Relatório de estágio IPFN

Battery Management System

Grupo:

Ana Raquel Carmo

Henrique Lopes

João Ramiro

Pedro Messias

Data: 14 de Setembro de 2017

Instituto Superior Técnico

Ano letivo 2016/2017

1 Descrição do projeto

O projeto que pretendemos desenvolver consiste na implementação e programação de uma BMS (*"Battery Management System"*). Desta forma, o controlo do sistema será realizado através de um micro-controlador (dsPIC30F4011) que irá efetuar o balanceamento passivo das células que compõem a bateria.

O balanceamento de células é uma técnica usada para manter os níveis de tensão iguais ou quase iguais em todas as células da bateria. Isto é alcançado monitorizando e controlando o processo de carga e descarga da bateria. Este sistema visa garantir o uso ideal da energia dentro da bateria, alimentando o equipamento, enquanto o risco de danos infligidos sobre o mesmo é minimizado.

Os fluxogramas foram usados para simplificar os algoritmos de codificação e para criar um código de programação lógica.

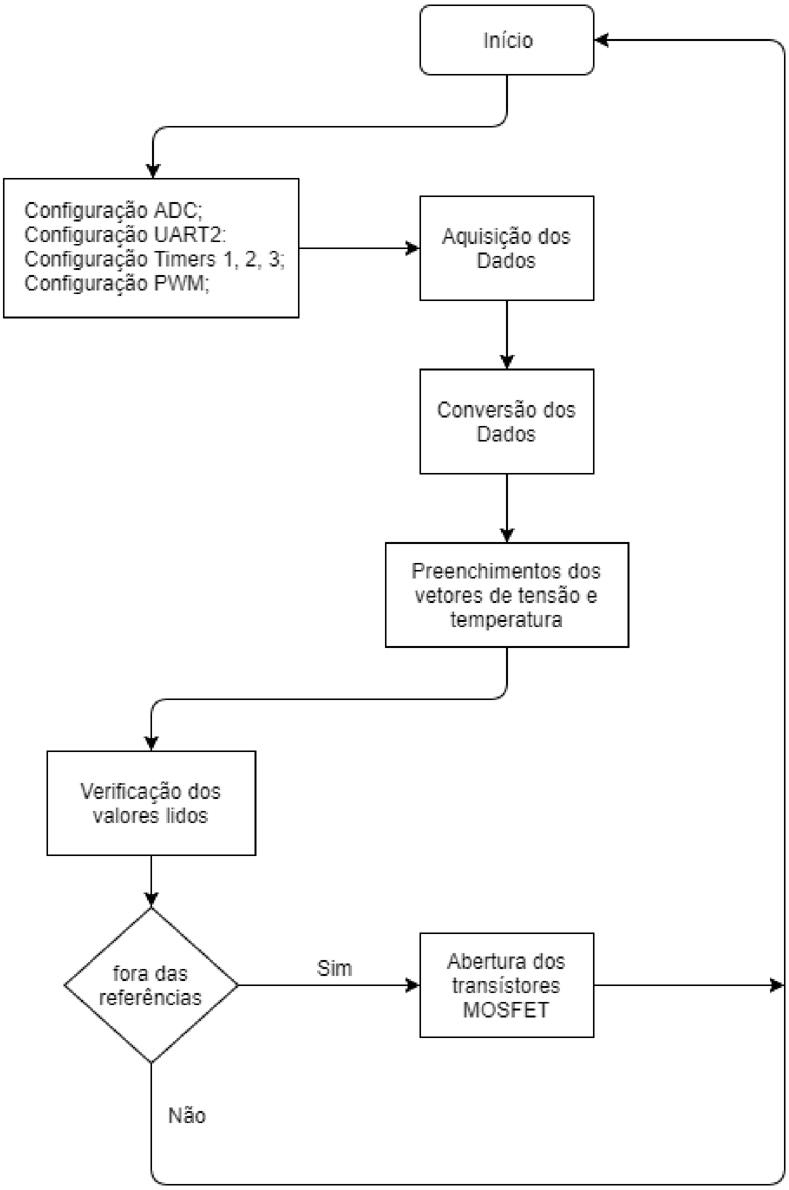
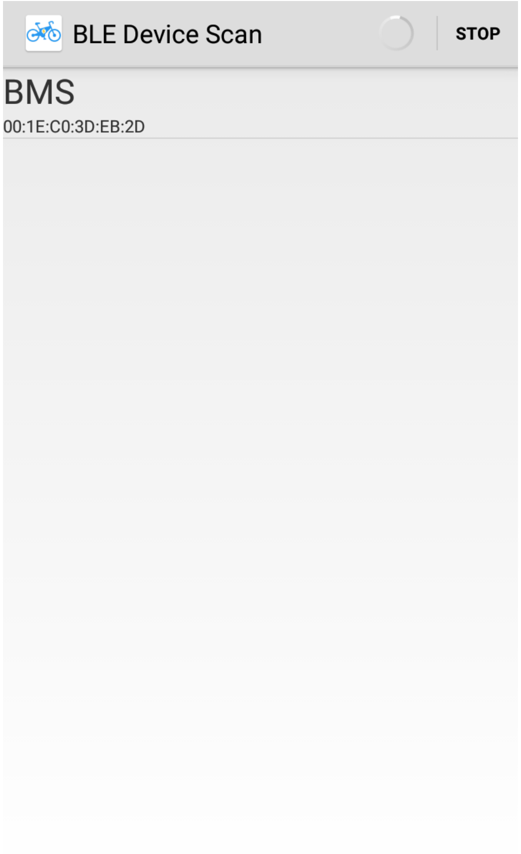

Início
Configuração ADC;
Configuração UART2:
Configuração Timers 1, 2, 3;
Configuração PWM;
Aquisição dos Dados
Conversão dos Dados
Preenchimentos dos vetores de tensão e temperatura
Verificação dos valores lidos
fora das referências
Sim
Abertura dos transístores MOSFET
Não

Figura 1: Fluxograma do projeto.

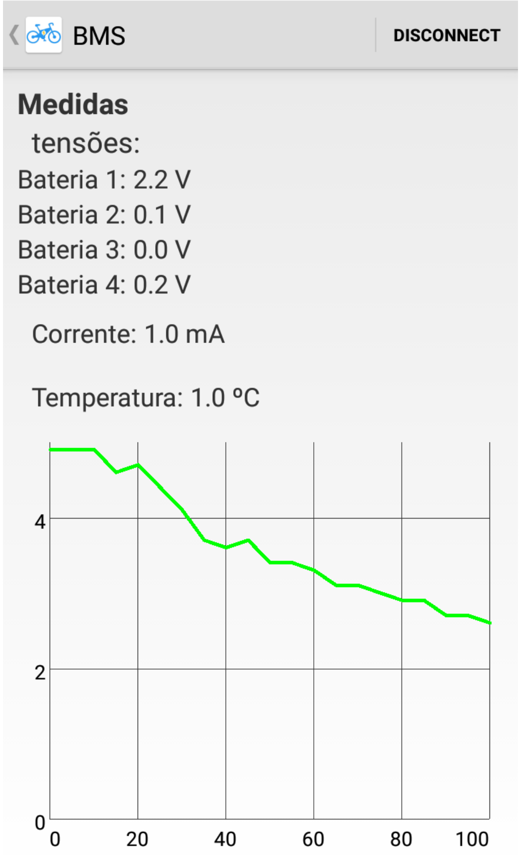
2 Aplicação

A função da aplicação desenvolvida, é obter os valores de tensão das células, a corrente que o motor requer e a temperatura das baterias. Toda esta aplicação foi baseada no código open source disponibilizado pelo Google no exemplo Android BLE. A aplicação contém dois ecrãs, um ou de procura dispositivos BLE, tendo o utilizador que selecionar o que diz BMS, e outro que mostra as informações relativas à bateria.



BMS
00:1E:C0:3D:EB:2D

Figura 2: Scan BLE



Medidas
tensões:
Bateria 1: 2.2 V
Bateria 2: 0.1 V
Bateria 3: 0.0 V
Bateria 4: 0.2 V

Corrente: 1.0 mA

Temperatura: 1.0 ºC

Figura 3: Monitorizar BMS

(Os valores nestes *screenshots* são apenas *placeholders* e não deverão ser tidos em conta.)

3 3D PCB

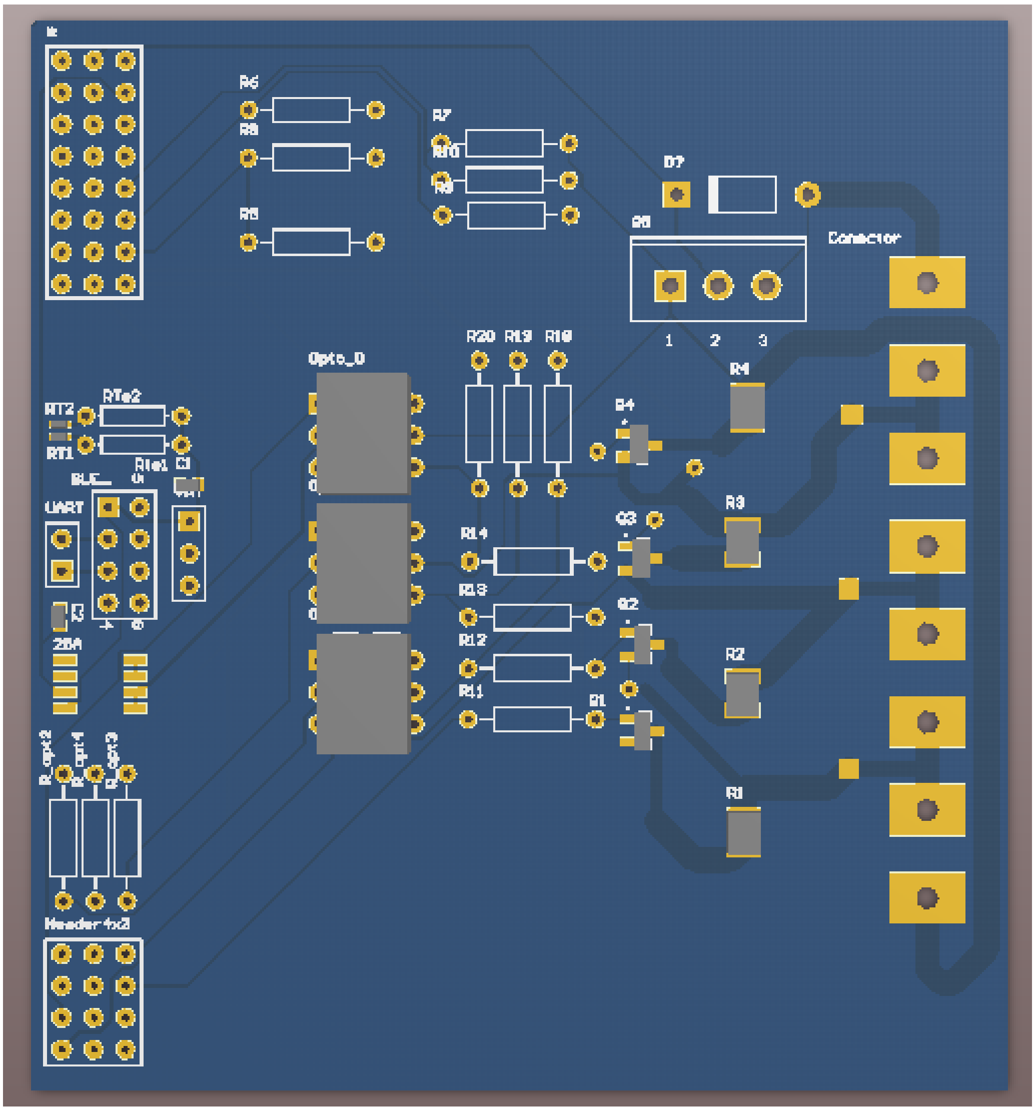


Figura 4: 3D PCB.

4 Código

Código principal:

```
#define FCY 29641200L                          //number of instructions per milisec
#define FOSC (FCY*4)                           //number of clock cycles
#define UART_BUFFER_SIZE 256                   //UART receive buffer size
#define BAUDRATE 115200                        //baudrate board is running at
#define BRGVAL ((FCY/BAUDRATE)/16)-1           //equivalent baudrate constant for
    setting correct uart baudrate
#define SETUP_BUF_LENGTH 20                    //auxiliary buffer size
#define M_SEC FCY*0.01 //0.001                 //insctruction in 10ms (0.01s))
#define RX_BUFF_LENGTH 60

#include <p30F4011.h>            //defines dspic registers
#include <stdio.h>               //standart IO library C
#include <libpic30.h>            //C30 compiler definitions
#include <uart.h>                //UART (serial port) function and utilities library
```

```
#include <timer.h>          //timer library
#include <string.h>
#include <math.h>
#include "funcoes_controlo_celulas.h"     //header com funcoes utilizadas no programa
#include <limits.h>

//Configuration settings
#pragma config FCKSMEN=CSW_FSCM_OFF
#pragma config FOS=PRI                        //fonte e o cristal
#pragma config FPR=XT_PLL16                   //oscilador a 16x cristal
#pragma config WDT=WDT_OFF                    //watchdog timer off
#pragma config MCLRE=MCLR_EN                  //turn MCLR pin ON and
#pragma config FPWRT=PWRT_OFF

unsigned int UMODEvalue, U2STAvalue, str_pos = 0;    //auxiliary UART config variables

char RXbuffer[RX_BUFF_LENGTH];


float temp_ref = 40.0;
int check_flag = 0;
int interrupt_1 = 0;
int interrupt_3 = 0;

volatile unsigned int valor=0;
volatile unsigned char auxi[20];
volatile unsigned char aux[20];
volatile unsigned int UART_write = 0;      //pointer in UART receive buffer
volatile unsigned int UART_read = 0;        //pointer in UART transmit buffer
volatile unsigned char UARTbuffer[UART_BUFFER_SIZE];    //Receive buffer size

void UART1_config(void);                              //sets up basic UART settings
void UART_send(char *UARTdata, int n);                //sends UART msg
void __attribute__((interrupt, auto_psv)) _U1RXInterrupt(void);
//interupts program when character is received via UART
void timer1_init(void);                              //sets up timer1 setting


//Funcao para limpar o RXbuffer

void cleanRX(){

    int i = 0;
    for(i = 0;i < RX_BUFF_LENGTH;i++){

        RXbuffer[i] = '\0'; //Inicializa cada posicao no RXbuffer.

    }

    str_pos = 0;            //Inicializa a posicao no buffer.

}

void config_timer1(){

    T1CONbits.TSIDL = 0;  //continue in idle mode
    T1CONbits.TGATE = 0;  //disable gated time accumulation
    T1CONbits.TCS = 0;     //use internal clock (TSYNC is ignored)
    T1CONbits.TCKPS = 1;   //prescaler
    IEC0bits.T1IE = 1;     //interrupt enable timer1
    IFS0bits.T1IF = 0;     //clear interrupt flag
    PR1 = 10000;       //valor final do timer1 (FCY/PRESCALE)
    TMR1 = 0;          //valor inicial do timer1
    T1CONbits.TON = 1;     //starts timer1

}
void config_timer2(){

    T2CONbits.TSIDL = 0; //continue in idle mode
    T2CONbits.TGATE = 0; //disable gated time accumulation
    T2CONbits.TCS = 0;   //use internal clock (TSYNC is ignored)
    T2CONbits.TCKPS = 1; //prescaler
    IEC0bits.T2IE = 0; //interrupt disable timer2
    IFS0bits.T2IF = 0; //clear interrupt flag
    PR2 = 2929; //valor final do timer2 (FCY/PRESCALE)
    TMR2 = 0;   //valor inicial do timer2
    T2CONbits.TON = 1; //starts timer2

}

void init_TMR3() {

    T3CON = 0;                // Clear the Timer 1 configuration
    T3CONbits.TCKPS = 3;      // internal Fcy divider (pre-scaler)


    IEC0bits.T3IE = 1;        //Enable interrupt
    IFS0bits.T3IF = 0;


    TMR3 = 0x0000;            // Sets timer value to zero
    PR3 = 32000;              // Timer Period
    T3CONbits.TON = 1;        // turn on timer 1

}

void config_PWM(){

    OC1CONbits.OCSIDL = 1; //disable output compare in idle
    OC1CONbits.OCTSEL = 0; //use timer2
    OC1CONbits.OCM = 6;    //pwm without fault mode
    OC1R = 0;              //valor inicial
    OC1RS = PR2;             //duty cycle
    OC2CONbits.OCSIDL = 1; //disable output compare in idle
    OC2CONbits.OCTSEL = 0; //use timer2
    OC2CONbits.OCM = 6;    //pwm without fault mode
    OC2R = 0;              //valor inicial
    OC2RS = PR2/2;         //duty cycle
    OC3CONbits.OCSIDL = 1; //disable output compare in idle
    OC3CONbits.OCTSEL = 0; //use timer2
    OC3CONbits.OCM = 6;    //pwm without fault mode
    OC3R = 0;              //valor inicial
    OC3RS = PR2/3;         //duty cycle
    OC4CONbits.OCSIDL = 1; //disable output compare in idle
    OC4CONbits.OCTSEL = 0; //use timer2
    OC4CONbits.OCM = 6;    //pwm without fault mode
    OC4R = 0;              //valor inicial
    OC4RS = PR2/4;          //duty cycle

}

void init_UART2(){

   // init_bib();
    /* Serial port config */
    UMODEvalue = UART_EN & UART_IDLE_CON & UART_NO_PAR_8BIT;   //activates the uart in continuos mode (no sleep) and 8bit no parity mode
    U2STAvalue = UART_INT_TX & UART_TX_ENABLE & UART_INT_RX_CHAR & UART_RX_TX;   //activates interrupt of pin Tx + enables Tx + enable Rx interrupt for every char
    OpenUART2(UMODEvalue, U2STAvalue, 15);   //configures and activates UART2 at 115000 bps



    //BRG = 15 (value changed to several values)
    U2STAbits.URXISEL = 1;
    _U2RXIE = 1; //0-Interruption off, 1-Interruption on
    U2MODEbits.LPBACK = 0; //disables hardware loopback on UART2. Enable only for tests
    __C30_UART = 2; //define UART2 as predefined for use with stdio library, printf etc

    printf("\n\rSerial port ONLINE \n"); //to check if the serial port is working

}

void configure_adc() { // ADC 12-bits

    //****************************//
    //ADCON1: A/D Control Register 1
    //****************************//

    ADCON1bits.ADON = 1; //**A/D Operating Mode bit**//
    // 0-A/D converter is off
    // 1-A/D converter module is operating

    ADCON1bits.ADSIDL = 0; //**Stop in Idle Mode bit**//
    // 0-Continue module operation in Idle mode
    // 1-Discontinue module operation when device enters Idle mode

    ADCON1bits.FORM = 0; //**Sata Output Format bits**//
    // 0-Integer
    // 1-Signed integer
    // 2-Fractional
    // 3-Singed fractional

    ADCON1bits.SSRC = 7; //**Conversion Trigger Source Select bits**//
    // 0-Clearing SAMP bit ends sampling and starts conversion
    // 1-Active transition on INT0 pin ends sampling and starts conversion
    // 2-General purpose Timer3 compare ends sampling and starts conversion
    // 3-Motor Control PWM interval ends sampling and starts conversion
    // 4-Reserved
    // 5-Reserved
    // 6-Reserved
    // 7-Internal counter ends sampling and starts conversion (auto convert)

    ADCON1bits.ASAM = 0; //**A/D Sample Auto-Start bit**//
    // 0-Sampling begins when SAMP bit set
    // 1-Sampling begins immediately agter last conversion completes. SAMP bit is auto set.

    ADCON1bits.SAMP = 0; //**A/D Sample Enable bit**//
    // 0-A/D sample/hold amplifiers are holding
    // 1-At least one A/D sample/hold amplifier is sampling
    //***************************//
    //ADCON2: A/D Control Register 2
    //***************************//

    ADCON2bits.VCFG = 0; //**Voltage Reference Configuration bits**//
    // 0- AVdd          AVss
    // 1- External_Vref+_pin     AVss
    // 2- AVdd          Esternal_Vref-_pin
    // 3- External_Vref+_pin     External_Vref-_pin
    // (4-7)- AVdd          AVss

    _CSCNA = 0; //**Scan Input Selections for CH0+ S/H Input for MUX A Multiplexer Setting bit**/
    // 0-Do not scan inputs
    // 1-Scan inputs

    _BUFS = 0; //**Buffer Fill Status bit**//
    // 0-A/D is currently filling buffer 0x0-0x7, user should access data in 0x8-0xF
    // 1-A/D is currently filling buffer 0x8-0xF, user should access data in 0x0-0x7

    _SMPI = 0; //**Sample/Convert Sequences Per Interrupt Selection bits**//
    // 0-Interrupts at the completion of conversion for each sample/convert sequence
    // 1-Interrupts at the completion of conversion for each 2nd sample/convert sequence
    // ...
    // 14-Interrupts at the completion of conversion for each 15th sample/convert sequence
    // 15-Interrupts at the completion of conversion for each 16th sample/convert sequence

    _BUFM = 0; //**Buffer Mode Select bit**//
    // 0-Buffer configured as one 16-word buffer ADCBUF(15...0)
    // 1-Buffer configured as one 8-word buffer ADCBUF(15...8), ADCBUF(7...0)

    _ALTS = 0; //**Alternate Input Sample Mode Select bit**//
    // 0-Always use MUX A input multiplexer settings
    /* 1-Uses MUX A input multiplexer settings for first sample, then alternate between MUX B and
    MUX A input multiplexer settings for all subsequent samples*/

    //***************************//
    //ADCON3: A/D Control Register 3
    //***************************//

    _SAMC = 31; //**Auto Sample Time bits**//
    // (0-31) Tad

    _ADRC = 0; //**A/D Conversion Clock Source bit**//
    // 0-Clock derived from system clock
    // 1-A/D internal RC clock

    _ADCS = 63; //**A/D Conversion Clock Select bits**//
    // (1-64)*Tcy/2
    // Usou-se Tad=1000ns, logo obtem-se ADCS=14
    //******************************//
    //ADCHS: A/D Input Select Register
    //******************************//

    _CH0NB = 0; //**Channel 0 Negative Input Select
    _CH0SB = 0;
    _CH0NA = 0;
    _CH0SA = 0;

    //************************************//
    //ADPCFG: A/D Port Configuration Register
    //*************************************//
    // Este registo e configurado na funcao configure_adc_channel //
    //*************************************//
    //ADCSSL: A/D Input Scan Select Register
    //*************************************//
    // Este registo nao precisa ser configurado, pois o CSCNA e 0 //

}

void configure_adc_channel(int channel){

    TRISB |= (1 << channel);            // ADC_CHANNEL defined as input
    ADPCFG &= ~(1 << channel);          // ADC_CHANNEL defined as analog

}

int read_adc(int channel){

    int x;
    _CH0SA = channel;           //CH0SA = channel vai ler o AN(Channel), caso seja 0 le o valor do AN0
    _SAMP = 1;
    while (_SAMP);
    while (!_DONE);
    x = ADCBUF0;

    return x;

}

//Descarga das celulas
int descarga_tensao(int vec[4]){

    float tensao_total = 0;
    float tensao_med = 0;
    float tensao_min = tensao[0];
    int k;


    //descobre tensao minima
    for(k=1;k<4;k++){
        if(tensao[k] < tensao_min){         //verifica se as tensoes nas celulas sao superiores a tensao minima
```

```
            tensao_min = tensao[k];          //regista a tensao minima
        }
    }

    //faz tensao total
    for(k=0;k<4;k++){
        tensao_total = tensao_total + tensao[k];

    }

    tensao_med = tensao_total/4;                          //calcula a tensao media


    for(k=0;k<4;k++){

        if(tensao[k] > tensao_med ){              //caso a tensao na celula seja superior a media, abre a celula
                vec[k]=1;                         //ate a tensao descer entre a tensao media e a minima
        }
        else if(tensao[k] <   tensao_med)
            vec[k]=0;
    }

    return tensao_med;

}

//Carga das celulas
void carga_tensao(int vec2[4]){

    int tensao_total2 = 0;
    int tensao_med2 = 0;
    int tensao_min2 = tensao[0];
    int tensao_ref = 0;
    int k,l,m;

    for(k=1;k<4;k++){
        if(tensao[k] < tensao_min2){        //verifica se as tensoes nas celulas sao superiores a tensao minima
            tensao_min2 = tensao[k];        //regista a tensao minima
        }
    }



    for(l=0;l<4;l++){
        tensao_total2 = tensao_total2 + tensao[l];
    }

    tensao_med2 = tensao_total2/4;
    tensao_ref = tensao_med2 + tensao_min2;          //calcula a tensao de referencia

    for(m=0;m<4;m++){
        if(tensao[m] > tensao_ref){            //caso a tensao na celula seja superior a de referencia, abre a celula

            //ate a tensao descer entre a tensao de referencia e a minima
            while(tensao[m]  > tensao_min2){
                vec2[m] = 1;
            }
        }
    }
    return;
}

//Funcao que converte o valor da temperatura de analogico para digital
float get_temp(unsigned int ADCvalue){

    float R = 0;
    float Vo;
    float Vin = 4.95;
    float Ro = 10000;
    float To = 298.15;
    float B = 3435;
    float T;

    // * Note: enable and switching time can take up to ~20ns which is faster than the clock frequency.
    // * Still, consider adding a delay here if measurements are not consistent.

    // * Note: Acquisition and calculations may take too long to be worthwhile.
    // * Measure/compare gains and use table if needed.

    //New Code for temperature Calculation
    Vo = Vin*(((float)ADCvalue)/1024);
    R =  Ro/((Vin - Vo)/(float)Vo);
    T = (float)(B*To)/(float)(To*log(R/Ro)+B);

    return (float)(((T - 273.15))*10);                  //Valor da temperatura em graus celsius

}

float get_val( unsigned int ADCvalue){

    return (float)(4.94 * ((float)ADCvalue)/1024);      //converte de bits para volts
}

//funcao de divisor de tensao
float get_voltage(int r1, int r2 , float val ){

    return (float) (val* ((float)r2 / (float)(r1 + r2)));
}

//calcula a corrente
float get_curr(int ADCvalue)
{
    //faz cenas

    return ADCvalue;

}

//Junta as funcoes get_val e get_voltage e retorna o valor final da tensao
float bms_voltage(unsigned int ADCvalue, int r1, int r2 ){

    float adc_val = 0;

    adc_val = get_val( ADCvalue);

    return get_voltage(r1, r2, adc_val );
}

//Funcao que verifica se alguma celula esta acima da temperatura de referencia
float analise_temp(){

    float temp_max = 0;
    int j = 0;

    for(j=0;j<SIZE_TEMP;j++){

        if(temp[j] >= temp_max){
            temp_max = temp[j];
        }
    }


    if (temp_max > temp_ref){

        //SHUT DOWN
        _LATB0 = 1;   //abrir mosfet (deixa de passar corrente)
        check_flag = 1;
        printf("Check flag: %d\n", check_flag);

    }

    return temp_max;
}

void timer1_init(void){

  T1CONbits.TCS   = 0;     /* use internal clock: Fcy              */
  T1CONbits.TGATE = 0;     /* Gated mode off                       */
  T1CONbits.TCKPS = 3;     /* prescale 1:1                         */
  T1CONbits.TSIDL = 0;     /* don't stop the timer in idle         */

  TMR1 = 0;          /* clears the timer register            */

  PR1 = M_SEC;         /* value at which the register overflows *

                    * and raises T1IF                       */

  /* interruptions */
  IPC0bits.T1IP = 2;       /* Timer 1 Interrupt Priority 0-7       */
  IFS0bits.T1IF = 0;       /* clear interrupt flag                 */
  IEC0bits.T1IE = 1;       /* Timer 1 Interrupt Enable             */
  T1CONbits.TON = 1;       /* starts the timer                     */

  return;
}

/*********************************************************************
 * Name:     UART1_config
 * Args:     -
 * Return:   -
 * Desc:     Configures UART channel 1.
 *********************************************************************/
```

```
void UART1_config(void){
  //configures LED to denote USD/UART activity

  TRISFbits.TRISF2 = 1;
    TRISFbits.TRISF3 = 0;
  LATFbits.LATF2 = 0;

  U1BRG = BRGVAL;

  U1MODEbits.PDSEL = 0;    //8-bit data, no parity
  U1MODEbits.STSEL = 0;    //1 Stop-bit
  U1MODEbits.USIDL = 0;    //Continue operation in idle mode
  U1MODEbits.ALTIO = 0;    //Use U1TX and U1RX only
  IEC0bits.U1TXIE = 0;     //No interrupt when transmitting
  U1STAbits.UTXISEL = 0;     //Interrupt when a character is transferred to the Transmit Shift register

  IEC0bits.U1RXIE = 1;     //Enable UART Receive interrupt
  IPC2bits.U1RXIP = 5;     //UART1 Receiver Interrupt Priority is 5
  U1STAbits.URXISEL = 0;     //Interrupt flag bit is set when a character is received
  IFS0bits.U1RXIF = 0;     //clear the Rx Interrupt Flag
  U1MODEbits.UARTEN = 1;     //Enanble UART
  U1STAbits.UTXEN = 1;     //UART transmitter enabled, UxTX pin controlled by UART (if UARTEN = 1) */

  return;
}

/******************************************************************
 * Name:     UART_send
 * Args:     char *UARTdata, int n
 * Return:   -
 * Desc:     Sends n bytes by UART. No security checks!
 ******************************************************************/
void UART_send(char *UARTdata, int n){

  int i=0;

  while(i<n){
    while(U1STAbits.UTXBF == 1); /* hold while buffer is full */
    U1TXREG = UARTdata[i];
    i++;
  }
}


//Envia UART para o BLE
void send2BLE(){

    char command[60];    //onde vai ser construido o commando completo
    char values[50];     //onde vao ser postos os valores  a enviar
    int z=0;             //iteradora
    int auxint=0;        //var auxiliar

    strcpy(command, "SUW,010203040506070809000A0B0C0D0E0F,");        //copia comando base

  //tensao
  for(z = 0 ; z<4; z++){

        auxint=(int) (tensao[z]*10);            //converte float x10 em int
        if(auxint<0)                            //faz valor ser positivo
      auxint*=-1;

    sprintf(values, "%d", auxint);          //converte em string

        if(strlen(values)!=2){                          //se numero e pequeno

          values[1]=values[0];                     //passa de  4\0 para 04\0
            values[0]='0';
            values[2]='\0';
    }
    strcat(command, values);                       //concatena valor
  }

  //temperatura media
  auxint=(int) (((temp[0]+temp[1])/2)*10);    //converte float x10 em int

  if(auxint<0)                          //faz valor ser positivo
      auxint*=-1;

  sprintf(values, "%d", auxint);        //converte em string

    if(strlen(values)!=2){                    //se numero e pequeno

        values[1]=values[0];                  //passa de  4\0 para 04\0
        values[0]='0';
        values[2]='\0';
  }
  strcat(command, values);                 //concatena valor

    //corrente
    auxint=(int) (curr*10);            //converte float x10 em int

    if(auxint<0)                         //faz valor ser positivo
      auxint*=-1;

    sprintf(values, "%d", auxint);        //converte em string

    if(strlen(values)!=2){                  //se numero e pequeno

    values[1]=values[0];                //passa de  4\0 para 04\0
        values[0]='0';
        values[2]='\0';
  }

    strcat(command, values);                  //concatena valor
    strcat(command, "\r\n");
    printf("\r\nSending %s\n",command);
    UART_send(command, strlen(command));
}



void init_BLE(){

  __delay_ms(1500);
    UART_send("+\r\n", strlen("+\r\n"));                  //echo toggled on
    __delay_ms(100);
    UART_send("S-,BMS\r\n", strlen("S-,BMS\r\n"));     //sets internal name to BMS
    __delay_ms(100);
    UART_send("SB,4\r\n", strlen("SB,4\r\n"));        //set baudrate to 115200
    __delay_ms(100);
    UART_send("SF,1\r\n", strlen("SF,1\r\n"));      //factory reset of some settings
    __delay_ms(100);
    UART_send("SR,00000000\r\n", strlen("SR,00000000\r\n"));       //peripheral, NO autoadvertise, no MLDP , no UART flow control
    __delay_ms(100);
    UART_send("SS,C0000001\r\n", strlen("SS,C0000001\r\n"));     //enables creation of private services and characteristics
    __delay_ms(100);
    UART_send("PZ\r\n", strlen("PZ\r\n"));      //clears all previous private services and characteristics
    __delay_ms(100);
    UART_send("PS,11223344556677889900AABBCCDDEEFF\r\n", strlen("PS,11223344556677889900AABBCCDDEEFF\r\n"));         //creates private service
    __delay_ms(100);
    UART_send("PC,010203040506070809000A0B0C0D0E0F,02,06\r\n", strlen("PC,010203040506070809000A0B0C0D0E0F,02,06\r\n"));     //creates private characteristic (02 readable)(6 bytes of data))
    __delay_ms(100);
    UART_send("SN,BMS\r\n", strlen("SN,BMS\r\n"));      //set external name
    __delay_ms(100);
    UART_send("R,1\r\n", strlen("R,1\r\n"));            //reset BLE module
    __delay_ms(2000);                                   //give it time to Restart
    UART_send("A\r\n", strlen("A\r\n"));                //advertise
}


//Envia 0 (fechar) ou 1 (abrir) para o gate dos Mosfets
void controlMosfet(int vec[4]){

    if(vec[0]==1){
      LATEbits.LATE1 = 1;
    }else
    {
      LATEbits.LATE1 = 0;
    }
    if(vec[1]==1)
    {
         LATEbits.LATE2 = 1;
    }else
    {
        LATEbits.LATE2 = 0;
    }

     if(vec[2]==1)
    {
         LATEbits.LATE3 = 1;
    }else
    {
        LATEbits.LATE3 = 0;
    }

     if(vec[3]==1)
    {
         LATEbits.LATE4 = 1;
    }
     else
    {
        LATEbits.LATE4 = 0;
    }

}

int main(){

    int milh=1000000;           //1 Mega Ohm
    int resist[6] = {1.904*milh, 1.246*milh, 4*milh, 1.246*milh, 6.02*milh, 1.246*milh};          //Resistencias do divisor de tensao a entrada do AN
    int k = 0;
    int abrir_celulas[4];         //holds which mosfets are ON or OFF
    int aux = 0;
    float tensao1 = 0;


    RCONbits.SWDTEN=0;           //Enable Watchdog timer

    cleanRX();
    init_UART2();              //used for debugging with putty
    UART1_config();            //configure basic UART
    init_TMR3();               //inicia a funcao do timer e cada vez que chega a 3200 vai ao interrupt e mete a flag a 0
    configure_adc();
    config_timer1();
    config_timer2();
    //config_PWM();

    init_BLE();                //sets up RN4020

    //define controlo para os mosfets
    TRISEbits.TRISE1 = 1;    //1 - output
    TRISEbits.TRISE2 = 1;      //1 - output
    TRISEbits.TRISE3 = 1;      //1 - output
    TRISEbits.TRISE4 = 1;      //1 - output


    //inicializa mosfets com em OFF
    for(aux = 0;aux<4;aux++){
        abrir_celulas[aux] = 0;
    }


    //configures 6 ADC channels
    for(aux=0;aux<6;aux++)
        configure_adc_channel(aux);

    //Main Loop
    while(1){

        //if interruption happened
        if (interrupt_3){
            interrupt_3=0;          //reset interruption

            //get actual current values
            //reads from adc and applies voltage divider
            tensao[0] = get_val(read_adc(0));
            temp[0] = get_temp(read_adc(1));
            tensao[1] = bms_voltage(read_adc(2), resist[0], resist[1]);
            temp[1] = get_temp(read_adc(3));
            tensao[2] = bms_voltage(read_adc(4), resist[2], resist[3]);
            tensao[3] = bms_voltage(read_adc(5), resist[4], resist[5]);

            curr = get_curr(read_adc(6));  //get current from AN6

            //printf zone
            for (k=0;k<4;k++){
                printf("\n\rtensao:\t %f", tensao[k]);
            }
            for (k=0;k<2;k++){
                printf("\n\rtemperatura:\t %f", temp[k]);
            }
            printf("\n\rCorrente:\t %f", curr);

//compara o valor maximo da temperatura com o valor obtido e abre ou fecha o mosfet geral
            analise_temp();
            check_flag=0;
            tensao1 = descarga_tensao(abrir_celulas);   //define que mosfet abrir e fechar
            controlMosfet(abrir_celulas);               //abre ou fecha mosfets
            send2BLE();
        }
    }
    return 0;
}


/*****************************************************

 ************     Interruptions    ******************

 *****************************************************/

/* This is UART2 receive ISR */
// UART Interruption handler sprintf(st_valor, "%d\r\n", tensaoint);
void __attribute__((__interrupt__,auto_psv)) _U2RXInterrupt(void){

    IFS1bits.U2RXIF = 0;  //resets and reenables the Rx2 interrupt flag
    //Read the receive buffer until at least one or more character can be read
    while(U2STAbits.URXDA){

        RXbuffer[str_pos] = U2RXREG;  //stores the last received char in the buffer
        //printf("%c", RXbuffer[str_pos]);  //prints the last received char
        str_pos++;                          //increments the position in the buffer to store the next char

        if(str_pos >= 80){
            str_pos = 0;    //if the last position is reached then return to initial position
        }
    }
}

//Timer 1 Interrupt handler
void __attribute__((interrupt, auto_psv, shadow)) _T1Interrupt(void){

    IFS0bits.T1IF = 0; //clear interrupt flag
    interrupt_1 = 1;
}

//Timer 3 Interrupt handler
void __attribute__((interrupt, no_auto_psv)) _T3Interrupt(void) {

    IFS0bits.T3IF = 0; //Clears interrupt flag
    interrupt_3 = 1;
}

/*******************************************************************
 * Assign UART1 interruption
 *******************************************************************/

void __attribute__((interrupt, auto_psv)) _U1RXInterrupt(void){

  UARTbuffer[UART_write] = U1RXREG;
  UART_write = (UART_write+1)%UART_BUFFER_SIZE;
  IFS0bits.U1RXIF = 0;     /* clear the Rx Interrupt Flag */
  return;
}
```

Código do header "funções controlo células":

```
#ifndef __FUNCOES_CONTROLO_CELULAS_H__
#define __FUNCOES_CONTROLO_CELULAS_H__

#define SIZE_TEMP 2
#define SIZE_TENSAO 4


float tensao[4];          //tensao

float temp[2];        //temperatura

float curr;        //corrente

void cleanRX();

void config_timer1();    //configuracao do timer 1

void config_timer2();    //configuracao do timer 2

void init_TMR3();

void config_PWM();

void init_UART2();

void configure_adc();

void configure_adc_channel(int channel);

int read_adc(int channel);

void analise_tensao(int vec[4]);

float get_temp(unsigned int ADCvalue);

float analise_temp();
```

```

void send2BLE();

float get_curr(int ADCvalue);

#endif
```

Android App

Todo o código para a aplicação pode ser encontrado comentado em Aplicação BMS Monitoring. O ficheiro apk necessário para a instalação da aplicação encontra-se aqui.