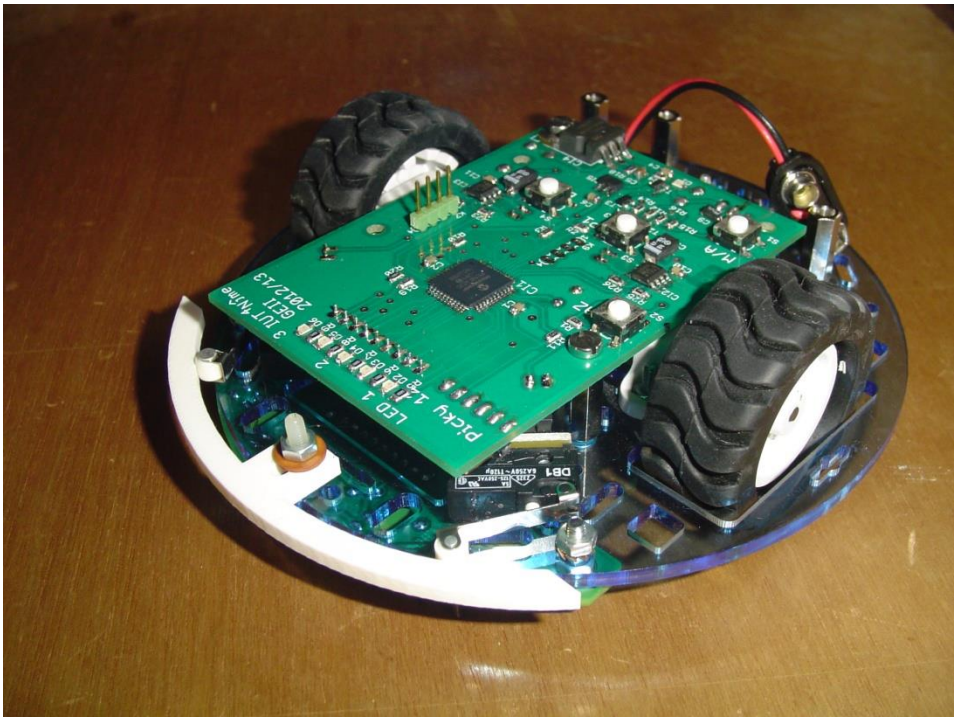
PICKY 12

ER S3 - Module Robotique - Systèmes Embarqués



Version 2.6

12 septembre 2017

Frédéric Giamarchi

Jean Roch Vaille

Département : Génie Électrique et Informatique Industrielle

IUT de Nîmes - Université de Montpellier

Sommaire

1 Présentation du projet 4

2 Structure mécanique du robot 4

2.1 Châssis 4

2.2 Moteurs 4

2.3 Roues 4

2.4 Pare-chocs 5

3 Électronique 5

3.1 Schéma bloc 5

3.2 Puissance 5

3.3 Interface Homme-Machine 7

3.4 Capteurs 8

3.5 Commande 9

4 Programmation 11

4.1 Environnement de développement (MPLAB IDE) 12

4.2 Compilateur (XC8) 12

4.3 Implantation du programme dans le microcontrôleur 13

5 Exercices 2ème semestre 15

5.1 Utilisation du projet « S2_Rob_00 » 15

5.2 Utilisation du simulateur 17

5.3 Exercice 1 : LEDs 21

5.4 Exercice 2 : Boutons Poussoir & Pare-Chocs 21

5.5 Exercice 3 : Capteurs de ligne 21

6 Exercices 3ème semestre 22

6.1 Introduction 22

6.2 Utilisation du projet 22

6.3 Organisation du projet 23

7 Exercices 25

7.1 Exercice 1 : Réaliser un chenillard 25

7.2 Exercice 2 : Programmer un code Erreur 25

7.3 Exercice 3 : Programmer un code Erreur par clavier 25

7.4 Exercice 4 : Test des moteurs 25

7.5 Exercice 5 : Comportement " pare-chocs" 25

7.6 Exercice 6 : Comportement " Suivre la ligne noire" 25

7.7 Exercice 7 : Comportement "Figures" 26

7.8 Exercice 8 : Comportement "Recharge" 26

7.9 Exercice 9 : Comportement "Eviter un obstacle" 26

1 Présentation du projet

Plate-forme robotique à vocation pédagogique, ce projet est destiné aussi bien aux lycéens qu'aux étudiants en écoles d'ingénieur. Mais tout amateur, souhaitant s'initier à la programmation en dehors d'un cadre académique, trouvera un support pour s'initier à cette science.

2 Structure mécanique du robot

2.1 Châssis

Figure 1 : Photo du Châssis

Basé sur un châssis de type Pololu, la structure supporte deux moteurs CC et une roue folle à l'arrière.

2.2 Moteurs

La référence des deux moteurs est MOT-993 de chez Pololu.
Caractéristiques:

- *Dimensions: 24 x 10 x 12 mm*
- *Poids: 0,34 oz*
- *Réduction: 30:1*
- *Alimentation 3 à 7 V env.*
- *Vitesse à vite @ 6 V: 440 rpm*
- *Courant à vide: 40 mA*
- *Courant en charge: 360 mA*
- *Couple: 4 oz.in*

Figure 2 : Photo d'un moteur

2.3 Roues

La référence des deux roues est POL1090 de chez Pololu.
Modèles avec pneus gomme.

- *Convient pour axe en "D".*
- *Dimensions: Ø 42 x 19 mm.*

Figure 3 : Photo des roues

2.4 Pare-chocs

Le pare-chocs a été dessiné sous Solidworks et réaliser avec une imprimante 3D. En cas de contact avec un obstacle, le pare-chocs déplace les éléments de 2 contacts de fin de course à galet.

3 Électronique

3.1 Schéma bloc

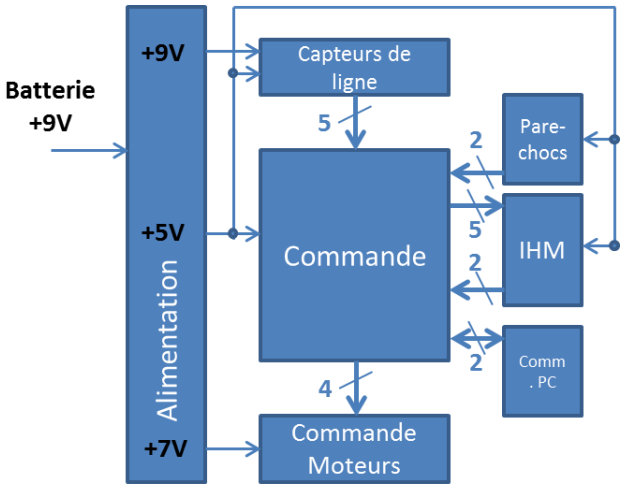


Figure 4 : Schéma Bloc Picky 12

3.2 Puissance

3.2.1 Alimentation

La mise sous tension du robot se fait par un « soft switch » construit autour des transistors T1, T2 et T3. Le transistor PMOS T1 joue le rôle d'interrupteur commandé. La commande de ce dernier est assurée par le système bistable construit autour des deux transistors bipolaires T2 (PNP) et T3 (NPN). Le changement d'état de ce système bistable est assuré par le bouton poussoir S1 et le condensateur C9.

La partie régulation est assurée par les trois régulateurs de tension CI4, CI5 et CI6 :

- *CI4 : LM317T, régulateur ajustable, 9V vers 7V : Alimentation des moteurs*
- *CI5 : MCP1702-5002, régulateur 5V : Alimentation de la commande*
- *CI6 : MCP1702-3302, régulateur 3.3V : Alimentation module Bluetooth*

La Led D1 permet de visualiser la mise sous tension du système.

Les résistances R19 et R20 forment un pont diviseur permettant au microcontrôleur de mesurer la tension de la batterie, sans risque de destruction du composant :

$$V_{mes} = \frac{R_{20}}{R_{20} + R_{19}} * V_{bat} = \frac{1}{3} V_{bat}$$

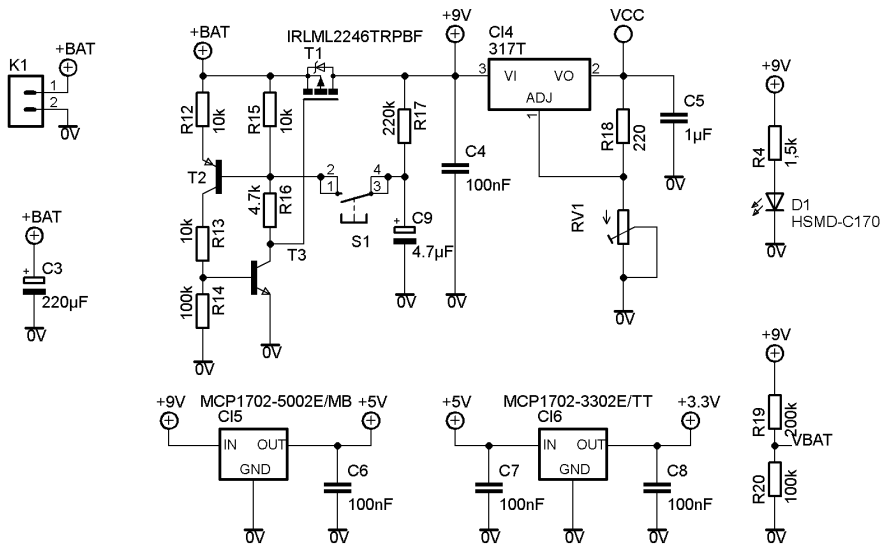


Figure 5 : Schéma Alimentation

3.2.2 Commande des moteurs

La commande des moteurs utilise deux interfaces de puissance. Ces interfaces ne sont que des « buffers » (la sortie recopie l'entrée). Les quatre résistances de pull-down permettent d'imposer un niveau bas à l'entrée des interfaces pendant la phase d'initialisation, phase pendant laquelle les sorties du microcontrôleur sont encore configurées en entrée et donc dans un état « haute impédance ».

Pour faire varier la vitesse de rotation des moteurs, on doit générer des signaux MLI (PWM) par le microcontrôleur. Celui-ci dispose de deux modules PWM (voir table de vérité).

Les Moteurs sont connectés à la carte principale par l'intermédiaire des connecteurs K3 (Moteur gauche) et K4 (Moteur droit)

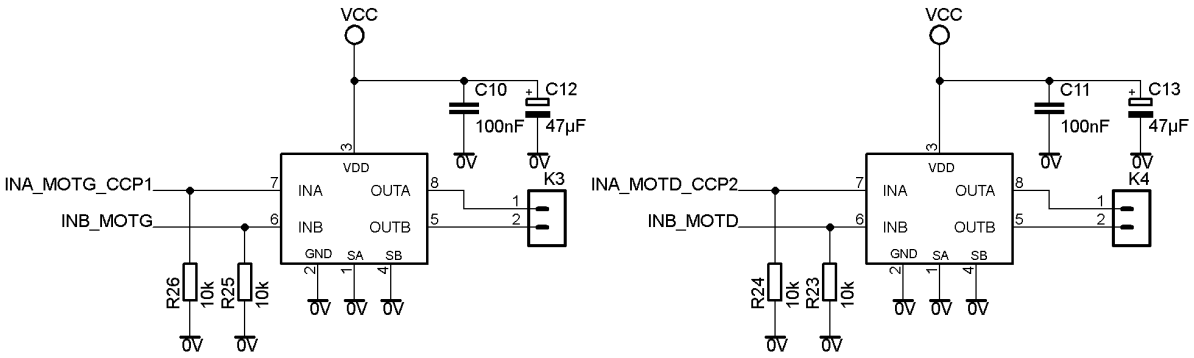


Figure 6 : Schéma Interface commande Moteurs

	Fonction	Ligne µC	INA	INB	MOT
INA_MOTG_CCP1	Mot. G. Av	RC2 – CCP1 (36)	PWM	0	AV
INB_MOTG	Mot. G. Ar	RC1 (35)	1	$\overline{PWM}$	AR
INA_MOTD_CCP2	Mot. D. Av	RB3 – CCP2 (11)	0	0	STOP
INB_MOTG	Mot. D. Ar	RB2 (10)	1	1	STOP

3.3 Interface Homme-Machine

L'interface Homme-Machine s'articule autour de trois systèmes d'interaction :

- *5 leds*
- *2 Boutons-poussoir*
- *Un buzzer Piezo*

3.3.1 LEDs

Pour des raisons de facilité de routage de la carte, les 5 leds ne sont pas connectées sur un seul et même port du microcontrôleur. Elles sont réparties sur les ports B, D et E.

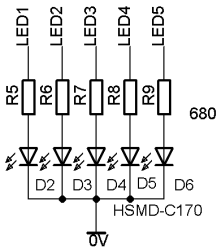


Figure 7: Schéma Leds

	Ligne µC (Pin)	
LED1	RE2	(27)
LED2	RE1	(26)
LED3	RB4	(14)
LED4	RD2	(40)
LED5	RD3	(41)

3.3.2 Boutons-Poussoir

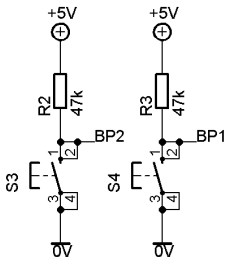


Figure 8 : Schéma boutons-poussoirs

Action	BP
Appui	0
Relâchement	1

	Ligne µC (Pin)	
BP1	RD0	(38)
BP2	RD1	(39)

3.3.3 Piezzo

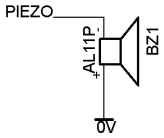


Figure 9 : Schéma piezo

	Ligne µC Pin)	
PIEZO	RC0	(32)

3.4 Capteurs

3.4.1 Capteurs de ligne

Les capteurs de ligne sont placés sur une carte séparée. Cette carte est connectée à la carte principale par l'intermédiaire des connecteurs JP1 (Carte Capteurs) et K2 (Carte Principale).

Les capteurs peuvent être lus soit en tout-ou-rien soit en analogique.

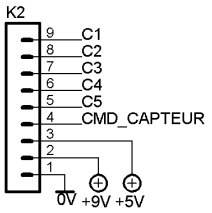


Figure 10 : Connecteur K2 (connexion carte capteurs - Carte principale)

	Fonction	Ligne µC (Pin)
C1	Capt. Ext. Gauche	RA5 – AN4 (24)
C2	Capt. Gauche	RA3 – AN3 (22)
C3	Capt. Milieu	RA2 – AN2 (21)
C4	Capt. Droit	RA1 – AN1 (20)
C5	Capt. Ext. Droit	RA0 – AN0 (19)
CMD_CAPTEUR	Commande Led IR des capteurs de ligne	RB5 (15)

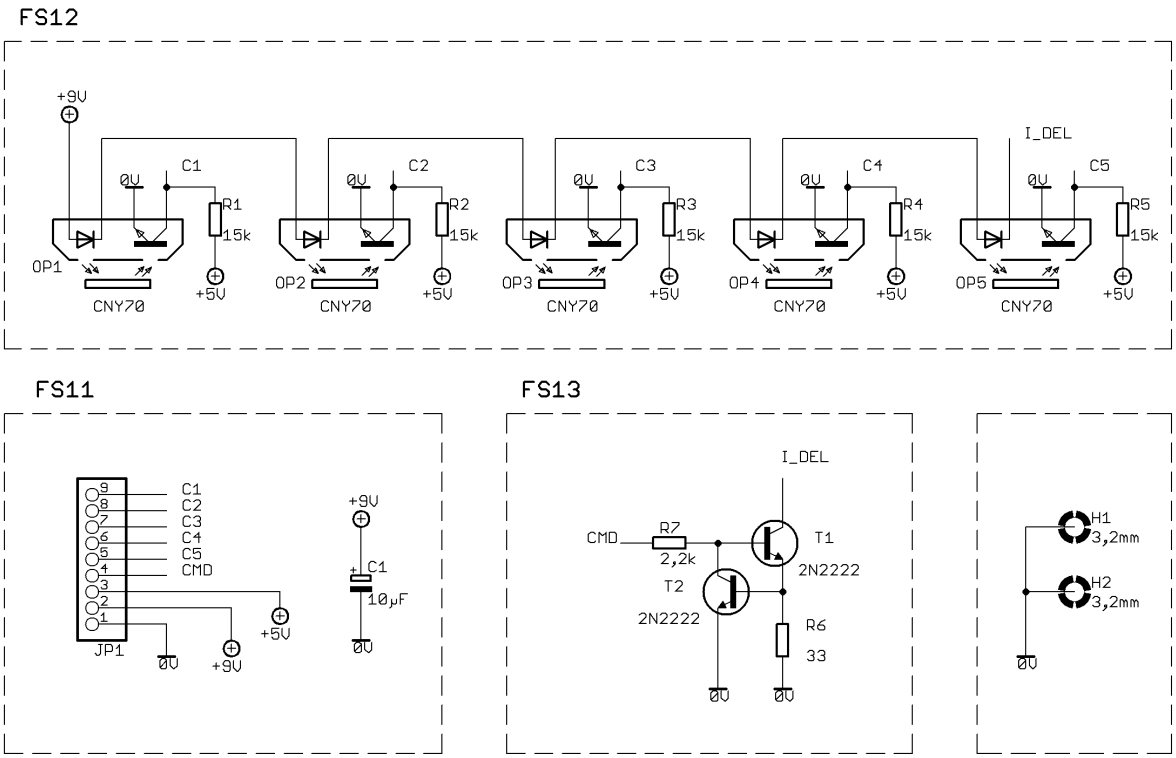
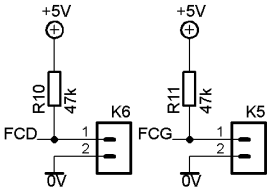


Figure 11 : Schéma Carte Capteurs

3.4.2 Pare-Chocs

Le pare-chocs du robot agit mécaniquement sur deux « fin de course » vissés sur le châssis. Ces deux switch sont connectés à la carte principale par l'intermédiaire des deux connecteurs K5 et K6.



	Ligne µC (Pin)	
BP1	RD0	(38)
BP2	RD1	(39)

Figure 12 : Schéma connecteurs K5 & K6

3.5 Commande

La commande du robot s'articule autour d'un microcontrôleur PIC18F4520. C'est un microcontrôleur 8 bits commercialisé par Microchip. Le microcontrôleur ***est cadencé à 32MHz*** *par un oscillateur interne. Compte tenu de l'architecture du composant et de la fréquence d'oscillation,* ***le temps de cycle*** *(temps nécessaire à l'exécution d'une instruction (code machine)* ***est de 125 ns***

3.5.1 Schéma bloc

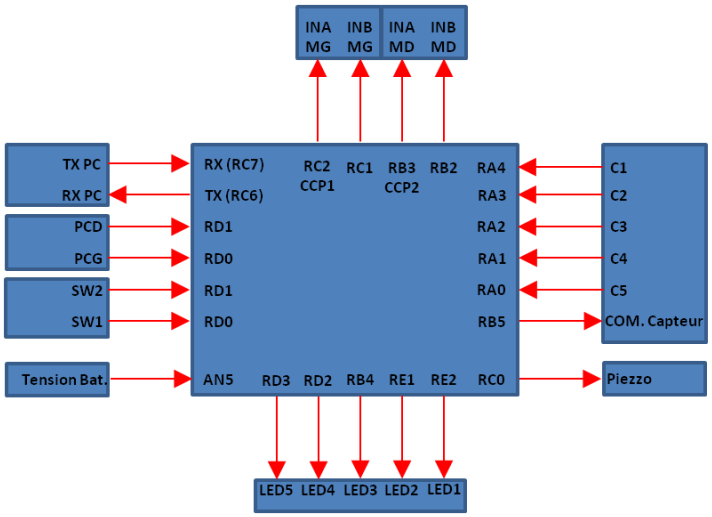


Figure 13 : Schéma bloc partie commande

3.5.2 Schéma électrique

Le bouton poussoir S2 permet de faire un « RESET » du microcontrôleur.

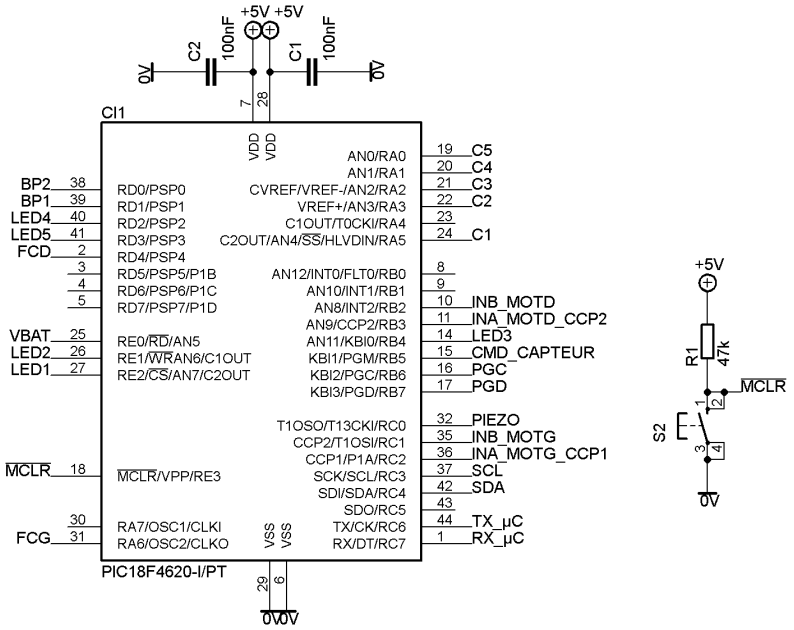


Figure 14 : Schéma Commande

3.5.3 Bilan Entrées – Sorties du microcontrôleur

	Ligne	Pin	Type	Module µC	Fonction
BP1	RD1	39	IN		Bouton-poussoir
BP2	RD0	38	IN		Bouton-poussoir
C1	RA5	24	IN	CAN (AN4) (Option)	Capt. Ext. Gauche
C2	RA3	22	IN	CAN (AN3) (Option)	Capt. Gauche
C3	RA2	21	IN	CAN (AN2) (Option)	Capt. Milieu
C4	RA1	20	IN	CAN (AN1) (Option)	Capt. Droit
C5	RA0	19	IN	CAN (AN0) (Option)	Capt. Ext. Droit
CMD_CAPTEUR	RB5	15	OUT		Commande Led IR des capteurs de ligne
FCD	RD4	2	IN		Fin de course (Pare-chocs) Droit

FCG	RA6	31	IN		Fin de course (Pare-chocs) Gauche
INA_MOTD_CCP2	RB3	11	OUT	CCP2 (PWM)	Commande PWM Moteur Droit
INA_MOTG_CCP1	RC2	36	OUT	CCP1 (PWM)	Commande PWM Moteur Gauche
INB_MOTD	RB2	10	OUT		Commande « sens » Moteur Droit
INB_MOTG	RC1	35	OUT		Commande « sens » Moteur Gauche
LED1	RE2	27	OUT		Commande LED1 (Actif niveau haut)
LED2	RE1	26	OUT		Commande LED2 (Actif niveau haut)
LED3	RB4	14	OUT		Commande LED3 (Actif niveau haut)
LED4	RD2	40	OUT		Commande LED4 (Actif niveau haut)
LED5	RD3	41	OUT		Commande LED5 (Actif niveau haut)
MCLR	RE3	18	IN	MCLR	Reset µC (Actif niveau bas)
PGC	RB6	16	IN	ICSP	Lignes de programmation du microcontrôleur
PGD	RB7	17	IN/OUT	ICSP	
PIEZO	RC0	32	OUT		Commande du Buzzer Piezo
RX_µC	RC7	1	IN	UART 1	Réception liaison série
SCL	RC3	37	OUT	I^2C 1	Horloge I^2C
SDA	RC4	42	IN/OUT	I^2C 1	Données I^2C
TX_µC	RC6	44	OUT	UART 1	Emission liaison série
VBAT	RE0	25	IN	CAN (AN5)	Mesure tension batterie

4 Programmation

Le « workflow » de programmation comprend trois étapes :

1. *Écriture du programme*
2. *Compilation*
3. *Implantation du code machine dans la mémoire programme du µC*

4.1 Environnement de développement (MPLAB IDE)

La première étape du workflow consiste à éditer le programme (i.e. écrire les lignes de code). L'écriture doit se faire dans le langage défini par le compilateur utilisé. Dans notre cas, le programme sera écrit en ***langage C.***

Historiquement l'écriture du programme se faisait avec un simple éditeur de texte. Aujourd'hui, on utilise un Environnement de Développement Intégré (IDE). Ces outils comme Eclipse, Netbean, MPLAB, ***MPLABX****,…) permettent de faciliter cette opération en apportant plus de fonctionnalités qu'un simple éditeur de texte. Parmi ces fonctionnalités, on citera par exemple :*

- ***Gestion des projets :*** *L'IDE permet de gérer l'ensemble des fichiers liés à un projet. En effet, un projet est très souvent constitué de nombreux fichiers (sources, header…)*
- ***Coloration syntaxique*** *: l'éditeur intégré à l'IDE adapte la couleur du texte à la syntaxe du langage utilisé. Ceci permet une meilleure lecture du programme.*
- ***Auto-Complétion :*** *Dans certains cas, l'éditeur peut suggérer automatiquement des noms d'instruction, de variable ou de fonction au cours de la frappe. (On notera que dans certains cas, ces suggestions doivent être déclenchées manuellement par l'appui sur une combinaison de touches). On notera que cette fonctionnalité n'est pas disponible sur les versions 8.xx de MPLAB. Pour en bénéficier, il faut utiliser la nouvelle version :* ***MPLABX****.*
- ***Détection des erreurs de syntaxe à la volée :*** *l'éditeur intégré à l'IDE peut vérifier la syntaxe du programme de façon automatique au cours de la frappe (un peu à la manière du correcteur orthographique d'un logiciel de traitement de texte.)*

4.2 Compilateur (XC8)

Le microcontrôleur ne comprend que le code machine (Chaque famille de µC possède son propre jeu d'instruction). Si on veut utiliser un langage évolué utilisable sur un grand nombre de composant, il est nécessaire de traduire le programme écrit en C, en code machine. Cette étape de programmation incombe au compilateur. Cet outil qui est propre à chaque architecture de µC va traduire le code C en code machine afin qui puisse être exécuté par le µC. Implantation du programme dans le microcontrôleur.

On notera que pour une architecture de µC donnée (dans notre cas l'architecture PIC18F), il peut exister plusieurs compilateurs C (CCS, C18, HI-TECH, XC8…).

Pour notre projet nous utiliserons le compilateur ***XC8.***

4.3 Implantation du programme dans le microcontrôleur

Dans cette partie, le terme « programmation » est un raccourci pour parler de l'implantation du programme en langage machine dans le microcontrôleur. Nous avons à notre disposition deux modes de programmation :

4.3.1 In Circuit Serial Programing (ICSP)

Ce mode de programmation est le mode natif du microcontrôleur. Il est simple d'accès car directement accessible via MPLAB. Mais il nécessite l'utilisation d'un programmateur spécial (ICD2, PICkit2, PICkit3…). Par contre, l'utilisation d'un tel programmateur peut s'avérer très utile pour « débuguer » un programme. En effet, en règle générale, ces programmateurs peuvent être utilisés en mode « débuggeur ». Ce mode permet de placer des points d'arrêt, d'exécuter le programme pas-à-pas ou d'aller voir le contenu de la mémoire du microcontrôleur pendant d'exécution du programme.

4.3.2 Bootloader

Il est également possible de programmer le microcontrôleur en utilisant un bootloader. Le bootloader est un petit programme qui est préalablement implanté sur le microcontrôleur et qui permet d'établir une liaison série entre le microcontrôleur et un ordinateur. Une fois cette connexion établie, l'ordinateur va envoyer le fichier HEX au microcontrôleur. Ce dernier va copier les octets reçus dans sa propre mémoire flash (mémoire programme). Afin de pouvoir communiquer avec le bootloader, il est nécessaire d'utiliser un logiciel spécial installé sur l'ordinateur.

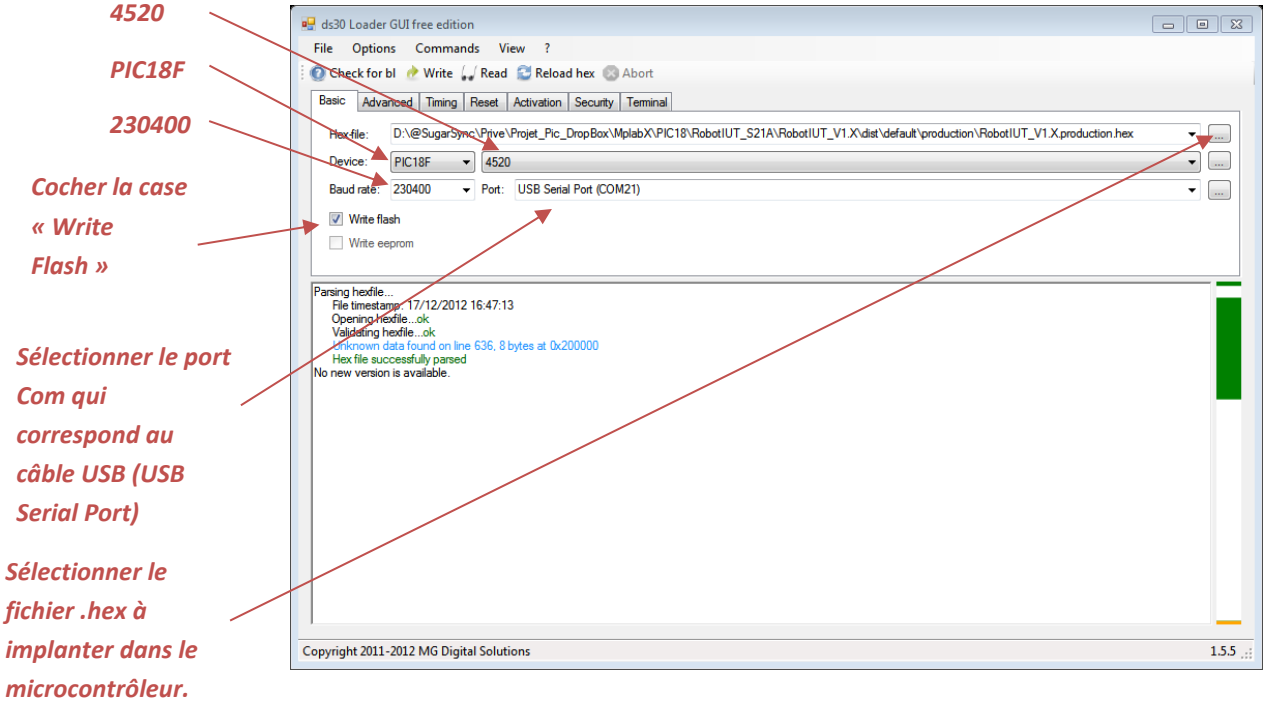
4.3.3 Comparatif des deux modes de programmation

	Avantages	Inconvénients
ICSP	• Géré nativement par MPLAB • Permet le mode debug • Pas besoin de pré-programmation du microcontrôleur	• Obligation d'avoir un programmateur • Double connexion entre le PC et le microcontrôleur si on utilise en plus la liaison série
Bootloader	• Une seule connexion entre l'ordinateur et le microcontrôleur (liaison RS-232) • Plus besoin de programmateur spécial après la première programmation	• Nécessité de pouvoir programmer le microcontrôleur une première fois. • En règle générale, obligation d'utiliser un logiciel tiers pour la programmation

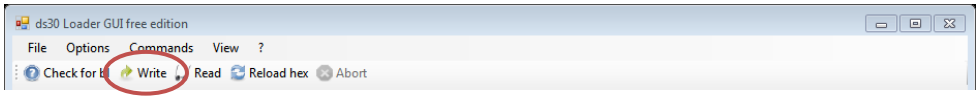
4.3.4 Utilisation du bootloader

Le microcontrôleur présent sur le Robot Picky 12 est déjà programmé avec le bootloader.

1. *Ouvrir le programme « ds30LoaderGui.exe »*
2. *Dans la barre de menu cliquer sur « View → Advanced Mode »*
3. *Sélectionner les éléments suivant :*



4. *Connecter le câble USB au Robot (attention au sens du connecteur)*
5. *Cliquer sur le bouton « Write »*

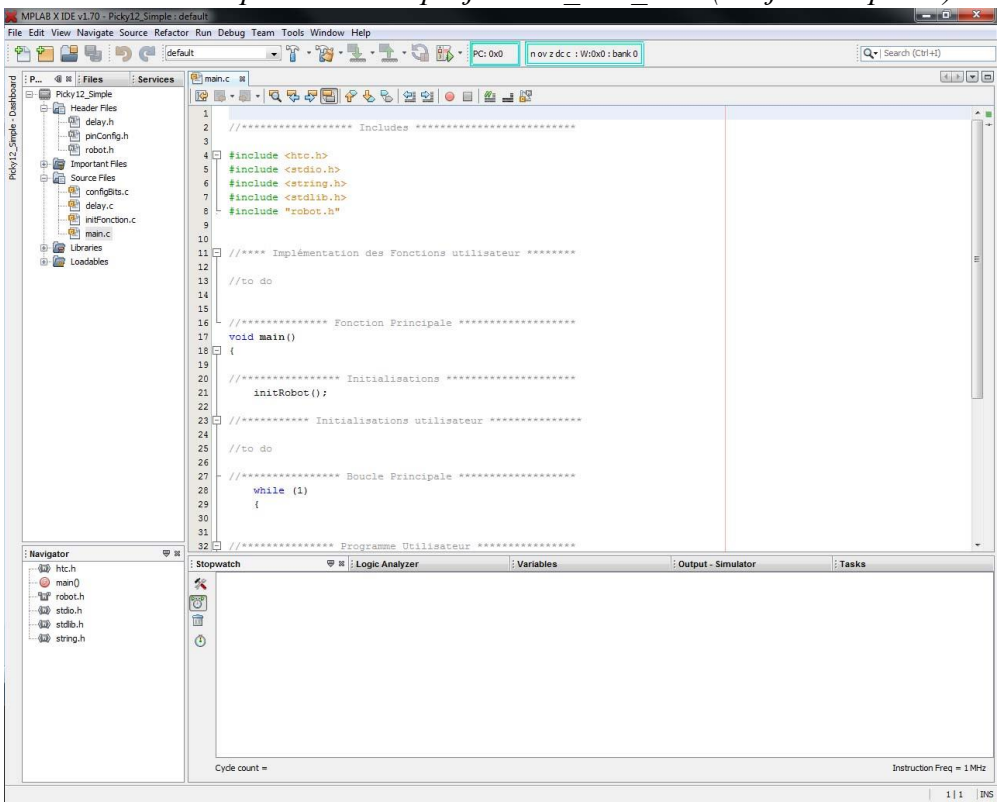


6. *Le logiciel se met en attente du microcontrôleur.*
7. *Faire un reset du microcontrôleur (ou mettre le robot sous tension)*
8. *La programmation commence…*
9. *Attention, à la fin de la programmation, le microcontrôleur va se réinitialiser afin d'exécuter le nouveau programme. Si les moteurs sont commandés, le robot risque d'arracher le câble de programmation.*

5 Exercices 2ème semestre

5.1 Utilisation du projet « S2_Rob_00 »

Ouvrir MPLAB X IDE puis ouvrir le projet « S2_Rob_00 » (Project→Open…).



Ce projet contient les fichiers de base permettant de faire un programme simple. Il contient les fichiers suivants :

- *Fichier « entête » (header *.h)*
  - *delay.h : déclarations des fonctions :*
    - *delayMs(int ms)*
    - *delayUs(int us)*
  - *pinConfig.h : Macros définissant toutes les lignes d'Entrée/Sortie*
  - *robot.h : include de tous les fichiers entête nécessaires*

- *Fichiers « sources » (*.c)*
  - *configBits.c : Configurations des bits de configurations du µC.*
  - *delay.c : Implémentation des fonctions déclarées dans le fichier delay.h*
  - *fonctions.c : Implémentation de toutes les fonctions d'initialisation.*

De plus amples informations sur la documentation en ligne.

Sans indication contraire, les modifications seront faites dans le fichier « main.c ». Ce fichier contient la fonction principale du projet : la fonction « void main ( ) »

main.c :

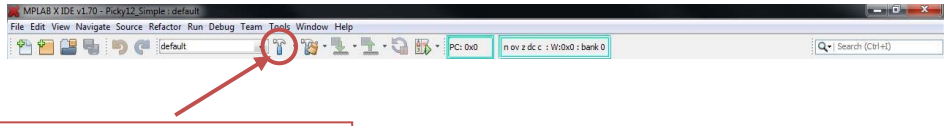
```
//****************** Includes *************************

#include <htc.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "robot.h"



void main()
{
//**************** Initialisations ********************
    initRobot();

//*********** Initialisations utilisateur ***************

// to do


//**************** Boucle Principale ******************
    while (1)
    {

//*************** Programme Utilisateur ***************


        // to do

    }
}
```

On peut remarquer trois zones « to do ». Ces deux zones sont à compléter en fonction de l'exercice.

La première zone devra contenir l'implémentation des fonctions utilisateur.

La zone « Initialisations utilisateur » contiendra les lignes de programme à exécuter une seule fois alors que la zone « Programme Utilisateur » contiendra les lignes qui s'exécuteront à chaque boucle.

Avant de se lancer dans les exercices, il est conseillé de compiler le projet une première fois.

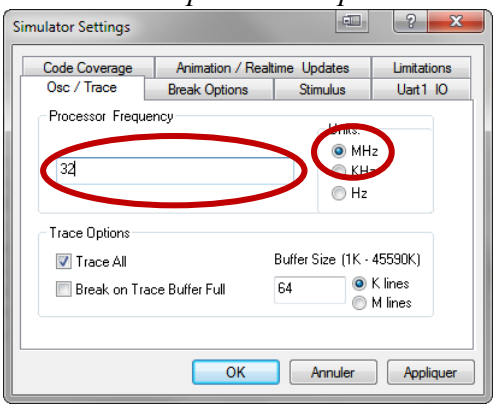


Bouton « Build Project » permettant de compiler le projet

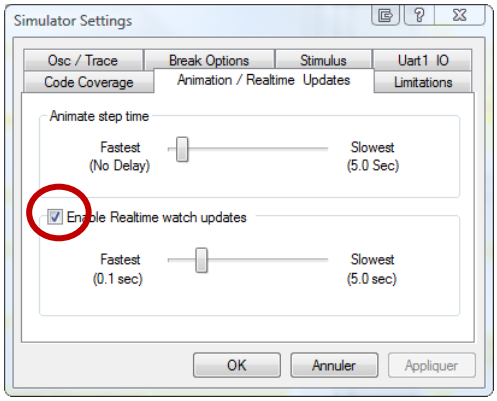
5.2 Utilisation du simulateur

5.2.1 Exercice 1

- *Cliquer sur : Debugger → Select Tool → MPLAB SIM.*
- *Cliquer sur : Debugger → Settings*
- *Dans l'onglet « Osc / Trace », remplacer la fréquence processeur par 32 MHz. Ceci permet d'avoir une base de temps correcte pour les simulations.*



- *Dans l'onglet « Animation / Realtime update », cocher la case « enable Realtime watch update". Ceci permet d'avoir un rafraichissement de la fenêtre "watch" en temps réel.*

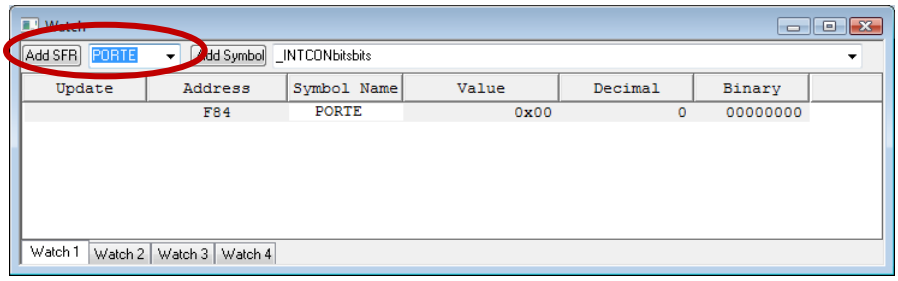


- *Valider en cliquant sur « OK »*
- *Taper la ligne de code suivante dans la boucle principale :*

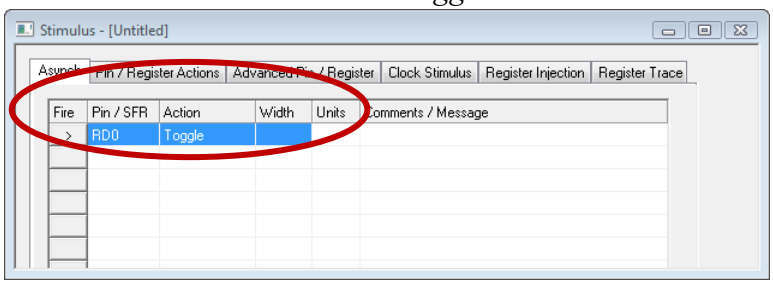
```
PORTEbits.RE2=PORTDbits.RD0;
```

Cette ligne recopie l'état de la ligne RD0 sur la ligne RE2. La configuration de ces lignes est faite par ailleurs.

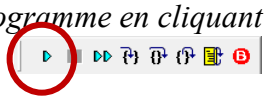
- *Cliquer sur :View → Watch. Cette fenêtre permet de suivre le contenu d'une variable (Symbol) ou d'un registre du microcontrôleur (SFR)*
- *Dans la liste déroulante choisir le PORTE (c'est la sortie)*
- *Cliquer sur « Add SFR »*



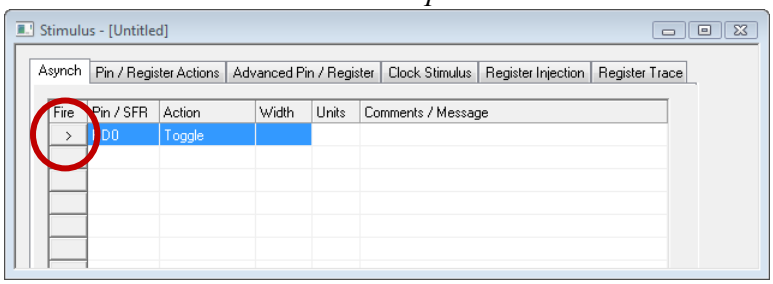
- *Cliquer sur Debugger → Stimulus → New Workbook. Cette fenêtre permet, entre autres, de gérer un stimulus sur une ligne du microcontrôleur.*
- *Cliquer sur l'onglet « Asynch »*
- *Dans la colonne « Pin/SFR » choisir « RD0 » (c'est l'entrée)*
- *Dans la colonne « Action » choisir « Toggle »*



- *Lancer l'exécution du programme en cliquant sur le bouton « Run »*



- *Remarquer l'évolution du registre « PORTE » fenêtre « Watch » lorsqu'on clique sur le bouton « Fire » du stimulus crée précédemment*

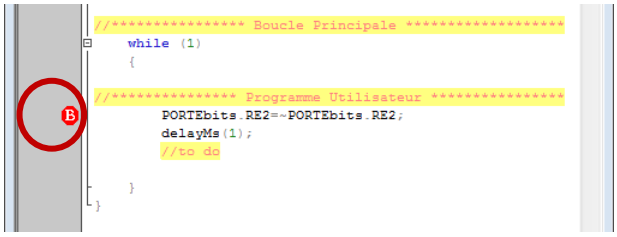


5.2.2 Exercise 2

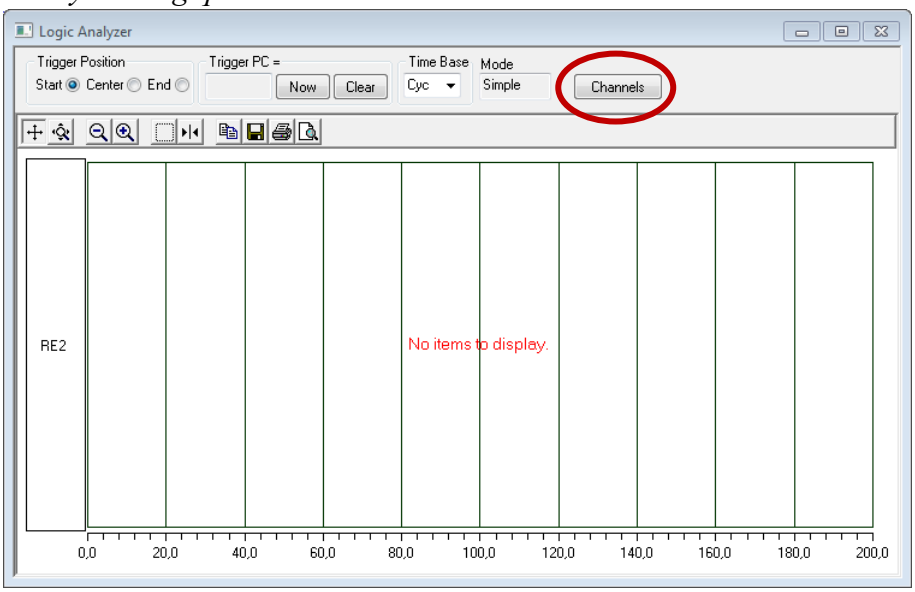
- *Taper les lignes de code suivantes dans la boucle principale à la place de la précédente :*

```
PORTEbits.RE2=~PORTEbits.RE2;
delayMs(1);
```

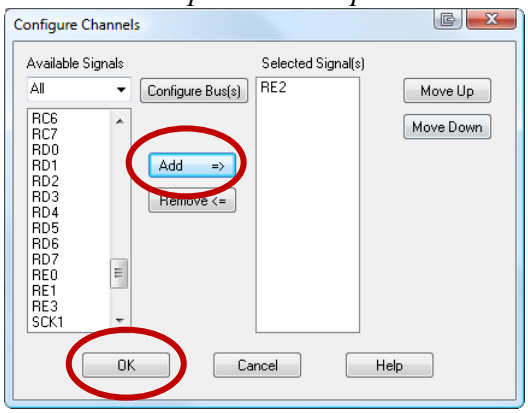
- *Placer un point d'arrêt (breakpoint) en double-cliquant dans la marge grise en face de la ligne souhaitée.*



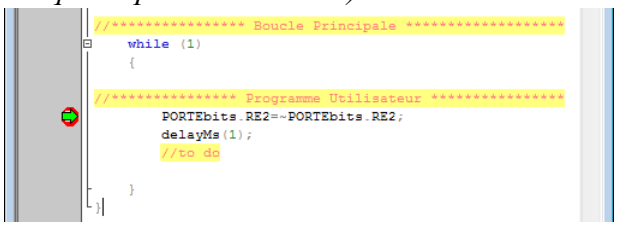
- *Cliquer sur « View → Simulator Logic Analyser » afin de faire apparaitre l'analyseur logique*



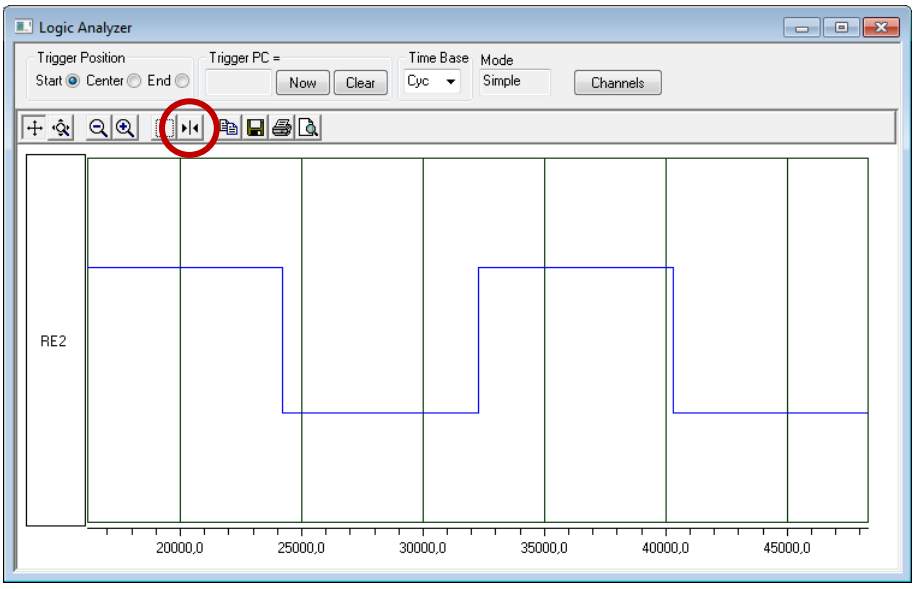
- *Cliquer sur le bouton « Channels »*
- *Dans la liste « Available Signal » sélectionner la ligne « RE2 »*
- *Cliquer sur le bouton « Add » puis « OK » pour valider*



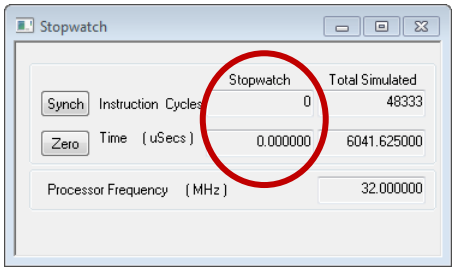
- *Lancer la simulation. Remarquer que l'exécution s'arrête sur le point d'arrêt. (la flèche verte indique la position actuelle)*



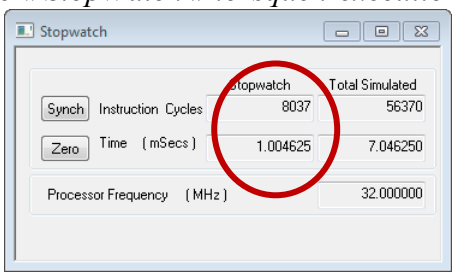
- *Remarquer qua chaque arrêt, la fenêtre de l'analyser logique est rafraichie. Relancer plusieurs fois la simulation en rappuyant sur le bouton « Run ».*
- *Observer la fenêtre de l'analyseur logique :*
- *Cliquer sur le bouton « Cursor » afin de faire apparaitre deux curseurs permettant de faire des mesures de temps (en nombre de cycles machine)*



- *A l'aide des deux curseurs, mesurer le temps à l'état bas. En prenant en compte le temps de cycle, vérifier le bon fonctionnement de l'instruction delayMs( ).*
- *Cliquer sur « Debugger → StopWatch » pour faire apparaitre le chronomètre. Cette fenêtre permet de compter le nombre de cycle machine (et donc le temps écoulé) entre deux arrêts du programme.*
- *Vérifier que le programme est bien en pause (flèche verte dans le code)*
- *Vérifier que le chronomètre est bien à 0*



- *Relancer l'exécution du programme en cliquant sur « Run »*
- *Observer la fenêtre « StopWatch » lorsque l'exécution s'interrompt.*



- *Conclure*

5.3 Exercice 1 : LEDs

1. *Chercher dans le fichier « pinConfig.h » la macro définissant la led LED1. Vérifier avec le schéma électrique.*
2. *Chercher dans le fichier « pinConfig.h » la macro définissant le bit de direction associé à la led LED1*
3. *Retrouver dans le fichier « initFonction.c » la ligne permettant de configurer en sortie la ligne de la led LED1. Est-ce que cette ligne est exécutée lors de l'initialisation ?*
4. *Écrire la ligne permettant d'allumer la led LED1.*
5. *Utiliser la fonction « delayMs( ) » pour faire clignoter la led LED1.*
6. *Est-ce que les 5 leds sont connecté au même port ? Est-il facile de piloter les 5 leds simultanément ? (Affichage d'une valeur donnée)*
7. *Écrire une fonction « setLed( ) » permettant de piloter les 5 leds comme un afficheur binaire de 5 bits (on prendra la LED5 comme lsb). Utiliser cette fonction dans le programme principal.*
8. *En utilisant l'opérateur de décalage à gauche (<<) ou à droite (>>) réaliser un chenillard qui change d'état toutes les 200 ms.*
9. *Écrire une fonction permettant d'afficher une séquence particulière sur les 5 leds. Cette fonction sera appelée lors de l'initialisation.*

5.4 Exercice 2 : Boutons Poussoir & Pare-Chocs

1. *En utilisant les macros définies dans le fichier « pinConfig.h », écrire un programme réalisant les opérations suivantes :*
   - *Recopie de l'état du pare-choc gauche sur la LED5*
   - *Recopie de l'état du pare-choc droit sur la LED1*
   - *Recopie de l'état du bouton poussoir S4 sur la LED4*
   - *Recopie de l'état du bouton poussoir S3 sur la LED2*
   - *Clignotement de la LED3 avec une période de 200ms*
2. *Expliquer le comportement observé et écrire les lignes permettant d'obtenir un comportement plus « naturel »*
3. *Expliquer les problèmes de latence observés.*
4. *Proposer une solution simple pour résoudre ce problème*

5.5 Exercice 3 : Capteurs de ligne

1. *En utilisant les macros définies dans le fichier « pinConfig.h », écrire un programme réalisant les opérations suivantes :*
   - *Recopie de l'état des 5 capteurs de lignes sur les 5 LEDs*
2.

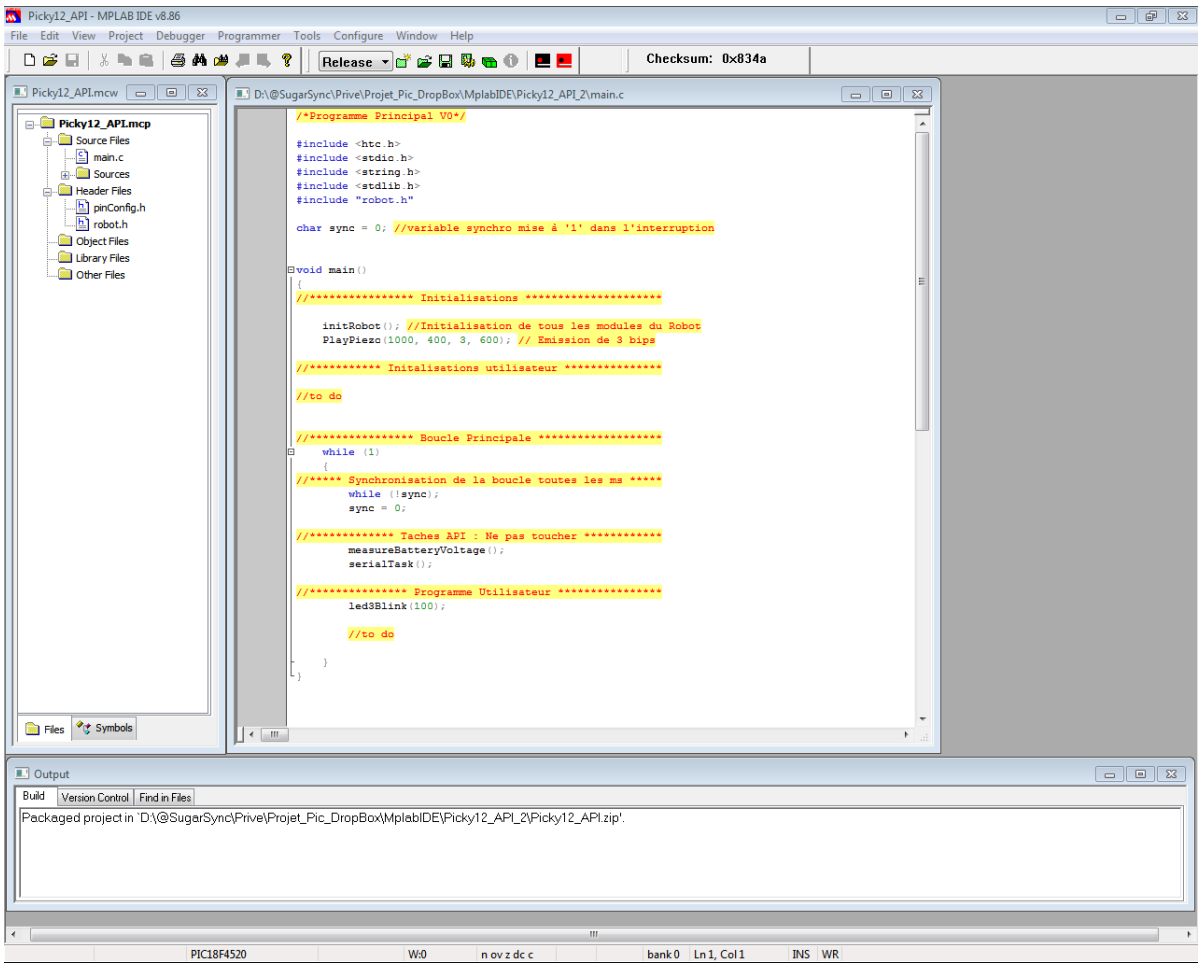
6 Exercices 3ème semestre

6.1 Introduction

Afin de faciliter la programmation de taches complexes, une couche logicielle évoluée est mise à disposition. Cet ensemble de fonctions doit permettre de simplifier la programmation du robot Picky 12 en masquant un grand nombre de considération matérielle. De plus, le programme possède un gestionnaire d'interruption qui permet, entre-autre, de cadencer la boucle principale.

6.2 Utilisation du projet

Ouvrir MPLAB IDE puis ouvrir le projet «S3_Rob_00» (Project→Open…).



6.3 Organisation du projet

Le programme de base est articulé autour de trois premiers fichiers :

- *main.c*
- *fonctions.c*
- *robot.c*

Les autres fichiers .c contenus dans le dossier « Lib » contiennent le code source de toutes les fonctions de base du programme. Il est fortement conseillé de ne pas les modifier.

6.3.1 main.c

Ce fichier contient deux choses :

- *Déclaration de la variable globale* **char synchro.** *Cette variable, initialiser dans la fonction d'interruption, permet de synchroniser la boucle principale toute les 10ms.* ***On peut utiliser cette fonctionnalité pour gérer le temps en comptant tout simplement les boucles en incrémentant une variable.***
- *Implémentation de la fonction* **void main(void).** *Cette fonction est composée de deux grandes parties :*
  - *La partie « Initialisation » qui est exécutée qu'une seule fois contient les blocs suivants :*
    - *<u>Initialisations</u> : Appel de la fonction d'initialisation des modules du Robot.*
    - *<u>Initialisations</u> utilisateur : A faire…*
  - *Le partie « Boucle » qui est exécutée toute les 10 millisecondes.*
    - *<u>Synchronisation de la boucle</u> : On attend que la variable de synchro soit positionnée à '1' par la routine d'interruption avant d'exécuter les diverses fonctions dans la boucle.*
    - *<u>Programme utilisateur</u> : Cette partie contiendra le programme principal du robot. Afin d'avoir un programme qui fait quelque chose par défaut, une instruction permettant de faire clignoter la LED3 avec une période de 1s a été ajouté.*

main.c :

```
/*Programme Principal V0*/

#include "system.h"
#include "delay.h"
#include "user.h"
#include "interrupt.h"
#include "fonctions.h"
#include "robot.h"

char sync = 0; //variable synchro mise à '1' dans
l'interruption


void main()
{
    /* Configure l'oscillator pour le composant */
    ConfigureOscillator();
    /* Configure le composant pour l'application */
    InitRobot();
    /* Test de la liaison serie */
    printf("Module Robotique * Exo 00\r\n");
    /* Test du fonctionnement des Leds */
    chenillard();
//**************** Boucle Principale ******************
    while (1)
    {
//***** Synchronisation de la boucle toutes les 10ms *****
        synchro = 0;
        while (!synchro);
    /* Led de test de bon fonctionnement de la boucle
principale */
        coeur_bat();
    /* Interaction Prioritaire */
        Action_Moteurs();
        Suivi_de_ligne();
        Pare_chocs();
        Figures();
        Recharge_accu();
        Arbitre();
     }
}
```

6.3.2 fonctions.c

Ce fichier contient une partie des exercices :

- *La fonction "cœur_bat(void)" pour indiquer le bon déroulement du programme.*
- *La fonction "chenillard(void)" pour générer une séquence lumineuse en début de programme pour indiquer une initialisation du microcontrôleur.*
- *La fonction « codeErreur(char n_cli) » pour générer un clignotement d'une Del pour indiquer une type d'erreur relevé.*

Des informations plus détaillées sont disponibles dans la documentation.

7 Exercices

7.1 Exercice 1 : Réaliser un chenillard

Compléter la fonction "void chenillard(void)". Il s'agit d'un défilement d'une seule Led parmi les 5 Leds disponibles et réalisant un seul aller-retour.

Cette action lumineuse est activée lors de la phase initialisation avant d'entrer dans la boucle infinie "while(1)". Elle permet de vérifier que le programme dans le µC passe par cette phase. Vous pouvez utiliser des délais ici, uniquement.

Le but de cet exercice simple est de reprendre la main sur l'ensemble du processus MplabX – Compilateur – ds30Loader - µC

7.2 Exercice 2 : Programmer un code Erreur

Cet exercice est un premier défi sur l'apprentissage des bonnes techniques de programmation en suivant le principe des diagrammes d'états.

Suivre le schéma bloc donné dans le cours pour réaliser cette programmation.

L'activation de cette fonction est demandée dans le fichier "main.c". Analyser cette action pour comprendre l'effet déclencheur.

Exercice 2.1 : Programmer la fonction Beeper avec un nombre de bip comme entrée, en prenant comme exemple la fonction Code_Erreur.

7.3 Exercice 3 : Programmer un code Erreur par clavier

Dans cet exercice, c'est par le clavier du PC que l'on va donner le nombre de clignotement. On utilise, ici, une possibilité du programme ds30Loader en mode Terminal pour envoyer des commandes vers le robot.

7.4 Exercice 4 : Test des moteurs

Ici vous devrez activer les deux moteurs du robot sur l'appui sur le bouton poussoir BP1 et les arrêter sur un appui sur le bouton poussoir BP2. Mais le départ du robot doit se faire que 3s après l'appui sur BP1.

Trouver les bonnes valeurs pour les vitesses des moteurs droit et gauche afin que le robot avance en ligne droite. Mais pas trop rapide.

7.5 Exercice 5 : Comportement " pare-chocs"

Cet exercice reprend le principe des diagrammes d'états, mais au sein d'une architecture à interactions prioritaires (voir cours).

Suivre le schéma bloc donné dans le cours pour réaliser cette programmation.

7.6 Exercice 6 : Comportement " Suivre la ligne noire"

Utiliser les capteurs infrarouges en mode analogique pour détecter la position du robot par rapport à la ligne noire. Une variable « erreur » sera utilisée pour augmenter la vitesse de la roue extérieure dans les virages et réduire la vitesse de la roue intérieure.

Un algorithme de détection de position doit être utilisé pour obtenir cette variable « erreur ».

Remarque : ne pas oublier d'activer les Dels infrarouges avant de lire les capteurs, puis de les éteindre, après. Il faut attendre 50µs avant de commencer à lire les capteurs.

7.7 Exercice 7 : Comportement "Figures"

Sur une piste ovale, vous placerez un croisement et un T de 10cm sur les 2 lignes droites. Le robot doit détecter le T et faire un demi-tour vers le côté opposé au T. Mais lorsqu'il rencontre un croisement, le robot doit continuer tout droit.

Utiliser les capteurs extérieurs pour différentier le T d'un croisement.

7.8 Exercice 8 : Comportement "Recharge"

Afficher la tension de la pile sur le PC. Pour un seuil de 7,7Volts, le robot doit passer en mode recharge et arrêter le robot. Un code erreur 5 sera généré en même temps.

Remarque *:*

La tension de la pile est divisée par 3 par un pont diviseur de tension avant d'être mesurée par le µC.

Pour une pile moyennement chargée, il se peut que le robot passe directement en mode recharge après un premier départ. Comment remédier à ce problème.

Option *:*

Vous pouvez émettre un bip toutes les 5 secondes pour indiquer cet état.

7.9 Exercice 9 : Comportement "Eviter un obstacle"

Pour éviter un obstacle, il faut le détecter avant une collision. Le capteur utilisé est un dispositif à ultrasons permettant de mesurer des distances.

Le capteur utilisé est un module HC-SR04. Rechercher la documentation sur Internet pour comprendre son fonctionnement.

1. *Compléter la fonction associée, HC_SR04() et tester celle-ci avec des « printf ».*
2. *Détecter un obstacle à 10cm et stopper le robot*
3. *Contourner cet obstacle*