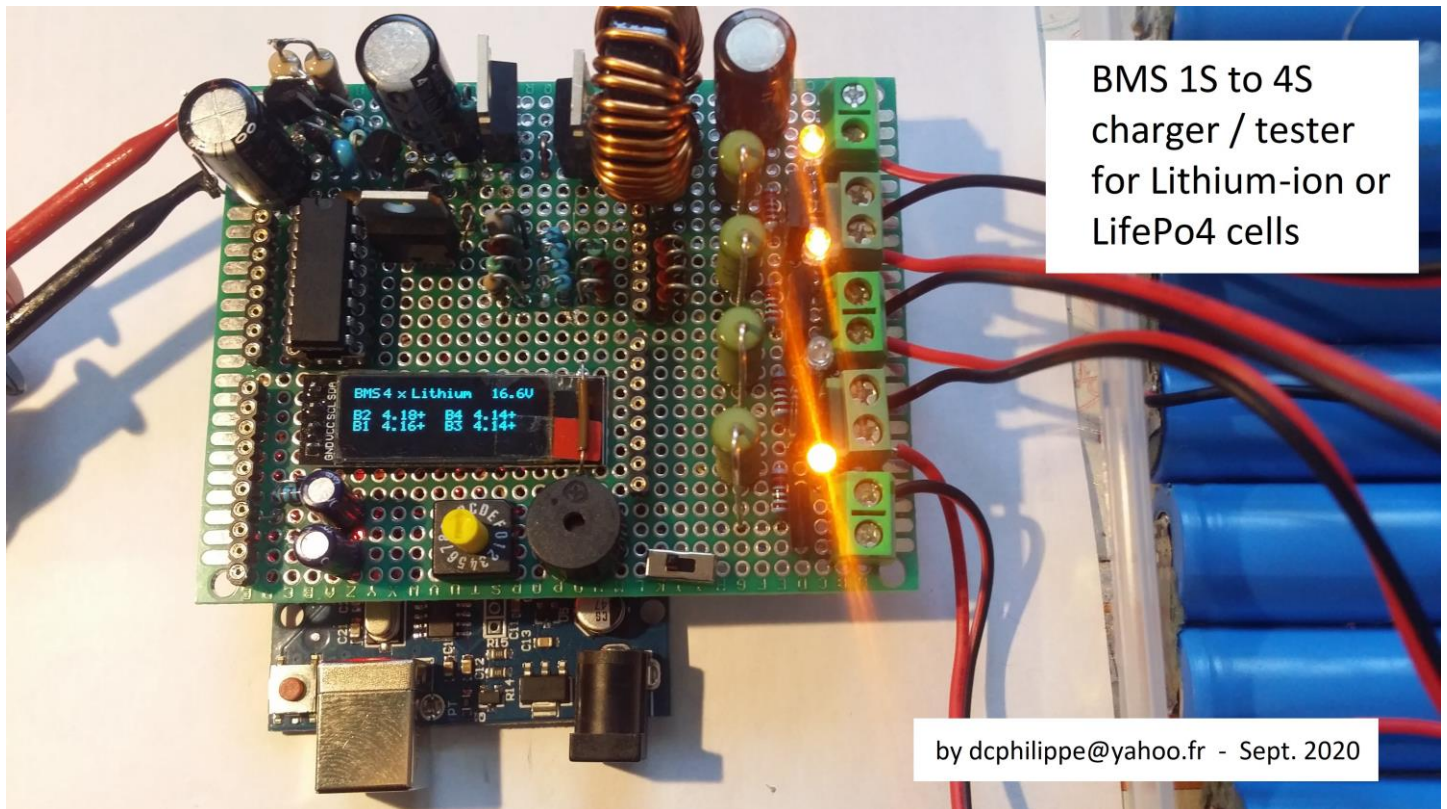


# How to build a BMS 1S to 4S charger / tester for Lithium-ion or LifePo4 cells



Author : Philippe de Craene  
[dcphilippe@yahoo.fr](mailto:dcphilippe@yahoo.fr)

Date: Sept 2020  
Manual version: 1.0  
Program version: 1.0

It can be strange to study a BMS in regard of the overall choice available on the market for quite a very low price. However, the objective of this project is to build a charging and testing device that can be universal, for a different among of cells to charge and for different technologies. The Arduino Uno based device, manage voltage readings, and according to results manages cells' shunt or a buzzer. A SSD screen displays where we are. A buck DC-DC converter adjust the charging voltage for a constant charging current.

Whatever if you have the habit or if you are starting in the modern batteries world – e-bikes batteries repair, solar powered systems, etc.... - you can notice that it is not easy to manage Li-ion nor LifePo4 cells:

- When you repair a battery pack, if it is easy to detect a dead cell, how do you detect a half-dead one?
- When you are building a new battery pack, BMS user manual requests that cells must be assembled once they are equilibrated. How do you perform such an assembling pack?
- With a usual BMS charger, you need a dedicated voltage value's power supply. It is not so easy to fit with...

This device may help you ☺

- It can charge simultaneously from 1 to 4 cells
- Cells can be either Lithium-ion technology – for instance 18650 – or LifePo4 cells (you cannot mix)
- Monitoring of any cells voltage, plus overall voltage
- A internal protection against short circuit also limit for a constant current of about 1 A – hardware setup –

- Any power supply DC 17V to 25V @ 1.5A can fit this device. An internal buck DC-DC converter adapt the needed voltage value. A old DC 19V laptop power supply is perfect
- The balancing is performed by shunts between cells
- Under voltage alarm by buzzer

This manual describes how to build such a BMS 1S...4S charger / tester.

Table of content

Introduction ..... 4

List of materials ..... 4

Hardware study ..... 5

    Cells monitoring ..... 5

    Cells shunt control..... 5

    Constant charging current & short circuit safety ..... 7

    DC-DC buck converter ..... 7

    All in one diagram circuit ..... 8

Software study ..... 9

    Cells voltage acquisition ..... 9

    Voltages limits management..... 9

    Power management ..... 10

    Parameters setup ..... 10

    The final code ..... 11

User manual ..... 18

Illustration in use..... 19

## Introduction

The use of Li-ion or Lifepo4 batteries needs some requirement:

- The discharge voltage must never be lower than a  $V_{min}$  (that is technology dependent - see below).
- The charging voltage must never be over a  $V_{max}$  (that is also technology dependent – see below).
- These batteries are very sensitive to any overcharging. A particular care must be done to stop the charging process as soon as the charging voltage  $V_{max}$  is reached.
- The charging current must be under a maximum value, that is usually under 1/10 the capacity current of the cell (0.1C)

The BMS available on the market are based on shunts that prevent against cells overcharging, but there is no monitoring to detect a nearly defective cell... Moreover if the  $V_{min}$  is reached during discharging, the power delivery is stopped. Once again, we do not know if a particular cell is lighter and should be replaced. In addition, it may be very critical situations where power is needed even if the battery life is in balance...

Li-ion and Lifepo4 voltage limit:

	$V_{min}$ = limit discharge voltage	$V_{max}$ = limit charging voltage
Lithium-ion	3.6 V	4.2 V
LifePo4	3.2 V	3.7 V

The help of the serial monitor can adjust these limit voltages, and then new values are kept in EEPROM.

It should be great to go over a 4S BMS, but the Atmega328p of the Arduino Uno has only 4 analog inputs to measure 4 cells voltages; it could be 6 analog inputs, but without the possibility of any data display anyway.

## List of materials

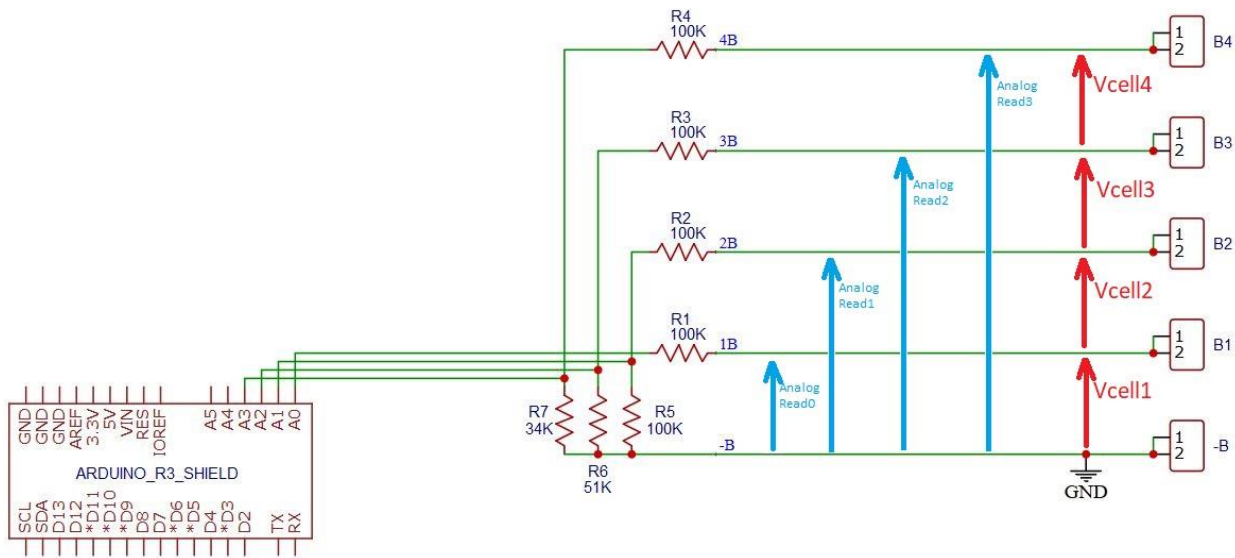
- 1- One Arduino Uno R3
- 2- One SSD1306 128x32
- 3- 4x NPN bipolar transistor BC635 or BD139 for shunts under 1A / 1.5A
- 4- 4x 10 $\Omega$  3W or 5 $\Omega$  5W resistors (see text)
- 5- One SN754410
- 6- One 0.68 $\Omega$  2W resistor
- 7- 2x low power NPN transistors like BC639 or 2N2222 or equivalent
- 8- 2x low power PNP transistors like BC640 or 2N2907 or equivalent
- 9- One 150uH / 2A inductor
- 10- One P-channel CMOS IRF9Z34N
- 11- One Schlocky rectifier diode 10A 30V like MBR30200 or equivalent
- 12- One 9V or 12V regulator 7809 or 7812
- 13- One binary rotary switch
- 14- One switch
- 15- One buzzer
- 16- 4x LEDs
- 17- 2x 1N4001 rectifier diode
- 18- One 1N4148 or 1N4001 diode
- 19- Resistors and capacitors, see text for values

## Hardware study

The hardware device is composed of four main parts:

- The cells monitoring that constantly measure the voltages between cells boards
- The shunt control that decrease the load current through the cells once the limit charging voltage is reached
- The load control e.g. the constant charging current with a safety against short circuit
- The DC-DC buck converter defines the power voltage according to a constant load current.

### Cells monitoring



Cells are connected between  $-B$  and  $B1$ ,  $B1$  and  $B2$ ,  $B2$  and  $B3$ ,  $B3$  and  $B4$ . The objective is to monitor voltage for each cell. However, the Arduino is only able to make any measure referenced to GND. What we can notice is that `AnalogRead0` directly gives  $V_{cell1}$ , and `AnalogRead3` gives the power supply voltage value. However it is possible to deduce  $V_{cell2} = \text{AnalogRead1} - \text{AnalogRead0}$ ,  $V_{cell3} = \text{AnalogRead2} - \text{AnalogRead1}$ , and so on.

The second point is any `AnalogRead` must NOT go over 5V otherwise the Atmega328p will very quickly smoke...

Hence the presence of voltage dividers around  $R1$  to  $R7$ : `AnalogRead0` is directly read (with  $R1$  as a protection), `AnalogRead1` is divided by 2 ( $R2$  and  $R5$ ), `AnalogRead2` is divided by 3, etc.... each time the result fit a voltage value under 5V.

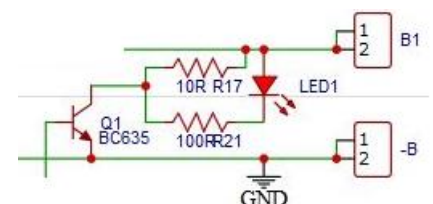
I know.... The grumpy ones will say that with voltage divisors the reading precision will lose too much. OK so with an analog to digital converter of 10 bits, the analog from 0 to 5V signal is converted to 1024 bits. So, a one bit precision is 4.8mV. The worst divider ration is 4, then the precision "falls" to 19.2mV.... 0.0192V. My opinion is it is good enough for monitoring that a LifePo4 cell does not go over 3.7V, or even 3.65V. We only need a reading precision of 0.05V

### Cells shunt control

The shunt consists of switching a power resistor to the cell, so that a part of the charging current circulates in it, according to the rule  $I = V/R$ , with a Li-ion  $V_{max} = 4.2V$  and  $R = 10\ \Omega$ , the current is 0.42A.

So, R-value must be set according to the part of the current to be drifted:

- Too small the cell will discharge
- Too high its use is not effective



Here with  $R = 10\ \Omega$  and a charging current of 1A only approximately half of total current is drifted. With  $R = 5\ \Omega$  it would drifted 0.82A. It seems better, and in fact it is with the use of LifePo4 cells: with a LifePo4  $V_{max} = 3.7V$  and  $R = 5\ \Omega$ , the current is 0.74A.

A care about this resistor heat dissipation:  $P = V^2/R$ , with a Li-ion  $V_{max} = 4.2V$  and  $R = 10\ \Omega$ ,  $P = 1.7W$ , 3.5W in case of  $R = 5\ \Omega$ . Do not use single 1/4W resistors!

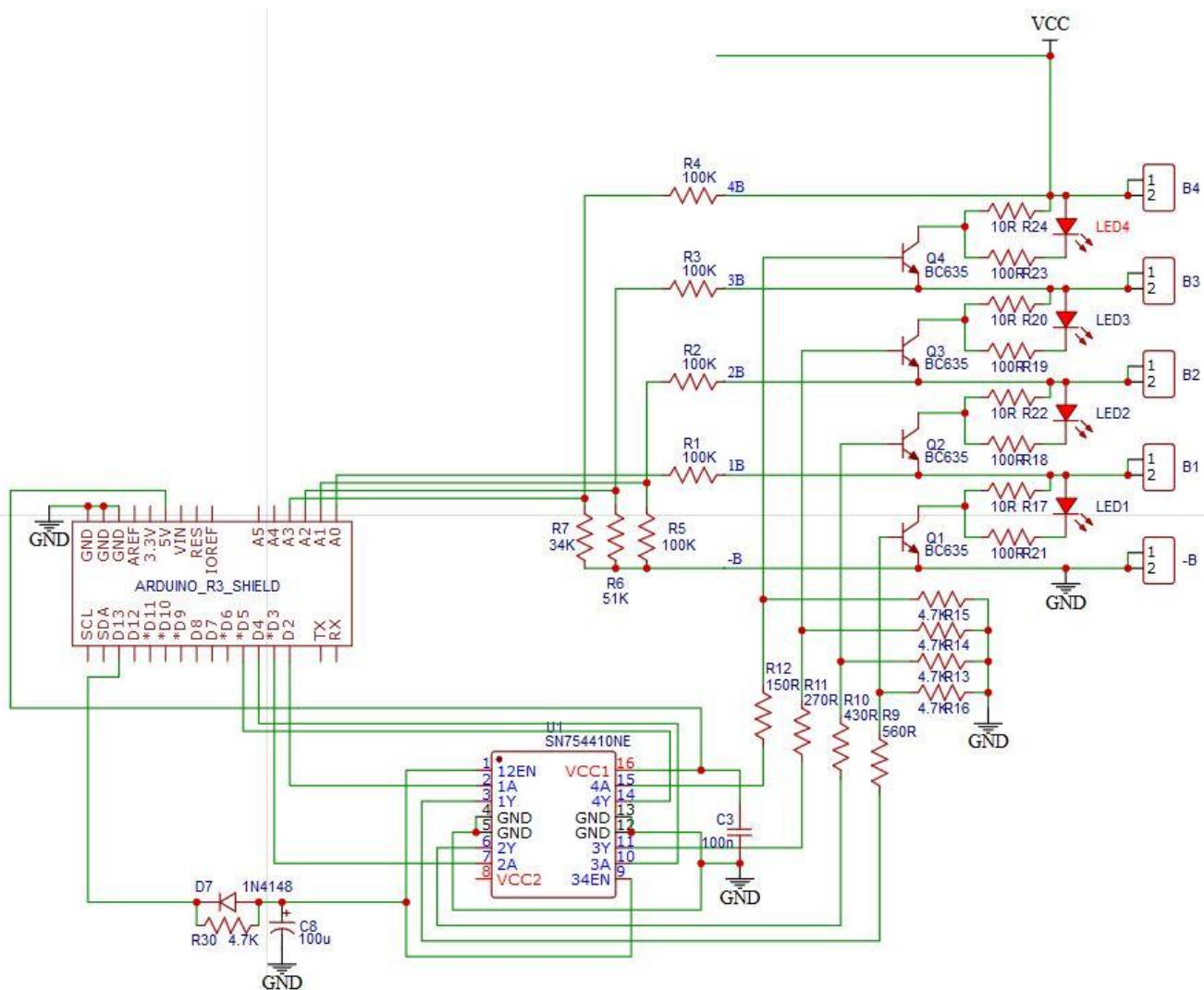
The LED shows whenever the shunt is active.

A question for the grumpy ones... Why am I using bipolar transistors instead of CMOS? Because bipolar transistors are driven in current, in opposite CMOS are driven with voltage. At least 10V. If it is not a problem for the shunt of B1 nor B2, the GND reference for B3 is 8.4V and 12.6V for B4. It is supposed to have somewhere at least 22.6V to drive it. With a 19V laptop power supply, it is not easy to get.

The dark side by using bipolar transistor is that they may need a lot of current. Power bipolar transistors'  $H_{fe}$  is about 40. That means if collector current is 1A, the  $I_b > 25mA$ . Less the transistor will not act as a switch. That explains the rule of R9 to R12 that limit the base current just enough, depending of the GND reference of each of them. They form with R13 to R16 the voltage divisor and block transistor (= open switch) in case of no current. R9 to R12 should be 1/2W resistors.

Finally, a simple way to be able to give some current at any voltage is the use of a SN754410 buffer.

The SN754410 offers the advantage to ENABLE or DISABLE the buffer follower with pins 1 and 9. R30 and C8 make a time delay before enabling the IC, therefore D7 disable instantly.





## Constant charging current & short circuit safety

It would be nice to insert a current sensor like the ACS712 module, but unfortunately, an analog entry would miss with the Arduino Uno. However, I do not expect to use a bigger MPU for just driving a 4S BMS....

However, there are some digital inputs free; the solution is to build a logical current sensor.

The load current passes through R8. A voltage appears at the resistor terminals according to  $V_{r8} = R8 \times I$ .

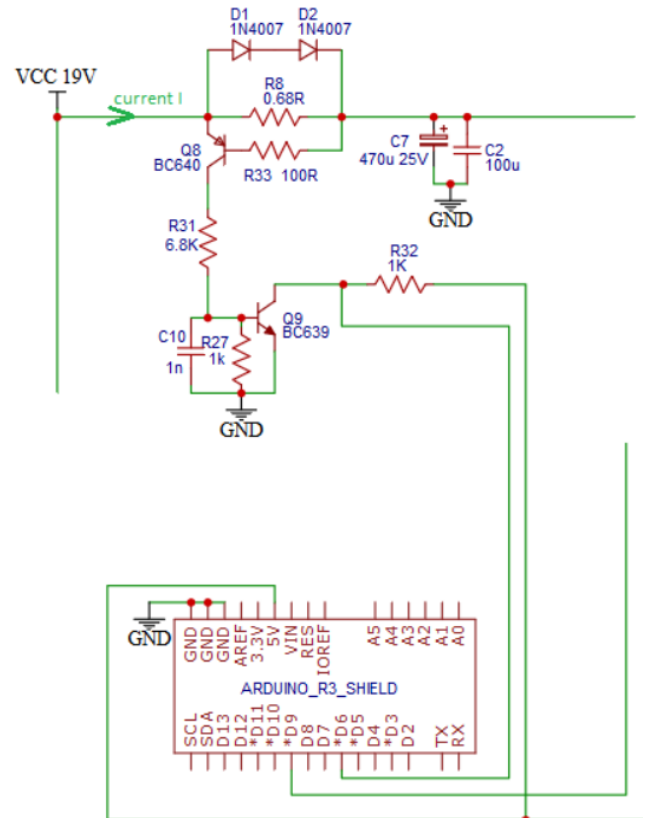
You can notice that R8 is connected in parallel with the Base and the Emitter of a bipolar transistor Q8: As long as  $V_{r8}$  stays below the  $V_{be}$  of saturation (something around 0.7V), Q8 is blocked with no Collector current. However as soon as this voltage increases to  $V_{be}$  of saturation Q8 starts to conduct and a collector current appears, this current makes a voltage through R31 and R27. Then Q9 enters in saturation, the digital input D6 that were at logical level 1 (=5V) falls down to logical level 0 (=0V).

D1 + D2 safe Q8 in case of short circuit with  $V_{be}$  below 1.4V.

R33 safes Q8 by limiting the  $I_b$  current

The level of current  $I$  that switch Q8 and Q9 is defined by R8:

Current  $I \approx 0.7 / R8$ , so with  $R8 = 0.68 \Omega$ , current  $I \approx 1 \text{ A}$

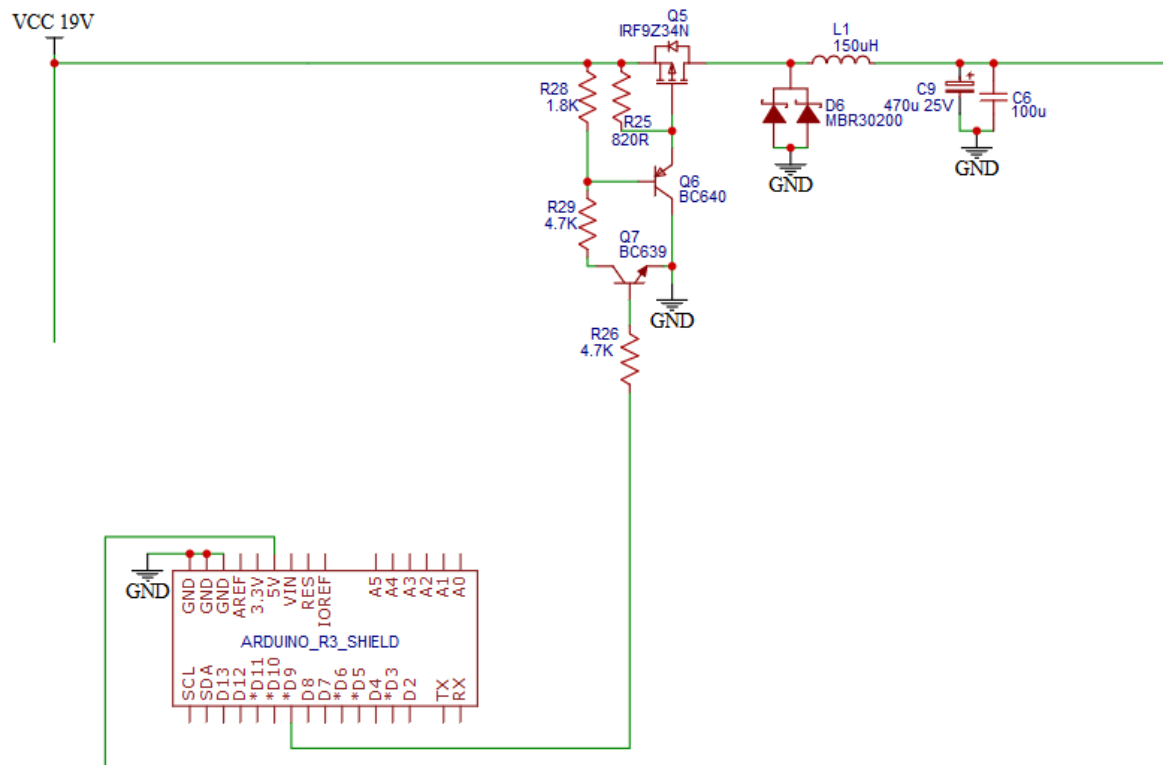


## DC-DC buck converter

One of the interests of this project is its capability to use either Li-ion or LifePo4 cells. However, we have yet seen in the introduction chapter that their voltages characteristics are different and incompatible. When addressing four cells at the same time the power supply must be 16.8V for Li-ion, 14.8V for LifePo4.

Moreover, this power supply must be able to decrease depending of the current through the load, to be able to obtain a constant load current. Of course exceed voltage limit must be defined.

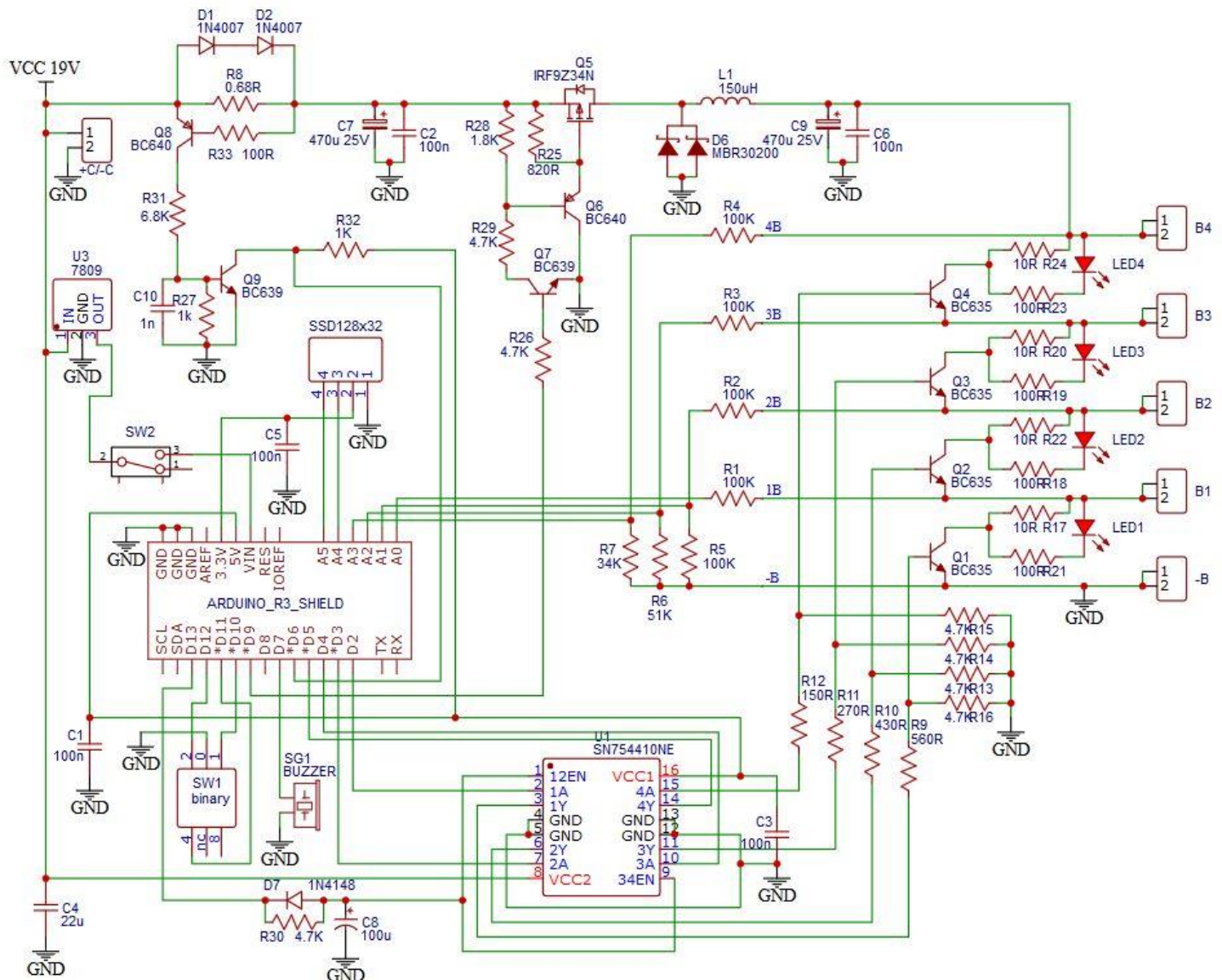
The buck converter is a classic: the 30KHz PWM signal comes from digital output 9 of the Arduino Uno. Q7 makes the supply



adaptation from the 5V to the power supply (around 19V). When D9 is “high” Q7 enters in saturation, the current in Collector makes a voltage in R28. Then Q6 enters in saturation, the voltage at the Gate of Q5 passes from power supply to 0V.

The P-channel CMOS is very less efficient than a N-channel. But in cutting configuration the Source pin must at the most negative reference for a N-channel CMOS, and in this case the most negative point would be the power which load the cells. It is not very good. Moreover, the Gate signal must be at least 10V,  $10V + 14.8V = 24.8V$ , we have yet seen such a problem with the shunt for B4... That is why we choose a P-channel.

### All in one diagram circuit



SW1 allows selecting between 1 to 4 Li-ion or lifePo4 cells, 8 positions / 3 bits required.

SW2 allows switching on or off the device,

U3 is a 9V or 12V regulator, as it is now safe to power the Arduino Uno under more than 15V.

SG1 is a buzzer to alert under voltage cells

Cells shunt resistors should better be 5Ω instead of 10Ω .



## Software study

Like the 4 parts that make up the hardware, the software is itself divided into 3 main parts:

- Cells voltages acquisition
- Cells voltages limits management: check if the shunt or the buzzer must be set on or off
- Power management : automatically set the power voltage
- Parameters setup

### Cells voltage acquisition

Every loop a series of `analogRead` get the analog values from GND for each cell. First of a better accuracy there is a series of 10 `analogRead`, then the working values is an average of them. Note a `delay(1)` after each `analogRead` to let the time to the internal ADC to translate the analog value to bytes.

Another interesting point is `vcalibration[i]`. The usual resistors have a tolerance value of 5%. It is too much. Each final value has its `vcalibration` to compensate these 5% of error. By default `vcalibration = 1.0` (=100% of value), but it is possible to adjust it in parameters setup with the serial monitor.

Here is this part code:

```
// this is run every loop cycle
//-----
// read the analog values
for( byte i=0; i<4; i++ ) {
    int VcellByte = analogRead( VcellPin[i] );
    delay(1);
    VcellCumul[i] += VcellByte;
}
Imax = !digitalRead( ImaxPin );           // Imax reached when IMax 'true'

// the following is done after every analogReadsCount cycles of reading analog values
//-----
if( ++analogReadsCount > 100 ) return;

// calculate voltages
for( byte i=0; i<4; i++ ) {
    Vcell[i] = (VcellCumul[i] / (float)analogReadsCount / 1023.0) * 5.0 * vcalibration[i] * (i+1);
    VcellCumul[i] = 0;
}
analogReadsCount = 0;
Vlim = Vcell[3];
for( byte i=3; i>0; i-- ) {
    Vcell[i] -= Vcell[i-1]; // get the voltage for each cell
}
// end of for
```

### Voltages limits management

Every seconds the cells voltage are compared to limit values:

```
// check balancing and undervoltage
for( byte i=0; i<4; i++ ) {
    if( Vcell[i] > Vmax[cellModel] ) digitalWrite( shuntPin[i], HIGH ); // check balancing
    else if( oneMinute && ( Vcell[i] < Vmin[cellModel] ) ) buzzerON = true; // check undervoltage
}
// end of for
if( buzzerON ) tone( buzzerPin, 440, 50 );           // one short tone every seconds
```

- Under *Vmin* the buzzer flag is set on.
- Over *Vmax* the shunt is set on.

Note that there is no way to stop either the buzzer or the shunt. If such a reset task has to be done, either you press the Arduino Uno reset push-button, either you wait for one minute the code to initiate the runOnce procedure that reset all flags:

```
// this is run only one time or after any config change or every 5 minutes
//-----
if( runOnce ) {
    static byte counter = 0;
    static unsigned long memo_tempo_shuntON = 0;
    buzzerON = false;
    for( byte i=cellNumber; i<4; i++ ) digitalWrite( shuntPin[i], HIGH ); // stop the buzzer if any
    if( counter > 5 ) {
        counter = 0;
        runOnce = false;
    }
    else if( tempo - memo_tempo_shuntON > 300 ) {
        memo_tempo_shuntON = tempo;
        counter++;
        for( byte i=0; i<cellNumber; i++ ) digitalWrite( shuntPin[i], HIGH ); delay(1);
        for( byte i=0; i<cellNumber; i++ ) digitalWrite( shuntPin[i], LOW ); // active cells reset
    }
}
shunt
} // end of test runOnce
```

Every runOnce routine will quickly blink the 4 LEDS 5 times

## Power management

```
// this is run every loop cycle
//-----
// read the analog values
for( byte i=0; i<4; i++ ) {
    int vcellByte = analogRead( vcellPin[i] );
    delay(1);
    vcellCumul[i] += vcellByte;
}
Imax = !digitalRead( ImaxPin ); // Imax reached when IMax 'true'

// define the DC-DC buck converter pwm ratio
// choose and comment the test with or without power voltage safety
//if( Imax || ( valim >= (4*vmax[cellModel])+0.5) ) { // test with power voltage safety
if( Imax || ( vcell[0] < 4.9) ) { // take care B1 does not exceed 5V !
    if( pwm > 0 ) pwm--;
}
else if( pwm < 255 ) pwm++;

analogWrite( pwmPin, pwm );
```

First of all every loop the load current status is read: the *Imax* flag is true if the current reaches the target value (remember, it is defined by R8).

Note: two ways for power voltage safety are available: either the power voltage cannot exceed the total of cells maximum voltage, either there is only a protection for analog input A0 not exceed 5V.

Then, depending of the *Imax* flag the PWM signal increases or decreases. The load voltage is constrained not to grow up to an excessive value.

The PWM signal frequency is set in the setup with:

```
// set Timer1 pin9 and pin10
TCCR1B = TCCR1B & B11111000 | B00000001; // set Timer1 divisor to 1 for PWM frequency of
31372.55 Hz
```

## Parameters setup

The `serialEvent()` function is a specific built-in function that wait after any serial input. It is very useful and easy to implement for parameters adjustment. The code treats of characters type input instead of strings, for a safe memory use.

```
//
// serialEvent() : Arduino builtin function for any console input
//
void serialEvent() {
  if( Serial.available() ) {
    char incomingChar = Serial.read();      // no timeout nor delay unlike Serial.readBytesUntil()
    if( incomingChar != '\n' ) {
      consoleInput[index] = incomingChar;
      index++;
    }
    else {
      consoleInput[index] = '\0';           // null character
      index = 0;

      int val, cell;
      switch(consoleInput[0]) {
        case 'b':
        case 'B': whatToDisplay = 'B';
                  etc.
                  break;

        case 'l':
        case 'L': whatToDisplay = 'L';
                  etc.
                  break;

        case 'h':
        case 'H': whatToDisplay = 'H';
                  etc.
                  break;

        case 'e':
        case 'E': whatToDisplay = 'A';
                  Serial.println(F("cancel all changes!"));
                  EEPROM_Get();
                  break;

        case 's':
        case 'S': whatToDisplay = 'A';
                  Serial.println(F("all data saved!"));
                  EEPROM_Update();
                  break;
      }
    }
  }
}
```

The code first looks at the first character (lower or upper case) of any console keyword input, and it can start with:

- “B” : for *battery cell*. The case instruction will treat of cell voltages calibrations (remember the 5% of resistor tolerance we need to correct)
- “H” : for *high limit voltage*. The case instruction will treat the  $V_{max}$  new value
- “L” : for *low limit voltage*. The  $V_{min}$  new value
- “S” : for *save*. New data is recorded and stored in EEPROM for any future operation.
- “E” : for *exit*. New data is lost.

## The final code

```
/*
BMS Li-ion/LiFePo4 automatic charger 1S 2S 3S 4S

author : Philippe de Craene <dcphilippe@yahoo.fr>
Free of use - Any feedback is welcome
*/
```

The charger works in 2 parts:

- First the BMS part from the idea of <https://simple-ee.com/2019/07/20/arduino-4s-bms-version-7/>
- Then the charger part with a buck converter

Calibration of measured voltages are done with the console. Connect the Arduino Uno to the usb of the PC.

The model (Li-ion or LiFePo4) of cell and the number (1 to 4) to be charged are set with the binary rotary button.

When the yellow LED assigned to each cell is continually lighted, the cell is charged (the shunt is active).

#### Arduino Uno pinout

```
-----
A0 ==> B1 input
A1 ==> B2 input
A2 ==> B3 input
A3 ==> B4 input
A4 ==> SSD1306 display SDA
A5 ==> SSD1306 display SCL
 2 ==> B1 shunt output - pin2 of 1A / SN754410
 3 ==> B2 shunt output - pin7 of 2A / SN754410
 4 ==> B3 shunt output - pin10 of 3A / SN754410
 5 ==> B4 shunt output - pin15 of 4A / SN754410
 6 ==> input from Imax charging sensor
 7 ==> buzzer
 9 ==> PWM output for buck converter
10 ==> x1 BCD rotary contactor
11 ==> x10 BCD rotary contactor
12 ==> x100 BCD rotary contactor
13 ==> ENABLE output for SN754410
```

#### Versions history

```
-----
version 0.1 - 21 august 2020 - first operational version
version 0.3 - 1 sept 2020    - add the limit charging current
version 1.0 - 16 sept 2020   - add the buck converter
```

#### Remarks

```
-----
About Serial.print(F("bla bla")) usage see https://www.baldengineer.com/arduino-f-macro.html
RAM usage decrease from 81% to 34% inside this code
*/
```

```
#include <EEPROM.h>
#include "ssd1306.h"      // https://github.com/lexus2k/ssd1306

// Parameters
//-----

const bool FIRST_USE = false; // must be set "true" the very first use to record parameters in
EEPROM
bool cellModel = LOW;         // LOW = Li-ion / HIGH = LifePo4
byte cellNumber = 4;          // number of cells to charge
float vmax[2] = { 4.2, 3.7 }; // maximum voltage for Li-ion / LifePo4 cells
float vmin[2] = { 3.6, 3.2 }; // minimum volatge for Li-ion / LifePo4 cells
float vcalibration[4] = { 1.0, 1.0, 1.0, 1.0 }; // calibration to fit real voltage measures

// Hardware connexion
//-----

byte vcellPin[4] = { 0, 1, 2, 3 }; // analog read of each cell : A0:B1, A1:B2, A2:B3, A3:B4
byte shuntPin[4] = { 2, 3, 4, 5 }; // output to shunt for each cell
const byte ImaxPin = 6;           // digital input from Imax charging sensor/detector
const byte buzzerPin = 7;         // alarm undervoltage cell
const byte pwmPin = 9;            // pwm for buck converter
const byte UrcPin = 10;           // x1 BCD rotary contactor
const byte DrcPin = 11;           // x10 BCD rotary contactor
const byte CrcPin = 12;           // x100 BCD rotary contactor
const byte enablePin = 13;        // SN754410 ENABLE

// Global variables
//-----

float valim = 0;                  // power supply voltage
float vcell[4] = { 4.9, 4.9, 4.9, 4.9 }; // cells measured voltage
float memo_vcell[4] = { 0.0, 0.0, 0.0, 0.0 }; // past cells voltage
bool runOnce = true;             // run one time only flag
byte pwm = 0;                    // buck converter pwm
bool parametersMenu = false;     // flag when parameters are set by the console inputs
byte index = 0;                  // input character counter
char consoleInput[7];            // console input
char whatToDisplay = 'A';        // select console messages to display, 'A' for all
bool Imax = false;               // flag for charging max current
bool memo_Urc, memo_Drc, memo_Crc;

//
// setup
//-----

void setup() {
// define inputs & outputs
```

```

pinMode( enablePin, OUTPUT ); digitalWrite( enablePin, LOW );
for( byte i=0; i<4; i++ ) {
    pinMode( shuntPin[i], OUTPUT ); digitalWrite( shuntPin[i], LOW );
}
pinMode( pwmPin, OUTPUT ); analogWrite( pwmPin, 0 );
pinMode( UrcPin, INPUT_PULLUP );
pinMode( DrcPin, INPUT_PULLUP );
pinMode( CrcPin, INPUT_PULLUP );
pinMode( ImaxPin, INPUT_PULLUP );
pinMode( buzzerPin, OUTPUT );

// Initialise the oled display & console
Serial.begin(250000);
Serial.println(F("Starting..."));

ssd1306_128x32_i2c_init();
//ssd1306_128x64_i2c_init();
ssd1306_fillScreen(0x00);
ssd1306_setFixedFont(ssd1306x1led_font6x8);
ssd1306_clearScreen();

// EEPROM check and data upload :
// stored data are always positive from 0 to 255.
// it seems that in cas of first use all are set to 255.
if( FIRST_USE ) EEPROM_Update();
else EEPROM_Get();

// set Timer1 pin9 and pin10
TCCR1B = TCCR1B & B11111000 | B00000001; // set Timer1 divisor to 1 for PWM frequency of
31372.55 Hz

// last tasks....
RotactorConfig(); // check configuration from binary rotary contactor
digitalWrite( enablePin, HIGH ); // enable SN754410
} // end of setup

//
// loop
//


---


void loop() {

    static float vcellCumul[4] = { 0.0, 0.0, 0.0, 0.0 }; // cumulative analogRead in bytes
    static unsigned int analogReadsCount = 0; // number of analogRead counter
    static bool buzzerON = false;
    static bool oneMinute = false;

    unsigned long tempo = millis();

    // this is run only one time or after any config change or every 5 minutes
    //-----
    if( runOnce ) {
        static byte counter = 0;
        static unsigned long memo_tempo_shuntON = 0;
        buzzerON = false; // stop the buzzer if any
        for( byte i=cellNumber; i<4; i++ ) digitalWrite( shuntPin[i], HIGH ); // shunt inactive cells
        if( counter > 5 ) {
            counter = 0;
            runOnce = false;
        }
        else if( tempo - memo_tempo_shuntON > 300 ) {
            memo_tempo_shuntON = tempo;
            counter++;
            for( byte i=0; i<cellNumber; i++ ) digitalWrite( shuntPin[i], HIGH ); delay(1);
            for( byte i=0; i<cellNumber; i++ ) digitalWrite( shuntPin[i], LOW ); // active cells reset
        }
    }
    shunt
} // end of test runOnce

// this is run every loop cycle
//-----
// read the analog values
for( byte i=0; i<4; i++ ) {
    int vcellByte = analogRead( vcellPin[i] );
    delay(1);
    vcellCumul[i] += vcellByte;
}
Imax = !digitalRead( ImaxPin ); // Imax reached when IMax 'true'

// define the DC-DC buck converter pwm ratio
// choose and comment the test with or without power voltage safety
//if( Imax || ( valim >= (4*vmax[cellModel])+0.5) ) { // test with power voltage safety
if( Imax || ( vcell[0] < 4.9) ) { // take care B1 does not exceed 5V !
    if( pwm > 0 ) pwm--;
}
else if( pwm < 255 ) pwm++;
}

```



```

    analogWrite( pwmPin, pwm );

// the following is done after every analogReadsCount cycles of reading analog values
//-----
    if( ++analogReadsCount > 100 ) return;

// calculate voltages
    for( byte i=0; i<4 ; i++ ) {
        vcell[i] = (vcellCumul[i] / (float)analogReadsCount / 1023.0) * 5.0 * vcalibration[i] * (i+1);
        vcellCumul[i] = 0;
    }
    analogReadsCount = 0;
    valim = vcell[3];
    for( byte i=3; i>0; i-- ) {
        vcell[i] -= vcell[i-1]; // get the voltage for each cell
    } // end of for

// this is done every second: read buttons and display data
//-----
    static unsigned long memo_tempo = 0;
    if( tempo - memo_tempo < 1000 ) return;
    memo_tempo = tempo;

// check balancing and undervoltage
    for( byte i=0; i<4; i++ ) {
        if( vcell[i] > Vmax[cellModel] ) digitalWrite( shuntPin[i], HIGH ); // check balancing
        else if( oneMinute && ( vcell[i] < Vmin[cellModel] ) ) buzzerON = true; // check undervoltage
    } // end of for
    if( buzzerON ) tone( buzzerPin, 440, 50 ); // one short tone every seconds

// check configuration from binary rotary contactor
    RotactorConfig();

// console display
    ConsoleDisplay( whatToDisplay );

// oled display
    oledDisplay();

// this is done every 60 seconds: voltage tendencies
//-----
    static unsigned long memo_tempo_tendancy = 0;

    if( tempo - memo_tempo_tendancy < 60000 ) return;
    memo_tempo_tendancy = tempo;
    for( byte i=0; i<4; i++ ) memo_vcell[i] = vcell[i]; // remember past cells voltages for
tendencies
    oneMinute = true;
    runOnce = true;

} // end of loop

//=====
// list of functions
//=====

// check configuration from binary rotary contactor
//-----

void RotactorConfig() {

    bool Urc = digitalRead( UrcPin );
    bool Drc = digitalRead( DrcPin );
    bool Crc = digitalRead( CrcPin );
    if( Crc ) cellModel = HIGH; // LifePo4 model cell
    else cellModel = LOW; // Li-ion model cell
    if( Urc ) {
        if( Drc ) cellNumber = 4;
        else cellNumber = 2;
    }
    else {
        if( Drc ) cellNumber = 3;
        else cellNumber = 1;
    }
    if((Urc != memo_Urc) || (Drc != memo_Drc) || (Crc != memo_Crc)) runOnce = true;
    memo_Urc = Urc;
    memo_Drc = Drc;
    memo_Crc = Crc;
} // end of RotactorConfig()

//
// TendancySet() : set the tendancy of cells voltage
//-----

char TendancySet( byte i ) {

```

```

    if( vcell[i] - memo_vcell[i] > 0.0 ) return '+';
    else if((vcell[i] - memo_vcell[i]) == 0 ) return '=';
    else return '-';
    return '?';
} // end of TendancySet()

//
// EEPROM_Get() : read values stored in the EEPROM
//
void EEPROM_Get() {
    for( byte i=0; i<4; i++ ) {
        int var = (EEPROM.read(2*i) << 8) + EEPROM.read((2*i)+1);
        vcalibration[i] = var/1000.0;
    }
    vmax[0] = (EEPROM.read(8)+300)/100.0;
    vmax[1] = (EEPROM.read(9)+300)/100.0;
    vmin[0] = (EEPROM.read(10)+300)/100.0;
    vmin[1] = (EEPROM.read(11)+300)/100.0;
} // end of EEPROM_Get()

//
// EEPROM_Update() : update values stored in the EEPROM
//
void EEPROM_Update() {
    for( byte i=0; i<4; i++ ) {
        int var = 1000*vcalibration[i];
        EEPROM.update((2*i), highByte(var));
        EEPROM.update((2*i)+1, lowByte(var));
    }
    EEPROM.update(8, ((vmax[0]*100.0)-300));
    EEPROM.update(9, ((vmax[1]*100.0)-300));
    EEPROM.update(10, ((vmin[0]*100.0)-300));
    EEPROM.update(11, ((vmin[1]*100.0)-300));
} // end of EEPROM_Update()

//
// OledDisplay() : display to oled
//
void OledDisplay() {
    char flt2str[6];

    //first line
    ssd1306_printFixed ( 0, 0, "BMS", STYLE_NORMAL);
    dtostrf( cellNumber, 1, 0, flt2str ); // usage : ( number_value, number_of_digits,
nubler_of_decimal, char_output)
    ssd1306_printFixed (20, 0, flt2str, STYLE_NORMAL);
    ssd1306_printFixed (30, 0, "x", STYLE_NORMAL);
    if( cellModel ) ssd1306_printFixed (40, 0, "LifePo4", STYLE_NORMAL);
    else ssd1306_printFixed (40, 0, "Li-ion", STYLE_NORMAL);

    dtostrf( valim, 4, 1, flt2str );
    ssd1306_printFixed (96, 0, flt2str, STYLE_NORMAL);
    ssd1306_printFixed (120, 0, "v", STYLE_NORMAL);

    //cells value: B2, B1, B4, B3
    if( vcell[1] < 0 ) dtostrf( vcell[1], 5, 2, flt2str );
    else dtostrf( vcell[1], 5, 3, flt2str );
    flt2str[4] = TendancySet(1);
    ssd1306_printFixed (0, 16, "B2 ", STYLE_NORMAL);
    ssd1306_printFixed (20, 16, flt2str, STYLE_NORMAL);

    if( vcell[0] < 0 ) dtostrf( vcell[0], 5, 2, flt2str );
    else dtostrf( vcell[0], 5, 3, flt2str );
    flt2str[4] = TendancySet(0);
    ssd1306_printFixed (0, 24, "B1 ", STYLE_NORMAL);
    ssd1306_printFixed (20, 24, flt2str, STYLE_NORMAL);

    if( vcell[3] < 0 ) dtostrf( vcell[3], 5, 2, flt2str );
    else dtostrf( vcell[3], 5, 3, flt2str );
    flt2str[4] = TendancySet(3);
    ssd1306_printFixed (64, 16, "B4 ", STYLE_NORMAL);
    ssd1306_printFixed (84, 16, flt2str, STYLE_NORMAL);

    if( vcell[2] < 0 ) dtostrf( vcell[2], 5, 2, flt2str );
    else dtostrf( vcell[2], 5, 3, flt2str );
    flt2str[4] = TendancySet(2);
    ssd1306_printFixed (64, 24, "B3 ", STYLE_NORMAL);
    ssd1306_printFixed (84, 24, flt2str, STYLE_NORMAL);
} // end of OledDisplay()

//

```

```
// ConsoleDisplay() : console displays
//
```

---

```
void ConsoleDisplay( char what ) {
```

```
    Serial.println(F("\nBMS charger/tester general menu, type the command according to desired
action[:value], then ENTER\n"));
    if( what == 'B' || what == 'A' ) {
        Serial.println(F("Bxaaaa with x=1..4 for cell number x with a.aaa the new voltage calibration
value"));
        Serial.print(F(" Cal1= ")); Serial.print( vcalibration[0],3 ); Serial.print(F(" "));
        Serial.print(F(" Cal2= ")); Serial.print( vcalibration[1],3 ); Serial.print(F(" "));
        Serial.print(F(" Cal3= ")); Serial.print( vcalibration[2],3 ); Serial.print(F(" "));
        Serial.print(F(" Cal4= ")); Serial.print( vcalibration[3],3 ); Serial.println();
        Serial.print(F(" B1= ")); Serial.print( vcell[0],2 ); Serial.print(F(" "));
        Serial.print(F(" B2= ")); Serial.print( vcell[1],2 ); Serial.print(F(" "));
        Serial.print(F(" B3= ")); Serial.print( vcell[2],2 ); Serial.print(F(" "));
        Serial.print(F(" B4= ")); Serial.print( vcell[3],2 ); Serial.println("\n");
    }
    if( what == 'H' || what == 'A' ) {
        Serial.println(F("HTaaa for Li-ion to set the cell maximum voltage to a.aa"));
        Serial.println(F("HFaaa for LifePo4 to set the cell maximum voltage to a.aa"));
        Serial.print(F(" Li-ion Vmax = ")); Serial.print( Vmax[0]);
        Serial.print(F(" LifePo4 Vmax = ")); Serial.println( Vmax[1]);
        Serial.println();
    }
    if( what == 'L' || what == 'A' ) {
        Serial.println(F("LTaaa for Li-ion to set the cell minimum voltage to a.aa"));
        Serial.println(F("LFaaa for LifePo4 to set the cell minimum voltage to a.aa"));
        Serial.print(F(" Li-ion Vmin = ")); Serial.print( Vmin[0]);
        Serial.print(F(" LifePo4 Vmin = ")); Serial.println( Vmin[1]);
        Serial.println();
    }
    Serial.print(F(" Actual : "));
    if( cellModel ) Serial.print(F("LifePo4"));
    else Serial.print(F("Li-ion"));
    Serial.print(F(" pwm = ")); Serial.print(pwm); Serial.print(F(" power supply = "));
    Serial.print(Valim); Serial.print(F(" / ")); Serial.print(4*Vmax[cellModel]);
    Serial.println(F(" maxi\n"));

    Serial.println(F("S to save data to EEPROM"));
    Serial.println(F("E to recover last saved data\n\n\n"));
}
```

```
    // end of ConsoleDisplay()
```

```
//
// serialEvent() : Arduino builtin function for any console input
//
```

---

```
void serialEvent() {
```

```
    if( Serial.available() ) {
        char incomingChar = Serial.read(); // no timeout nor delay unlike Serial.readBytesUntil()
        if( incomingChar != '\n' ) {
            consoleInput[index] = incomingChar;
            index++;
        }
        else {
            consoleInput[index] = '\0'; // null character
            index = 0;

            int val, cell;
            switch(consoleInput[0]) {
                case 'b':
                case 'B': whatToDisplay = 'B';
                    val = (consoleInput[5]-48)+(10*(consoleInput[4]-48))+(100*(consoleInput[3]-48))+(1000*(consoleInput[2]-48));
                    cell = consoleInput[1] -49; // -48 (because ASCII) -1 (because 1..4 becomes 0..3)
                    if( cell > cellNumber ) Serial.println(F("cell number out of range"));
                    else if( val < 800 || val > 1200 ) Serial.println(F("Calibration value out of
range"));
                    else vcalibration[cell] = val/1000.0;
                    break;
                case 'l':
                case 'L': whatToDisplay = 'L';
                    val = (consoleInput[4]-48)+(10*(consoleInput[3]-48))+(100*(consoleInput[2]-48));
                    if( val < 270 || val > 460 ) {
                        Serial.println(F("cell voltage value out of range"));
                        break;
                    }
                    if((consoleInput[1] == 't') || (consoleInput[1] == 'T')) Vmin[0] = val/100.0;
                    else if((consoleInput[1] == 'f') || (consoleInput[1] == 'F')) Vmin[1] =
val/100.0;
                    else Serial.println(F("unrecognised cells model"));
                    break;
                case 'h':
                case 'H': whatToDisplay = 'H';
```

```

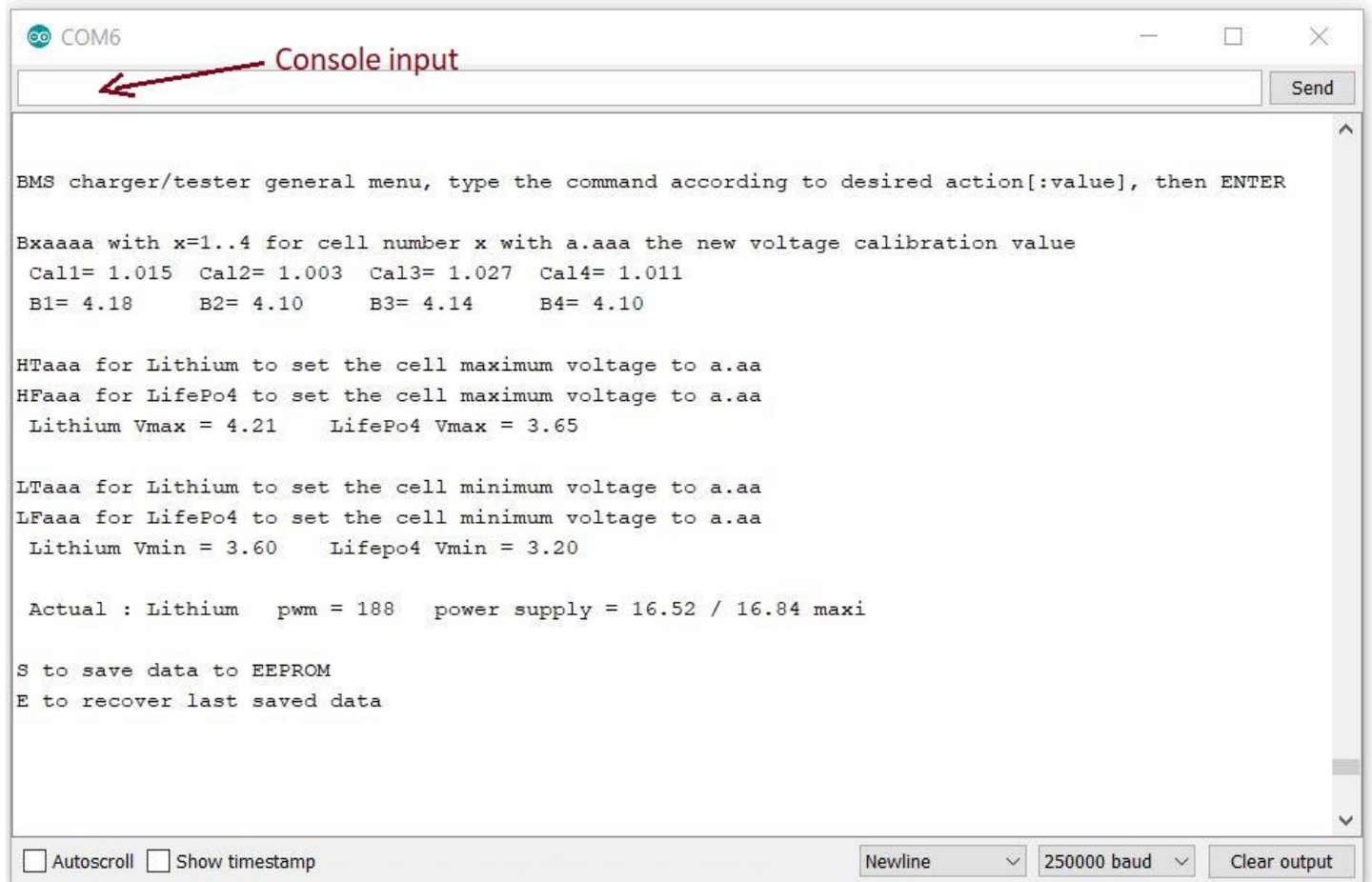
        val = (consoleInput[4]-48)+(10*(consoleInput[3]-48))+(100*(consoleInput[2]-48));
        if( val < 270 || val > 460 ) {
            Serial.println(F("cell voltage value out of range"));
            break;
        }
        if((consoleInput[1] == 't') || (consoleInput[1] == 'T')) vmax[0] = val/100.0;
        else if((consoleInput[1] == 'f') || (consoleInput[1] == 'F')) vmax[1] =
val/100.0;
        else Serial.println(F("unrecognised cells model"));
        break;
    case 'e':
    case 'E': whatToDisplay = 'A';
              Serial.println(F("cancel all changes!"));
              EEPROM_Get();
              break;
    case 's':
    case 'S': whatToDisplay = 'A';
              Serial.println(F("all data saved!"));
              EEPROM_Update();
              break;

    default : whatToDisplay = 'A';
              Serial.println(F("unrecognised command"));
              break;
    } // end of switch
  } // en of else
} // end of test Serial.available()
} // end of serialEvent()

```

The first time you use the device, some setup adjustment must be performed: the cells voltage calibration.

- 1- Connect 4 Li-ion or LifePo4 cells to -B/B1, B1/B2, B2/B3 and B3/B4.
- 2- Select with SW1 the cell technology: either 4 Lithium or 4 LifePo4
- 3- Switch on the device with SW2. Note it is possible to connect the cells once the device is on, if some shunts become active press the reset push-button of the Arduino Uno.
- 4- Connect the USB of the Arduino Uno to a PC, the console display should looks like:



```
COM6

BMS charger/tester general menu, type the command according to desired action[:value], then ENTER

Bxaaaa with x=1..4 for cell number x with a.aaa the new voltage calibration value
Cal1= 1.015  Cal2= 1.003  Cal3= 1.027  Cal4= 1.011
B1= 4.18    B2= 4.10    B3= 4.14    B4= 4.10

HTaaa for Lithium to set the cell maximum voltage to a.aa
HFaaa for LifePo4 to set the cell maximum voltage to a.aa
Lithium Vmax = 4.21    LifePo4 Vmax = 3.65

LTaaa for Lithium to set the cell minimum voltage to a.aa
LFaaa for LifePo4 to set the cell minimum voltage to a.aa
Lithium Vmin = 3.60    Lifepo4 Vmin = 3.20

Actual : Lithium  pwm = 188  power supply = 16.52 / 16.84 maxi

S to save data to EEPROM
E to recover last saved data

Autoscroll Show timestamp Newline 250000 baud Clear output
```

With the help of a multimeter you need to adjust cells voltages displayed on the SSD so that they are as close as possible to the multimeter.

For instance:

- to set cell 1 to 101.1%, you type B11011<ENTER>
- to set cell 3 to 99.5%, you type B30995<ENTER>
- to set LifePo4 Vmax to 3.65V, you type HF365<ENTER>

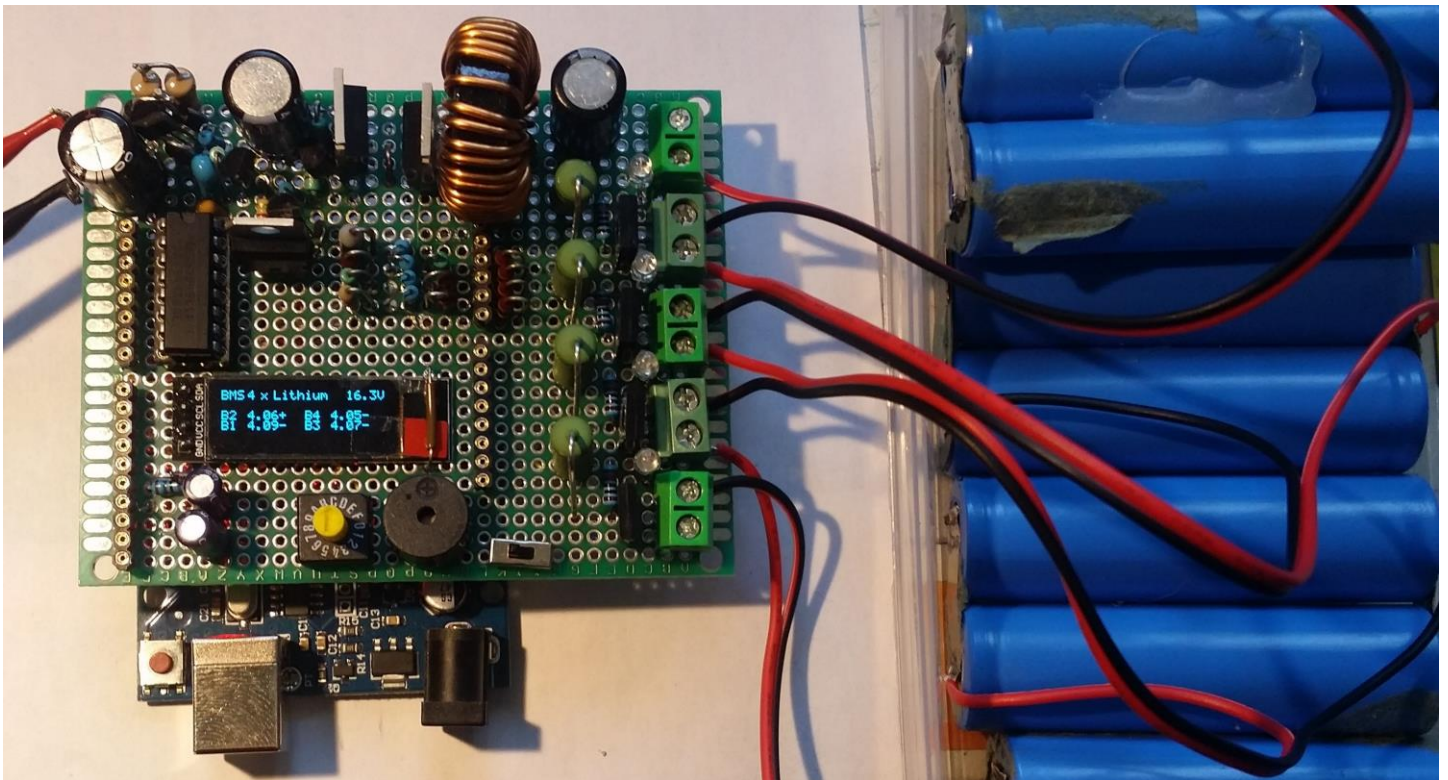
If the new voltage values are correct type S<ENTER> to save them.

- 5- Once a cell is charged, the according LED is ON as a witness to the shunt function. The shunt will stay on until an Arduino reset or after a delay of 5 minutes that reset all flags. By the way if the cell is really charged to according LED will remain bright within a short delay.
- 6- In opposite any deficient cell or under voltage will tone the buzzer. Once gain only an Arduino reset or a delay of 5 minutes that reset all flags.
- 7- Once all shunts are ON it is possible to disconnect the power supply. The cell are then discharging though their own shunt resistor. Any defective cell is then easily noticeable.

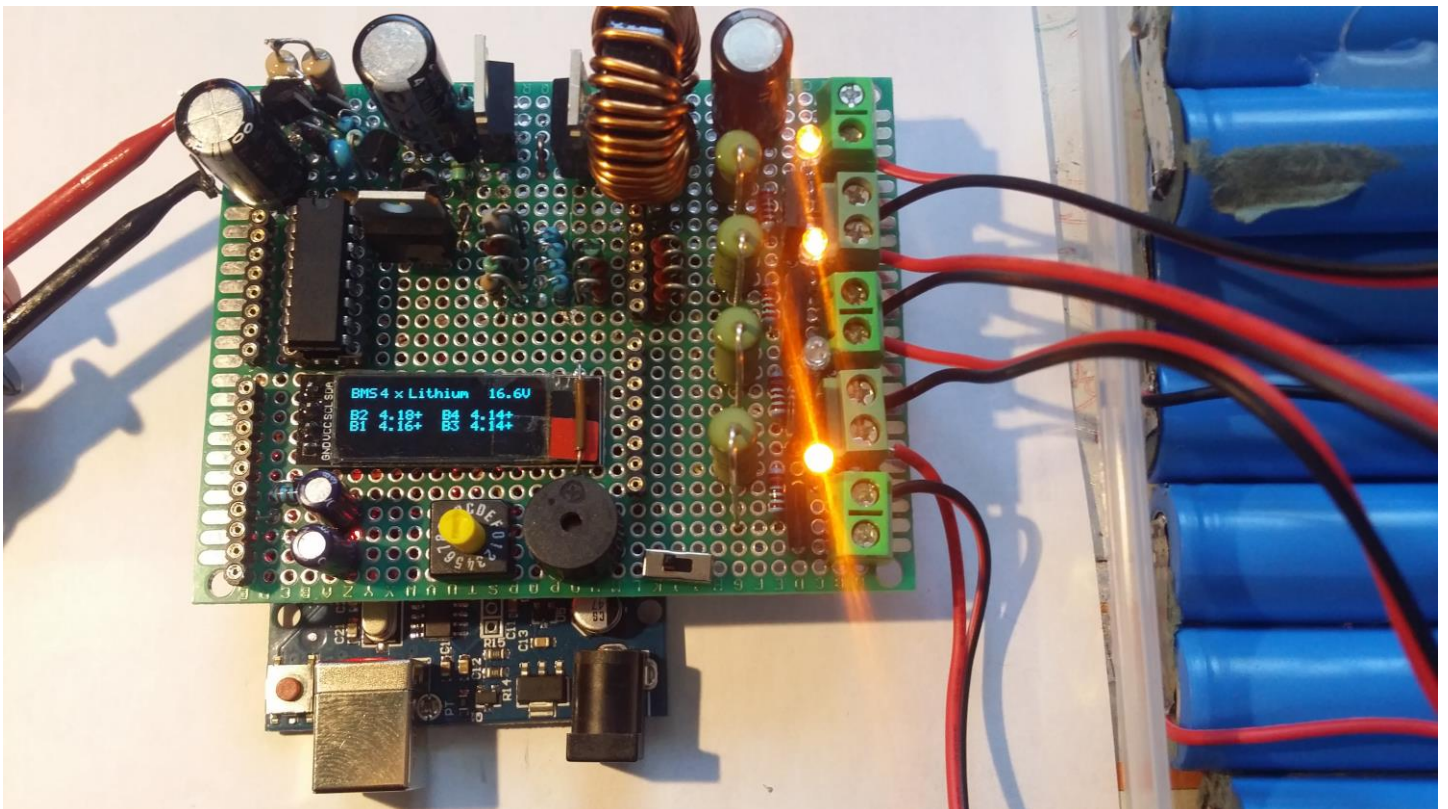


## Illustration in use

Here is an example when cells are connected with no power supply.

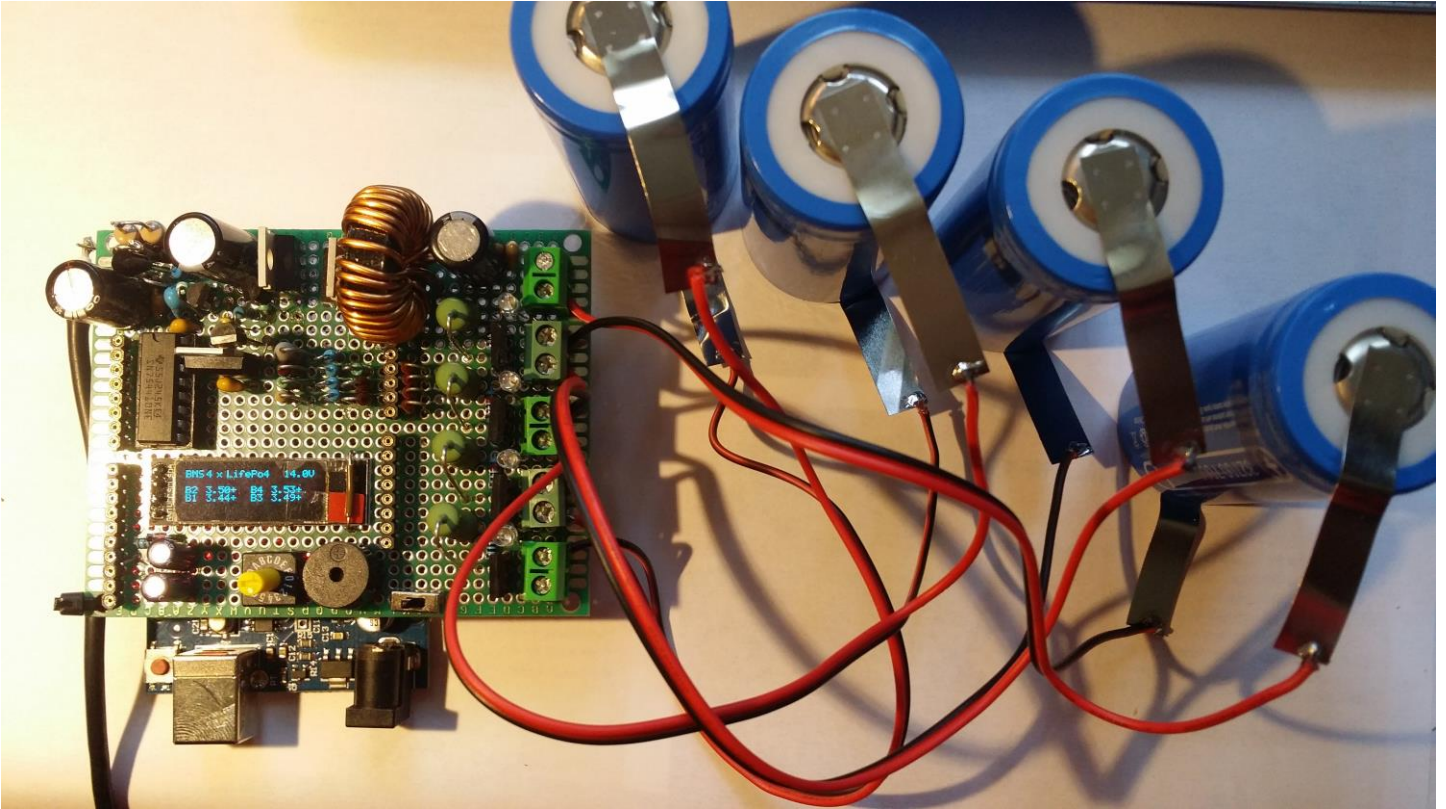


Here is an example during charging process with the power supply, when all cells are full charged except B1:





Here is a picture during charging 4 standalone LifePo4 7000mA cells:



Here is a picture during discharging test for 12x4 Li-ion battery pack (no power supply):

