Institut de recherches sur les lois fondamentales de l'Univers

Université Paris-Saclay

# Conception multitâches et OS temps réel

**Shebli Anvar** PhD.
CEA Irfu, Centre de Saclay
91191 Gif-sur-Yvette
France
shebli.anvar@cea.fr
+33 1 69 08 78 32
+33 6 63 31 92 26
+33 1 69 08 31 47

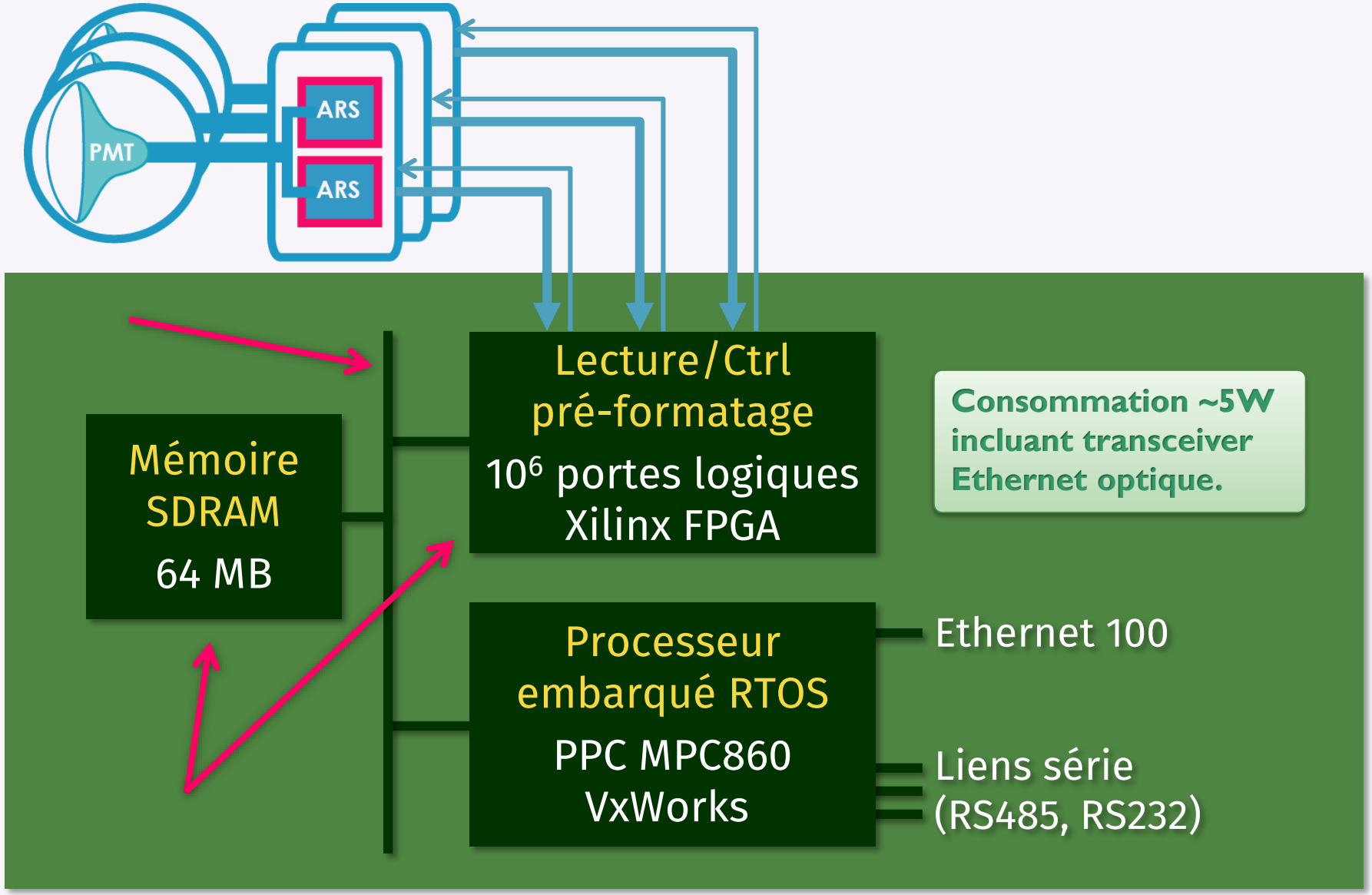
# Système informatique temps réel

Le comportement d'un système informatique est qualifié de ***temps réel*** lorsqu'il est assujetti à ***l'évolution d'un procédé*** qui lui est connecté et qu'il doit piloter ou suivre en ***réagissant*** à tous ses changements d'état.

# Mapping Mémoire

**Shebli Anvar** PhD.
CEA Irfu, Centre de Saclay
91191 Gif-sur-Yvette
France
shebli.anvar@cea.fr
+33 1 69 08 78 32
+33 6 63 31 92 26
+33 1 69 08 31 47

# Carte d'acquisition embarquée

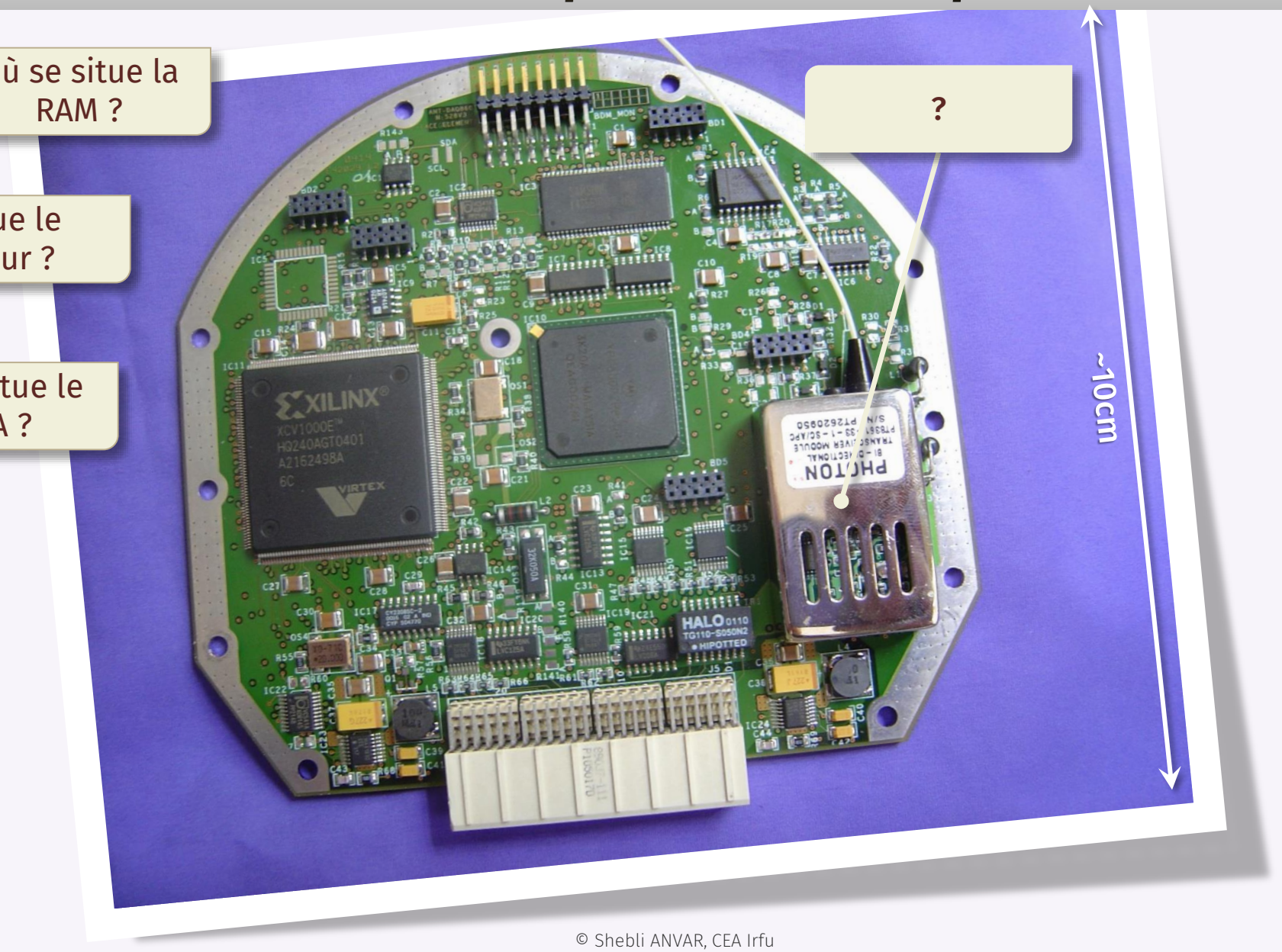


# Carte d'acquisition embarquée

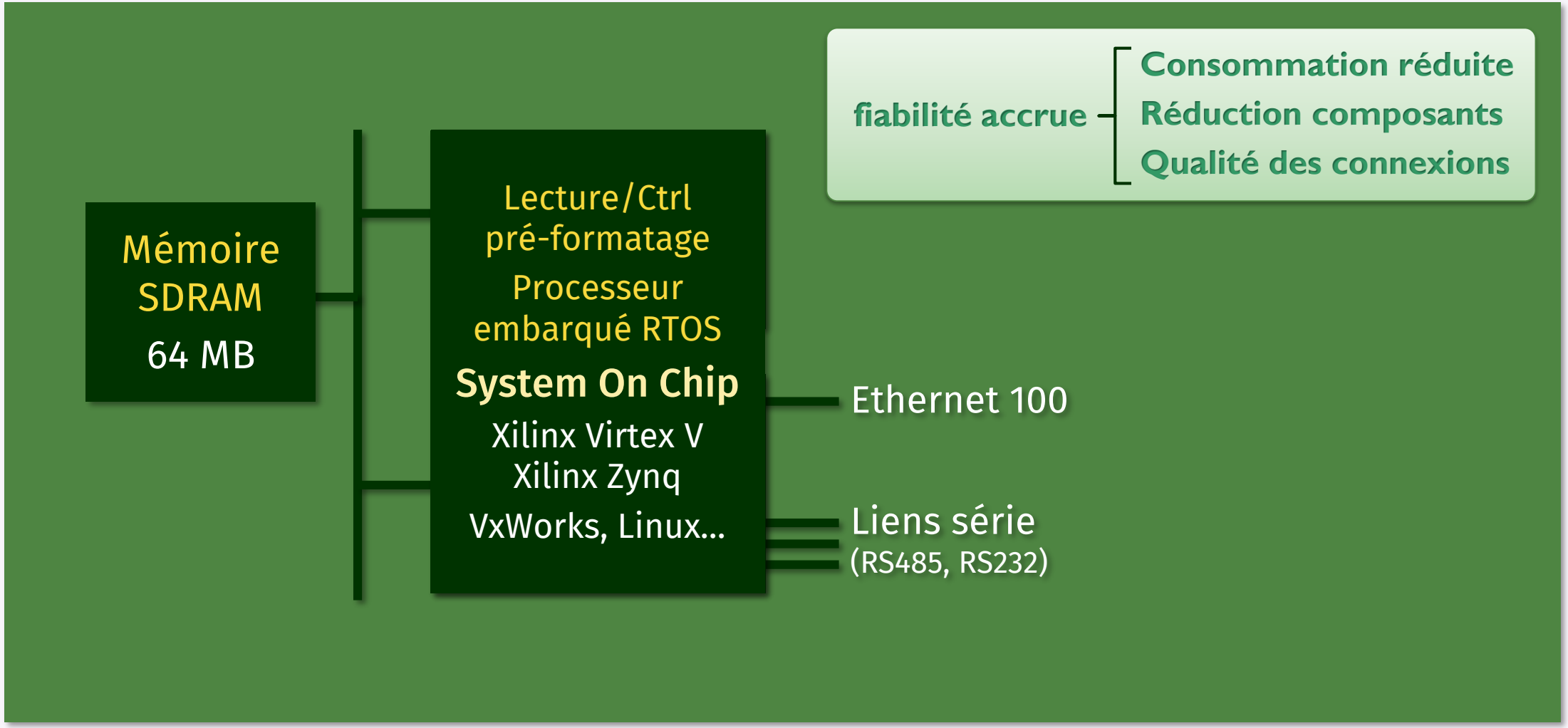
Où se situe la RAM ?

Où se situe le processeur ?

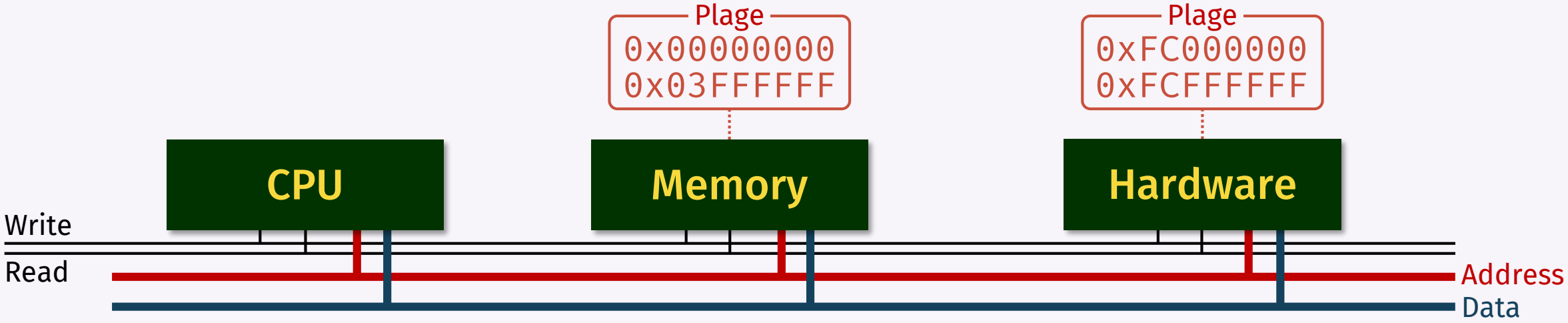
Où se situe le FPGA ?



# Carte d'acquisition embarquée : System On Chip

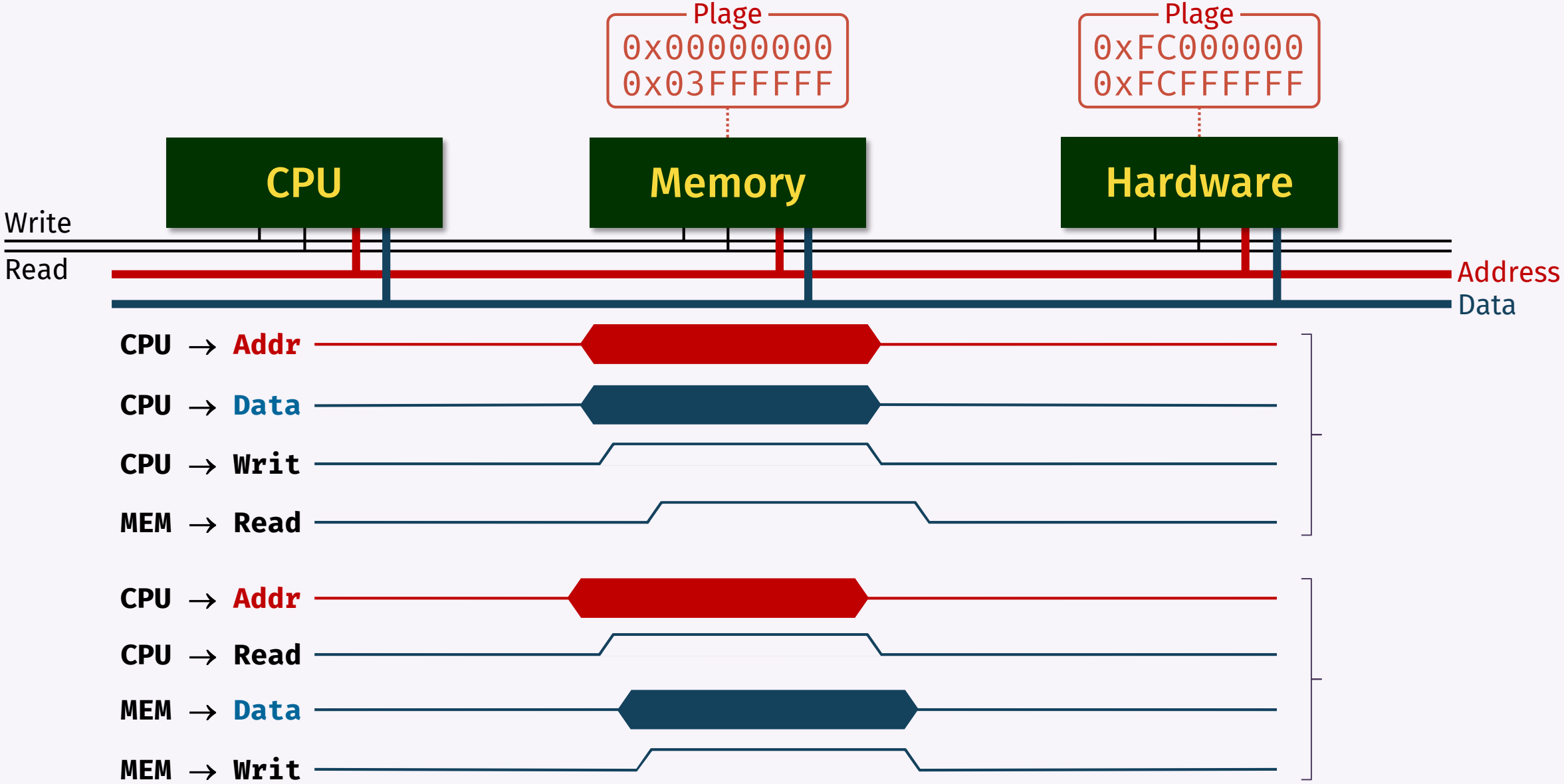


# Principe de communication par bus

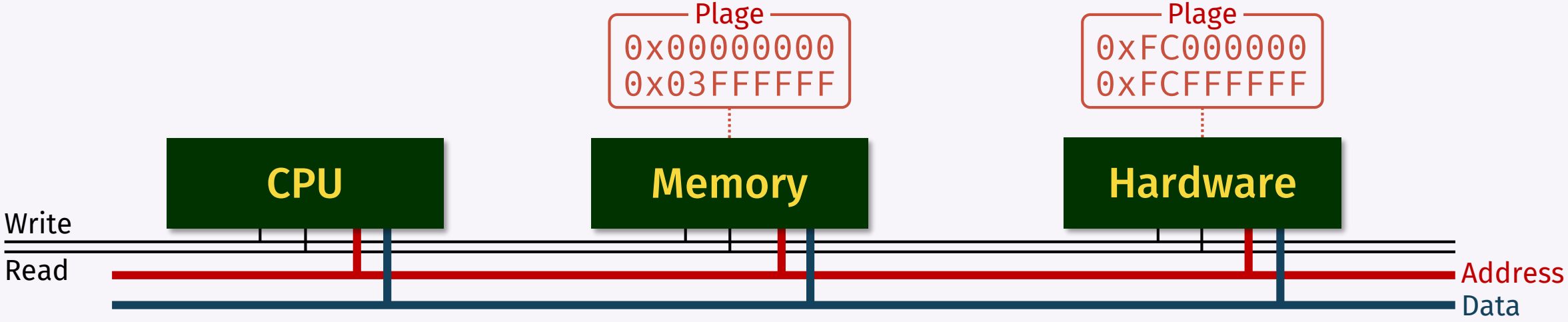



- **CPU: active l'adresse sur le bus Address (ex: `0xFC002000`).**
- **Périphérique : un seul est sensible à cette adresse**
- **Si écriture :**
  - CPU active la donnée sur le bus Data.
  - Périphérique : concerné traite la donnée sur le bus Data.
- **Si lecture :**
  - Périphérique : active la donnée sur le bus Data.
  - CPU : traite (lis) la donnée sur le bus Data.
- **Modes de lecture/écriture spéciaux (rafale synchrone, etc.)**
- **Contrôleur mémoire : séparé ou intégré au CPU**

# Principe de communication par bus



# Communication par bus : versant logiciel




```
unsigned* ptr = (unsigned*) 0x02000100u;
int toto = *ptr;
printf("val = %d\n", toto);
```

quelles opérations sur le bus mémoire ?

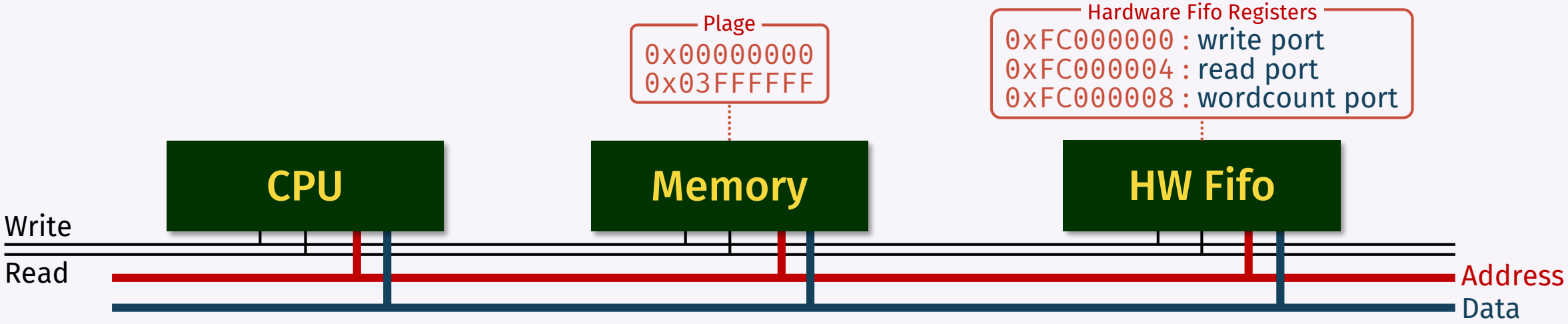
```
unsigned* ptr = (unsigned*) 0xFC002000u;
ptr[0]  = 123u;
ptr[2]  = 0x10000u;
printf("reg = %x\n" , ptr[2];
```

À quelle adresse se fait l'écriture ?

```
ptr = (unsigned*) 0x04001000u;
*ptr++ = 0u;
```

Que se passe-t-il au niveau du processeur ?

# Communication par bus : précautions logicielles




Écrivez le code qui écrit les 1000 premiers nombres impairs dans la fifo hardware,
puis qui vide les valeurs de la fifo dans un tableau

# Taille d'une variable scalaire en mémoire

- **BYTE = unité d'information référencée par une adresse mémoire**
- **Taille en BYTES des éléments scalaires en langage C/C++ : « sizeof »**

| Architectures 32 bits typiques | | |
|---|---|---|
| type | sizeof | bits |
| char | 1 | 8 |
| short | 2 | 16 |
| int | 4 | 32 |
| long | 4 | 32 |
| long long | 8 | 64 |
| void* | 4 | 32 |

Validité de l'opérateur « cast »

```
char* pc = (char*) 0xFC000000;
int n = (int) pc;
void* pv = (void*) n;
short x = (short) pv;
int* pi = (int*) x;
```

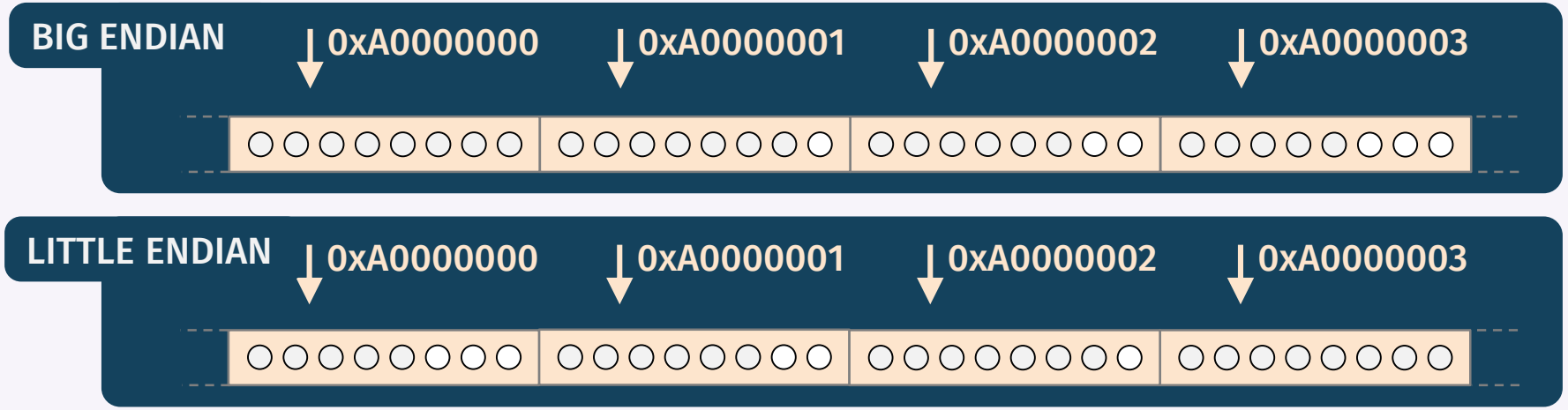
| Architectures 64 bits typiques | | |
|---|---|---|
| type | sizeof | bits |
| char | 1 | 8 |
| short | 2 | 16 |
| int | 4 | 32 |
| long | **8** | **64** |
| long long | 8 | 64 |
| void* | **8** | **64** |

Validité de l'opérateur « cast »

```
char* pc = (char*) 0xFC000000;
int n = (int) pc;
void* pv = (void*) n;
short x = (short) pv;
int* pi = (int*) x;
```

# Endianité (Endianness)

- **BYTE = unité d'information référencée par une adresse mémoire**
- **Lorsqu'un type de donnée primitif (char, short, int, long, etc.) est composé de plus d'un BYTE, la question de l'ordre de ces BYTES en mémoire se pose**
- **Deux grandes classes d'architecture sont aujourd'hui courantes :**
  - LITTLE ENDIAN (architectures Intel i86)
  - BIG ENDIAN (architectures PowerPC, SPARC, etc.)
- **Exemple : *entier 32 bits n* sur mémoire adressant des *BYTES de 8 bits***
  **Supposons que *n* contient la valeur 66311 = 0x10307 = 0b10000001100000111**
  **Supposons que *&n* (adresse de *n* en mémoire) est égale à 0xA0000000**



# Endianité (Endianness)

```
int n = 0x10307; // Élément 32 bits
char* p = (char*) &n; // Pointeur sur octet (8 bits)
printf("%d, %d, %d, %d\n", p[0] , p[1] , p[2] , p[3]);
```

Affichage sur architecture BIG ENDIAN

Affichage sur architecture LITTLE ENDIAN

```
int n = 0x10307; // Élément 32 bits
short* p = (short*) &n; // Pointeur sur élément 16 bits
printf("%d, %d\n", p[0] , p[1]);
```

Affichage sur architecture BIG ENDIAN

Affichage sur architecture LITTLE ENDIAN

# Opérateurs binaires (bitwise operators)

## bitwise OR

```
int n = 0x23 | 0x11;
printf("%x, %d\n", n, n);
```

## bitwise AND

```
int n = 0x23 & 0x11;
printf("%x, %d\n", n, n);
```

## bitwise XOR

```
int n = 0x23 ^ 0x11;
printf("%x, %d\n", n, n);
```

## bitwise NOT

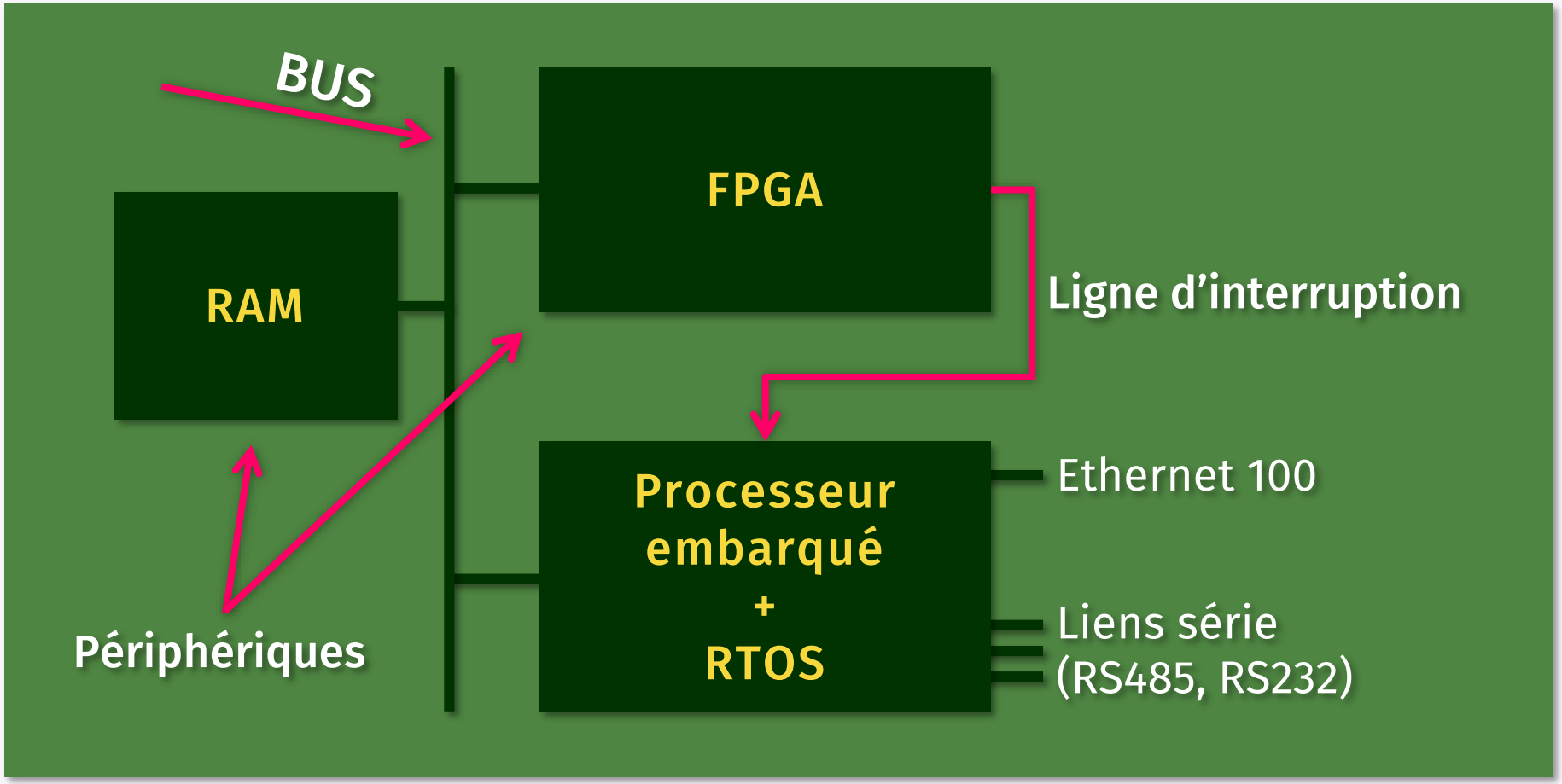
```
int n = ~0x23;
printf("%x, %d\n", n, n);
```

Nombre opposé = complément à 2 :

-N ⇔ ~N+1

# Interruption matérielle

- **L'autre moyen de communication entre un périphérique et le CPU est l'interruption matérielle. C'est le seul moyen communication qui qui soit à l'initiative du périphérique.**



# Interruption matérielle

- **L'autre moyen de communication entre un périphérique et le CPU est l'interruption matérielle. C'est le seul moyen communication qui qui soit à l'initiative du périphérique.**
- **Son fonctionnement exact dépend de l'architecture matérielle de la carte et des liaisons entre le périphérique et les broches d'interruption du CPU.**
- **Le CPU associe à une broche une fonction d'interruption. Dans cette fonction, l'interruption est traitée. Si le CPU est animé par un OS :**
  - Tous les appels susceptibles d'être bloquants sont interdits au sein de la fonction d'interruption.
  - Le temps d'exécution de l'interruption doit être minimisé.
  - Tout traitement lourd doit être effectué par une tâche standard en attente de libération d'un sémaphore. Typiquement, ce sémaphore est libéré par la fonction d'interruption.