PROJET DE MICRO INFORMATIQUE EN 111

Mise en œuvre d'un afficheur LCD sur carte PICDEM2+

L'objectif de ce projet est tout d'abord de revoir les concepts vus dans les TPs (configuration d'un timer. gestion d'interruptions...). Ce sera aussi l'opportunité d'aborder un autre microcontrôleur de la même famille mais plus complexe et ainsi d'apprendre à exploiter une datasheet conséquente pour en extraire l'information souhaitée. Enfin, la problématique de la structuration du code sera mise en avant, avec le développement d'une librairie dédiée au contrôle de l'afficheur pour ensuite l'exploiter dans le cadre de l'application du projet.

On souhaite ici réaliser une horloge à quartz avec affichage de l'heure sur module LCD. En fonction du temps disponible, on pourra lui adjoindre des fonctions supplémentaires telles que l'affichage de la température ambiante, une fonction minuterie avec alarme, …

I) Cahier des charges

On utilisera la carte de développement PICDEM2+ fournie par Microchip. Cette carte peut fonctionner avec plusieurs types de PICs et dispose de périphériques assez nombreux dont un afficheur LCD 2×16 caractères, un buzzer (tweeter piézoélectrique), un capteur de température TC74. Elle sera équipée pour le projet avec un PIC 16F877a. On notera que ce PIC dispose d'un module de débuggage incorporé permettant de suivre l'exécution du programme depuis MPLABX ce qui facilitera grandement sa mise au point.

La carte PICDEM2+ intègre un quartz de 32,768 kHz relié aux broches OSO (RC0) et OSI (RC1) du PIC, ce qui permettra une incrémentation du TIMER1 et un débordement de celui-ci toutes les 2 secondes ($2^{16}/32768$). Ceci constituera la base de temps de notre horloge très précise ($\approx 10^{-5}$) si on laisse évoluer librement le TIMER1 sans jamais le réinitialiser (introduction d'un décalage…). Chaque débordement du TIMER1 générera une interruption toutes les 2 secondes. Afin d'obtenir une incrémentation toutes les secondes on exécutera également cette même routine d'interruption avec un déclenchement par un des 2 modules COMPARE qui fera la comparaison du TIMER1 avec la valeur intermédiaire 0x8000 correspondant à 1 seconde (on notera, comme indiqué §6.4 p.59 de la datasheet, que le TIMER 1 doit être resynchronisé avec l'horloge du processeur pour pouvoir être utilisé avec le module CAPTURE/COMPARE). L'horloge sera ainsi incrémentée et mise à jour toutes les secondes.

La LED reliée à RB1 servira de témoin de bon déroulement du programme ("heart beat") et clignotera avec une période de 2 s (commutation effectuée dans la routine d'IT décrite ci-dessus)

Le réglage de l'heure se fera au moyen du poussoir S3 (relié à RB0) et du potentiomètre (relié à RA0) situé à gauche de l'afficheur selon une procédure originale :

- Un appui prolongé d'au moins 2 s sur S3 permettra d'activer le mode "réglage" et fera clignoter les heures. Si celles-ci doivent être ajustées la rotation du potentiomètre fera évoluer le nombre entre 00 (potentiomètre en butée dans le sens anti-horaire) et 23 (potentiomètre en butée dans le sens horaire). Si le potentiomètre n'est pas manipulé après l'appui sur S3, l'heure reste inchangée, invitant ainsi à modifier les champs suivants (minutes ou secondes).
- Un nouvel appui (court) sur S3 permettra un réglage des minutes selon la même procédure, entre 00 et 59 si rotation du potentiomètre, valeur inchangée sinon.
- Idem après un autre appui sur S3 pour le réglage des secondes.
- Le mode "réglage" sera quitté automatiquement 10 s après la dernière rotation du potentiomètre ou appui sur S3.

Cette méthode de réglage "analogique" a ainsi l'avantage de permettre un réglage très rapide des différents champs de l'affichage de l'horloge comparé aux méthodes plus classiques par incrémentation à chaque appui sur un poussoir.

En dehors du mode de réglage, on pourra aussi se servir du poussoir S3 pour changer l'affichage de la fonction secondaire sur la 2^{e} ligne de l'écran (température, minuterie…).

II) Contraintes logicielles

La programmation s'effectuera en langage C en adoptant une approche modulaire :

① On commencera par valider la base de temps en faisant simplement clignoter la LED ("heart beat") à la fréquence imposée de 2 Hz (changement d'état toutes les 1 secondes).
On écrira pour cela un fichier **test_led.c**

② Pour obtenir des temporisations, notamment celles pour l'initialisation du contrôleur de l'afficheur, on pourra faire appel aux macros **__delay_us(x)** et **__delay_ms(x)** du compilateur XC8. L'argument **x** correspond à la durée voulue dans l'unité de temps choisie. Ces 2 macros génèrent en fait des boucles calculées à la compilation du code source, à partir de la fréquence du processeur qu'il faudra indiquer par le paramètre **_XTAL_FREQ**, à ne pas oublier de définir : **#define _XTAL_FREQ** *fréquence en Hz*

③ On développera une bibliothèque de fonctions dédiées à la commande de l'afficheur (fichier **lib_LCD.c**, voir prototypes dans la partie V du sujet) que l'on testera en affichant une chaîne de caractères (fichier **test_LCD.c**).

④ On affichera l'heure en temps réel en faisant appel aux fonctions de la bibliothèque en écrivant le fichier final **horloge.c** qui remplacera dans le projet le fichier **test_LCD.c**.

III) Recommandations

Tous les documents utiles à l'élaboration de ce projet se trouvent sur la page :
http://bedenes.vvv.enseirb-matmeca.fr (section **datasheets** de l'onglet MICROCHIP)

III-1) Utilisation du PIC 16F877a

Le **PIC16F877a** est l'un des plus complets de la série 16 (5 PORTS, 3 TIMERS, CAN, module de communications USART, …) mais possède le même cœur de processeur que le **PIC16F84a** utilisé dans les TPs. La difficulté supplémentaire, qui constitue un des objectifs de ce projet, sera donc d'être capable de comprendre le fonctionnement d'un périphérique interne à partir du flot d'informations important fourni dans la datasheet (désignée par DS dans la suite).

- Propriétés du projet : après avoir bien sûr choisi la bonne référence de PIC, les paramètres du projet sont identiques à ceux des TPs à part que l'option *Power target circuit from ICD3* ne doit pas être cochée puisque la carte dispose d'une alimentation séparée.

- Configuration du PIC : on définit les paramètres dans la fenêtre *Configuration Bits* comme pour le PIC16F84a, en choisissant ici les options FOSC = HS , WDTE = OFF et LVP = OFF, les autres champs étant sans importance dans notre application (cf p.144 de la DS).

III-2) Génération de la base de temps

- Après avoir configuré le registre T1CON (cf DS p.57) pour obtenir un débordement du TIMER1 toutes les 2 secondes, on pourra commencer par valider le déclenchement correct de la fonction d'interruption (IT) sur cet événement que l'on matérialisera par le changement d'état de la LED reliée à RB1. On notera que la plupart des flags tels que T1IF ne peuvent déclencher une IT que si le bit PEIE est actif (en plus de GIE=1 et T1IE=1) tel qu'indiqué DS p.153.

- On configure ensuite un des 2 modules de comparaison (CCP1 par exemple) en initialisant correctement CCP1L / CCP1H et en utilisant le mode de fonctionnement permettant le déclenchement d'une IT lors de l'égalité avec le contenu de TMR1 (CCP1M="1010" dans le registre CCP1CON, cf DS p. 64). L'autorisation de CCP1 à déclencher une IT à mi-parcours avant débordement du TIMER1 va ainsi permettre d'obtenir l'exécution de la même routine toutes les secondes (changement d'état de la LED à la fréquence de 2 Hz).

III-3) Développement de la librairie de contrôle de l'afficheur LCD

- Le module d'affichage est constitué d'un panneau LCD derrière lequel se trouve un microcontrôleur chargé de piloter chaque pixel, rendant ainsi le module plus simple à exploiter. La première étape sera donc de découvrir les commandes que peut exécuter ce microcontrôleur, le protocole de communication à respecter avec le PIC ainsi que la phase d'initialisation nécessaire. On pourra consulter pour cela la DS de l'*afficheur LCD 162F* et surtout le document explicatif plus général et plus simple à aborder sur le fonctionnement des afficheurs (document *Commande Afficheurs LCD*).

- L'afficheur dialogue via un bus de données (de 4 ou 8 bits) et le contrôle se fait par 3 signaux RS, RW et E. En observant le schéma de la carte PICDEM2+ (cf p.19 de la DS dédiée) on identifie facilement les interconnexions avec le PIC et la taille du bus de données qui est ici de 4 bits. On peut remarquer que l'alimentation de l'afficheur n'est pas permanente et qu'elle est également commandée par le PIC. On constate aussi que tous les signaux sont reliés à un même port ce qui signifie qu'il y aura des précautions à prendre pour y accéder : utilisation de la structure définie sur le port dans le fichier PIC16F877a.h ou plus universellement par opération booléenne sur le registre PORTD (OU = forçage à 1 / ET = forçage à 0) à l'aide d'un masque.

  Attention : un gros danger à éviter est de commuter involontairement l'alimentation via RD7 (= réinitialisation de l'afficheur !) ou bien de générer un front actif non désiré sur l'horloge E (= acquisition d'un quartet de données qui, en plus de donner un code erroné, décalerait d'un demi-mot toutes les acquisitions suivantes !)

- On notera que les 3 signaux de contrôle sont toujours générés par le PIC qui est le maître (donc ils seront toujours configurés en sortie) alors que les 4 bits de données sont bi-directionnels en fonction du sens du transfert (mode écriture pour RW=0 / lecture pour RW=1). Sachant que nous n'utiliserons pas les 2 commandes de l'afficheur qui utilisent le mode lecture, nous pouvons donc nous permettre de configurer de façon permanente le bus de données 4 bits en sortie sur le PIC.

- On veillera à respecter les chronogrammes d'écriture (cf p.7 et 10 du document *Commande Afficheurs LCD*) en notant que les transferts se font en un cycle de 2 paquets de 4 bits (poids fort suivi du poids faible) en réactivant seulement l'horloge E entre les 2 paquets. Celle-ci devra toujours revenir à l'état de repos (E=0) entre chaque paquet afin d'éviter de générer tout front parasite indésiré. On peut aussi remarquer qu'il n'est pas nécessaire de mettre en place des temporisations pour tenir compte des retards vu l'ensemble des contraintes temporelles (toujours < 1 µs). Il est par contre primordial de bien respecter la chronologie sur la transition des signaux de contrôle comme illustré sur les chronogrammes :

  ① positionnement de RS et RW depuis l'état de repos E =0

  ② passage de E à 1 (front montant) pour permettre l'acquisition de RS et RW par l'afficheur

  ③ positionnement des 4 bits de données sur le bus

  ④ passage de E à 0 (front descendant qui permet à l'afficheur d'échantillonner la donnée en mode écriture ou lui signifier qu'elle a été mémorisée en mode lecture)

- La procédure d'initialisation se faisant au départ sur 4 cycles de 1 paquet (seuls les 4 bits de poids fort étant significatifs), on écrira cette routine spécifique en contrôlant directement tous les signaux et en respectant toutes les temporisations indiquées, alors que les écritures (commandes) suivantes pourront se faire en utilisant la fonction `lcd_write_instr`.

  On peut noter que ces 4 cycles correspondent au début (1er quartet) de la commande FUNCTION SET qui fixe la taille du bus de données (bit DL) sur le 4^{e} cycle, la commande complète (avec N=1 pour 2 lignes et F=0 pour une matrice de caractère de 5x7 pixels) étant envoyé dans le cycle suivant en respectant désormais le format choisi. Arrivent ensuite les commandes DISPLAY OFF, DISPLAY CLEAR, et ENTRY MODE SET (choisir "incrémentation" "sans décalage de l'affichage"). On n'oubliera pas de réactiver l'affichage par la commande DISPLAY ON avec mode "clignotement du curseur" ce qui permettra de valider le bon déroulement de l'initialisation si le curseur apparaît effectivement sur l'écran après l'exécution de cette commande.

- Après l'écriture des fonctions `lcd_init` et `lcd_write_instr`, on développera ensuite dans l'ordre logique :
  ① `lcd_putch` (écriture d'un caractère à droite du précédent)
  ② `lcd_puts` (écriture d'une chaîne de caractères par appels successifs à `lcd_putch`)
  ③ `lcd_home` (curseur en haut à gauche) et `lcd_clear` (idem + effacement écran) en exploitant `lcd_write_instr`
  ④ `lcd_pos` (positionnement du curseur en se référant à la correspondance avec les adresses de la DDRAM indiquées p.7 de la DS LCD_162F)

III-4) Affichage de l'heure

- Etant donné la faible charge du processeur on pourra directement effectuer le formatage heures / minutes /secondes dans la routine d'interruption.

- L'affichage se fera dans le programme principale en utilisant un tableau ou un buffer (grâce par exemple à la fonction *sprintf* de la librairie *stdio.h*).

III-5) Réglage de l'heure

- un compteur de temps (variable incrémentée chaque seconde dans la routine d'IT) permettra de mesurer les durées nécessaires pour entrer et sortir du mode "réglage". Les rebonds du poussoir seront neutralisés par une temporisation (100 ms par exemple) à chaque passage dans la boucle principale du main ().

- Chaque champ pourra être ajusté grâce au potentiomètre qui permet de faire varier de 0 à VDD la tension sur la broche RA0 interfacée avec l'entrée du convertisseur analogique numérique du PIC. La résolution du CAN étant de 10 bits, on pourra considérer que le potentiomètre est en rotation si la valeur numérisée évolue de plus de 10 unités (≈ 1/100 e de tour du potentiomètre) de manière à s'affranchir du bruit qui pourrait induire une détection erratique.

III-6) Utilisation du CAN (cf. DS p.127)

- Il faudra configurer le CAN dans la routine d'initialisation pour le relier à RA0 avec une pleine échelle à VDD. On pourra laisser les autres entrées analogiques potentielles (AN1 … AN7) en mode Digital (**PCFG** = "1110") pour laisser le contrôle possible des broches correspondantes (cf DS Fig 11.1) par les autres périphériques du PIC.

- Le CAN étant de type "à approximations successives", chacun des bits est donc calculé après l'obtention du précédent en commençant par le MSB et avec un temps d'établissement **T_{AD} qui ne doit pas être inférieur à 1,6 µs** (cf DS section 11-2 p.131) sous peine d'une aberration du résultat de la conversion !!...attention donc à bien configurer le champ **ADCS**.

- On relancera une conversion (procédure détaillée DS p.129) à chaque passage dans la boucle principale de la fonction *main()* juste après avoir terminé la précédente. Avant d'exploiter le résultat de la conversion dans le mode réglage de l'heure et valider le bon fonctionnement du CAN on pourra commencer par afficher sa valeur sur la 2e ligne de l'afficheur (en format decimal de 0 à 1023).

ATTENTION : Manipuler le potentiomètre avec délicatesse car il est très fragile !!

IV) Fonctions supplémentaires (optionnel…)

Parmi les différentes possibilités qu'offre la carte PICDEM2+, une des plus intéressantes est l'exploitation du capteur de température TC74 qui communique avec le PIC via un bus I^2C. Si le temps le permet (prévoir au moins une séance) on cherchera à afficher sur la 2^e ligne du module LCD la température ambiante mesurée régulièrement par le capteur TC74 (au moins 4 fois par seconde).
S'il vous reste moins d'une séance, on s'orientera plutôt vers une fonction plus simple à mettre en œuvre comme un minuteur avec activation du buzzer au bout du temps écoulé.

Recommandations sur la mise en œuvre du capteur TC74

Le TC74 convertit la température sur 8 bits (binaire signé) avec une résolution de 1 °C (voir DS dédiée) et la stocke dans un registre interne **TEMP**. Son contenu est transmis sur le bus à chaque accès en lecture, une fois que la commande READTEMP a été envoyée.

La principale difficulté sera la configuration et l'exploitation en mode I^2C du module MSSP (Master Synchronous Serial Port) du PIC. On pourra au préalable consulter la description du protocole I^2C (voir document *bus I2C* dans les ressources de l'onglet MICROCHIP).

Pour la programmation on commencera par configurer le module MSSP en mode I^2C dans la fonction *init ()*. Celle-ci appellera ensuite une fonction *init_TC74 ()* qui sera chargée d'envoyer la commande RTR (Read Temperature). Un accès en lecture sera fait à chaque passage dans la boucle principale du *main ()*, on pourra créer pour cela une fonction dédiée *lecture_temp ()* qui retournera le contenu du buffer **SSPBUF** image du registre **TEMP** envoyé par le capteur.

V) Fonctions de la librairie `lib_LCD`

On placera dans le fichier **lib_LCD.h** tous les prototypes des fonctions définies dans le fichier **lib_LCD.c** ainsi que tous les alias permettant de faciliter la lecture du code.

Fichier `lib_LCD.h`

```
#define     RS    PORTDbits.RD4                // entrée RS reliée à RD4
…
…       (+ tous les alias utiles pour designer les broches, constantes,…)

void lcd_init (void);
void lcd_write_instr (unsigned char c);
void lcd_putch (unsigned char c);
void lcd_puts (const unsigned char *s);
void lcd_pos (unsigned char ligne , unsigned char pos);  // origine = (0,0)
void lcd_clear (void);
void lcd_home (void);
```