EDL Final Report

Tue - KT - 2 - 1

SoC Estimation of $LiFePO_4$ cells

Authors:
Parvik Dave, 190070044
Nirmal Shah, 190100085
Tirthankar Adhikari, 190070003

Faculty Advisor
Prof. Kushal Tuckley

April 11, 2022

Contents

0.1 Introduction

The aim of the project was to build a system that can estimate the State of Charge (SoC) of four $LiFePO_4$ cells connected in series. Implementing a SoC estimator has several industrial applications as a part of a BMS system. Since there is quite a boom in the EV industry, building an SoC estimator is crucial for high performance and life of the batteries in the vehicles. This industry is very competitive, and hence, our primary goal was to build a reliable SoC estimator whilst keeping the price as low as possible.

State of Charge, as is quite self-explanatory, gives an estimate of how much charge is still remaining in the battery, which can be used to estimate the run-time left. This information is used by BMS systems to ensure that the batteries are used optimally, and do not get damaged due to overuse. The BMS manages the batteries to ensure they aren't used outside their safe operating ranges, by monitoring voltage, current, SoC, and SoH (State of Health).

0.2 Block Diagram

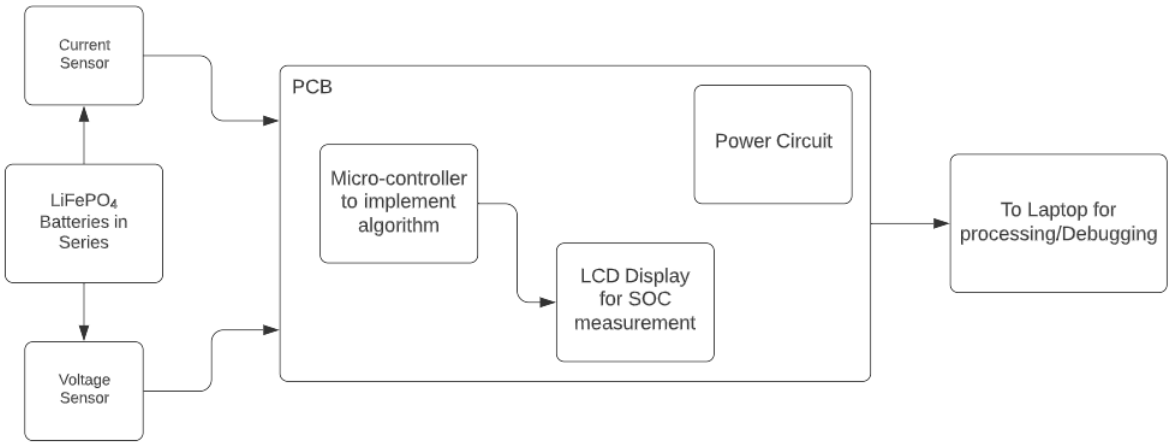

Figure 1: Block Diagram representing our various subsystems

0.3 Design Approach

As the block diagram suggests, a simple SoC estimator requires two sensor readings:

- Voltage
- Current

Voltage To measure individual cell voltages, we have used a voltage divider circuit between each of the four cells. The purpose of the voltage divider is to scale down the voltage value, so that it can be read from the ADC of the ATmega328P micro-controller. (The ADC of Arduino can only read between 0-5V, while our operating ranges were upto 15V). The positive terminal of one cell connects to the ground of the successive cell, and thus we can obtain individual cell voltages by subtracting two consecutive ADC readings.

Current To measure the overall current flowing in the circuit, we used a current sensor (ACS 712-5A). This is connected in series with the cells, and the output of the sensor then goes into the ADC of the ATmega328P, the reading of which after some computation gives the current flowing through the circuit.

Once we have obtained the current and individual cell voltages, we now need to estimate the SoC. For this, we used the Coulomb Counting method. Here, we integrate the current that has flown out of the batter so far, to get an estimate of the charge flown out of the battery. Subtracting this from the initial charge can be used to output an estimate of the current SoC.

This obtained SoC is then sent via the micro-controller to an onboard LCD which displays the current SoC, along with displaying the individual cell voltages, and the current flowing.

0.4 Subsystem Design

0.4.1 LCD for displaying SOC measurement

We used the following circuit to interface a 16× 2 LCD with the ATmega328P:
We also used the LCD in the 4-bit mode, so as to reduce the requirement of routing

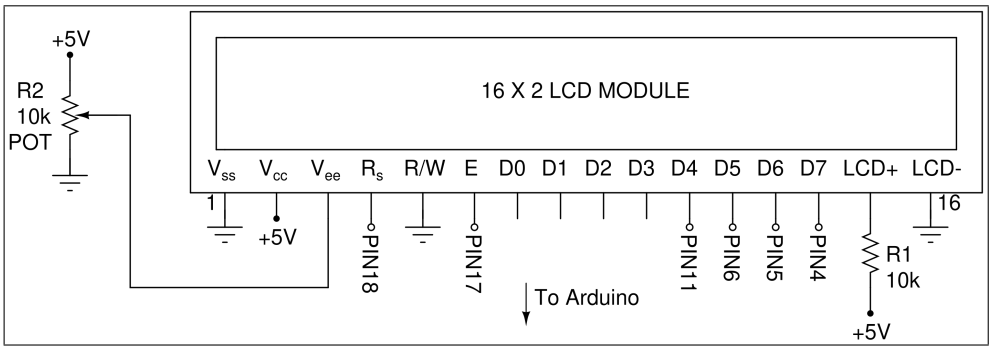

Figure 2: Circuit Diagram for LCD connections

on the PCB. A potentiometer was used for changing the contrast on the LCD, and the "LiquidCrystal" Library made available by the Arduino IDE was used to display messages on the LCD. The LCD displays all the individual cell voltages, the current passing through the load, and also displays the estimated SoC.

0.4.2 Power delivery

We have utilized the batteries themselved to supply power to the entire PCB, along with the microcontroller. This eliminates the need for any external power supply, making the entire SoC estimator highly portable and compact. The ATmega328P requires a regulated 5V supply, which we provide by using the LM7805 IC. It converts the voltage from the series connected cells to 5V, and powers the PCB. Given below is the circuit of the implementation:
According to the specifications of the batteries, the minimum voltage under safe operation is 2V, and the maximum voltage until safe operation is 3.65V. Adding the 4 cells in series would give us an input voltage in the range [8V, 14.6V]. The LM7805 converts all voltages in this range to a reliable, regulated 5V supply.

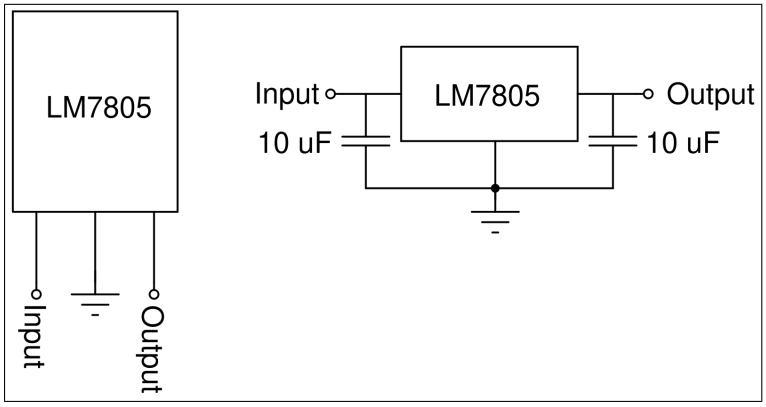

Figure 3: LM7805 pin-out

0.4.3 Micro-controller interfacing

In the initial literature review phase, we looked at various BMS ICs which were available in the market, like the BQ-series from Texas Instruments. These are very powerful ICs which can give superbly reliable estimates of SoC, perform cell balancing, and much more. These require some basic interfacing and programming. However, due to the several constraints some of which include: a very long lead time, high price, and SMD packaging (which would have been difficult to solder on WEL PCBs), we decided to use the ATmega328P, a general purpose micro-controller which is cheap, and easily available. To measure current and voltage, we used the multiplexed ADC channels of the micro-controller, and sent data to the LCD using the GPIO pins. Following is the connection of the Arduino IC to the entire PCB:

A 28-pin IC holder was used on the PCB to easily detach/attach the ATmega328P, which

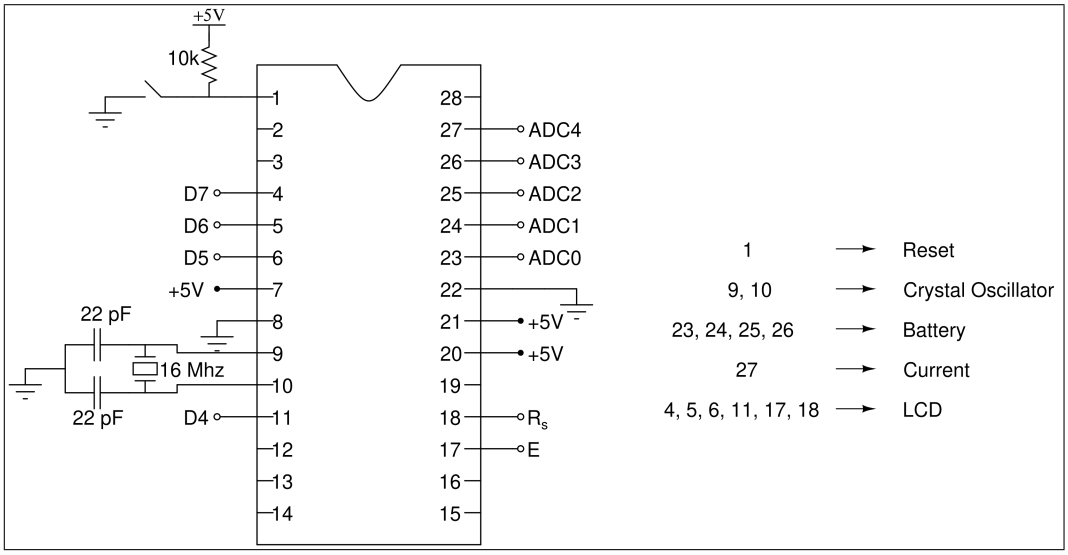

Figure 4: Interfacing of various components with the ATmega328P

we programmed by attaching to the Arduino UNO, and then uploading the code via IDE. To generate a clock, we used a 16 MHz crystal oscillator, and coupled it with 22 pF capacitors as specified in the datasheet.

0.4.4 Current and Voltage sensor

We have used an ACS712-5A to measure the current passing through the load. The sensitivity of the sensor is 185mV/A, with an offset of 2.5V. In order to obatin the current,

we need to measure the voltage using the ADC channel. The relation between the current and the voltage read is as follows:

$$I(\text{in mA}) = \frac{V_{ADC} - 2500}{185}$$

Following is the circuit used for current sensor:

The ATmega328P can measure any analog voltage up to 5V, and has a 10-bit ADC ,

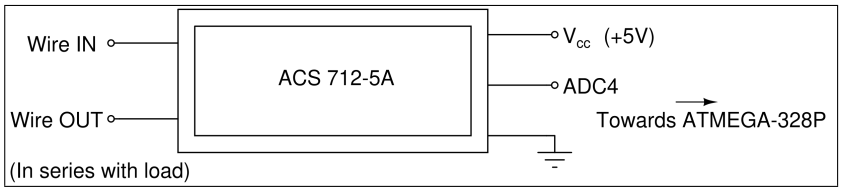

Figure 5: ACS712-5A circuit diagram

providing us with an output of 0 corresponding to 0V, and 1023 corresponding to 5V. Obviously, we cannot measure the battery voltage w.r.t GND since, the previous battery's voltages would add up, and the number would exceed 5V, rendering our micro-controller useless, and also potentially damaging the IC.

We then explored several methods to read off the individual cell voltages. We experimented using difference amplifiers to read the voltage between two cells. This approach, however, has the issue that the opamps would have to be supplied with the highest voltage, and since most opamps do not support rail-to-rail operation, the output would have saturated well before. We would also have to make sure that the opamp has very low noise in the DC region. Considering this, we decided to use a simple approach, which was to use a voltage divider to scale down the voltage value, then give it to the ADC for conversion, and then scale the voltage back up.

$$\frac{R_1}{R_1 + R_2} V_{in} = V_{out}$$

where R_1 and R_2 are the divider resistors. Using 5 band resistors having very less %tolerances in their values, we selected R_1=1.8kΩ an R_2 = 8.2kΩ. This would scale the voltage down by a factor of 0.18. The maximum voltage at any analog pin would therefore come out to 0.18$\times$14.6 = 2.68V, which is well below the 5V range. With this simple modification, we ensure that all individual cell voltages can be read. In the measurement we take average of 10 readings.

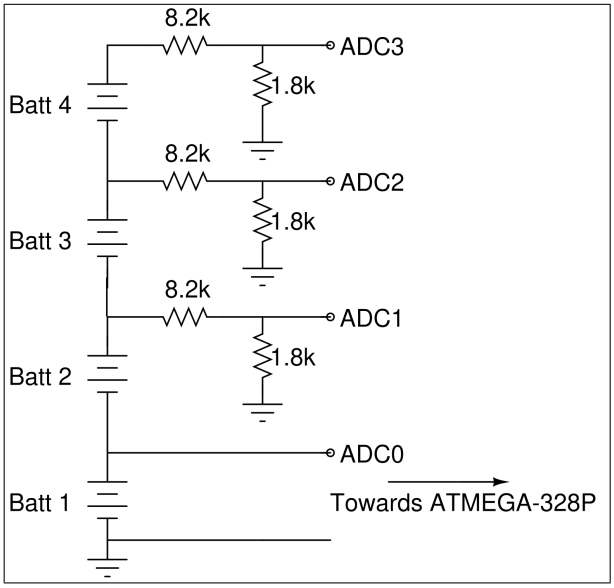

Figure 6: Voltage divider circuit

0.5 Results

Our method of estimating SoC has given fairly accurate results. The individual cell voltages we obtained from the voltage divider circuit and then passing through the ADC agree with the actual voltage (measured using a DMM) to two decimal places. We have recorded a small clip showing the cell voltages and current in our working PCB, which can be seen here

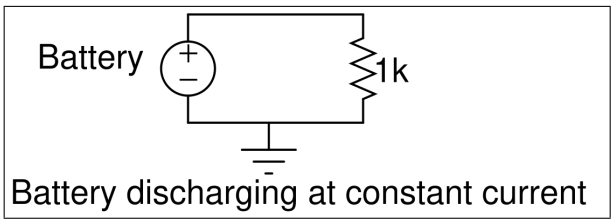

Figure 7: Discharging circuit

In order to obtain a more accurate estimate of the SoC, we tried to obtain the peukert constant of the battery. For this, we used a 1kΩ resistance to discharge the battery at constant current, and measured the voltage and current in the circuit, at regular intervals. In the Peukert method, the charge flown is adjusted by a constant k, called the Peukert constant. The relation of chrage flown is:

$$C = T \times I^k$$

Following measurements were made: Due to limits on experimental setups and accuracy, we weren't able to find the Peukert constant very accurately, but by the results we obtained, the constant evaluates to 1 approximately.

Time(min)	Voltage(V)	Current(mA)
0	3.335	3.35
5	3.334	3.35
10	3.333	3.35
15	3.331	3.35
17	3.328	3.34
19	3.328	3.34
24	3.327	3.34
36	3.327	3.35
62	3.326	3.35
80	3.325	3.35

0.6 BOM

Component	Cost (Rs.)
$LiFePO_4(\times 4)$ batteries	800
2-pin Phoenix connectors	(from WEL)
ATmega328P	(from WEL)
ACS712-5A	(from WEL)
LM7805	(from WEL)
16x2 LCD Display	(from WEL)
10k Pot	(from WEL)
Wires, resistances, and capacitors	(from WEL)
Push switch	(from WEL)
16 MHz crystal	(from WEL)

0.7 Future Work

Due to time constraints and limitations on resources, some of the planned work could not be completed. However, we do have some plans for some work that can be implemented in this project in the future:

- Using a Wi-Fi module, and transmitting SoC data to a webpage, which a user can access from his/her smartphone itself
- Make a 3D casing, so that the entire system is closed and robust. Only the battery terminals and load terminals would be open as user input
- We had also implemented passive cell balancing after going through several papers online, and implemented a model suggested by TI. The schematic for the same is ready, and in the future we could also design the PCB for the same and add it to the SoC estimator

0.8 Appendix

Below is the code used to program our ATmega328P micro controller:

```
#include <LiquidCrystal.h>
double t1,t2;
double initial_soc = 0.95; //estimate of the initial SoC, set at 95%
```

```
double total_charge = 1500*3600*initial_soc; //calculating the total charge double vbat_1,
double const k = 10.0/1.8;
const double sensitivity = 185.0; //sensitivity of the ACS712
const int offsetVoltage = 2500; //offset of the ACS712
const double charge_out = 0;
const double current_soc;
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
  // put your setup code here, to run once:
 pinMode(13,OUTPUT);
 pinMode(A0,INPUT);
 pinMode(A1,INPUT);
 pinMode(A2,INPUT);
 pinMode(A3,INPUT);
 pinMode(A4,INPUT);
 lcd.begin(16, 2);
}

void loop() {
  // put your main code here, to run repeatedly:
  t1 = millis();
  vbat_1 = 0;
  vbat_2 = 0;
  vbat_3 = 0;
  vbat_4 = 0;
  curr = 0;
  for(int i = 0; i<10; i++)
  {
    vbat_1 += analogRead(A0)*5.0/1024.0;
    vbat_2 += analogRead(A1)*k*5.0/1024.0 - vbat_1;
    vbat_3 += analogRead(A2)*k*5.0/1024.0 - vbat_2;
    vbat_4 += analogRead(A3)*k*5.0/1024.0 - vbat_3;
    curr += ((analogRead(A4)-offsetVoltage)/sensitivity);
    delay(10);
  }
  //Averaging readings over 10 cycles
  vbat_1 /= 10.0;
  vbat_2 /= 10.0;
  vbat_3 /= 10.0;
  vbat_4 /= 10.0;
  curr /= 10.0;
  t2 = millis();
  charge_out += curr * (t2-t1+1500) * 1/1000.0; //measuring charge flown out
  current_soc = (total_charge - charge_out)/(total_charge)*100.0; //subtracting from total
  //The next lines of code are for printing the various important parameters to the LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Cell 1: ");
  lcd.setCursor(0, 8);
  lcd.print(vbat_1);
  lcd.setCursor(1, 0);
```

```
  lcd.print("Cell 2: ");
  lcd.setCursor(1, 8);
  lcd.print(vbat_2);
  delay(500);

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Cell 3: ");
  lcd.setCursor(0, 8);
  lcd.print(vbat_3);
  lcd.setCursor(1, 0);
  lcd.print("Cell 4: ");
  lcd.setCursor(1, 8);
  lcd.print(vbat_4);
  delay(500);

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Current: ");
  lcd.setCursor(0, 9);
  lcd.print(curr);
  lcd.setCursor(1, 0);
  lcd.print("SoC: ");
  lcd.setCursor(1, 5);
  delay(500);
}
```

0.9 References

- https://create.arduino.cc/projecthub/akshayjoseph666/interface-16x2-lcd-parallel-interface-with-arduino-uno-2e87e2
- https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf