Document 3

En langage C, l'accès à un registre ***mappé*** en mémoire (qui a une adresse dans l'espace mémoire), se fait en utilisant un pointeur. On part de l'adresse donnée explicitement par une constante entière, généralement en codage hexadécimal, ici en vert **0x8000** ou **0x8004**.

La taille du registre (8, 16 ou 32 bits) définit le type de base du langage C qui doit être utilisé (unsigned char, short, int, ou mieux **uint8_t**, uint16_t , uint32_t définis dans **<stdint.h>**). A partir de ce type de base, il faut construire un pointeur, par ex. ici **(uint8_t *)**, c'est l'objet de l'opération de ***cast*** (changement de type) en bleu dans le code.

Le mot clé « **volatile** » précise au compilateur qu'il ne s'agit pas réellement d'une mémoire, la donnée écrite n'est pas réellement mémorisée, il faut donc que tous les accès en lecture et en écriture soit effectivement réalisés et éviter des optimisations du compilateur qui pourraient en supprimer certains.

Pour acccéder au registre, ce n'est pas l'adresse qu'on souhaite manipuler, mais la données qui s'y trouve, il faut donc ***déréférencer*** le pointeur (accéder à la donnée qui se trouve à cette adresse) avec l'opérateur de ***déférence*** * (en rouge ici).

Finalement, il reste à définir une **constante symbolique** (ici en **orange**) pour pouvoir accèder facilement au registre en écriture (dans une affectation) ou en lecture (dans une condition, un test).

Les adresses sont des multiples de 4 pour simplifier l'accès aux registres en un cycle car le CPU possède un bus de données de largeur 32 bits (4 octets).

```
//test.c
#include <stdint.h>

#define   LEDRADDRESS   (*(volatile uint8_t *)  0x8000)
#define   SWADDRESS   (*(volatile uint8_t *)  0x8004)

int main (void){
 int i=1;
 while (1){
  LEDRADDRESS =  i++;
  while (SWADDRESS & 0x01);
  }
return 0;
}
```

Écriture des LED à l'adresse 0x8000

Lecture des switchs à l'adresse 0x8004