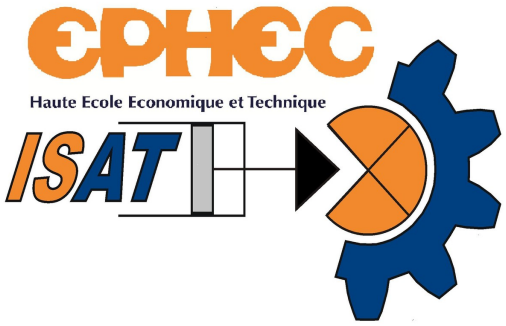


UE A211 : Systèmes Embarqués I

Professeur : COSTA Emile

RAPPORT DE LABORATOIRE

Etudiants : FINYA Alban
KORKUT Caner

TP2 - Les entrées numériques et analogiques

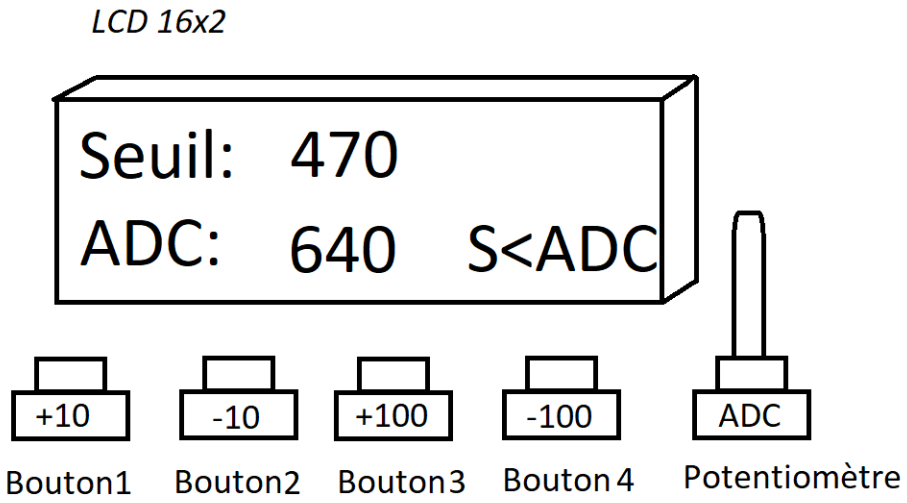
Table des matières

1 Objectif du TP

Dans cette deuxième séance de TP nous avons introduit trois nouveaux concepts à savoir :

— Les entrées numériques ;
— Les entrées analogiques ;
— Utilisation du LCD ;

Pour ce faire, il nous a été demandé de réaliser un mini-projet consistant à incrémenter/décrémenter deux valeurs et les comparer entre elles. La première valeur appelée *seuil* sera modifiée à l'aide de 4 boutons ($+10$, -10, $+100$, -100) tandis que la valeur ADC sera liée au potentiomètre. Enfin, ces valeurs et le résultat de la comparaison devront être affichés sur un écran LCD de la manière suivante :



Croquis du travail à réaliser

2 Introduction

Afin de réaliser le travail demandé, nous avons dû prendre connaissance des bibliothèques associées. Il nous a semblé plus pertinent de réaliser le schéma de principe et de passer ensuite à la programmation. Pour y parvenir, la démarche que nous avons suivie fut de découper le travail en sous-problèmes afin de faciliter la réalisation. Nous avons donc testé indépendamment les une des autres les

bibliothèques *ADC*, *LCD* et *Button*. Une fois ces outils maîtrisés, il nous a suffi d'assembler nos différents programmes et de les adapter à notre sauce.

3 Schéma de principe

Le schéma de principe ainsi que la simulation ont été réalisés sur Proteus. Les composants utilisés sont les suivants :

- PIC18F45K22 ;
- 4 Résistances 330Ω ;
- 1 Potentiomètre 1kΩ ;
- 1 LCD 16x2 ;
- 4 Boutons poussoir ;
- 6 Alimentations 5V.
- 1 Masse.

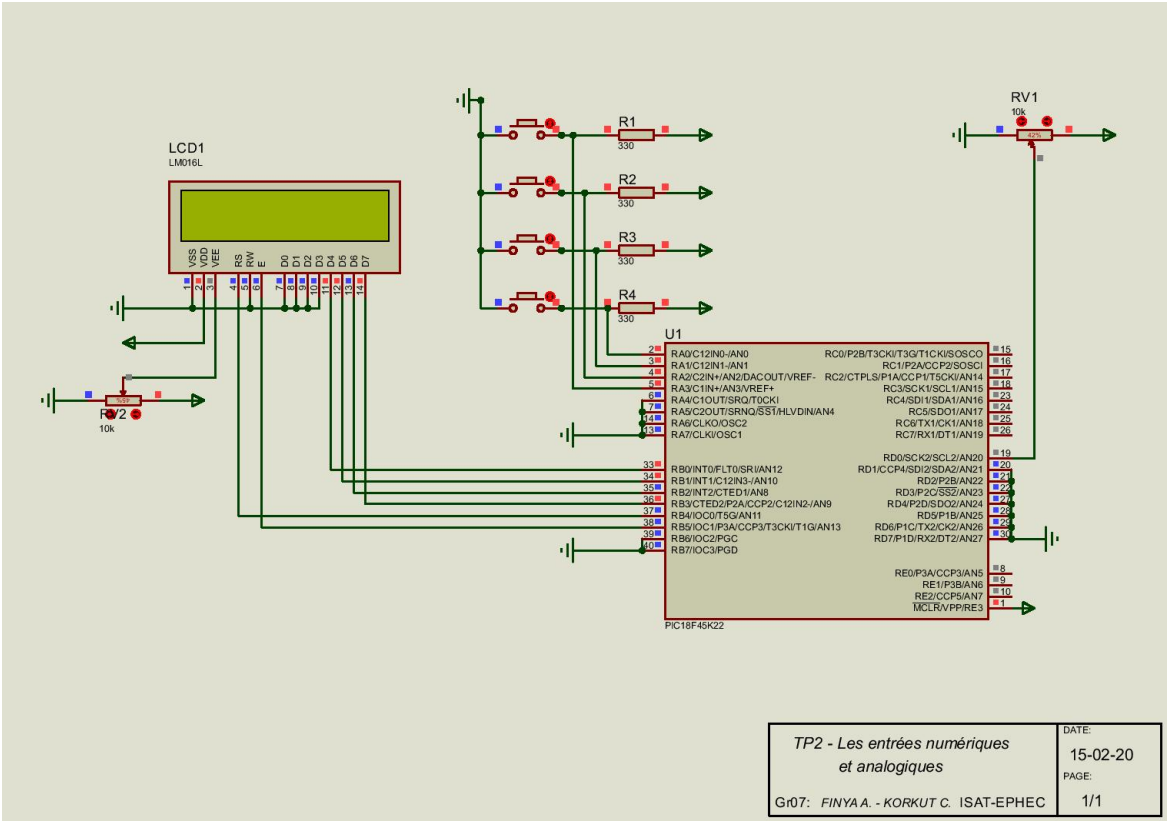
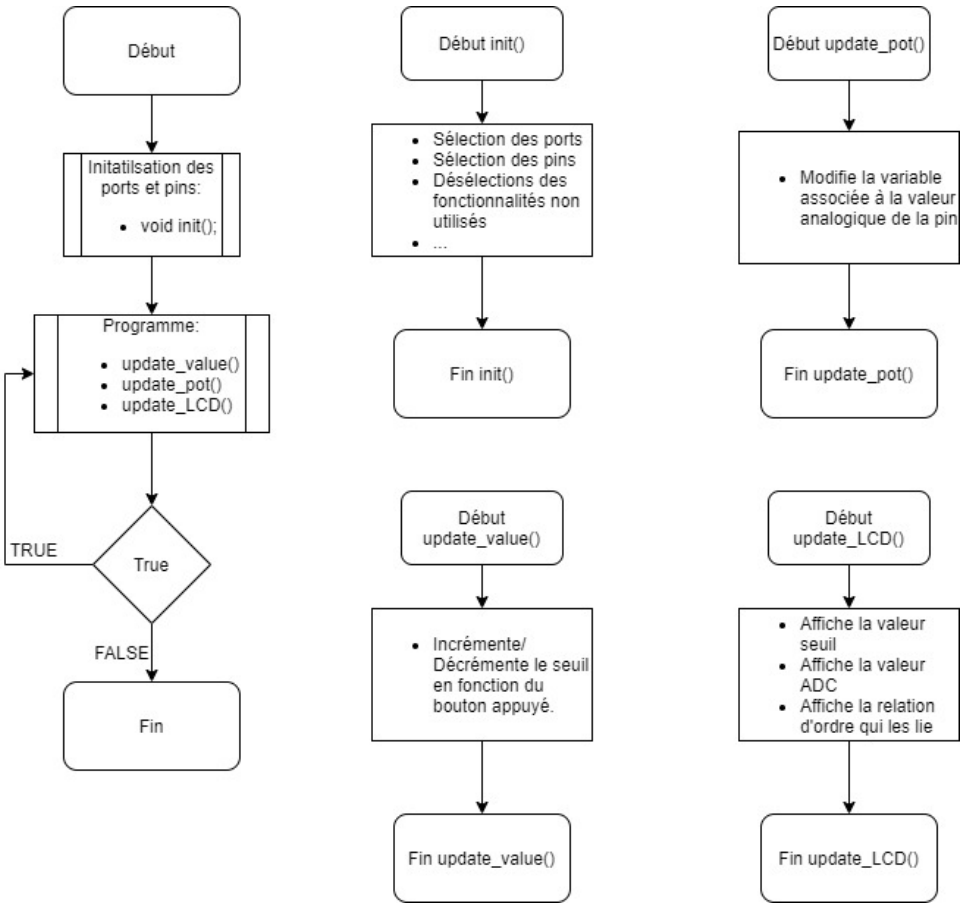


Schéma de principe sur Proteus

Il est à noter que les pins flottantes ont été reliées à la masse afin que les mesures ne soient pas biaisées par des perturbations externes lors de la lecture des pins. Par rapport à notre dernière utilisation du logiciel, nous avons également apporté quelques changements à notre présentation. En effet nous y avons rajouté un cartouche et l'avons exporté au format `.PDF`.

4 Algorigramme

L'agorigramme du programme a été réalisé via la plateforme Draw.io. L'avantage de cette plateforme est qu'il est possible d'exporter son travail au format .draw.io et de l'éditer ensuite par après. Nous n'avions plus qu'à reprendre notre algorigramme de la semaine dernière et l'adapter à notre travail.



Algorigramme du programme principal

5 Code source

```
// Connexions du module LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// Fin des connexions du module LCD


// Declaration des variables globales
int value;
unsigned int pot_value;
char txt_1[] = "Bonjour";
char txt_2[] = "Seuil";
char txt_3[] = "ADC";
char valeur[256] = "";
char potentio[256] = "";

// Lecture de l'etat des boutons
void update_value(){

    if (Button(&PORTA, 0, 1, 0) && value + 10 <= 1023 ) {
    value += 10;
    delay_ms(300);
    }

    if (Button(&PORTA, 1, 1, 0) && value -10 >= 0) {
    value -= 10;
    delay_ms(300);
    }

    if (Button(&PORTA, 2, 1, 0) && value +100 <= 1023) {
    value += 100;
```

```
42      delay_ms(300);
43      }
44
45      if (Button(&PORTA, 3, 1, 0) && value -100 >= 0) {
46      value -= 100;
47      delay_ms(300);
48      }
49  }
50
51  // Lecture de l'etat du potentiometre
52  void update_pot(){
53    pot_value = ADC_Read(20);
54  }
55
56  // Comparaison des valeurs value et pot_value
57  void update_sign(unsigned int a, unsigned int b){
58    if (a < b){
59      Lcd_Out(2, 12, "S<ADC");
60    }
61    else{
62      if (a > b){
63        Lcd_Out(2, 12, "S>ADC" );
64      }
65      else{
66        Lcd_Out(2, 12, "S=ADC" );
67      }
68    }
69  }
70
71  // Met a jour l'ecran du LCD
72  void update_LCD(){
73    IntToStr(value, valeur);
74    Lcd_Out(1, 7, valeur);
75    IntToStr(pot_value, potentio);
76    Lcd_Out(2,4, potentio);
77    update_sign(value, pot_value);
78  }
79
80  // Initialisation des routines uniques
81  void init(){
82    ANSELA  = 0;
83    ANSELB = 0;
84    ANSELD  = 0b00000001;
```

```
85
86    // Desactive les comparateurs
87    C1ON_bit = 0;
88    C2ON_bit = 0;
89
90    // Initialise les entrees et sorties
91    TRISA = 0b00001111;
92    TRISD = 0b00000001;
93
94    /*Initialise les objets lies
95    au bibliotheques utilisees*/
96    ADC_Init();
97    Lcd_Init();
98
99    /* Affichage des caracteres
100   "fixes" de l'ecran du LCD */
101   Lcd_Cmd(_LCD_CLEAR);
102   Lcd_Cmd(_LCD_CURSOR_OFF);
103   Lcd_Out(1,1,txt_1);
104   Delay_ms(1000);
105   Lcd_Cmd(_LCD_CLEAR);
106   Lcd_Out(1,1,txt_2);
107   Lcd_Out(2,1,txt_3);
108 }
109
110 // Fonction principale:
111 void main() {
112
113   init();
114
115   // Programme:
116   for(;;){
117     update_value();
118     update_pot();
119     update_LCD();
120   }
121 }
```

6 Analyse du code source

Etant donné que nous avons déjà analysé certains éléments lors du premier rapport, nous nous focaliserons ici sur les nouvelles bibliothèques et fonctionna-

lités utilisées et plus précisément sur la fonction void `main()`.

La premier appel de fonction fait référence à void `init()`. Cette fonction commence toujours par l'initalisation des PORTS et des PINS associés. Etant donné que dans notre cas nous avons connecté notre potentiomètre à la PIN1 du PORTD, nous faisons appel à l'instruction suivante :

```
84  ANSELD  = 0b00000001;
```

Cette commande ne fait qu'activer l'utilisation **analogique** de ladite PIN. Par souci de clareté, nous avons opté pour l'écriture binaire de la commande.

Les deux lignes suivantes désactivent la fonctionnalité *comparateur* des pins utilisées :

```
87  C1ONbit = 0;
88  C2ONbit = 0;
```

Cette commande, au même titre que la précédente, sert à la sélection de l'utilisation des ports.

On peut ensuite apercevoir les lignes suivantes :

```
96  ADCInit () ;
97  LcdInit () ;
```

Ces deux instructions instancient deux objets liés aux bibliothèques *ADC* et *LCD*. La première instruction fait la relation entre la PIN1 du PORTD et la fonction `ADCRead(20)` qui sera expliquée plus tard. La seconde instruction initialise les commandes liées à la bibliothèque LCD. Elle permet également de faire le lien entre les pins et le LCD comme suit :

```
2 sbit LCD_RS at RB4_bit;
3 sbit LCD_EN at RB5_bit;
4 sbit LCD_D4 at RB0_bit;
5 ...
```

On peut ensuite afficher nos premiers caractères sur le LCD :

```
2  Lcd_Cmd(_LCD_CLEAR);
3  Lcd_Cmd(_LCD_CURSOR_OFF);
4  Lcd_Out(1,1,txt_1);
5  ...
```

Les commandes associées à la bibliothèque peuvent aisément être trouvées sur la documentation officielle de MikroC [1].

On peut enfin analyser le contenu de notre programme :

```
117  update_value();
118  update_pot();
119  update_LCD();
```

Les noms de ces routines ont été choisis de manière à être le plus explicite possible. Nous avons estimé que ces lignes ne nécessitaient pas plus de commentaires.

Passons à l'analyse des fonctions créées :

- void update_value();
- void update_pot();
- void update_sign();
- void update_LCD().

La première fonction void update_value() a été défini comme suit :

```
28  void update_value(){
29
30    if (Button(&PORTA, 0, 1, 0) && value + 10 <= 1023 ) {
31    value += 10;
32    delay_ms(300);
33    }
34
35    if (Button(&PORTA, 1, 1, 0) && value -10 >= 0) {
36    value -= 10;
37    delay_ms(300);
38    }
39    ...
40
41 }
```

Elle incrémente/décrémente la valeur de la variable **value** grâce à la fonction *Button*. Observons son prototype afin de comprendre le fonctionnement :

```
Button(*port, pin, time, active_state);
```

1. http ://bit.ly/2OPEGxw, consulté le 15 février 2020.

La documentation disponible sur le site de MikroC [2] nous permet dès lors de traiter les cas des différents boutons. A noter qu'un délai de $300ms$ (appelé *debouncing time*) a été introduit afin de supprimer l'effet du rebond lors des appuis.

La fonction void `update_pot()` a pour but d'assigner la valeur de la pin $AN20$ à la variable `pot_value` :

```
52 void update_pot(){
53   pot_value = ADC_Read(20);
54 }
```

La raison pour laquelle la fonction `ADC_Read()` prend 20 pour argument est liée à la datasheet du PIC18F45K22 de la Figure A qui se trouve en annexe.

La prochaine fonction qui nous intéresse est void `update_sign()` :

```
57 void update_sign(unsigned int a, unsigned int b){
58   if (a < b){
59     Lcd_Out(2, 12, "S<ADC");
60   }
61   else{
62     if (a > b){
63     ...
64 }
```

Elle permet d'afficher le résultat de la comparaison entre les variables `value` et `pot_value`. Il est à souligner que nous avons remarqué qu'en programmant de la sorte, **le cas des nombres négatifs n'est pas traité** correctement. Cela ne pose pas vraiment problème dans notre cas car nous avons borné la valeur de la variable `value` entre 0 et 1023. Cependant, si un jour ne étions amenés à devoir comparer des nombres négatifs, nous passerions par la conversion en hexadécimal de ces valeurs. Dans notre cas précis, nous avons levé le bug en utilisant la fonction `IntToHex(int input, char *output)` proposée par la bibliothqèe *Conversion*.

Ce module nous permet d'introduire la fonction void `update_LCD()` dans laquelle elle a également été utilisée :

```
72 void update_LCD(){
73   IntToStr(value, valeur);
74   Lcd_Out(1, 7, valeur);
75   ...
```

2. http ://bit.ly/39swFGE, consulté le 15 février 2020

```
76 }
```

Regardons ce qui se passe de plus près : nous incrémentons une valeur qui devra ensuite être affichée dans un écran LCD. La documentation de la librarie *LCD* de MikroC nous informe que la fonction `Lcd_Out()` attend un argument de type **char**. Pour pallier à ce problème nous avons converti la valeur `value` qui est un **int** en **char**. Bien qu'en cours il nous a été proposé d'utiliser la fonction `sprinti()` pour le formatage, nous n'avons pas réussi à comprendre et à mettre en place ce mécanisme et avons pu contourner le problème en faisant appel à la fonction `IntToStr(value, valeur)`.

Nous comptons bien entendu poser nos questions au prochain cours à savoir la séance TP3 - Mise en oeuvre de la communication série (UART).

7 Conclusion

Nous retenons de cette séance l'importance de prendre connaissance du datasheet du matériel utilisé lors d'utilisations spécifiques comme l'utilisation des pins analogiques. La lecture de documentations officielles est également devenu un reflexe pour nous étant donné que nous nous avons manipulés différentes bibliothèques (*ADC, Button, LCD, Conversion*).

L'utilisation de ces outils facilite la conception du programme et raccourcit considérablement le code mais en contre-partie le fonctionnement de ce dernier est inconnu pour la personne qui programme. On entend par là qu'on se voit pas directement les interactions qui se cachent derrière. Cependant ceci n'est pas un problème pour le développeur souhaitant aller vite et optimiser son temps de travail.

Ensuite, nous avons été interpellés par le fait que la comparaison des nombres négatifs ont posé problème. Nous voyons là l'importance de tester son code et d'envisager tous les cas dans l'utilisation du programme et ne laisser aucune chance à l'utilisateur de générer des cas non envisagés. Certains de ces cas non envisagés peuvent même aller jusqu'à donner suite à des failles sécurité et permettre une utilisation frauduleuse de l'application. Anticiper et se préoccuper de ces problemes fait partie du quotidien de l'automaticien en industrie.

A ANNEXE - Datasheet PIC18F45K22

17.3 Register Definitions: ADC Control

Note: Analog pin control is determined by the ANSELx registers (see Register 10-2)

REGISTER 17-1: ADCON0: A/D CONTROL REGISTER 0

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	CHS<4:0>					GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'

bit 6-2 **CHS<4:0>: Analog Channel Select bits**

00000 = AN0
00001 = AN1
00010 = AN2
00011 = AN3
00100 = AN4
00101 = AN5(1)
00110 = AN6(1)
00111 = AN7(1)
01000 = AN8
01001 = AN9
01010 = AN10
01011 = AN11
01100 = AN12
01101 = AN13
01110 = AN14
01111 = AN15
10000 = AN16
10001 = AN17
10010 = AN18
10011 = AN19
10100 = AN20(1)
[redacted]
10110 = AN22(1)

Module Analog-Digital provenant du datasheet