|  |  |
| --- | --- |
|  | L2 EEA  L2 Informatique |

Programmation en C du microcontrôleur PIC16F690 avec MPLAB X et XC8

I. Présentation de la plateforme logicielle/matérielle de TD/TP : 1

II. Installation de l'IDE "MPLABX" et du compilateur "XC8". 2

III. Manuel de référence du compilateur XC8 2

IV. Créer un nouveau projet 2

V. Ajouter un nouveau code source au projet 3

VI. Associer au projet un code source existant 3

VII. Configuration des registres fonctions du PIC16F690 3

VIII. Découverte du programmateur picKit2 et de la platine de test « Low Pin Count » 4

IX. Compilation 6

X. L’écriture du code compilé dans la mémoire du PIC 6

XI. Premier programme 6

XII. Introduction d’une fonction dans le code C 7

XIII. La fonction delay de la fonction « chenil » 7

XIV. Utilisation de code de condition et du Pin RA0 pour contrôler le chenillard 8

XV. Bibliographie 8

Objectifs : découverte de l’environnement logiciel et matériel, utilisation des I/Os numériques du microcontrôleur, découverte des registres de paramétrage du microcontrôleur, révision de la notion de fonction en C, code de condition d’exécution.

# Présentation de la plateforme logicielle/matérielle de TD/TP :

|  |  |
| --- | --- |
|  |  |
| 1. Programmateur Pickit 2 + Carte de test « low pin count » à 4 LEDs contenant le microcontrôleur PIC16F690 | 1. Le logiciel MPLAB-X pour saisir les codes sources C, compiler ceux-ci et les charger en mémoire du microcontrôleur 16F690 |
|  |  |
| 1. Le logiciel « Pickit 2 programmer » pour charger votre fichier compilé dans la mémoire du PIC16F690(1) (autre possibilité) | 1. Relation en les registres TRIS et PORT du µC |

*(1) Il est possible de programmer le µC 16F690 directement à partir de MPLAB-X*

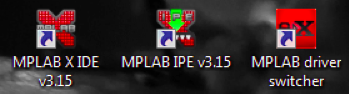
Votre séquencement d’actions pendant les séances de TDs/TPs sera le suivant :

* Saisir votre code dans MPLAB X (le simuler éventuellement)
* Compiler votre code dans MPLAB X
* Charger votre fichier compilé (.hex pour le PIC16F690) dans le le PIC16F690 :
  + Directement à partir de MPLAB-X
  + Ou par le logiciel Pickit 2 programmer qui va écrire votre fichier compilé (.hex pour le PIC16F690) dans le PIC16F690 (fonction « write » de l’interface)
* Vérifier la bonne fonctionnalité de votre code au niveau hardware.

# Installation de l'IDE "MPLABX" et du compilateur "XC8".

Avant de commencer la programmation en "C" des microcontrôleurs, il faut installer les logiciels compatibles avec la programmation C de ces µC (micro-contrôleurs). Il faut :

* un éditeur "IDE" (MPLAB X IDE) pour écrire le programme en langage "C",
* un compilateur pour transformer le programme écrit "C" en un fichier "hex" compatible avec le microprocesseur,
* optionnel : un logiciel « PicKit 2 programmer » (ou MPLAB IPE qui est un programme pour programmer, lire, effacer, la mémoire du microcontrôleur) pour transférer dans le microcontrôleur le fichier "hex" obtenu après compilation.



Ces logiciels sont disponibles sur le site de Microchip, il faut télécharger "MPLAB X IDE" pour votre système d’exploitation (OS), puis le fichier "MPLAB XC8 Compiler" qui est le compilateur (à installer dans un 2ème temps dans la logique des choses). Optionnel : le programme « PicKit 2 programmer » (et sa documentation) est sur le Moodle de l’UE (Moodle/FST/L2/EEA/…SYST MICRO).

XC8 est le compilateur que l'on va utiliser. Il en existe d'autres; toutefois celui-ci a l’avantage d’être gratuit, de manière analogue à GCC, il va convertir les instructions écrites en C (langage compréhensible par l'homme) en langage machine (compréhensible par le microcontrôleur). Il est spécifique pour la programmation de µCs 8 bits. Il existe le même compilateur pour les µCs 16 bits et 32 bits.



# Manuel de référence du compilateur XC8

Document clé pour trouver les mots du langage de programmation : Voir le fichier sur Moodle : MPLAB\_XC8\_C\_Compiler\_User\_Guide.PDF : (Moodle/FST/L2/EEA/…SYST MICRO, dossier « Logiciels et documents pour le PIC16F690 »). Vous utiliserez aussi avantageusement des moteurs de recherche pour étudier des codes et tutoriaux pour XC8.

# Créer un nouveau projet

Vous devez au préalable lancer MPLABX IDE.

* Lancer par la suite l’assistant de création de projet (File → New Project).
* A la première page, il faut choisir la catégorie « Microchip Embedded » et un projet du type « Standalone project ». Puis cliquer sur Next.
* Choisir ensuite le microcontrôleur sur lequel vous travaillez dans Device (MidRange 8 bits, le PIC16F690). Puis cliquer sur Next.
* Laisser l’option None dans la partie « Select Header ». Puis cliquer sur Next.
* A la page « Select Tool », sélectionner l’option Pickit2. Puis cliquer sur Next.
* A la page « Select Compiler », sélectionner l’option XC8 pour développer en C. Puis cliquer sur Next.
* A la page Select Project Name and Folder, donner un nom à votre projet (TP1\_prog1 par exemple) et choisissez votre répertoire de travail. Puis cliquez sur Finish.

# Ajouter un nouveau code source au projet

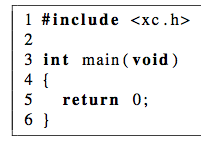
Si le fichier source n’est pas donné, il faudra le créer par la procédure suivante :

Faire un *clic droit* sur le dossier « Source Files ».

Sélectionner ensuite « New… ».

Choisir ensuite la catégorie Microchip Embedded / XC8 Compiler et le type de fichier main.c. Puis cliquer sur Next. Donner un nom à votre fichier source (TP1\_prog1\_main.c) et cliquer sur Finish.

Vous devriez obtenir un fichier \*.c ressemblant à ceci :



# Associer au projet un code source existant

Faire un *clic droit* sur le dossier Source Files. Sélectionner « Add existing file ». Suivre les indications proposées.

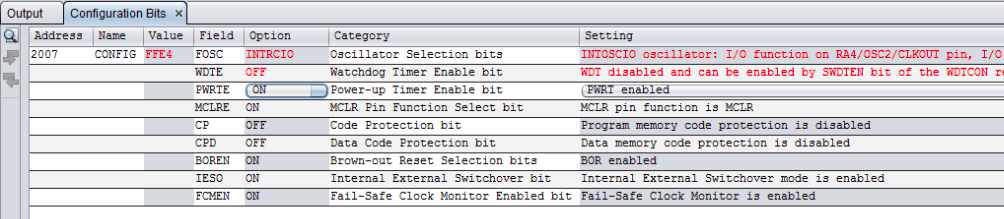
# Configuration des registres fonctions du PIC16F690

Les µCs sont des machines numériques autonomes [pmc1,pat1]. La programmation des µCs a une spécificité : il faut configurer les fonctions internes (horloge, convertisseur analogique numérique, watchdog timer, …) du µC pour placer le µC dans une configuration adéquate en relation avec le cas traité. Ces fonctions internes sont incontournables pour le fonctionnement de cette machine numérique. Pour cela :  cliquez sur le menu "Window" puis "Pic Memory Views" et enfin "Configuration Bits"  pour obtenir une fenêtre qui permet de configurer le microcontrôleur.

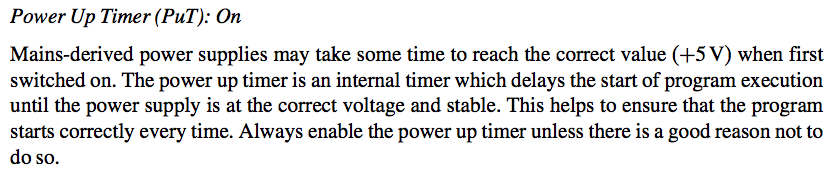
|  |  |
| --- | --- |
|  |  |
| 1. Exemple de fonctions internes du µC | 1. Assistant de configuration par le menu |

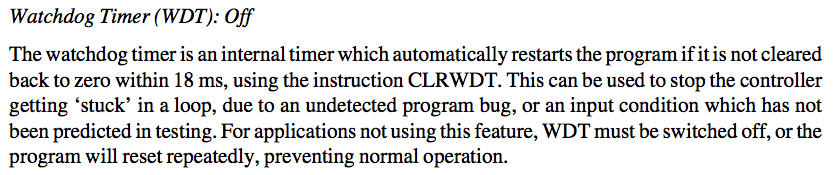
Il faut maintenant configurer notamment l'horloge, le watchdog, ... pour cela une lecture de la datasheet PIC16F690 correspondante est conseillée pour comprendre la configuration.

Dans l'exemple ci dessous nous voyons : que l'horloge interne est choisie comme horloge principale (Fosc=INTRCIO), le watchdog (WDTE) est placé sur OFF, le pin reset (MCLRE) est actif sur ON :



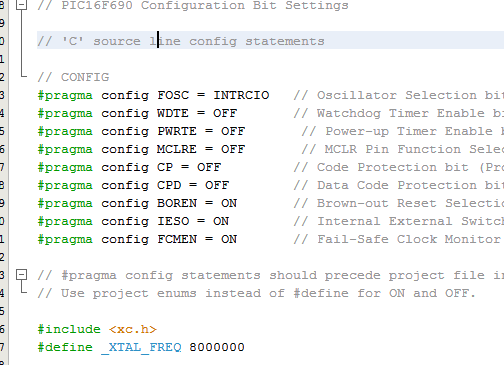
Voici la signification de quelques un des champs présentés :





Changez dans MPLAB-X, le MCLRE sur OFF (vous le ferez pour tous les codes C).

Un clic sur "Generate Source Code to Output" pour obtenir dans une nouvelle fenêtre le code C correspondant. On sélectionne tout le texte et un "Ctrl +c" et Ctrl + v", pour copier le texte dans la fenêtre "main.c" et le résultat obtenu est :



1. code configuration du PIC

Vous ajouterez la ligne suivante à votre code C pour avoir accès aux fonctions **delay** :

**#define \_XTAL\_FREQ 8000000.**

Remarque : vous verrez lors de vos lecture des lignes du type :

\_\_CONFIG(INTIO & WDTDIS & PWRTEN & MCLREN & UNPROTECT & UNPROTECT & BORDIS & IESODIS & FCMDIS);

qui remplaceront le code précédent.

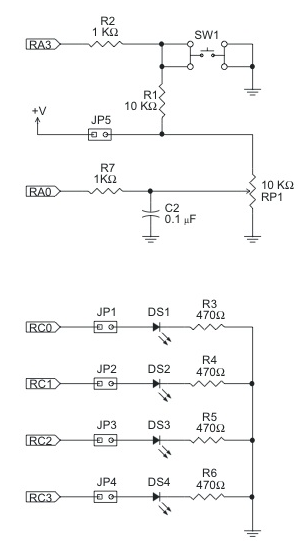
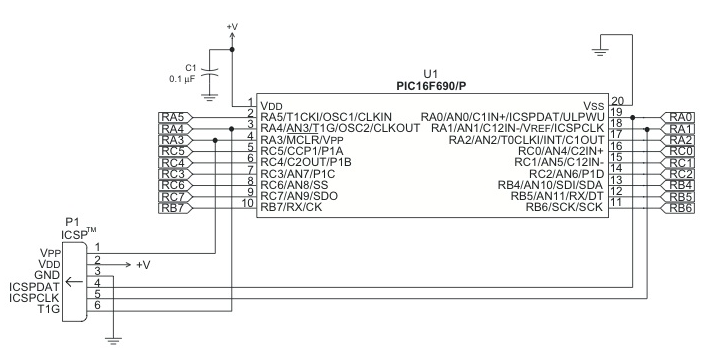
# Découverte du programmateur picKit2 et de la platine de test « Low Pin Count »

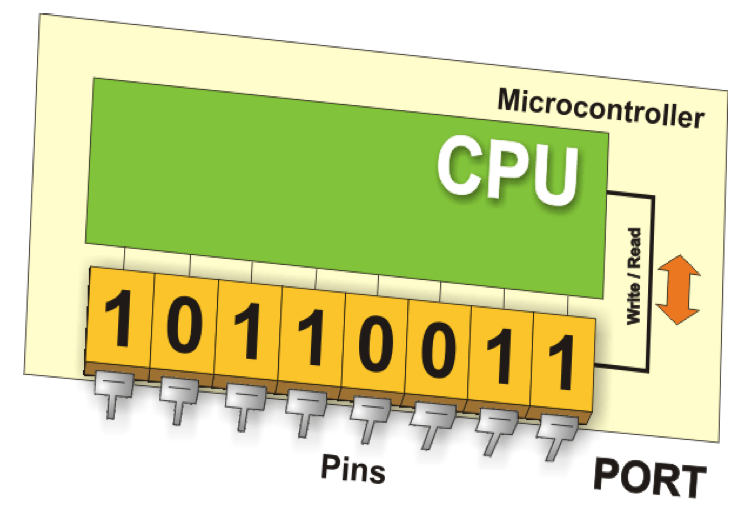
Voici la platine « **low pin count**» et le programmateur **pickit 2**. Faites bien la différence entre les deux cartes en terme de fonctionnalité.

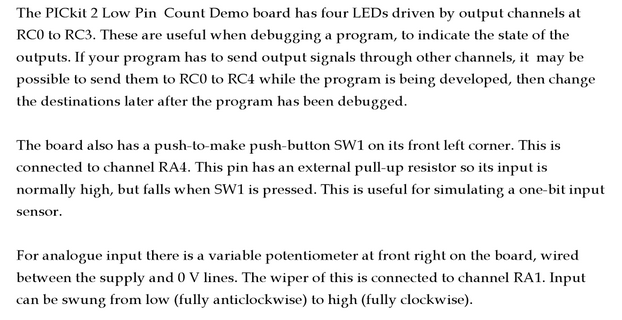
|  |  |
| --- | --- |
|  | pickit2_c |

1. Avec quels µCs PIC le « low pin count demo board » est-il compatible (étude du document correspondant sous Moodle : dossier « Logiciels et documents pour PIC16F690 ») ?
2. Quelle est l’extension du format de fichier compilé dans le cas du PIC ?
3. Quel(s) logiciel(s) employer pour charger votre exécutable dans le PIC (faites au minimum une proposition)?

Voici le schéma électronique de la carte « low pin count » :





[bis1] 

1. Sur quel **port** du PIC16F690 sont connectées les 4 LEDs (le terme PORT a une signification spécifique)?
2. Donner plus précisément les numéros des pattes (ou PIN) sur lesquelles sont connectées les LEDs et les références exactes des numéros de ports concernés par la connexion avec les LEDS ? Quelle valeur booléenne placer sur les PINs reliées aux LEDs, pour allumer ces LEDs ?
3. En appuyant sur le switch présent sur la platine (voir schéma électronique et la platine réelle), quelle valeur binaire applique-t-on et sur quelle patte ?
4. y-a-t’il un crystal sur la platine « low pin count » ? A quoi sert-il sur une carte à microcontrôleur ?

# Compilation

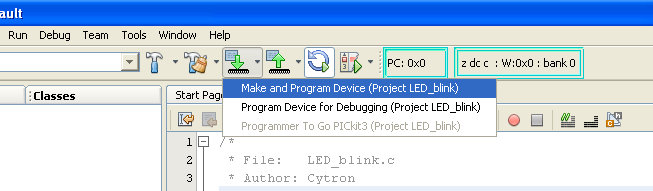
La compilation se fait en cliquant sur l’icône suivante de l’interface :



Si la compilation ne retourne pas d’erreurs, un fichier \*.hex est alors généré dans le répertoire /dist/default/production/ du répertoire de travail. Vérifier ce point avec votre explorateur de fichier spécifique à votre OS.

# L’écriture du code compilé dans la mémoire du PIC

Toujours à partir de l’interface MPLAB-X, voici l’icône à utiliser (il faut bien sûr que le « pickit 2 » soit connecté à l’ordinateur via le port USB). Observez la fenêtre de MPLAB-X qui vous renseigne alors sur le bon déroulement de la procédure. Vous pouvez en outre programmer le µC 16F690 avec le «Pickit 2 programmer» si un souci se présente avec la première méthode.



# Premier programme

Repassez par les étapes proposées pour arriver au début du code suivant (vous taperez le code restant pour vous familiariser avec la syntaxe) :

// PIC16F690 Configuration Bit Settings

// 'C' source line config statements

// CONFIG

#pragma config FOSC = INTRCIO // Oscillator Selection bits (INTOSCIO oscillator: I/O function on RA4/OSC2/CLKOUT pin, I/O function on RA5/OSC1/CLKIN)

#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled and can be enabled by SWDTEN bit of the WDTCON register)

#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT enabled)

#pragma config MCLRE = OFF // MCLR Pin Function Select bit (MCLR pin function is MCLR)

#pragma config CP = OFF // Code Protection bit (Program memory code protection is disabled)

#pragma config CPD = OFF // Data Code Protection bit (Data memory code protection is disabled)

#pragma config BOREN = ON // Brown-out Reset Selection bits (BOR enabled)

#pragma config IESO = ON // Internal External Switchover bit (Internal External Switchover mode is enabled)

#pragma config FCMEN = ON // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is enabled)

// #pragma config statements should precede project file includes.

// Use project enums instead of #define for ON and OFF.

#include <xc.h>

#define \_XTAL\_FREQ 8000000

int main()

{

TRISC = 0x0;

PORTC=0x0;

while(1)

{

PORTC=0b00000011;

\_\_delay\_ms(1000);

PORTC=0b00001100;

\_\_delay\_ms(1000);

}

return 0;

}

1. code0.c : premier programme

Tester ce code sur la carte « low pin count ».

1. Que réalise ce code ?
2. Pourquoi place t’on un octet dans le registre PORTC ? Pourquoi un octet ?
3. Expliquer la ligne TRISC = 0x0; (travaillez à partir des informations que vous trouverez sur le registre TRISC du PIC16F690) ?
4. Expliquez ce que font les instructions C de ce code ?
5. Proposez alors à partir de l’exemple précédent un code (code1.c) d’un chenillard utilisant les 4 LED ?

# Introduction d’une fonction dans le code C

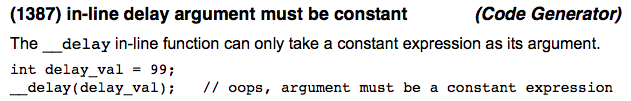
1. Proposez un nouveau code (code2.c) qui réalise la même fonctionnalité que le chenillard de la question précédente, avec l’utilisation d’une déclaration de fonction « chenil ()» (même délai constant, même séquence d’allumage) ?

# La fonction delay de la fonction « chenil »

Modifier la fonction « chenil() » pour y introduire une variable entière « temps » qui permet de contrôler le temps d’évolution du chenillard.

1. Donner votre code (code3.c) ? Ce code fonctionne t’il ?

Voici une partie de la documentation qui explique le souci :



Pour résoudre le souci, il faut créer notre propre fonction « delay » :

1. Remettez dans l’ordre le code suivant pour créer la fonction voulue ? Donnez alors le nouveau code (code4.c) du chenillard incluant cette fonction « delay\_ms » qui est différente de la fonction « \_\_delay\_ms » de XC8.

\_\_delay\_ms(1);

{

void delay\_ms(unsigned int milliseconds)

{

milliseconds--;

}

while(milliseconds > 0)

}

# Utilisation de code de condition et du Pin RA0 pour contrôler le chenillard

L’objectif ici est de lire la valeur numérique présente sur RA0. Si RA0 est à « 1 » le chenillard fonctionne. Si RA0 est à « 0 » le chenillard est éteint ()

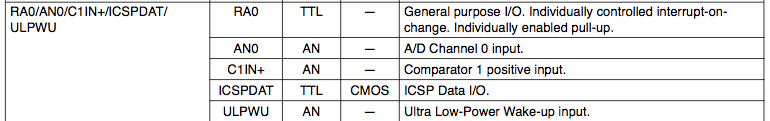
Pour réaliser ce mini projet voici les éléments d’information :

1. A partir de l’étude de cas "EtudeCas\_Using Push Button Switch.pdf" (Moodle/FST/L2/EEA/…SYST MICRO, dossier « Logiciels et documents pour le PIC16F690 ») :
2. The ANSEL and ANSELH are the analog select registers; by default all the analog ports in PIC16F690 are set to analog input not for digital I/O, therefore by clearing all bits (set to logical 0) in these special function registers we instruct the PIC microcontroller to use all the available analog input as a digital I/O port. This is one important preparation before we use the PIC port for digital I/O :

ANSEL = 0; /\* Set PORT ANS0 to ANS7 as Digital I/O \*/

ANSELH = 0; /\* Set PORT ANS8 to ANS11 as Digital I/O \*/

1. Le « 0 » logique est physiquement représenté par la masse, noté Vss sur la platine « low pin count ». Il faudra ajouter un fil faisant office d’interrupteur.
2. Par défaut le PULL-UP ( ?) est actif sur RA0



1. Proposer un code (code5.c) pour contrôler l’exécution du chenillard suivant l’algorithme souhaité.

# Bibliographie

[ele1] https://electrosome.com/switch-pic-microcontroller-mplab-xc8/

[bis1] Robot Builder's Cookbook: Build and Design Your Own Robots, Par Owen Bishop

[pmc1] http://www.pmclab.fr/wiki/dokuwiki/doku.php?id=wiki:tutoriels:pic

[pat1] http://patrickelectro.blog.free.fr