# Les systèmes embarqués

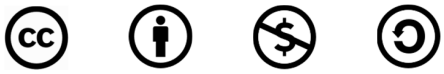
## Complément programmation C

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

REDS

Etienne Messerli
Référence: cours de Daniel Rossier

mars 2019







## Plan présentation

- Accès au E/S
  - Types
  - Déclarations variables et constantes
  - Pointeurs
- Accès mémoire et I/O (FPGA)

# Accès au E/S

- Les microcontrôleurs ARM utilise usuellement des données 32 bits
- L'accès aux entrées/sorties se fait par un bus spécifique du microcontrôleur
  - il peut avoir différentes tailles : 32, 16 ou 8 bits
- Les accès généré depuis le microcontrôleur doivent respecté le format du bus des entrées/sorties



# Type en C

- Le langage C fourni différents types qui permettent la manipulation d'information de différentes tailles
- Voici les principaux types pour des nombres entiers:
  - char: caractère, données 8 bits, 1 byte
  - short: données de 16 bits, 2 bytes
  - long: données de 32 bits, 4 bytes
  - long long: données de 64 bits, 8 bytes

# Types en C

| Type de donnée | Signification | Taille | Plage de valeurs standardisée |
|---|---|---|---|
| char | Caractère | 8 bits / 1 byte | -128 à 127 |
| unsigned char | Caractère non signé | 8 bits / 1 byte | 0 à 255 |
| short int | Entier court | 16 bits / 2 bytes | -32 768 à 32 767 |
| unsigned short int | Entier court non signé | 16 bits / 2 bytes | 0 à 65 535 |
| int | Entier | 32 bits / 4 bytes | -2 147 483 648 à 2 147 483 647 |
| unsigned int | Entier non signé | 32 bits / 4 bytes | 0 à 4 294 967 295 |
| long int | Entier long | 32 bits / 4 bytes | -2 147 483 648 à 2 147 483 647 |
| unsigned long int | Entier long non signé | 32 bits / 4 bytes | 0 à 4 294 967 295 |
| long long | Entier long 64 bits | 64 bits / 8 bytes | -9 223 372 036 854 775 807 à +9 223 372 036 854 775 807 |
| unsigned long long | Entier long 64 bits non signé | 64 bits / 8 bytes | 0 à 18 446 744 073 709 551 615 (0xFFFFFFFFFFFFFFFF) |
| float | Flottant (réel) | 32 bits / 4 bytes | $3.4*10^{-38}$ à $3.4*10^{38}$ |
| double | Flottant double | 64 bits / 8 bytes | $1.7*10^{-308}$ à $1.7*10^{308}$ |



# Déclaration de variables

- Toute variable doit être déclarée avant son utilisation.
- Exemple de déclaration généralement utilisée pour les variables :

```
char      c, ligne[40];
int       nbr_entier;  //signé
long      nbr_32bits;
unsigned  nbr_positif;
short     nbr_16bits;
unsigned short   nbr_pos_16bits;
```

# Déclaration de constantes

- Très utilisé pour toutes valeurs fixes. Mot clé :

```
#define <NAME>    valeur;
```

- Déclaration au début du programme
- Identificateur tout en majuscule
- Exemple de déclaration de constantes :

```
#define MAX           100;
#define CAR_ZERO      '0';
#define VAL_HEXA      0x3F8D;
#define MASK_BIT(x)     (1 << x)
```



# Pointeurs en C

- **Pointeurs**

```
int var;        /* déclaration d'une variable (réservation mémoire) */
int *pvar;      /* déclaration d'un pointeur (réservation mémoire) */
pvar = &var;    /* écrit l'adresse de la variable dans le pointeur
                   pvar prend la valeur de l'adresse mémoire de var */
*pvar = 3;      /* écrit la valeur 3 dans la variable var (déréférencement)
                   on écrit dans la position mémoire où est stocké var */
```

# Types et pointeurs

```
int var;
int *pvar;


/* accès à la variable par valeur */
var = 15;

printf("valeur = %d\n", var) ; /* valeur = 15 */


/* accès à la variable par pointeur/référence */
pvar = &var;
*pvar = 30;

printf("valeur = %d\n", var) ; /* valeur = 30 */
```



# Accès mémoire (1/2)

- Accès aux adresses physiques (sans MMU) ou virtuelles (avec MMU)

0x20AF630 Adresse physique/virtuelle du registre GPIO_P1

#define GPIO_P1 *(unsigned int *) 0x20AF630

↓

opération de cast (pointeur sur adresse physique/virtuelle)

GPIO_P1 = 0x12345678; /* écrit la valeur 0x12345678 dans le registre GPIO_P1 */

int var;

var = GPIO_P1; /* Lit le contenu du registre GPIO_P1 et l'écrit dans var */

# Accès mémoire (2/2)

```
/* Définition d'un masque pour bit x   */
#define MASK_BIT(x)    (1 << x)

/* Accès aux bits d'un registre */
#define  GPIO_P1  *(unsigned int *) 0x020AF630

#define  MASK_B1  MASK_BIT(1)  /* masque pour le bit no 1 de GPIO_P1 */

GPIO_P1 |= MASK_B1 ;              /* met à 1 le bit no 2 */
GPIO_P1 &= ~ MASK_B1;             /* met à 0 le bit no 2 */

GPIO_P1 ^= MASK_B1;               /* toggle du bit no 2 */

if (GPIO_P1 & MASK_B1 != 0) { … }  /* test du bit 2 */

#define  P1_MSK    0x38   /* masque pour les bits no 3-4-5 de GPIO_P1 */

/* Ecrit 6 dans les bits 3-4-5 */

GPIO_P1 = (GPIO_P1 & ~P1_MSK) | (0x6 << 3);
```



# Accès I/O FPGA

- Les accès à la FPGA sont sur 16 bits, utiliser `short`
- L'état dépend de I/O et peut changer à tout moment il faut interdire la mise en cash, utiliser `volatile`
- Utiliser un type prédéfinit

```
typedef volatile unsigned short vushort;
```

- Exemples :

```
typedef volatile unsigned short vushort;

#define FPGA_BASE_ADDR      0x1A000000
#define FPGA_CST            *(vushort *)(FPGA_BASE_ADDR + 0x0)
#define FPGA_BUTTON         *(vushort *)(FPGA_BASE_ADDR + 0x2)
```