



Durant les deux premiers TP, le développement de nos applications a été fait en assembleur. Il faut savoir que le langage de programmation le plus rencontré dans le domaine de l'embarqué est le 'C'. Nous devons notamment ceci aux grandes performances des compilateurs associés. Les autres langages rencontrés sont notamment le C++ et l'assembleur. Le premier pour son approche objet et ses liens forts avec le 'C' et le second pour ses performances ou lorsqu'on est amené à travailler sur de vieilles architectures.

Parmi ces langages, le 'C' est celui offrant le meilleur ratio entre performances du code (taille et/ou rapidité) et temps de développement. Le passage à l'assembleur permettant "dans certains cas" d'obtenir des codes plus performants au détriment du temps de développement.

*Le développement d'application en 'C' utilise une approche fonctionnelle. Cela signifie que pour effectuer un traitement ou communiquer avec un périphérique nous allons utiliser des fonctions, nous parlerons d'**APIs**. Par exemple, l'API ou fonction **putsXLCD('2x16 LCD')** fournie avec le compilateur C18 à l'installation permet d'afficher une chaîne de caractères sur un afficheur LCD 2x16 connecté à notre MCU. (2 lignes de 16 caractères, cf. **figure 1**). Microchip fournit un grand nombre de bibliothèques pour piloter par exemple l'ADC, les ports séries, le contrôleur USB ... Tous les sources 'C' de ces bibliothèques sont fournis sous **C:\MCC18\src**.*



figure 1 : afficheur LCD 2x16

*L'API 'putsXLCD()', se situe dans une bibliothèque (librairie) proposée par C18 (sources sous **C:\MCC18\src\pmc_common\XLCD**). Durant les deux prochaines séances de TP, vous serez amenés à développer en 'C' votre propre bibliothèque et donc vos propres APIs pour piloter un afficheur LCD 2x16 ...*

1. Comment développer une librairie pour afficheur ?

Votre travail consistera donc à développer une librairie pour un afficheur LCD 2x16. Le développement de cette librairie se fera en "C". Il aurait très bien pu être effectué en assembleur, mais nous souhaitons dans cet exercice nous focaliser sur la méthodologie de développement d'une bibliothèque. Ce travail demande un certain nombre de compétences, notamment les suivantes :

- ➔ **Connaissance des outils de développement :** MPLAB, C18 ...
- ➔ **Connaissance du langage de programmation :** C
- ➔ **Connaissance du MCU :** Gestion des GPI/O et temporisation software
- ➔ **Câblage de l'afficheur :** Électronique analogique (cf. **figure 2**)
- ➔ **Connaissance du contrôleur LCD :** composant se trouvant au dos de l'afficheur, nous allons le découvrir dans la suite de la présentation !

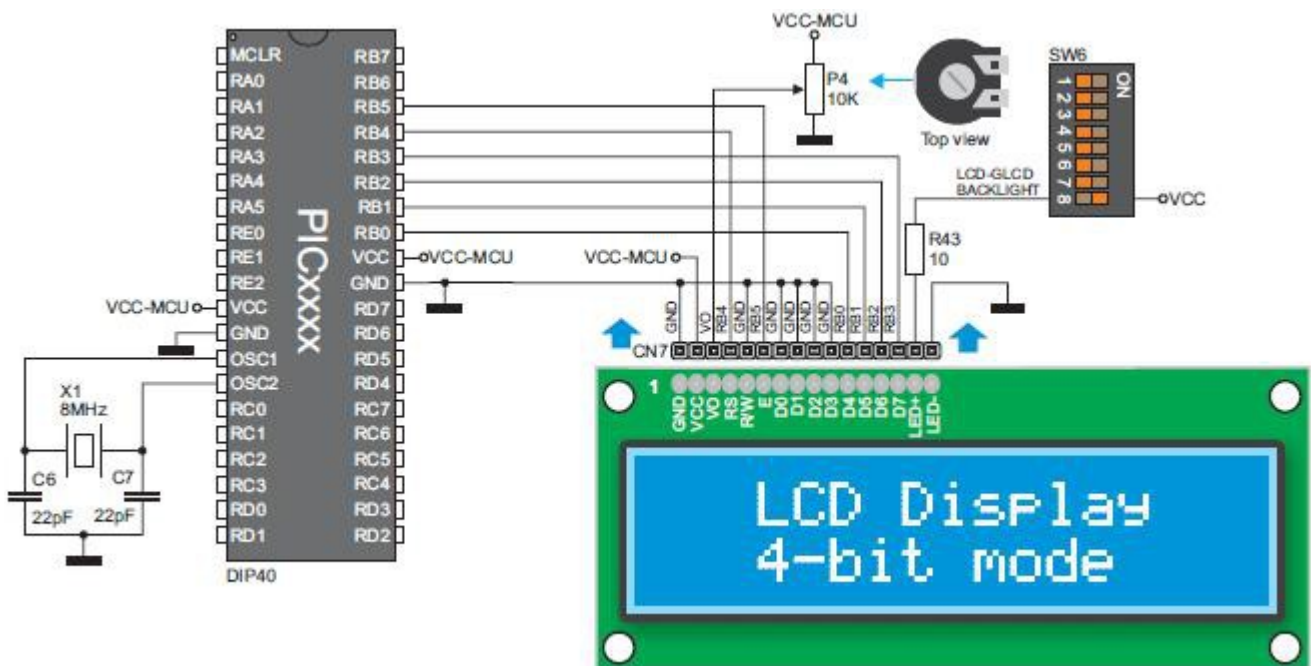


figure 2 : câblage du contrôleur LCD au PIC18F4550



*Notre afficheur peut travailler selon deux modes, 4bits ou 8bits. Nous travaillerons en mode 4bits, celui câblé par défaut sur la maquette. Cela signifie que le bus de données est sur 4 fils (D4-D7) et non sur 8 (cf. **figure 2**).*

2. Que devons-nous connaître au niveau du MCU ?

Les acquis à avoir au niveau de l'utilisation du micro-contrôleur sont relativement minces. Durant ces deux TP nous aurons seulement à envoyer des données sur les broches RB0 à RB5 du port B, qui seront connectées en sorties. Nous n'aurons donc qu'à gérer les registres TRISB et LATB. La gestion du registre TRISB se fera dans le "main()" et les définitions des macros affectées au portB se feront dans le fichier d'en-tête "portLCDUser.h" (ex: #define E_PIN_LCD LATBbits.LATB5). Ce fichier, donné ci-dessous, est très important et dénote de la philosophie à adopter pour développer notre librairie.

Nous allons chercher à **développer une librairie portable**. Cela signifie qu'au terme de ces deux séances, nous devons être capable de porter "relativement" facilement notre librairie sur d'autres MCUs, quelque soit le fabricant (Microchip, Atmel, Analog Device ...). En cas de portage, le seul fichier à modifier est **portLCDUser.h**. Ce fichier, donné ci-dessous, permet de définir les macros pour la gestion des broches ainsi que pour la gestion des délais spécifiés par le fabricant du contrôleur LCD. Il s'agit des seuls éléments à modifier en cas de portage vers un autre MCU.

Contenu du fichier d'en-tête "portLCDUser.h"	
<pre> /***** @file : portLCDUser.h @brief : include file à modifier en cas de portage *****/ #ifndef __PORTAGE_LCD_USER_HEADER__ #define __PORTAGE_LCD_USER_HEADER__ #include <delays.h> // fichier d'en-tête pour la gestion des Temporisations /** Gestion des broches - LCD 2x16 en mode 4bits */ #define E_PIN_LCD_USER LATBbits.LATB5 #define RS_PIN_LCD_USER /** à compléter ! */ #define D7_PIN_LCD_USER /** à compléter ! */ #define D6_PIN_LCD_USER /** à compléter ! */ #define D5_PIN_LCD_USER /** à compléter ! */ #define D4_PIN_LCD_USER /** à compléter ! */ /** Gestion des temporisations logicielles avec: PLLDIV=2, FOSC=HSPLL_HS et CPUDIV=OSC1_PLL2 on a: TCY = 83,2ns */ #define DelayUser_2ms() Delay1KTCYx(24) // 24.1000.TCY ~ 2ms #define DelayUser_5ms() Delay1KTCYx(61) // 61.1000.TCY ~ 5ms #define DelayUser_15ms() Delay1KTCYx(181) // 181.1000.TCY ~ 15ms #define DelayUser_400us() Delay100TCYx(48) // 48.100.TCY ~ 400us #define DelayUser_4us() Delay10TCYx(5) // 5.10.TCY ~ 4us #endif </pre>	



Prenons l'exemple de la macro "E_PIN_LCD" qui est ensuite utilisée dans la librairie. Le compilateur remplacera à chaque fois dans notre librairie "E_PIN_LCD" par "LATBbits.LATB5", ce qui signifie que l'on accédera à chaque fois à la broche n°5 du port B. Donc en cas de portage ne n'avons pas à modifier la librairie mais seulement cette macro ... pratique !

3. Comment utiliser le contrôleur LCD ?

Tout afficheur n'est jamais utilisé seul, il est toujours associé à un contrôleur. Le contrôleur utilisé dans notre cas est un **HD44780U** de chez HITACHI (cf. **figure 3**). Ce contrôleur est très couramment rencontré dès qu'il s'agit d'interfacer un afficheur LCD 2x16. Ce composant sert d'interface entre notre MCU et l'afficheur qui possède un trop grand nombre de broches pour être directement contrôlé par notre micro-contrôleur.

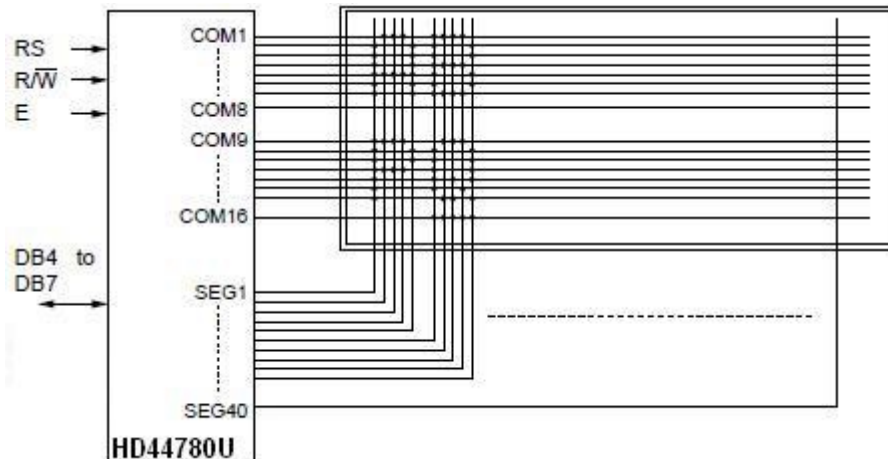
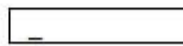
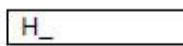


figure 3 : Schéma fonctionnel simplifié du contrôleur LCD HD44780U

Ce composant se comporte comme un esclave vis à vis de notre MCU. Pour l'utiliser il nous faudra lui envoyer des **commandes et des données**. Les commandes et les données seront passés par les broches DB4 à DB7. Prenons un exemple **figure 4** de communication entre notre MCU et le contrôleur, cet exemple est issu de la datasheet du composant.

4-Bit Operation, 8-Digit × 1-Line Display Example with Internal Reset

Step No.	RS	R/W	DB7	DB6	DB5	DB4	Display	Operation
Display on/off control								
n°1	0	0	0	0	0	0		Turns on display and cursor. Entire display is in space mode because of initialization.
n°2	0	0	1	1	1	0		
Write data to CGRAM/DDRAM								
n°3	1	0	0	1	0	0		Writes H. The cursor is incremented by one and shifts to the right.
n°4	1	0	1	0	0	0		



Le code ACSII sur 7bits du caractère "H" est : 1001000

figure 4 : exemples de cycles d'écriture d'une commande puis d'une donnée



Nous attirons votre attention sur quelques aspects :

➔ *Durant la totalité des TP nous ne ferons qu'envoyer des commandes ou des données au contrôleur. Le broche "R//W" sera donc toujours à "0" (cf. figures 4&5).*

➔ Lorsque nous envoyons des commandes ou des données au contrôleur, celles-ci sont respectivement chargées dans **LE** registre de contrôle ou **LE** registre de donnée du HD44780U. La sélection du bon registre se fait par la broche "**RS**". Si **RS** = 1, alors on écrit dans le registre de donnée, sinon on écrit dans le registre de contrôle (cf. **figures 4&5**).

➔ Pour qu'une écriture soit effective dans l'un de ces deux registres, il faut en dernier lieu mettre à "1" la broche de validation "**E**" (enable) pendant au moins **4µs** (cf. **figure 5**).

➔ Entre deux transactions (commande ou donnée), il faut attendre un certain temps correspondant au temps de traitement interne du contrôleur (cf. **figure 5**). Expérimentalement nous constatons qu'il faut attendre approximativement **2ms**.

Le chronogramme **figure 5** reprend l'exemple de la **figure 4**. Votre programme devra donc gérer l'envoi des bonnes valeurs aux bons moments sur les différentes broches. Il vous faudra également respecter scrupuleusement les contraintes temporelles minimums préconisés par le fabricant du composant.

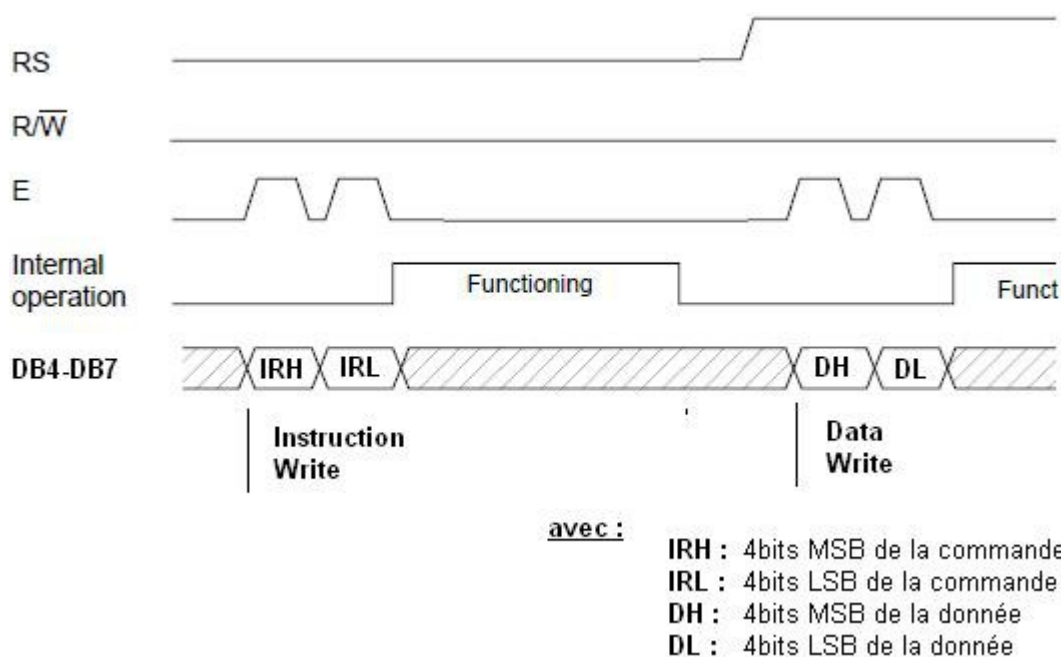


figure 5 : Chronogramme présentant les cycles d'écriture d'une commande puis d'une donnée



*Même si cela n'est pas une nécessité, le dernier aspect à présenter est le jeu de commandes du contrôleur. Une partie des commandes les plus utilisées vous sont déjà données dans le fichier d'en-tête "**LCDUser.h**" (effacer écran, effacer curseur ...). Ce fichier est donné en ANNEXE.*

4. Présentation du jeu de commandes

Cette partie n'est pas indispensable au bon déroulement des TP. En revanche elle permet d'expliquer les valeurs des différentes commandes données dans le fichier "LCDUser.h". Prenons l'exemple de la commande "LCD_COMMAND_CLEAR" qui vaut **0x01** et qui permet d'effacer l'écran. Nous constatons dans le tableau **figure 6** issu de la datasheet du contrôleur que la valeur **0x01** permet effectivement d'effacer l'écran. Les commandes sont les même en mode 4bits et 8bits.

Instructions												Execution Time (max) (when f_{op} or f_{osc} is 270 kHz)
Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 μ s
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 μ s
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 μ s
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 μ s

Instructions (cont)

Code											Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)	
Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		Description
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 μ s $t_{ADD} = 4 \mu$ s*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 μ s $t_{ADD} = 4 \mu$ s*
	I/D = 1:	Increment									DDRAM: Display data RAM CGRAM: Character generator RAM	Execution time changes when frequency changes Example: When f_{cp} or f_{osc} is 250 kHz, 37 μ s $\times \frac{270}{250} = 40 \mu$ s
	I/D = 0:	Decrement										
	S = 1:	Accompanies display shift										
	S/C = 1:	Display shift										
	S/C = 0:	Cursor move										
	R/L = 1:	Shift to the right										
	R/L = 0:	Shift to the left										
	DL = 1:	8 bits, DL = 0: 4 bits										
	N = 1:	2 lines, N = 0: 1 line										
	F = 1:	5 \times 10 dots, F = 0: 5 \times 8 dots										

figure 6 : détail du jeu de commandes du contrôleur



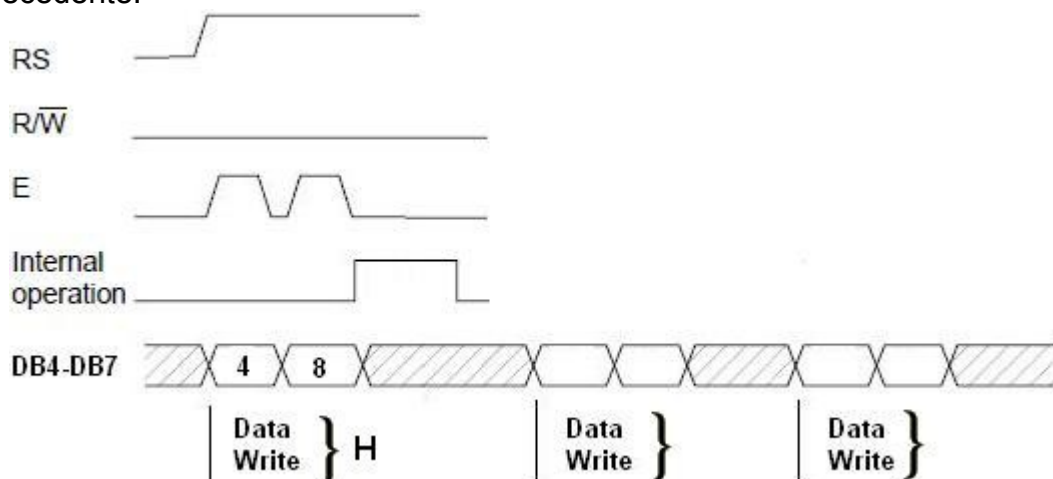
Bien, vous êtes maintenant prêt à attaquer le TP ! ... vous pouvez d'ailleurs dès à présent commencer à compléter les fichiers sources présents juste après ces questions

Cependant, dans un soucis pédagogique, il vous est demandé un petit travail préparatoire. Il sera vérifié en début de séance et vous permettra d'anticiper une perte de temps non négligeable dans l'avancement des TP (nb ★ = difficulté).

1. (★) Pourquoi, à partir de ce TP, les différents exercices seront-ils écrits en "C" ?
2. (★) Pourquoi doit-on passer par un contrôleur pour piloter l'afficheur LCD 2x16 ?
3. (★★) Complétez la figure ci-dessous de façon à afficher "**HAB**" sur l'afficheur. Le code ACSII du caractère A est **0x41**.

Step No.	RS	R/W	DB7	DB6	DB5	DB4	Display	Operation
Write data to CGRAM/DDRAM							H_	Writes H The cursor is incremented by one and shifts to the right.
1	0	0	1	0	0			
1	0	1	0	0	0			
Write data to CGRAM/DDRAM							HA_	Writes A The cursor is incremented by one and shifts to the right.
Write data to CGRAM/DDRAM							HAB_	Writes B The cursor is incremented by one and shifts to the right.

4. (★★) Complétez le chronogramme ci-dessous correspondant à l'affichage précédente.





Listing du programme ex3.c

```

/*****
@file : ex3.c
@brief : programme de test pour développer la librairie LCDUser.c
@author :
last modification :
*****/

/** Configuration bits */
#pragma config PLLDIV=2, CPUDIV=OSC1_PLL2, FOSC=HSPLL_HS, BOR=OFF, WDT=OFF, MCLRE=ON

/** Includes files */
#include <p18f4550.h>
#include <LCD.h>
#include "LCDUser.h"

/** chaînes de caractères pour le Debug */
char testString1[] = "  ENSICAEN  ";
char testString2[] = "Bienvenue a l'ENSICAEN";

/*****
*** PROGRAMME PRINCIPAL ***
*****/
void main() {

/** Configuration des broches RB0 à RB5 en sorties */

    /** à compléter ! */

    /** n°6 : Initialisation du LCD */

        LCD_Init();

    /** n°1 : Envoi d'un caractère au LCD */
    /** n°1' : Ecriture fonction LCD_Write_Register(char DR_or_IR); */
    /** n°1" : Ecriture fonction LCD_write_Data(char caracter); */

        LCD_Char('d');

    /** n°2 : Envoi d'une chaîne de caractère au LCD */

        LCD_String(testString1);

    /** n°3 : Envoi d'une commande au LCD */

        LCD_Command(LCD_COMMAND_PANRIGHT);

    /** n°4 : Envoi d'un caractère au LCD + positionnement curseur */
    /** n°4' : Ecriture fonction LCD_Cursor_XY(unsigned short row, unsigned short col); */

        LCD_Char_XY(2, 3, 'D');

    /** n°5 : Envoi d'une chaîne de caractère au LCD + positionnement curseur */

        LCD_String_XY(2, 5, testString2);

    while (1);
}

```




Listing du programme LCDUser.h

```
/*
*****
@file : LCDUser.h
@brief : include file pour travailler avec la librairie LCDUser.c
@author :
last modification :
*****
*/
#ifndef __LCD_HEADER__
#define __LCD_HEADER__

#include "portLCDUser.h"

/*
**** Caractéristiques LCD ****
#define ROW_NB      2           // nombre de lignes
#define COLUMN_NB   16          // nombre de colonnes

**** Commandes standard pour l'affichage ****
#define LCD_COMMAND_CLEAR      0x01 // Clear screen, home cursor
#define LCD_COMMAND_HOME      0x02 // Home cursor, unshift display
#define LCD_COMMAND_BACKSPACE  0x10 // Move cursor left one
#define LCD_COMMAND_FWDSPACE   0x14 // Move cursor right one
#define LCD_COMMAND_PANLEFT    0x18 // Move screen left one
#define LCD_COMMAND_PANRIGHT   0x1C // Move screen right one
#define LCD_COMMAND_CURSOROFF  0x0C // clear cursor
#define LCD_COMMAND_SECONDRROW  0xC0 // go to second row

**** Sélection registre Data ou Instruction (contrôleur LCD) ****
#define REG_IR  0 // écriture dans le registre d'instruction
#define REG_DR  1 // écriture dans le registre de donnée

**** Déclaration fonctions externes pour ex3.c ****
extern void LCD_Char_User(char character);
extern void LCD_String_User(char StringData[16]);
extern void LCD_Command_User(char command);
extern void LCD_Char_XY_User(unsigned short row, unsigned short col, char character);
extern void LCD_String_XY_User(unsigned short row, unsigned short col, char StringData[16]);
extern void LCD_Init_User(void);

#endif
```



Listing du programme LCDUser.c

```

/*****
@file : LCDUser.c
@brief : Librairie pour piloter un afficheur LCD 2x16
@author :
last modification :
*****/
#include "LCDUser.h"

/**** Déclaration variables globales et fonctions (locales à LCDUser.c)****/
void LCD_Write_Register(char DR_or_IR);
void LCD_write_Data(char character);
void LCD_Cursor_XY(unsigned short row, unsigned short col);

/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/*XXXXXXXXXXXXX FONCTIONS PRIVEES xxxxxxxxxxxxxx*/
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/

/****
*** FONCTION : Envoie d'une DATA 4bits au LCD ***
*** @param   character : donnée 8bits >> envoi des 4LSB ***
*****/
void LCD_write_Data(char character){

    /*** envoie des 4bits LSB de character vers le LCD ***/
    //D4_PIN_LCD_USER = (0x01 & character); //envoi du bit de poids faible de....

    /*** à compléter ! ***/
}

/****
*** FONCTION : Envoie d'une COMMANDE au LCD ***
*** @param   DR_or_IR : registre de destination ***
*****/
void LCD_Write_Register(char DR_or_IR){

    /*** Sélection du registre à écrire : instruction ou data ***/

    /*** à compléter ! ***/

    /*** écriture dans le registre sélectionné ***/

    /*** à compléter ! ***/

    /*** delay de 2ms : spécifications électriques ***/
    DelayUser_2ms();
}

/****
*** FONCTION : Positionnement curseur ***
*** @param   row : n° de la ligne (position curseur) ***
*** @param   col : n° de la colonne (position curseur) ***
*****/
void LCD_Cursor_XY(unsigned short row, unsigned short col){

    /*** à compléter ! ***/
}

```

```
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/*XXXXXXXXXXXXX FONCTIONS PUBLIQUES XXXXXXXXXXXXXXX*/
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
```

```

/*****/
/** FONCTION : Envoie caractère au LCD      **/
/** @param   character : caractère à envoyer **/
/*****/
void LCD_Char_User(char character){

char characterL, characterH;

    /** Sauvegarde des 4bits LSB et 4 bits MSB du caractère **/

        /** à compléter ! **/

    /** envoie des 4bits MSB du caractère sur le portB **/

        /** à compléter ! **/

    /** envoie des 4bits LSB du caractère sur le portB **/

        /** à compléter ! **/
}

/*****/
/** FONCTION : Envoie chaîne de caractère au LCD      **/
/** @param   StringData : pointeur sur la chaîne de caractères **/
/*****/
void LCD_String_User(char *StringData){

    /** à compléter ! **/
}

/*****/
/** FONCTION : Envoie d'une commande au LCD      **/
/** @param   commande : commande sur 8bits à envoyer **/
/*****/
void LCD_Command_User(char commande){
char commandeL, commandeH;

    /** Sauvegarde des 4bits LSB et 4 bits MSB de la commande**/

        /** à compléter ! **/

    /** envoie des 4bits MSB de la commande sur le portB **/

        /** à compléter ! **/

    /** envoie des 4bits LSB de la commande sur le portB **/

        /** à compléter ! **/
}

```

```

/*****
*** FONCTION : Envoie caractère au LCD + positionnement curseur ***
*** @param   row : n° de la ligne (position curseur) ***
*** @param   col : n° de la colonne (position curseur) ***
*** @param   character : donnée à envoyer ***
*****/
void LCD_Char_XY_User(unsigned short row, unsigned short col, char character){

    /*** Positionnement du curseur ***/

        /*** à compléter ! ***/

    /*** envoie d'un caractère à la position courante du curseur ***/

        /*** à compléter ! ***/

}

/*****
*** FONCTION : Envoie chaîne de caractères au LCD + positionnement curseur***
*** @param   row : n° de la ligne (position curseur) ***
*** @param   col : n° de la colonne (position curseur) ***
*** @param   StringData : pointeur sur la chaîne de caractères ***
*****/
void LCD_String_XY_User(unsigned short row, unsigned short col, char *StringData){
    unsigned short i=0;

    /*** Positionnement du curseur ***/

        /*** à compléter ! ***/

    /*** Envoie d'une chaîne de caractères à la position courante du curseur ***/

        /*** à compléter ! ***/

}

/*****
*** FONCTION : Initialisation LCD ***
*****/
void LCD_Init_User(void){
#define LCD_COMMAND_FCTSET8    0b0011    // Function set 8bits (initialisation)
#define LCD_COMMAND_FCTSET4    0b0010    // Function set 8bits (initialisation)

    /*** PHASE D'INITIALISATION ***/

        /*** à compléter ! ***/

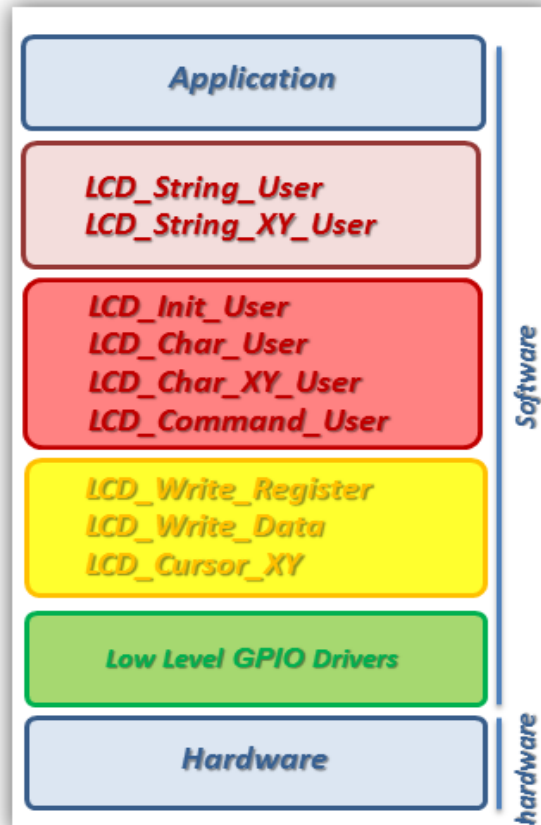
    /*** PHASE DE CONFIGURATION ***/

        /*** à compléter ! ***/

}

```


Votre travail consiste à développer une bibliothèque de fonctions C permettant de piloter l'afficheur LCD présent sur la maquette de développement. Comme beaucoup de bibliothèque, notre librairie sera découpée en couches. Nous parlons souvent de stack. Observons le découpage en couche de notre librairie :



Ce découpage en couche représente graphiquement les dépendances des fonctions entre elles. Par exemple, la fonction `LCD_Char_User` utilise les fonctions `LCD_Write_Register_User` et `LCD_Write_Data_User`. La fonction `LCD_String_User` utilise quant à elle la fonction `LCD_Char_User`. **Dans un premier temps, votre travail consistera donc à développer les fonctions C de plus bas niveau de la bibliothèque.**

1. Travail pratique (ex3 : partie 1) : afficher un caractère

Durant cette première séance de 3h, votre cahier des charges est d'afficher **un caractère** à la position courante du curseur. Une librairie existe déjà (LCD.lib), donc n'hésitez pas à vous aider des APIs déjà existantes pour développer votre code en ajoutant le fichier d'en-tête "LCD.h". Votre travail va se découper en quatre parties.

1. Dans un premier temps il vous faut configurer les broches n°0 à n°5 du port B en sortie et compléter le fichier "**portLCDUser.h**". Ce fichier contient des macros qui seront ensuite utilisées dans votre librairie.
2. Dans un second temps, vous allez devoir écrire une fonction (**LCD_write_Data(char character)**) prenant en paramètre d'entrée une donnée sur 8bits. Cette fonction, locale à votre librairie **LCDUser.c**, devra envoyer les quatre bits de poids faible de la valeur passée en entrée vers les broches D4-D7 du contrôleur LCD.
3. Dans cette troisième partie, votre travail consistera à écrire une nouvelle fonction (**LCD_write_Register(char DR_or_IR)**) permettant de sélectionner le registre du contrôleur avec lequel on souhaite travailler (reg. contrôle ou reg. donnée). Une fois le registre sélectionné, en activant le signal **Enable**, on écrira la valeur présente sur les broches D4-D7 du contrôleur vers le registre de destination.
4. Et pour conclure, vous pourrez alors écrire la toute première API de votre librairie (**LCD_Char_User(char character)**). Cette API permet d'afficher un caractère à la position courante du curseur. La position du curseur est automatiquement incrémentée par le contrôleur à la réception d'un caractère. **Les APIs que vous développerez par la suite dépendront de celle-ci !**

➡  Lire et compléter les listing des programmes **ex3.c**, **LCDUser.c**, **LCDUser.h** et **portLCDUser.h**.

➡  Créer un projet **ex3** dans votre répertoire de travail (cf. **ANNEXE 1**)

➡ Ajouter la librairie LCD.lib à votre projet (C:\MCC18\lib\LCD.lib) afin de tester les fonctions déjà existantes.

➡ Modifier le fichier portLCDUser.h et configurer le port B

➡ Ecrire la fonction **LCD_write_Data(char character)**

➡ Ecrire la fonction **LCD_write_Register(char DR_or_IR);**

➡ Ecrire la fonction **LCD_Char_User(char character);** puis l'appeler dans le main() tout en utilisant la fonction **LCD_Init()** déjà existante.



A vous de jouer maintenant !

2. Travail pratique (ex3: partie 2) : développer la librairie

Dans cette seconde partie de l'exercice 3, vous allez devoir développer le reste de la librairie. Pour ne pas vous embêter, utilisez l'API précédemment écrite et vous constaterez que votre travail s'en trouvera fortement simplifié. Il ne s'agit plus que d'un problème de "C". En ce qui concerne l'écriture des APIs suivantes, nous vous conseillons cependant de respecter l'ordre donné ci-dessous :

- ➔ Ecrire la fonction **LCD_String_User(char *StringData);**. Cette fonction écrit une chaîne de caractères sur l'afficheur à la position courante du curseur.
- ➔ Ecrire la fonction **LCD_Command_User(char commande);**. Cette fonction envoie une commande au contrôleur. Les différentes commandes sont présentes dans le fichier **LCDUser.h**.
- ➔ Ecrire la fonction **LCD_Cursor_XY(unsigned char row, unsigned char col);**. Cette fonction est locale à votre librairie **LCDUser.c** et permet de positionner le curseur. Nous vous conseillons d'utiliser l'API **LCD_Command_User(char commande);** pour arriver à vos fins !
- ➔ Ecrire la fonction **LCD_Char_XY_User(unsigned char row, unsigned char col, char character)**. Cette fonction permet d'afficher un caractère à la position spécifiée en paramètres d'entrée (row et col).
- ➔ Ecrire la fonction **LCD_String_XY_User(unsigned char row, unsigned char col, char *StringData);**. Cette fonction permet d'afficher une chaîne de caractères à la position spécifiée en paramètres d'entrée (row et col).



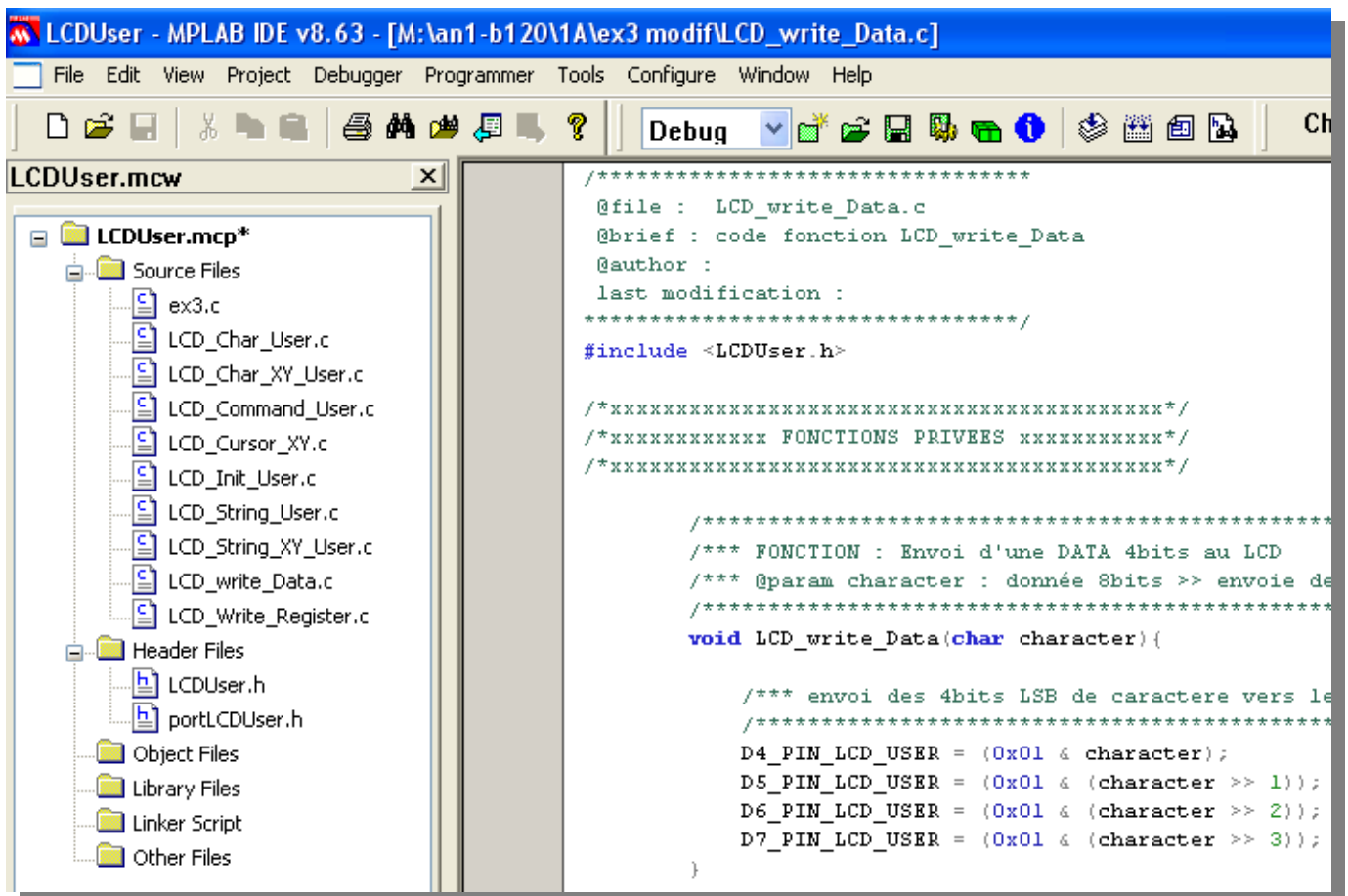
*En vous aidant de la datasheet du
contrôleur LCD écrivez la fonction **LCD_InitUser()**
assurant l'initialisation de celui-ci !*

- ➔ Ecrire la fonction **LCD_Init_User(void);**. Cette fonction permet d'initialiser le contrôleur. Il vous faudra aller voir le protocole d'initialisation du mode 4bits dans la datasheet du contrôleur (cf. **ANNEXE ?**).

3. Travail pratique (ex3: partie 3) : générer une librairie

Vous allez maintenant générer votre première librairie (.lib). La démarche est un peu fastidieuse mais relativement simple, vous n'avez qu'à suivre les instructions données ci-dessous :

- Recréez un nouveau projet et nommez le **LCDUser.mcp** en incluant les sources développés précédemment. Le nom du projet fixera le nom de la future librairie. Recompilez et assurez-vous du bon fonctionnement du programme.
- Créez maintenant un fichier source par fonction en donnant à chaque fichier le même nom que la fonction. Exemple du fichier **LCD_write_Data.c** :



The screenshot shows the MPLAB IDE interface. On the left, the project tree for 'LCDUser.mcp' is visible, showing a 'Source Files' folder containing various .c files including 'LCD_write_Data.c'. The main editor window displays the source code for 'LCD_write_Data.c'.

```

/*****
@file : LCD_write_Data.c
@brief : code fonction LCD_write_Data
@author :
last modification :
*****/
#include <LCDUser.h>

/*****
***** FONCTIONS PRIVEES *****/
*****/

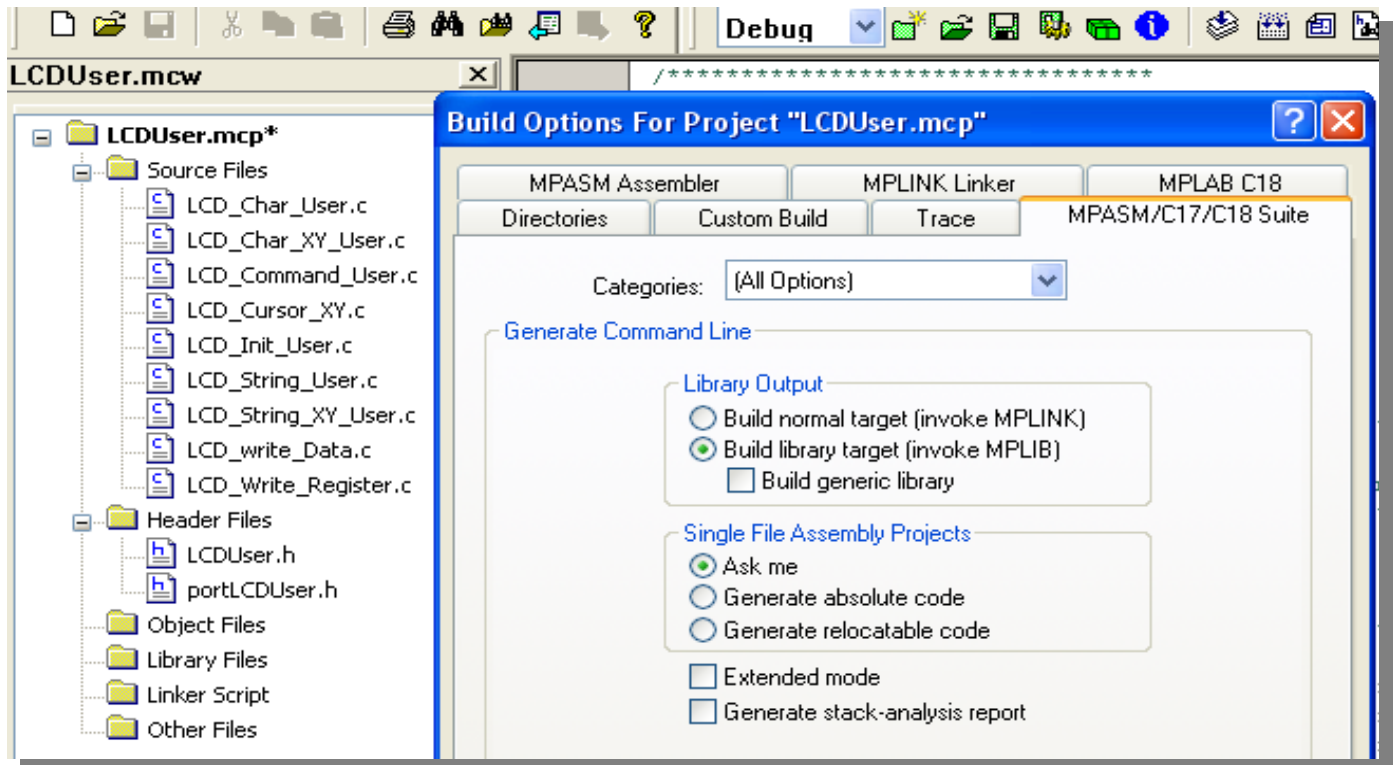
/*****
*** FONCTION : Envoi d'une DATA 4bits au LCD
*** @param character : donnée 8bits >> envoie de
*****/
void LCD_write_Data(char character){

    /*** envoi des 4bits LSB de caractere vers le
    *****/
    D4_PIN_LCD_USER = (0x01 & character);
    D5_PIN_LCD_USER = (0x01 & (character >> 1));
    D6_PIN_LCD_USER = (0x01 & (character >> 2));
    D7_PIN_LCD_USER = (0x01 & (character >> 3));
}

```

- Enlevez ensuite LCDUser.c de votre projet, recompilez ... votre programme doit toujours fonctionner !

- Enlevez maintenant le fichier ex3.c puis sélectionnez le générateur de bibliothèques MPLIB à la place du linker MPLINK (Build Options) :



- Compilez votre programme. Vous constaterez dans la fenêtre de build qu'à la fin de la compilation MPLAB n'appelle plus MPLINK mais MPLIB.



Vous venez de générer votre première librairie. Il reste à la tester !

- Sélectionnez à nouveau le linker dans les Build Options, enlevez tous les sources de votre projet sauf ex3.c. Ajouter votre librairie (LCDUser.lib) au projet puis recompilez votre programme.

