Elaboration of a battery management system

By jean-claude.feltes@education.lu

https://github.com/jean-claudeF/BMS

1 Steps to take

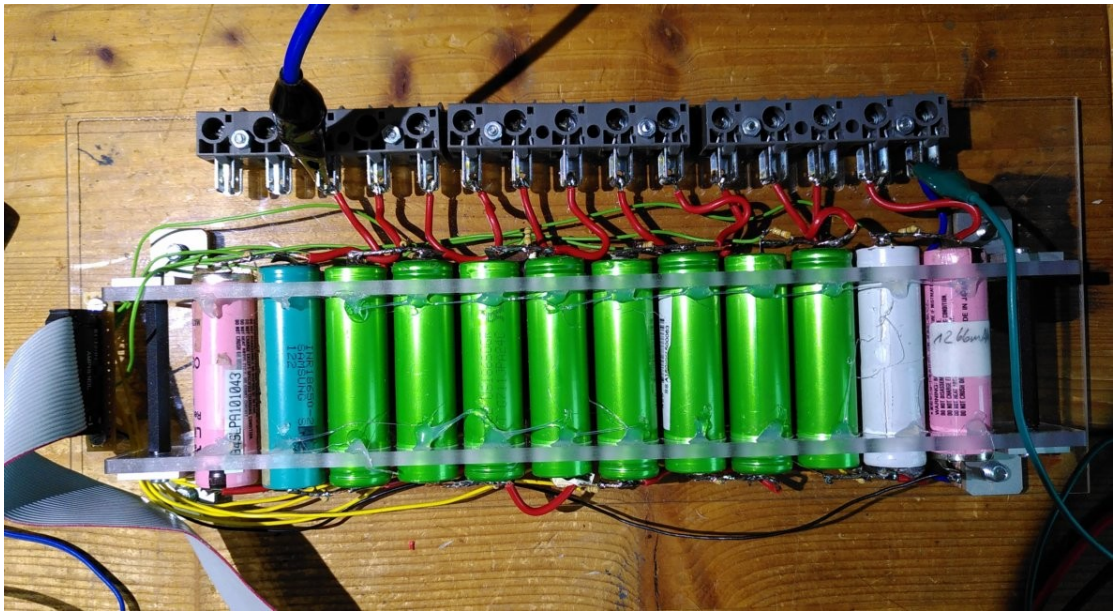
The battery consists of 12 LiIon cells.
The BMS must

- measure the voltage of each individual cell
- check this voltage for over- or underload
- switch off the charge if any cell has overvoltage
- balance the cells so that the voltage is practically equal for each cell
- switch off the load in case of undervoltage
- display informations via OLED and serial interface.

2 Test or dummy battery

As the battery on which the BMS shall be used, is potentially dangerous in case of short circuit, all development is to be done on a dummy battery, consisting of twelve (reclaimed) 18650 cells, connected in series.

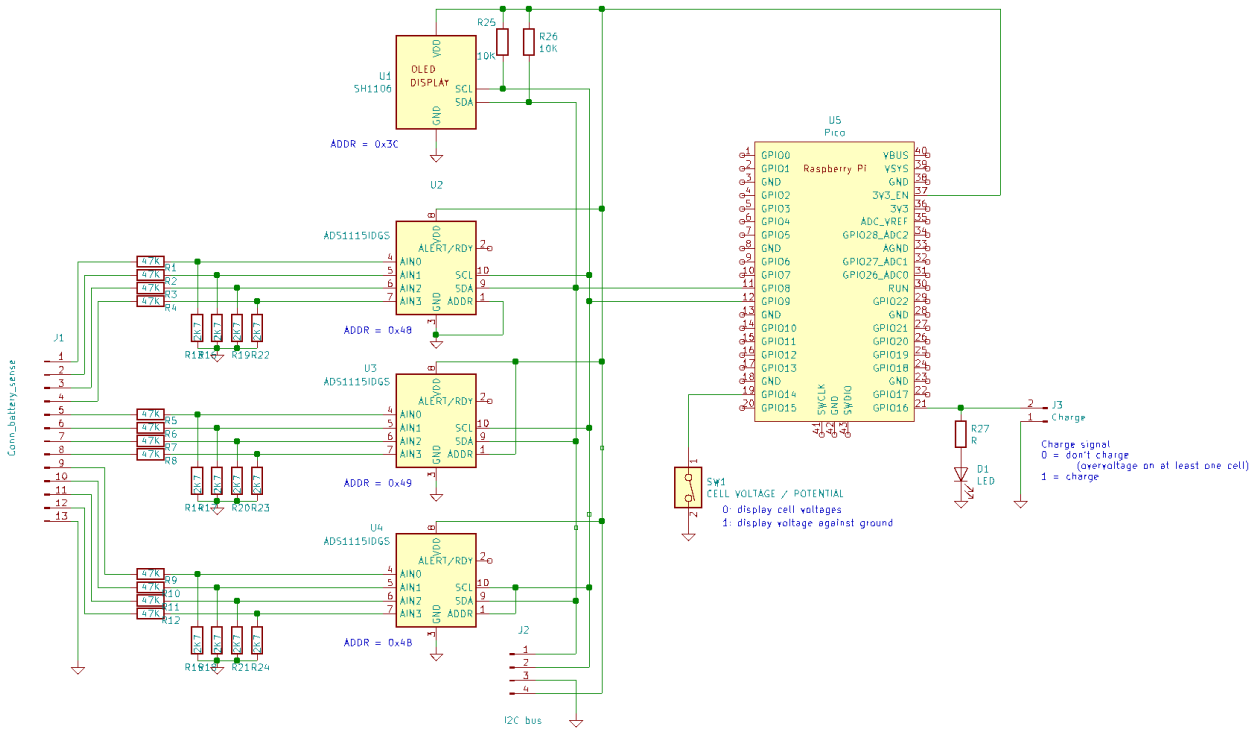
No care is taken as to use identical cells, as it is interesting to see how the circuit will cope with differences.



The battery has two connectors: one that connects directly to the cells, and one for measuring the voltages, that has 39 Ohm protection resistances in the signal connection.

3 12 channel voltmeter

A Raspi Pico programmed in Micropython is used to read the voltage of three ADC converters with 4 channels each:



The ADCs are ADS115 with theoretically 16 bit accuracy, controlled via I2C.
The addresses can be chosen out of 4 possibilities by connecting the address pin to either GND, VCC, SDA or SCL, resulting in the addresses 0x48, 0x49, 0x4A and 0x4B.

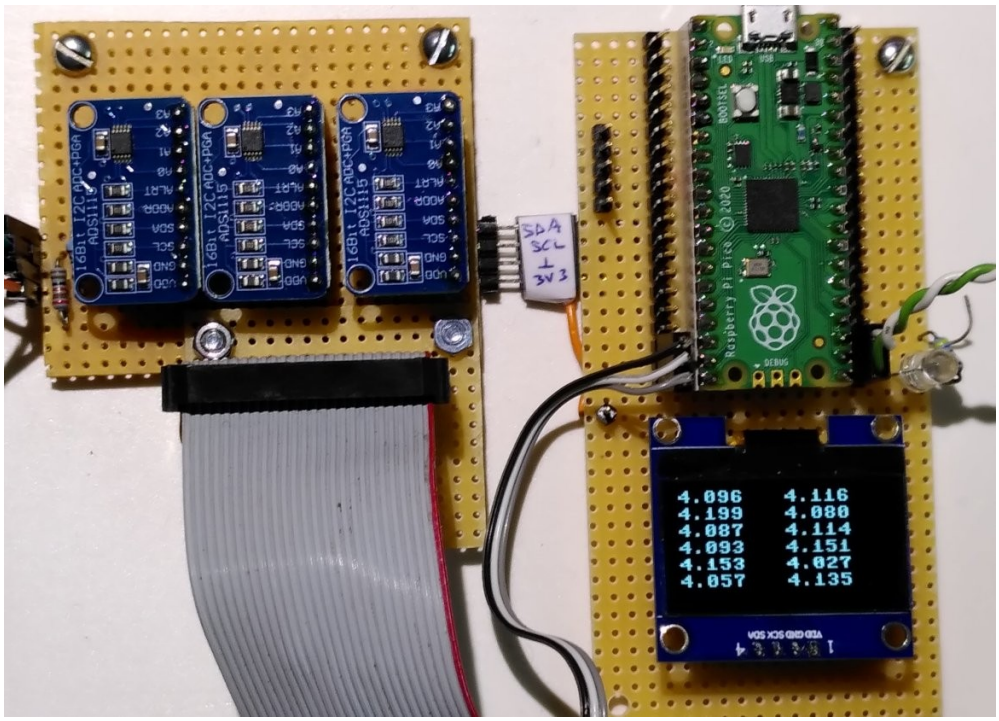
The data sheet recommends not using SDA if possible, and, indeed I had problems with this configuration, so I used the SCL connection for the third ADC.

As the maximum voltage on the battery is in the order of 50V, I use a divider 47k / 2.7k to reduce it to max. about 3V.
Without this, the accuracy was around 1mV, so with the the divider we should expect about 18-20mV inaccuracy.

The OLED display is an 1.3” display using the SH1106 driver.

Switch SW1 allows switching between display of voltages against ground or individual cell voltage display (this is what we need for the BMS)

GPIO16 will provide a signal telling if the battery should be charged or if the charge should stop (at least one cell with overvoltage).



Voltmeter software

All software is written in Micropython.

Once the hardware is ready, the I2C addresses can be controlled with i2cscan:

```
from machine import Pin, I2C
i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=100000)
addresses = i2c.scan()
for a in addresses:
    print(hex(a))
```

This is important to control the ADC addresses.

The drivers for ADC and OLED can be found on Github:

https://github.com/robert-hh/SH1106

https://github.com/robert-hh/ads1x15

The nice thing about Micropython is that it is very easy to organise the software in reusable modules.

I wrote some extensions for the OLED and the ADC lib. They can be found here:

https://github.com/jean-claudeF/BMS

The module ADC_multi_02.py provides an easy way to read all 3 ADCs in one command:

```
import time
i2c_channel = 0
sclpin = 9
sdapin = 8
```

```
i2c = I2C(i2c_channel, scl=Pin(sclpin), sda=Pin(sdapin))


addresses = [72, 73, 75]

multi_adc = Multi_ADC(i2c, addresses)
voltages = multi_adc.read_all()
for v in voltages:
    print(v)
```

The module OLED_03 has a print-like function that is very useful:

```
 from machine import Pin, I2C
import time


i2c = I2C(0, scl=Pin(9), sda=Pin(8))
oled = OLED(128, 64, i2c)

oled.clear()
for i in range(0,8):
     oled.print("TEST " + str(i))
oled.show()
```

Another function is useful for printing multiple lines in one statement:

```
 s = "Hello \t world \t ! \t 3.14   \t The answer is \t 42"
oled.print_s(s)
```

This way I can use practically the same syntax to print tabular data over the serial port (all in one line, tab separated), and to print them on the OLED display in multiple lines.

Here is a simple voltmeter program that also checks the cells for overload:

```
from ADC_multi_02 import Multi_ADC

from machine import I2C, Pin
from OLED_03 import OLED
import time

# Switch
d_switch = Pin(14, Pin.IN, Pin.PULL_UP)
charge_OK = Pin(16, Pin.OUT)
overvoltage = Pin(17, Pin.OUT)


# charge_ok = Pin(
# I2C init
i2c_channel = 0
sclpin = 9
sdapin = 8
i2c = I2C(i2c_channel, scl=Pin(sclpin), sda=Pin(sdapin))

# OLED init
oled = OLED(128, 64, i2c)
oled.clear()

# ADC init
addresses = [72, 73, 75]                # 74 does not work though it is scanned ????
multi_adc = Multi_ADC(i2c, addresses)

# voltage divider on input for ca. 50V range
factor = 49.7/2.7                # voltage divider 47k / 2.7k on input
```

```
V_MAX = 4.18
#-------------------------------------------------------------------------------
def print_voltages(volt_str, separator = '\t'):
    '''print array of voltages, tab separated'''
    i = 1
    for v in volt_str:
        print( v, end = separator)
        #print(i, ": ", v, end = separator)
        i+=1


def display_voltages(volt_str):
    ''' display array of voltages on OLED
    left col.: 1-6
    right col.: 7-12'''
    oled.clear()
    i = 0
    for v in volt_str[0:6]:
        oled.text(v, 0, i*10)
        i+=1
    i = 0
    for v in volt_str[6:12]:
        oled.text(v, 64, i*10)
        i+=1
    oled.show()
#-------------------------------------------------------------------------------
def calculate_cellvoltages(voltages):
    ''' calculate voltages of individual cells
        where cells are connected in series and
        voltages is the array of voltages to ground'''
    diff =[]
    diff.append(voltages[0])
    for i in range(0, len(voltages)-1):
        d = voltages[i+1] - voltages[i]
        diff.append(d)
    return diff

def plausibility_check(cellvoltages, threshold = 0.05):
    '''sets difference voltage to 0 if negative
       or below threshold
       This is done to visualize non existing cells'''
    d = []
    for v in cellvoltages:
        if v < threshold:
            v = 0
        d.append(v)
    return d

def check_cell_voltages(voltages, vmax = V_MAX):
    too_high = 0
    too_high_cells = []
    i = 1
    for v in voltages:
        if v > vmax:
            too_high = 1
            too_high_cells.append(i)
        i+=1
    if too_high:
        charge_OK.value(0)
        overvoltage.value(1)
    else:
        charge_OK.value(1)
        overvoltage.value(0)
    return too_high, too_high_cells



def main():

    while True:
        voltages = multi_adc.read_all(factor = factor)
        volt_str = [ '%.3f' %v for v in voltages]

        cellvoltages = calculate_cellvoltages(voltages)
        cellvoltages = plausibility_check(cellvoltages)
        cell_v_str = [ '%.3f' %v for v in cellvoltages]
```

```
            # check
            too_high, th_cells = check_cell_voltages(cellvoltages)



            # print & display
            if d_switch.value():
                print_voltages(volt_str)
                display_voltages(volt_str)
            else:
                print_voltages(cell_v_str)
                display_voltages(cell_v_str)

            if too_high:
                print ("Voltage too high on cells: ", th_cells, end = '\t')
            print()
            time.sleep(1)

    main()
```

The voltages printed via serial are tabular data, with an indication when the cell voltage is too high:

```
4.100   4.186   4.093   4.103   4.160   4.068   4.126   4.091   4.126   4.174   4.034   4.149   Voltage too high on cells:  [2]
4.103   4.186   4.087   4.110   4.163   4.057   4.137   4.087   4.126   4.165   4.034   4.158   Voltage too high on cells:  [2]
4.103   4.183   4.087   4.112   4.160   4.061   4.133   4.089   4.123   4.174   4.029   4.153   Voltage too high on cells:  [2]
4.100   4.186   4.087   4.110   4.156   4.068   4.126   4.093   4.119   4.174   4.029   4.163   Voltage too high on cells:  [2]
4.107   4.172   4.093   4.105   4.165   4.068   4.121   4.091   4.123   4.172   4.029   4.160
4.126   4.197   4.114   4.133   4.195   4.093   4.160   4.119   4.146   4.202   4.043   4.197   Voltage too high on cells:  [2, 5, 10, 12]
```

There is a function plausibility_check that I added when testing with a dummy battery with less then 12 cells. It sets the displayed voltage to zero so the display is less confusing.

4 Charge control

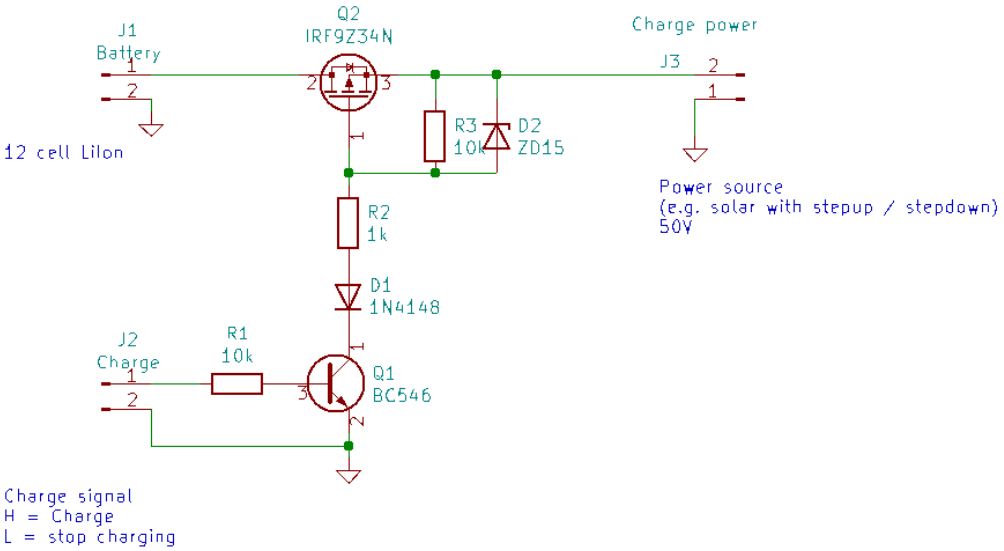
The next step is to use the check_cell_voltages function to get a signal switching the charge current on and off. This is an extract of bms_003.py:

```
...
charge_OK = Pin(16, Pin.OUT)
discharge_OK = Pin(17, Pin.OUT)
...

def check_cell_voltages(voltages, vmin = V_MIN,  vmax = V_MAX):
    # check high limit
    too_high = 0
    too_high_cells = []
    i = 1
    for v in voltages:
        if v > vmax:
            too_high = 1
            too_high_cells.append(i)
        i+=1
    if too_high:
        charge_OK.value(0)
        #overvoltage.value(1)
    else:
        charge_OK.value(1)
        #overvoltage.value(0)

    # check low
    ...

    return too_high, too_high_cells, too_low, too_low_cells
```

The check function also delivers a signal for discharge control.

The signal charge_OK now must be used to switch the charge current on and off.

This is done with a P-MOS transistor:



The zener diode protects the MOSFET from too high gate voltages, while the diode D1 protects Q1 from too high collector-emitter voltages.

5 Cell balancing

We have 12 cells, so we need 12 identical balancing circuits.

The schematic shows one of them, together with the I2C driver.
Two PCF8574 port expanders are used, so 16 outputs could be driven (we need 12).

The PCF8574 can only drive 1mA (could be OK, but is a bit on the limit), so we use it in Active Low logic. This way it can sink 20mA, that is more than we need.

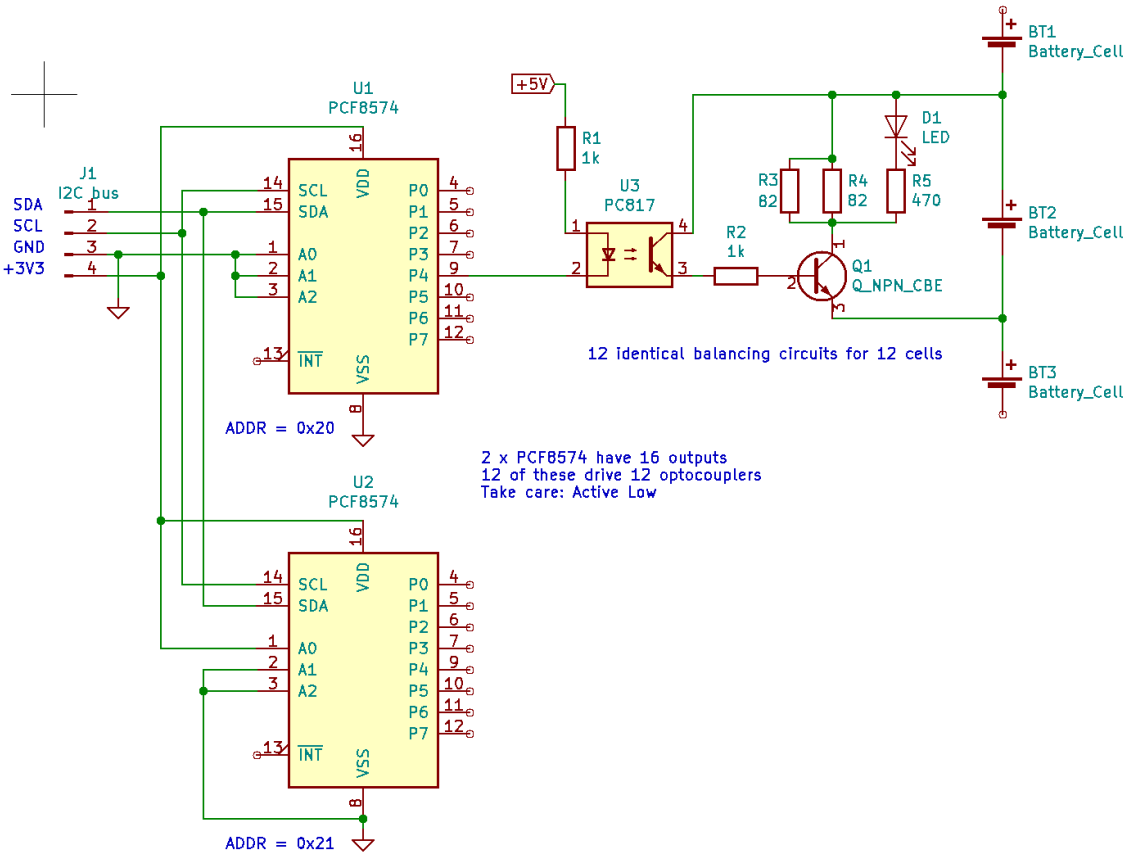
As each cell must have its own balancing circuit, we have no common ground for these, and we need optocouplers to isolate.

What current do we need for the balancing? A good question!
Here http://www.liionbms.com/php/wp_balance_current.php I found some rules of thumb:

> 10 mA is sufficient for small back-up supply applications (10 kWh),
> 100 mA for large applications (100 kWh)
>
> 100 mA is sufficient to handle any automotive application (10 kWh, plugged in nightly)
>
> 1 A is sufficient for large pack applications, other than back-up (> 100 kWh, cycled daily)

I opted for about 100mA that can easily be switched by a small signal transistor.

The schematic shows a cell balancing circuit for one cell, driven by an output of the PCF8574



So far until now: the 18.11.2021