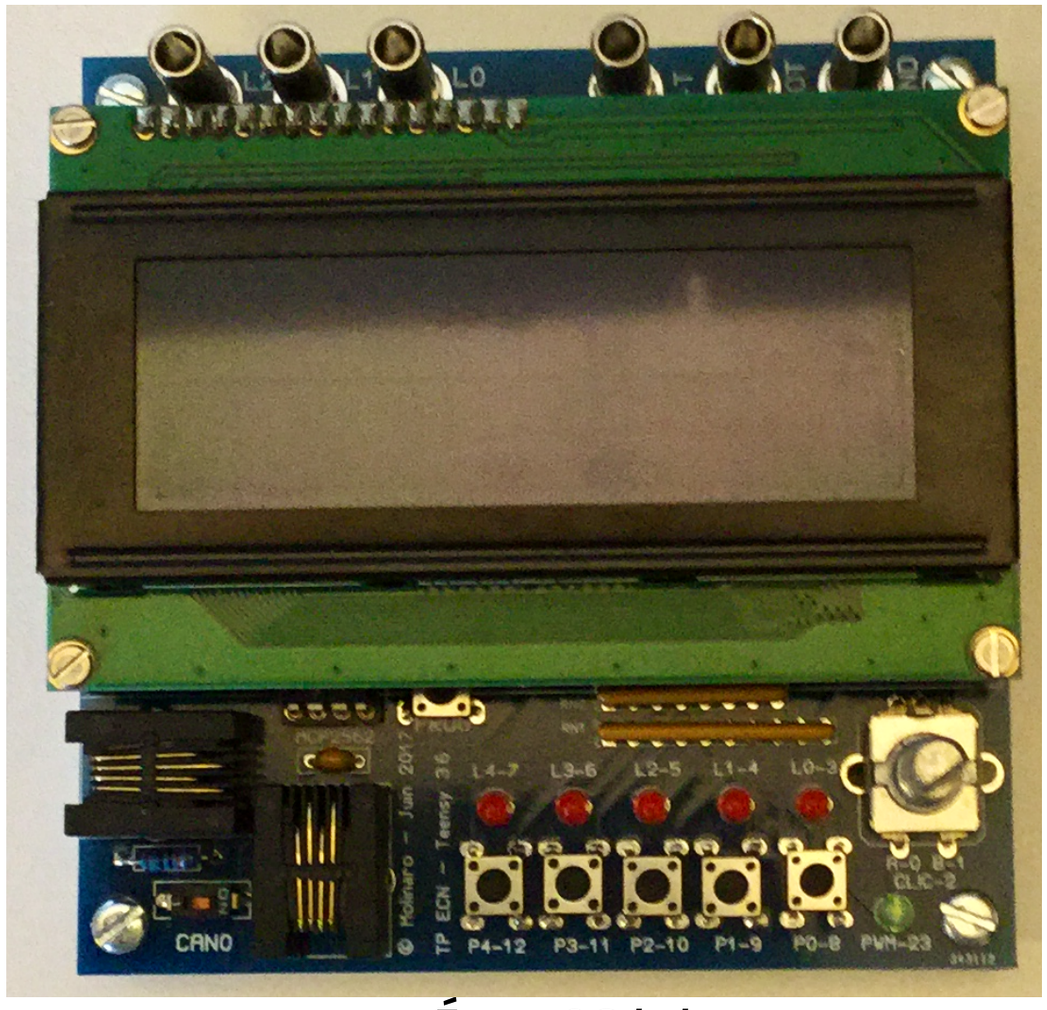
Temps Réel



Étape 06-lcd

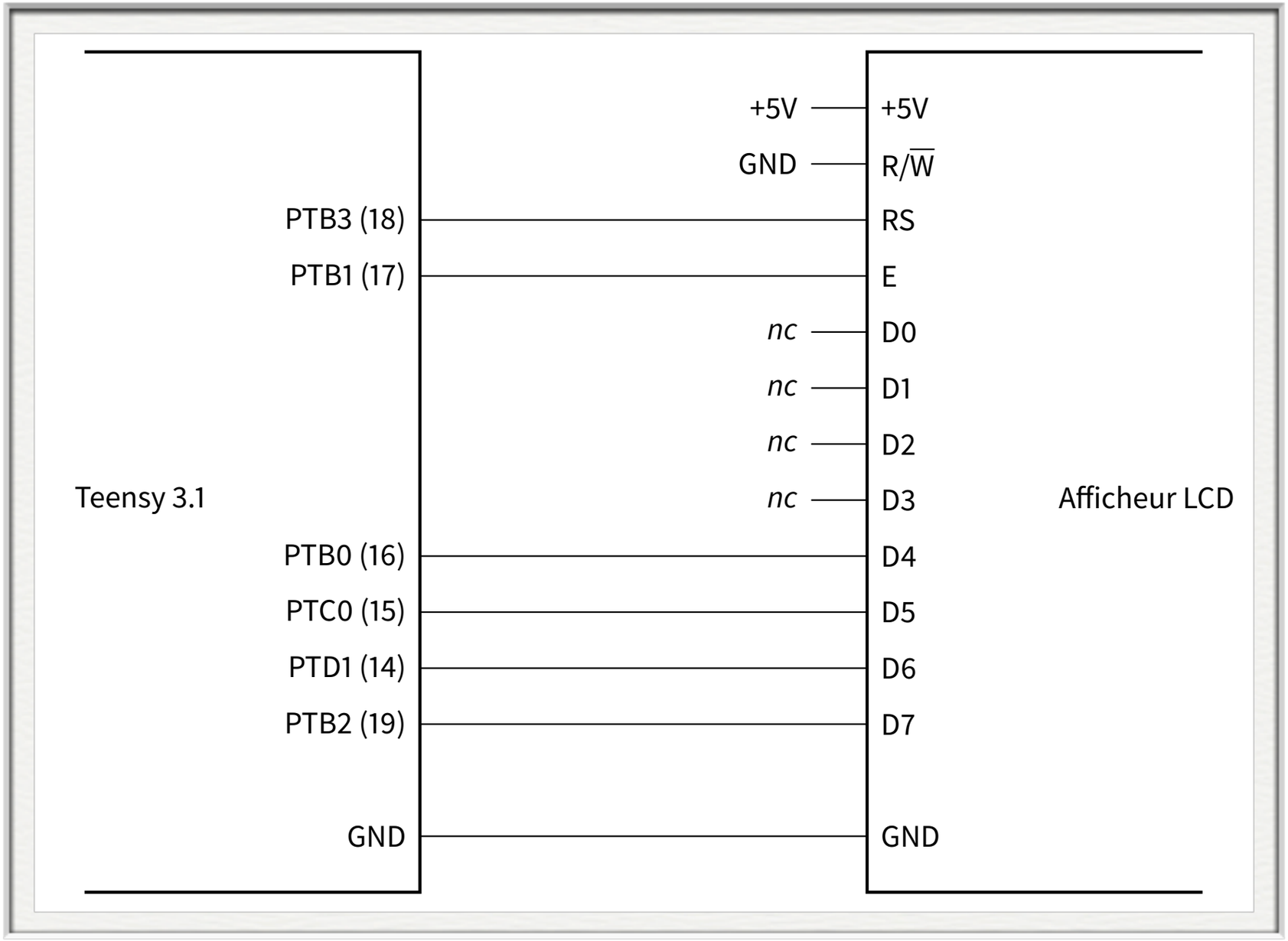
Description de cette partie

Objectif : disposer de fonctions permettant d'afficher sur l'afficheur LCD des chaînes de caractères et des nombres.

Pour cela, l'archive **06-files.tar.bz2** contient les fichiers **lcd-wo-fault-mode.h** et **lcd-wo-fault-mode.cpp**. Ils sont nommés ainsi car ils ne prennent pas en charge le mode logiciel **FAULT** (ce qui sera fait à partir de l'étape 09).

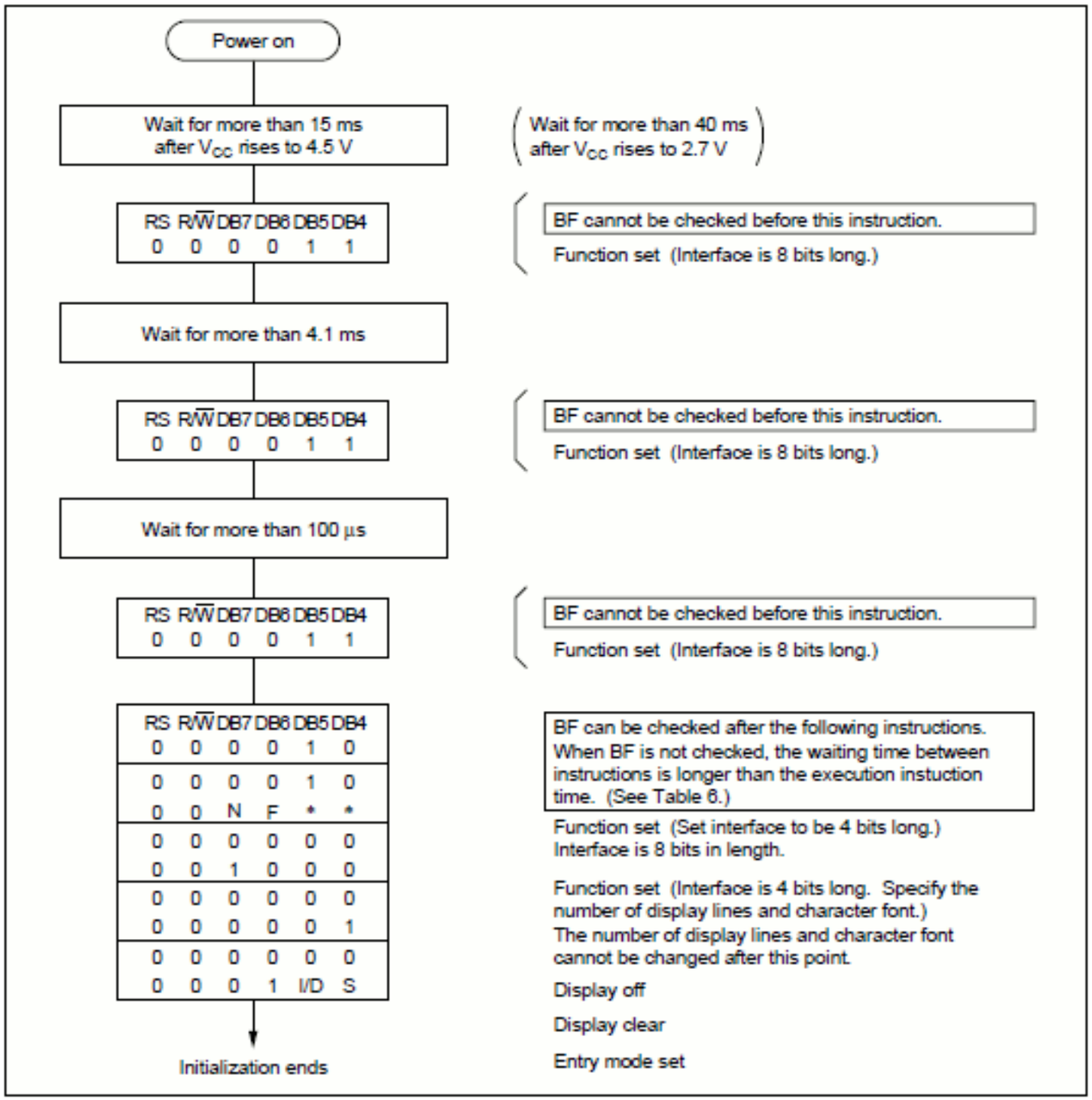
Le travail à faire est d'utiliser les fonctions d'affichage pour afficher du texte et des nombres.

Connexion de l'afficheur LCD



La connexion est de type « 4 bits ».

Initialisation d'un afficheur LCD en mode 4 bits



http://web.alfredstate.edu/faculty/weimandn/lcd/lcd_initialization/lcd_initialization_index.html

CENTRALE NANTES

Initialisation : fonction `setupLCD`

La fonction **setupLCD** programme les ports correspondants aux signaux d'interface, et initialise l'afficheur LCD suivant la procédure illustrée à la page précédente. Elle est automatiquement exécutée lors du démarrage du micro-contrôleur.

```
static void setupLCD (INIT_MODE) {
//--- Step 1: configure ports
  pinMode (LCD_D4, DigitalMode::OUTPUT) ;
  pinMode (LCD_D5, DigitalMode::OUTPUT) ;
  pinMode (LCD_D6, DigitalMode::OUTPUT) ;
  pinMode (LCD_D7, DigitalMode::OUTPUT) ;
  pinMode (LCD_RS, DigitalMode::OUTPUT) ;
  pinMode (LCD_E,  DigitalMode::OUTPUT) ;
//--- Step 2: wait for 15 ms
  ....................
}

MACRO_INIT_ROUTINE (setupLCD) ;
```

CENTRALE NANTES

Fonctions d'affichage (1/3)

```
void clearScreen (USER_MODE) ;
```
Efface l'afficheur, en plaçant le curseur au début de la ligne du haut.

```
void gotoLineColumn (USER_MODE_ const uint32_t inLine, const uint32_t inColumn) ;
```
Déplace le curseur sans rien écrire. `inLine` vaut `0` (ligne du haut) à `3` (ligne du bas), `inColumn` vaut `0` (colonne de gauche) à 19 (colonne de droite).

```
void printString (USER_MODE_ const char * inString) ;
```
Imprime la chaîne de caractères à la position du curseur.

```
void printChar (USER_MODE_ const char inChar) ;
```
Imprime le caractère à position du curseur.

```
void printSpaces (USER_MODE_ const uint32_t inCount) ;
```
Imprime `inCount` espaces à position du curseur.

CENTRALE NANTES

Fonctions d'affichage (2/3)

```
void printUnsigned (USER_MODE_ const uint32_t inValue) ;
```
Imprime le nombre non signé à l'endroit du curseur.

```
void printUnsigned64 (USER_MODE_ const uint64_t inValue) ;
```
Imprime le nombre non signé à l'endroit du curseur.

```
void printSigned (USER_MODE_ const int32_t inValue) ;
```
Imprime le nombre signé à l'endroit du curseur.

CENTRALE NANTES

Fonctions d'affichage (3/3)

```
void printHex1 (USER_MODE_ const uint32_t inValue) ;
```
Imprime à l'endroit du curseur les 4 bits de poids faible du nombre non signé de 32 bits sous la forme d'un chiffre hexadécimal.

```
void printHex2 (USER_MODE_ const uint32_t inValue) ;
```
Imprime à l'endroit du curseur l'octet de poids faible du nombre non signé de 32 bits sous la forme de deux chiffres hexadécimaux.

```
void printHex4 (USER_MODE_ const uint32_t inValue) ;
```
Imprime à l'endroit du curseur les deux octets de poids faible du nombre non signé de 32 bits sous la forme de quatre chiffres hexadécimaux.

```
void printHex8 (USER_MODE_ const uint32_t inValue) ;
```
Imprime à l'endroit du curseur le nombre non signé de 32 bits sous la forme de 8 chiffres hexadécimaux.

```
void printHex16 (USER_MODE_ const uint64_t inValue) ;
```
Imprime à l'endroit du curseur le nombre non signé de 64 bits sous la forme de 16 chiffres hexadécimaux.

CENTRALE NANTES

Modifier `time.h` et `time.cpp`

Si vous jetez un coup d'œil sur le code contenu dans le fichier **lcd-wo-fault-mode.cpp**, vous verrez qu'il y a deux fonctions d'attente différentes qui sont appelées :

- **busyWaitDuring**, appelée à partir des fonctions en mode **USER** ;
- **busyWaitDuring_initMode**, appelée à partir des fonctions en mode **INIT**.

La première est déjà implémentée dans **time.cpp**, la seconde ne l'est pas.

Donc, dans le fichier **time.h**, ajouter :

```
void busyWaitDuring_initMode (INIT_MODE_ const uint32_t inDelayMS) ;
```

Dans le fichier **time.cpp**, implémenter la fonction :

```
void busyWaitDuring_initMode (INIT_MODE_ const uint32_t inDelayMS) {
  const uint32_t COUNTFLAG_MASK = 1 << 16 ;
  for (uint32_t i=0 ; i<inDelayMS ; i++) {
    while ((SYST_CSR & COUNTFLAG_MASK) == 0) {} // Busy wait, polling COUNTFLAG
  }
}
```

Vous constatez que l'implémentation de cette fonction est identique à celle de **busyWaitDuring** : les modes logiciels peuvent provoquer une duplication du code. De toute façon, la fonction **busyWaitDuring** sera modifié quand l'attente passive sous exécutif sera implémentée. La fonction **busyWaitDuring_initMode** ne sera pas modifiée, l'exécutif n'étant pas actif durant l'initialisation.

Travail à faire

Dupliquer le répertoire de l'étape précédente et renommez-le par exemple **06-lcd**.

Ajoutez aux sources les deux fichiers **lcd-wo-fault-mode.h** et **lcd-wo-fault-mode.cpp** contenus dans l'archive **06-files.tar.bz2**.

Modifier la fonction **setup** de façon à écrire `"Hello!"` au début de la première ligne.

Modifier la fonction **loop** de façon à ce qu'elle exécute un délai de 500 ms, puis affiche au début de la deuxième ligne le nombre de ses exécutions. On peut ajouter la complémentation d'une led.

Dans un deuxième temps, modifier la fonction **loop** que vous avez écrite de façon à afficher le nombre *modulo 20* de ses exécutions (il y a un (petit) piège). Note : en C / C++, l'opérateur modulo est « % ».

Question : la fonction **setupLCD** s'exécute en mode `INIT`. Peut-on l'exécuter en mode `BOOT` ?