Michel GALLIEN **Groupe : 1 – Poste : 1** Séance : 10/03/2014
Binôme : Olivier MORET-BAILLY

CR-TP-1 Découverte & GPIO

Sujet :

Mise en œuvre d'un environnement de développement en langage C pour un microcontrôleur Microchip.
Découverte de la structure du microcontrôleur PIC 18F8722 et de la carte d'essai Explorer18.

Description du travail :

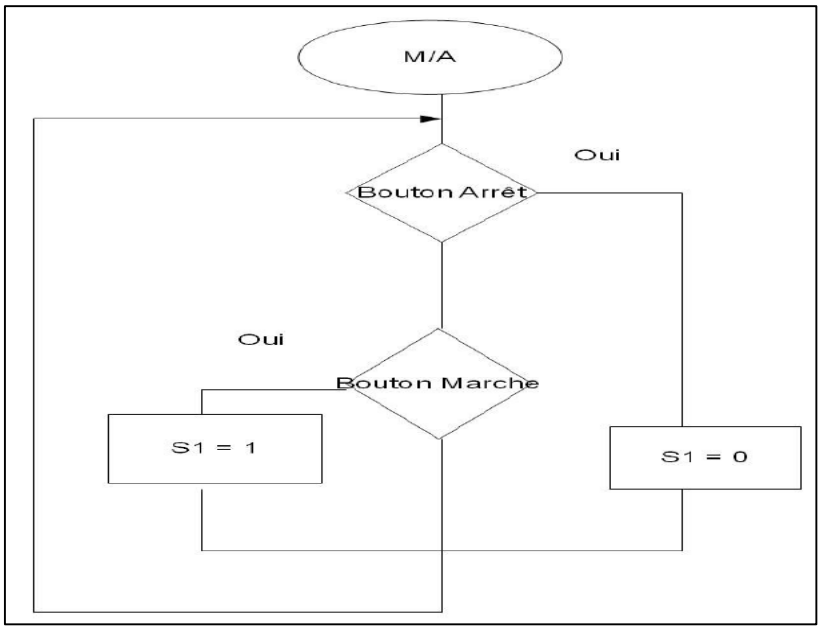
Après ouverture de la session sur le poste de travail, nous avons recopié le dossier « Gpio.c » sur le bureau de notre PC dans un répertoire correspondant à notre TP.
Lancement par la sélection du fichier « Gpio.mcw » qui appelle l'environnement de travail « Mplab ».

Cahier des charges du programme exemple :

Après l'initialisation des ports GPIO correspondant aux boutons et aux leds qui sont sur la carte d'essai, le programme doit scruter « **polling** » l'état des boutons et allumer respectivement 2 leds s'ils sont enclenchés. Pour cela on emploie une boucle 'sans fin' qui est constituée de l'instruction « **while (1)** ». Le test de l'état des boutons est réalisé par une séquence « **if…else** ».

Exercice-1- :

Réaliser la fonction **M/A** avec une priorité sur l'action « Arrêt »



Particularités du programme et solutions trouvées :

= Pour rendre le programme plus clair on utilise des définitions « #define » qui permettent d'associer une étiquette à un périphérique matériel précis.

- ➔ #define Sw1 PORTBbits.RB0 ← Signifie que le bouton Sw1 est câblé sur le PortB bit0.
- ➔ #define D1 PORTDbits.RD0 ← Signifie que la led D1 est câblée sur le PortD bit0.

! On s'aperçoit que « 0 » ou « false » la valeur testée du bouton lorsqu'il est enclenché !

- ➔ En regardant le schéma le bouton quand on l'enclenche remet l'entrée correspondante à la masse -> l'état « 0 » ou « False », au repos une résistance de « pull-up » maintien l'entrée à l'état « 1 » ou « True »

! Le bouton SW2 ne fonctionne pas !

- ➔ Chaque patte du microcontrôleur à plusieurs fonctions, il faut veiller à valider celle que l'on désire utiliser et invalider les autres. **> ADCON1=0x0f ; // Inhibition du port analogique, validation du port digital**

Exercice-2- :

Réaliser la fonction **minuterie**

Une action sur le bouton S1 provoque l'allumage de la led D2 pendant 10s environ.

Particularités du programme et solutions trouvées :

= Le microcontrôleur réalise une instruction (Opcode) en 400ns environ ; il faut perdre du temps pour réaliser cette temporisation soit environ 25 000 000 instructions pour obtenir 10s. Une boucle sera nécessaire.

➔ Une bibliothèque « Delays.h » est fournie par le compilateur, elle permet un décompte de temps précis ; il faut néanmoins signaler au compilateur qu'on doit l'utiliser avec la directive « #include <delay.h> »

➔ L'horloge interne est à la fréquence de l'oscillateur (Quartz 10MHz) divisée par 4. Donc 1 Cycle = 2.5Mz = 400ns => pour attendre 1 seconde il faut dérouler 2 500 000 cycles soit 250 x 10KTCyx.

➔ A noter que pour l'appel de ce fichier les signes « <> » sont utiliser pour spécifier au linker de prendre ce fichier dans le dossier du compilateur et non de l'application.

Exercice-3- :

Réaliser la fonction **télérupteur**

Une action sur le bouton S2 provoque le changement d'état de la Led D3.

Particularités du programme et solutions trouvées :

! On s'aperçoit que lorsque S2 est appuyé la sortie D3 change en permanence de valeur « 0 » ou « 1 ».

➔ Il faut déterminer le changement d'état de S2 et non son état On/Off

! On s'aperçoit que le résultat est incertain …

= Le microcontrôleur réalise une instruction (Opcode) en 400ns environ…

➔ En fait un bouton est un système mécanique qui est réalisé avec un ressort, et lorsqu'on appuie sur ce bouton, le contact rebondi plusieurs fois avant de s'établir durant qq ms; il y a donc une succession d'état « 0 » et d'état « 1 » qui se présente sur l'entrée du port concerné pendant quelques ms. Vue que le microprocesseur scrute rapidement ce port (en quelques µs) il faut introduire dans la boucle une temporisation supérieure à ces « rebonds » Là encore nous avons employé les fonctions de la bibliothèque « delay.h »

Exercice-4- :

Réaliser la fonction **Décalage**

Une action sur le bouton S1 provoque un décalage à gauche, une action sur S2 provoque un décalage à droite

Particularités du programme et solutions trouvées :

= Le programme boucle les décalages dès que l'on appuie sur un bouton

! Ce n'est pas un problème de rebond mais il faut réaliser l'action de décalage une seule fois.

➔ Il faut détecter le changement d'état du bouton (détection de front) et ne réaliser le décalage qu'à cet instant ; 2 mémoires d'état mémoriseront respectivement l'état des 2 boutons pour vérifier à quel moment ils changent et donc réaliser le décalage.

! Il y a une perte du point lumineux lorsqu'on dépasse les côtés.

➔ Il est souhaitable de réaliser un test de limite après chaque mouvement.

CONCLUSION :

Ce TP nous a permis de découvrir quelques caractéristiques du microcontrôleur et certaines fonctions du **C**.

Il est important d'étudier le travail à réaliser avant de produire du code ; c'est la phase « **Analyse** ». Une recherche sur les documents qui décrivent la carte (matériel), sur le document qui décrit les fonctions du langage C (le cours) et le document qui décrit la structure du microcontrôleur (**datasheet**).

Nous avons mis en œuvre lors de ce TP les ports **GPIO**, les entrées-sorties digitales du microcontrôleur, qui représentent des éléments essentiels pour l'interface dans un système industriel (automate).

Avec l'exercice 3 & 4 nous avons mis en œuvre des **variables** pour détecter le changement d'état d'une entrée digitale. Ces exercices font également appel à des bibliothèques standard.