```
//P14219 M.A.R.S.U.P.I.A.L Power Management Controller Firmware
//Rev 1.2

//Changes:
//Incorperated Ros_Lib
//Formatting
//made chnages to allow for more than two batteries
//reduced data size stored in EEPROM
//changed millis to micros for faster time base

/*************************************************************************/
/*****************************Defines and Constants************************/
/*************************************************************************/
#include <EEPROM.h>
#include <ros.h>
#include <std_msgs/Int32.h>
#include <std_msgs/Byte.h>


/*************************ROS CONFIG*************************************/
ros::NodeHandle nh;

std_msgs::Int32 volts_msg;
std_msgs::Int32 amps_msg;
std_msgs::Byte level_msg;

ros::Publisher pub_volts("Vbatt", &volts_msg);
ros::Publisher pub_amps("Ibatt", &amps_msg);
ros::Publisher pub_level("BattLevel", &level_msg);
```

```
/************************Scaling Macros*************************************/
#define vbattScale(ADC) 40 * ADC //macro to convert ADC reading to mV
#define vbattPin   1   //analog pin connected to Vbatt divider

#define ibattScale(ADC) 391 * (ADC - 512)   //macro to convert ADC reading to mA
#define ibattPin   0   //analog pin connected to Ibatt sensor


/************************Timing Intervals*************************************/

#define sampleInterval   1000U  //interval in us between vbatt and ibatt samples
#define calcInterval  10000U  //interval in us between recalculations
#define pubInterval 100000U
#define vCheckInterval 30000 //interval to check that battery voltage is above minimum

/************************Serial Comms Stuff*************************************/

#define ackByte    'a' // (ascii a = 97 Decimal)byte sent from system when polling to trigger serial tx

/************************Battery Parameters*************************************/

#define defaultBattCap 97920UL //charge capacity of battery in coulombs ie 4(batteries)*6.8(Ah)*60^2
#define vBattMin 24000 //Minimum Battery voltage in mV

/************************uC Configuration stuff*************************************/

const byte offPins[] = {
  2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, A2, A3, A4, A5}; // unused pins
```

```
/*******************************Variables*************************************/

long Capacity = 0L; //holds battery capacity in mC

unsigned long sampleTime = 0; //for storing last sample time
unsigned long calcTime = 0;   //for storing last calculation time
unsigned long pubTime = 0; //for storing last publish time

unsigned long currentTime = 0;
byte dataIndex = 0;

int vbattADC[10];
int ibattADC[10];

long avgV = 0;
long avgI = 0;

long expChg = 0L;

byte battLevel = 0;

//float battTest = 0.0;



// the setup routine runs once when you press reset:
void setup() {

  nh.initNode();
```

```
  nh.advertise(pub_volts);
  nh.advertise(pub_amps);
  nh.advertise(pub_level);

  initPullups();



  Capacity = (1000L* defaultBattCap); //get battery capacity and convert to milliCoulombs

}

// the loop routine runs over and over again forever:
void loop() {

  currentTime = micros();
  //Check if serial buffer has anything waiting
  if (currentTime - pubTime > pubInterval){

    sendData(); //calls function to package and send power data via serial uart

    pubTime = micros();
    }



  //sample inputs if sampling interval exceeded then increment or restart array index
  if(currentTime - sampleTime > sampleInterval){
```

```
  //notIdle(1);
  //Serial.println("Sampling"); //DEBUG OUTPUT
  //Serial.println(dataIndex); //DEBUG OUTPUT

  sampleInputs();

  sampleTime = micros(); //update current sample time
  //notIdle(0);
 }

 //Calculate averages, charge expenditures, and battery life level if interval has expired
 if(currentTime - calcTime > calcInterval){
  //notIdle(1);
  calcValues();

  calcTime = micros();
  //notIdle(0);
 }
}

/*********************************************************************
 * ----------------------------Functions----------------------------
 *********************************************************************/


//loop to set to input and enable internal pullups of unused pins
void initPullups(){

 Serial.println("Initalizing Pullups on unused pins"); //DEBUG OUTPUT
```

```
 for (byte i = 0; i < sizeof(offPins); i++){

   //Serial.print("Initalizing pullup on:"); //DEBUG OUTPUT
   //Serial.println(offPins[i]); //DEBUG OUTPUT



   pinMode(offPins[i],INPUT);
   digitalWrite(offPins[i],HIGH);
 }
}

/*****************************************************************/
// takes care of both writing and reading capacity values from eeprom
unsigned long accessROM(unsigned int measChg, byte set){
 if (set == 1){
   if (EEPROM.read(0) != 1){
     EEPROM.write(0,1); //set flag indicating measured value present
   }
   EEPROM.write(1,highByte(measChg)); //write high byte
   EEPROM.write(2,lowByte(measChg)); //write low byte
   }
 else{
   if (EEPROM.read(0) == 1) {
    //EEPROM has a measured value saved
    return( ((unsigned int)EEPROM.read(1)<<8)+EEPROM.read(2) );


   }
   else {
```

```
    //no number has been stored previously
    return (defaultBattCap);
   }
  }
}
```

```
/*****************************************************************/
//sends data via serial write to MCU
void sendData(){

 volts_msg.data = avgV;
 amps_msg.data = avgI;
 level_msg.data = battLevel;

 pub_volts.publish(&volts_msg);
 pub_amps.publish(&amps_msg);
 pub_level.publish(&level_msg);

 nh.spinOnce();

 //Serial.println("Entered sendData() function"); //DEBUG OUTPUT

 //Serial.print("Average Voltage: "); //DEBUG OUTPUT
 //Serial.println(avgV);
 //Serial.write(highByte(avgV));
```

```
 // Serial.write(lowByte(avgV));

  //Serial.print("Average Current: "); //DEBUG OUTPUT
  //Serial.println(avgI);
  //Serial.write(highByte(avgI));
  //Serial.write(lowByte(avgI));

  //Serial.print("Capacity: ");
  //Serial.println(Capacity);

  //Serial.print("Consumed Charge: "); //DEBUG OUTPUT
  //Serial.println(expChg);

  //Serial.print("Battery Level: ");   //DEBUG OUTPUT
  //Serial.println(byte(battTest));
  //Serial.println(battTest);

  //Serial.print("Free SRAM: ");
  //Serial.println(freeRam());

}



/*************************************************************/
//sample inputs then increment or restart array index
void sampleInputs(){

  analogRead(vbattPin); //throw away first value after changing ADC Mux
```

```
 vbattADC[dataIndex] = analogRead(vbattPin);
 //Serial.print("Vbatt sample: "); //DEBUG OUTPUT
 //Serial.println(vbattADC[dataIndex]); //DEBUG OUTPUT


 analogRead(ibattPin); //throw away first value after changing ADC Mux

 ibattADC[dataIndex] = analogRead(ibattPin);

 //Serial.print("Ibatt sample: "); //DEBUG OUTPUT
 //Serial.println(ibattADC[dataIndex]); //DEBUG OUTPUT

 //increment index for storing data in array or reset if last element
 if(dataIndex == 9){
   dataIndex = 0;
 }
 else{
   dataIndex++;
 }
}

/*************************************************************/
//calculates average of 10 values sampled for vbatt and ibatt
//adds charge expenditure to running total
//subtracts charge expenditure for total availible and computes batt level
void calcValues(){

 //Serial.println("Calculations");//DEBUG OUTPUT
 //average voltage samples
```

```
avgV = 0L;
avgI = 0L;

for (byte i = 0; i < 10; i++){

  avgV = avgV + (long)vbattADC[i];
}

avgV = vbattScale((avgV / 10L));
//Serial.print("average voltage: "); //DEBUG OUTPUT
// Serial.println(avgV);



//average current samples
for (byte i = 0; i < 10; i++){

  avgI = avgI + (long)ibattADC[i];

}

avgI = ibattScale((avgI / 10));

//Serial.print("average current: "); //DEBUG OUTPUT
//Serial.println(avgI);

//add charge over interval to total
expChg = (expChg + (avgI * (calcInterval / 1000)) / 1000); //units: A*ms=mC

battLevel = (byte)(100.0 * (Capacity - expChg) / Capacity);
```

```
}

/*************************************************************/
//just function to return the free SRAM for diagnostic purposes from Adafruit
int freeRam ()
{
  extern int __heap_start, *__brkval;
  int v;
  return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
```