Architectures des microcontrôleurs

Yannick Bornat (bornat@enseirb-matmeca.fr)

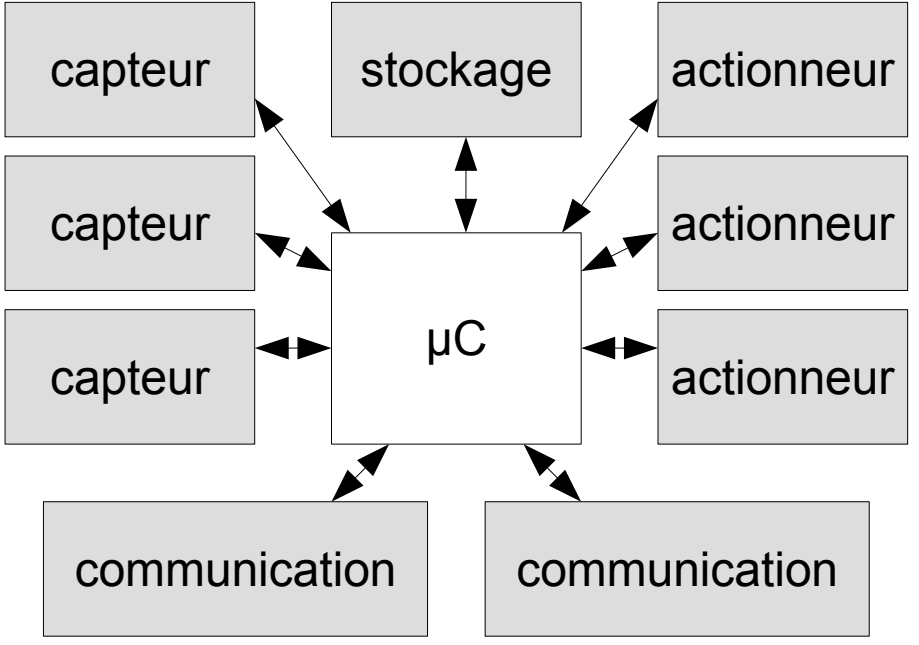
44 heures sur le S6 :

- 28 heures de cours intégré (MI100) – 7 séances
  - → evaluation : examen sur feuille (QCM)
- 16 heures de projet (MI105) – 4 séances
  - → evaluation : rapport de projet

Rôle d’un microcontrôleur (µC)

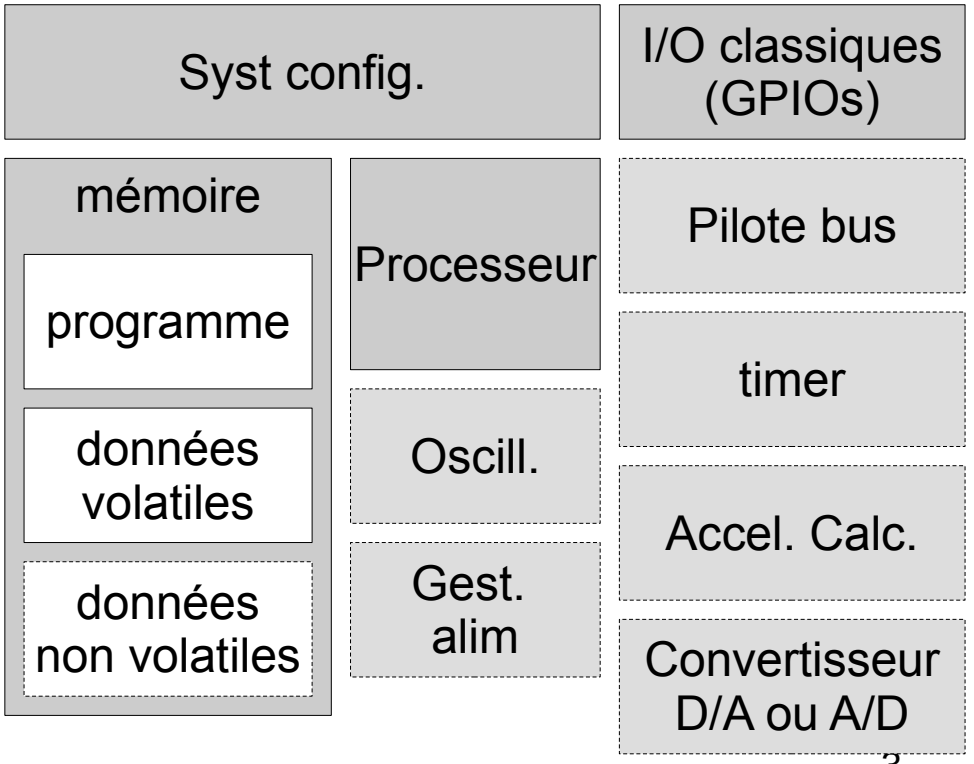
- Cadencement de différentes actions
- Réaction/prise de décision rapides
- Implémentation d’un régulateur
- ...

- Fortes contraintes sur :
  - Le temps
  - La consommation
  - Les dimensions physiques



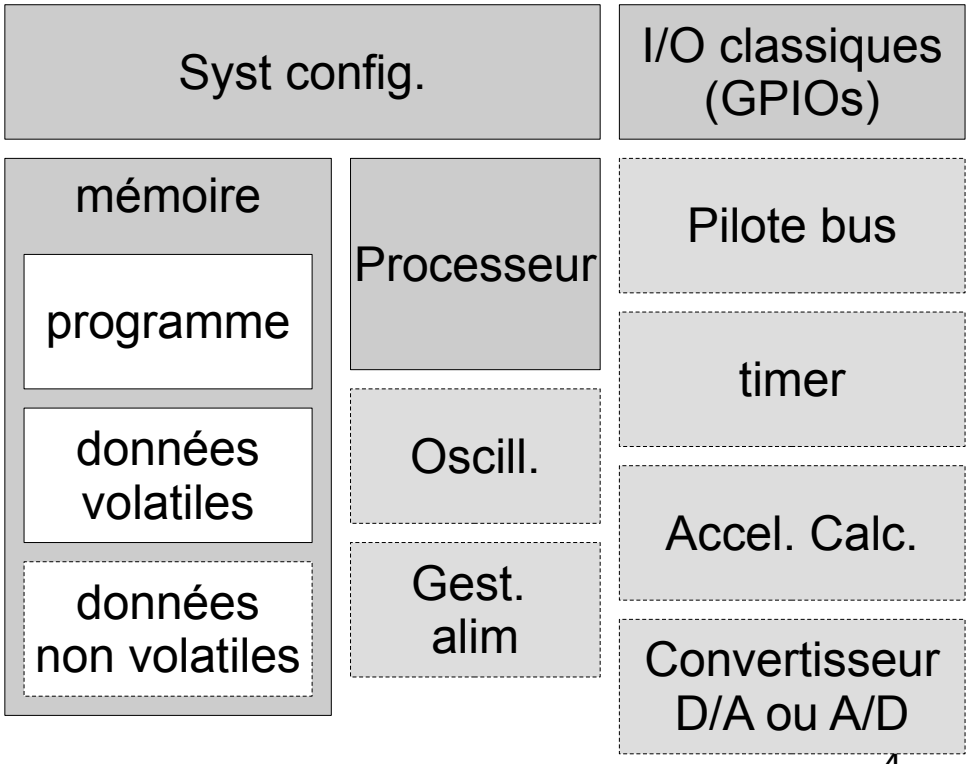
Contenu d’un µC

- Indispensable :
  - Processeur
  - Mémoire programme (non volatile)
  - Mémoire de données (volatile)
  - GPIOs (General Purpose Input/outputs)
- Tout l’environnement nécessaire à l’éxécution des programmes



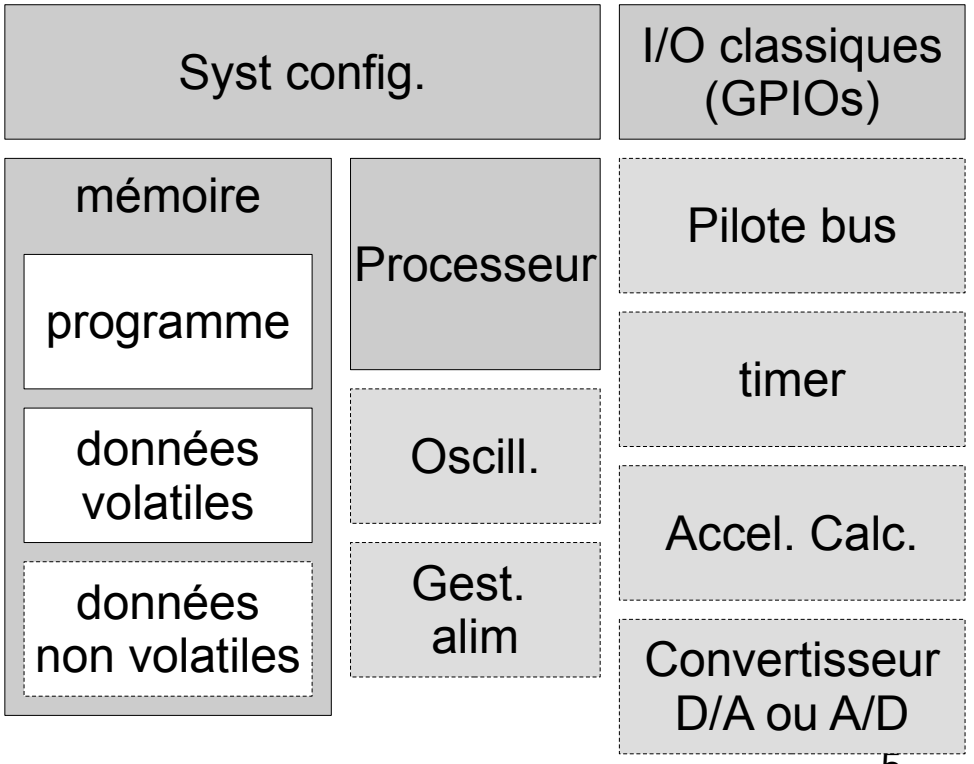
Contenu d'un µC

- Fréquent :
  - Mémoire de données non volatile
  - Oscillateur (RC, quartz int. ou ext.)
  - Gestion de l'alimentation
  - Pilote de bus (SPI, I^2C, CAN…)
  - Timer(s)
  - Accélérateur de calcul (*, /, …)
  - Convertisseur A/D ou D/A



Contenu d’un µC

- Plus rare :
  - Communication haut niveau
    - BT / Wifi / USB / Ethernet
  - Gestion d’écran
  - Stockage (carte SD, …)
  - Accélérateur de calcul (chiffrement, CRC, …)
  -



Exemples de µC

- Séries les plus fréquentes :
  - PIC (10Fxxx, 12Fxxx, 14Fxxx, 16Fxxx, 18Fxxx, ...)
  - Atmel (ATmega, AT91SAMxxx, …)
  - Freescale (68HCxx, …)
- Cas particulier : cartes Arduino :
  - Arduino N'EST PAS un fabricant de µC
  - Arduino est une marque de cartes à base de µC (souvent Atmel)
- Cartes Raspberry :
  - Ces sont des SoCs (System on Chip)

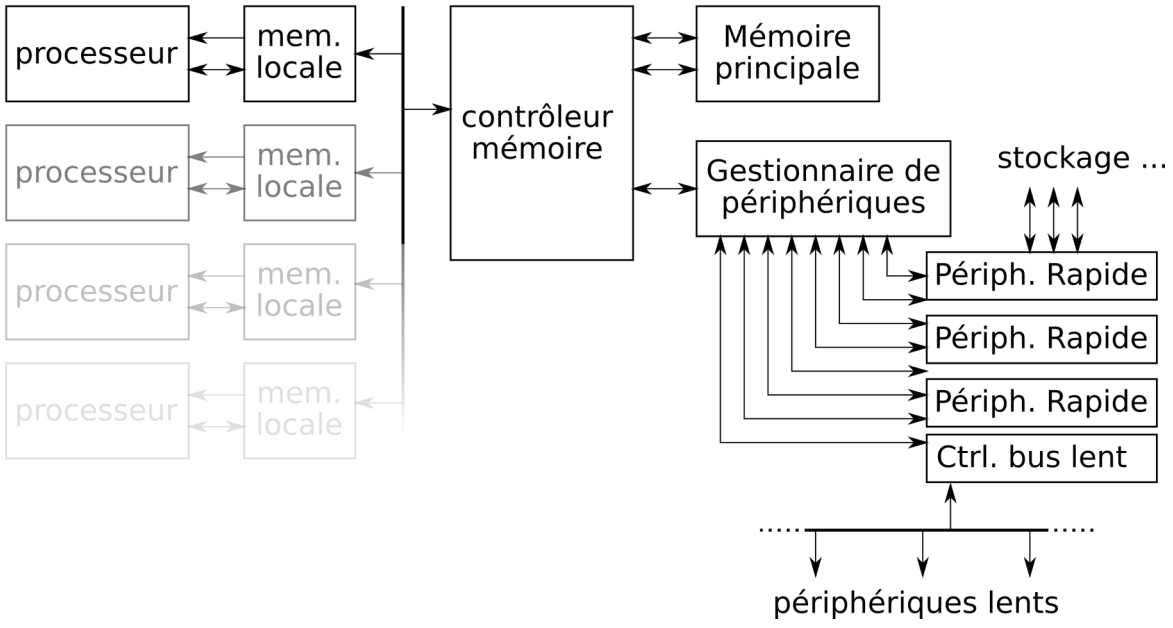
Différence SoC / µC

- Vu de l'extérieur :

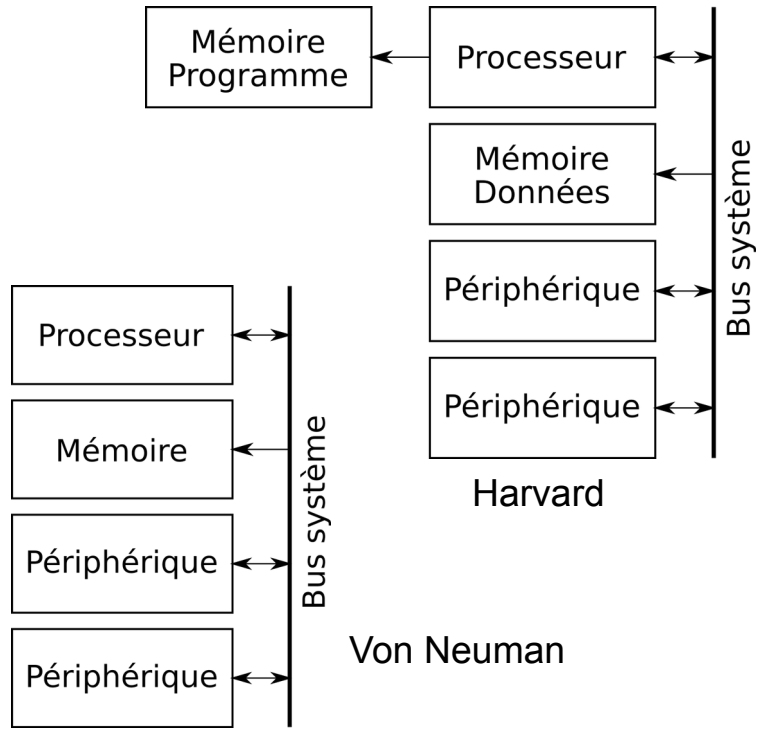
	Microcontrôleur	SoC
Freq.	De 1 à 200MHz	De 500MHz à 2GHz++
Processeur(s)	1 ou 2	De 1 à 8++
Mémoire	De 16 octets à 1Mo	De 512Mo à 8Go++
Système	Aucun ou propriétaire	~~Windows~~ / iOS / Linux / ...
Communication	SPI / I^2C / CAN / UART	BT / WiFi / Ethernet / USB / SATA
Aspect physique	Tout sur la même puce	Mémoire en puces externe… Contrôleurs d'interfaces externes
Usage	Petits périphériques, electro-ménager, jouets, interfaces	Equipements réseau, smartphones, objets connectés,

Différence SoC / µC

- Vu de l'intérieur :



SoC



Microcontrôleur

Accès aux périphériques

- Un processeur ne sait qu'échanger des données et faire des calculs dessus…

  ⇨Les périphériques se font passer pour des *données en mémoire*

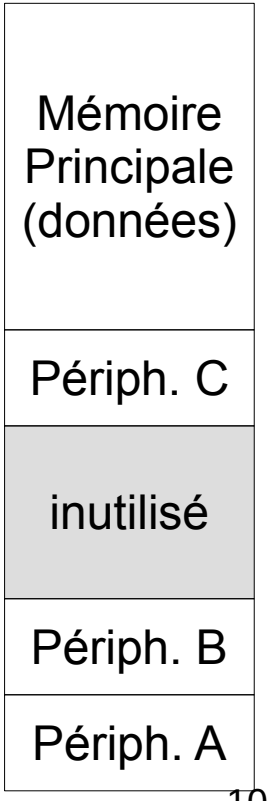
  ⇨La mémoire n'est qu'un périphérique

  - Particularité de la mémoire : ce que l'on écrit à une *adresse* donnée peut être relu à l'identique à la même *adresse*.

- Chaque périphérique présente des données qui correspondent à son état dans les *adresses* qui lui correspondent.
- Chaque périphérique interprète les données qui sont écrites sur ses adresses selon ses propres critères.

Accès aux périphériques

- Représentation des adresses mémoire
  - Il n'y a pas de règle particulière sur la répartition
  - Une adresse mémoire liée à un périphérique est appelée registre
  - Un périphérique utilise généralement plusieurs registres
  - L'adresse de chaque registre est fixe et donnée dans la datasheet
  - La signification des données présentes dans chaque registre est décrite dans la datasheet
- Le métier du développeur sur microcontrôleur:
  - Connaitre la signification des registres utilisés
  - Maîtriser la datasheet pour obtenir rapidement les détails nécessaires (composant connu)
  - Exploiter *rapidement* un composant inconnu



Accéder à un périphérique en C

- Un fichier de définition (.h) est fourni.
  - Le fichier est spécifique à chaque composant
  - Le fichier fourni un nom à chaque registre du microcontrôleur (il peut y en avoir plusieurs centaines…)

- Rappels en C au tableau
  - Pointeurs
  - Cast
  - Types construits

Accéder à un périphérique en C

- Quick'n'Dirty
  - Pour un registre TOTO situé à l'adresse 0x1234

    #define TOTO (*((unsigned int *)0x1234))

    - 0x1234 est de type int (adresse de TOTO)
    - (unsigned int *)0x1234 est de type unsigned int *
    - *((unsigned int *)0x1234) est la valeur pointée par 0x1234

      est le contenu de TOTO

Accéder à un périphérique en C

- Plus propre...
  - Pour un registre TOTO situé à l'adresse 0x1234

```
typedef     unsigned int    REGISTRE;
const       REGISTRE        *PTOTO = 0x1234;
#define     TOTO            (*PTOTO)
```

- Problèmes :
  - Beaucoup de lignes pour pas grand-chose
  - Pollution de l'espace de noms
  - Le code obtenu n'est pas (peu) portable

- Solutions
  - Syntaxe spécifique (non standard au C)
  - Formalisation plus haut niveau en passant par des types évolués (propres aux périphériques)

Accéder à un périphérique en C

- Définition d'un type construit pour chaque périphérique, représentant l'ensemble des registres qui lui sont associés :

```
Typedef    struct   _tototype {
     REGISTRE    TOTO1;
     REGISTRE    TOTO2;
} *pointeur_vers_periph_toto;
```

- Définition de l'adresse de base du périphérique:

```
#define     BASE_TOTO ((pointeur_vers_periph_toto) 0x1234)
```

- Utilisation par référencement classique :
  - (*BASE_TOTO).TOTO1
  - BASE_TOTO->TOTO1

Masquage de Données

- Comment lire un sous ensemble de bits ?
- Comment écrire quelques bits sans modifier les autres ?

- Utilisation d'un masque et d'opérations logiques.

- Rappels sur les opérations logiques en C
  - Booléenes (&& || !)
  - Bit à bit (& | ~ ^)
  - Décalages de bits