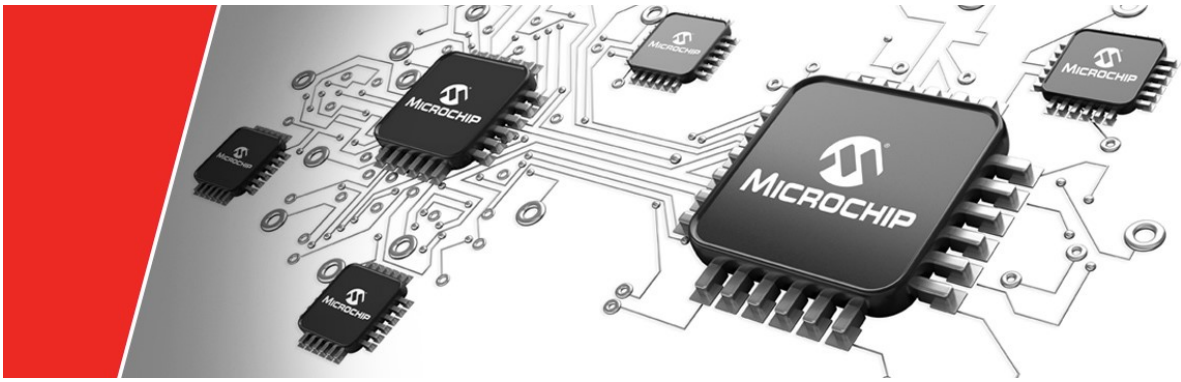


TP microcontrôleur

Mini-projets sur PIC18

JUNIA ISEN

Novembre 2024

Sommaire

1 – Introduction

L'objectif de ce mini-projet est d'étudier le fonctionnement d'un microcontrôleur 8 bits, de la famille PIC18 de Microchip, ainsi que son environnement de développement **MPLAB X**.

Ce dernier est ce que l'on appelle un IDE (Integrated Development Environment) : il intègre tous les outils de développement d'un logiciel embarqué :

- gestionnaire de projet,

- écriture d'un programme, que ce soit en assembleur ou en utilisant un langage évolué (langage C),

- compilation / édition de lien,

- simulation,

- chargement du fichier exécutable dans un composant,

- test du programme sur le composant cible.

La version du logiciel nécessaire au TP est la **version 5.35**, et est disponible à cette adresse : https://www.microchip.com/en-us/tools-resources/archives/mplab-ecosystem#MPLAB%20X%20IDE%20Archives

Attention, ce n'est pas la dernière version du logiciel, vous devez la récupérer dans les versions archivées du logiciel MPLAB X.

Durant ce mini-projet, une série d'exercices permettra d'aborder les principales fonctionnalités d'un microcontrôleur.

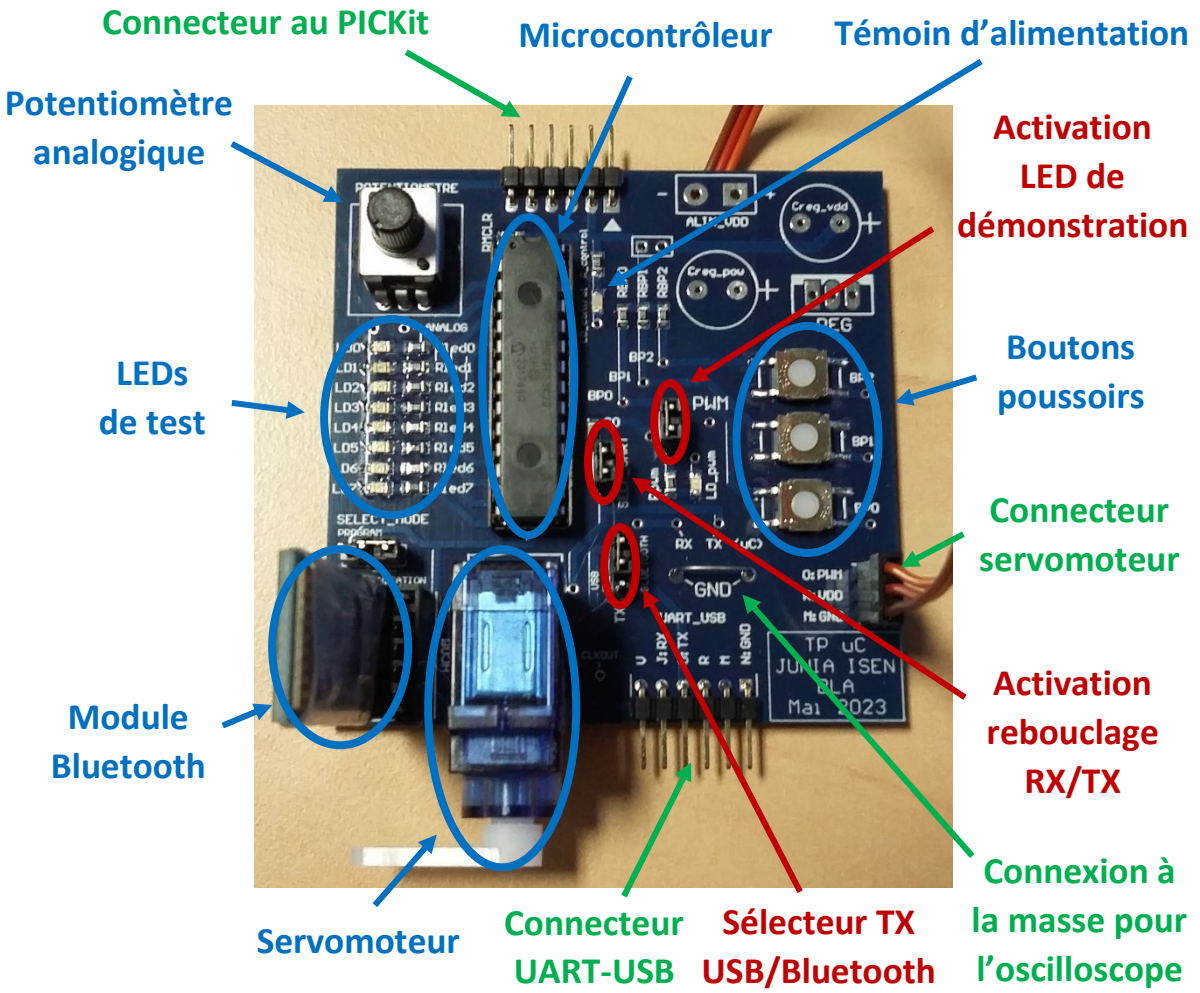
Tous les programmes seront écrits **en assembleur**. Pour certains exercices, la vérification se fera en simulation uniquement, tandis que pour d'autres, le test se fera à l'aide d'une carte d'apprentissage comportant un microcontrôleur ainsi que quelques modules (bouton poussoir, LEDs, potentiomètre, etc.).

La datasheet du microcontrôleur utilisé est fournie dans le répertoire du TP et sera prise pour référence pour l'ensemble des commandes utilisées.

Il est fortement conseillé de créer un nouveau projet sous MPLAB à chaque exercice. Il n'est pas possible d'utiliser plusieurs fichiers sources en assembleur au sein d'un même projet.

2 – Carte de prototypage

La carte de prototypage utilisée pour cette série d'exercices est basée sur un microcontrôleur **PIC18F23K20**. Elle est décrite dans la figure ci-dessus :



Elle permet de mettre en œuvre les principales fonctionnalités du composant (écriture et simulation d'un programme, gestion des entrées/sorties, utilisation des périphériques, etc.).

Cette carte sera utilisée via un module Debugger/Programmateur **PICkit3 (**ou **PICKit4** selon les stocks). Ce dernier permet de charger un programme dans un microcontrôleur afin de le tester et le mettre au point. Celui-ci peut être lancé en continu, ou en mode pas-à-pas (une instruction ou quelques instructions à la fois) afin de vérifier le comportement du système. Après validation du programme, il peut être chargé de façon permanente dans la mémoire non-volatile du composant.

3 – Exercices

a. Exercices sur simulateur

Dans cette première série d'exercices (Exercices 1 à 7), la vérification des résultats se fera via le simulateur inclus dans l'IDE MAPLAB, car cette vérification passera par le contenu des registres du microcontrôleur et pas par les périphériques externes présents sur la carte de démonstration.

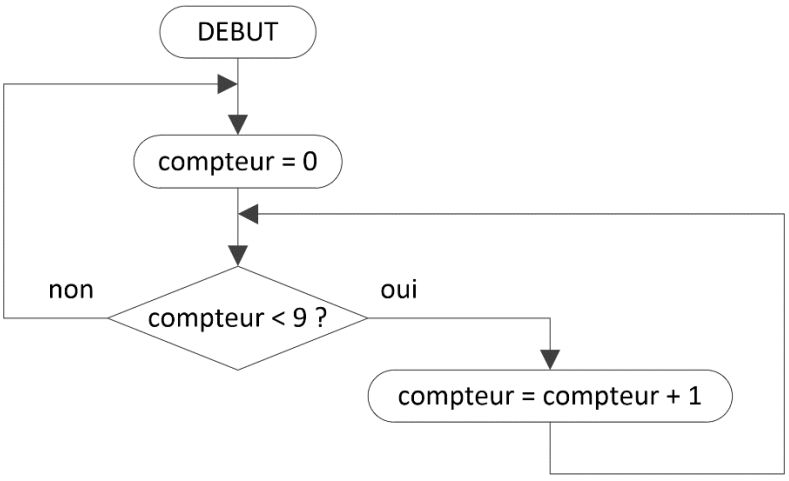
Vous pouvez, à chaque exercice, créer un nouveau projet (voir Annexe 1), ajouter un fichier source en assembleur (voir Annexe 2), et vérifier le fonctionnement en simulation (voir Annexe 3).

Les listes d'instructions permettant de réaliser vos programmes peuvent être trouvées des pages 318 à 320 de la datasheet du microcontrôleur, et l'utilisation de chaque instruction est détaillée à partir de la page 321.

Exercice 1- Réalisation d'un compteur modulo 10

L'exercice consiste à écrire et tester un programme qui compte de 0 à 9, puis recommence à 0, en bouclant indéfiniment.

La valeur du compteur sera mémorisée dans une variable appelée (par exemple) « compteur », qu'il faudra définir dans la zone des variables.



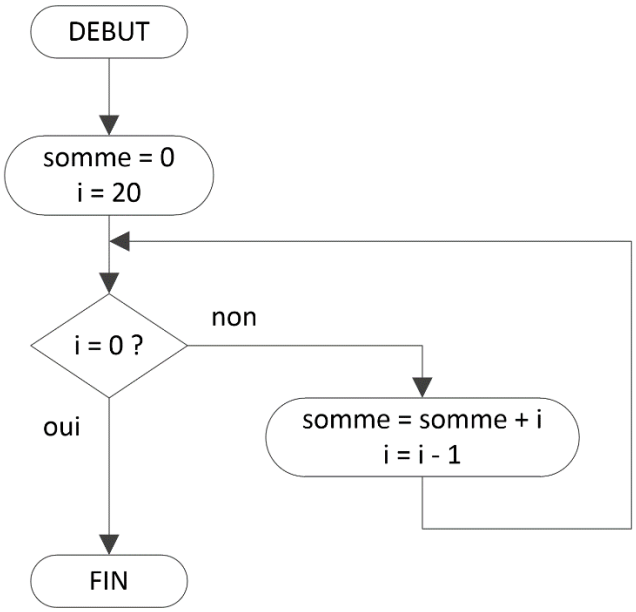
Tester le programme par simulation en mode pas à pas.

Visualiser l'évolution de la variable « compteur », du compteur ordinal (Program Counter), et du registre d'état.

Exercice 2- Réalisation d'une boucle « for » : somme des 20 premiers entiers

Cet exercice illustre comment réaliser en assembleur une boucle « for ». Il consiste à calculer la somme des 20 premiers entiers.

Le résultat sera stocké dans une variable « somme » qu'il conviendra de déclarer dans la zone mémoire dédiée. L'indice de boucle, appelé par exemple « i », sera quant à lui stocké dans une autre variable.



Vérifier le bon fonctionnement du programme en le testant en mode pas-à-pas.

Exercice 3- Manipulation de données codées sur plusieurs octets : somme des 40 premiers entiers

Cette partie reprend l'exercice précédent, mais on cherche maintenant à sommer les 40 premiers entiers. Comme le résultat est supérieur à 255, il faudra le coder sur plus d'un octet (étant inférieur à $2^{16} - 1 = 65535$, 2 octets seront suffisants).

En suivant la même démarche que précédemment, calculer la somme et stocker le résultat dans 2 variables « somme_High » (représentant l'octet de poids fort) et « somme_Low » (représentant l'octet de poids faible).

Exercice 4- Appel d'un sous-programme

En assembleur comme en langage évolué, il est souvent utile de faire appel à des routines (sous-programmes).

Ceci permet :

- d'améliorer la lisibilité et la compréhension d'un programme,
- de n'écrire qu'une seule fois un ensemble d'instructions qui peuvent être exécutées de façon répétitive.

On repartira dans le cas présent sur la base de l'exercice précédent qui consiste à calculer la somme des premiers entiers, mais en faisant cette fois appel à un sous-programme « calcul_somme ».

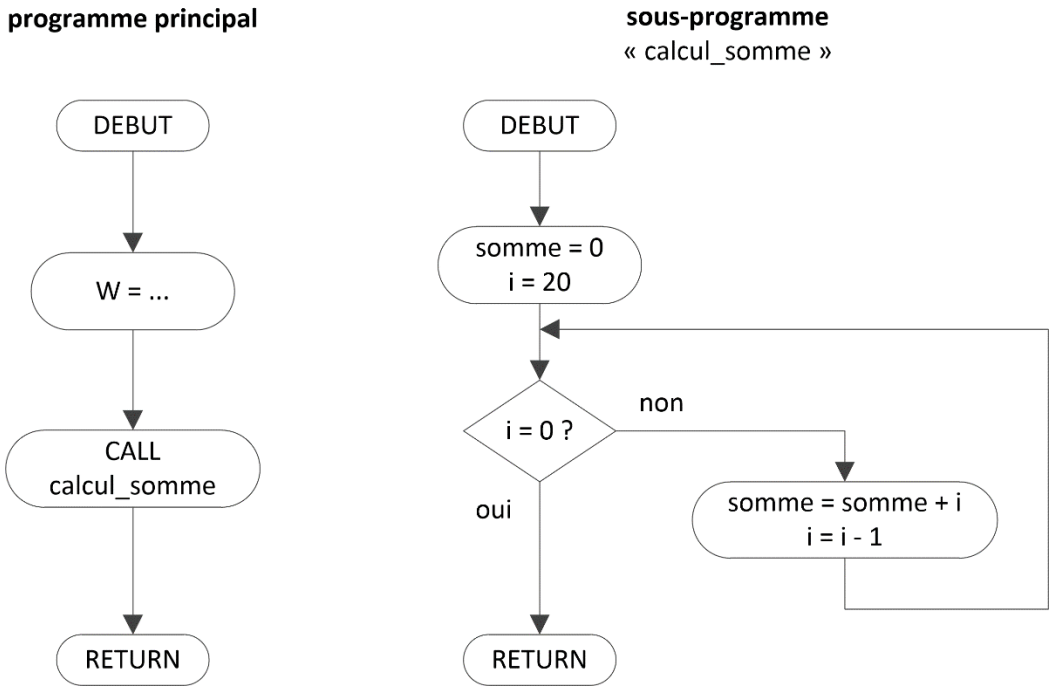
Ce sous-programme :

- prendra en entrée le nombre d'entiers à additionner, stocké au préalable dans l'accumulateur,
- retournera le résultat dans les variables « somme_High » et « somme_Low ».

Rappel : le retour de sous-programme se fait avec l'instruction RETURN.

Le programme principal devra :

- écrire dans l'accumulateur le nombre d'entiers à additionner,
- appeler le sous-programme « calcul_somme », grâce à l'instruction CALL.



Tester le programme en mode pas-à-pas. Visualiser le haut de la pile (Top Of Stack : registres TOSH et TOSL), ainsi que le pointeur de pile (Stack Pointer : registre STKPTR), notamment lors de l'exécution des instructions CALL et RETURN.

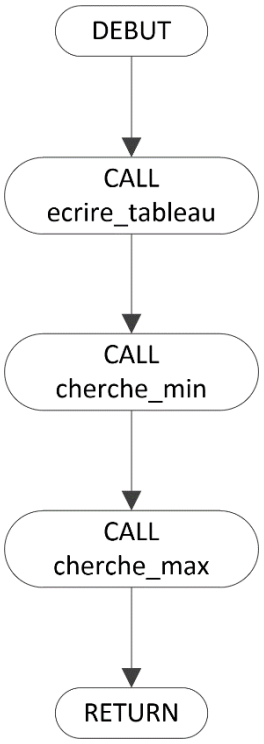
Exercice 5- Utilisation d'un tableau (données stockées dans la mémoire RAM) : recherche de minimum et de maximum

Dans certaines situations, il est utile d'organiser les données sous formes de tableaux. C'est notamment le cas lorsque l'on souhaite passer en revue (écrire ou lire) un groupe de données avec lesquelles on travaille.

Dans l'exemple qui suit, on cherchera le minimum et le max d'un ensemble de données organisées dans un tableau, et stockées dans la mémoire RAM. Cette dernière étant volatile (effacée lors d'une coupure d'alimentation), il faudra dans un premier temps remplir le tableau, pour ensuite le scruter afin d'en extraire les valeurs min et max.

1) Le programme principal devra d'abord appeler un sous-programme « ecrire_tableau » qui initialise un tableau avec les valeurs suivantes :

tableau[0]	25
tableau[1]	4
tableau[2]	2
tableau[3]	15
tableau[4]	16
tableau[5]	101
tableau[6]	33
tableau[7]	3



Les données seront écrites à partir de l'adresse 0x100.

Utiliser pour cela l'**adressage indirect** (initialiser par exemple le registre FSR0 à 0x100, puis travailler avec le registre INDF0, ou mieux encore, POSTINC0). Ce sous-programme retournera également la taille du tableau dans une variable « taille_tableau ».

2) Le programme principal appellera ensuite le sous-programme « cherche_min » qui lit successivement toutes les valeurs du tableau, en partant de l'adresse de départ (ici, 0x100), jusqu'à atteindre la dernière donnée (que l'on peut identifier grâce à la variable « taille_tableau » mise à jour précédemment). Ce sous-programme retournera la plus petite valeur qu'il a trouvée, mise dans la variable « min ».

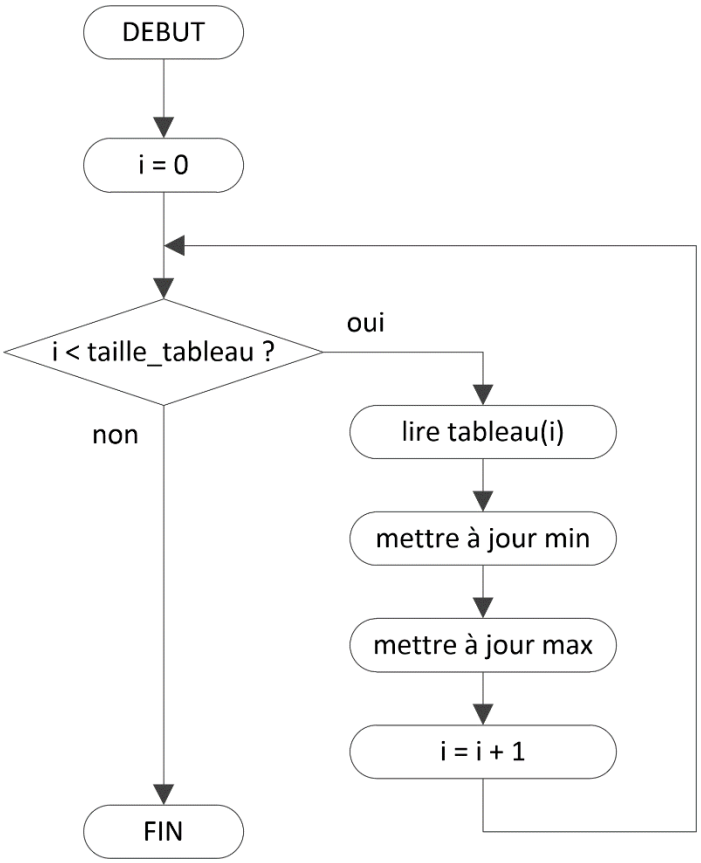
3) Le programme principal appellera enfin le sous-programme « cherche_max » qui, de la même manière que précédemment, retournera la plus grande valeur du tableau, mise dans la variable « max ».

Tester le programme par simulation.

Lorsque les données sont écrites dans la mémoire RAM (volatile), il faut au préalable les initialiser, comme c'est le cas dans l'exemple précédent. Cette étape est cependant inutile si l'on travaille avec des constantes déjà écrites dans la mémoire Flash (ou mémoire programme, donc non volatile). Pour lire ces informations, il faudra donc les récupérer de la mémoire programme, afin de les placer dans la mémoire RAM et pouvoir ainsi travailler avec.

Il est demandé ici de chercher comme précédemment le minimum et le maximum dans la table de valeurs de l'exercice 5, mais comme ces dernières seront des constantes, il ne faudra pas initialiser le tableau : il suffira de le lire.

Dans les exercices 6 et 7 (l'exercice 7 est facultatif), l'algorithme du programme sera le suivant :



Deux méthodes sont possibles pour réaliser le stockage des données en mémoire Flash. Pour chacun des exercices 6 et 7 (l'exercice 7 est facultatif), tester le programme en simulation. Vous pourrez ensuite étudier l'espace mémoire nécessaire utile pour stocker les éléments du tableau, c'est-à-dire combien d'octets doivent être réservés dans la mémoire programme pour stocker un élément supplémentaire ?

Exercice 6- Utilisation d'un tableau (données stockées dans la mémoire Flash) : *Table Read*

(cf. page 89 de la datasheet du PIC18F23K20, exemple de code page 93)

Avec cette méthode, il est possible de transférer un octet de la mémoire programme dans un registre (TABLAT). L'octet dont l'adresse (sur 21 bits) est indiquée dans un pointeur dédié (registres TBLPTRL, TBLPTRH et TBLPTRU) est ainsi transféré dans TABLAT par une instruction spécifique (TBLRD*, TBLRD*+, etc.).

Le registre TABLAT peut ensuite être lu comme n'importe quel registre de la mémoire RAM.

Dans le cas où l'on souhaite placer les données à une adresse particulière dans la mémoire Flash (par exemple en 0x0100), il faut créer un bout de programme dont la syntaxe est la suivante :

```
label       CODE  0x0100
autre_label DB    0x19, 0x04, ..... (les valeurs peuvent être en décimal, hexadécimal, etc.)
```

Cette partie du programme sera stockée à l'adresse 0x0100, et doit être bien séparée du programme principal (indiqué par la ligne *MAIN_PROG CODE*). Dans la mémoire Flash, la programme sera donc séparé en deux : la partie indiquée par le mot-clé *label* à l'adresse 0x0100, et l'autre partie indiquée par le mot-clé *MAIN_PROG* à une adresse automatiquement affectée par le compilateur.

Pour chaque donnée stockée dans la mémoire programme, un seul octet est utilisé dans la mémoire Flash (contre 2 avec la méthode « Computed GOTO » de l'exercice suivant).

Exercice 7- Utilisation d'un tableau (données stockées dans la mémoire Flash) : *Computed GOTO*

(Exercice facultatif, utile pour comparaison avec la méthode précédente)
(cf. page 68 de la datasheet du PIC18F23K20)

Avec cette méthode, la constante est associée à l'instruction « RETLW nn », qui a le même effet que l'instruction RETURN, mais retourne en plus la valeur 'nn' dans le registre W.

Il faut pour cela appeler une routine, par exemple grâce à l'instruction « CALL Table ». W devra au préalable être chargé avec la valeur de l'indice de l'élément à aller chercher dans le tableau. Au début de la routine « Table », la valeur du Program Counter (registre PCL) sera décalée ce qui permettra d'exécuter ensuite l'une des instructions « RETLW nn » de la liste.

Dans le cas où l'on souhaite placer les données à une adresse particulière dans la mémoire Flash (par exemple en 0x0100), la syntaxe est la suivante :

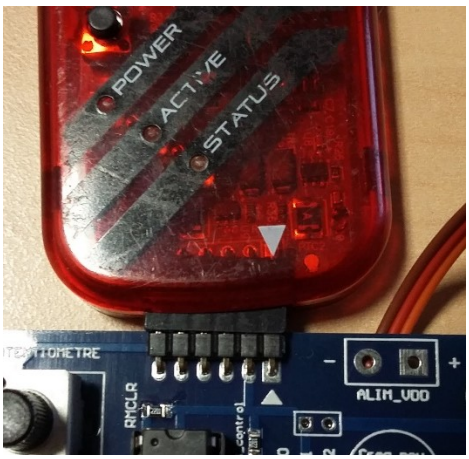
```
...                 ; programme précédent l'appel de lecture du tableau
CALL        Table
...                 ; programme mettant à jour les min et max

label       CODE  0x0100
Table
...                 ; modification du Program Counter pour
...                 ; aller pointer vers l'un des instructions RETLW qui suit
RETLW       …
RETLW       …
RETLW       …
…
```

Pour récupérer un octet de donnée dans la mémoire programme, cette méthode utilise une instruction, soit 2 octets dans la mémoire programme (opcode + donnée). De plus, la technique « CALL ... / RETURN » occupe une place dans la pile pour stocker l'adresse de retour. Elle n'est donc pas optimale.

b. Exercices sur carte de prototypage

Dans cette deuxième série d'exercices (Exercices 8 à 18), la vérification des résultats se basera sur l'utilisation de la carte de prototypage décrite en section 2 – Carte de prototypage. Comme la cible de programmation est dorénavant un microcontrôleur physique, il faut, au moment de la programmation, définir une configuration de démarrage (fréquence d'oscillation par défaut, état de ports d'entrée/sortie spécifique, etc.). Pour cela, au début de chaque nouveau programme (à chaque exercice donc), il faut éditer les paramètres de cette configuration, comme décrit en Annexe 4. Ensuite, une fois le programme prêt à être tester, il faut le charger sur la carte en suivant la démarche décrite en Annexe 5.

Note : Les images suivantes montrent le branchement d'un PICkit3/PICkit4 sur la carte de prototypage, en utilisant le connecteur en haut de la carte.



Exercice 8- Utilisation des ports d'Entrées/Sorties

La carte de prototypage comporte 8 LEDS LD0 à LD7 connectées sur les ports A et C (pattes RA0 à RA3 pour LD0 à LD3, et pattes RC0 à RC3 pour LD4 à LD7, respectivement).

L'objectif de cet exercice est d'allumer alternativement une LED sur deux.

Le programme devra configurer partiellement les ports A et C en sortie, puis écrire alternativement b'01010101' et son complément sur les pattes correspondantes.

Les registres nécessaires à la configuration des ports d'entrée/sortie (TRISx, PORTx et LATx) sont décrits en page 121 de la datasheet du microcontrôleur.

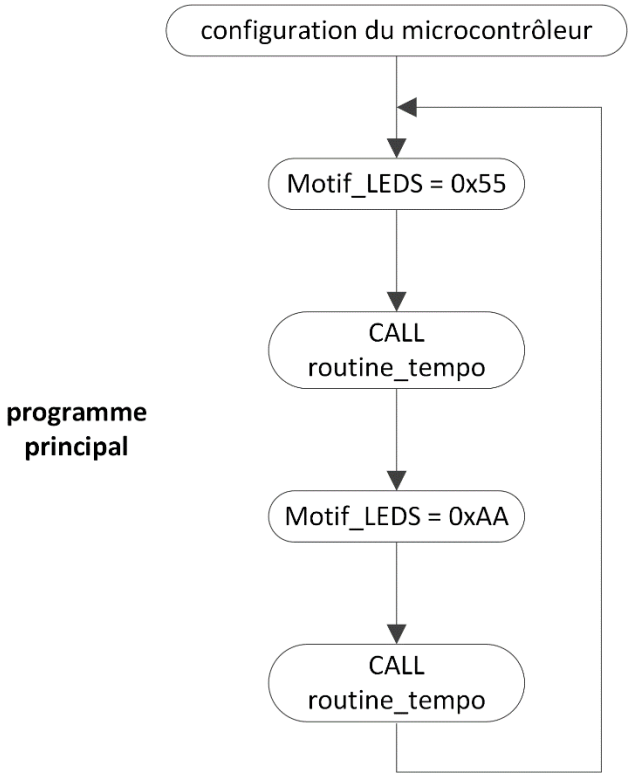
Cependant, afin que le programme puisse passer d'une instruction à l'autre, il faut configurer son horloge. L'oscillateur interne sera utilisé (la carte de prototypage ne comprend pas de quartz), et on fera en sorte que le temps d'exécution d'une instruction soit de 1 µs (= $(1\ \text{MHz})^{-1}$). Il faudra donc configurer la fréquence d'oscillateur à 4 MHz (registre OSCCON, cf. pages 28-29 de la datasheet du PIC18F23K20, fixer à '0' les options inutiles).

Un exemple de description d'un registre extrait de la datasheet est montré en Annexe 6.

Tester sur la carte le programme en mode pas-à-pas.

Exercice 9- Réalisation d'une routine de temporisation software

L'exécution en continu du programme précédent ne permet pas de voir clignoter les LEDs, car la fréquence du microcontrôleur est trop grande. Pour cet exercice, il faudra donc écrire une routine de temporisation qui sera appelée avant de changer l'état du port de sortie.

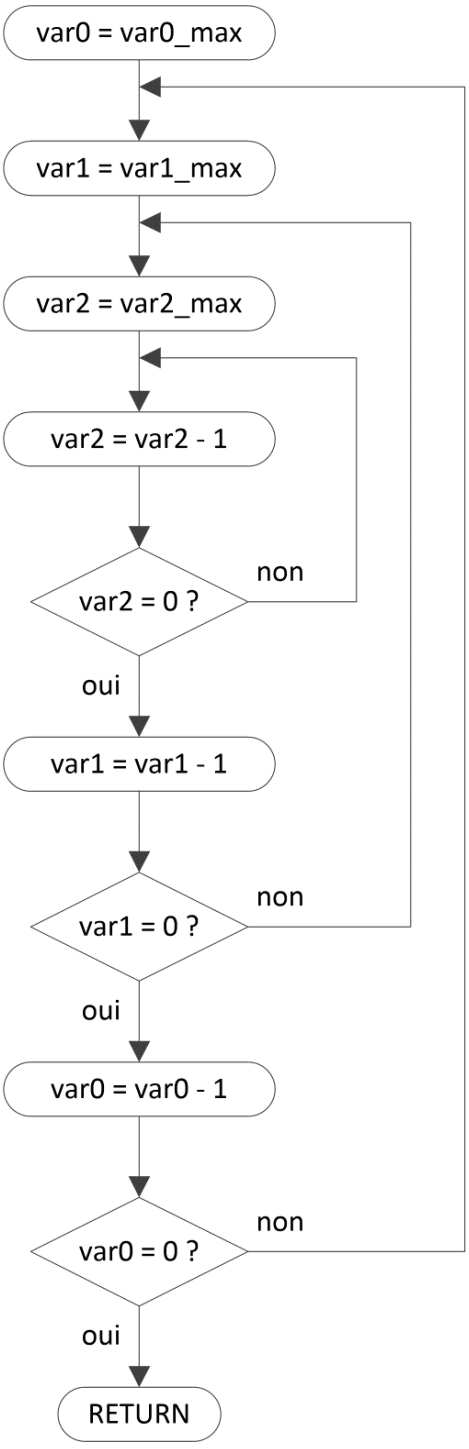


La routine de temporisation ne servira qu'à « perdre du temps ». Elle sera constituée de boucles imbriquées qui décrémenteront des variables (var0, var1, var2), à partir de leur valeur maximale, et jusqu'à ce qu'elles arrivent à 0.

Les valeurs maximales peuvent être modifiées pour obtenir une temporisation la plus proche possible d'une seconde.

(cf. figure page suivante)

sous-programme
« routine_tempo »



Attention, il est à noter que les boucles sont imbriquées, c'est-à-dire qu'une boucle va se décrémenter en entier avant que la suivante se décrémente de 1 seulement, et la première boucle recommence l'opération jusqu'à ce que toutes les variables aient atteint 0.

Tester le fonctionnement de ce programme en mode continu sur la carte de prototypage.

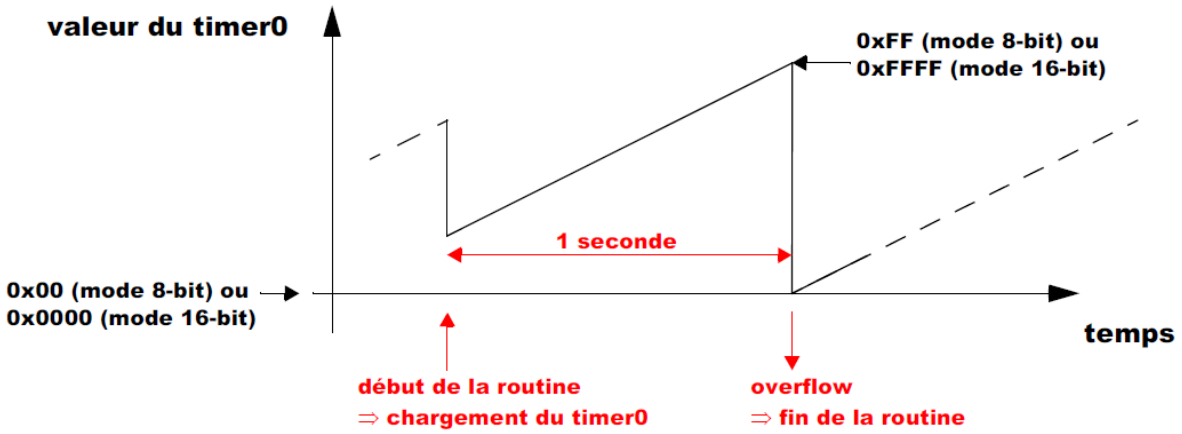
Exercice 10- Réalisation d'une routine de temporisation hardware (utilisation d'un timer)

Pour cet exercice, l'objectif principal est le même que précédemment, et le programme principal gardera la même structure, mais la routine de temporisation se basera sur l'utilisation d'un timer pour fixer la routine de temporisation à **exactement 1 seconde** (à quelques µs près), précision presque impossible à atteindre avec une temporisation software.

C'est le timer0 qui sera utilisé (cf. page 155 de la datasheet du PIC18F23K20). Il sera activé et configuré de manière que sa période totale soit au moins égale à 1 seconde (registre T0CON).

Au début de la routine de temporisation, il faudra charger le timer à une valeur adéquate, puis attendre qu'il ait atteint sa valeur maximale (ou « overflow »). Cet événement peut être détecté en testant le flag d'overflow TMR0IF dans le registre INTCON (cf. page 109 de la datasheet).

Tester le fonctionnement de ce programme sur la carte de prototypage.



Exercice 11- Utilisation des interruptions

La carte de prototypage comporte trois boutons poussoirs qui permettent de forcer des entrées du microcontrôleur (RB0/INT0, RB1/INT1 et RB2/INT2) à l'état bas lors d'un appui (cf. schéma électrique de la carte dans le répertoire du TP).

Deux de ces boutons seront utilisés pour mettre en marche et arrêter le timer0 à chaque appui, en modifiant le bit TMR0ON dans le registre T0CON.

Comme ces événements peuvent arriver à n'importe quel moment, ils seront traités comme des interruptions.

La routine d'interruption devra à chaque appel vérifier quel bouton a été appuyé, et en fonction de cela mettre en marche ou arrêter le timer.

Le programme principal devra configurer le microcontrôleur de manière qu'une interruption soit déclenchée à chaque front descendant détecté sur les pattes d'interruption (RB0 pour INT0, RB1 pour INT1, RB2 pour INT2, configurées dans les registres INTCON, INTCON2 et INTCON3, respectivement pages 109, 110 et 111 de la datasheet du microcontrôleur).

Mettre en place le programme et tester le fonctionnement sur la carte de prototypage.

Exercice 12- Utilisation du convertisseur analogique/numérique

Le but de cet exercice est d'utiliser la barrette de LEDs LD0 à LD7 comme un « bar graph », pour allumer plus ou moins de LEDs selon la position du potentiomètre monté sur la carte de prototypage.

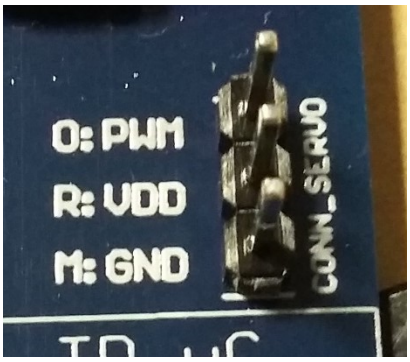
Ce dernier permet d'appliquer sur l'entrée RA5/AN4 du microcontrôleur une tension comprise entre 0 et V_{DD}. Il faudra donc utiliser ici le convertisseur analogique-numérique pour mesurer la tension d'entrée, puis écrire en fonction de celle-ci une valeur particulière sur les ports de sortie branchés sur les LEDs (une LED allumée, puis 2, et ainsi de suite jusqu'à allumer toute la barrette).

- Les ports de sortie branchés sur les LEDs et l'entrée RA5 seront respectivement configurées en sorties et en entrée (registres TRISA et TRISC).
- Le registre ANSEL (page 136 de la datasheet) devra être configuré pour désactiver le buffer d'entrée numérique de la patte RA5, car elle sera utilisée comme entrée analogique.
- Le registre ADCON1 (page 272 de la datasheet) sera configuré de manière que les tensions de référence de l'ADC soient V_{SS} (0V) et V_{DD} (3,5V).
- Le registre ADCON2 (page 273 de la datasheet) sera configuré de manière que le résultat de la conversion soit « left-justified », que la fréquence de fonctionnement du convertisseur f_{AD} soit égale à $f_{OSC}/4$, soit 1 MHz, et que le temps d'acquisition soit égal à 8 T_{AD}.
- Le registre ADCON0 (page 271 de la datasheet) sera utilisé pour sélectionner l'entrée analogique RA5/AN4, mettre en route le convertisseur, lancer une nouvelle conversion, et tester la fin de la conversion en scrutant (méthode dite de « polling ») le bit GO/DONE. Un exemple d'utilisation du convertisseur (configuration + conversion) est montré page 270 de la datasheet (attention, la configuration montrée dans cet exemple est différente de ce qui est demandé ici !).

Tester le fonctionnement de ce programme sur la carte de prototypage.

Exercice 13- Génération d'un signal PWM

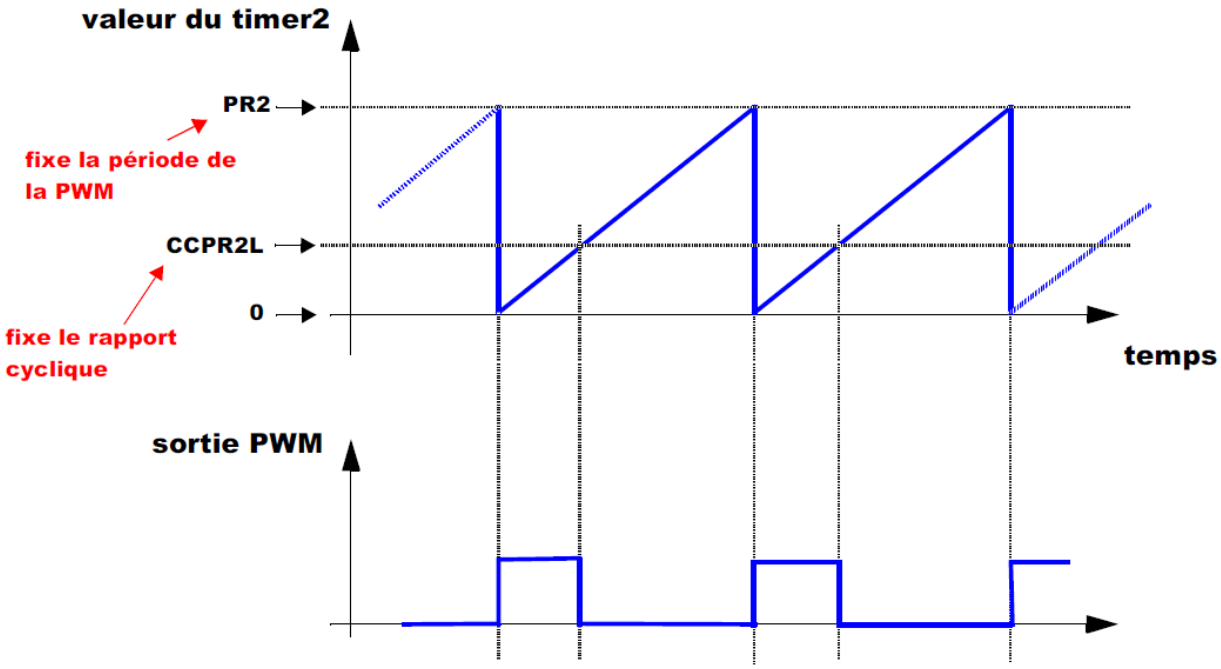
Cet exercice a pour but de générer un signal de rapport cyclique variable (PWM), compris entre 0 et 100%, selon la position du potentiomètre. Dans un premier temps, débrancher le connecteur du servomoteur situé en bas à droite de la carte :



On utilise pour cela la fonction PWM du timer2 (cf. page 149 de la datasheet du PIC18F23K20).

Le fonctionnement simplifié est le suivant :

- le timer2 s'incrémente en continu, de 0 à la valeur définie de PR2. Comme la fréquence de comptage du timer est connue ($f_{OSC}/4$ avec éventuellement un prescaler à définir), la valeur de PR2 définit donc la période totale du signal PWM (PR2 + 1 cycles de comptage).
- Le signal PWM passe à l'état haut lorsque le timer2 revient à 0, et passe à l'état bas lorsque la valeur du timer atteint une valeur seuil CCPR2L comprise entre 0 et PR2. La valeur de CCPR2L définit donc le rapport cyclique du signal PWM, c'est-à-dire la proportion du temps passée à l'état haut.

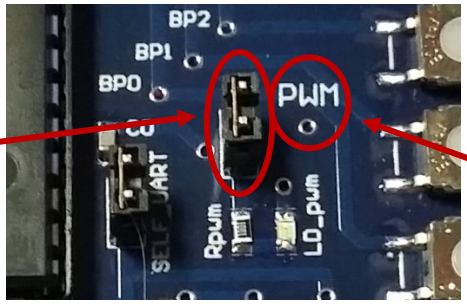


Le programme principal devra :

- configurer le convertisseur analogique/numérique comme précédemment,
- configurer le timer2 pour le mode PWM, de façon à avoir une résolution maximale du rapport cyclique (cf. page 151 de la datasheet), avec la plus grande fréquence de signal possible. Il faudra utiliser les registres T2CON (page 167 de la datasheet), CCP2CON (page 143 de la datasheet), et PR2. Vous devrez aussi configurer la patte RB3 (sortie de la PWM par défaut) comme une sortie.
- convertir en boucle l'entrée analogique RA5/AN4 et modifier en conséquence le registre CCPR2L afin que le rapport cyclique varie de 0 à 100% lorsque le potentiomètre est tourné.

La sortie PWM sera observée à l'oscilloscope, sur la patte RB3. Un point de mesure appelé « PWM » est prévu sur la carte de prototypage à cet effet. De plus, il est possible de connecter une LED appelée LD_pwm au signal de sortie grâce à un jumper, ce qui permet de visualiser la valeur moyenne du signal PWM et donne donc un aperçu de la valeur du rapport cyclique.

Jumper d'activation de la LED de vérification LD_pwm (Jumper enlevé = LED déconnectée)



Point de mesure pour la PWM (connexion à RB3)

Exercice 14- Utilisation d'un servomoteur

La carte de prototypage comporte un servomoteur qui peut être contrôlé à l'aide d'un signal PWM. La datasheet de ce servomoteur peut être trouvée dans le répertoire du TP. La connexion à ce servomoteur doit tout d'abord être rétablie en rebranchant le connecteur en bas à droite de la carte :

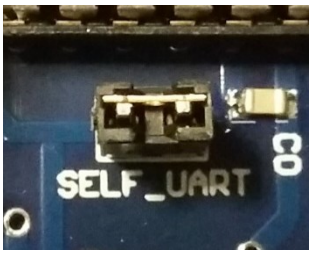
L'objectif de cet exercice est de contrôler la rotation de ce servomoteur grâce au potentiomètre de manière semblable à l'exercice précédent. Le fonctionnement du servomoteur impose quelques contraintes sur le signal PWM à générer :

- PR2 doit être configuré pour que la période du signal PWM soit de 3ms.
- La position extrême du servomoteur 0° est atteinte pour un temps à l'état haut de 0.5ms (soit un rapport cyclique de 17% environ), et l'autre position extrême 180° est atteinte pour un temps à l'état haut de 2.5ms (soit un rapport cyclique de 83% environ). Il faut donc convertir en boucle l'entrée analogique RA5/AN4 et modifier en conséquence le registre CCPR2L afin que le rapport cyclique varie de 17 à 83% lorsque le potentiomètre est tourné.

Tester le programme sur la carte de prototypage, en mesurant la sortie PWM à l'oscilloscope si besoin.

Exercice 15- Utilisation d'une communication série UART

Les microcontrôleurs intègrent des périphériques de communication série pour échanger des informations avec d'autres composants. Le principe est de transmettre un ou plusieurs octets de données en utilisant le minimum de connexions en utilisant des protocoles standardisés (UART, I2C, SPI). Dans cet exercice, on se propose de configurer un lien série UART (appelé EUSART dans la datasheet du microcontrôleur, page 239). L'objectif est d'envoyer un octet de données via la patte de sortie TX du microcontrôleur (RC6), qui pourra être reçu par la patte de réception RX du microcontrôleur (RC7). Pour cela, on pourra connecter le jumper appelé SELF_UART reliant ces deux pattes :



La plupart des composants communiquant en UART émettent leurs données de manière asynchrone, c'est-à-dire sans utiliser de signal d'horloge. La fréquence du signal (débit de données ou *baudrate*, exprimé en bauds) doit donc être configurée de la même manière en émission et en réception pour que les deux parties soient synchronisées. Typiquement, on adoptera un débit à 9600 bauds ici.

Dans cet exercice, il faut donc :

- Régler le baudrate à 9600 bauds en utilisant les registres BAUDCON (page 248 de la datasheet) et le bit 2 de TXSTA (page 246 de la datasheet), ainsi que la paire de registres SPBRGH et SPBRG, dont le fonctionnement est décrit aux pages 249-252 de la datasheet. Il faut aussi préciser dans ces registres que l'état par défaut du périphérique est l'état haut et que la donnée n'est pas inversée par défaut, et enfin désactiver la détection automatique du baudrate (Auto-Baud Detect).
- Configurer le reste du registre de transmission TXSTA, dans lequel il faut préciser que l'on utilise une transmission asynchrone sur 8 bits, et enfin activer le périphérique en transmission.
- Configurer le registre de réception RXSTA (page 247 de la datasheet). Il faut activer le périphérique port série, ainsi que le récepteur, en mode 8 bits également.

Une fois le périphérique configuré en entier, il est possible d'envoyer un octet en le déplaçant dans le registre TXREG. Le module va automatiquement transmettre l'octet sur le port série en suivant la configuration mise en place.

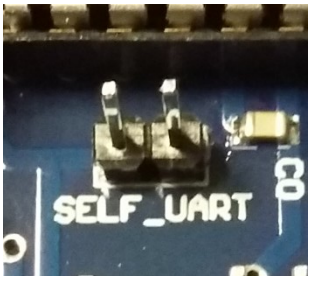
Pour lire l'octet reçu, il faut visualiser le registre RCREG. Le bit RCIF du registre PIR1 permet de plus d'indiquer qu'un octet vient d'être reçu par le port série, et peut donc servir de condition pour une boucle d'attente de réception ou d'interruption. Ce flag est désactivé en lisant le contenu du registre RCREG (copie dans WREG ou variable par exemple).

Valider le fonctionnement de votre programme en mode debug.

Exercice 16- Communication UART avec un autre périphérique

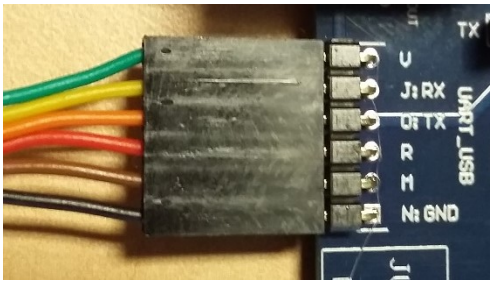
Dans cet exercice, il est demandé de configurer puis d'effectuer une communication en suivant le protocole UART en émission et en réception avec un dispositif externe et d'effectuer des actions sur la carte de prototypage en conséquence. Le dispositif externe pourra être soit connecté via Bluetooth à la carte de prototypage, soit via USB.

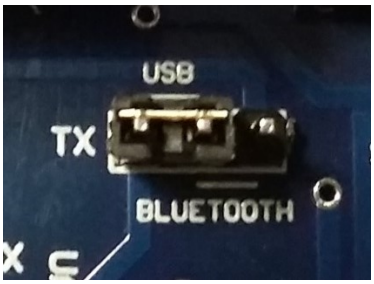
Dans les deux cas, il faudra débrancher le jumper SELF_UART afin de déconnecter RX et TX :



1. *Communication via USB*

Dans ce cas de figure, un câble de communication UART-USB sera utilisé pour relier la carte de prototypage à un terminal sur ordinateur. Le terminal utilisé est un logiciel tiers laissé au choix (des terminaux gratuits supportant le protocole UART/RS-232 comme Termite ou RealTerm peuvent être utilisés). La connexion du câble sur la carte se fera par le connecteur appelé UART_USB (en bas de la carte) selon l'image de gauche ci-dessous, et le jumper appelé TX sera positionné comme sur la photo de droite :



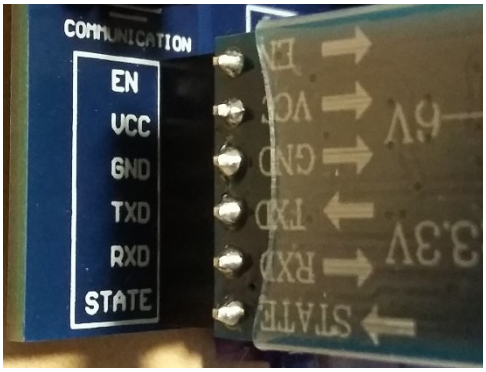


En utilisant les mêmes réglages que dans l'exercice précédent (liaison asynchrone sur 8 bits avec un baudrate de 9600 bauds), utiliser le terminal pour transmettre un octet de données vers le microcontrôleur et faire tourner le servomoteur d'une position extrême à l'autre. Cette action sera complétée par une transmission d'un octet de confirmation du microcontrôleur vers le terminal.

Valider le fonctionnement de votre programme en mode debug, puis en mode continu.

2. Communication via Bluetooth

Pour mettre en place une communication Bluetooth, on utilisera un module HC-05 qui pourra s'appairer à un appareil mobile (smartphone ou PC portable). Un terminal supportant le protocole UART/RS-232 et les communications via Bluetooth devra alors être installé sur l'appareil distant (par exemple une application comme Serial Bluetooth Terminal est disponible sur Google Play). La connexion du module Bluetooth sur la carte se fera par le connecteur appelé HC-05 (dans le coin en bas à gauche de la carte) selon l'image de gauche ci-dessous, et le jumper appelé TX sera positionné comme sur la photo de droite :



Le nom du module utilisé pour l'appairage est indiqué sur la pastille autocollante (« TP… » de manière générale), et le code d'appairage est « 1234 ».

En utilisant les mêmes réglages que dans l'exercice précédent (liaison asynchrone sur 8 bits avec un baudrate de 9600 bauds), utiliser le terminal pour transmettre un octet de données vers le microcontrôleur et faire tourner le servomoteur d'une position extrême à l'autre. Cette action sera complétée par une transmission d'un octet de confirmation du microcontrôleur vers le terminal.

Valider le fonctionnement de votre programme en mode debug, puis en mode continu.

Exercice 17- Station domotique

Dans ce dernier exercice, il est demandé de compiler toutes les notions vues précédemment pour réaliser une station de commande pour une installation domotique. Les actionneurs présents sur la carte de prototypage permettront de simuler un volet roulant (par le biais du servomoteur) et un variateur lumineux (par le biais de la barre de LEDs). Les différentes actions pourront se déclencher par un appui sur les boutons (montée et descente du volet), ainsi que par une rotation du potentiomètre (allumage progressif de la barre de LEDs de la même manière que dans l'exercice 12), mais aussi par des commandes à distance en utilisant la communication série UART du microcontrôleur (avec interface Bluetooth ou USB au choix).

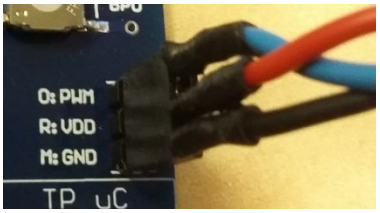
Vous pourrez également intégrer des fonctionnalités supplémentaires à votre convenance, toute idée pertinente est la bienvenue !

Valider le fonctionnement de votre programme en mode continu.

Exercice 18- LEDs adressables

Cet exercice supplémentaire permet de découvrir la mise en œuvre de LEDs RGB (Red-Green-Blue) programmables de référence SK6812 utilisées dans de nombreux dispositifs d'affichage industriels. L'avantage de ces LEDs RGB est qu'elles peuvent se connecter les unes aux autres en série et être commandées par une seule sortie du microcontrôleur. Pour cela, un prototype de 3 LEDs RGB de démonstration pourra être demandé aux moniteurs de TP. Ce dernier pourra être branché à la place du servomoteur sur le connecteur dédié :

- Fil bleu sur la sortie PWM
- Fil rouge sur le VDD
- Fil noir sur le GND

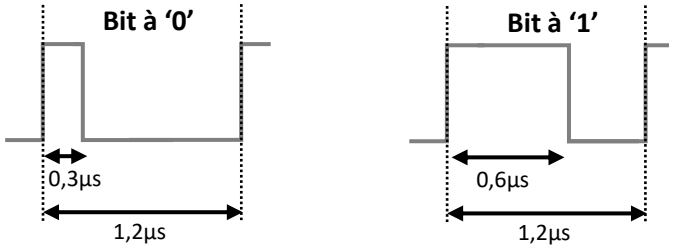


Dans chaque boîtier de LED RGB se trouvent 3 LEDs individuelles (une rouge, un verte et une bleue), et un système gérant leur intensité. En mélangeant les couleurs de chacune des LEDs individuelles, il est possible d'obtenir une multitude de couleurs globales pour la LED RGB. L'intensité des couleurs des LEDs individuelles est codée sur 8 bits, de la valeur 0 (LED éteinte) à la valeur 255 (LED complètement allumée). Il faut donc commander chaque LED RGB sur 24 bits, ce qui donne 72 bits pour envoyer une commande aux 3 LEDs RGB. Il n'y a besoin d'envoyer tous les bits de commande qu'une seule fois pour mettre à jour l'ensemble des LEDs RGB, sans le répéter en boucle.

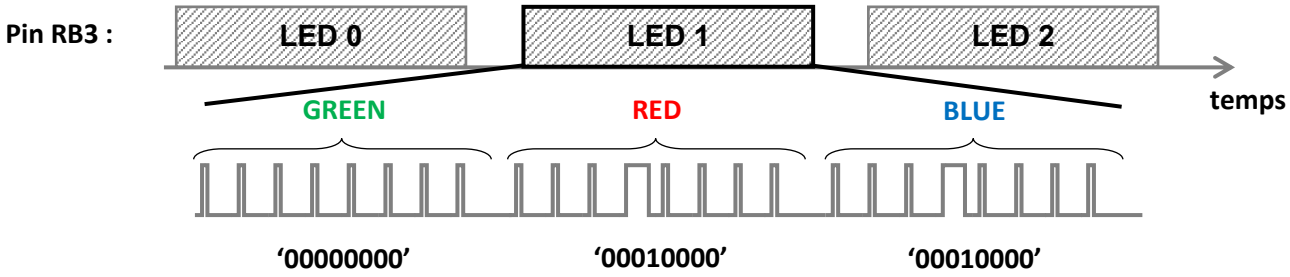
Ces 72 bits sont envoyés en série (c'est-à-dire via un seul signal, étalés dans le temps) sur la patte RB3 du microcontrôleur selon le protocole décrit dans le paragraphe suivant. **Ce n'est pas nécessairement un signal PWM !**

Chaque bit est codé par une impulsion, c'est-à-dire une succession d'un état haut puis d'un état bas du signal de commande pendant un certain temps. La somme des temps à l'état haut et à l'état bas reste identique pour coder un bit à '0' ou à '1', la différence se fait sur la répartition entre état haut et état bas, comme le montre le schéma suivant :

ATTENTION : respecter le timing le plus fidèlement possible !



Le signal de commande de la série de LEDs RGB a donc l'allure suivante :



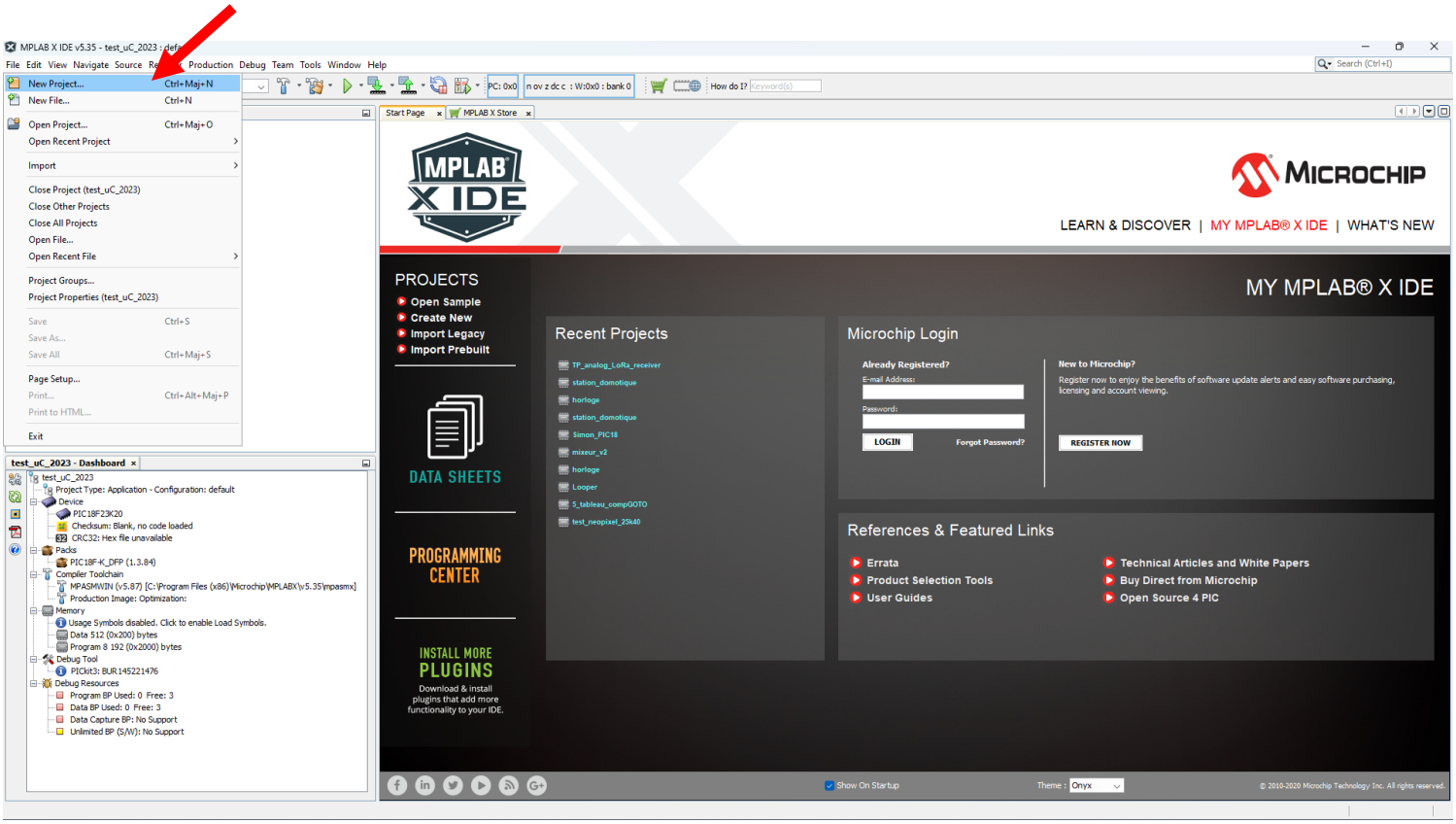
Dans l'exemple ci-dessus, la LED n°1 s'allume en violet (un peu de rouge + un peu de bleu).

Tester et valider la mise en œuvre de ces LEDs dans un programme à valider en mode continu.

Annexes

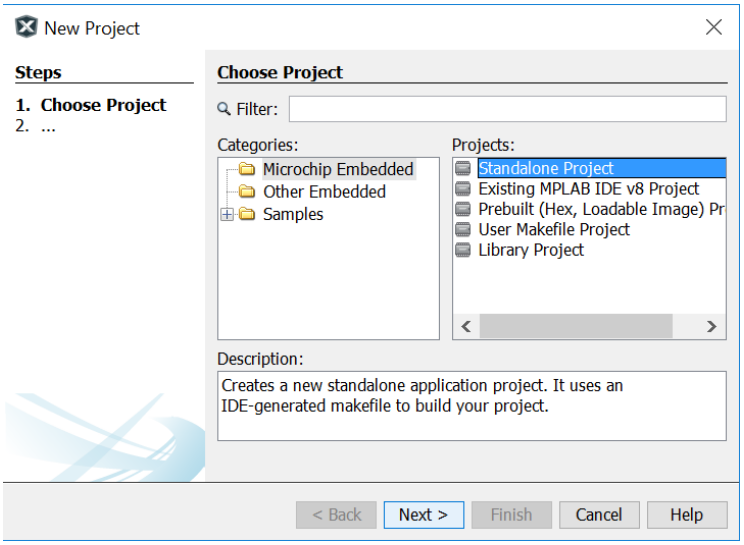
1 – Création d'un projet sous MPLAB X

En démarrant MPLABX, commencer par créer un nouveau projet.
File → New Project...



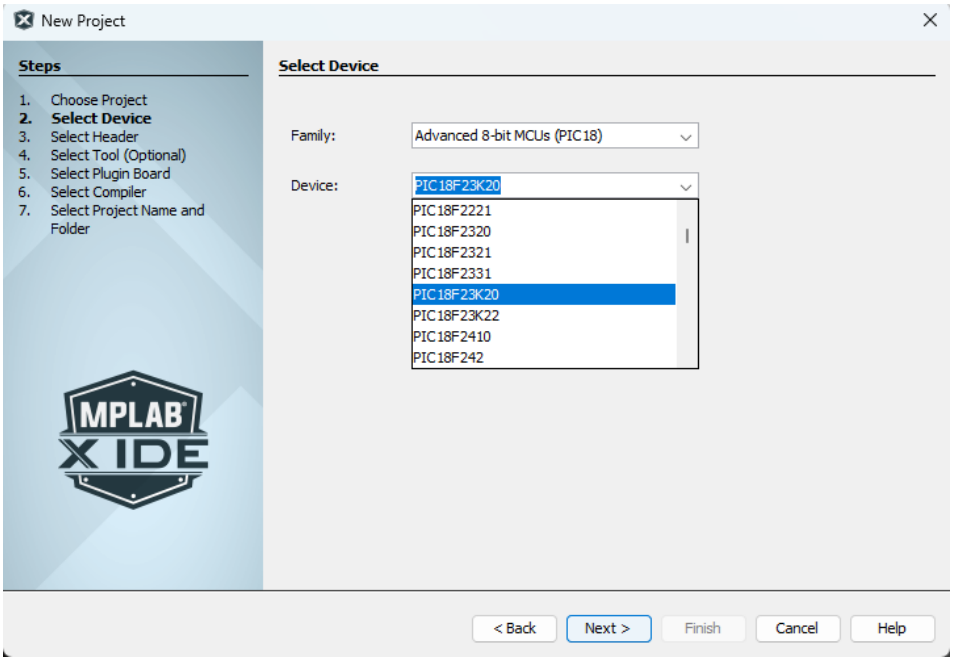
Dans l'étape ***Choose Project***, garder les choix par défaut
(Categories : *Microchip Embedded* – Projects : *Standalone Project*)

Puis sélectionner ***Next >***.



Dans l'étape ***Select Device***, sélectionner
Family : ***Advanced 8-bit MCUs (PIC18)***
Device : ***PIC18F23K20*** (ou toute autre référence, en fonction du composant qui sera utilisé)

Puis sélectionner ***Next >***.



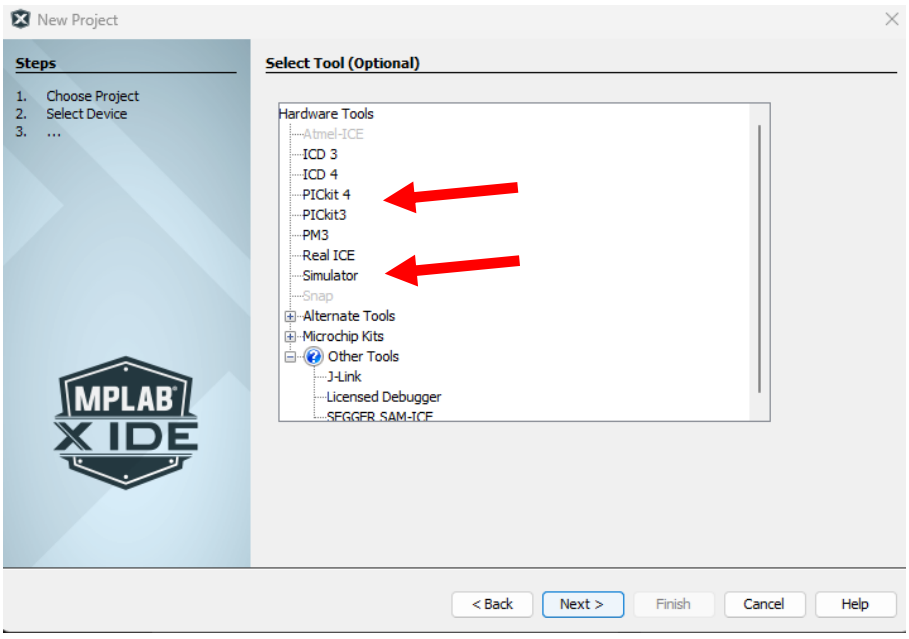
Dans l'étape ***Select Tool***, sélectionner :

- ***Simulator*** pour débugger un programme uniquement en simulation

  ou

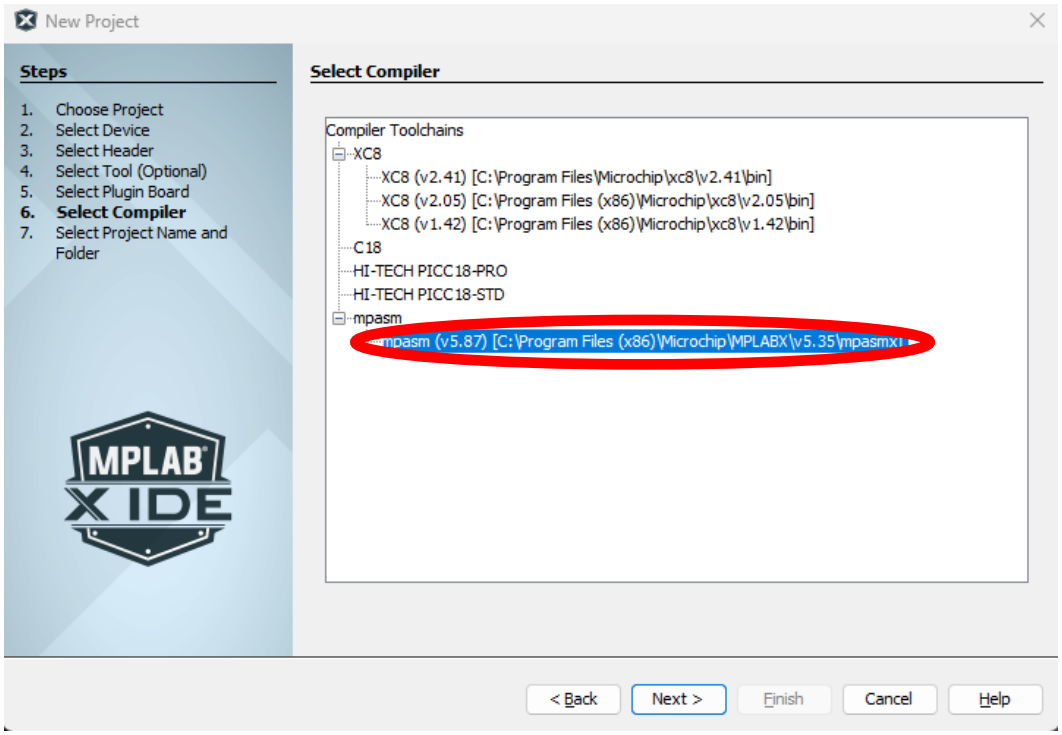
- ***PICkit3/PICkit4*** pour débugger et faire tourner un programme sur un microcontrôleur réel (une carte de prototypage doit dans ce cas être connectée au PC)

Puis sélectionner ***Next >***.



Dans l'étape ***Select Compiler***, sélectionner
mpasm (v5.87 ou ultérieure) .....
(Ce qui est la seule possibilité si aucun compilateur C n'a été installé pour MPLABX)

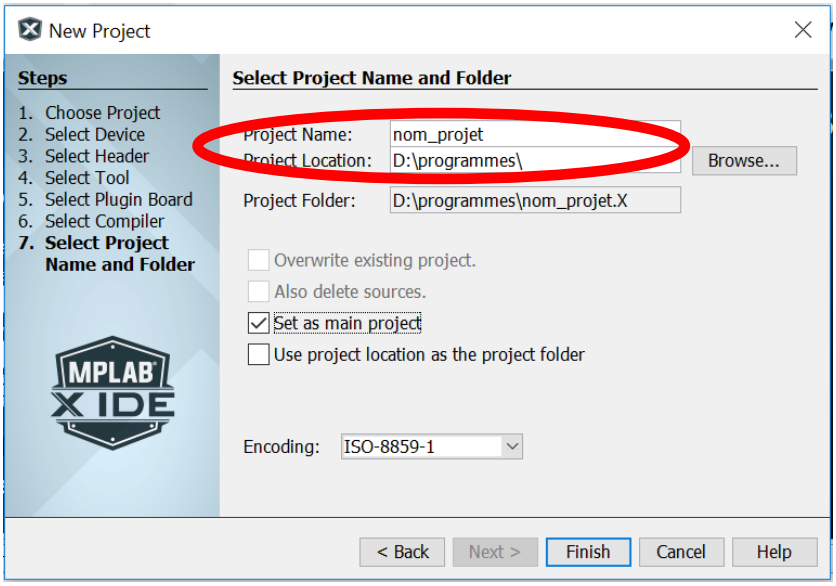
Puis sélectionner ***Next >***.



Dans l'étape ***Select Project Name and Folder***, indiquer le nom du projet ainsi que son emplacement (un sous-répertoire xxxxxx.X sera automatiquement créé).

Puis sélectionner ***Finish***.

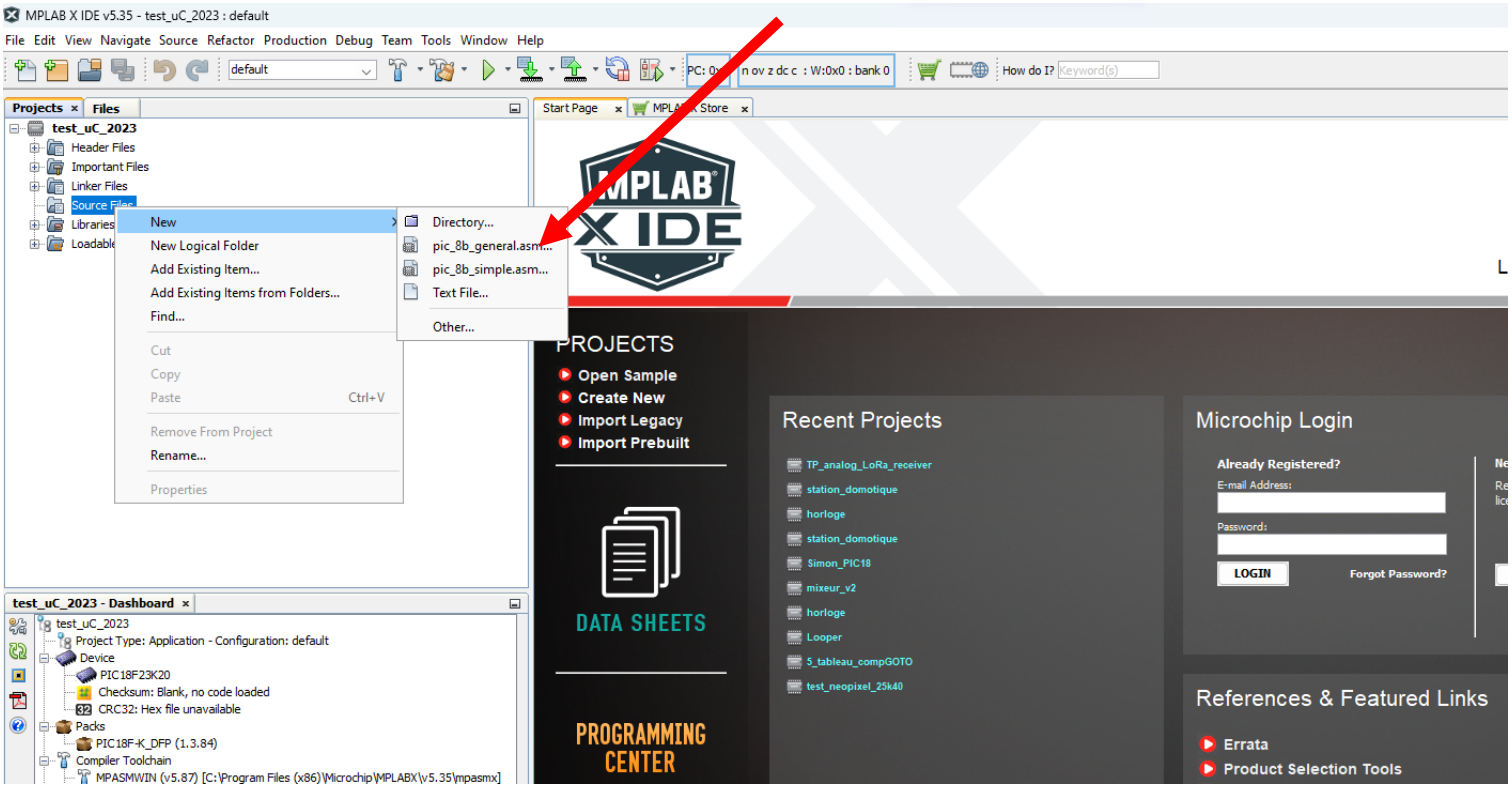
Remarque importante : Pour les noms de projets, répertoires ou fichiers, n'utiliser que les caractères alpha-numériques (pas d'espace, d'accent, de &, #, etc.).



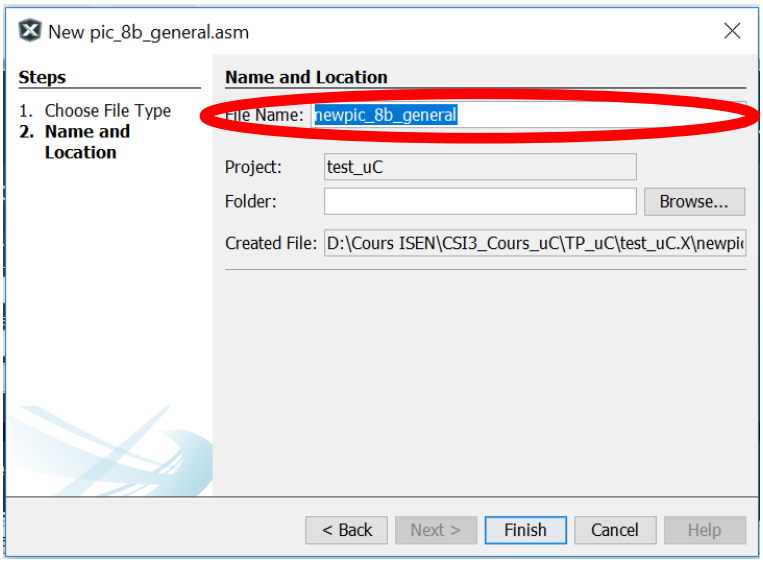
2 – Ajout d'un fichier source (programme assembleur) à un projet

L'onglet ***Projects*** de la fenêtre principale indique le projet actuellement ouvert : il faut maintenant lui associer un fichier qui permette d'écrire le programme à tester (fichier source).

Par un clic-droit sur ***Source Files***, sélectionner ***New → pic_8b_general.asm...***



Dans la fenêtre suivante, spécifier un nom de fichier, puis sélectionner ***Finish***.



Le fichier créé par défaut est divisé en plusieurs sections indiquées par des commentaires et permettant de placer au bon endroit les différentes parties du programme (variables, routines d'interruptions, etc.).

<u>Ne pas effacer ces commentaires pour garder l'organisation du programme !</u>

Il contient un également un programme basique qui écrit la valeur 0x55 dans le registre de travail W (*MOVLW 0x55*), puis boucle indéfiniment sur la même instruction (*GOTO $*).
Il « ne reste plus » qu'à éditer le fichier pour écrire le programme souhaité.

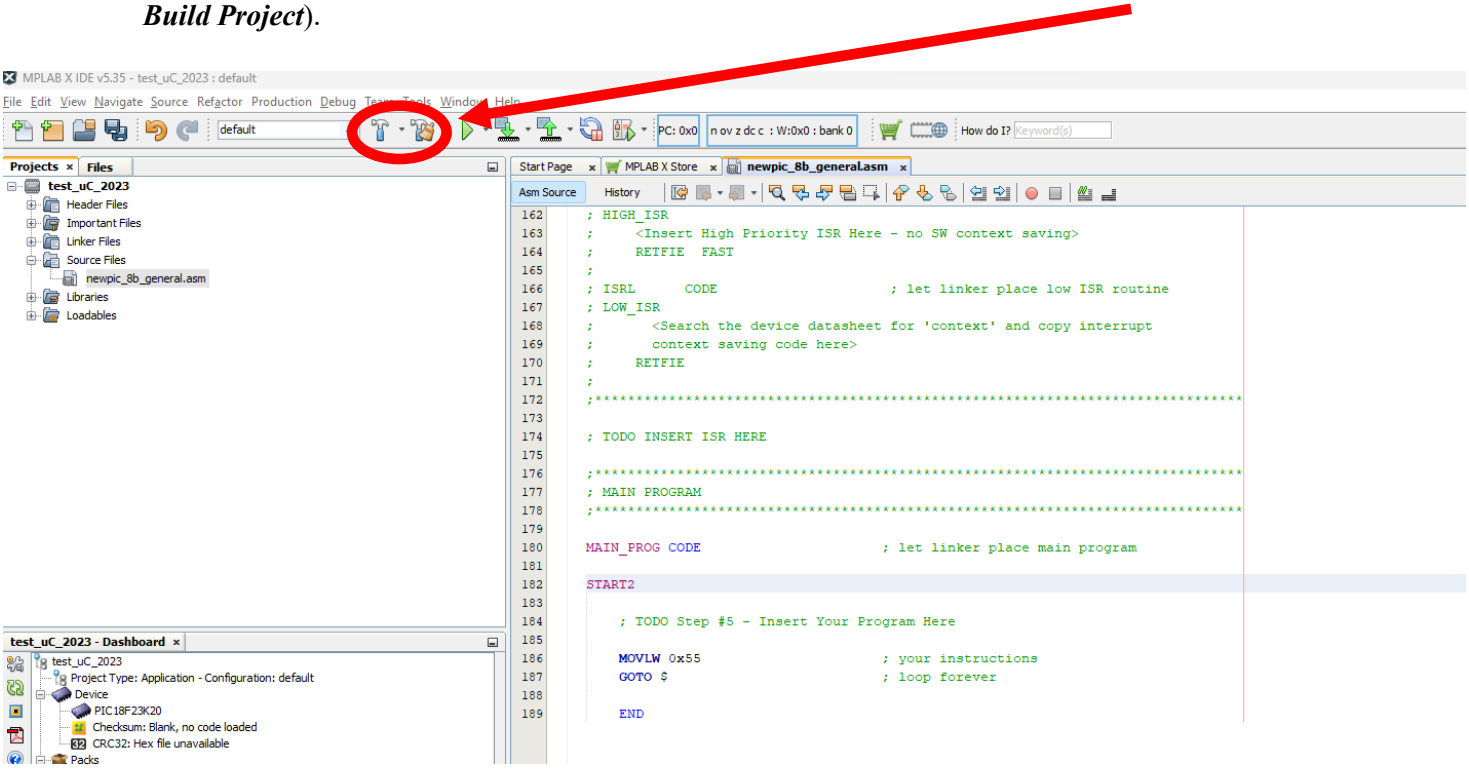
Remarques importantes :

- Au début du programme (un espace dans le squelette de programme appelé « TODO Step #1 – Processor Inclusion » est prévu pour), inclure la ligne de commande : ***#include "p18f23k20.inc"***
(Ceci permet au compilateur d'interpréter les noms de registres du microcontrôleur utilisé, tels que WREG, STATUS, PORTA, etc.)

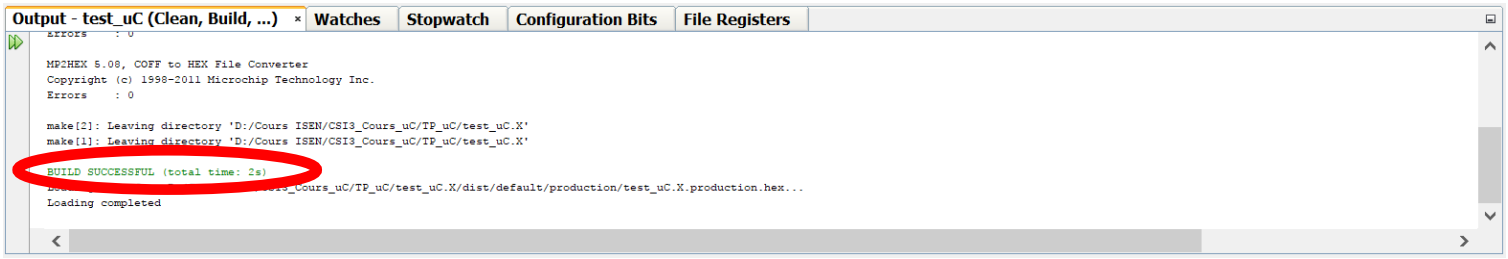
- Changer le nom *START* utilisé par défaut comme label du programme principal (car ce nom est déjà employé pour une autre raison dans le fichier « p18F23K20.inc »). Vous pouvez utiliser n'importe quel autre nom valide (START2, debut, toto, etc.). Remplacer toutes les occurrences de *START* en les trouvant avec ***CTRL+F***.

Après édition du fichier, le sauvegarder (***File → Save*** ou ***Save All***).

Puis le compiler pour traduire le programme en langage machine (***Run → Build Project*** ou ***Clean and Build Project***).



S'il n'y a pas d'erreur de syntaxe dans le programme, le message BUILD SUCCESSFUL doit apparaître dans la fenêtre « **Output** ».



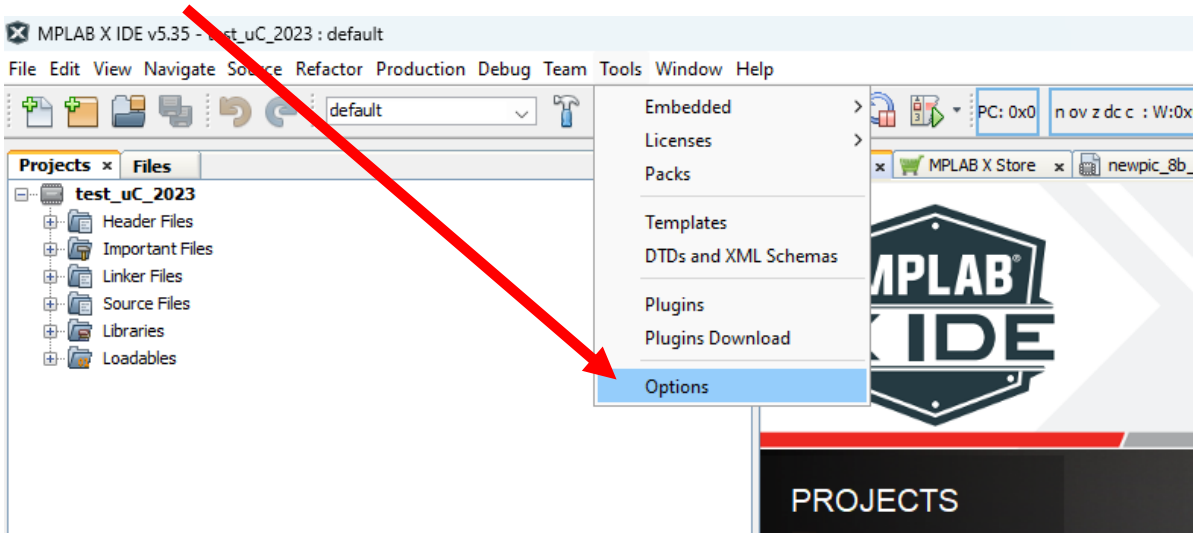
3 – Utilisation du « debugger » : Simulation d'un programme

3.1 Lancement du debugger

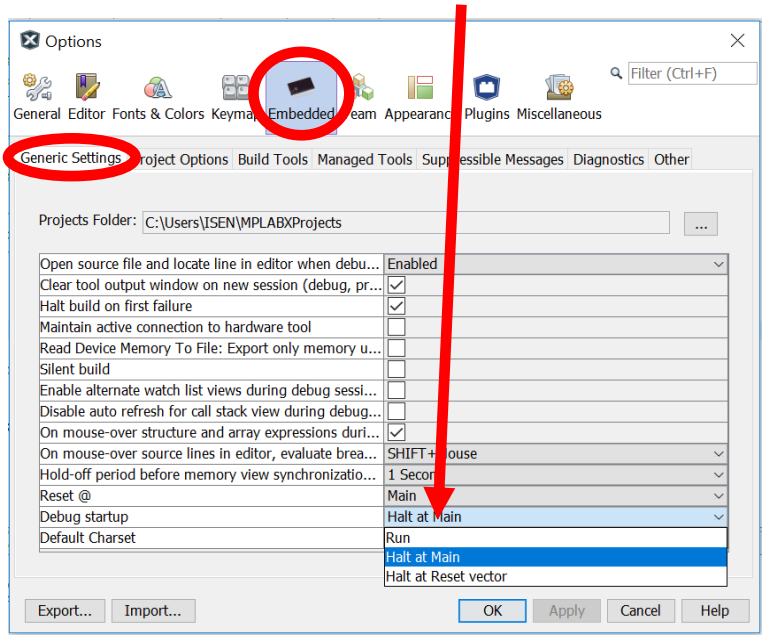
Après compilation du programme, il est possible de le simuler et de visualiser le contenu des divers registres du microcontrôleur en lançant le debugger**.**

Dans un premier temps, si l'on souhaite lancer la simulation en mode pas-à-pas pour étudier le comportement du programme, il faut éviter que le simulateur n'exécute tout le programme au démarrage. Une option de MPALB doit pour cela être modifiée.

Tools → Options

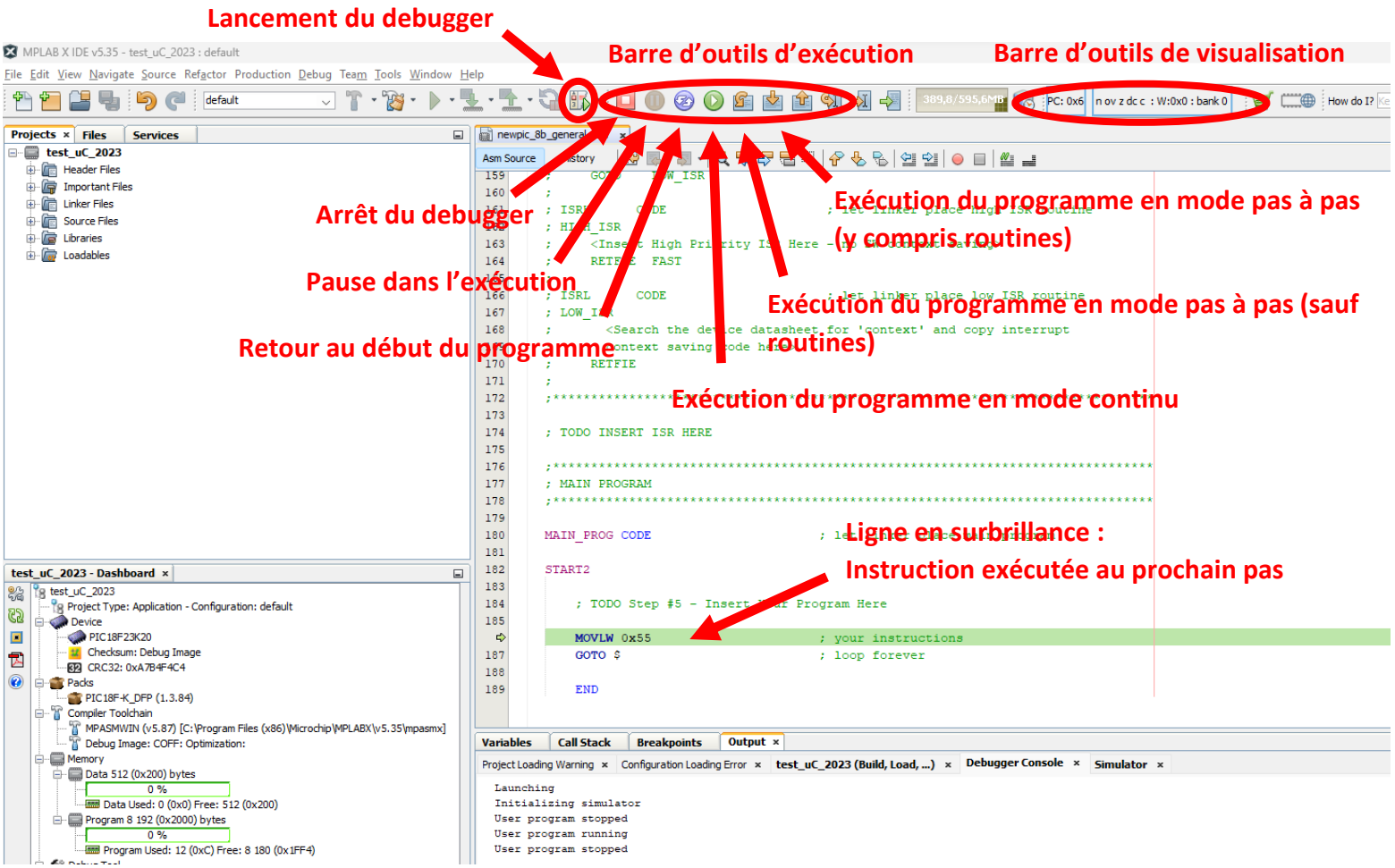


Embedded → Generic Settings → Debug Startup : Halt at Main.



Le debugger se lance ensuite avec ***Debug → Debug Main Project.***

La fenêtre principale permet ensuite de tester le programme en mode pas-à-pas (Step Into) ou de lancer son exécution (Continue), tout en visualisant le contenu des divers registres.

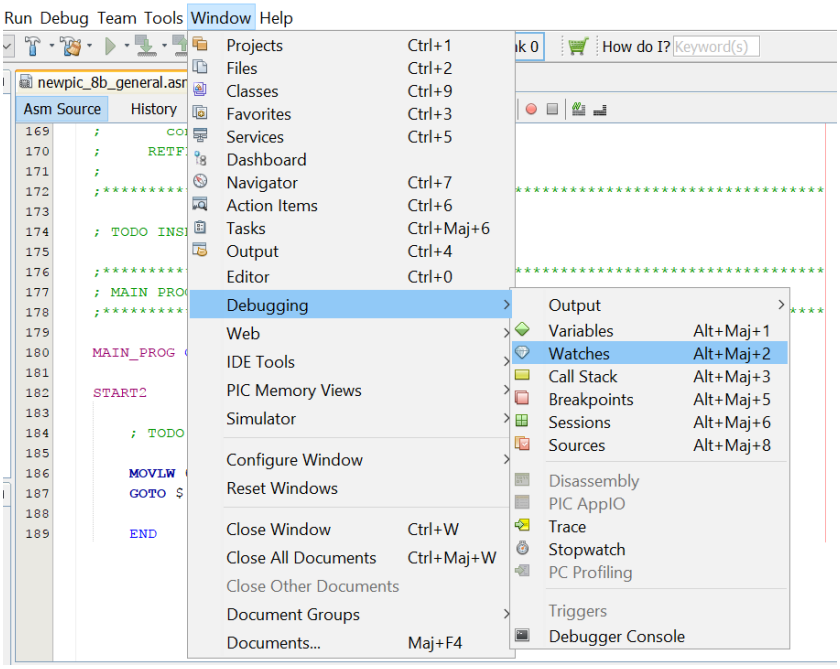


La barre d'outils en haut de la fenêtre principale permet de visualiser :

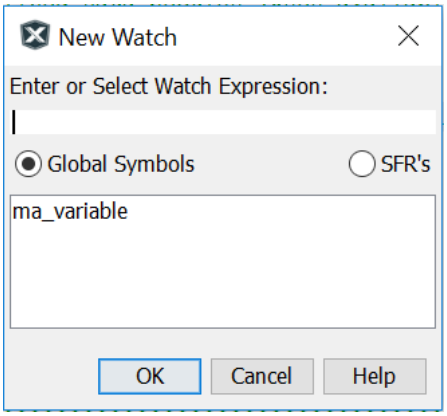
- le compteur ordinal ou Program Counter (adresse de l'instruction qui sera exécutée au cycle CPU suivant),
- le registre d'état ou Status Register (contient les flags qui renseignent sur le résultat de la dernière opération - une lettre minuscule signifie un flag à 0, une lettre majuscule signifie un flag à 1),
- l'accumulateur ou Working Register,
- le numéro de banque courante.

3.2 Visualisation du contenu des registres (contenu de la RAM)

Ouvrir la fenêtre « Watches » par la commande ***Window → Debugging → Watches***



Un clic-droit dans la fenêtre « Watches » permet d'ouvrir la fenêtre « New Watch ».

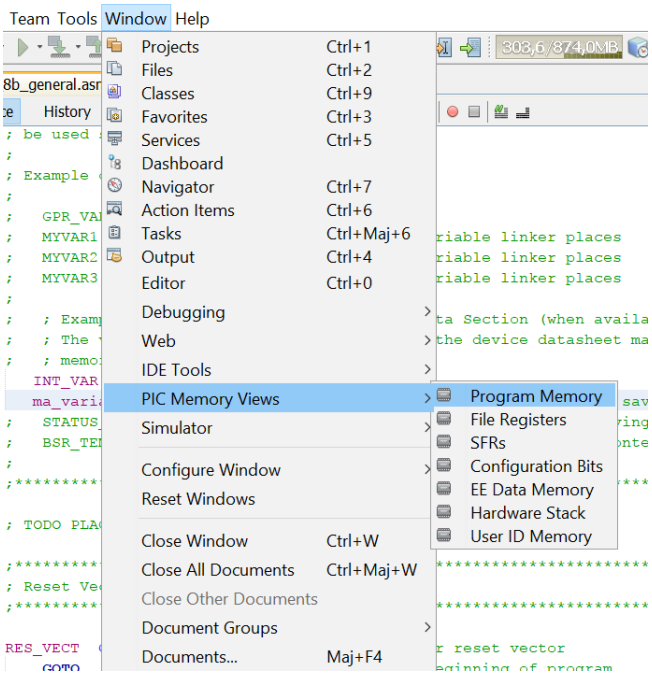


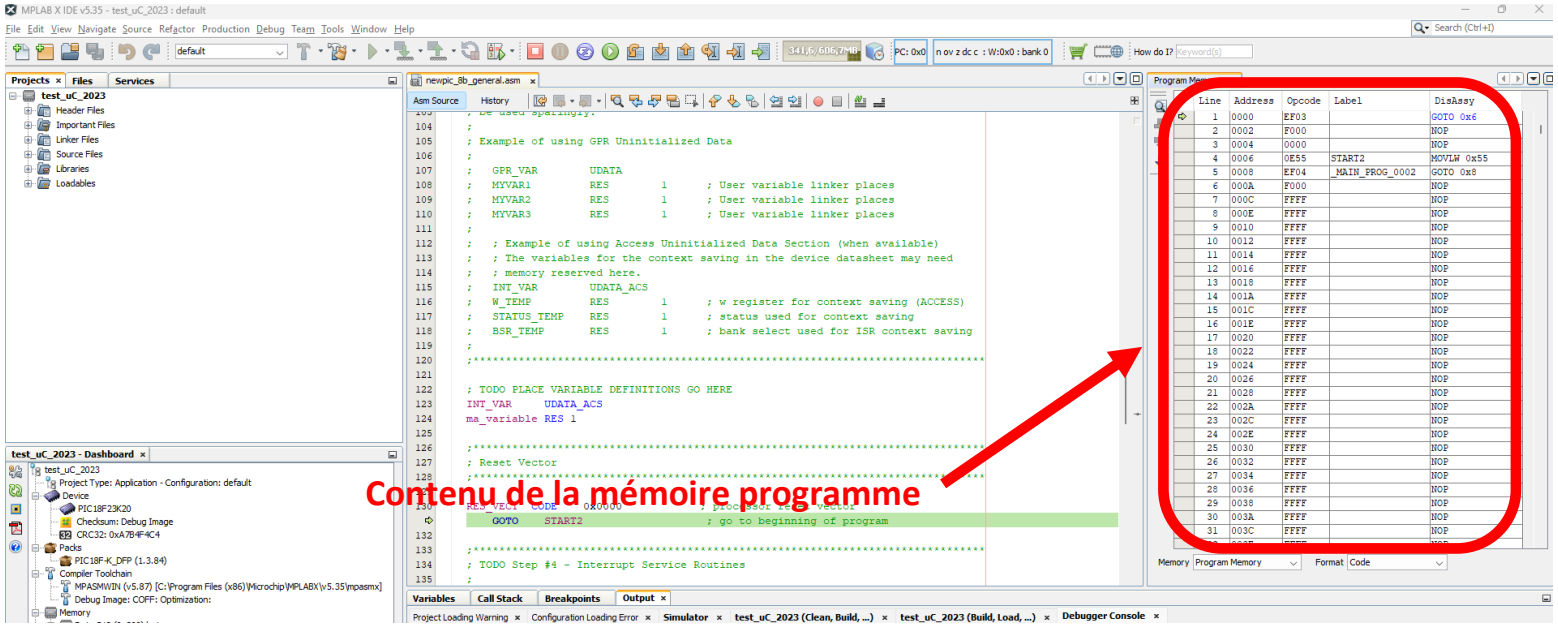
« Global Symbols » permet de visualiser les variables définies par l'utilisateur.

« SFR » (Special Function Register) permet de visualiser les registres spécifiques au microcontrôleur, qui sont liés aux périphériques et au matériel (ports d'entrée/sortie, timers, ADC, etc.). Se reporter à la datasheet du composant pour connaître la fonction de ces registres.

3.3 Visualisation de la mémoire programme (contenu de la Flash)

Ouvrir la fenêtre « Program Memory » par la commande ***Window → PIC Memory Views → Program Memory***





La fenêtre donne les indications suivantes :

- Line : numéro d'instruction,
- Address : emplacement de l'instruction dans la mémoire programme,
- Opcode : instruction en langage machine (sur 16 bits),
- Label : label éventuel donné à une instruction par l'utilisateur,
- DisAssy (Disassembly) : instruction en langage assembleur.

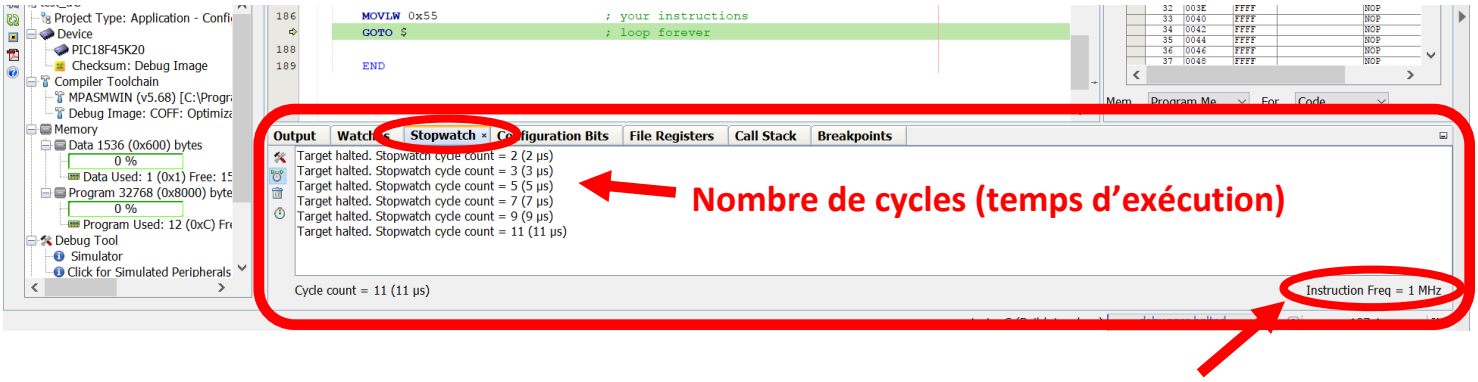
3.4 Visualisation du nombre de cycles (temps d'exécution)

Il est possible de visualiser le nombre de cycles nécessaire pour exécuter une instruction, une routine, ou une partie de programme.

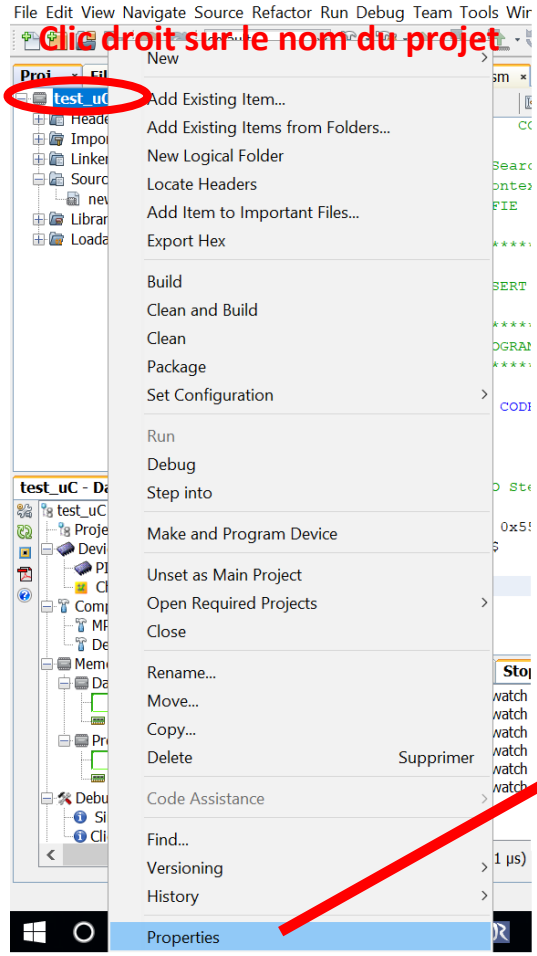
Ouvrir dans un premier temps la fenêtre « Stopwatch » :

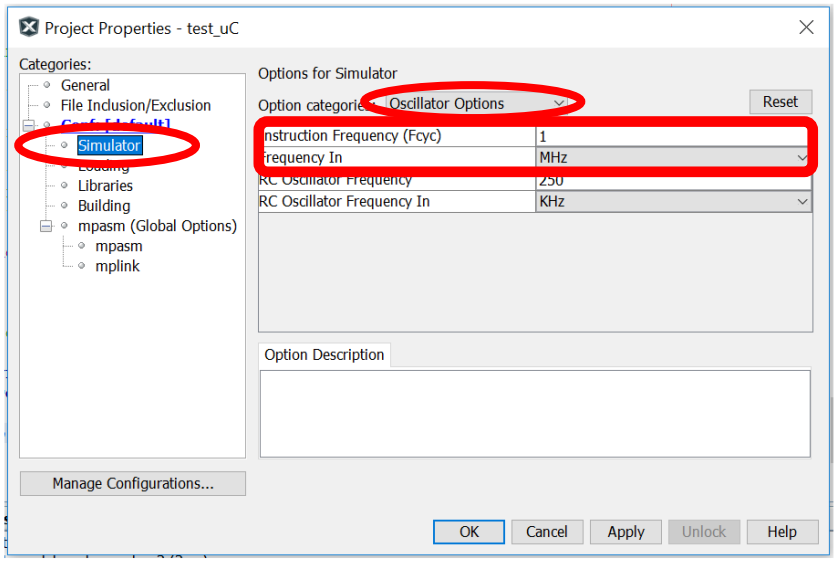
Window → Debugging → Stopwatch

En lançant la simulation, que ce soit en mode pas-à-pas ou en mode continu, le nombre de cycles s'affiche dans la fenêtre « Stopwatch ».



Remarque : Le simulateur prend par défaut 1 µs comme temps d'exécution d'une instruction (soit une fréquence d'oscillateur de 4 MHz). Il est possible de modifier ce paramètre de la manière suivante :



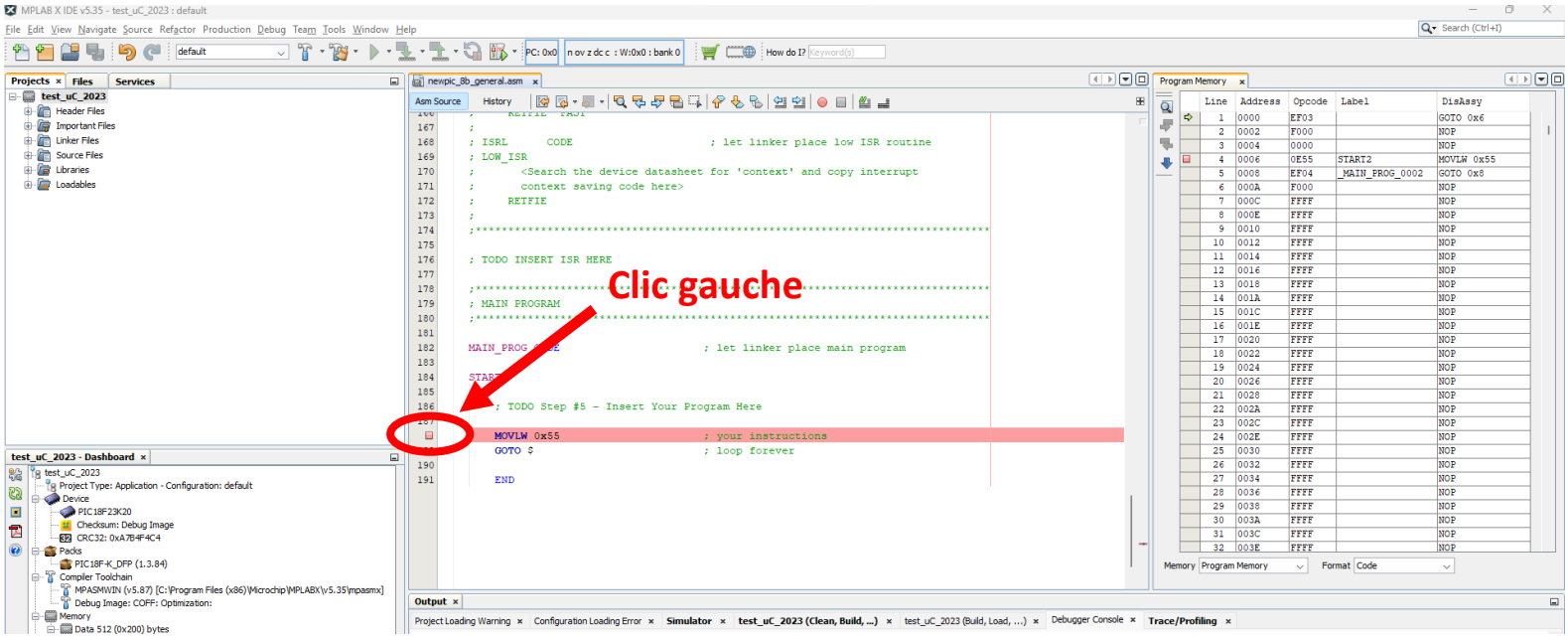


Remarque (2) : La simulation en mode pas-à-pas peut rapidement s'avérer fastidieuse lorsque les programmes sont conséquents. Il peut dans ce cas être utile de simuler directement toute une partie de programme, jusqu'à une instruction donnée.

Il suffit pour cela de placer un point d'arrêt (breakpoint) sur l'instruction concernée, en cliquant dans la marge, puis de lancer la simulation en mode continu (bouton « lecture », cf. 3.1).

L'instruction où est placé le point d'arrêt est surlignée en rouge.

Un nouveau clic dans la marge supprime de la même manière le point d'arrêt.

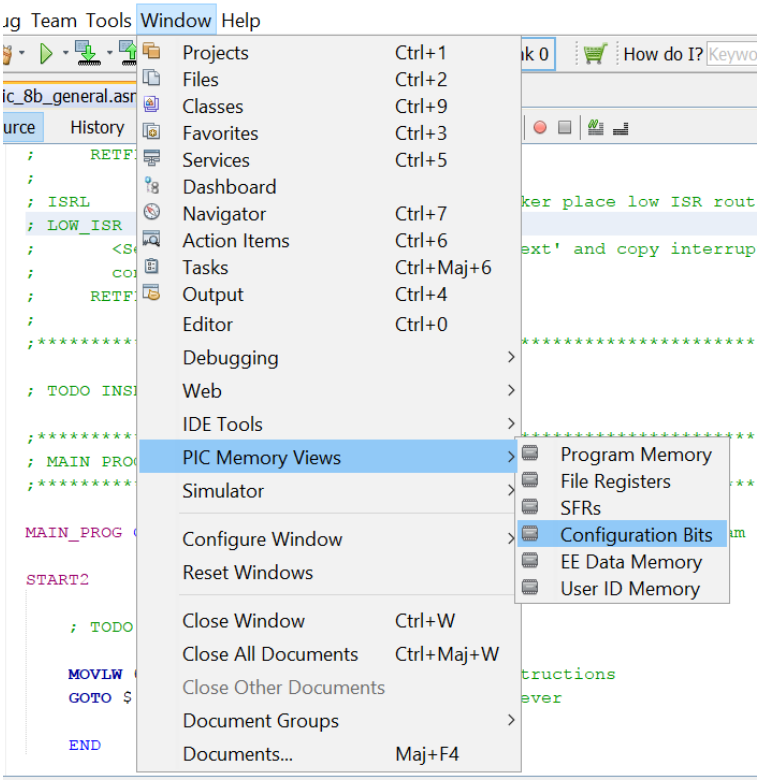


4 – Édition des bits de configuration (Configuration Bits)

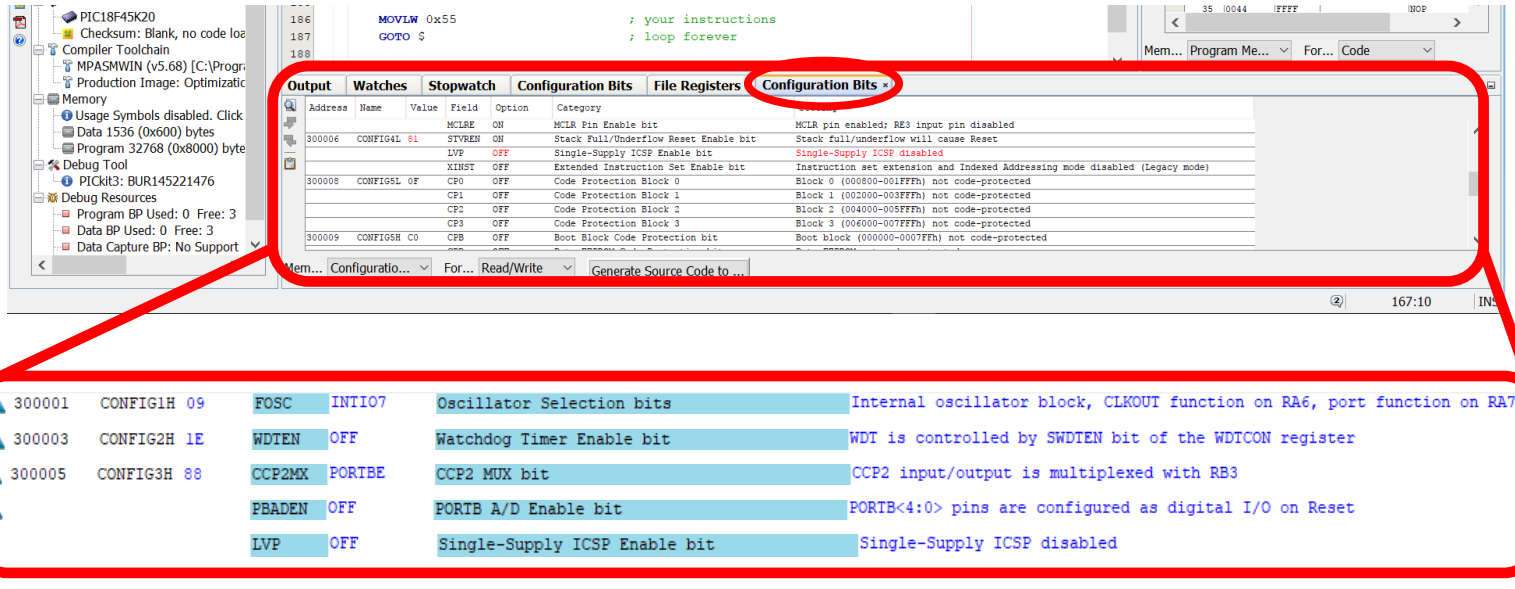
Avant de lancer le debugger sur la carte de prototypage, il est nécessaire d'éditer les bits de configuration qui vont spécifier les options matérielles du microcontrôleur.

Ouvrir la fenêtre « Configuration Bits ».

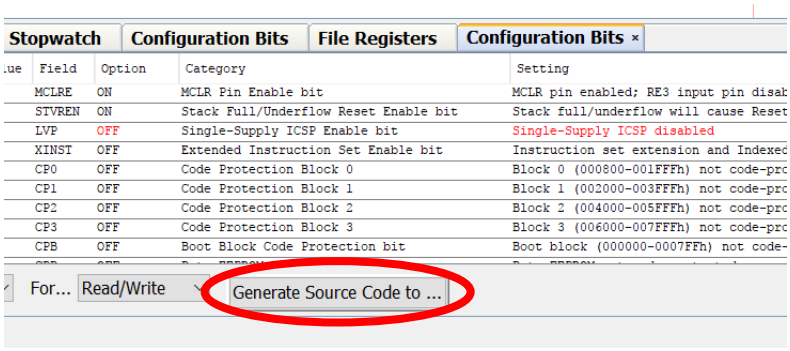
Window → PIC Memory Views → Configuration Bits



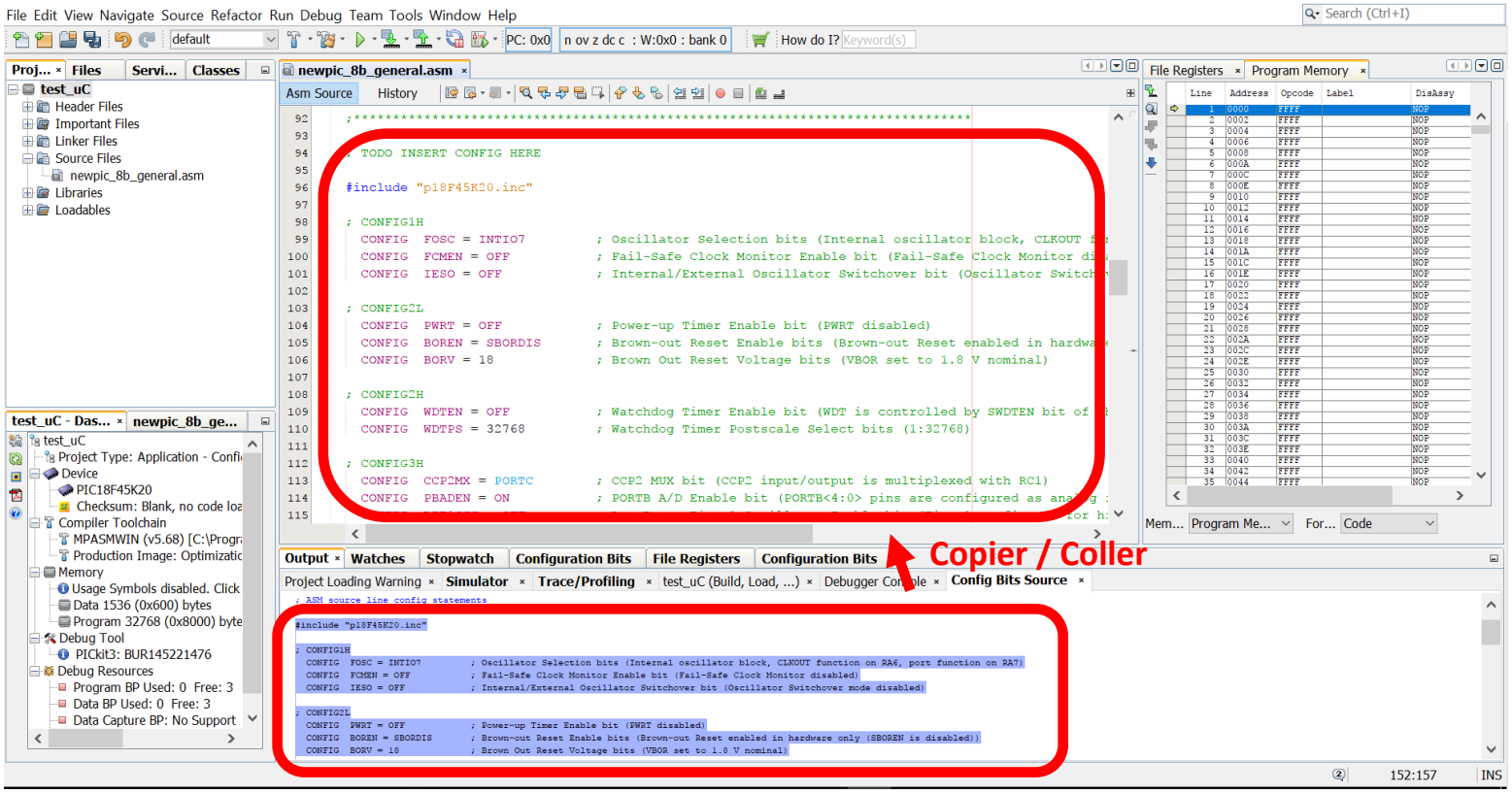
Dans la fenêtre « Configuration Bits », un certain nombre de bits de configuration doivent être modifiés :



Sélectionner ensuite « Generate Source Code to Output ».



Copier-coller le code ainsi généré au début du programme (section Step#2 dans le squelette de programme).



Remarque : Dans le texte généré se trouve la ligne de commande ***#include "p18F23K20.inc"***. Si vous l'avez déjà placée précédemment la deuxième occurrence copiée va générer des problèmes à la compilation. Dans ce cas, il suffit de supprimer cette ligne du programme.

Le debugger peut ensuite être lancé comme précédemment, mais le programme sera réellement exécuté sur le microcontrôleur, et tous les registres observés dans l'IDE (données ou programme) refléteront le contenu réel du microcontrôleur.

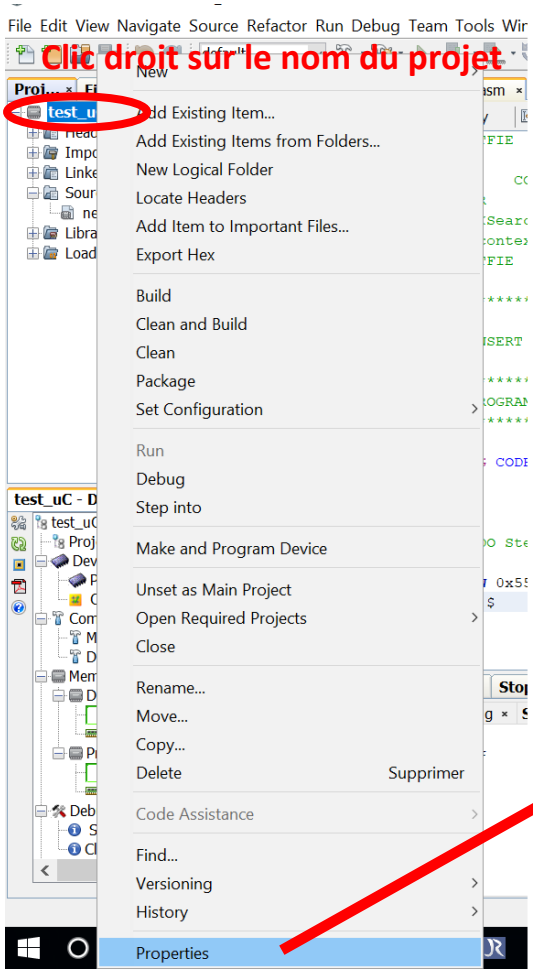
5 – Utilisation du « debugger » : Exécution d'un programme sur le composant

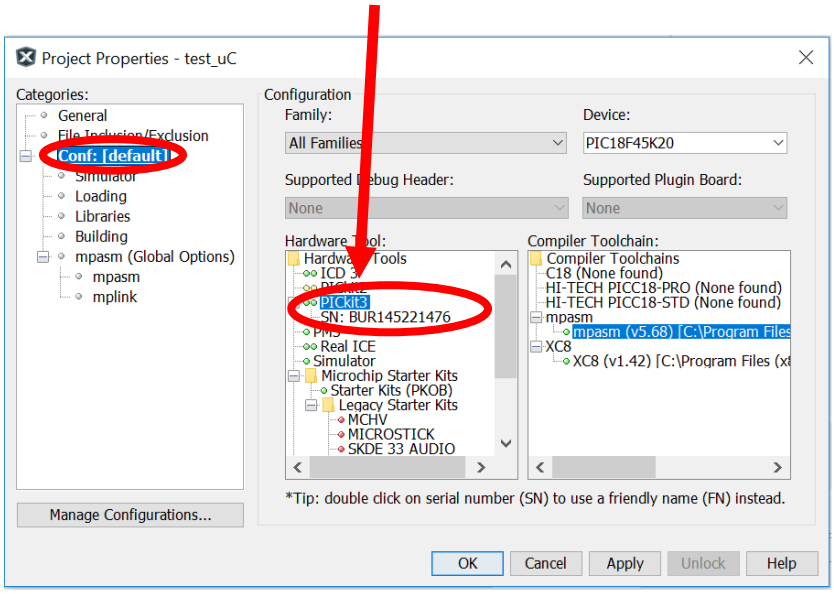
5.1 Configuration du PICkit3/PICkit4

Pour exécuter le programme sur le composant réel, et non pas en simulation pure, un outil de debug (par exemple PICkit3) doit être connecté entre le PC et le microcontrôleur.

Si c'est le cas, lors de la création du projet (cf. Annexe 1) il faut choisir comme outil de debug « PICkit3 » ou « PICkit4 » au lieu de « Simulator ».

Remarque : Ceci peut toujours être modifié par la suite dans les propriétés du projet.
Clic-droit sur le nom du projet, dans la fenêtre ***Projects → Properties***
puis ***Conf:[default] → Hardware Tool***



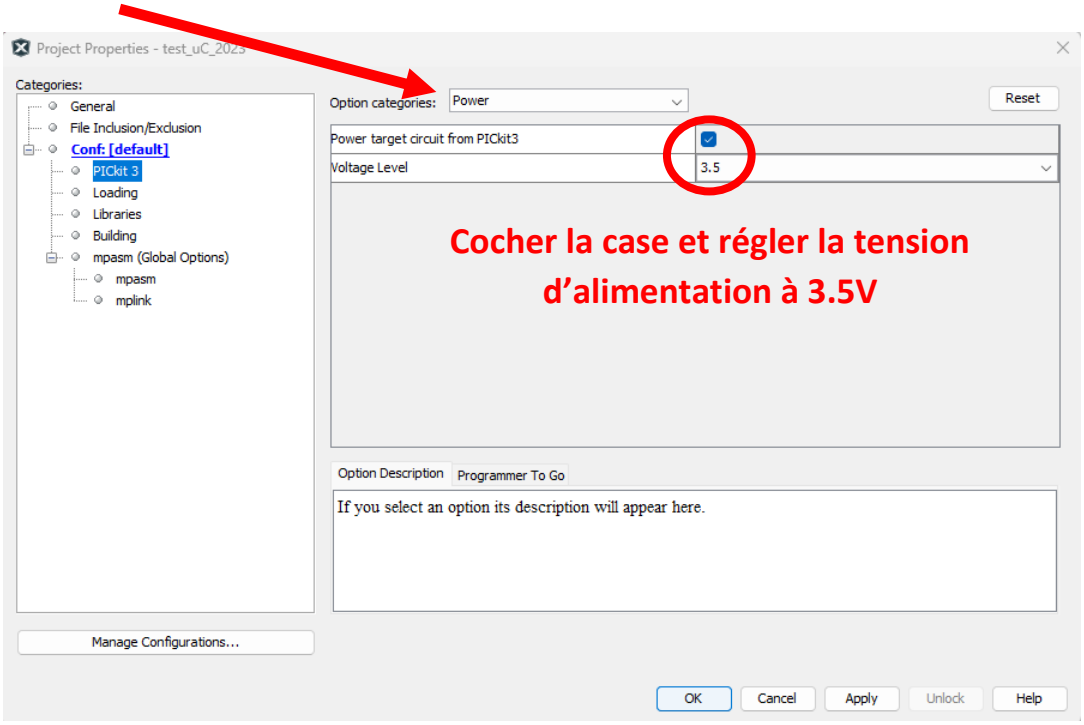


5.2 Alimentation de la carte de prototypage par le PICkit3/PICkit4

Lorsque la carte de prototypage ne dispose pas d'une source d'alimentation dédiée, il est possible de l'alimenter à partir du PICkit, **<u>à condition qu'elle consomme moins de 30 mA</u>**.

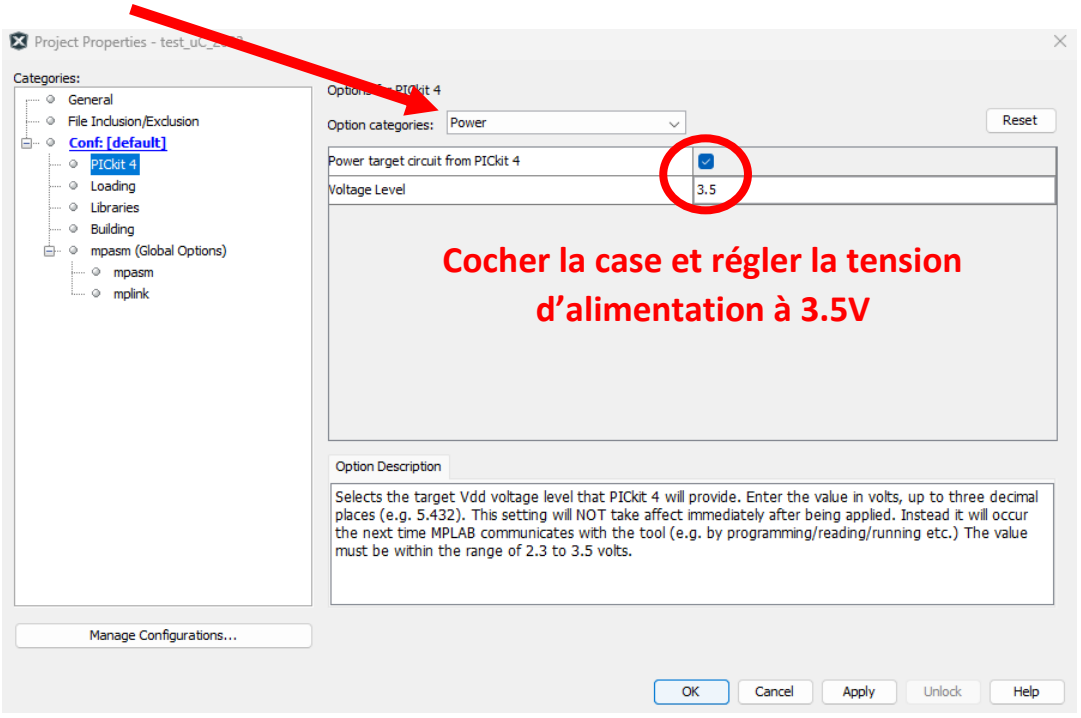
Pour utiliser cette option, éditer les propriétés du projet (cf. page précédente).

PICkit3 → Power → Power target circuit from PICKit3



ou

PICkit 4 → Power → Power target circuit from PICKit 4



6 – Description d'un registre dans la datasheet du PIC18F23K20

Dans la datasheet du PIC18F23K20, la description d'un registre se présente de la manière suivante :

Nom du registre

État par défaut du registre décrit ci-dessous

REGISTER 2-1: OSCCON: OSCILLATOR CONTROL REGISTER

R/W-0	R/W-0	R/W-1	R/W-1	R-q	R-0	R/W-0	R/W-0
IDLEN	IRCF2	IRCF1	IRCF0	OSTS(1)	IOFS	SCS1	SCS0
bit 7							bit 0

Legend:

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

bit 7 **IDLEN:** Idle Enable bit
1 = Device enters Idle mode on SLEEP instruction
0 = Device enters Sleep mode on SLEEP instruction

bit 6-4 **IRCF<2:0>:** Internal Oscillator Frequency Select bits
111 = 16 MHz (HFINTOSC drives clock directly)
110 = 8 MHz
101 = 4 MHz
100 = 2 MHz
011 = 1 MHz(3)
010 = 500 kHz
001 = 250 kHz
000 = 31 kHz (from either HFINTOSC/512 or LFINTOSC directly)(2)

bit 3 **OSTS:** Oscillator Start-up Time-out Status bit(1)
1 = Device is running from the clock defined by FOSC<2:0> of the CONFIG1 register
0 = Device is running from the internal oscillator (HFINTOSC or LFINTOSC)

bit 2 **IOFS:** HFINTOSC Frequency Stable bit
1 = HFINTOSC frequency is stable
0 = HFINTOSC frequency is not stable

bit 1-0 **SCS<1:0>:** System Clock Select bits
1x = Internal oscillator block
01 = Secondary (Timer1) oscillator
00 = Primary clock (determined by CONFIG1H[FOSC<3:0>]).

Description des différents bits composant le registre