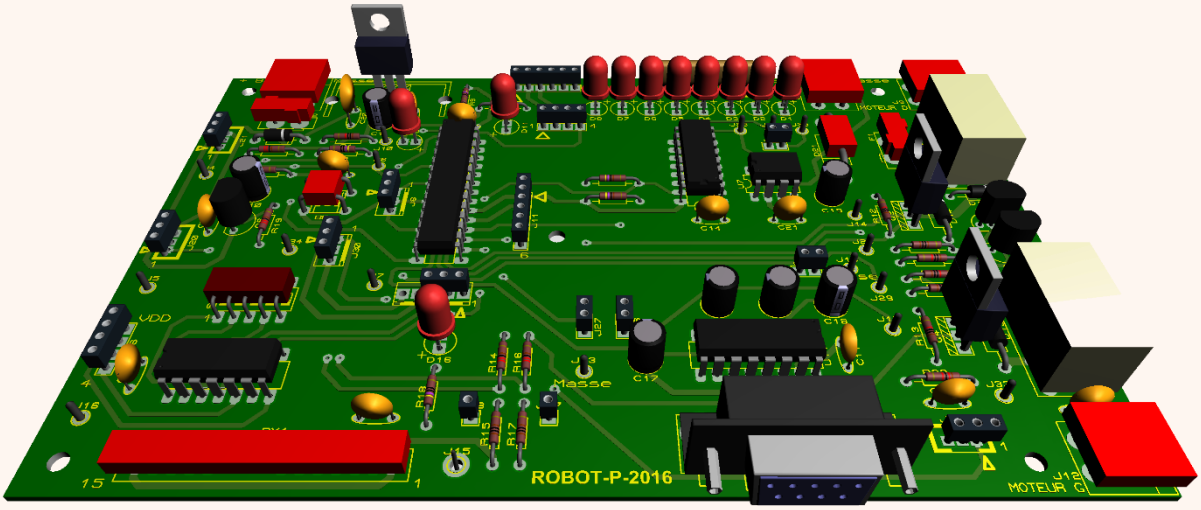
PROJET ROBOT



Lexane Cassaing & Rémi Boes

Table des matières

Table des matières 1
Introduction 2
Cahier des charges 2
Organigrammes 3
Main : 3
Initialisation : 4
Interruption : 4
Surveillance batterie : 5
Allumage des leds : 5
Programmation 6
Tests des codes avec MPLAB et Proteus 7
Test avancer et reculer à 5 vitesses 7
Test batterie 9
Résultats expérimentaux 10
Conclusion 11

Introduction

Durant la première partie de notre cursus, en projet robot, nous avons assemblé la carte électronique pour le robot. La suite du projet est donc de programmer la carte afin de répondre à un contrat spécifiant les fonctionnalités du robot. Notre groupe a reçu le contrat 4. Nous allons donc détailler dans ce rapport nos réalisations : nos codes, organigrammes et les simulations réalisées sur Proteus à l'aide de MPLAB.

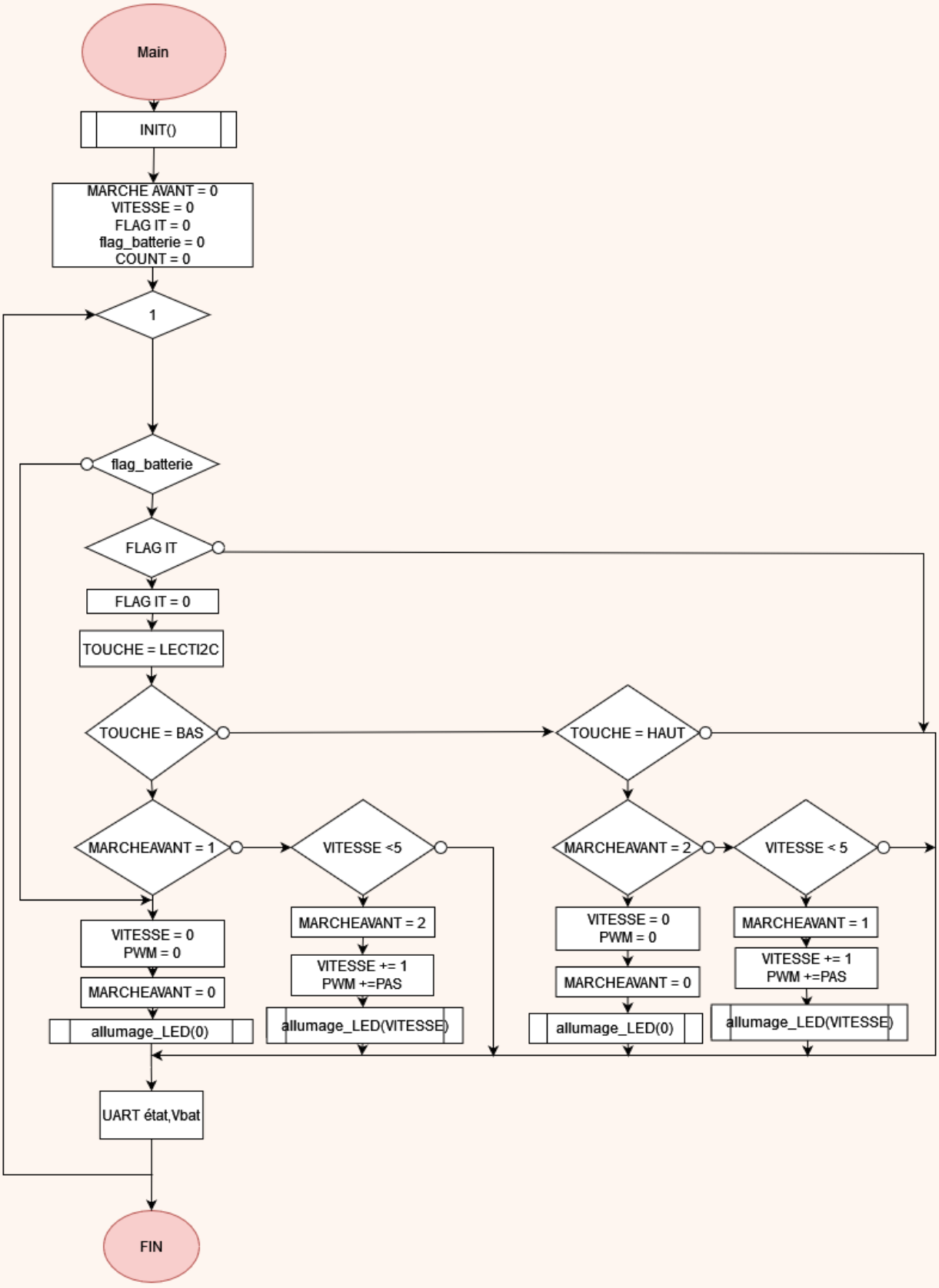
Cahier des charges

L'objectif du contrat 4 est de faire déplacer le robot en ligne droite, en marche avant ou en marche arrière. De plus, il est demandé 5 valeurs différentes de vitesse en marche avant et 5 en marche arrière. Les deux seuls boutons utilisés seront donc les boutons Avant et Arrière. A chaque appui sur un de ces boutons, le robot accélère ou décélère. Si le robot est en marche avant et que l'on active le bouton Avant, le robot doit accélérer. Cependant, s'il est en marche Avant et que l'on active le bouton Arrière, le robot doit s'arrêter et inversement. La vitesse maximale est de 14m/mn à +/- 10%.

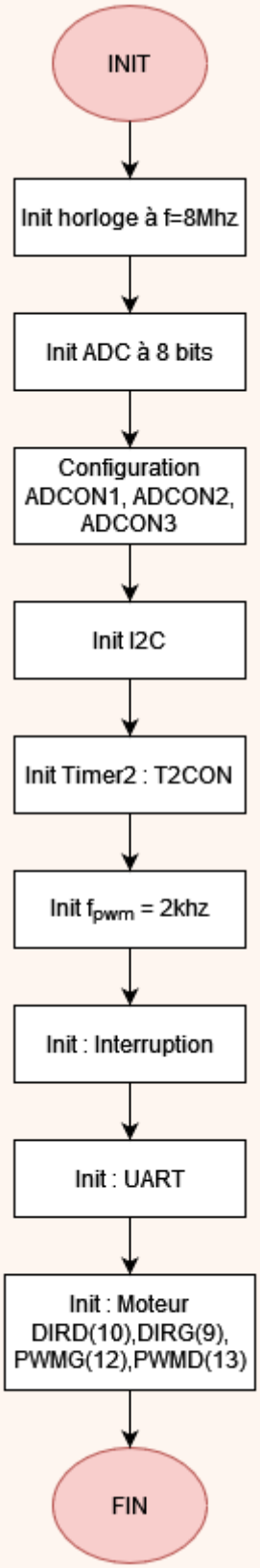
Le cahier des charges impose que la fréquence du microcontrôleur soit réglée sur 8 MHz. De plus, la surveillance de la batterie s'effectue grâce à une moyenne faite sur 4 mesures de tension de la batterie. En cas de batterie faible, inférieure à 2,5V, un message doit être adressé à l'utilisateur pour l'informer. Le microcontrôleur communique avec la télécommande grâce au bus I2C en master à 100Hz. Les moteurs droit et gauche sont régis par des PWM de fréquence 2kHz et les rapports cycliques sont inférieurs à 50%. Le rapport cyclique est géré par le Timer2. Le Timer2 servira aussi pour la chronométrie, utilisé avec un compteur pour la surveillance batterie et un autre pour éviter les rebonds lors de l'appui d'un bouton de la télécommande. Enfin, les LEDs seront utilisées pour informer l'utilisateur de la vitesse du robot ainsi que de l'état de la batterie.

Organigrammes

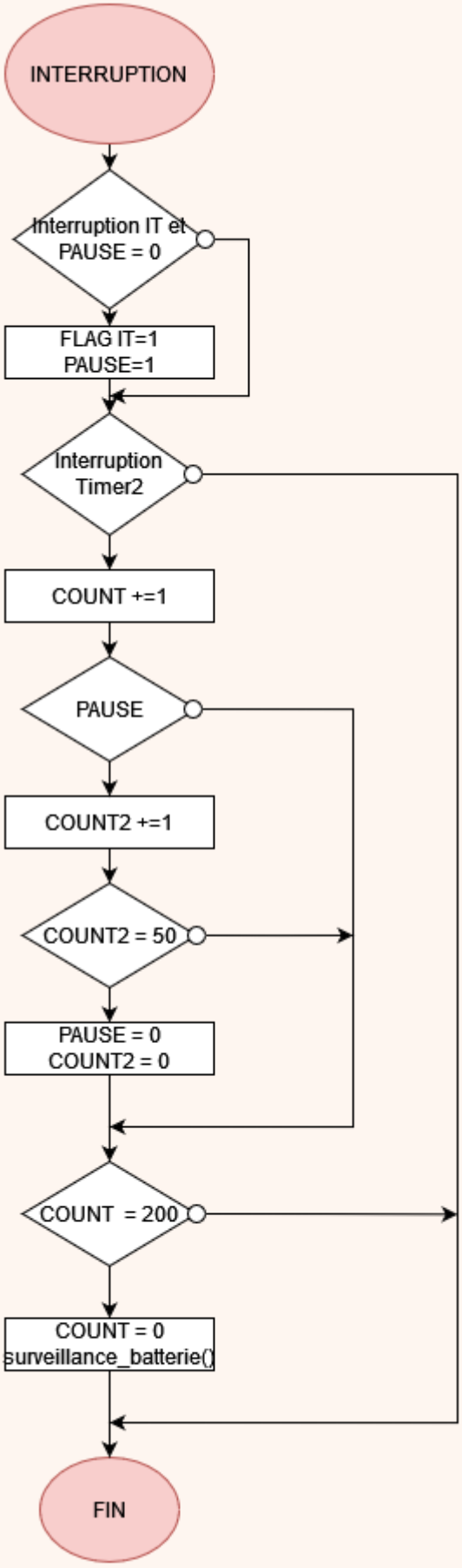
Main :



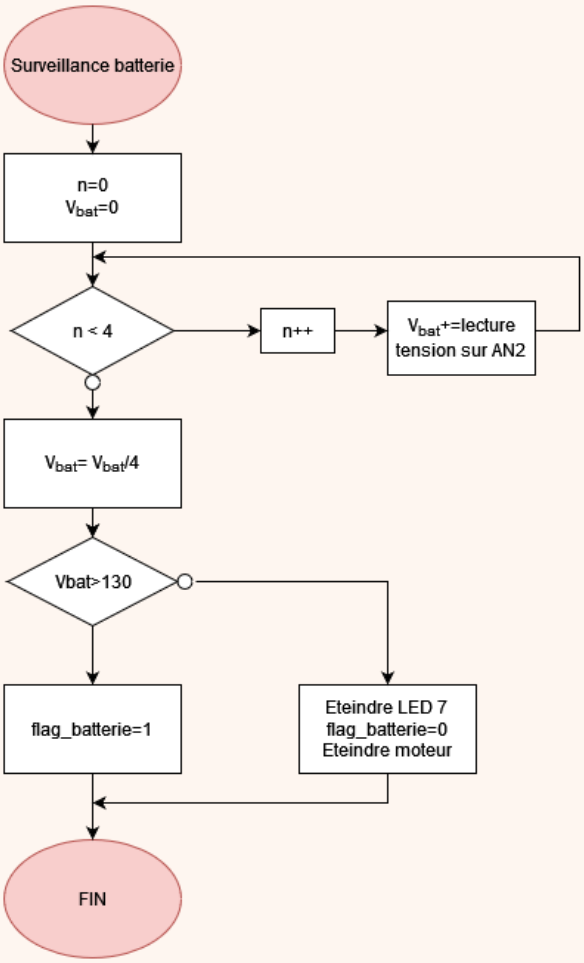
Initialisation :



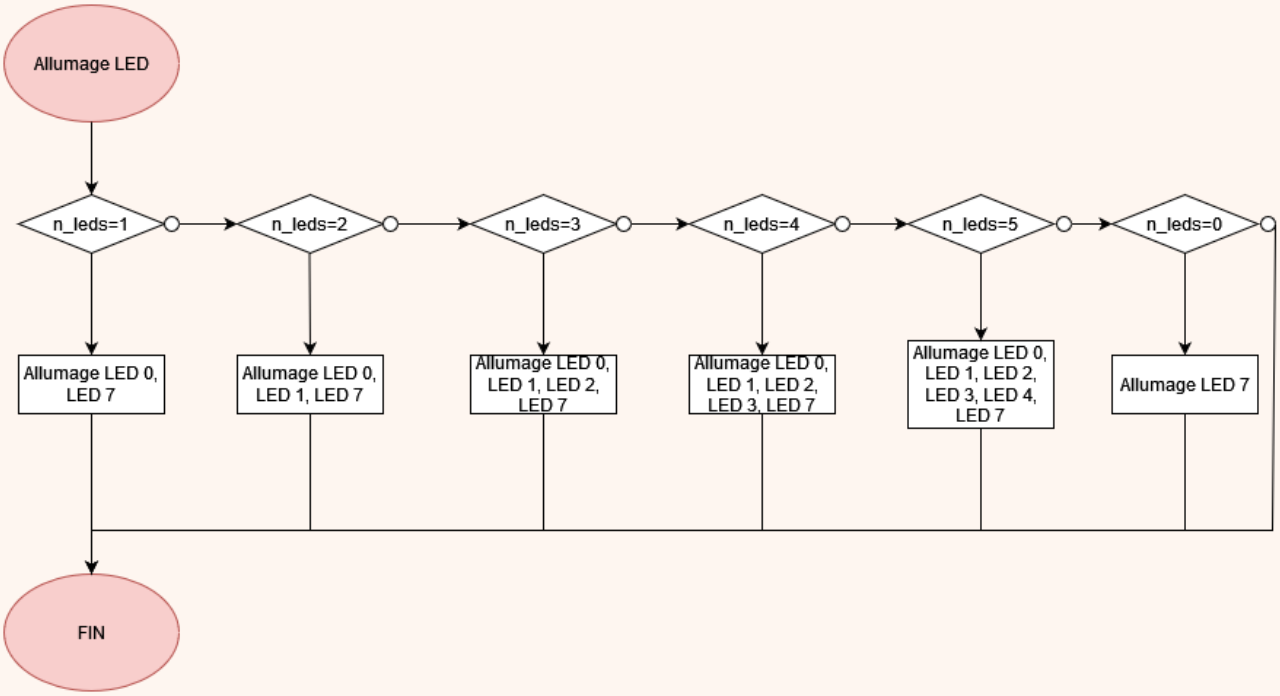
Interruption :



Surveillance batterie :



Allumage des leds :



Programmation

Le fichier init.c contient tous les initialisations nécessaires pour la carte électronique. Il permet de définir les ports d'entrées et de sorties, la fréquence de l'horloge interne, la configuration des ADCON, de l'UART, des moteurs, du Timer2, des interruptions.

Le fichier main.c regroupe les fonctions nécessaires au fonctionnement du robot que nous avons écrites. Il est accompagné de son header : main.h.

Dans main.c, il se trouve la fonction d'interruption de la télécommande HighISR(). Le count2 de cette fonction permet d'imposer une pause entre deux interruptions de la télécommande. Son utilité sera détaillée dans la partie expérimentale. Le compteur count permet d'effectuer le contrôle de la batterie en se basant sur le Timer2 et en n'ayant pas recourt à un Timer0 pour l'effectuer. Ce compteur permet de ne pas effectuer de manière trop rapide la surveillance de la batterie.

La fonction de surveillance de la batterie surveillance_batterie est aussi présente. Elle permet de vérifier le bon état de la batterie en faisant la moyenne de 4 valeurs de batterie consécutives. La fonction allumageled est celle qui gère l'allumage des LEDs. Enfin, dans main(), toutes les données sont initialisées grâce à l'appel à la fonction init(), puis la batterie est surveillée et enfin on entre dans une boucle infinie. Le code de la fonction main() suit la logique de l'organigramme.

Tests des codes avec MPLAB et Proteus

Test avancer et reculer à 5 vitesses

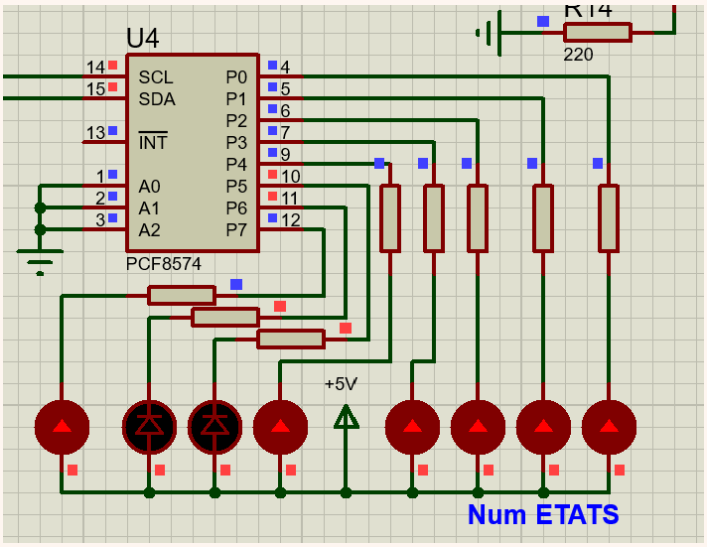
Les séquences suivantes ont été entrées dans le bus I2C :

```
{SEQUENCE000=A 0X31 0X32 0X00}
{SEQUENCE001=A 0X31 0X32 0X00}
{SEQUENCE002=A 0X31 0X32 0X00}
{SEQUENCE003=A 0X31 0X32 0X00}
{SEQUENCE004=A 0X31 0X32 0X00}
{SEQUENCE005=A 0X31 0X34 0X00}
{SEQUENCE006=A 0X31 0X34 0X00}
{SEQUENCE007=A 0X31 0X34 0X00}
{SEQUENCE008=A 0X31 0X34 0X00}
{SEQUENCE009=A 0X31 0X34 0X00}
{SEQUENCE010=A 0X31 0X34 0X00}
{SEQUENCE011=A 0X31 0X32 0X00}
```

Les séquences de 0 à 4 simulent les cinq vitesses du robot en marche arrière. A chaque appui, la vitesse du robot est incrémentée. La séquence 5 simule l'appui sur le bouton marche avant de la télécommande. Le robot avançant en marche arrière, il doit donc s'arrêter avec la séquence 5. Les séquences 6 à 10 simulent l'appui successif sur le bouton marche avant de la télécommande. Le robot augmentera donc en vitesse à chaque appui de bouton. Tout comme la séquence 5, la séquence 11 permet d'arrêter le robot.

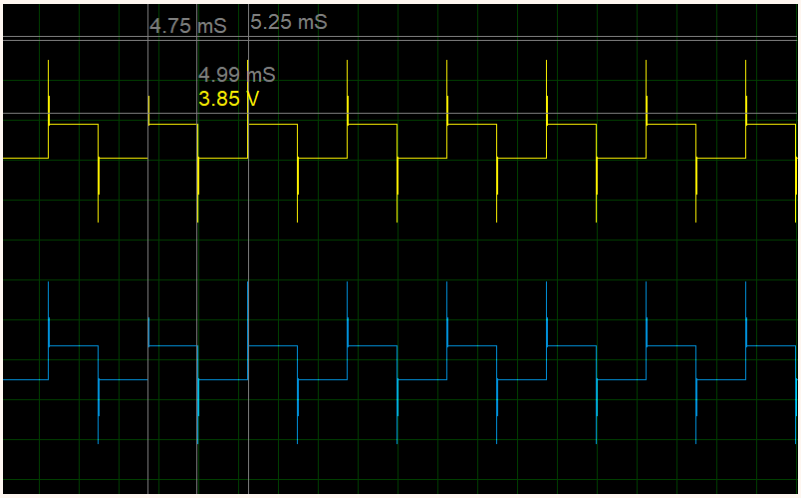
```
Fin init
Batterie 255
Batterie 255
Reculer 1
Reculer 2
Batterie 255
Reculer 3
Batterie 255
Reculer 4
Batterie 255
Reculer 5
Arret
Batterie 255
Avancer 1
Avancer 2
Avancer 3
Batterie 255
Avancer 4
Avancer 5
Batterie 255
Arret
```

L'affichage dans la console permet de suivre l'avancement du programme conjointement à ce qui est observée par la rotaion des moteurs dans la simulation. En effet, « Fin init » confirme le fait que l'initialisation a été réalisée. Les affichages batterie 255 seront expliqués par la suite. A l'appui du premier bouton, la séquence 0 est exécutée et d'après ce qui précède le robot recule à la vitesse 1. C'est bien ce qui est affiché dans la console : « Reculer 1 ». Toutes les séquences s'exécutent correctement. La séquence 5 qui permet d'arrêter le robot est bien fonctionnelle avec le premier « Arrêt » puis le robot avance conformément aux séquences entrées dans le bus I2C.



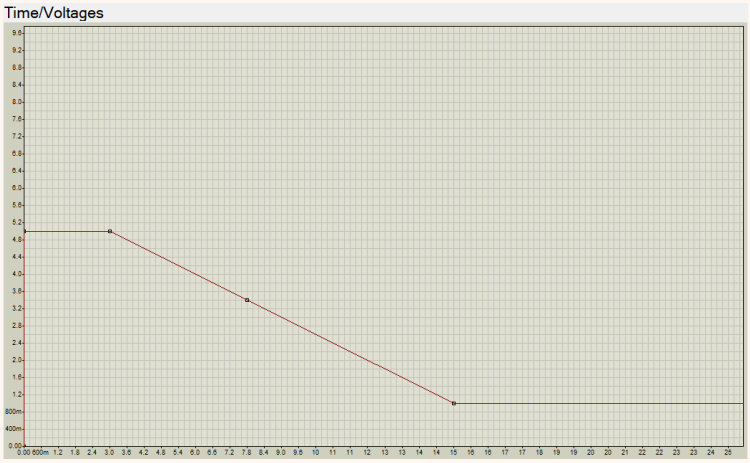
Proteus permet aussi de confirmer le bon fonctionnement de la fonction d'allumage des LEDs. La LED 8 est allumée lorsque la tension de la batterie est supérieure à 2,5V Elle s'éteint lorsque la batterie est faible.

Lorsque le robot est à la vitesse 1 que cela soit en marche avant ou en marche arrière, la première LED doit être allumée. Puis en vitesse 2, les deux premières LEDs doivent s'allumer et ainsi jusqu'à la vitesse 5. C'est bien ce qui se passe avec le test sur Proteus. Grâce à la séquence dans le bus I2C, on remarque que l'affichage des LEDs est correct que cela soit en marche avant ou arrière. Ici, se trouve un exemple d'allumage des LEDs lorsque le robot est en vitesse 5 soit à sa vitesse maximale et la batterie est bien fonctionnelle car la LED 8 est allumée.

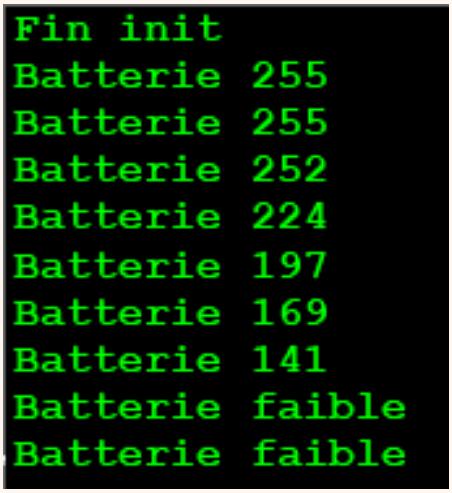


On visualise les PWM sur l'oscilloscope. On observe déjà que la période de 0,5ms est respectée pour les 5 vitesses. De plus, lorsque la vitesse est maximale, le rapport cyclique des moteurs ne dépasse pas 50%.

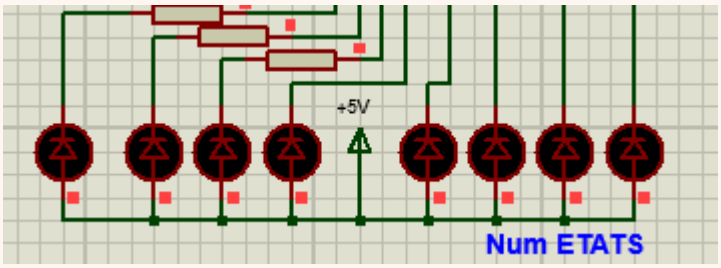
Test batterie



On choisit cette courbe de tension pour tester la batterie et savoir si l'on a bien un message d'alerte lorsque la batterie est déchargée (tension inférieure à 2,5V), conformément à notre code. La tension varie de 5V à 1V avec une chute de tension à partir de 3s.



L'information « Batterie 255 » dans le terminal indique que la batterie délivre une tension de 5V. Ceci est bien conforme à la courbe précédente. La surveillance batterie s'effectue une première fois après l'initialisation puis dès que la première interruption a eu lieu. Elle se base sur le Timer2 mais utilise un compteur afin qu'elle ne soit pas faite trop fréquemment et permet ainsi d'économiser de la batterie. A chaque nouvelle surveillance_batterie(), la tension de la batterie est indiquée sur 8 bits. Lorsque la batterie a une valeur inférieure à 127 (soit 2,5V), l'affichage dans le terminal devient « Batterie faible » en accord avec la fonction surveillance_batterie(). Lorsque la batterie est faible tous les moteurs s'arrêtent.



Lorsque la batterie est faible, toutes les LEDs s'éteignent ainsi que la LED 8 qui indique l'état de la batterie.

Résultats expérimentaux

En ce qui concerne les résultats expérimentaux, ils ont été satisfaisants. L'allumage des LEDs étaient bien en accord avec les stades de vitesse imposés au robot. De plus, nous avons pu mesuré expérimentalement la vitesse maximale du robot. Elle était de 14,3m/min. Le cahier des charges impose 14m/min comme vitesse maximale à +/- 10%. Le cahier des charges est donc bien respecté.

Expérimentalement, nous nous sommes rendus compte que si l'appui d'un bouton n'était pas suffisamment rapide (on reste appuyé dessus), le robot agissait comme si plusieurs interruptions par la télécommande avaient été activée et donc plusieurs vitesses du robot passaient . Pour résoudre ce léger problème, nous avons implémenté un compteur dans notre interruption HighISR() qui impose un temps de pose entre deux interruptions reçu par la télécommande. Nous avons réglé expérimentalement une valeur qui nous semblait adéquate en utilisant le Timer2. Ce compteur règle le problème d'interruptions successives.

Conclusion

Le projet robot a été enrichissant car il a permis d'expérimenter toutes les étapes de conception d'une carte électronique, de son assemblage à sa programmation puis à son expérimentation concrète tout en respectant un cahier des charges précis. Ce projet nous a aussi permis d'expérimenter de nombreuses fonctionnalités de MPLAB. Être parvenu à finir ce projet, nous a procuré une satisfaction et nous donne le sentiment d'avoir acquis des compétences en lien avec notre avenir en tant qu'ingénieurs.

Annexe

init.c

```
#include "MI2C.h"
#include <stdlib.h>
#include<p18f2520.h>
#pragma config OSC = INTIO67
#pragma config WDT = OFF, LVP = OFF, DEBUG = ON

void init(void) {
  //Clock de 8Mhz
 OSCCONbits.IRCF0 = 1 ;
 OSCCONbits.IRCF1 = 1 ;
 OSCCONbits.IRCF2 = 1 ;

 //Configuration ADC
   //Configuration ADCON1: on utilise ports ANO, AN1, AN2
   ADCON1bits.VCFG1 = 0;
   ADCON1bits.VCFG0 = 0;
   ADCON1bits.PCFG3 = 1; //Config entrée analogique
   ADCON1bits.PCFG2 = 1; //Config entrée analogique
   ADCON1bits.PCFG1 = 0; //Config entrée analogique
   ADCON1bits.PCFG0 = 0; //Config entrée analogique ANO, AN1, AN2

   //Configuration ADCON2
   ADCON2bits.ADCS2 = 1;
   ADCON2bits.ADCS1 = 0;
   ADCON2bits.ADCS0 = 1; //Fréquence de fonctionnement de l'ADC
   ADCON2bits.ADFM = 0;
   ADCON2bits.ACQT2 = 0;// TACQ = 6u3
   ADCON2bits.ACQT1 = 1;
   ADCON2bits.ACQT0 = 1;

   //Configuration ADCONO
   ADCON0bits.CHS0 = 0; // AN2 est sélectionnée en entrée
   ADCON0bits.CHS1 = 1;
   ADCON0bits.CHS2 = 0;
   ADCON0bits.CHS3 = 0;
   ADCON0bits.ADON = 1; //Activation de l'ADC

 //Configuration I2C
   MI2CInit();

 //Configuration Timer2
   T2CONbits.T2OUTPS3=1;
   T2CONbits.T2OUTPS2=1;
   T2CONbits.T2OUTPS1=1;
   T2CONbits.T2OUTPS0=1; //Postscaler a 16
   T2CONbits.TMR2ON=1; //Activation du Timer 2
   T2CONbits.T2CKPS1=0;
   T2CONbits.T2CKPS0=1; //Prescaler A 4
```

```
49
50 //Configuration des moteurs : mode PWM
51  CCP1CONbits.CCP1M3 = 1;
52  CCP1CONbits.CCP1M2 = 1;
53  CCP2CONbits.CCP2M3 = 1;
54  CCP2CONbits.CCP2M2 = 1;
55
56  PR2 = 249; //Période PWM de 0.5ms (fpwm=2kHz)
57  CCPR1L = 0; //Rapports cycliques a 0
58  CCPR2L = 0;
59
60 // Configuration Interruptions
61  INTCONbits.GIE=1; //Autorisation des interruptions
62  INTCONbits.PEIE=1; //Autorisation des interruptions peripheriques   PIElbits.TMR2IE=1; //Autorisation de l'inter-
ruption du TMR2
63  PIE1bits.TMR2IE=1; //Autorisation de l'interruption du TMR2
64  INTCON2bits.INTEDG0 = 0;
65  INTCON2bits.INTEDG2=1;
66  PIR1bits.TMR2IF =0;
67  PIE1bits.ADIE=0;
68
69 // Configuration UART
70  TXSTAbits.SYNC = 0;
71  TXSTAbits.BRGH = 1;
72  BAUDCONbits.BRG16 = 0; //8 bits
73  SPBRG = 103; //baud rate de 4800
74  SPBRGH = 0;
75  RCSTAbits.SPEN = 1;
76  TXSTAbits.TXEN = 1; //Autorisation des transmissions
77  TRISCbits.RC6 = 0; //TX en sortie
78  TRISCbits.RC7 = 1; //RX en entrée - réception
79
80 //Configuration des entrées/sorties
81  TRISAbits.RA6=0; // DIRD sortie
82  TRISAbits.RA7=0; // DIRG sortie
83
84  TRISBbits.RB0=1; //Interruption télécommande en entrée
85  TRISBbits.RB5=0; // Port LED en sortie
86
87  TRISCbits.RC1=0; // PWM gauche sortie
88  TRISCbits.RC2=0; // PWM droit sortie
89
90  PORTAbits.RA6=0; //Initialisation des relais
91  PORTAbits.RA7=0;
92
93  PORTBbits.RB5=1; // LED vie éteinte
94 }
95
96
```

main.c

```
#include "main.h"
#include "p18f2520.h"
#include <stdio.h>
#include <stdlib.h>
#include "MI2C.h"

#define BAS 50
#define HAUT 52
#define adresse_telecommande 0xA2 // Code bouton Télécommande
#define adresse_led 0x40
#define PAS 0x19

#pragma code HighVector = 0x08
void IntHighVector(void) { _asm goto HighISR _endasm }
#pragma code

#pragma interrupt HighISR

void HighISR(void) {

  if (INTCONbits.INT0IF && pause==0) // Reception télécommande
  {
     pause=1;
    INTCONbits.INT0IF = 0;
    flag_it = 1;
  }

  if (PIR1bits.TMR2IF) {
    PIR1bits.TMR2IF = 0;
    count ++;
    if (pause){
       count2 ++;
       INTCONbits.INT0IF = 0;
       if (count2 == 50){
          pause=0;
          count2=0;
    }
    }
    if (count == 200){
       count = 0;
       surveillance_batterie();
    }
  }
}
void surveillance_batterie() {
  int Vbat = 0;
  int n=0;
  for (n; n < 4; n++) {
    ADCON0bits.GO = 1; // démarrage ADC
    while (ADCON0bits.GO_DONE) {}
    // On attend jusqu'à que la lecture soit fini
```

```
52    Vbat += ADRESH;
53    // ADRESH est le lieu de stockage de la lecture ADCON0
54   }
55   Vbat = Vbat / 4;
56   if (Vbat < 130) {
57    flag_batterie = 0;
58    printf("Batterie faible\n\r");
59    marche = 0;
60    vitesse = 0;
61    CCPR1L = 0;
62    CCPR2L = 0;
63    Write_PCF8574(adresse_led, 0b11111111);
64   }
65
66   else {
67    flag_batterie = 1;
68    printf("Batterie %d\n\r", Vbat);
69   }
70  }
71
72  void allumageled(int n_leds) {
73   if (n_leds == 0) {
74    Write_PCF8574(adresse_led, 0b01111111);
75   } else if (n_leds == 1) {
76    Write_PCF8574(adresse_led, 0b01111110);
77   } else if (n_leds == 2) {
78    Write_PCF8574(adresse_led, 0b01111100);
79   } else if (n_leds == 3) {
80    Write_PCF8574(adresse_led, 0b01111000);
81   } else if (n_leds == 4) {
82    Write_PCF8574(adresse_led, 0b01110000);
83   } else if (n_leds == 5) {
84    Write_PCF8574(adresse_led, 0b01100000);
85   }
86  }
87  void main(void) {
88
89   init();
90   printf("Fin init\n\r");
91   CCPR1L=0x0;
92   CCPR2L=0x0;
93   vitesse=0;
94   surveillance_batterie();
95   if (flag_batterie){
96     allumageled(0);
97   }
98    while (1) {
99      if (flag_batterie){
100       if (flag_it) {
101         Lire_i2c_Telecom(adresse_telecommande, touche);
102         flag_it = 0;
103         if (touche[1] == BAS) {
104          if (marche == 1) {
105           marche = 0;
106           vitesse = 0;
107           CCPR1L = 0;
108           CCPR2L = 0;
109           allumageled(0);
110           printf("Arret\n\r");
111          } else if (vitesse < 5) {
112           PORTAbits.RA6 = 1; // sens relais: reculer
113           PORTAbits.RA7 = 1;
114           marche = 2;
115           vitesse++;
```

```
116                 CCPR1L += PAS;
117                 CCPR2L += PAS;
118                 allumageled(vitesse);
119                 printf("Reculer %d\n\r",vitesse);
120               }
121             } else if (touche[1] == HAUT) {
122               if (marche == 2) {
123                 marche = 0;
124                 vitesse = 0;
125                 CCPR1L = 0;
126                 CCPR2L = 0;
127                 allumageled(0);
128                 printf("Arret\n\r");
129               } else if (vitesse < 5) {
130                 PORTAbits.RA6 = 0; // sens relais: avancer
131                 PORTAbits.RA7 = 0;
132                 marche= 1;
133                 vitesse++;
134                 CCPR1L += PAS;
135                 CCPR2L += PAS;
136                 allumageled(vitesse);
137                 printf("Avancer %d\n\r",vitesse);
138               }
139             }
140           }
141       }
142   }
143 }
144
```

main.h

```
1 #ifndef MAIN_H
2 #define MAIN H
3
4 unsigned int Vbat = 0;
5 volatile char flag_batterie = 0;
6 volatile char flag_it;
7 volatile char pause = 0;
8 volatile int count = 0;
9 volatile int count2 = 0;
10 volatile char marche=0;
11 signed int vitesse = 0;
12 unsigned int compt_TMR2;
13
14
15 char touche [3];
16
17 //UART
18 rom const char batterie[] = "Batterie ok\n ";
19 rom const char batterie2[] = "Batterie off\n ";
20 rom const char vitesse_max[] = "Vitesse maximale\n";
21 rom const char fin_init[] = "Fin initialisation\n";
22
23 void IntHighVector(void);
24 void HighISR(void);
25 void init(void);
26 void surveillance_batterie(void);
27 void allumage_led();
28
29 #endif
```