11/2023

# Test des sorties du robot RAPE sur MPLABX

Dans ce cours nous présentons le programme test des sorties et proposons de faire le test afin de s'assurer du bon fonctionnement des sorties avant de mettre le programme d'aspiration en œuvre.

## Table des matières



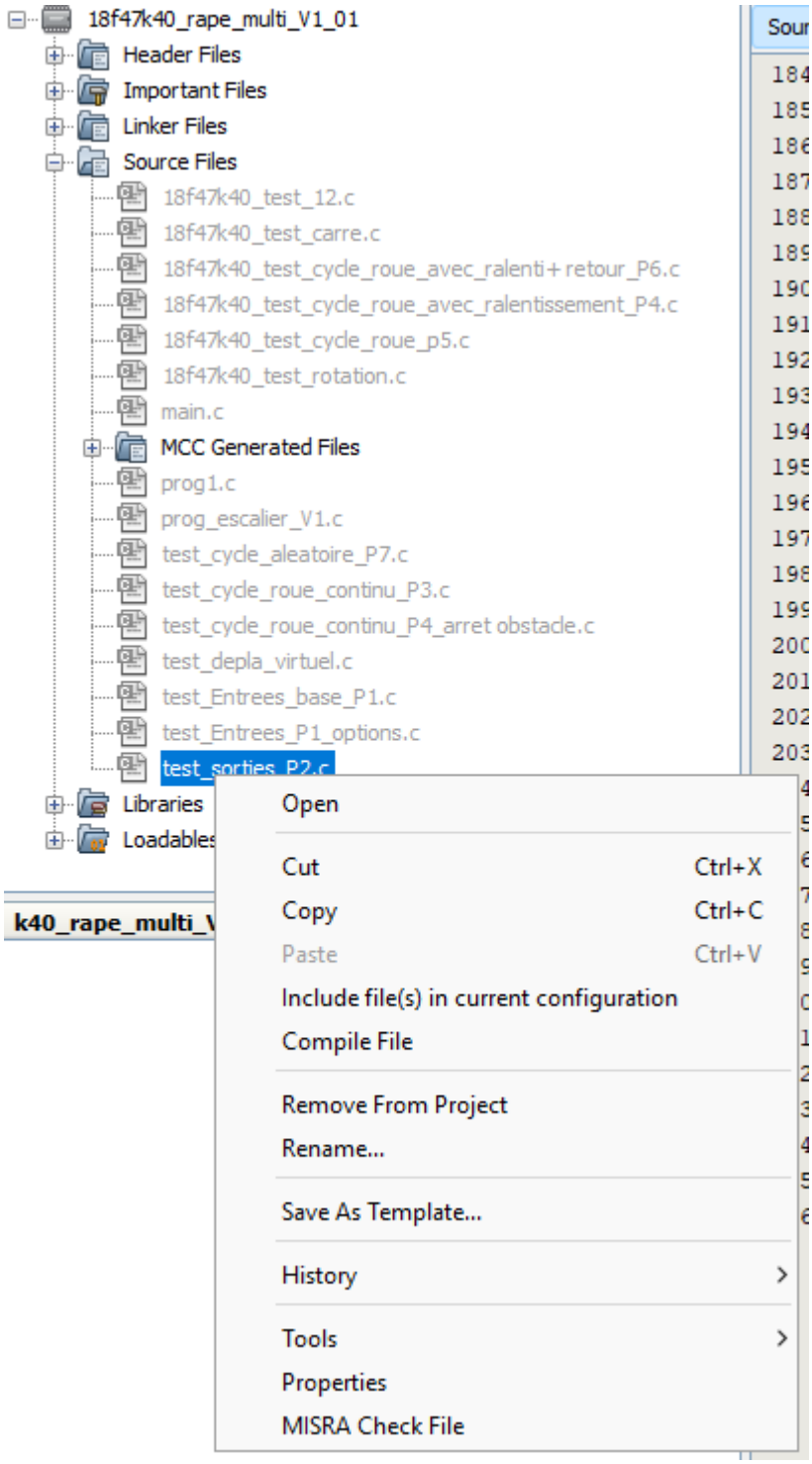
Présentation du programme

## Transfert du fichier dans le pic

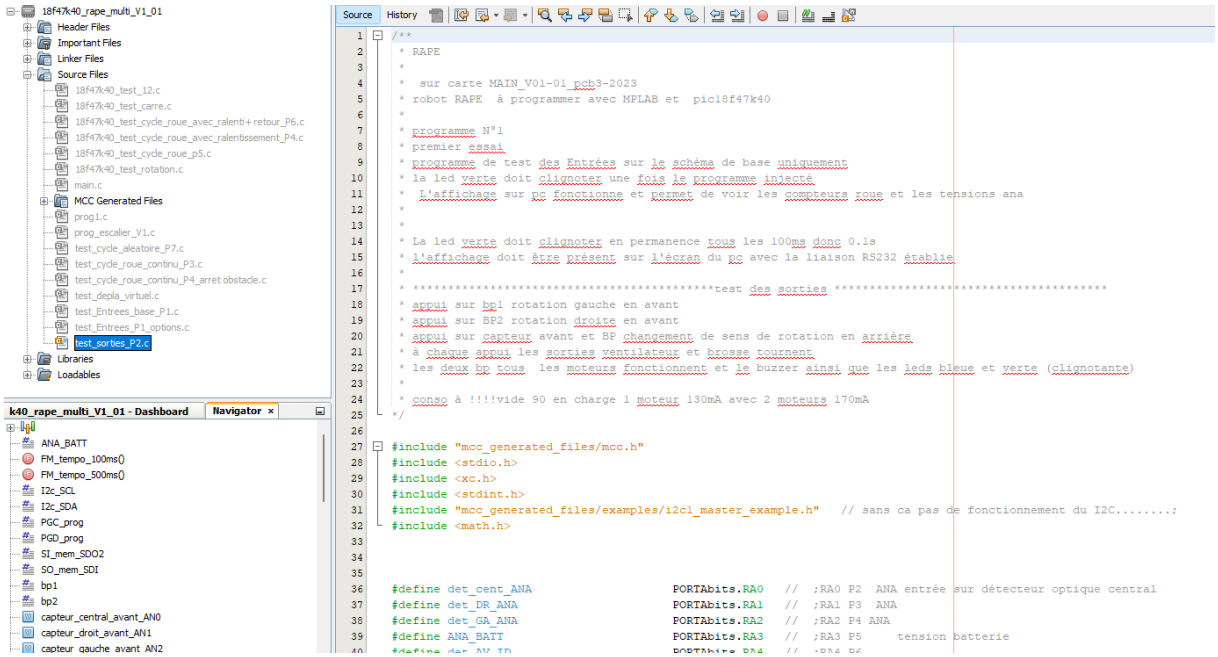
### Rappel rapide de la procédure de transfert dans le pic :

### choix du programme : 18f47k40_test_Sorties.c

Dans source Files, choisir le fichier « test_Sorties.c » en cliquant sur le bouton droit de la souris puis « include file… »



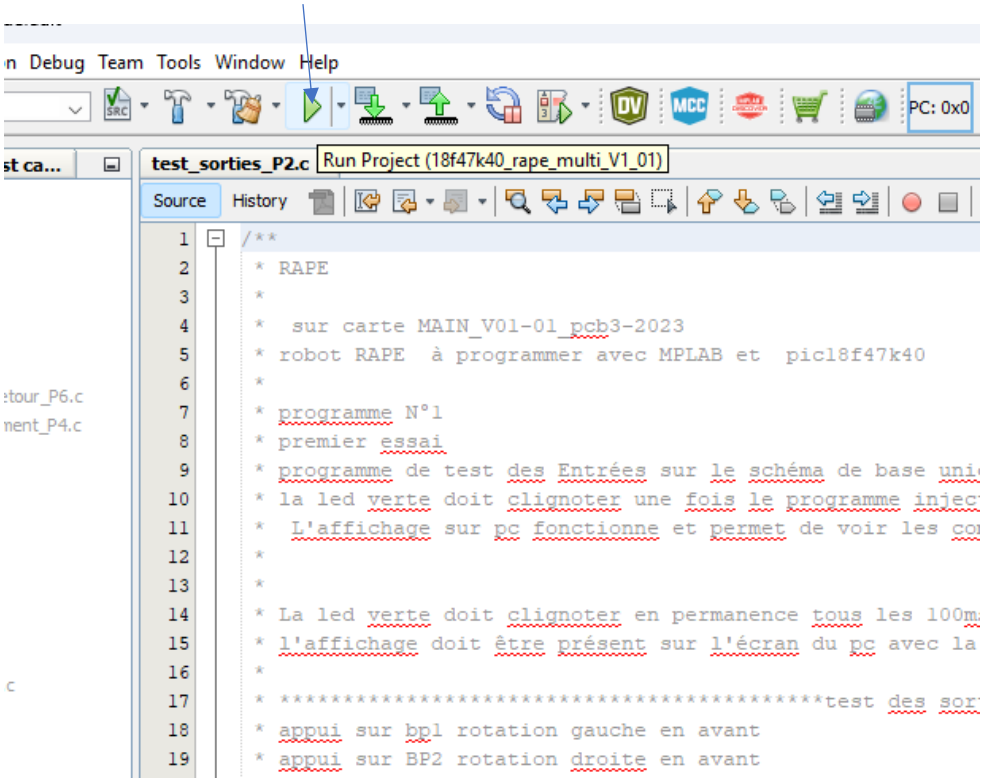

Le fichier passe en gras.



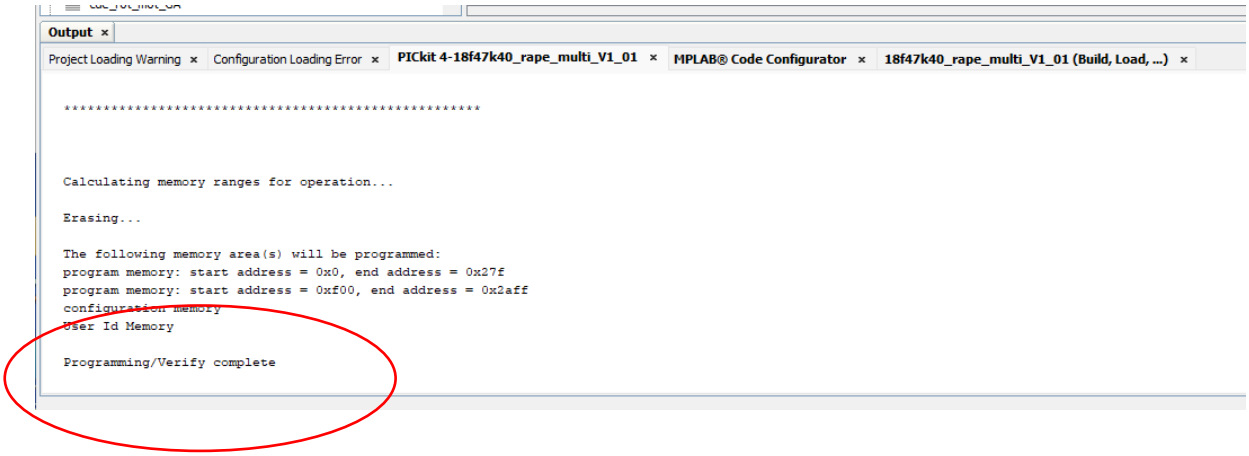

On clique deux fois dessus pour qu'il s'ouvre dans la fenêtre de droite.

## Transfert dans le pic

Lancer le transfert à l'aide du pickit3 ou 4 puis cliquez sur la flèche verte



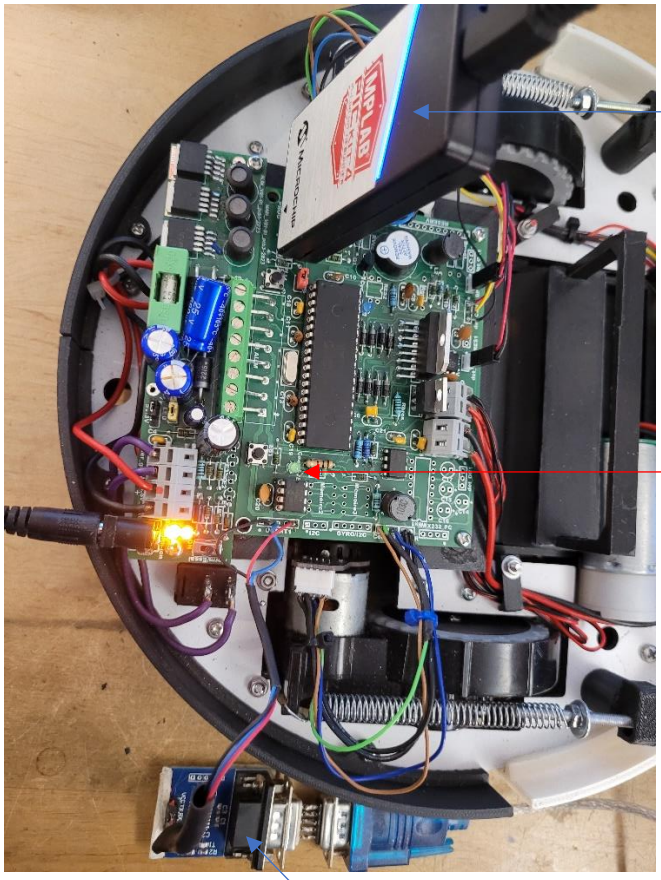

Si le transfert se passe bien vous devez voir la fenêtre suivante apparaître :




**Toujours vérifier que la led de cœur (verte) clignote suite à un transfert.**

## Connexion RS232 pour lire les informations venant du pic

On branche la liaison au pc (UART1) et boitier de conversion.

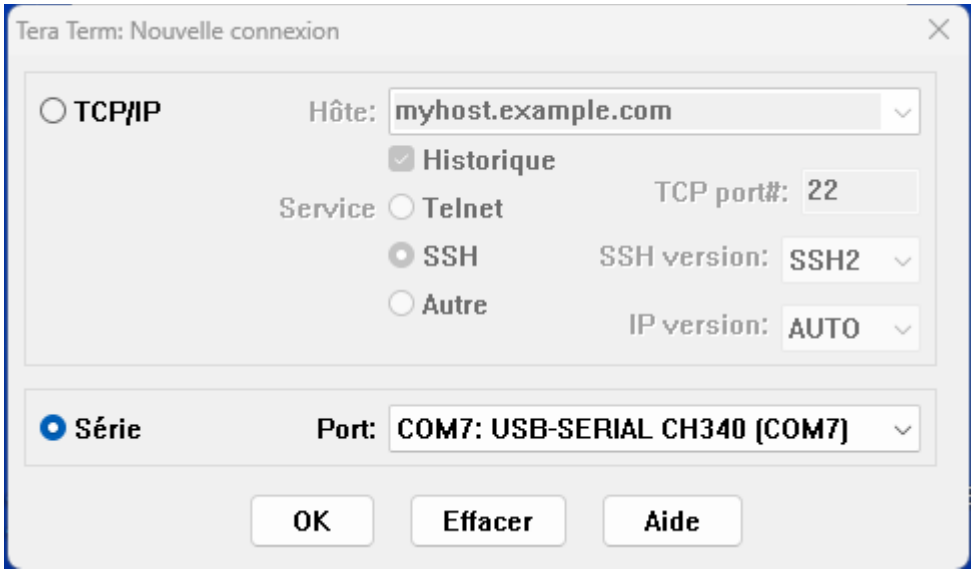


Pickit4 pour charger le programme

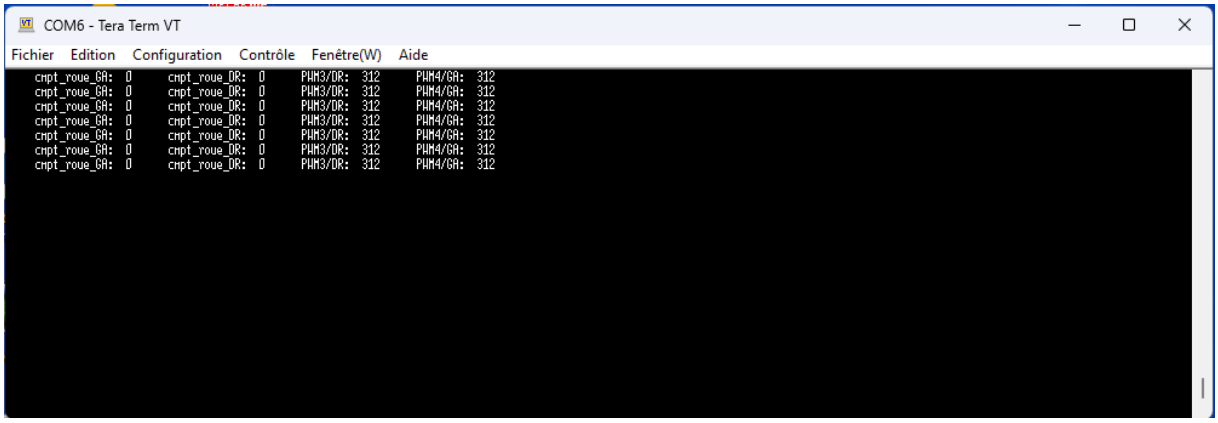
Led verte du coeur

Boitier interface pour lire les printf envoyés par le pic sur un écran de pc

## On lance TERATERM


Choix de la liaison : cliquez sur OK




Donc nous avons :

Compteur roue gauche (capteur qui est sur la roue)

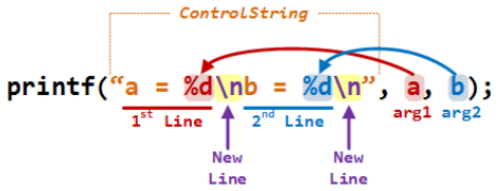
Compteur roue droite (même chose)

et les valeurs de PWM DRoit et GAuche (hacheur de 0 à 312).

```
if (fm_bit_500ms)
{
        printf("    cmpt_roue_GA:  %d " , cmpt_roue_GA);//
        printf("    cmpt_roue_DR:  %d " , cmpt_roue_DR);
        printf("    écart roue :   %d  " , ecart_roue);

        printf("    PWM3/DR:  %d " , duty_cycle_DR);//
        printf("    PWM4/GA:  %d " , duty_cycle_GA);//
        printf("\r\n");    // pour revenir en début de ligne et descendre d'une ligne
}
```

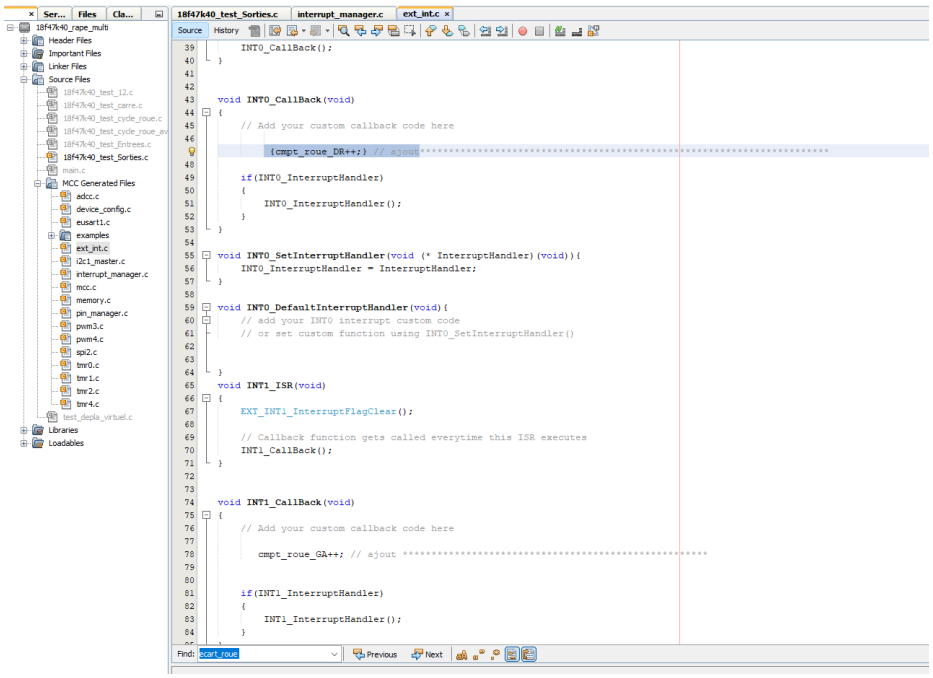
Fonctionnement du Printf :

https://microchipdeveloper.com/tls2101:printf



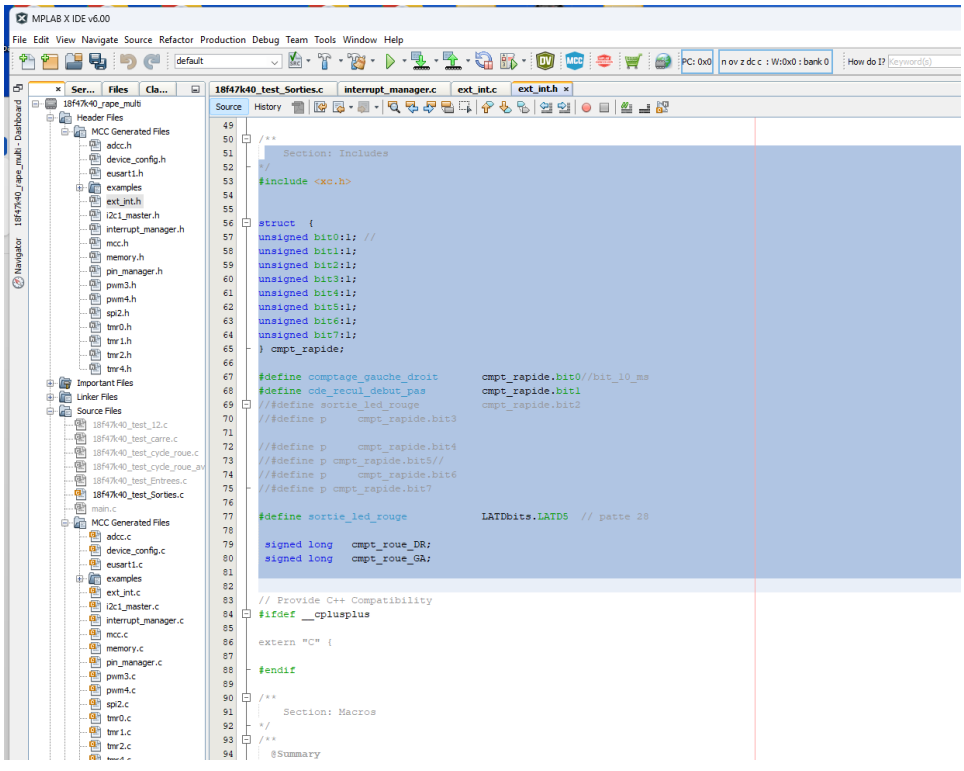

| | |
|---|---|
| %c | Single character |
| %s | String (all characters until '\0') |
| %d | Signed decimal integer |
| %o | Unsigned octal integer |
| %u | Unsigned decimal integer |
| %x | Unsigned hexadecimal integer with lower case digits (e.g. 1a5e) |
| %X | Same as %x but with upper case digits (e.g. 1A5E) |
| %f | Signed decimal value (floating point) |
| %e | Signed decimal value with exponent (e.g. 1.26e-5) |
| %E | Same as %e but uses upper case E for exponent (e.g. 1.26E-5) |
| %g | Same as %e or %f, depending on size and precision of value |
| %G | Same as %g but will use capital E for exponent |

La variable cmpt_roue_GA (et droite) sont traitées en tache interruption et c'est là qu'elles sont « sommées »



```
    INT0_CallBack();
}


void INT0_CallBack(void)
{
    // Add your custom callback code here

        {cmpt_roue_DR++;} // ajout**********************************************************************

    if(INT0_InterruptHandler)
    {
        INT0_InterruptHandler();
    }
}

void INT0_SetInterruptHandler(void (* InterruptHandler)(void)){
    INT0_InterruptHandler = InterruptHandler;
}

void INT0_DefaultInterruptHandler(void){
    // add your INT0 interrupt custom code
    // or set custom function using INT0_SetInterruptHandler()


}
void INT1_ISR(void)
{
    EXT_INT1_InterruptFlagClear();

    // Callback function gets called everytime this ISR executes
    INT1_CallBack();
}


void INT1_CallBack(void)
{
    // Add your custom callback code here

        cmpt_roue_GA++; // ajout ******************************************************

    if(INT1_InterruptHandler)
    {
        INT1_InterruptHandler();
    }
```

Lors de la création du projet on se fait aider par MCC qui va créer toute la structure en fonction de nos choix de fonctionnement (timer, eusart, analogique, spi, i2c….) dans les fichiers qui sont créer nous pouvons ajouter du code mais lorsqu'on utilise des variables il faut qu'elles soient déclarées dans le répertoire « Header Files/MCC Generated Files/ ext_int.h»



```
/**
    Section: Includes
*/
#include <xc.h>


struct  {
unsigned bit0:1; //
unsigned bit1:1;
unsigned bit2:1;
unsigned bit3:1;
unsigned bit4:1;
unsigned bit5:1;
unsigned bit6:1;
unsigned bit7:1;
} cmpt_rapide;

#define comptage_gauche_droit       cmpt_rapide.bit0//bit_10_ms
#define cde_recul_debut_pas         cmpt_rapide.bit1
//#define sortie_led_rouge          cmpt_rapide.bit2
//#define p      cmpt_rapide.bit3

//#define p      cmpt_rapide.bit4
//#define p cmpt_rapide.bit5//
//#define p      cmpt_rapide.bit6
//#define p cmpt_rapide.bit7

#define sortie_led_rouge            LATDbits.LATD5  // patte 28

 signed long   cmpt_roue_DR;
 signed long   cmpt_roue_GA;


// Provide C++ Compatibility
#ifdef __cplusplus

extern "C" {

#endif

/**
    Section: Macros
*/
/**
  @Summary
```

Revenons sur nos sorties :

```
void mot_gauche_manu(void)//
        {

        cde_sortie_mot_IN3_recul_GA=1;//à 1 pour freiner le moteur en fin de mouvement
        cde_sortie_mot_IN4_avance_GA =1; //à 1 pour freiner le moteur en fin de mouvement
        duty_cycle_GA = 254;// à 254 pour freiner le moteur en fin de mouvement


    if   (cde_rot_mot_GA ) {cde_sortie_mot_IN3_recul_GA=0 ;cde_sortie_mot_IN4_avance_GA =1 ;duty_cycle_GA = 254 ;} //
                            //else  sortie_mot_IN1_avance_GA=0;//hacheur_mot_gau=0,

    if   (cde_rot_mot_inv_GA )    {cde_sortie_mot_IN3_recul_GA=1 ;cde_sortie_mot_IN4_avance_GA =2 ;duty_cycle_GA = 254 ;} //   hacheur_mot_gau =1,
                            //else  sortie_mot_IN2_recul_GAU=0;//hacheur_mot_gau=0,

        }
```

## Traduction des mnémoniques :

cde_rot_mot_GA : commande de la rotation du moteur gauche (avance)

cde_rot_mot_inv_GA : commande de la rotation du moteur inverse gauche (recul)

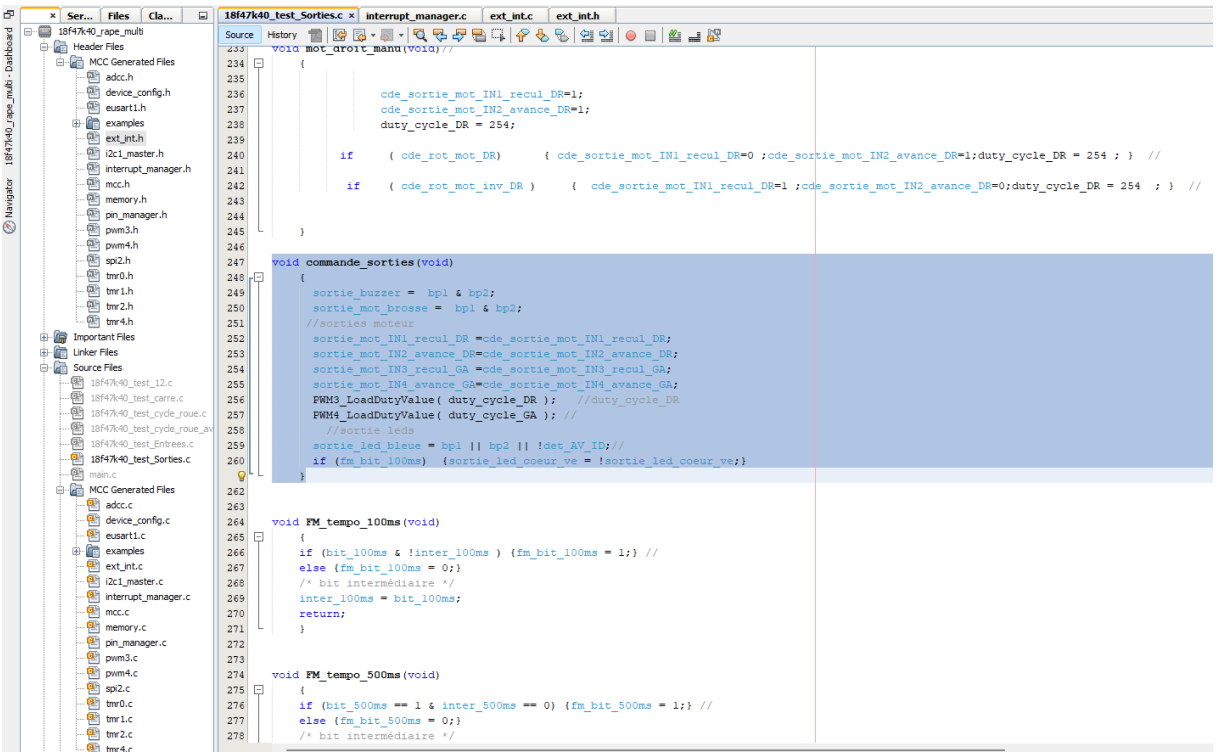
## Explications de la fonction :

Nous mettons toutes les entrées à 1 mais immédiatement après on active les sorties à 0 ou 1, sachant que nous passons par des bits internes sans agir directement sur la sortie donc il ne se passe rien sur la sortie directement.

*If (cde_rot_mot_GA ) {cde_sortie_mot_IN3_recul_GA=0 ;cde_sortie_mot_IN4_avance_GA =1 ;duty_cycle_GA = 254 ;} //*

Lors de la commande « *cde_rot_mot_GA* » on active le bit de sortie « *cde_sortie_mot_IN4_avance_GA* » et on désactive le *« bit de sortie cde_sortie_mot_IN4_avance_GA »*

Ensuite pour que la sortie soit activée :




Exemple suivi de la précédente commande :

*« sortie_mot_IN1_recul_DR =cde_sortie_mot_IN1_recul_DR; »*

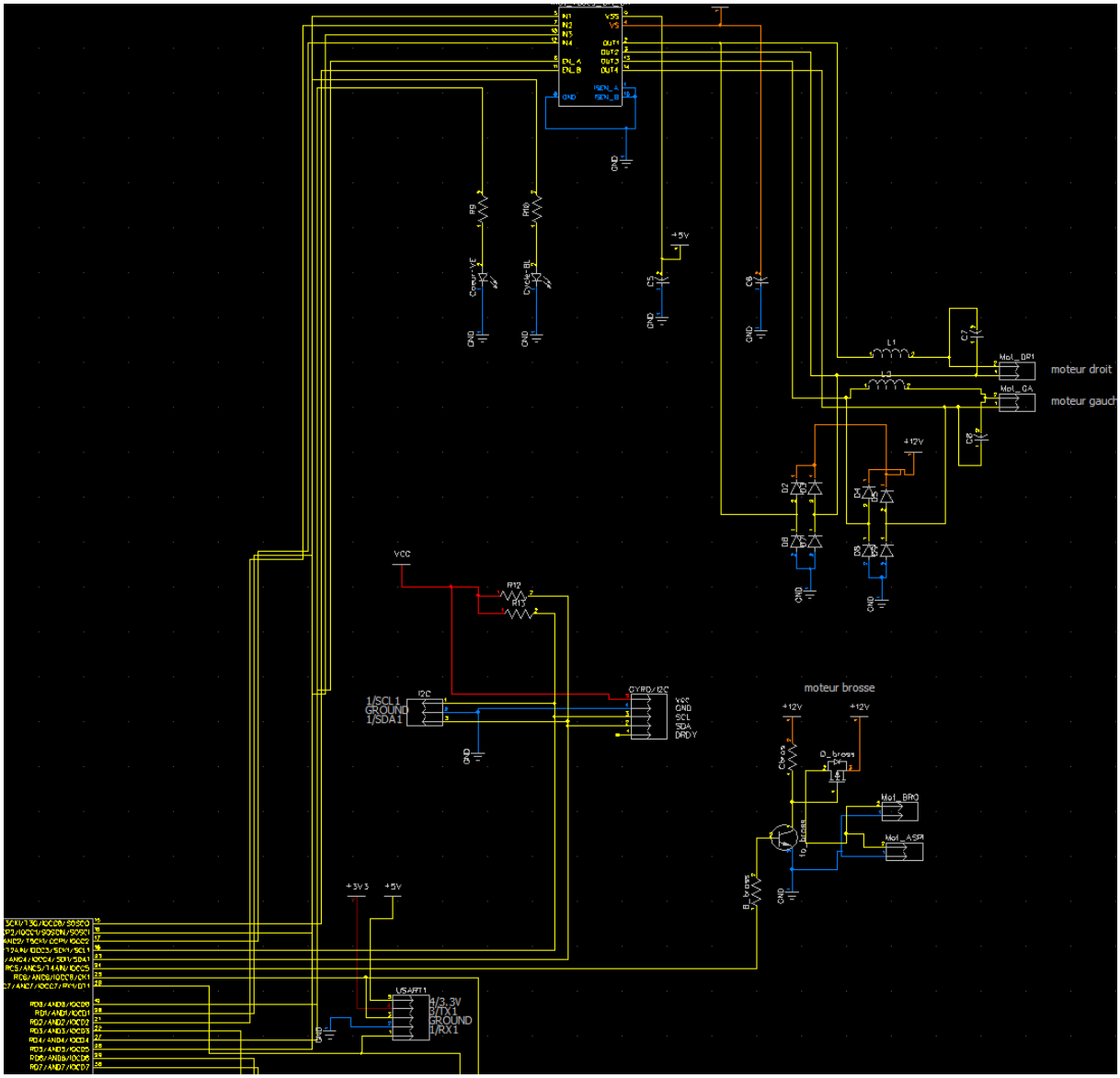
**sortie_mot_IN1_recul_DR ← cde_sortie_mot_IN1_recul_DR;**

le bit de sortie commandé dans les lignes précédentes s'écrit dans la sortie.

**Rappel :** pour éviter des dysfonctionnements, il est impératif de mettre une seule commande de sortie, ce qui n'empêche pas de faire plusieurs commandes.

## PWM

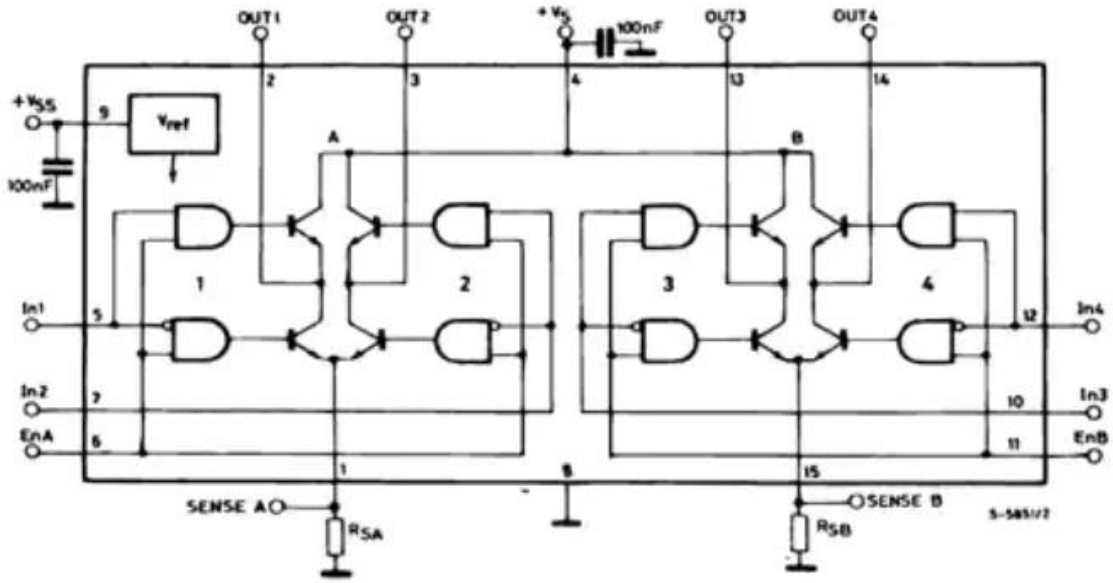
### Schéma :




### LM298N

Les sorties du pic attaque directement l'entrée du L298N

Nous avons deux ponts : à gauche commande du moteur de droite et le pont de droite moteur gauche.




## Les pattes sur le PIC

IN1 est liée à RD1 la patte 20 commande du moteur droit

IN2 RD2 patte 21 commande moteur droit

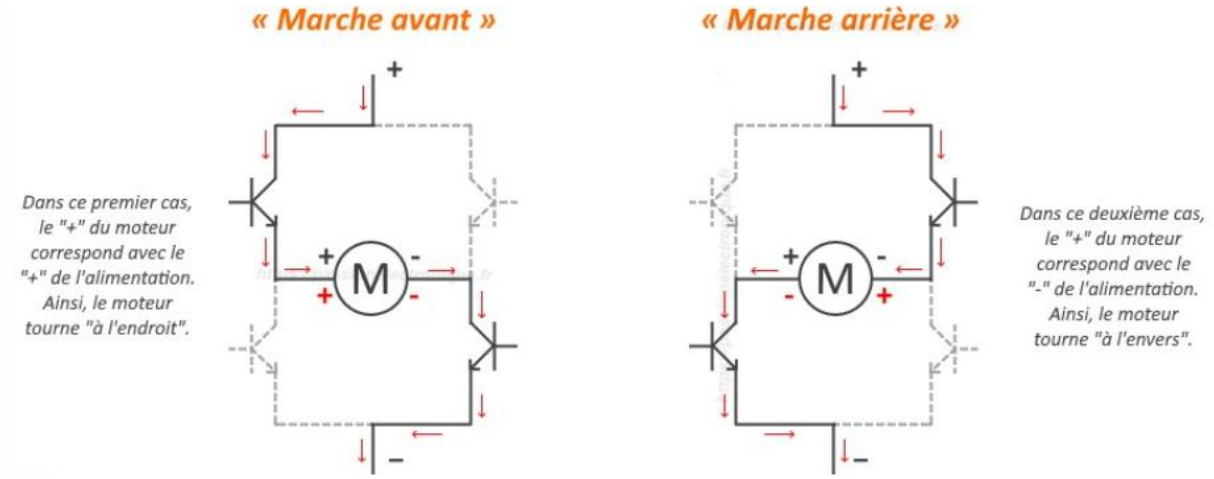
IN3 RC1 patte 16 commande moteur gauche

IN4 RC2 patte 17 commande moteur gauche

INA RD0 patte 19 PWM du moteur droit

INB RC0 patte 15 PWM du moteur gauche

on commande le moteur suivant le principe qui suit




## Freinage

Pour freiner le moteur on met le PWM au maxi et les deux to du haut ou du bas.

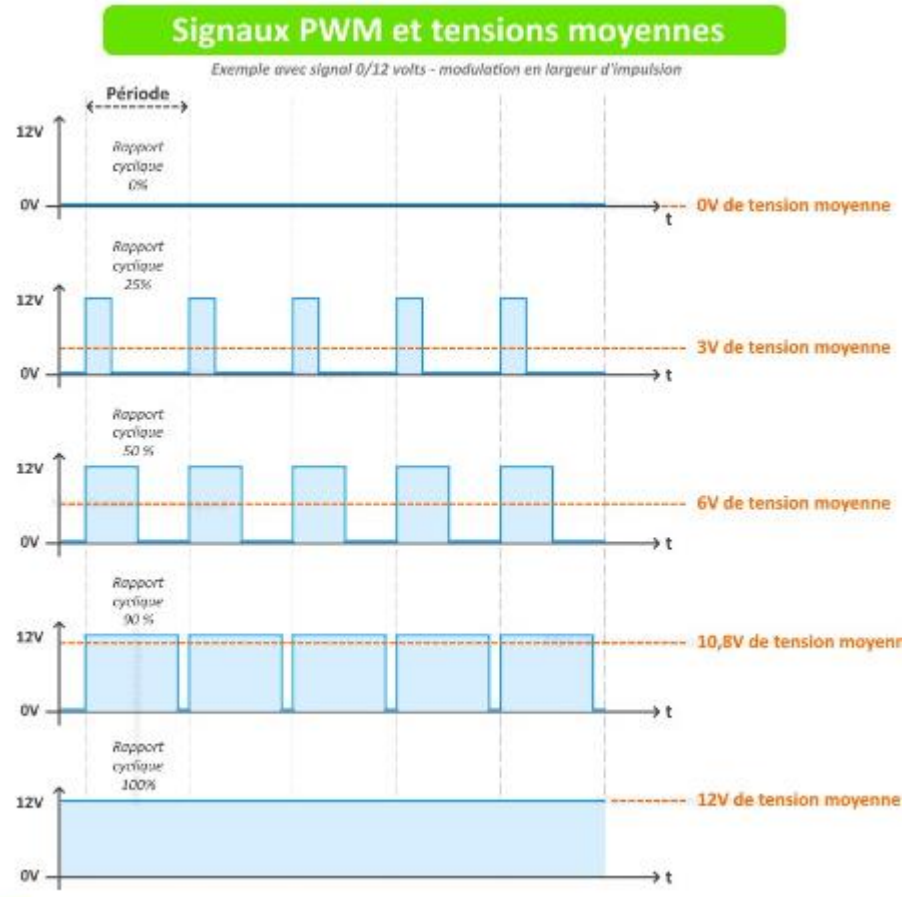
Les entrées INA et INB nous permettent de faire varier le rapport cyclique comme suit :

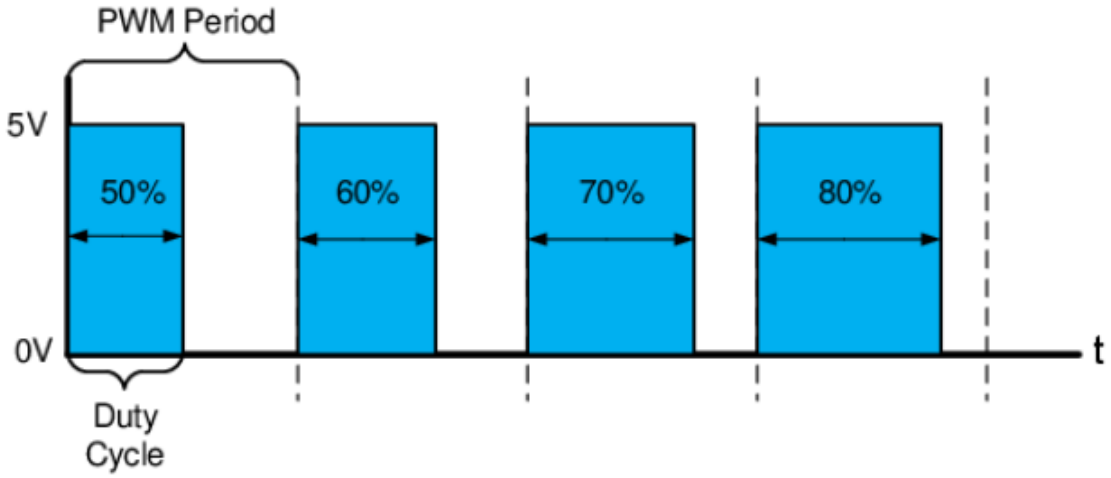
## Configuration et exploitation du pwm sur mplab

Pour info comment transformer une PWM en sortie ana….

https://ww1.microchip.com/downloads/en/Appnotes/90003250A.pdf

Nous avons sur les moteurs encore une petite chose le PWM. Nous envoyons des impulsions qui finissent par donner une tension moyenne voici les signaux envoyés :






## Le PWM est configuré sur 10bits.

Voici les valeurs issues de mesures :

| | |
|---|---|
| 100 % | 312 |
| 96 % | 300 |
| 93 % | 290 |
| 89 % | 280 |
| 83 % | 260 |
| 80 % | 250 |
| 76 % | 240 |
| 70 % | 220 |
| 64 % | 200 |
| 57 % | 180 |
| 50 % | 156 |

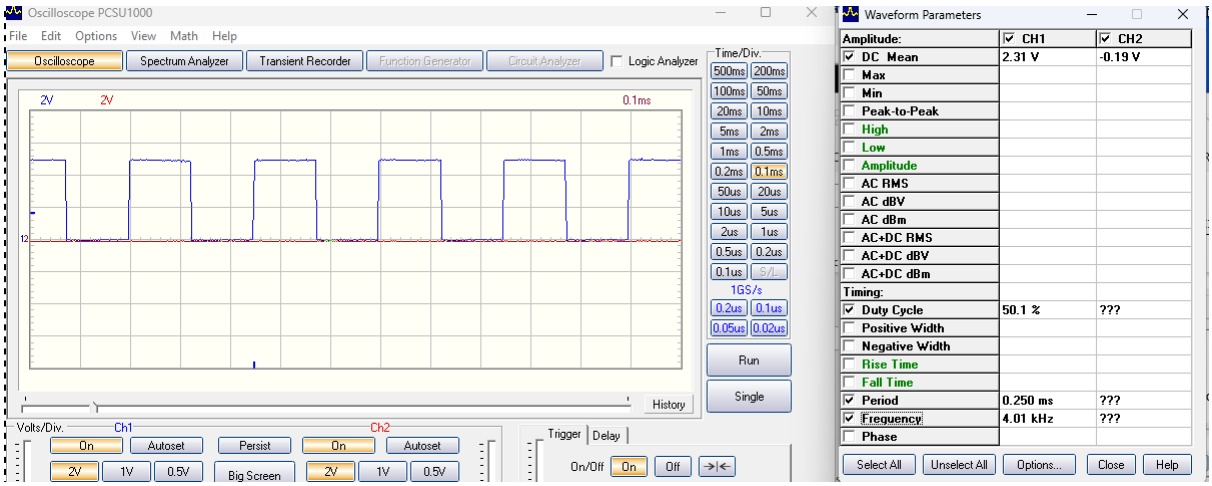
La valeur du PWM fixé par deux registres utilisé par le pic (voir datasheet).

```
PWM3DCH = (dutyValue & 0x03FC)>>2;      // 8 MSBs of PWM duty cycle
PWM3DCL = (dutyValue & 0x0003)<<6;      // 2 LSBs of PWM duty cycle
PWM3DCH = 0x27;                         // DC = 50%
PWM3DCL = 0xC0;
```

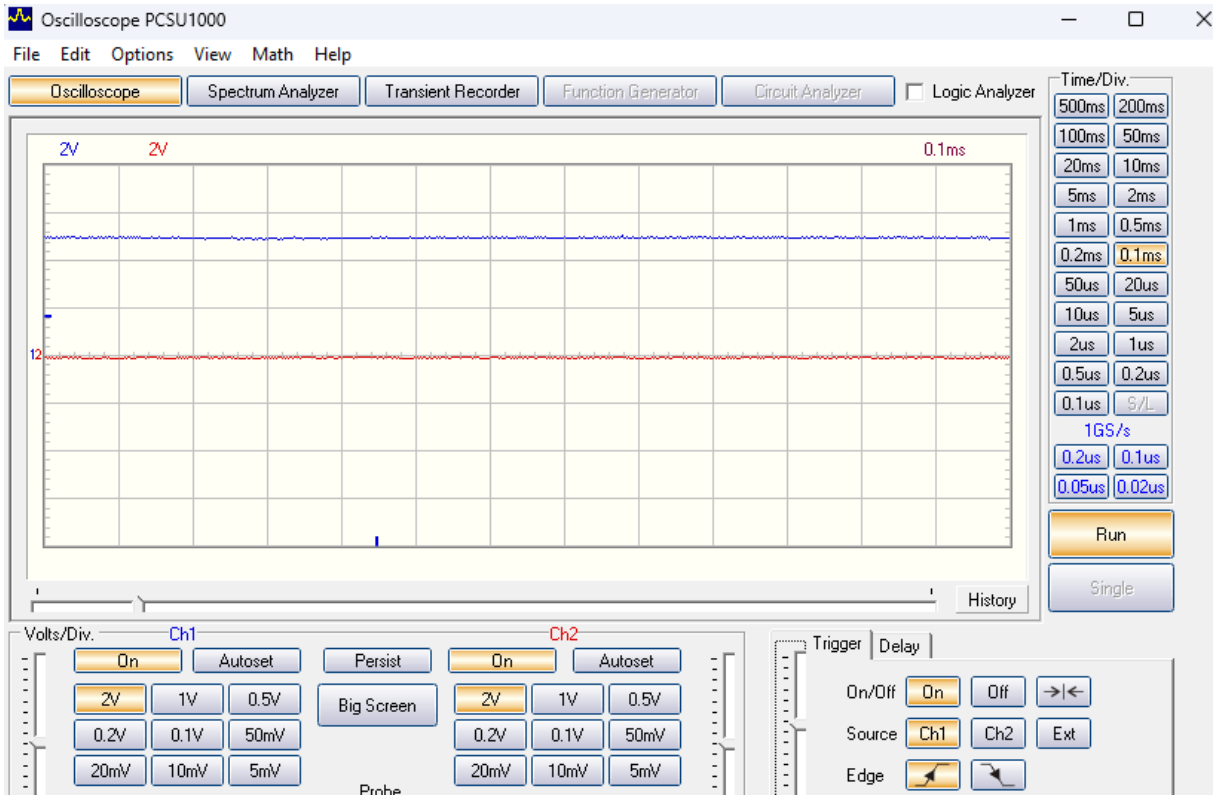
Nous avons dans ce cas une fréquence qui ne change pas, de 4khz réglé par le timer 2 et un duty cycle (rapport cyclique) est de 50% avec les valeurs ci-dessus mais dans le programme nous le ferons changer selon nos besoins.

**Attention** les valeurs affichées sont prises sur la patte du pic donc on sera en 0 à 5v, il faut penser que le signal est amplifié jusqu'à 12v dans le lm.

A 156 duty_cycle 50% (la fréquence reste inchangée de 4khz




Enfin à 312 on a 100%




Sachant qu'en dessous de 180 la tension n'est plus suffisante pour entrainer le robot.

Revenons au programme :

```
void mot_gauche_manu(void)//
        {

        cde_sortie_mot_IN3_recul_GA=1;//à 1 pour freiner le moteur en fin de mouvement
        cde_sortie_mot_IN4_avance_GA =1; //à 1 pour freiner le moteur en fin de mouvement
        duty_cycle_GA = 310;// à 300 pour freiner le moteur en fin de mouvement


    if   (cde_rot_mot_GA ) {cde_sortie_mot_IN3_recul_GA=0 ;cde_sortie_mot_IN4_avance_GA =1 ;duty_cycle_GA = 260 ;} //
                            //else  sortie_mot_IN1_avance_GA=0;//hacheur_mot_gau=0,

    if    (cde_rot_mot_inv_GA )     {cde_sortie_mot_IN3_recul_GA=1 ;cde_sortie_mot_IN4_avance_GA =2 ;duty_cycle_GA = 260 ;} //
                            //else  sortie_mot_IN2_recul_GAU=0;//hacheur_mot_gau=0,


        }
//
```

Les valeurs sont écrites au moment ou l'on en a besoin elle écrase la valeur du dessus, elle n'aura d'effet que lorsqu'elle sera envoyée dans le PWM sur la fonction de sortie (void).

**Exemple** quand on a aucune sortie activée on envoi 312 donc 12v sur le moteur alors qu'un mouvement écrase cette valeur et place 260 83% donc une tension moyenne de 10V.

**Commande d'un moteur :**

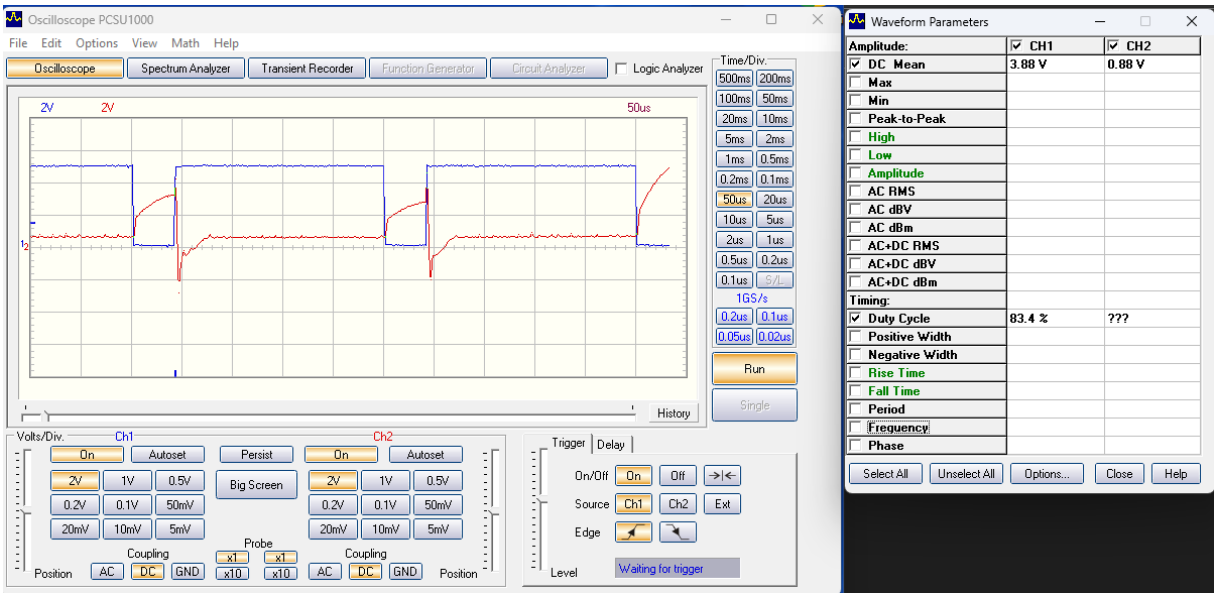
Dans le programme principal nous utiliserons toujours la commande « cde_rot_mot_GA » dans les commandes supérieures.

**Valeur moyenne choisie pour nos essais**

Après quelques tests**, 260** en décimal est la valeur qui est suffisante pour analyser les mouvements et avoir un couple suffisant pour que le robot se déplace sans blocage. (par exemple il peut monter sur un tapis).

Les signaux sur le moteur (rouge) et en sortie de Pic(bleu).

On peut voir les bobines se charger puis se décharger.


Oscilloscope PCSU1000
File Edit Options View Math Help
Oscilloscope
Spectrum Analyzer
Transient Recorder
Function Generator
Circuit Analyzer
Logic Analyzer
Time/Div.
2V
2V
50us
History
Run
Single
Volts/Div.
Ch1
Ch2
Persist
Big Screen
Probe
Coupling
Position
Trigger
Delay
On/Off
Source
Edge
Level
Waiting for trigger
Waveform Parameters
Amplitude:
CH1
CH2
DC Mean
3.88 V
0.88 V
Max
Min
Peak-to-Peak
High
Low
Amplitude
AC RMS
AC dBV
AC dBm
AC+DC RMS
AC+DC dBV
AC+DC dBm
Timing:
Duty Cycle
83.4 %
???
Positive Width
Negative Width
Rise Time
Fall Time
Period
Frequency
Phase
Select All
Unselect All
Options...
Close
Help