Note d'Application

Développement d'un BMS pour la Formula Student

Utilisation du PIC18F46Q83

GENEVAY Ludovic

Étudiant en Génie Électrique

Projet Formula Student

January 8, 2025

Contents

1 Introduction

Cette note d'application décrit le développement d'un programme pour un microcontrôleur PIC18F46Q83, utilisé dans le cadre d'un système de gestion de batterie (BMS) pour le projet Formula Student.

Le programme implémente toutes les fonctionnalités nécessaires pour gérer la surveillance et la sécurité des batteries, y compris l'acquisition des mesures, la vérification des seuils de sécurité, et (dans le futur) la transmission des données sur bus CAN. Cette note se concentre exclusivement sur les aspects logiciels du développement : structure du code, fonctions principales, gestion des données et déploiement sur la carte. Les aspects matériels (comme les capteurs ou l'intégration) sont gérés séparément et ne seront pas abordés dans ce document.

2 Structure Générale du Programme

Le programme est écrit en langage C et comporte les principales fonctionnalités suivantes :

- Initialisation des constantes et paramètres nécessaires aux calculs.
- Acquisition des mesures analogiques via des multiplexeurs (MUX).
- Conversion des mesures pour obtenir des températures, tensions et courants.
- Vérification des seuils de sécurité.
- Transmission des données via le protocole CAN (implémentation matériel faite, le développement du code est en cours et en phase de test).
- Gestion des erreurs et actions correctives.

3 Développement du Programme

3.1 Outils utilisés

Le développement du programme a été réalisé à l'aide des outils suivants :

- **MPLAB X IDE :** Environnement de développement intégré pour le microcontrôleur PIC18F46Q83.
- **XC8 Compiler :** Compilateur C pour les microcontrôleurs PIC.
- **Simulateur intégré :** Outil de simulation pour tester le code avant déploiement sur le matériel.
- **Pickit 4 :** Outil de programmation pour charger le code sur le microcontrôleur et debbuger directement sur le matériel.

3.2 Phase de Tests

Tout au long du développement, des tests ont été effectués pour vérifier le bon fonctionnement de chaque fonction et fonctionnalité. Ces tests ont été réalisés dans deux environnements principaux :

- **Simulation logicielle :** Les tests unitaires ont été réalisés directement dans MPLAB X IDE à l'aide des simulateurs intégrés. Cela a permis de vérifier les calculs, les conversions et les réactions du programme aux conditions simulées.
- **Tests sur breadboard :** Chaque fonction a été testée sur une breadboard en connectant le microcontrôleur PIC18F46Q83 aux capteurs (thermistances, diviseurs résistifs, capteurs à effet Hall) et en reproduisant les scénarios de fonctionnement réels. Ces tests ont permis de valider les interactions entre le programme et le matériel.

Les tests sur breadboard ont également permis d'évaluer le bon fonctionnement global du programme, en combinant toutes les fonctionnalités, telles que :

- La mesure des tensions et courants.
- La conversion des données en valeurs exploitables.
- La gestion des seuils pour assurer la sécurité du système.
- Les interruptions pour effectuer des mesures périodiques.

Ces étapes de validation ont été cruciales pour garantir que le programme fonctionnerait correctement lorsqu'il serait déployé sur le véhicule Formula Student.

3.3 Structure du Code

Le programme est structuré de manière modulaire pour faciliter la maintenance et les tests. Les modules principaux incluent :

- **Module d'initialisation :** Configure les périphériques (ADC, CAN, interruptions, etc.).
- **Module d'acquisition :** Gère les mesures analogiques (températures, tensions, courants).
- **Module de traitement :** Convertit les données brutes en valeurs exploitables.
- **Module de communication :** Transmet les données via le protocole CAN.
- **Module de vérification :** Surveille les seuils de sécurité et déclenche des actions en cas d'anomalie.

4 Description des Fonctions Principales

4.1 `initPentes`

Initialise un tableau de pentes utilisé pour convertir les mesures de tension des capteurs en température. Elle calcule la pente entre chaque paire de points du tableau de résistances.

4.2 `convUtempToTemp`

Convertit une mesure de tension (`Utemp`) en température en fonction de la résistance du thermistor.

4.3 `selMuxEntry`

Permet de sélectionner une entrée parmi les multiplexeurs (MUX) pour rediriger les signaux vers le canal ADC approprié.

4.4 `MesAllTemp`

Effectue les mesures sur tous les capteurs de température connectés aux MUX et les vérifie par rapport aux seuils définis.

4.5 `convert_voltage_to_current`

Convertit une tension mesurée en courant en utilisant la constante de sensibilité.

4.6 `MesAllVolt`

Effectue les mesures de tension sur les différentes cellules via les MUX et met à jour les tableaux de tensions.

4.7 `Timer0_CustomHandlerISR`

Interruption personnalisée utilisée pour lancer périodiquement les mesures de températures et de tensions, vérifier les seuils et déclencher des actions en cas de détection d'anomalies.

5 Transmission des Données (CAN)

Le programme utilise le protocole CAN (Controller Area Network) pour transmettre les données. La fonction `CANTransmitTest` est un exemple de transmission de trame CAN avec un ID standard (`0x100`) et 8 octets de données.

Cependant, le bus CAN n'est actuellement pas pleinement opérationnel. Une vérification effectuée à l'oscilloscope a confirmé la transmission des signaux de données sur le bus CAN, ce qui indique que le microcontrôleur génère correctement les trames CAN.

Malgré cela, aucune donnée n'a pu être visualisée à l'aide d'un analyseur CAN-USB. Ce problème suggère une possible incompatibilité ou une mauvaise configuration entre le programme et l'analyseur CAN. Des investigations supplémentaires sont nécessaires pour identifier et résoudre cette anomalie.

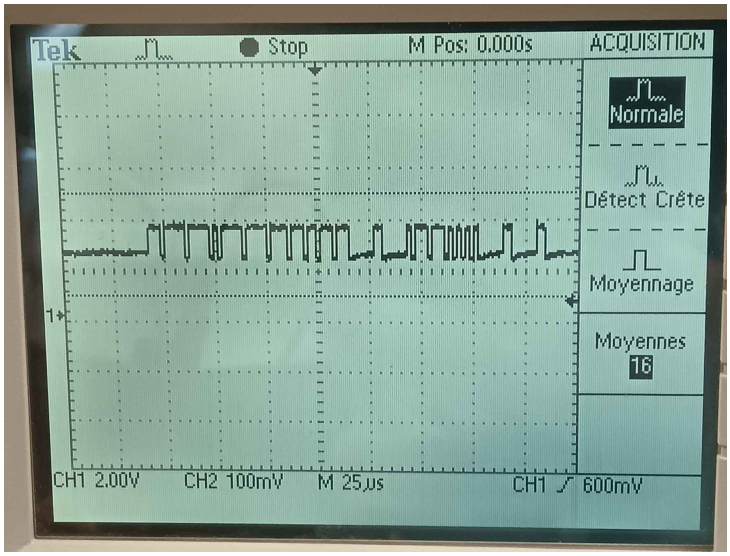

Figure 1: Visualisation du bus CAN

6 Gestion des Anomalies

Le programme vérifie en permanence les seuils de sécurité pour les tensions, les courants et les températures. Les fonctions principales incluent :

- `checkVolt`: Vérifie si les tensions des cellules sont dans les seuils définis.
- `checkCurrent`: Vérifie si le courant est en-dessous de 850 A.
- `checkTemp`: Vérifie si toutes les températures des cellules sont inférieures à 40°C.

Si une anomalie est détectée, des indicateurs tels que `xTrigVolt`, `xTrigCurrent` ou `xTrigTemp` sont activés, déclenchant une réponse corrective (envoi d'un signal 5V sur GPIO broche RD2, pinout S_RELAY1).

7 Gestion des Données

En raison de la structuration physique du pinout des multiplexeurs (MUX) utilisés dans le projet, il était nécessaire d'implémenter des fonctions spécifiques, appelées `pathMX` (où `X` représente le numéro du MUX). Ces fonctions ont pour rôle de ranger les valeurs de tension mesurées en entrée dans des tableaux organisés (`P1`, `P2`, `P3`, `P4`, `P5`).

7.1 Organisation des Données

Chaque tableau correspond à l'une des cinq rangées de cellules (cellules connectées en série) dans la batterie. Les fonctions `pathMX` permettent d'associer chaque tension mesurée à la cellule correspondante dans la rangée appropriée. Cette organisation est essentielle pour :

- **Simplifier le débogage :** En regroupant les mesures par rangées de cellules, il devient plus facile de visualiser et de vérifier les valeurs lors des tests.

- **Améliorer la lisibilité du code :** Cette structuration permet de mieux comprendre l'attribution des tensions mesurées aux différentes cellules.
- **Faciliter l'identification des anomalies :** Lorsqu'une cellule présente une tension anormale, cette organisation permet d'identifier rapidement la cellule défaillante et de localiser précisément la rangée concernée.

7.2 Intégration dans le Programme

Les tableaux `P1`, `P2`, `P3`, `P4`, et `P5` sont utilisés tout au long du programme pour centraliser les mesures de tension. Ces données structurées permettent également de vérifier efficacement si les valeurs mesurées se trouvent dans les seuils de sécurité prédéfinis.

7.3 Transmission des Données sur le Bus CAN

Grâce à cette organisation, le programme pourra transmettre sur le bus CAN des informations claires et spécifiques en cas d'anomalie. Par exemple, si une cellule est identifiée comme étant en défaut, il sera possible de transmettre son emplacement exact (rangée et position dans la rangée) afin de faciliter les diagnostics et les interventions. Cette fonctionnalité est en cours de développement pour rendre le système encore plus robuste et fiable.

8 Boucle Principale

La boucle principale (`main`) réalise les tâches suivantes :

- Initialisation des modules et constantes.
- Exécution périodique des mesures et des vérifications via des interruptions.
- Transmission des données aux autres modules via CAN.

9 Environnement de Test

Le programme a été testé dans les environnements suivants :

- **Banc de test :** Permettant de simuler des scénarios de fonctionnement du système, tels que des variations de température et des tensions hors seuil.
- **Simulation logicielle :** Validation des interruptions, de la gestion des seuils et des conversions numériques via MPLAB X IDE.

10 Déploiement du Programme

Cette section explique comment déployer le programme sur le microcontrôleur PIC18F46Q83 à partir du fichier ZIP fourni, en utilisant MPLAB X IDE et un programmateur PICkit 4.

10.1 Matériel Nécessaire

Pour le déploiement du programme, vous aurez besoin des éléments suivants :

- Une carte équipée du microcontrôleur PIC18F46Q83, correctement positionné à son emplacement sur la carte.
- Un programmateur PICkit 4 connecté à la carte comme illustré plus bas.
- Un ordinateur avec MPLAB X IDE installé (installation des composants détaillée dans les étapes suivantes).
- Le fichier ZIP contenant le projet (`.X`) à ouvrir dans MPLAB X IDE.

10.2 Étapes pour le Déploiement

Voici les étapes détaillées pour déployer le programme sur le microcontrôleur :

1. **Installer MPLAB X IDE :** La version utilisée au cour du développement était la 6.20, assurez-vous d'installer les composants suivants au cour de l'installation (8-bit MCUs et MPLAB XC8 Compiler):

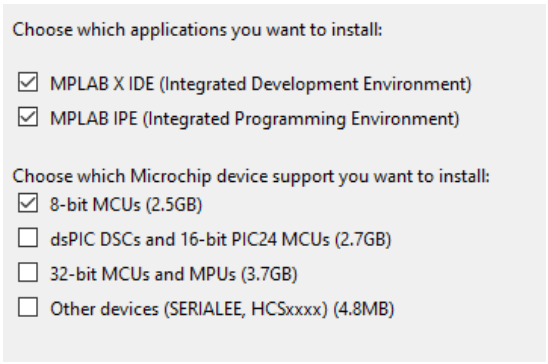

Figure 2: Composants nécessaires au développement.

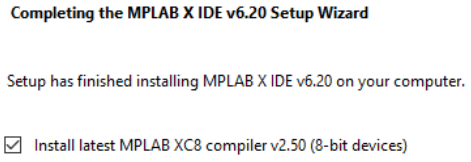

Figure 3: Compilateur nécessaire

2. **Télécharger le projet :** Téléchargez le fichier ZIP contenant le projet et décompressez-le sur votre ordinateur.
3. **Ouvrir le projet :** Lancez MPLAB X IDE et utilisez l'option `File > Open Project` pour ouvrir le fichier `.X` extrait du ZIP.
4. **Vérifier les configurations :** Vérifiez que la configuration du projet correspond au microcontrôleur PIC18F46Q83 :

- Assurez-vous que le compilateur XC8 est sélectionné dans les paramètres du projet.
- Vérifiez les options de compilation et les périphériques activés dans les fichiers `.c` et `.h`.
- Dans les paramètres “power” du PICkit 4 et afin de pourvoir effectuer un debug, il faut spécifier l’alimentation à fournir au PIC (5V) à travers le PICkit (dans le cas où le pic n’est pas alimenté de manière externe).

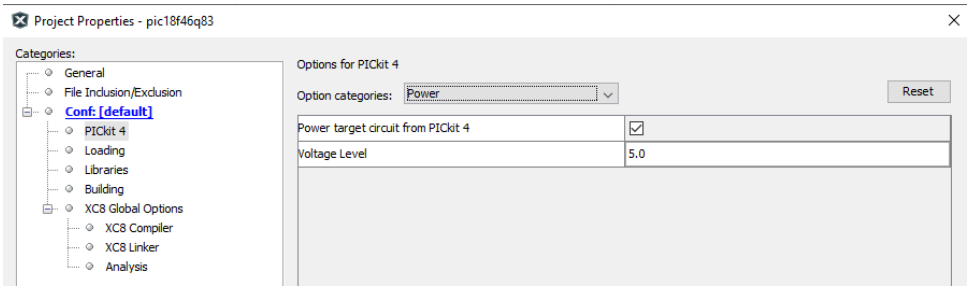

Figure 4: Alimentation de la carte par le PICkit

5. **Brancher le PICkit 4 :** Connectez le programmateur PICkit 4 à la carte contenant le microcontrôleur, puis branchez le programmateur à votre ordinateur via USB.

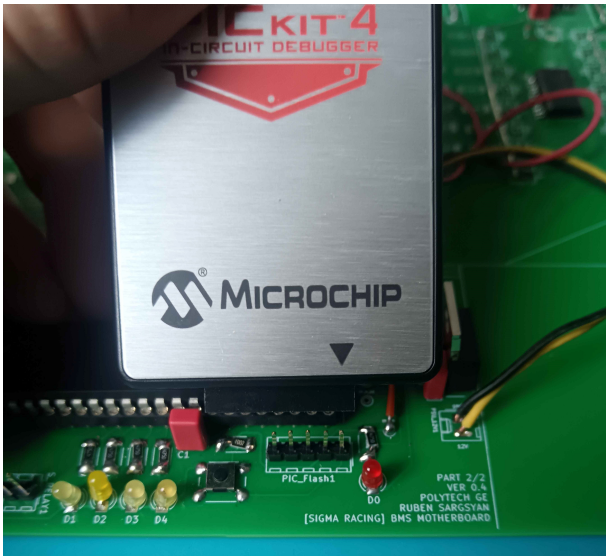

Figure 5: Connexion du PICkit 4 au microcontrôleur.

6. **Sélectionner le PICkit 4 :** Dans MPLAB X IDE, sélectionnez le PICkit 4 comme outil de programmation en naviguant dans : `Project > Properties > Hardware Tools`, puis choisissez `PICkit 4`.

7. **Compiler le projet :** Cliquez sur le bouton `Build Project` (icône de marteau) pour compiler le code source. Assurez-vous qu’il n’y a pas d’erreurs lors de la compilation.

8. **Programmer le microcontrôleur :** Une fois la compilation réussie, cliquez sur le bouton `Make and Program Device` (icône de flèche verte) pour transférer le programme sur le microcontrôleur.

9. **Vérifier la programmation :** Une fois la programmation terminée, un message de succès s'affichera dans la console de MPLAB X IDE. Débranchez le PICkit 4 et allumez la carte pour vérifier le bon fonctionnement du programme.

10.3 Notes et Recommandations

- Assurez-vous que le microcontrôleur est correctement alimenté pendant le processus de programmation.
- En cas d'erreurs de programmation, vérifiez les connexions du PICkit 4 et la configuration du projet dans MPLAB X IDE.
- Si nécessaire, référez-vous au manuel du PICkit 4 ou à la documentation de MPLAB X IDE pour résoudre les problèmes courants.

11 Conclusion

Ce programme constitue une solution robuste et modulaire pour la gestion des batteries dans un système BMS basé sur le microcontrôleur PIC18F46Q83. Les fonctionnalités d'acquisition, de conversion, de vérification et de transmission des données assurent une surveillance fiable des cellules et préviennent les conditions dangereuses.

12 Annexes

A Code Source du Programme

Voici le code source complet du programme en C utilisé pour le développement du BMS :

```
#define SENSITIVITY (800.0 / 1.5) // Sensibilite du capteur de courant (800 A pour 1.5 V)
#define tensionBridgeFactor (1082000.0/82000.0) //Facteur par lequel on multiplie les mesures de tensions des cellules car un PDT est applique en entree
#define voltLowerThreshold (2000.0/tensionBridgeFactor)
#define voltUpperThreshold (2700.0/tensionBridgeFactor)

double VREF = 2.5;
double voltCurrent = 0.0;
double currentMes = 0.0;

double tempResult[64];
bool boolMesTemp[64];
double tempMes[64];

double P1[22];
double P2[22];
```

```
double P3[22];
double P4[22];
double P5[22];
double PActiveCell[5];

bool boolMesVoltM0[16];
bool boolMesVoltM1[16];
bool boolMesVoltM6[16];
bool boolMesVoltM7[16];
bool boolMesVoltM8[16];
bool boolMesVoltM9[16];
bool boolMesVoltM10[16];

int tempTab[16] = {0,10,20,30,40,50,60,70,80,90,110,120,130,140,150};
double resTab[16] ={32554.202,
19872.168,
12487.744,
8059.079,
5329.869,
3605.267,
2489.951,
1753.042,
1256.387,
915.425,
677.299,
508.316,
386.598,
297.696,
231.910,
182.633
    };

double pentesTab[15];

bool initPentes(){ //Initialise le tableau des pentes pour la convetion
    des mesures de temperatures
    for(int i=0; i<15;i++){
        pentesTab[i] = (resTab[i+1] - resTab[i])/10;
    }
    return true;
}

double convUtempToTemp(double Utemp){ //Fonction pour convertir les
   mesures de tensions des capteurs de temperature en temperature

    double result = .0;
    double R2 = 2200.0;
    double vddMes = 4860.0;

    double RT = R2 * ((vddMes/Utemp)-1.0);

    for(int i = 0; i<15;i++){
        if (RT<resTab[i] && RT>resTab[i+1]){
            return tempTab[i] + (RT-resTab[i])/pentesTab[i]; //Les
               pentes sont negatives et le facteur aussi
        }
    }
    return -1;
```

```
71  }
72  bool selMuxEntry(char sel)
73  {   //Permet de selectionner l'entree des MUX
74      if (sel & 0b0001){
75          IO_RD4_SetHigh();
76      }
77      else{IO_RD4_SetLow();}
78      if (sel & 0b0010){
79          IO_RD5_SetHigh();
80      }
81      else{IO_RD5_SetLow();}
82
83      if (sel & 0b0100){
84          IO_RD6_SetHigh();
85      }
86      else{IO_RD6_SetLow();}
87
88      if (sel & 0b1000){
89          IO_RD7_SetHigh();
90      }
91      else{IO_RD7_SetLow();}
92
93      return true;
94  }
95  void flashLED(int time){ //Fait clignotter les leds en modifiant les
        gpio de controle des MUX (usage DEBUG uniquement)
96
97      selMuxEntry(0b0000);
98      for(int i = 0; i<time/100;i++){
99          __delay_ms(100);
100     }
101     selMuxEntry(0b1111);
102     for(int i = 0; i<time/100;i++){
103         __delay_ms(100);
104     }
105     selMuxEntry(0b0000);
106     for(int i = 0; i<time/100;i++){
107         __delay_ms(100);
108     }
109
110 }
111
112 double setAndGetChannel(adc_channel_t channel){ //Selectionne un
        channel analogique et effectue une mesure
113     ADC_ChannelSelect(channel);
114     ADC_ConversionStart();
115     while (!ADC_IsConversionDone());
116     return (double)ADC_ConversionResultGet();
117 }
118
119 bool isTempInThreshold(double input){
120     return (input < 40);
121 }
122
123 bool MesAllTemp() { //Effectue une mesure de tous les capteurs de
        temperatures
124     for (char i = 0; i <= 15; i++) {
125         // Selectionner l'entree du multiplexeur correspondant
```

```
126         selMuxEntry(i);
127 
128         // Conversion pour le canal AN2
129         tempMes[i] = setAndGetChannel(ADC_CHANNEL_ANA2);
130         tempResult[i] = convUtempToTemp(tempMes[i]);
131         boolMesTemp[i] = isTempInThreshold(tempResult[i]);
132         // Conversion pour le canal AN3
133         tempMes[i + 16] = setAndGetChannel(ADC_CHANNEL_ANA3);
134         tempResult[i + 16] = convUtempToTemp(tempMes[i + 16]);
135         boolMesTemp[i + 16] = isTempInThreshold(tempResult[i + 16]);
136         // Conversion pour le canal AN4
137         tempMes[i + 32] = setAndGetChannel(ADC_CHANNEL_ANA5);
138         tempResult[i + 32] = convUtempToTemp(tempMes[i + 32]);
139         boolMesTemp[i + 32] = isTempInThreshold(tempResult[i + 32]);
140         // Conversion pour le canal AN4
141         tempMes[i + 48] = setAndGetChannel(ADC_CHANNEL_ANE0);
142         tempResult[i + 48] = convUtempToTemp(tempMes[i + 48]);
143         boolMesTemp[i + 48] = isTempInThreshold(tempResult[i + 48]);
144     }
145 
146     return true;
147 }
148 
149 
150 double convert_voltage_to_current(double voltage) {
151     // Verification des bornes de la tension d'entree
152     if (voltage < (VREF - 1.5) ){//|| voltage > (VREF + 1.5)) {
153         //printf("Erreur : Tension hors des limites attendues !n");
154         return 0.0;
155     }
156 
157     double current = 0.0;
158     // Conversion de la tension en courant
159     if(VREF < 1.0){current = (voltage - 2.5) * SENSITIVITY;}
160     else{current = (voltage - VREF) * SENSITIVITY;}
161     return current;
162 }
163 
164 bool isVoltInThreshold(double input){
165     return (input < voltUpperThreshold && input>voltLowerThreshold);
166 }
167 
168 
169 void pathM0(int pin, double value){
170     switch (pin){
171         case 0:
172             P1[0] = value;
173             break;
174         case 1:
175             P1[1] = value;
176             break;
177         case 2:
178             P1[2] = value;
179             break;
180         case 3:
181             P1[3] = value;
182             break;
183         case 4:
```

```
            P1[4] = value;
            break;
        case 5:
            P1[5] = value;
            break;
        case 6:
            P1[6] = value;
            break;
        case 7:
            P1[7] = value;
            break;
        case 8:
            P1[8] = value;
            break;
        case 9:
            P1[9] = value;
            break;
        case 10:
            P1[10] = value;
            break;
        case 11:
            P1[11] = value;
            break;
        case 12:
            P1[12] = value;
            break;
        case 13:
            P1[13] = value;
            break;
        case 14:
            P1[14] = value;
            break;
        case 15:
            P1[15] = value;
            break;
    }
}

void pathM1(int pin, double value){
    switch (pin){
        case 0:
            VREF = value;
            break;
        case 1:
            P1[16] = value;
            break;
        case 2:
            P1[17] = value;
            break;
        case 3:
            P1[18] = value;
            break;
        case 4:
            P1[19] = value;
            break;
        case 5:
            P1[20] = value;
            break;
```

```
        case 6:
            P1[21] = value;
            break;
        case 7:
            P4[20] = value;
            break;
        case 8:
            P4[21] = value;
            break;
        case 9:
            P5[16] = value;
            break;
        case 10:
            P5[17] = value;
            break;
        case 11:
            P5[18] = value;
            break;
        case 12:
            P5[19] = value;
            break;
        case 13:
            P5[20] = value;
            break;
        case 14:
            P5[21] = value;
            break;
        case 15:
            voltCurrent = value;
            break;
    }
}

void pathM6(int pin, double value){
    switch (pin){
        case 0:
            P2[0] = value;
            break;
        case 1:
            P2[1] = value;
            break;
        case 2:
            P2[2] = value;
            break;
        case 3:
            P2[3] = value;
            break;
        case 4:
            P2[4] = value;
            break;
        case 5:
            P2[5] = value;
            break;
        case 6:
            P2[6] = value;
            break;
        case 7:
            P2[7] = value;
```

```
            break;
        case 8:
            P2[8] = value;
            break;
        case 9:
            P2[9] = value;
            break;
        case 10:
            P2[10] = value;
            break;
        case 11:
            P2[11] = value;
            break;
        case 12:
            P2[12] = value;
            break;
        case 13:
            P2[13] = value;
            break;
        case 14:
            P2[14] = value;
            break;
        case 15:
            P2[15] = value;
            break;
    }
}

void pathM7(int pin, double value){
    switch (pin){
        case 0:
            P2[16] = value;
            break;
        case 1:
            P2[17] = value;
            break;
        case 2:
            P2[18] = value;
            break;
        case 3:
            P2[19] = value;
            break;
        case 4:
            P2[20] = value;
            break;
        case 5:
            P2[21] = value;
            break;
        case 6:
            P3[16] = value;
            break;
        case 7:
            P3[17] = value;
            break;
        case 8:
            P3[18] = value;
            break;
        case 9:
```

```
            P3[19] = value;
            break;
        case 10:
            P3[20] = value;
            break;
        case 11:
            P3[21] = value;
            break;
        case 12:
            P4[16] = value;
            break;
        case 13:
            P4[17] = value;
            break;
        case 14:
            P4[18] = value;
            break;
        case 15:
            P4[19] = value;
            break;
    }
}

void pathM8(int pin, double value){
    switch (pin){
        case 0:
            P3[0] = value;
            break;
        case 1:
            P3[1] = value;
            break;
        case 2:
            P3[2] = value;
            break;
        case 3:
            P3[3] = value;
            break;
        case 4:
            P3[4] = value;
            break;
        case 5:
            P3[5] = value;
            break;
        case 6:
            P3[6] = value;
            break;
        case 7:
            P3[7] = value;
            break;
        case 8:
            P3[8] = value;
            break;
        case 9:
            P3[9] = value;
            break;
        case 10:
            P3[10] = value;
            break;
```

```
        case 11:
            P3[11] = value;
            break;
        case 12:
            P3[12] = value;
            break;
        case 13:
            P3[13] = value;
            break;
        case 14:
            P3[14] = value;
            break;
        case 15:
            P3[15] = value;
            break;
    }
}

void pathM9(int pin, double value){
    switch (pin){
        case 0:
            P4[0] = value;
            break;
        case 1:
            P4[1] = value;
            break;
        case 2:
            P4[2] = value;
            break;
        case 3:
            P4[3] = value;
            break;
        case 4:
            P4[4] = value;
            break;
        case 5:
            P4[5] = value;
            break;
        case 6:
            P4[6] = value;
            break;
        case 7:
            P4[7] = value;
            break;
        case 8:
            P4[8] = value;
            break;
        case 9:
            P4[9] = value;
            break;
        case 10:
            P4[10] = value;
            break;
        case 11:
            P4[11] = value;
            break;
        case 12:
            P4[12] = value;
```

```
            break;
        case 13:
            P4[13] = value;
            break;
        case 14:
            P4[14] = value;
            break;
        case 15:
            P4[15] = value;
            break;
    }
}

void pathM10(int pin, double value){
    switch (pin){
        case 0:
            P5[0] = value;
            break;
        case 1:
            P5[1] = value;
            break;
        case 2:
            P5[2] = value;
            break;
        case 3:
            P5[3] = value;
            break;
        case 4:
            P5[4] = value;
            break;
        case 5:
            P5[5] = value;
            break;
        case 6:
            P5[6] = value;
            break;
        case 7:
            P5[7] = value;
            break;
        case 8:
            P5[8] = value;
            break;
        case 9:
            P5[9] = value;
            break;
        case 10:
            P5[10] = value;
            break;
        case 11:
            P5[11] = value;
            break;
        case 12:
            P5[12] = value;
            break;
        case 13:
            P5[13] = value;
            break;
        case 14:
```

```
            P5[14] = value;
            break;
        case 15:
            P5[15] = value;
            break;
    }
}

bool MesAllVolt() { //Effectue une mesure de tension sur tous les mux de mesure de tension des cellules
    for (char i = 0; i <= 15; i++) {
        selMuxEntry(i);

        // Lecture et verification des tensions pour M0
        pathM0(i,setAndGetChannel(ADC_CHANNEL_ANA0));

        // Lecture et verification des tensions pour M1
        pathM1(i,setAndGetChannel(ADC_CHANNEL_ANA1));

        // Lecture et verification des tensions pour M6
        pathM6(i,setAndGetChannel(ADC_CHANNEL_ANE1));

        // Lecture et verification des tensions pour M7
        pathM7(i,setAndGetChannel(ADC_CHANNEL_ANE2));

        // Lecture et verification des tensions pour M8
        pathM8(i,setAndGetChannel(ADC_CHANNEL_ANB1));

        // Lecture et verification des tensions pour M9
        pathM9(i,setAndGetChannel(ADC_CHANNEL_ANB4));

        // Lecture et verification des tensions pour M10
        pathM10(i,setAndGetChannel(ADC_CHANNEL_ANB0));
    }

    return true;
}


void CANTransmitTest(void){
    struct CAN_MSG_OBJ Transmission;  //create the CAN message object
    uint8_t Transmit_Data[8]={0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77}; // data bytes
    Transmission.field.brs=CAN_NON_BRS_MODE; // No bit rate switching
    Transmission.field.dlc=DLC_8; //8 data bytes
    Transmission.field.formatType=CAN_2_0_FORMAT; //CAN 2.0 frames
    Transmission.field.frameType=CAN_FRAME_DATA; //Data frame
    Transmission.field.idType=CAN_FRAME_STD; //Standard ID
    Transmission.msgId=0x100; //ID of 0x100
    Transmission.data=Transmit_Data; //transmit the data from the data bytes
    if(CAN_TX_FIFO_AVAILABLE == (CAN1_TransmitFIFOStatusGet(CAN1_TXQ) & CAN_TX_FIFO_AVAILABLE))//ensure that the TXQ has space for a message
    {
        CAN1_Transmit(CAN1_TXQ, &Transmission); //transmit frame
    }
}
```

```
585
586 bool checkVolt(){
587
588     for (int i = 0; i<=14;i++){
589         PActiveCell[0] = (P1[i+1]-P1[i]);
590         if (isVoltInThreshold(PActiveCell[0])){
591             return false;
592         }
593         PActiveCell[1] = (P2[i+1]-P2[i]);
594         if (isVoltInThreshold(PActiveCell[1])){
595             return false;
596         }
597         PActiveCell[2] = (P3[i+1]-P3[i]);
598         if (isVoltInThreshold(PActiveCell[2])){
599             return false;
600         }
601         PActiveCell[3] = (P4[i+1]-P4[i]);
602         if (isVoltInThreshold(PActiveCell[3])){
603             return false;
604         }
605         PActiveCell[4] = (P5[i+1]-P5[i]);
606         if (isVoltInThreshold(PActiveCell[4])){
607             return false;
608         }
609     }
610     return true;
611 }
612
613 bool checkCurrent(){
614     currentMes = convert_voltage_to_current(voltCurrent);
615             if( currentMes > 850){return false;}
616 }
617 bool checkTemp(){
618     for(int i = 0; i<=63;i++){
619         if (!boolMesTemp[i]){return false;}
620     }
621     return true;
622 }
623
624 bool xTimer = false;
625 bool xTrigVolt = false;
626 bool xTrigCurrent = false;
627 bool xTrigTemp = false;
628
629 void Timer0_CustomHandlerISR(){
630
631         MesAllTemp();
632     MesAllVolt();
633
634     xTimer = !xTimer;
635
636     //Verification des tensions chaque 500ms
637
638     if(!checkVolt()){
639         if(!xTrigVolt){xTrigVolt = true;}
640         else{IO_RD2_SetHigh();return;}
641     }
642     else{
```

```
643             xTrigVolt = false;
644         }
645 
646         if(!checkCurrent()){
647             if(!xTrigCurrent){xTrigCurrent = true;}
648             else{}//IO_RD2_SetHigh();return;} Commente pour tester les
                    autres fonctions
649         }
650         else{
651             xTrigCurrent = false;
652         }
653 
654         //Verification des temperatures chaque seconde
655         if (xTimer || xTrigTemp){
656             if(!checkTemp()){
657                 if(!xTrigTemp){xTrigTemp = true;}
658                 else{IO_RD2_SetHigh();return;}
659             }
660             else{
661                 xTrigTemp = false;
662             }
663         }
664 
665         IO_RD2_SetLow();
666 }
667 
668 int main(void)
669 {
670     SYSTEM_Initialize();
671     // If using interrupts in PIC18 High/Low Priority Mode you need to
            enable the Global High and Low Interrupts
672     // If using interrupts in PIC Mid-Range Compatibility Mode you need
             to enable the Global Interrupts
673     // Use the following macros to:
674 
675     // Enable the Global Interrupts
676     INTERRUPT_GlobalInterruptEnable();
677 
678     // Disable the Global Interrupts
679     //INTERRUPT_GlobalInterruptDisable();
680 
681     initPentes();
682 
683     Timer0_OverflowCallbackRegister(Timer0_CustomHandlerISR);
684 
685     while(1)
686     {
687         //CANTransmitTest();
688         flashLED(100);
689     }
690 }
```

Listing 1: Programme en C pour le BMS