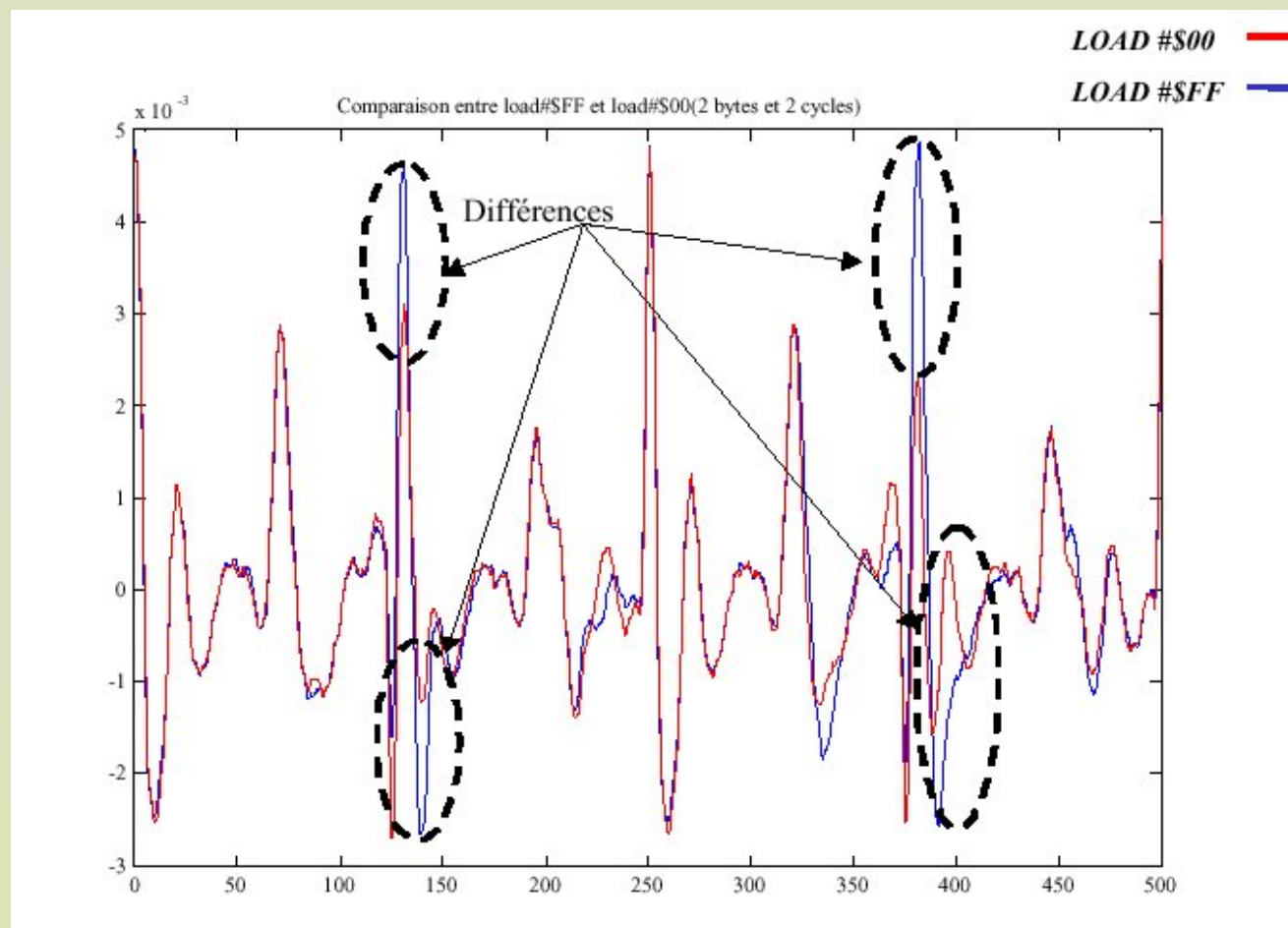
Une autre instruction :



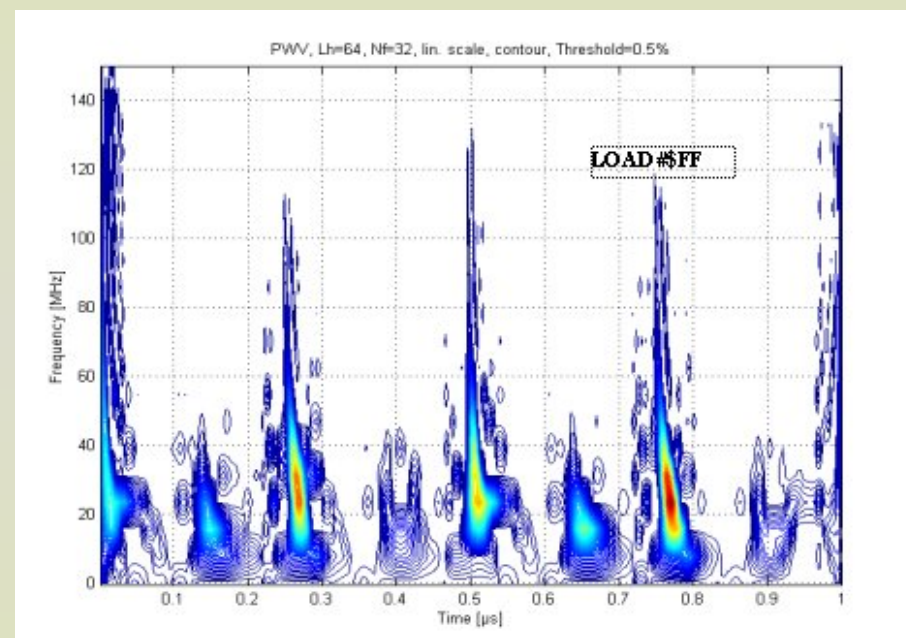
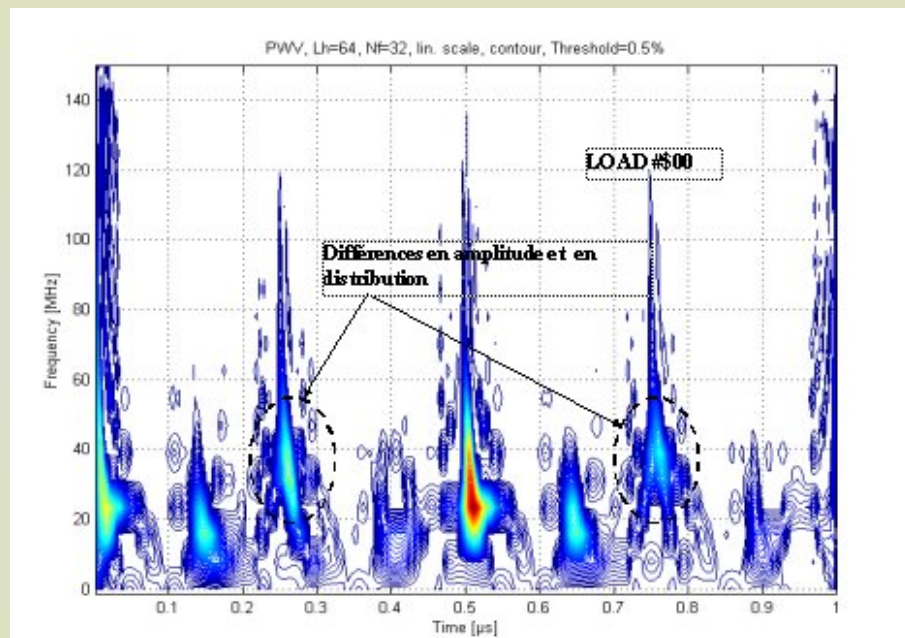
La séquence des deux :



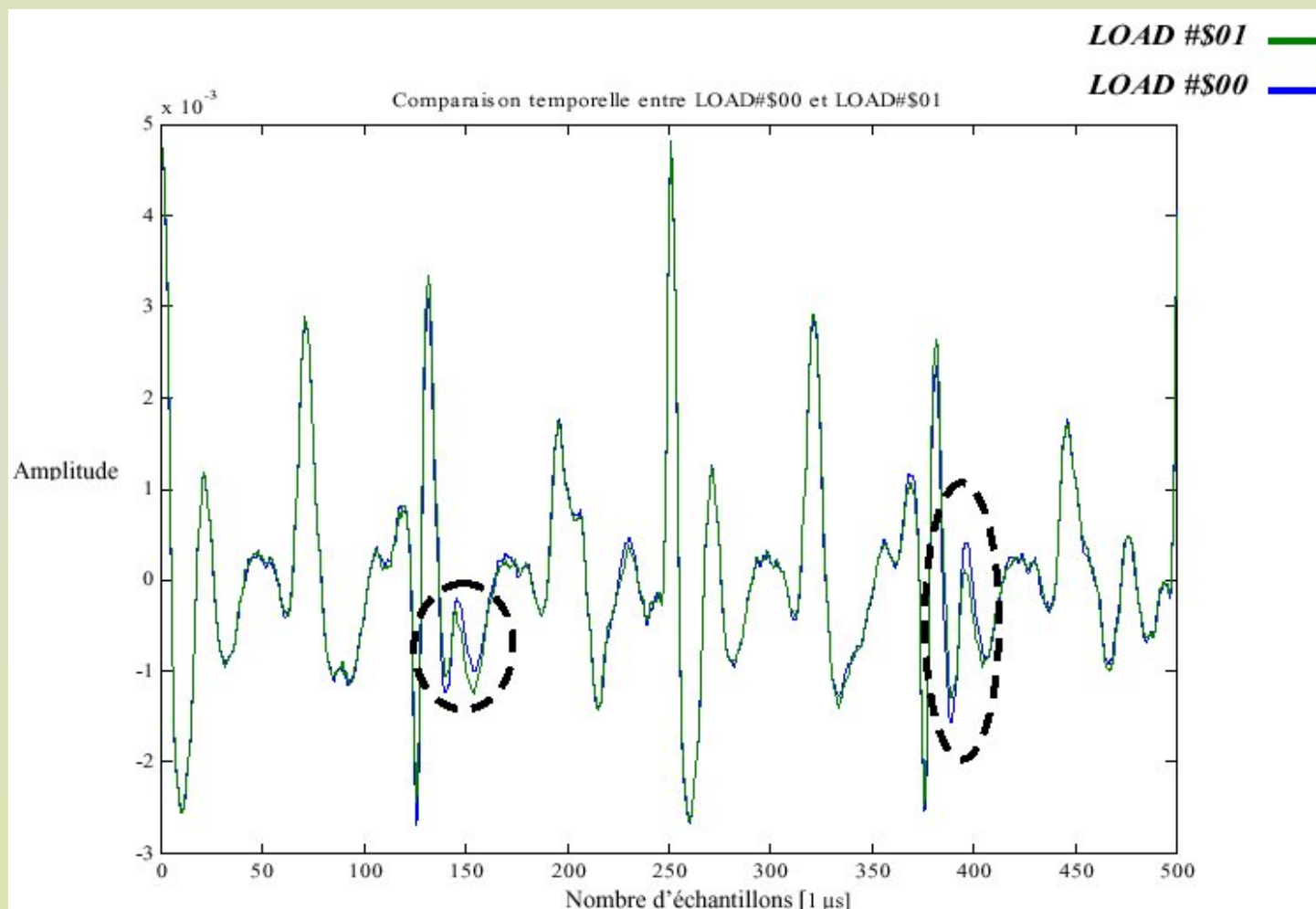
La méthode du temps-fréquence ?



La méthode du temps-fréquence ?

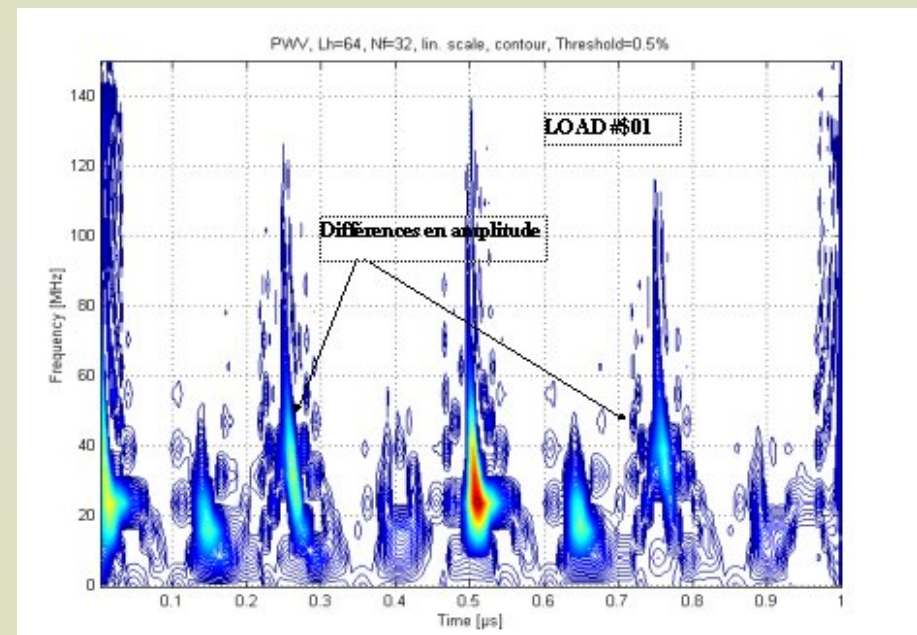
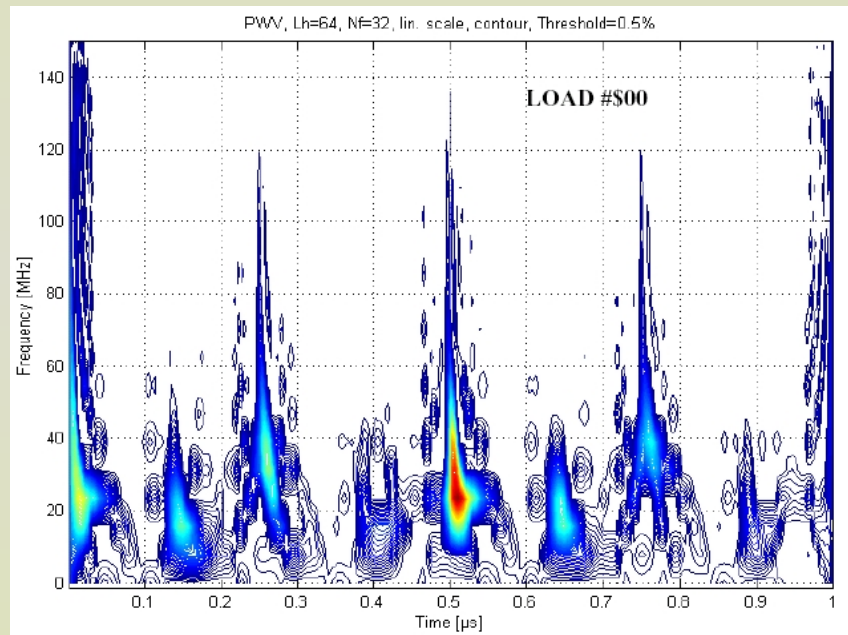


La méthode du temps-fréquence ?



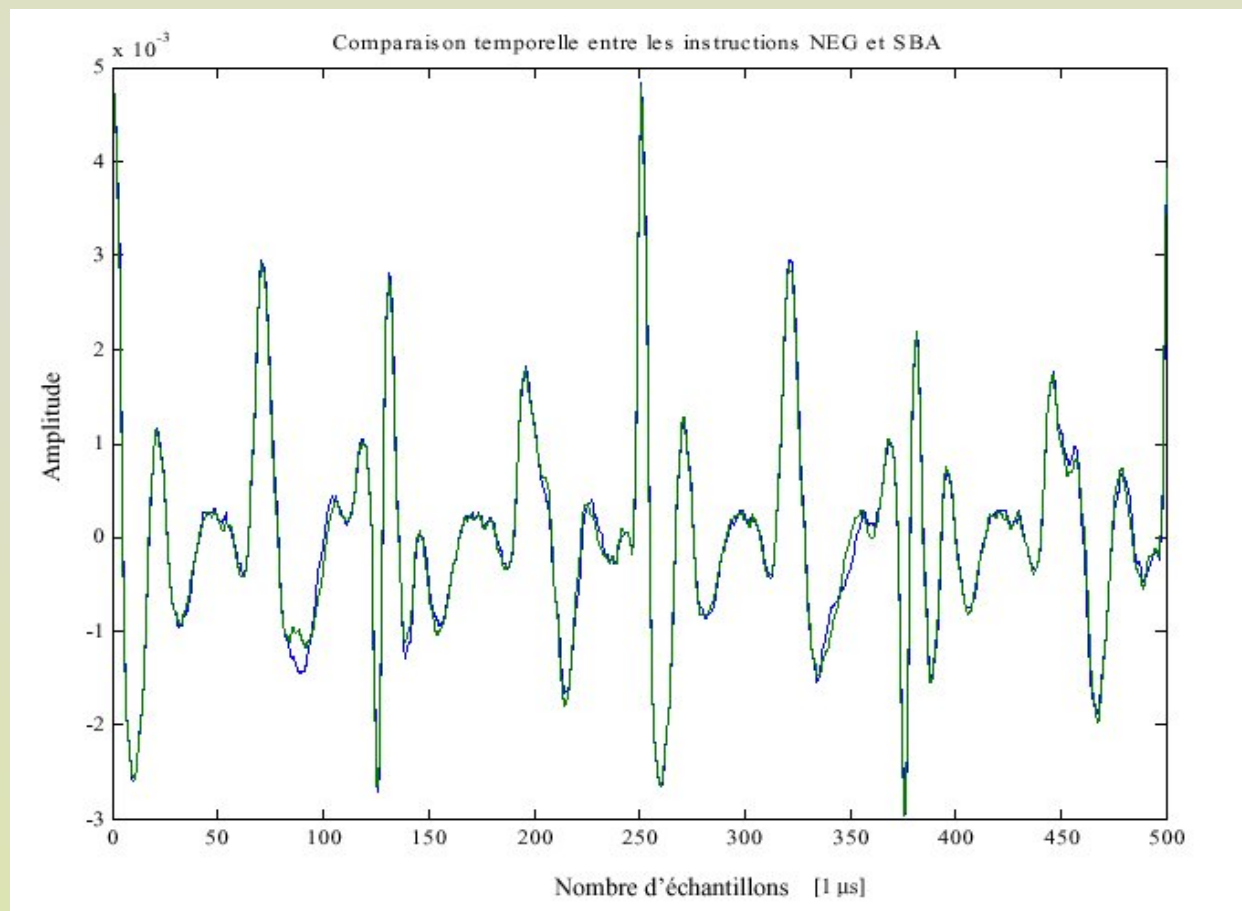
La méthode du temps-fréquence ?

Problème : Difficile de déterminer si on charge un \$00 ou un \$11.



Le dictionnaire ?

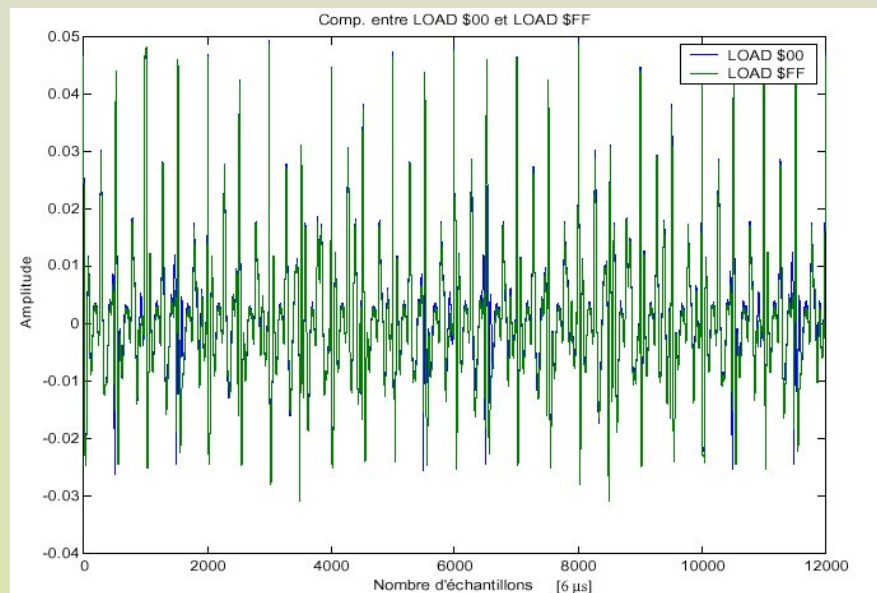
Problème : Certaines intructions très différentes ont la même allure.



On ne se décourage pas !

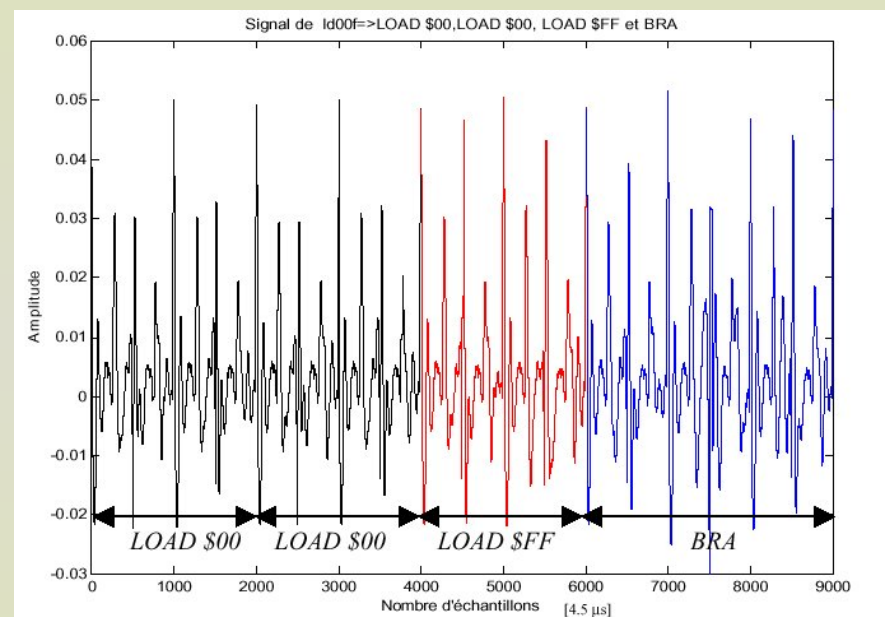
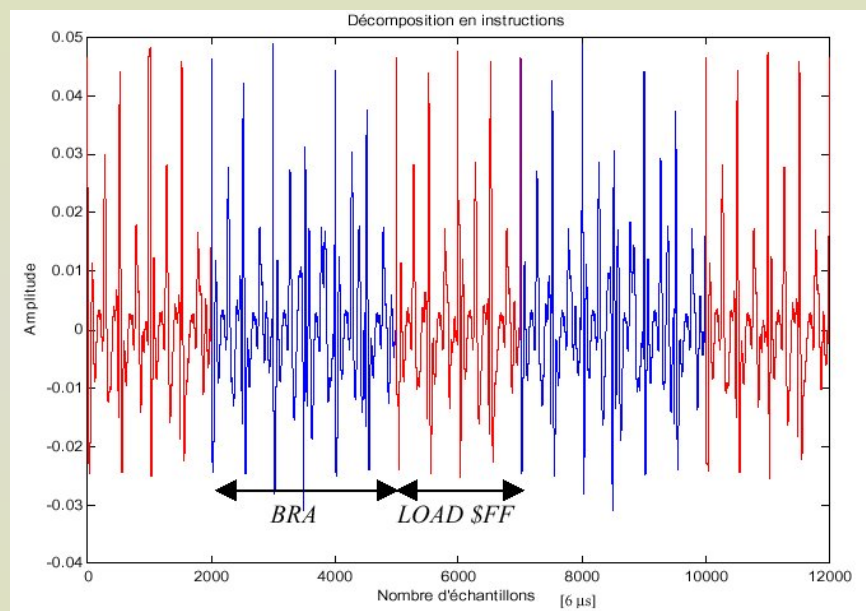
```
LDAB #$40
STAB $1004
LOOP LDAA #$FF (2 cycles)
BRA LOOP (3 cycles)
END
```

```
LDAB #$40
STAB $1004
LOOP LDAA #$00 (2 cycles)
BRA LOOP (3 cycles)
END
```

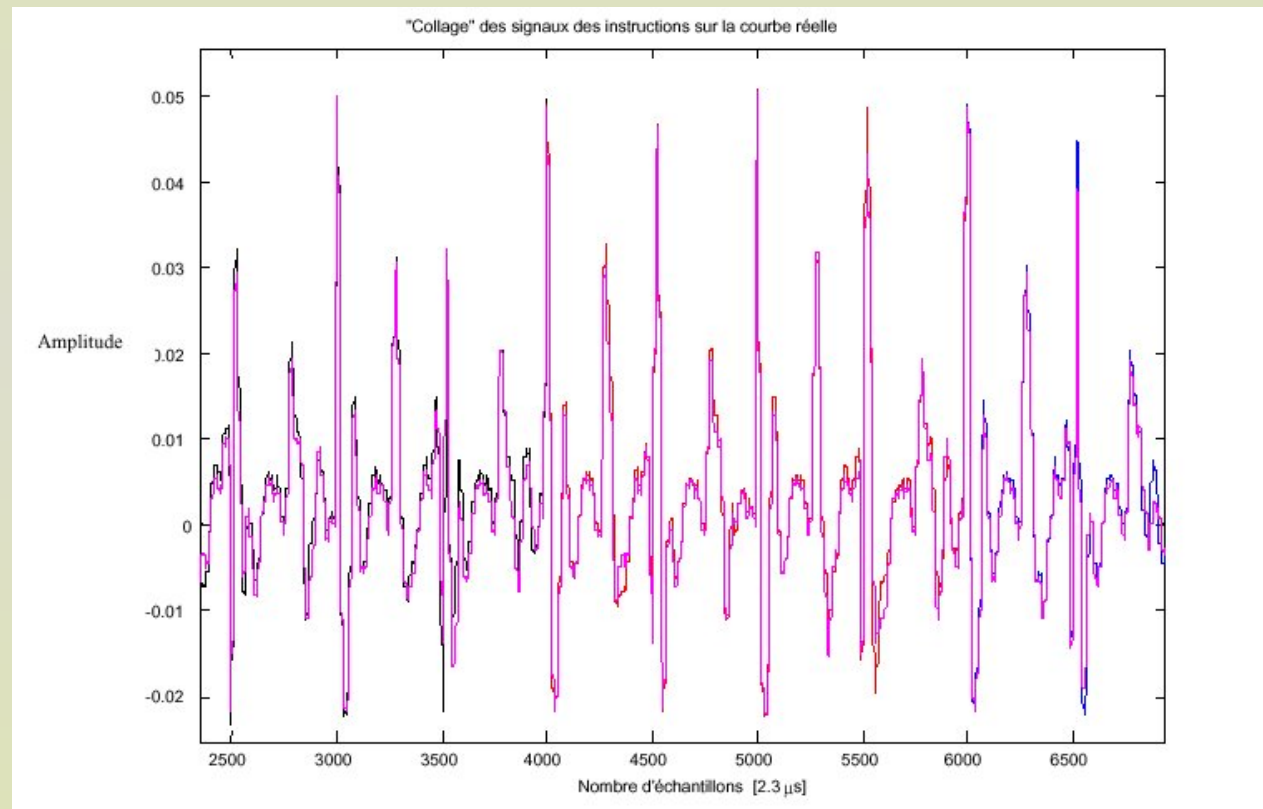


On distingue les instructions BRA.

Ca marche ! On isole des instructions



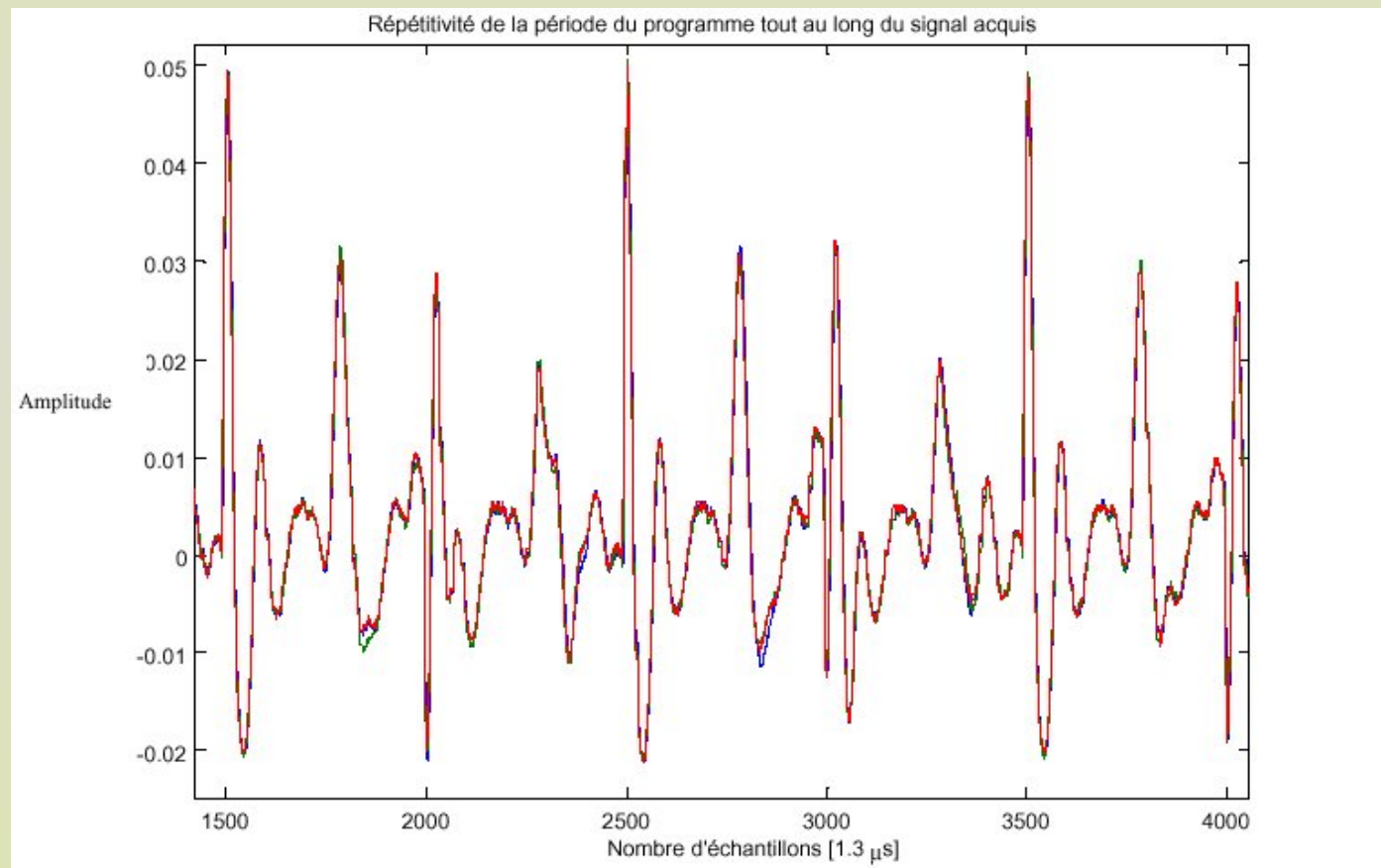
Vérification par collage



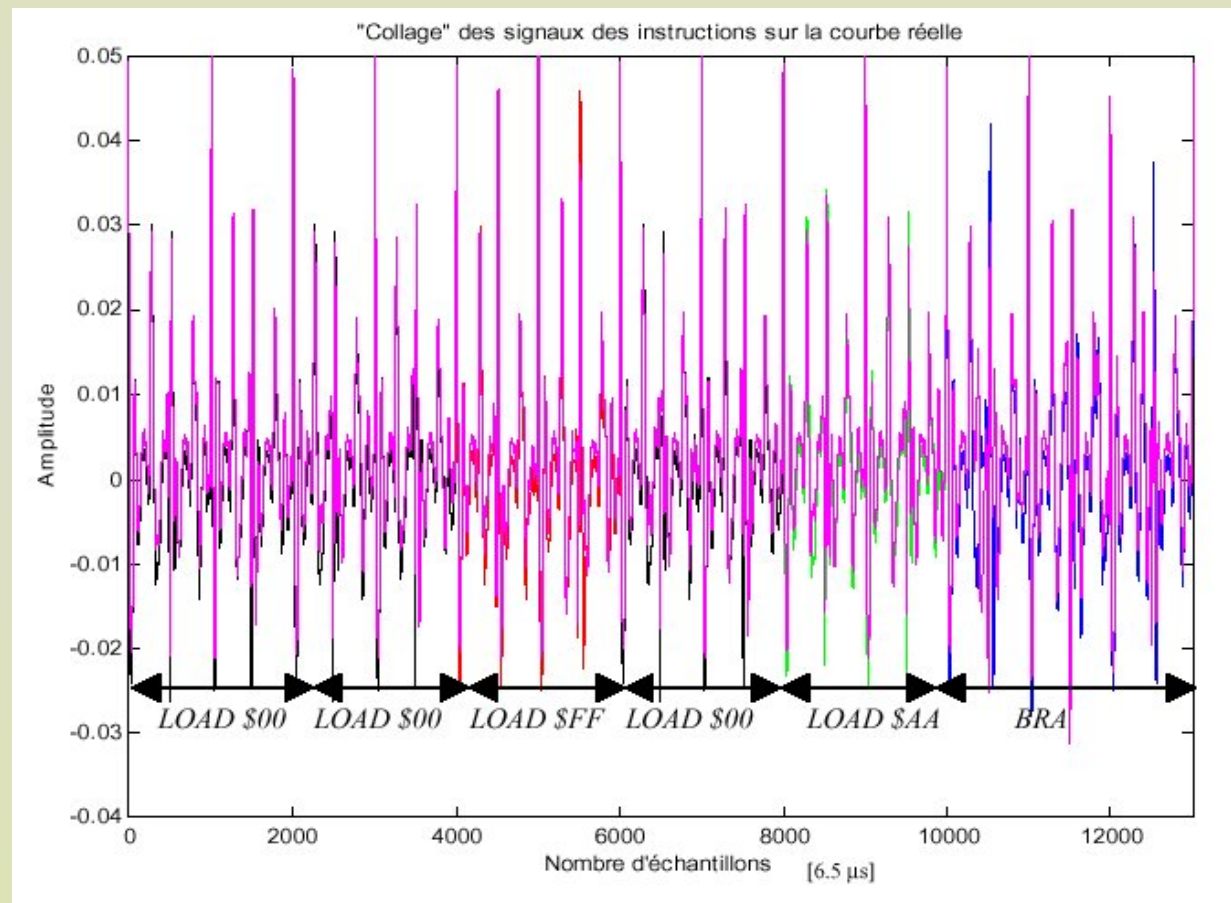
Déception : Pas exactement la même chose.

Hypothèse : Problème de répétitivité ?

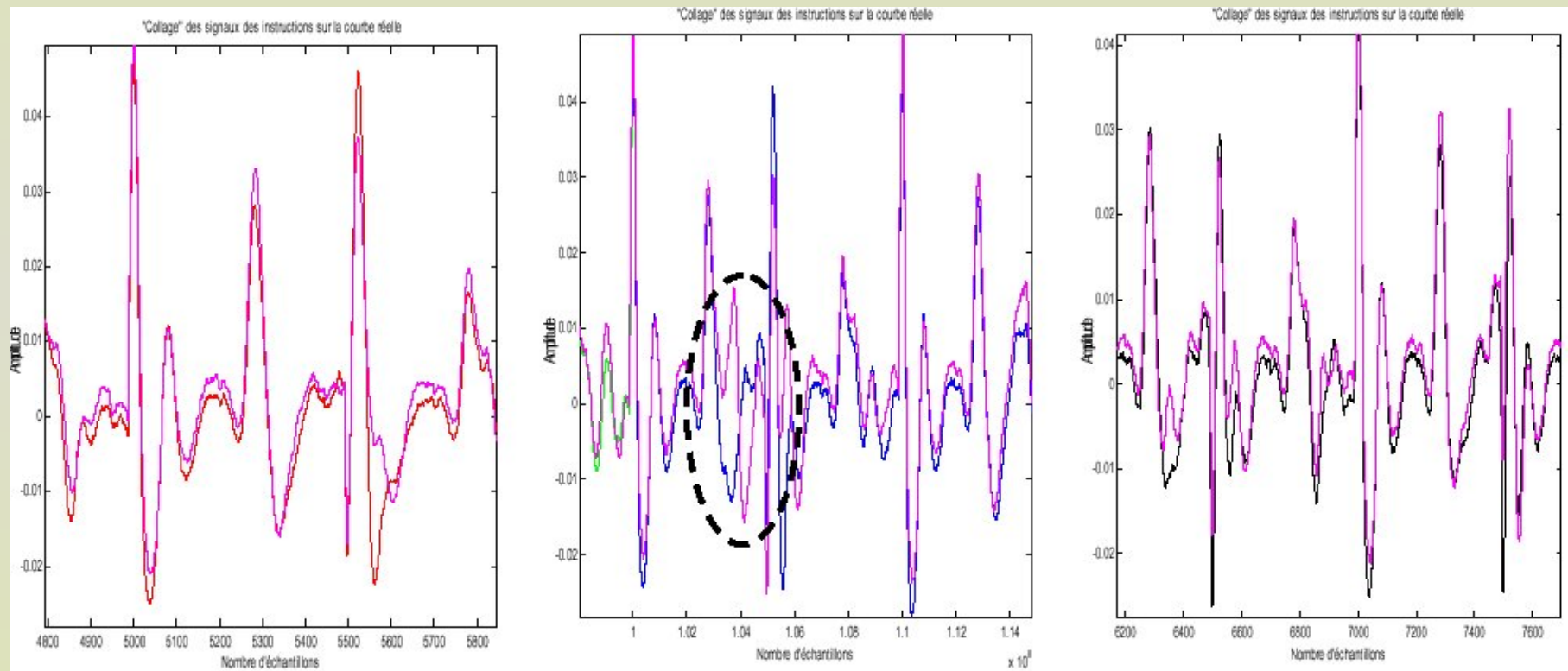
La répétitivité est bonne



On recommence notre collage ...



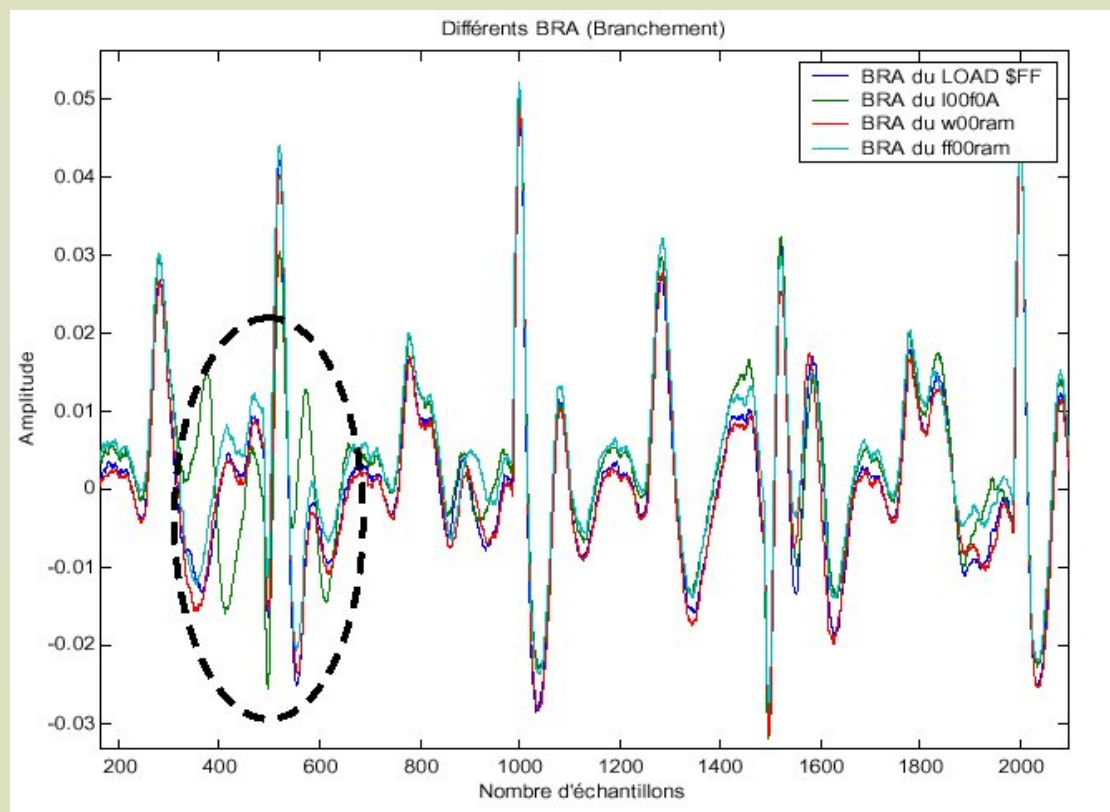
... et on zoome



Problème sur le BRA.

Cela demande confirmation.

La confirmation du problème



L'instruction BRA ne possède pas la même allure de signal selon son emplacement dans le programme.

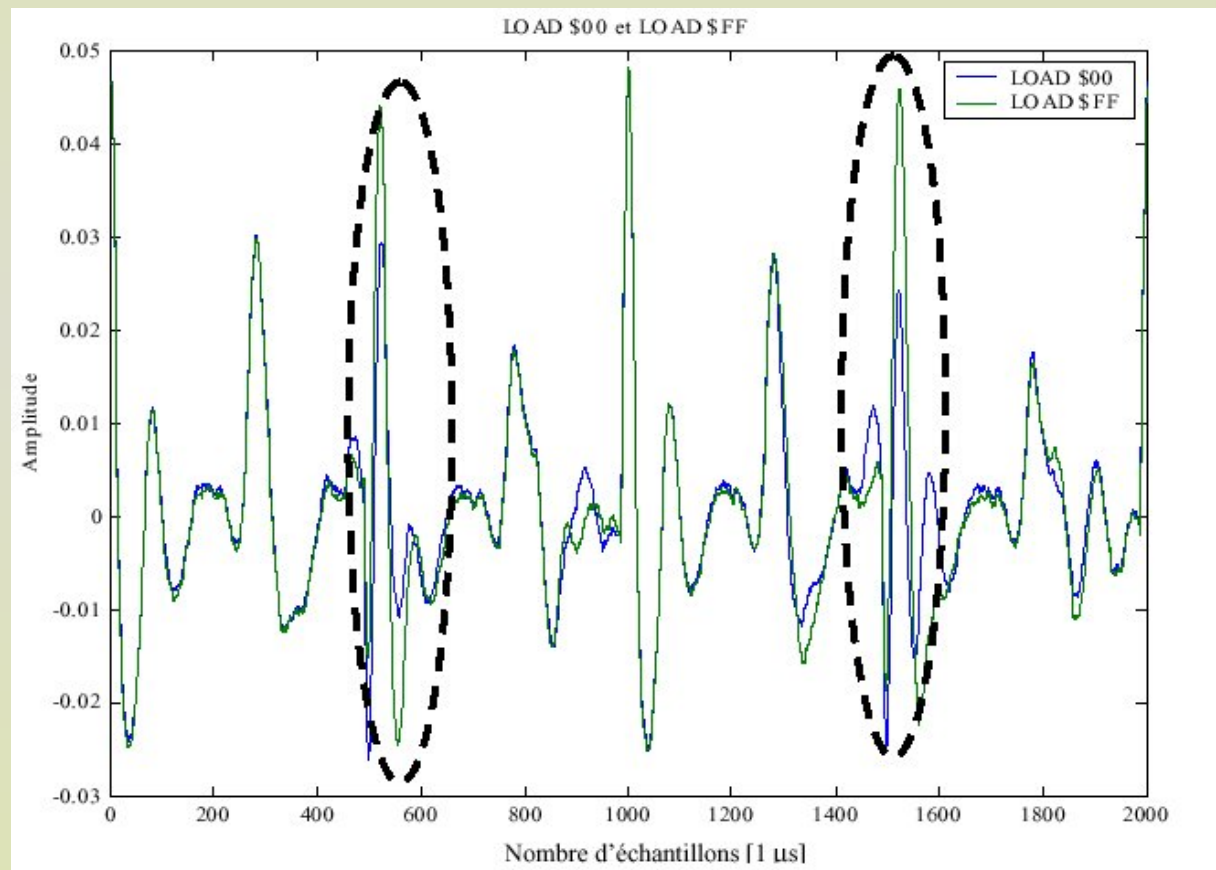
L'explication du problème

Ce problème vient du fait que l'on travaille sur un composant 68HC11A8 qui possède un bus commun **pour les données et pour les adresses**.

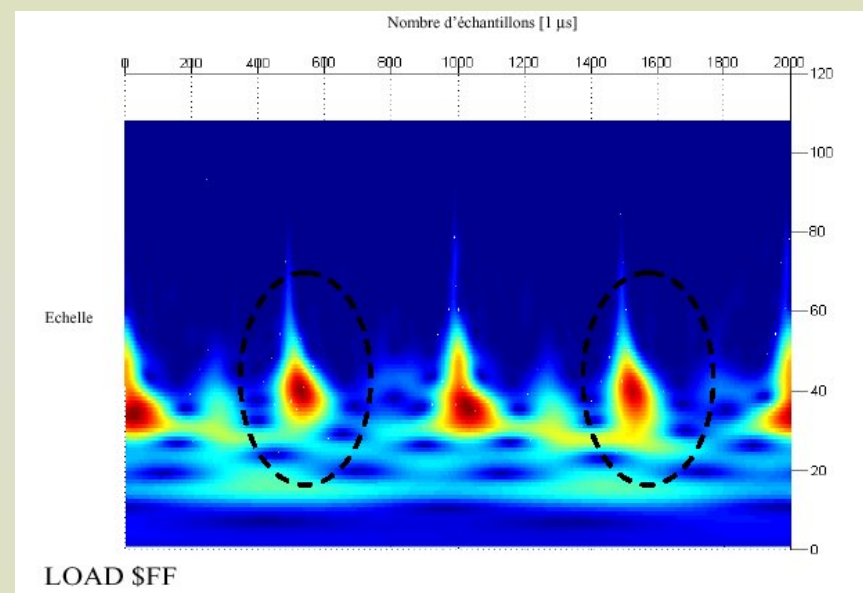
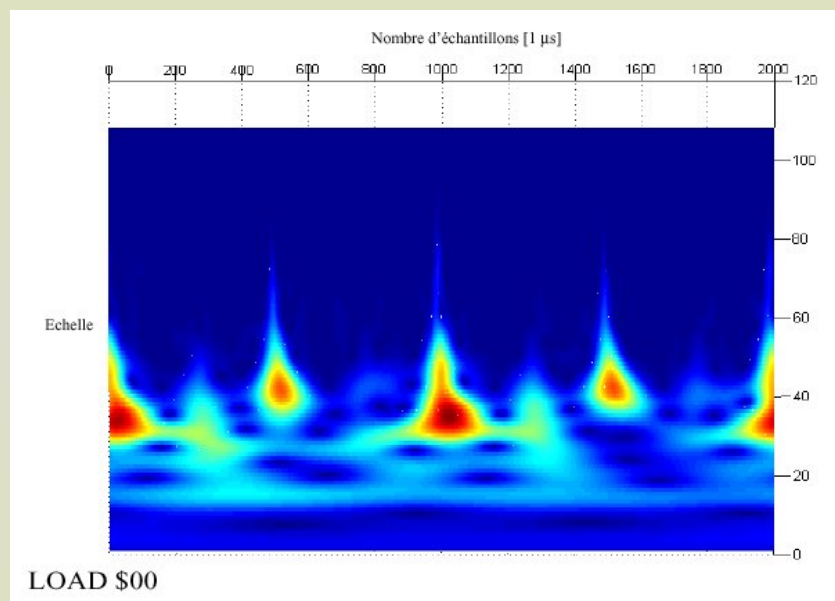
Résumé des problèmes :

- ➡ la non-répétitivité de l'instruction BRA,
- ➡ la ressemblance de certaines instructions.

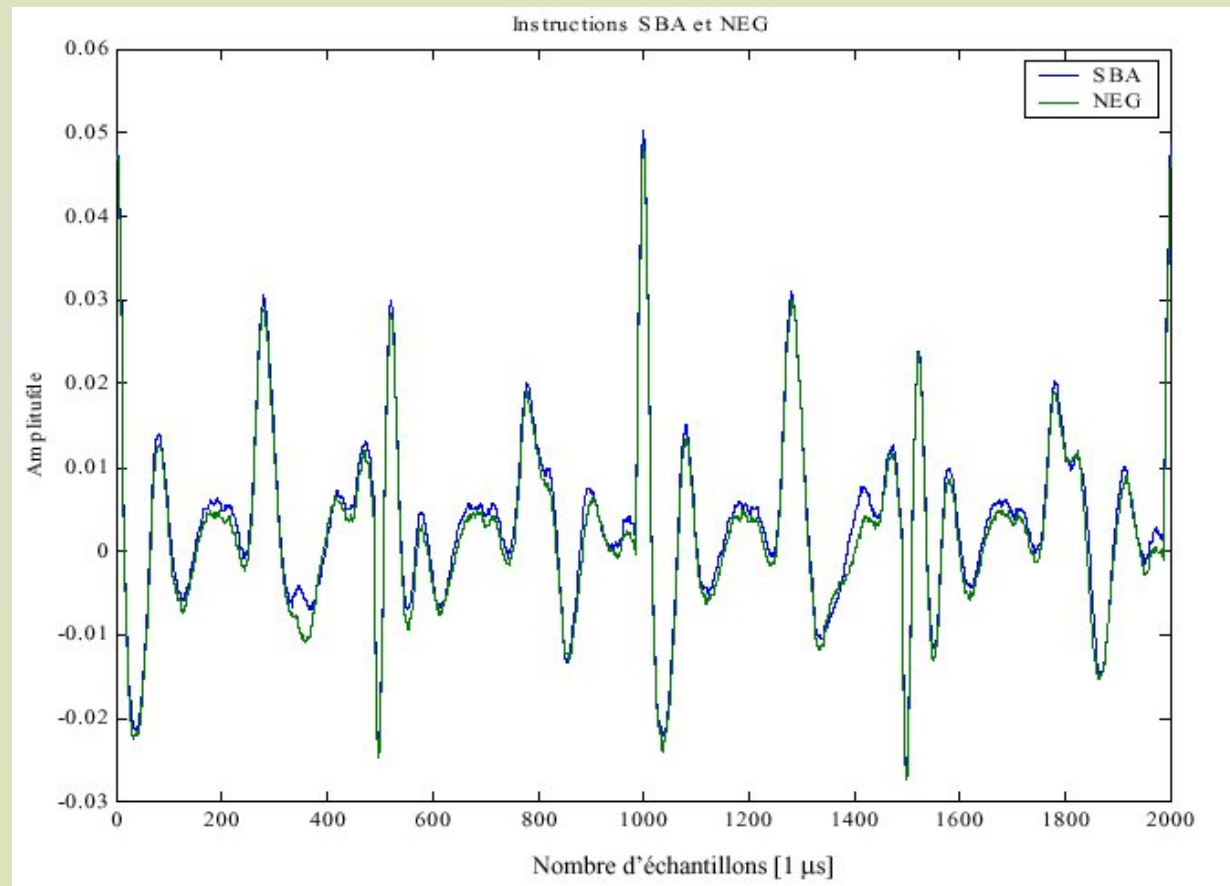
La méthode du temps-échelle ? (ondelettes)



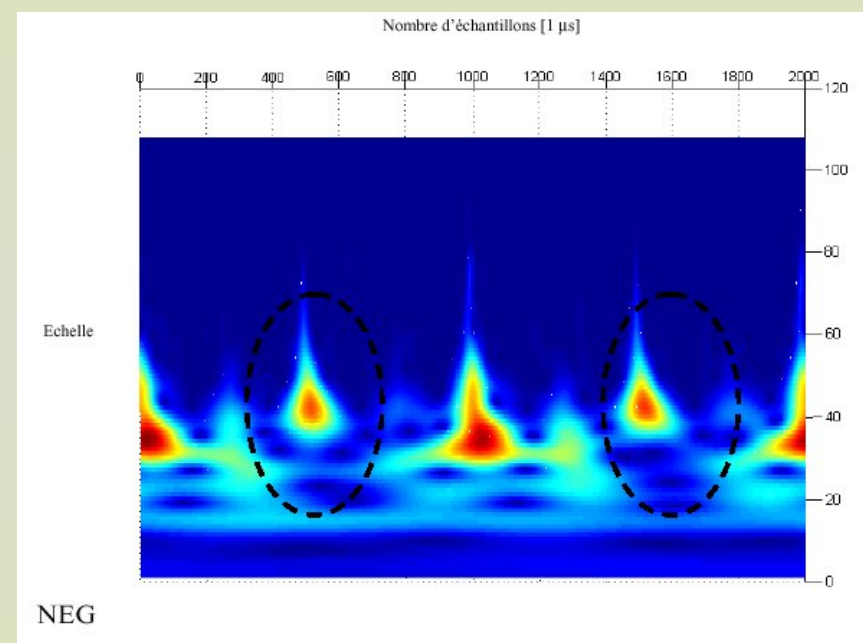
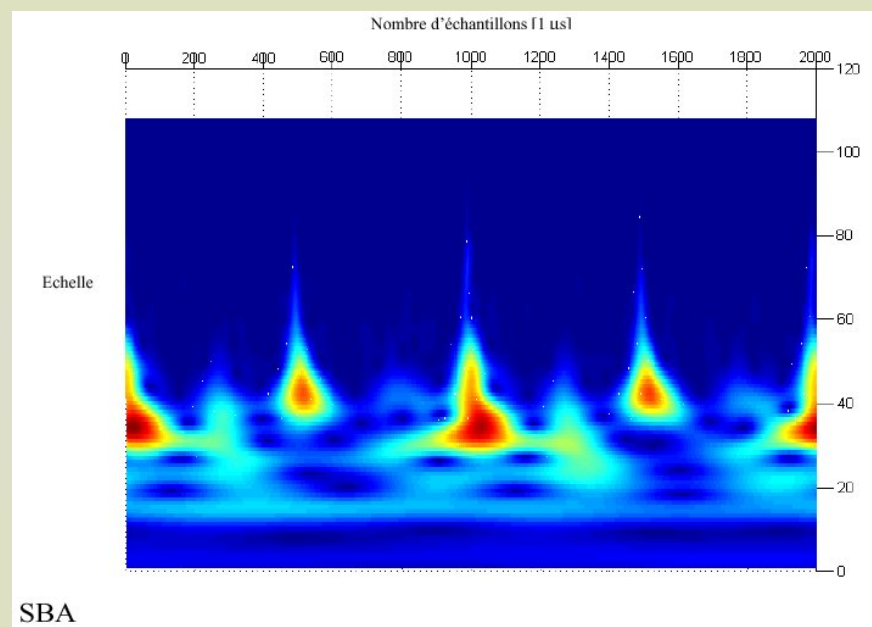
Tout fonctionne pour le moment !



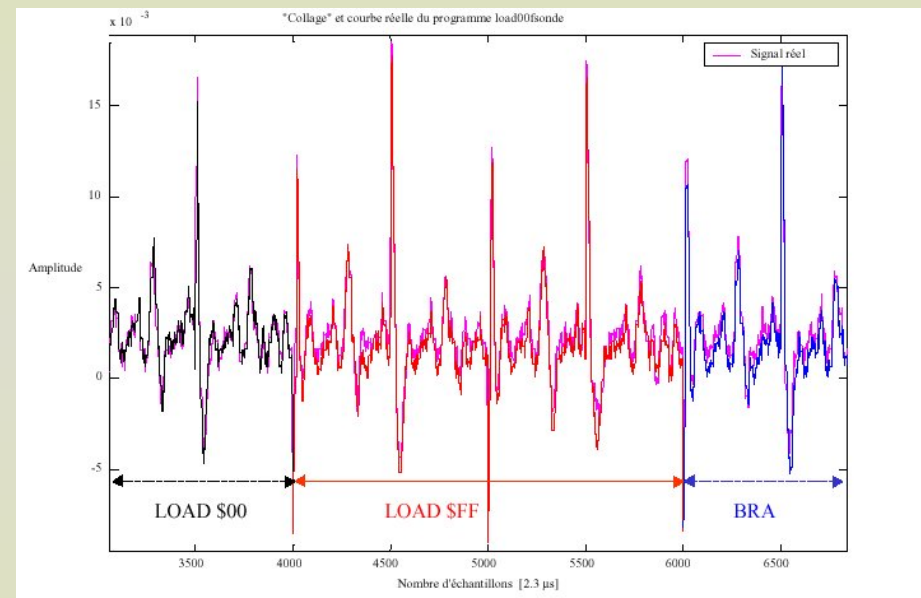
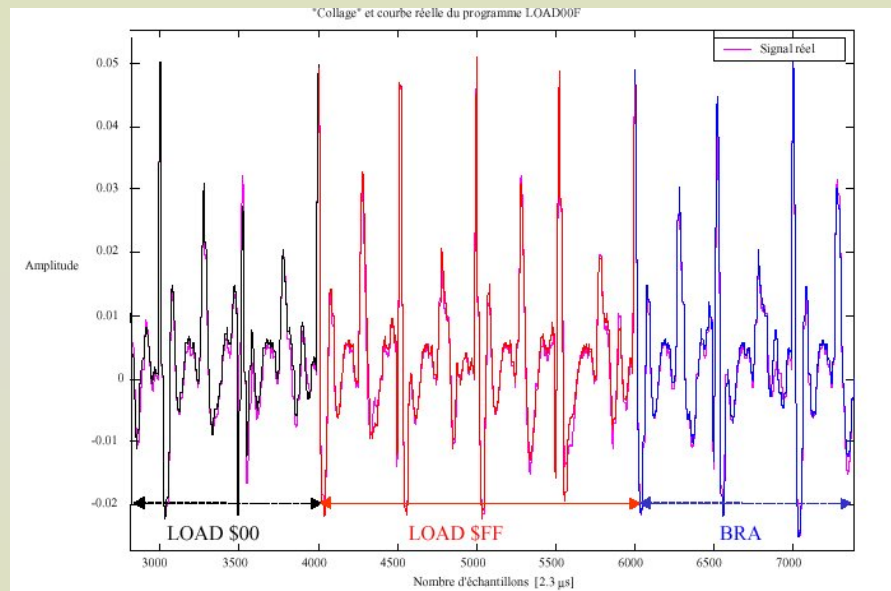
Comportement sur un de nos problèmes ?



KO!

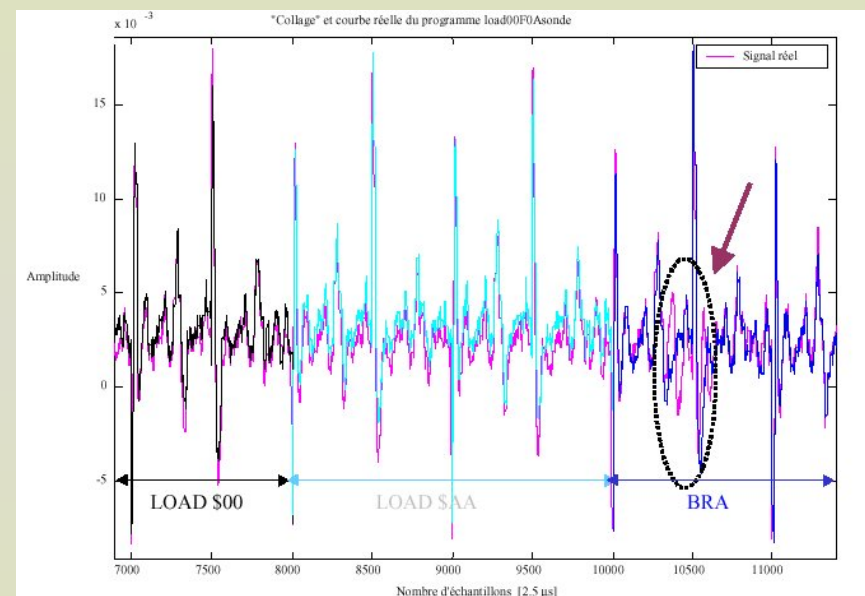
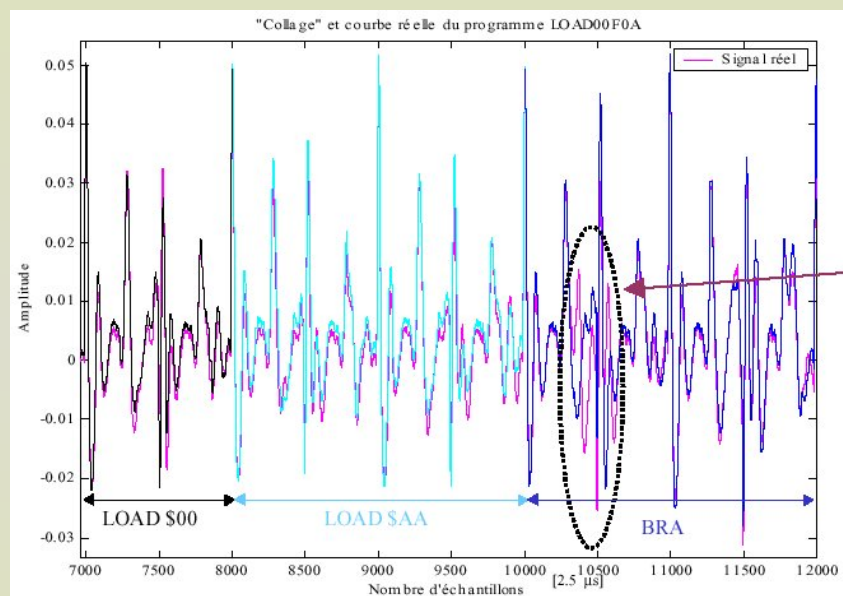


Et avec la CEM ? (1)



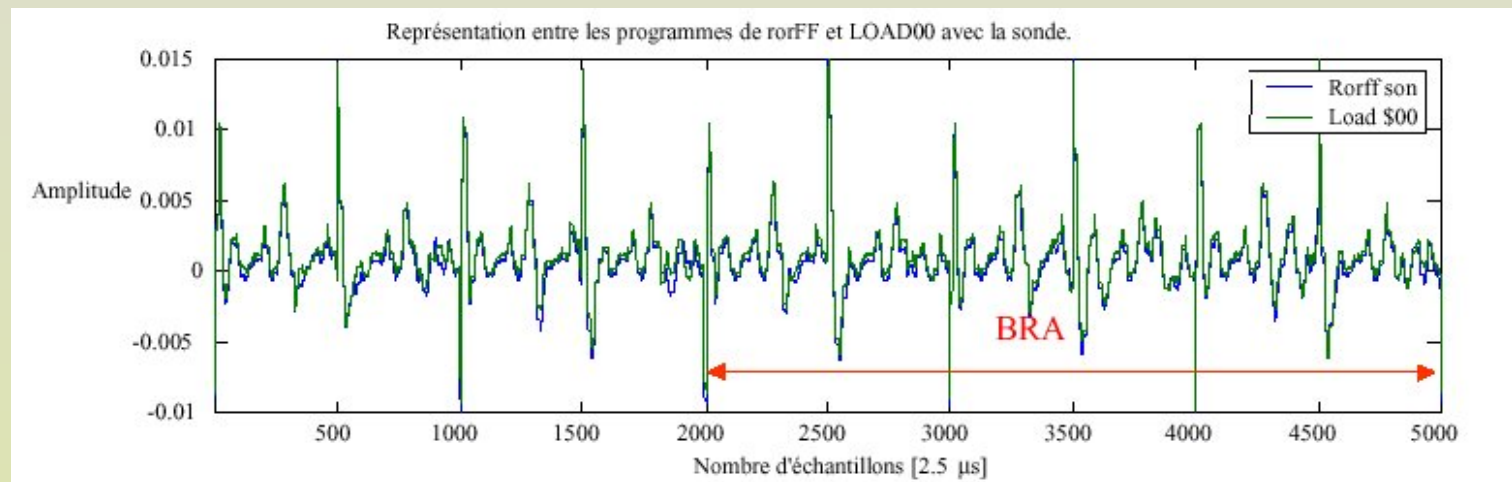
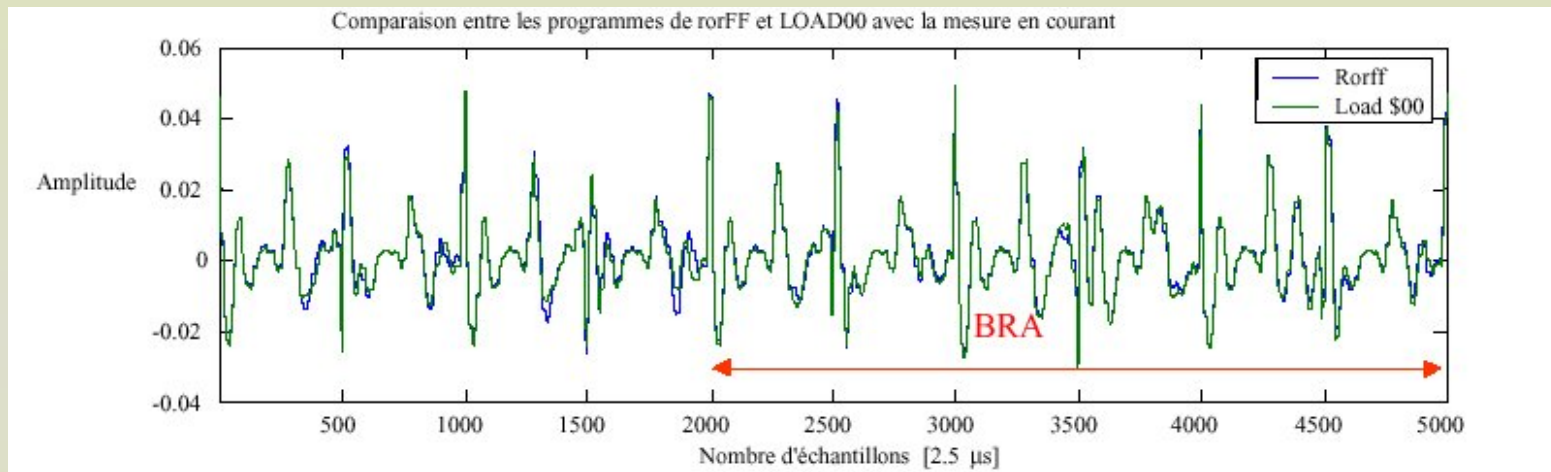
Sonde CEM au dessus du CPU

Et avec la CEM ? (2)

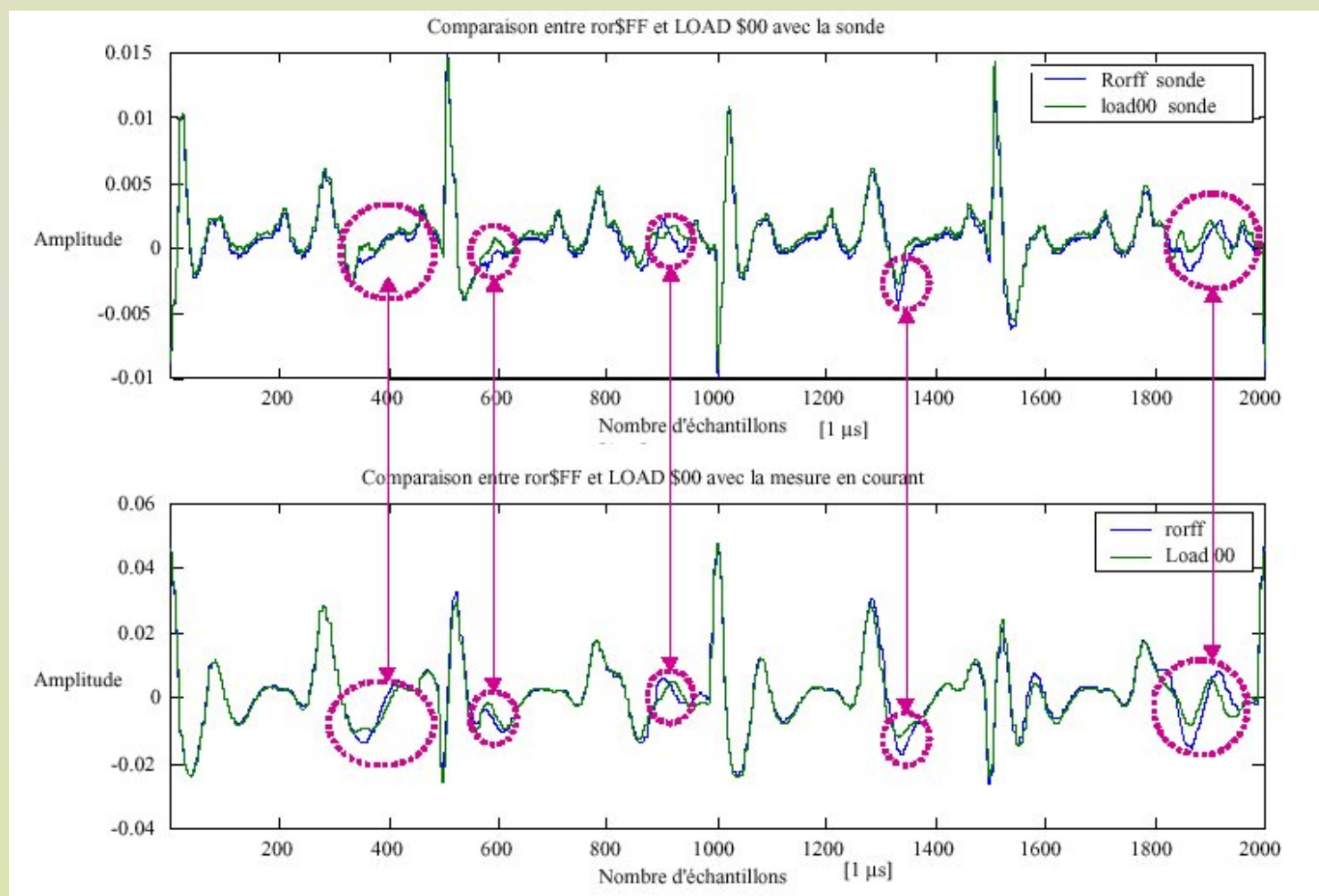


Sonde CEM au dessus du CPU

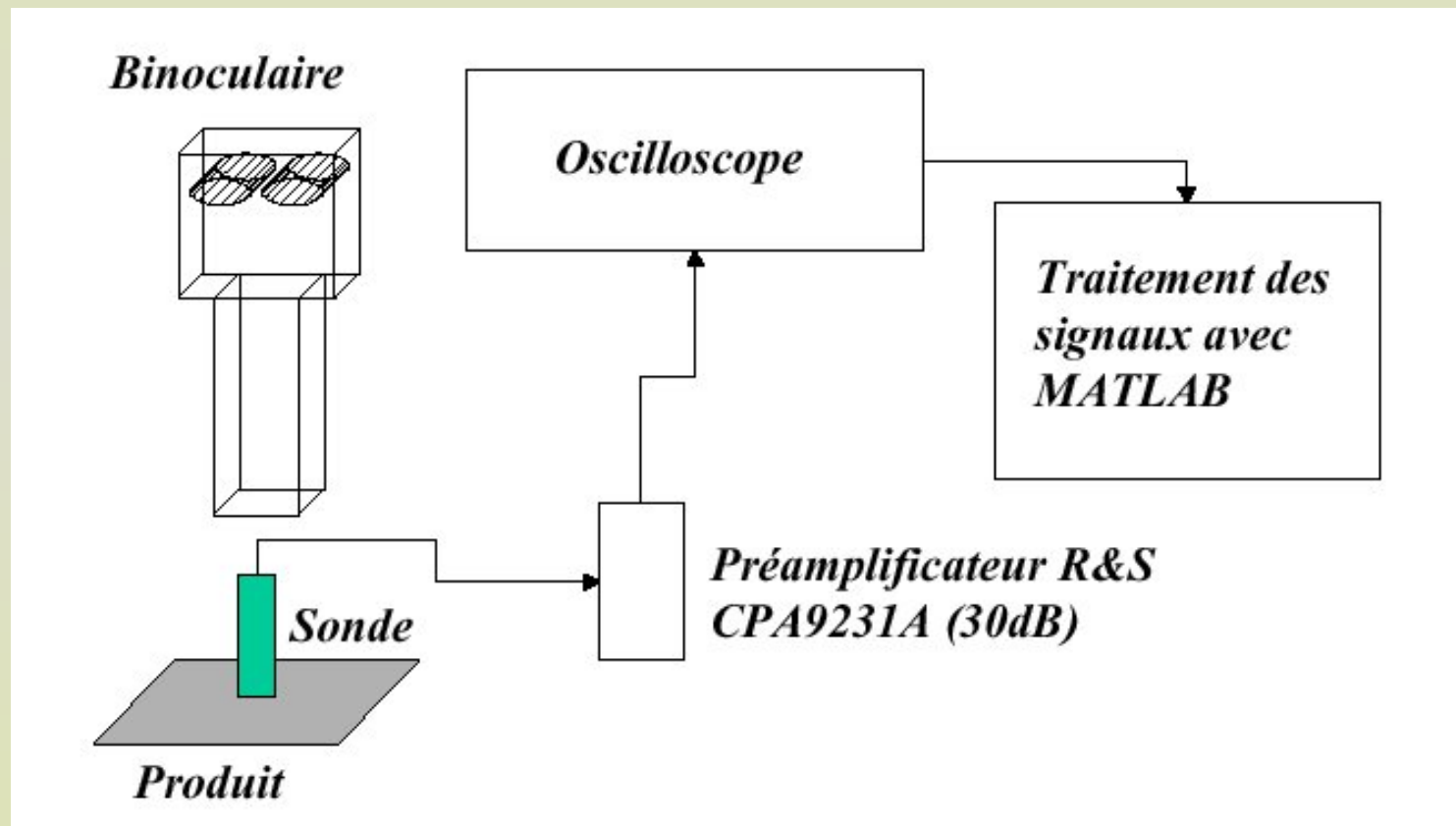
On a le même type d'information (1)



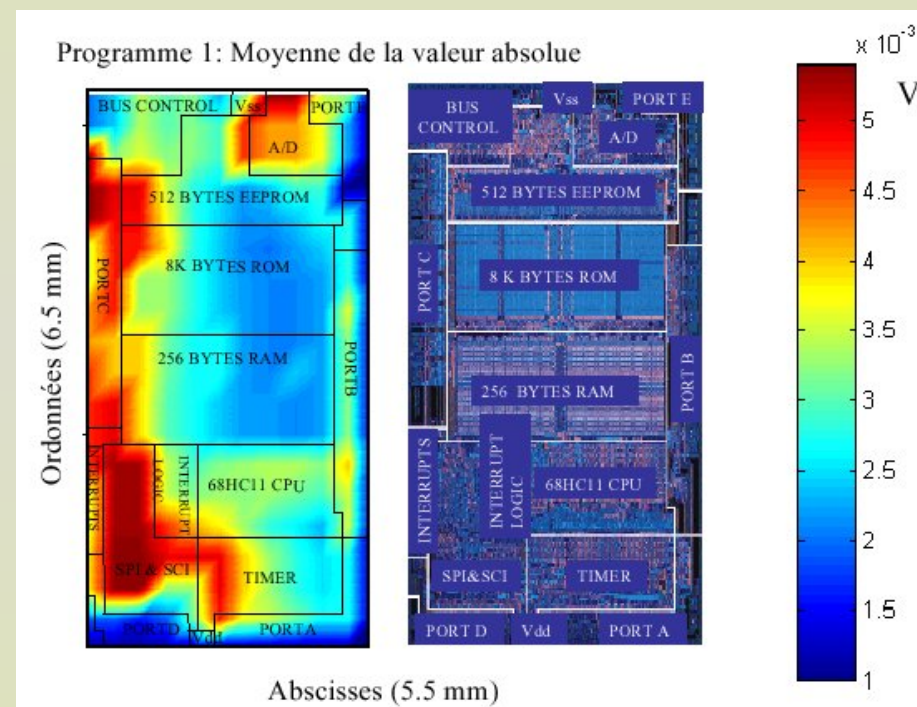
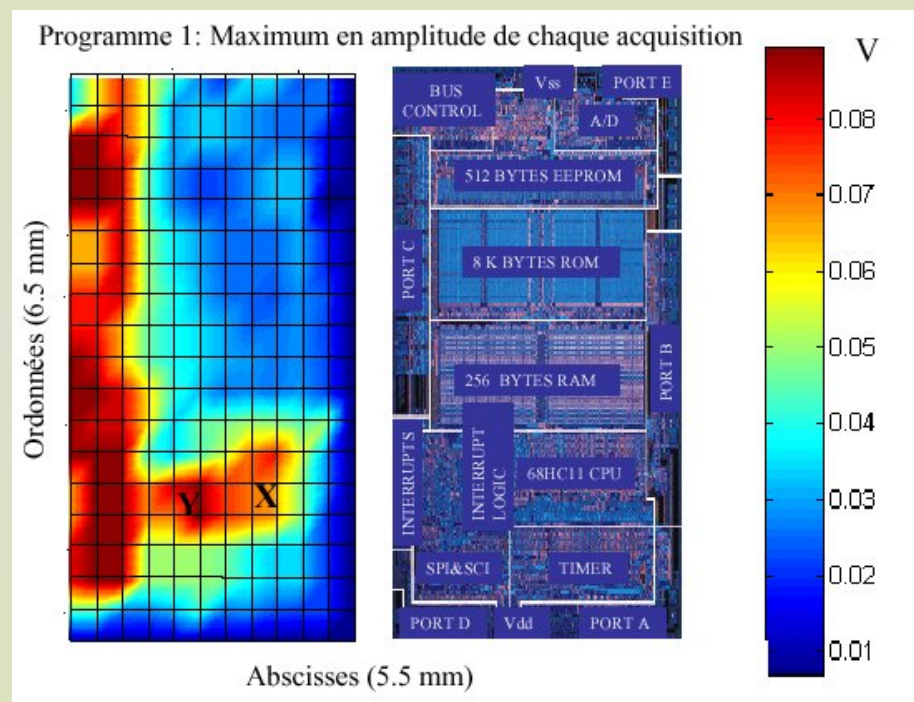
On a le même type d'information (2)



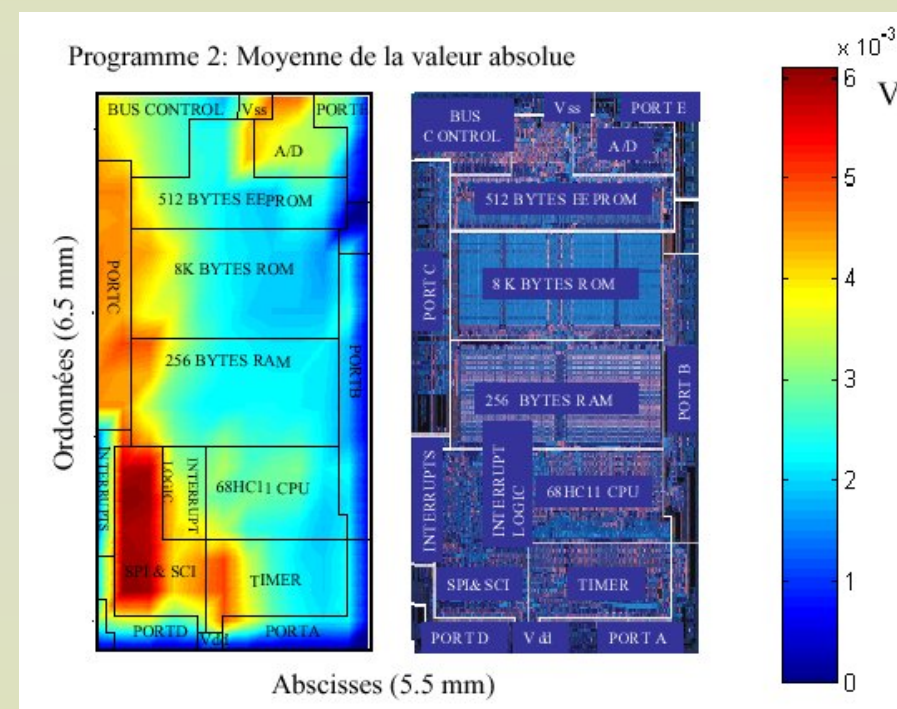
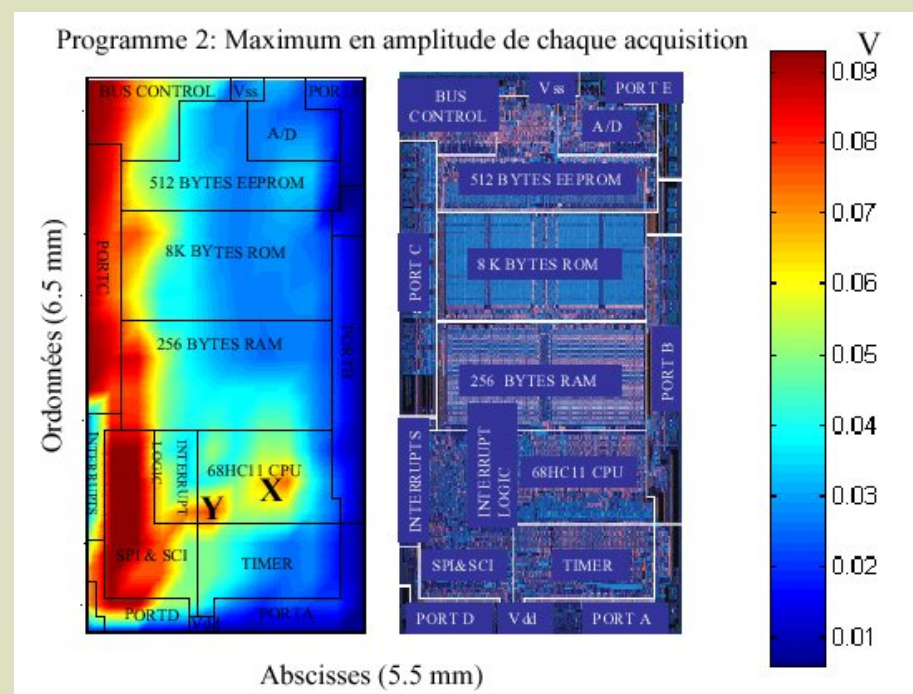
Le dispositif de cartographie



Cartographie pour l'algo 1



Cartographie pour l'algo 2



Les suites ...

Le positionnement sur les points chaud X et Y a permis de montrer que le CPU n'émet pas pareil partout et que l'on peut ainsi découvrir de l'information locale.

C'est une technique qui demande beaucoup de précision, mais qui a beaucoup d'avenir.

Application à la Java Card

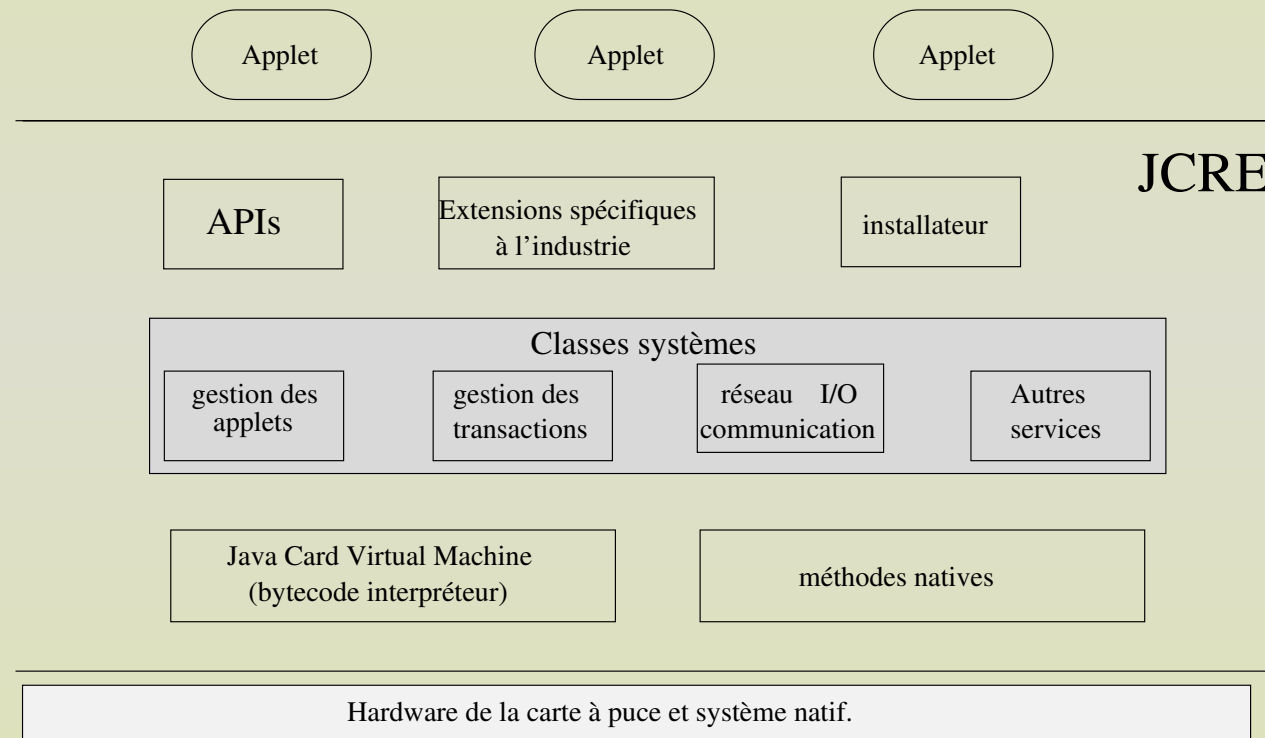
Problématique : Est-il possible d'appliquer des techniques d'exploitation des canaux cachés à la technologie Java Card ?

Réponse : oui

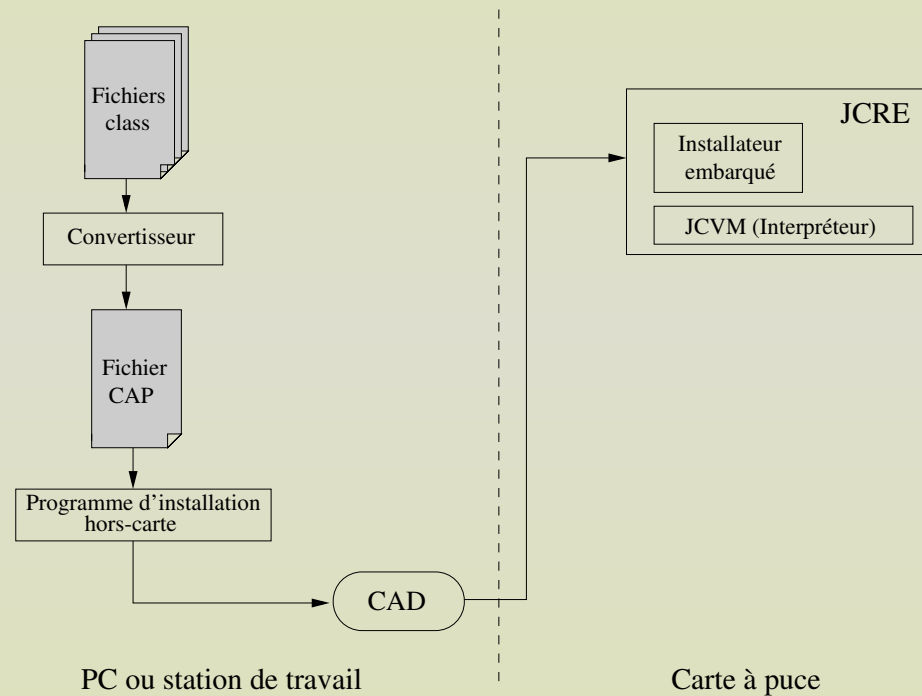
Avec succès : incertain, mais le verdict de faisabilité est plus facile et rapide à obtenir pour un attaquant car la Java Card est une carte multi-applicative **ouverte**.

Comment ? C'est ce que nous allons voir.

Architecture



Cycle de développement

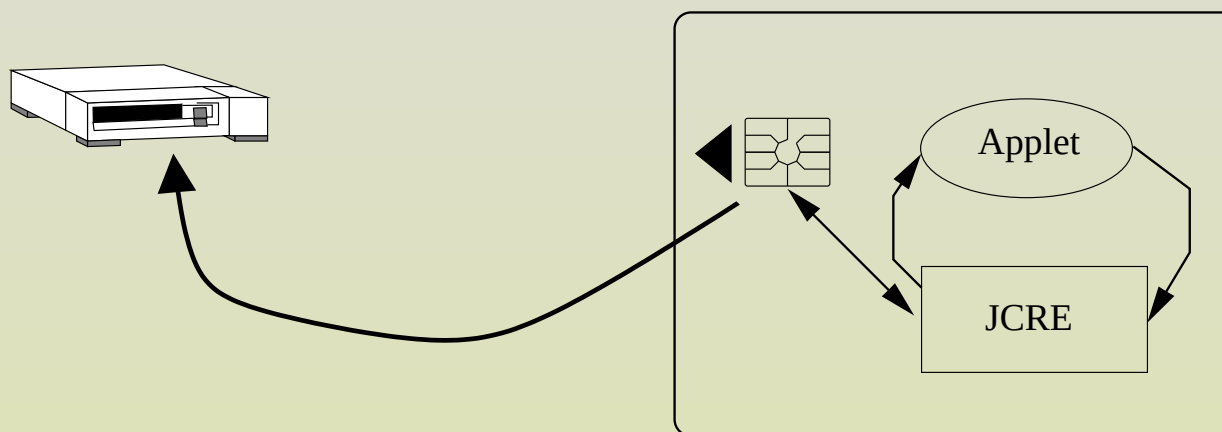


La communication

On a un modèle maître-esclave entre la carte et le lecteur.

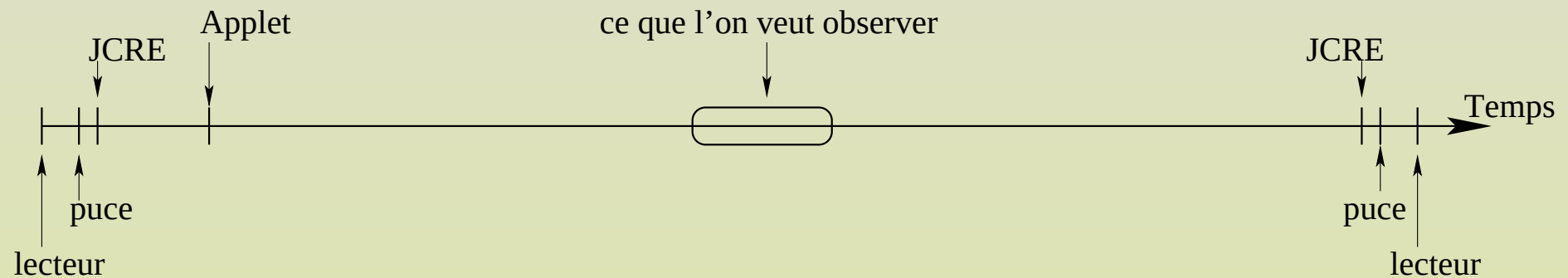
La communication se fait via :

- ➡ les APDUs pour la couche application,
- ➡ les TPDUs pour la couche transport.



Traitement normal

```
{
    ...
    // Encrypt sLength2 bytes from the offset 0 in the
    // apduBuffer and put the result into apduBuffer to
    // the offset 0
    sLength = cipher.doFinal(apduBuffer, (short) 0,
                             sLength2,
                             apduBuffer, (short) 0);
    apdu.sendBytes((short) 0, sLength);
}
```



Traitement avec 2 waitExtension

```
{
    ...
    // Send the synchro
    apdu.waitExtension();
    sLength = cipher.doFinal(apduBuffer, (short) 0,
                             sLength2,
                             apduBuffer, (short) 0);

    apdu.waitExtension();
    apdu.sendBytes((short) 0, sLength);
}
```

`apdu.waitExtension()` est une méthode statique qui envoie un octet d'attente au lecteur afin qu'il ne soit pas en timeout.

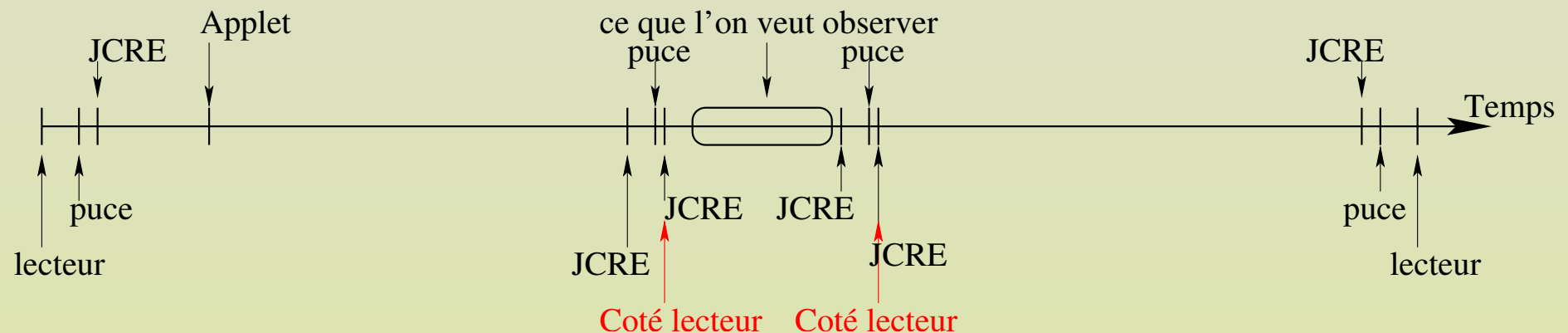
Traitement avec 1 waitExtension

```
{
    ...
    // Send the synchro
    apdu.waitExtension();
    sLength = cipher.doFinal(apduBuffer, (short) 0,
                             sLength2,
                             apduBuffer, (short) 0);
    apdu.sendBytes((short) 0, sLength);
}
```

Problème : Cette méthode est inhibée par la plupart des implémenteurs de plate-forme Java Card et la gestion des octets d'attente est automatique.

Traitement avec notre synchro

```
{
    ...
    // Here sLength = the next value of sLength + 1
    apdu.setOutgoingLength(sLength);
    // Send the synchro
    apdu.sendBytes((short)0x00, (short)0x01);
    sLength = cipher.doFinal(apduBuffer, (short) 1,
                             sLength2,
                             apduBuffer, (short) 0);
    apdu.sendBytes((short) 0, sLength);
}
```



Traitement avec notre synchro améliorée

```
(0x19)  aload_1
(0x1f)  sload_3
(0x8b)  invokevirtual    0x0  0xd
(0x19)  aload_1
(0x03)  sconst_0
(0x04)  sconst_1
(0x8b)  invokevirtual    0x0  0xa
(0xad)  getfield_a_this  0x1
(0x1a)  aload_2
(0x04)  sconst_1
(0x16)  sload            0x4
(0x1a)  aload_2
(0x03)  sconst_0
(0x8b)  invokevirtual    0x0  0x15
(0x32)  sstore_3
(0x19)  aload_1
(0x03)  sconst_0
(0x1f)  sload_3
(0x8b)  invokevirtual    0x0  0xa
(0x7a)  return
```

```
(0x19)  aload_1
(0x1f)  sload_3
(0x8b)  invokevirtual    0x0  0xd
(0x00)  nop
(0x00)  nop
(0x19)  aload_1
(0x03)  sconst_0
(0xad)  getfield_a_this  0x1
(0x1a)  aload_2
(0x04)  sconst_1
(0x16)  sload            0x4
(0x1a)  aload_2
(0x03)  sconst_0
(0x19)  aload_1
(0x03)  sconst_0
(0x04)  sconst_1
(0x8b)  invokevirtual    0x0  0xa
(0x8b)  invokevirtual    0x0  0x15
(0x8b)  invokevirtual    0x0  0xa
(0x7a)  return
```

Application au bytecode

Bytecode *pop*

```

aload_1
sconst_0
sconst_1
aconst_null
aload_1
sconst_0
sconst_1
invokevirtual    0x0 0xc
pop
invokevirtual    0x0 0xc
return
    
```

Liste des bytecodes instrumentés

<code>aconst_null__pop</code>	<code>sconst_3</code>
<code>aload_0__pop</code>	<code>sconst_4</code>
<code>astore_0</code>	<code>sconst_5</code>
<code>dup</code>	<code>sconst_m1__pop</code>
<code>dup2__pop2</code>	<code>sdiv</code>
<code>getfield_b</code>	<code>sload_0</code>
<code>getfield_b_this</code>	<code>sload_1</code>
<code>getfield_s</code>	<code>sload_2</code>
<code>getfield_s_this</code>	<code>sload_3</code>
<code>getstatic_b</code>	<code>smul</code>
<code>getstatic_s</code>	<code>sneg</code>
<code>invokestatic__aload_0__sconst_0__sconst_1</code>	<code>sor</code>
<code>invokevirtual__aload_1__sconst_0__sconst_1</code>	<code>srem</code>
<code>nop</code>	<code>sreturn</code>
<code>pop</code>	<code>sshl</code>
<code>pop2</code>	<code>sshr</code>
<code>putfield_s</code>	<code>sstore_0</code>
<code>putfield_s_this</code>	<code>sstore_1</code>

Liste des bytecodes instrumentés

`putstatic_s`

`s2b`

`sadd`

`sand`

`sconst_0__pop`

`sconst_1`

`sconst_2`

`sstore_2`

`sstore_3`

`ssub`

`sushr`

`sxor`

Perspectives

Un article à écrire sur ce problème et sur des proposition de solutions.

Étude des bytecodes suivant les méthodes présentées.

Utilisation du mécanisme de synchronisation pour mieux connaître les implémentations des différents objets sur la carte afin de pouvoir faire de l'injection de faute (DFA).

...