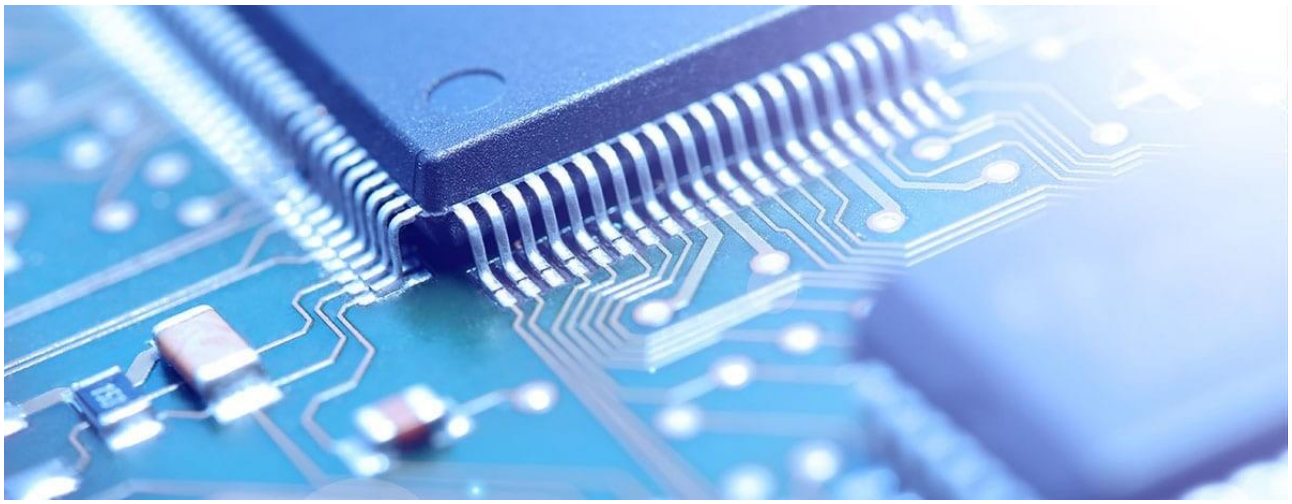
CAMPUS
D'ENSEIGNEMENT SUPÉRIEUR
ET DE FORMATION PROFESSIONNELLE

TP MICROCONTROLEUR

TP2 – Communication sur un écran LCD et gestion de la mémoire EEPROM



Par Damien DUBOIS & Marion ESCOUTELOUP

CESI S3E – P1G1 – FIPA26

29 juin 2022

TABLE DES MATIERES

TABLE DES FIGURES

I. INTRODUCTION

L'objectif de ce TP2 est de réussir à communiquer avec la mallette d'essai mis à notre disposition. Lors de ce TP, nous nous sommes attardés sur le développement des éléments nécessaire pour le contrôle du clavier ainsi que l'affichage sur l'écran LCD. Dans une seconde partie, nous attarderons sur la mémorisation d'éléments dans une mémoire EEPROM.

Pour réaliser au mieux ce TP, nous nous configurons sur un microcontrôleur de type PIC18F87K22.

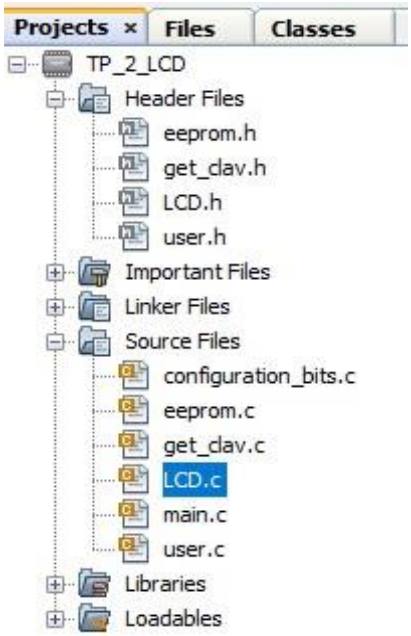


Figure 1 : Capture d'écran -- Architecture du projet

Pour rappel, nous connaissons les paramètres suivant :

- far unsigned char CLAVIER @ 0x180000 ;
- far unsigned char LCD_DATA @ 0x1A0002 ;
- far unsigned char LCD_FUNC @ 0x1A0000 ;

II. Pilotage du clavier et de l'écran LCD

1. Préparation : récupération des adresses

Afin de pouvoir communiquer avec les éléments, il nous est important d'être capable de retrouver leurs adresses. Pour ce faire, dans notre cas, nous pouvons le retrouver grâce à deux documents :

- Le schéma électrique de la mallette
- Datasheet constructeur du multiplexeur

Nous avons ainsi les informations suivantes :

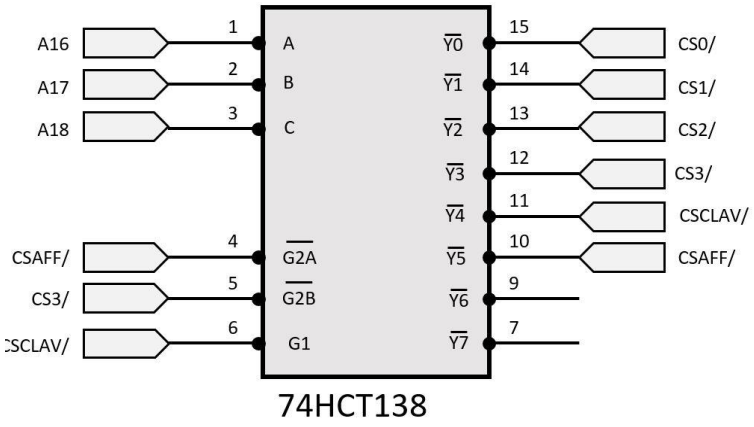


Figure 2 : Schéma du branchement du multiplexeur 74HCT138

Table 2. Pin description

Symbol	Pin	Description
A0, A1, A2	1, 2, 3	address input A0, A1, A2
$\overline{E}1$, $\overline{E}2$	4, 5	enable input $\overline{E}1$, $\overline{E}2$ (active LOW)
E3	6	enable input E3 (active HIGH)
$\overline{Y}0$, $\overline{Y}1$, $\overline{Y}2$, $\overline{Y}3$, $\overline{Y}4$, $\overline{Y}5$, $\overline{Y}6$, $\overline{Y}7$	15, 14, 13, 12, 11, 10, 9, 7	output $\overline{Y}0$, $\overline{Y}1$, $\overline{Y}2$, $\overline{Y}3$, $\overline{Y}4$, $\overline{Y}5$, $\overline{Y}6$, $\overline{Y}7$ (active LOW)
GND	8	ground (0 V)
V_{CC}	16	positive supply voltage

Table 3. Function table[1]

Control			Input			Output							
$\overline{E}1$	$\overline{E}2$	E3	A2	A1	A0	$\overline{Y}7$	$\overline{Y}6$	$\overline{Y}5$	$\overline{Y}4$	$\overline{Y}3$	$\overline{Y}2$	$\overline{Y}1$	$\overline{Y}0$
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X											
X	X	L											
L	L	H	L	L	L	H	H	H	H	H	H	H	L
			L	L	H	H	H	H	H	H	H	L	H
			L	H	L	H	H	H	H	H	L	H	H
			L	H	H	H	H	H	H	L	H	H	H
			H	L	L	H	H	H	L	H	H	H	H
			H	L	H	H	H	L	H	H	H	H	H
			H	H	L	H	H	H	H	H	H	H	H
			H	H	H	L	H	H	H	H	H	H	H

Figure 3 : Tableau description de la datasheet du multiplexeur 74HCT138

Nous pouvons donc, grâce à ces données, retrouver les adressages suivants :

Figure 4 : Décryptage des trames pour adressage au LCD

2. Ecriture sur l'écran LCD

Maintenant que nous avons l'ensemble des informations, nous pouvons lancer le développement pour pouvoir afficher sur l'écran LCD ce que l'on écrit via le clavier. Pour cela, il ne faut pas oublier d'utiliser les conversions de caractères ASCII. Sans cela, l'affichage ne sera pas juste. Cette conversion sera intégrée directement dans les fonctions propres à l'utilisation du clavier.

Pour pouvoir utiliser correctement l'écran, nous avons besoin de développer deux fonctions. La première, appelé *init_aff_lcd()* permet d'initialiser l'écran. Il lance toutes les interactions possibles avec cet écran afin de vérifier que l'ensemble fonctionnement bien correctement avant l'application de notre besoin.

Dans un second temps, nous avons programmé la fonction *AffCarac().* Cette fonction, elle, permet de venir directement afficher sur l'écran LCD un ou plusieurs caractères grâce un paramètre d'entrée dans la fonction.

3. Utilisation du clavier pour écrire sur l'écran LCD

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

Figure 5 : Table de conversion ASCII

Notre clavier est un clavier numérique, c'est-à-dire qu'il écrit seulement des chiffres allant de 0 à 9. Ainsi, d'après le tableau ASCII ci-dessus, pour afficher ces chiffres sur notre écran LCD nous avons besoin des lignes 0x30 à 0x39.

Utilisation du clavier nécessite l'utilisation d'interruption.

III. Développement mémoire EEPROM

1. Principe et utilisation d'une mémoire EEPROM

Une mémoire EEPROM est une mémoire dite non-volatile (aucune donnée supprimés lors de la mise hors tension) permettant de stocker des données. Le contenu de cette mémoire peut-être directement changer et modifier de manière quasiment illimité (en réalité modifiable approximativement 1millions de fois au maximum) grâce à un programme. Cette mémoire, interne au microcontrôleur ou externe de part une puce supplémentaire sur un PCB, a plusieurs applications :

- Mémorisation de réglages et configurations pour un système
- Mémorisation de données d'étalonnage
- Mémorisation de mesures sur une longue durée (Data Loger)

Il existe de nombreuses autres applications.

Dans notre cas, nous utilisons la mémoire EEPROM directement présente dans notre microcontrôleur PIC18K87K22. Cette mémoire peut contenir jusqu'à maximum 1024 octets (10 bits).

2. Mémorisation en EEPROM

Write : 05=20 ENT *écrire 20 en hexadécimal à l'emplacement 05 de l'EEROM.*

0	6	=	2	0	ENT										
2	0														

Après une écriture on effectue une lecture de vérification

Read : 06 ENT Lire le contenu de l'adresse 06 (résultat 01)

0	6	ENT													
0	1														

Les deux premiers caractère définissent la case mémoire où écrire et les deux caractères après le « = » définissent la données à sauvegarder dans l'EEPROM.

Nous n'avons pas pu valider cette fonctions de sauvegarde. Après avec fait le code la fonction qui lit la mémoire ne nous renvoi pas la donnée envoyé auparavant dans la même case mémoire. Problème de code ou de mallette, en effet la mallette TP dont nous disposions était quelque peu défaillante.

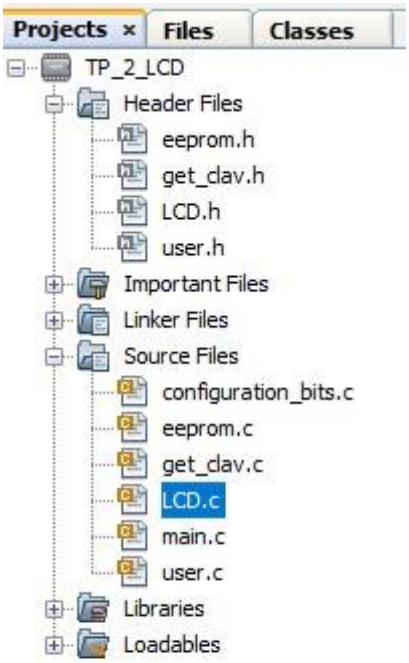
IV. CONCLUSION

Ce projet de TP microcontrôleur nous a permis de développer un programme en langage C pour la gestion d'un clavier numérique et d'un affichage sur écran LCD. Indirectement, cela nous a permis d'introduire plusieurs éléments de développement :

- Création de librairies de fonctions (files .c et header .h)
- Gestion des interruptions d'un microcontrôleur
- Gestion d'une conversion ASCII
- Gestion d'une mémoire EEPROM

Malheureusement, nous n'avons tout de même pas réussi à obtenir un résultat satisfaisant sur le développement mémoire de l'EEPROM. Cette partie ne fonctionne pas encore sur la mallette d'essai mise à notre disposition. Plusieurs approfondissements sur ce point sont à prévoir de manière personnels.

V. ANNEXE 1 : Programme final



1. Main.c : fichier source du programme globale

```
void main(void) {
////
    unsigned char i;
    unsigned char addr ;
    unsigned char data ;
    init();
    init_aff_lcd(); // init lcd
    init_interrupt();
    valeur_dispo = 0;
    j=0;
    for(i=0;i>5;i++){
        Tab[i]=0;
    }
    delai_ms(500);
    LCD_Clear ();

    while(1){
        if(valeur_dispo == 1){
            decod_clav();
            if(((valeur_to_affich >= '0') && (valeur_to_affich <= '9')) || (valeur_to_affich == '='))
                LCD_DATA = valeur_to_affich;

            if(valeur_to_affich == 1)
              LCD_FUNC = SHIFT_CURSOR_LEFT;

            if(valeur_to_affich == 2)
              LCD_FUNC = SHIFT_CURSOR_RIGHT;

            if(valeur_to_affich == 3){
              LCD_FUNC = DISPLAY_CLEAR;
              while((LCD_FUNC & 0x80)==0x80);
              LCD_FUNC = RETURN_HOME;
```

```
                j=0;
            }
            if(valeur_to_affich == 4){
                if(j==6){
                    j=0;
                    addr= Tab[0]*10+Tab[1];
                    data = (Tab[3]<<4) & Tab[4];
                    write_eeprom(addr, data);
                    Tab[3]=decod_tab(Tab[3]);
                    LCD_DATA = Tab[3];
                    while((LCD_FUNC & 0x80)==0x80);
                    Tab[4]=decod_tab(Tab[4]);
                    LCD_DATA = Tab[4];
                }
                if(j==3){
                    j=0;
                    addr= Tab[0]*10+Tab[1];
                    data = read_eeprom(addr);
                    LCD_DATA = data /10;
                    while((LCD_FUNC & 0x80)==0x80);
                    LCD_DATA = data %10;
                }
            }
            while((LCD_FUNC & 0x80)==0x80);
            valeur_dispo = 0;
        }
    }
    return;
}
```

2. LCD.c : fichier regroupant les fonctions propres à l'écran LCD

```
1.  void init_aff_lcd(void){
2.     delai_ms(100);
3.     LCD_FUNC=0x38; delai_ms(5);
4.     LCD_FUNC=0x38; delai_ms(1);
5.     LCD_FUNC=0x38; delai_ms(1);
6.     LCD_FUNC=SET_FUNC_8BIT_2LINE_5x7; delai_ms(1);
7.     LCD_FUNC=SHIFT_CURSOR_RIGHT; delai_ms(1);
8.     LCD_FUNC=DISPLAY_ON_CUR_ON_BLINKOFF; delai_ms(1);
9.     LCD_FUNC=ENTRY_MODE_SET_CI_DNS; delai_ms(1);
10.    LCD_FUNC=RETURN_HOME; delai_ms(2);
11.    LCD_FUNC=DISPLAY_CLEAR; delai_ms(2);
12. }
13.
```

3. Clavier.c : fonctions propres à l'utilisation d'un clavier numérique

```
1.  extern unsigned char valeur_clav;
2.  extern unsigned char valeur_dispo;
3.  extern unsigned char valeur_to_affich;
4.  extern unsigned char j;
5.  extern unsigned char Tab[6];
6.
7.  // autorisation interruption
8.  void init_interrupt(void)
9.  {
10.     INTCONbits.GIE = 1;
11.     INTCONbits.INT0IE = 1;
12.     INTCONbits.INT0IF = 0;
13. }
14.
15. // routine d'interruption
16. void high_priority interrupt Get_clav(void)
17. {
18.     if(INT0IF)
19.     {
20.         // valeur de la matrice du clavier
21.         valeur_clav = CLAVIER;
22.         Tab[j] = valeur_clav;
23.         valeur_dispo = 1;
24.
25.          j++;
26.          // remise a zero du flag
27.         INTCONbits.INT0IF = 0;
28.     }
29.
30. }
31.
32.
33. void decod_clav(void)
34. {
35.     // selon la valeur de la matrice clavier
36.     // valeur à afficher en ascii
37.     // non prise en compte du retour arriere ou avancer
38.     valeur_clav = (valeur_clav & 0x0F);
39.    switch(valeur_clav){
40.       case 0 : valeur_to_affich = '0'; break;
41.       case 1 : valeur_to_affich = '1'; break;
42.       case 2 : valeur_to_affich = '2'; break;
43.       case 3 : valeur_to_affich = '3'; break;
44.       case 4 : valeur_to_affich = '4'; break;
45.       case 5 : valeur_to_affich = '5'; break;
46.       case 6 : valeur_to_affich = '6'; break;
47.       case 7 : valeur_to_affich = '7'; break;
48.       case 8 : valeur_to_affich = '8'; break;
49.       case 9 : valeur_to_affich = '9'; break;
50.       case 11: valeur_to_affich = '='; break;
51.       case 12: valeur_to_affich = 1; break;
52.       case 13: valeur_to_affich = 2; break;
53.       case 14: valeur_to_affich = 3; break;
54.       case 15 : valeur_to_affich = 4; break;
55.
56.       default : valeur_to_affich = 'x'  ;
57.    }
58. }
59.
60.
61.
62.
63.
64.
65.
```

```
66. unsigned char decod_tab(unsigned char valeur)
67. {
68.     unsigned char tab;
69.     // selon la valeur de la matrice clavier
70.     // valeur à afficher en ascii
71.     // non prise en compte du retour arriere ou avancer
72.     valeur = (valeur & 0x0F);
73.    switch(valeur)
74.    {
75.    case 0 : tab = '0'; break;
76.    case 1 : tab = '1'; break;
77.    case 2 : tab = '2'; break;
78.    case 3 : tab = '3'; break;
79.    case 4 : tab = '4'; break;
80.    case 5 : tab = '5'; break;
81.    case 6 : tab = '6'; break;
82.    case 7 : tab = '7'; break;
83.    case 8 : tab = '8'; break;
84.    case 9 : tab = '9'; break;
85.
86.    default : valeur_to_affich = 'x'  ;
87.    }
88.    return tab;
89. }
90.
```

4. Eeprom.c : fonctions pour la mémorisation d'élément dans la mémoire EEPROM

```
1.  unsigned char read_eeprom(unsigned char address){
2.      int data_rd;
3.
4.      EEADRH = address >> 8;
5.      EEADR = address; /*si un erreur, ici voir EEADRH*/
6.
7.      EECON1bits.EEPGD = 0; /*Access data EEPROM memory*/
8.      EECON1bits.CFGS = 0; /*= Access Flash program or data EEPROM memory*/
9.      EECON1bits.RD =1; /*Initiates an EEPROM read*/
10.
11.     data_rd = EEDATA; /*read data*/
12.
13.     return data_rd;
14. }
15.
```

```
1.  void write_eeprom(unsigned char address, unsigned char data_wr){
2.      EEADRH = address >> 8;
3.      EEADR = address; /*si un erreur, ici voir EEADRH*/
4.
5.      EECON1bits.EEPGD = 0; /*Access data EEPROM memory*/
6.      EECON1bits.CFGS = 0; /*= Access Flash program or data EEPROM memory*/
7.
8.      EEDATA = data_wr; /*ecriture des data*/
9.
10.     EECON1bits.WREN = 1; /*Allows write cycles to Flash program/data EEPROM*/
11.
12.     INTCONbits.GIE = 0; /*Disable interrupt*/
13.
14.     EECON2 =0x55; //write sequence unlock
15.     EECON2 = 0xAA; //write sequence unlock
16.
17.     EECON1bits.WR = 1; //initiates a data EEPROM erase/write cycle
18.
19.     while(EECON1bits.WR); //waits for write cycle to complete
20.
21.     GIE = 1; //restore interrupts
22.     EECON1bits.WREN = 0; //disable write
23.  }
```