Final Report

Battery systems hardware

|  |
| --- |
|  |

**Client:** BatteryNL

**Tutors:** Ralph Goes | Harold Benten

**By:** Julian Meeuwis

**Document ID:** MDDMO8.docx

|  |  |  |
| --- | --- | --- |
| Release | Data | Changes |
| R1 | 24-02-2025 | Start document |
| R2 | 18-07-2025 | Finalizing document |

**Authenticated by:** Julian Meeuwis

Contents

[1. Introduction 3](#_Toc203751142)

[2. Host Board 4](#_Toc203751143)

[2.1. Introduction 4](#_Toc203751144)

[2.2. Schematics 5](#_Toc203751145)

[2.3. Full schematic 8](#_Toc203751146)

[3. Cell board 9](#_Toc203751147)

[3.1. Block diagram 10](#_Toc203751148)

[3.2. Design 11](#_Toc203751149)

[3.3. Calculations 13](#_Toc203751150)

[3.4. Schematic 14](#_Toc203751151)

[3.5. PCB 15](#_Toc203751152)

[4. Software 16](#_Toc203751153)

[4.1. Initial setup 16](#_Toc203751154)

[4.2. Digital Potentiometer AD5282 17](#_Toc203751155)

[4.3. ADG706 Multiplexer 20](#_Toc203751156)

[4.4. MCP9700 Temperature sensor 22](#_Toc203751157)

[4.5. ESP32 code 23](#_Toc203751158)

[4.6. Matlab GUI(App-designer) 24](#_Toc203751159)

[5. Conclusion 25](#_Toc203751160)

[5.1. Cell board 25](#_Toc203751161)

[5.2. Software 25](#_Toc203751162)

[6. Recommendations 26](#_Toc203751163)

[6.1. Host board 26](#_Toc203751164)

[6.2. Cell board 26](#_Toc203751165)

[6.3. Software 27](#_Toc203751166)

[7. Appendices 28](#_Toc203751167)

[BOM 28](#_Toc203751168)

[Code 29](#_Toc203751169)

[Sims 35](#_Toc203751170)

# Introduction

**Motivation:**

Battery management system (BMS) testing can be quite hazardous. Modern batteries store a lot of energy, and, if they catch fire, can be difficult to extinguish. On the other hand, an insufficiently tested BMS represents a hazard to users and the environment. The solution is to emulate the battery and the load so that testing can be completed under a wide range of conditions without taking excessive risks. In the past other people and groups have worked on this project. The goal is to continue development where the previous groups left off.

**Project results:**

For this project, a hardware-based battery emulator has already been developed. The software that came with it is barebone, and the hardware has some issues. This mean that there are a few main goals to successfully finish this project. This being:

* Testing previous hardware: The hardware that has been obtained from a previous group, who worked on it before, must be tested and validated to ensure that the system has full functionality for the use case.
* Creating a **stable** prototype and fix the known issues obtained from the previous group
* When possible, integrate emulating static load for specific battery types: many kinds of batteries are being used in the modern world. It is important to BatteryNL that the most used battery types (and maybe even future types) can all be emulated on the hardware used.

# Host Board

During the period the focus was creating a stable prototype, which required solving the known problems. Most known problems came from the cellboard. The discission was made to mostly focus on the cellboard. Most work done on the host board is coming from the previous group. This chapter is copied from the previous group and edited where necessary.

## Introduction

The primary purpose of the host board is to send commands to the cell boards, control their operations, receive all the data they transmit, and provide power to all the cell boards.

|  |
| --- |
| Figure 1 |

A detailed explanation of the host board's functionality and the role of each component will be provided later. In summary, as illustrated in the block diagram above, the main purpose of the host board is to supply power to the cell boards and communicate with them using the I2C protocol. Commands are sent directly to the cell boards via I2C, and in return, the host board receives data on the voltage, current, and temperature of the cells.

Since the microprocessor does not have enough pins to manage all these inputs and some input ranges may exceed its 3.3V rating, multiplexers are used. These multiplexers collect measurements from the cell boards, regulate the signals, and transmit the specific data for each measurement to the microcontroller (MCU) based on the commands given to the multiplexers.

## Schematics

All design steps that have been made for each part of the Host board are explained in the following sub-chapters.

### Current measuring

|  |
| --- |
| Figure 2 |

The first part of the system is the current measurement section that uses the ACS711ELCTR-12AB-T, a Hall-effect-based linear current sensor, to monitor and ensure safe current flow throughout the system. It measures the current through its input pins and outputs a voltage proportional to the detected current, which is processed by the microcontroller. This allows real-time monitoring to ensure the system only draws the required current, helping detect abnormal behavior, such as overcurrent or inefficiencies. Key components like decoupling capacitors stabilize the power supply, while a Schottky diode (5.0SMDJ14A) provides protection against reverse polarity and voltage spikes, ensuring accurate and reliable operation.

The sensor is outputting voltage so to be able read the measured current we will have to convert it to current using the following formula:

|  |
| --- |
|  |

Where 𝐼𝑃 is the measured current in Amps.

𝑉𝐼𝑂𝑈𝑇 is the output voltage from the sensor.

𝑉𝐼𝑂𝑈𝑇(𝑄) is the quiescent output voltage at 0A, (It’s VCC/2, which is 1.65V for the 3.3V Supply).

Sensitivity = 110mV/A

### Power management

|  |
| --- |
| Figure 3 |

In the power section, the main 12V power supply is stepped down to 5V and further to 3.3V using AMS1117 voltage regulators. Additionally, an MEA1D1212SC module is used to generate -12V, enabling the cell board amplifiers to operate with ±12V. A 3.3V indication LED is included to confirm the presence of the 3.3V supply. A voltage divider is used to deliver a 1V signal to the MCU when the input power supply is 12V, serving as an indicator of sufficient system power. Lastly, pull-up resistors are connected to the I2C data and clock lines, as required for proper I2C communication.

### MCU

|  |
| --- |
| Figure 4 |

One of the key components of the host board is the MCU, for which the ESP32-S3-WROOM-2 was selected due to its affordability, availability, and previous use in the earlier group's design. The ESP32 operates on the I2C protocol, and two buttons are required: the Enable (EN) button, connected to the EN pin to reset the processor, and the Boot button, connected to GPIO 0 for flashing firmware. However, in the current iteration, the Boot button was mistakenly connected to GPIO 8 instead of GPIO 0, which needs correction in the next iteration. Additionally, a USB micro-B connector is used to connect to the D+ and D- pins of the MCU. I2C connections are wired with SCL to IO47 and SDA to 8 IO48. An LED indicator is included, which can be controlled via the MCU to assign various system status signals.

### Multiplexer

|  |
| --- |
| Figure 5 |

The multiplexer used is the ADG706BRUZ, selected for its ability to handle 16 channels in a single device, eliminating the need for multiple multiplexers. It routes signals from the cell boards (voltage, current, and temperature) to a single output for microcontroller processing, efficiently managing up to 16 input channels while reducing the number of required MCU pins. The control pins (MUX0–MUX3) dynamically select the active input channel, allowing sequential monitoring of cell parameters. A voltage divider is placed at the output to scale down signals by half, ensuring that voltages exceeding 3.3V from the cell boards do not damage the MCU. The Enable pin must be connected to either the MCU or a 3.3V source to activate the chip; currently, it is mistakenly connected to ground, which needs correction in the next iteration.

### Connectors

|  |
| --- |
| Figure 6 |

The decision has been made to use four Cellboards for this iteration, leading to the addition of four connectors to the Host Board. Each connector is providing both power and communication. Through these connectors, all parameters from each Cellboard can also be read back.

## Full schematic

|  |
| --- |
| Figure 7 |

|  |  |
| --- | --- |
| Figure 8 | Figure 9 |

# Cell board

This chapter consists of debugging the previous cell boards and developing a new stable iteration of them.

Most of the known problems originated from the cell board. Some problems were already known, and after intensive testing of the previous hardware, the following problems were noted:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | Problem | Rational | Impact | Solution |
| LDO | Oscillation | Less efficient | Medium | * Remove “offset circuit” improvised by previous group. |
| Thermal | Runaway | Board damage | Critical | * Series resistance (few tens to hundred milli ohms) |
| Potentiometer | Rush current | Board and component damage. | Critical | * Memory (how much rewriting?). * Inversing wipers? |
| Potentiometer / I2C | No response after “full” reboot |  | Critical | * Power up sequency * Stable voltage supply * Good I2C lines |
| Thermal | Heat generation | Heat decreases overall efficiency. | Medium | * Lower supply voltage, less drop over LDO. * Increase pad size and improve overall thermal surface (Vias) |
| Sensor | Wrong current scaling |  | Small | * Adjust software scaling. AND/OR Adjust hardware scaling. |

The primary challenges that rendered testing "impossible" were related to the I2C communication and inrush current. A significant issue observed during system boot-up was the surge of current, which exceeded 3A. This excessive current led to a voltage drop that hindered communication with the digital potentiometer. The voltage drop occurred specifically when the potentiometer's values were adjusted. However, the system could still be tested by hotplugging the connector.

## Block diagram

|  |
| --- |
| Figure 10 |

* Potentiometer: this is a dual potentiometer which is controlled by the host board via I2C protocol. The potentiometer changes resistance what will result in different voltage and current at the output.
* Voltage controller: based on the resistance on the set pin of the controller a voltage is regulated at its output.
* Current controller: based on the voltage divider driven by the potentiometer the current is regulated using an opamp mosfet buffer.
* Temperature sensor: to maintain the temperature on the board a sensor is added to measure this and send back to the host board.
* Scaling: to check if the output behaves as desired the output will be scaled to 0 – 3.3V and send back to the host board

## Design

### LDO

The voltage controller employs two LT3080, as selected by previous teams. The output voltage can be adjusted by modifying the resistance between the set pin and ground, with the output voltage calculated as the resistance value multiplied by 10μA. A desired output voltage range of 0.8V to 4.5V is targeted. To achieve this output, the two voltage controllers are connected in parallel, which helps distribute the heat generated by the LDOs and allows for a higher current capacity. Additionally, a shunt resistor is placed in series with each LDO to prevent thermal runaway

|  |
| --- |
| Figure 11 |

### Linear current regulator

This system operates by regulating a current through a shunt resistor, known as a 'sense' resistor, by regulating the voltage across it. By utilizing an op-amp MOSFET buffer, the voltage drop across the shunt can be controlled by simply inputting the desired value into the op-amp. Once the voltage across the shunt is established, the current flowing through the system can be accurately determined.

|  |
| --- |
| Figure 12 |

### Digital potentiometer / resistor dividers

A digital potentiometer is used to control both the current and voltage controllers. In the previous iteration, the resistor divider of the current controller was incorrect. This has been corrected and inverted.

|  |
| --- |
| Figure 13 |

Step 0 for current wiper, current approaches 0 mA.

Step 0 for volt wiper, voltage approaches 0.8 V.

## Calculations

Something about the changes made in the voltage dividers (digital potentiometers)

**Voltage controller:**

As earlier said the output of the voltage controller is based on the set resistance, to calculate the output voltage the following formula is used:

|  |
| --- |
| 𝑉𝑜𝑢𝑡 = 10μ ∗ 𝑅𝑠𝑒𝑡 |

The 10μA is derived from the datasheet of the LT3080. Because two LT3080 are placed in parallel the formula becomes:

|  |
| --- |
| 𝑉𝑜𝑢𝑡 = 10μ ∗ 𝑅𝑠𝑒𝑡 ∗ 2 |

The output voltage must be between 0.8 V and 4.5V, to calculate the resistance value for these voltages. The potentiometer needs a range of

𝑅𝑟𝑎𝑛𝑔𝑒 = 𝑅𝑠𝑒𝑡𝑚𝑎𝑥 − 𝑅𝑠𝑒𝑡𝑚𝑖𝑛 = 225𝑘 − 50𝑘 = 175𝑘Ω

So a potentiometer of 200kΩ is chosen, but the maximum value can’t be achieved by only the potentiometer so an offset resistor is placed in series. The value of the offset resistor is 39kΩ, this is also chosen to make sure that potentiometer won’t have to go to its limits.

The minimum Rset value will then become: 𝑅𝑠𝑒𝑡𝑚𝑖𝑛 = 50𝑘 − 39𝑘 = 11𝑘Ω

The maximum Rset value will then become: 𝑅𝑠𝑒𝑡𝑚𝑎𝑥 = 225𝑘 − 39𝑘 = 186𝑘Ω

**Current controller**:

The current controller works by making use of the second part of the potentiometer. And using this as a voltage divider. The position of the wiper will determine the amount of voltage that will be sent to the opamp. This can be shown with the following formula:

|  |
| --- |
|  |

A maximum output voltage of 1.25 V is chosen since this result in a maximum current of 2.5A. This will allow a deviating from the required maximum current of 2A

## Schematic

|  |
| --- |
| Figure 14 |

## PCB

Several choices were made during the PCB design process:

* The heat dissipation pads under the ICs were enlarged, and vias were positioned more efficiently for better heat dissipation.
* A switch system was added to completely disconnect the 12V line from the system. This allows the user to control the boot sequence and prevent surge current. This switch can be controlled with either a jumper or the ESP32.
* The ground plane is as intact as possible.
* As few components as possible were used to keep the system as simple as possible—"less is more."

|  |  |
| --- | --- |
| Figure 15 | Figure 16 |

|  |  |
| --- | --- |
| Figure 17 | Figure 18 |

# Software

## Initial setup

This chapter describes all design steps made for the software. The MCU used in this project is ESP32S3 and the communication protocol is I2C. First, the software settings had to be set properly to avoid any errors in the beginning. As shown in the Figure [19] below, the correct settings were adjusted compared to the old code. It was important to choose **ESP32S3 Dev Module** for the board setting, and to set **USB CDC On Boot** to **Enabled**. It is recommended to Enable most of the “Tools” functions to avoid errors.

|  |
| --- |
| Figure 19 |

## Digital Potentiometer AD5282

The software was provided by the previous group. It is fully functional but not optimized. Some attempts have been made to improve the software.

### I2C protocol

Communication with digital potentiometer is necessary to change current and voltage on the cell board. To communicate with AD5282, the I2C protocol needs to be used and understood. When using I2C, the data sent to from master to slave need a specific binary sequence. It is shown in the figure [20] Below.

|  |
| --- |
| Figure 20 |

To understand what is happening during this data transfer, the instruction below needs to be followed:

• Both SDA and SCL should be high

• To start the protocol the Data pin should be pulled down, followed by the SCL pin

• After that, the Clock pin should work on its normal speed

• Then, you need to send the Slave address that you want to access with the master

• Now, the request for either read(0) or write(1) needs to be send

• After that, the acknowledge signal from the potentiometer (ACK) should be received back

• Now, the data cen be send. That should be followed by another ACK signal from the slave device.

• Finally the protocol can be stopped using Clock pin set to high and pulling the SDA to high after that.

The understanding of the protocol sequence from above, will help when dealing with I2C syntax. For this project, using the library <Wire.h> made it easier to work with the ISC using the Arduino IDE platform.

**Previous tests revealed that the system's bootup remained very inconsistent. This was due to low voltage on the 12V, 5V, and I2C lines. However, this sequence proved most successful:**

|  |  |  |
| --- | --- | --- |
| Steps | First connected everything | |
|  | Power on Power supply totalsystem | |
|  | Connect USB |
|  | Send Command to zero all | |
|  | Power on cells |
|  | Continue regular use | |

### AD5282 addresses

As mentioned in chapter above, digital potentiometer has two address pins: A0 and A1. As stated in the datasheet of the component, different configurations of those pins gives four possible addresses for cell boards. Since in the datasheet there was no given addresses, they were obtained by using I2Cscanner code online. This code implements a scanner to locate I2C devices and determine if a device is connected. By doing so the addresses were read and are shown in the table below.

|  |
| --- |
| Figure 21 |

### AD5282 wiper

The AD5282 digital potentiometer has two wipers, Wiper 1 and Wiper 2, which can be addressed via I2C using specific register values: 0 (binary 00000000) for Wiper 1 and 128 (binary 10000000) for Wiper 2, with the most significant bit indicating the target wiper. Using the <Wire.h> and the sequence of I2C protocol explained in chapter 3.2.1. the code for adjusting the wiper position was written. It is shown in Figure [22] below:

|  |
| --- |
| Figure 22 |

As shown in Figure above, communication begins with beginTransmission(SLAVE\_ADDRESS) to initiate the connection.

This is followed by the first Wire.write, which specifies the register corresponding to the desired wiper.

The second Wire.write then sends a value between 0 and 255 (minimum and maximum resistance) to set the wiper's position on the potentiometer's resistor network.

Finally, endTransmission completes the data transfer, applying the command.

Based on a hardware requirements, specific step limits had to be applied to the software:

Step 0 for current wiper, current approaches 0 mA

Step 0 for volt wiper, voltage approaches 0.8 V

## ADG706 Multiplexer

The second important component in the battery emulation project is ADG706 Multiplexer. This component was used to read the value of either voltage controller, current controller or the temperature sensor. By activating the multiplexer through its address pins, the value from the specific S-Pins can be read by the MCU.

### ADG706 Addresses

As mentioned above, multiplexer communicates with other components using S-pins. To request an input from the specific one, the sequence of bit value for A3 to A0 needs to be used. The truth table (from multiplexer’s datasheet) together with assigned S-pins and components connected is shown in Table [23] below:

|  |
| --- |
| Figure 23 |

With that instruction in mind, the code used to call S-pins of Multiplexer is shown in Figure [24] below:

|  |
| --- |
| Figure 24 |

### ADG706 Voltage

The function for reading voltage read and calculates the voltage from the Cell board via a multiplexer by first selecting the appropriate channel through the control pins (A3-A0). After a short delay, the function read the raw ADC value from the multiplexer’s output DOUT\_PIN. Due to the output fluctuation, the adc value is averaged by the number of bits. The averaged ADC value is converted to voltage using a formula:

|  |
| --- |
|  |

3.3 / 4096 which scales the 12-bit ADC reading to the microcontroller's 3.3V reference, multiplication by 2 stands for the voltage divider at the multiplexer output, division by 0.65 adjusts for the voltage divider caused by the voltage controller. The final voltage is printed with two decimal numbers.

### ADG706 Current

The function for reading current reads and calculates the current using the same principal as the function above. After a short delay, the function read the raw ADC value from the multiplexer’s output DOUT\_PIN. Similarly to the function for reading the voltage, the adc value is averaged by the number of bits to improve accuracy. For each sample, the ADC value is converted to current using the formula:

|  |
| --- |
|  |

3.3 / 4096 scales the 12-bit ADC reading to the microcontroller’s 3.3V reference voltage, multiplication by 2 compensates for the voltage divider at the multiplexer output, division by 3.3 normalizes the value for maximum resolution of the input range, and final division by 0.5 converts the voltage into current based on the 0.5Ω shunt resistor.

## MCP9700 Temperature sensor

The final component used for the testing with software is MCP9700 temperature sensor. By reading the analog value from the component, the temperature of the cell board near the most temperature depended components (voltage controller) could be calculated and displayed on the serial monitor.

### MCP9700 reading the temperature

The function for reading temperature reads the temperature from the MCP9700 temperature sensor using the multiplexer, following the same process as described for voltage and current readings. The analog signal is converted to voltage which scales the 12-bit ADC reading to the 3.3V reference range of the microcontroller. Then, the temperature is calculated using the formula from the MCP9700 datatsheet:

|  |
| --- |
|  |

Where V0°C​ is the sensor's output voltage at 0 °C and Tc​ is the temperature coefficient. Although the MCP9700 datasheet specifies V0°C​=0.5V, testing revealed that 0.16 V provided more accurate results for matching the actual room temperature. The temperature coefficient is defined as 10 mV/°C. The scaling factor of 1.25 is used to adjust the temperature calculation for calibration purposes, ensuring precision.

## ESP32 code

The final code dynamically configures and monitors the voltage and current.

It accepts user input with just one command: va(voltage1)ca(current1)vb(voltage2)cb(current2)vc(voltage3)cc(current3)vd(voltage4)cd(current4)

where:

• va[voltage1] – Potentiometer step for Cell’s 1 voltage

• ca[current1] – Potentiometer step for Cell’s 1 current

• vb[voltage2] – Potentiometer step for Cell’s 2 voltage

• cb[current2] – Potentiometer step for Cell’s 2 current

**etc**

This approach allows voltage and current values to be specified for each board. These inputs are validated and mapped to control the AD5282 digital potentiometer, where Wiper 1 adjusts the voltage and Wiper 2 adjusts the current for each board via I2C communication.

The multiplexer is used to read the voltage and current from each cell, with specific channel configurations set using digital pins. ADC values are sampled and averaged to minimize noise, and conversion formulas translate these values into voltage and current readings. This process allows for real-time, flexible control and monitoring of the cell boards, suitable for battery emulator applications.

During this period, the code itself remained largely unchanged. The available code was scaled from 2 to 4 cells. While the code is fully functional, it's not perfect. The previous group hard-coded many values due to limited flexibility. Also, some functions are not modular, which takes up a lot of space. The code was left as is due to time constraints. However, suggestions for implementing code simplification can be found in the appendices.

## Matlab GUI(App-designer)

To keep the software as universal as possible, the GUI is based on the existing serial system. Matlab App Designer was used to create the GUI. The GUI functionality is minimal, and it is recommended to create a new iteration. However, many lessons have been learned and can easily be implemented using the designer.

|  |
| --- |
| Figure 25 |

The serial COM port can be selected from the COM dropdown menu, after which pressing the Connect button establishes a connection. To change the values of a single cell, select the appropriate cell from the menu at the top. After changing the value, confirm the changes by clicking the Apply button.

**It's important to note that the GUI is designed for older hardware. The scaling applied to the cell boards is inverted, as described in the "Cell Board" section.**

|  |
| --- |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 26 |

The parts surrounded in green are functioning well/partially. The parts surrounded in red are not functioning.

The GUI code can be found in the appendices

# Conclusion

After testing the new hardware, it can be concluded that it is a success. The board responds to ESP32 commands, and the code has been scaled to four cells. A Matlab GUI was also experimented with. The conclusion is that the cell board provides sufficient stability for scaling. BOM is available in appendices.

## Cell board

The cell board functions well but gets very warm. I expected a lower supply voltage to work out of the box, but this doesn't guarantee good communication.

The system can be cooled well with a small fan. Passive cooling is preferred, of course. It's unclear how realistic this is. I suspect that if the supply voltage can drop to 9 V and the pads become a bit larger, this will be possible.

## Software

The software works, but it's very cluttered. A GUI has been experimented with, which has minimal functionality. A boot sequence is recommended for proper system operation.

# Recommendations

## Host board

Recommendations from pervious group:

* Connect the Boot button to the ESP32 GPIO0.
* Replace the USB micro-B connector with a USB type-C connector to align with the new European standard
* Remove power line from USB as a whole.
* The Enable pin of the multiplexer must be connected to either the MCU or a 3.3V source to activate the chip, now done by jumper

Recommendations from current:

* Change hostboard to handle lower supply voltage, allow lower drop over LDOs which increase efficiency.
* Pinheaders or connectors for cooling fans, this could be on the mainboard but i suggest on cellboards
* Simplefy by removing -12V supply. MAYBE!
* When possible, remove multiplexer to simplify design. I think ESP32 has more than enough pins for this.

## Cell board

* Improve PCB design with for example better symbols / text for better user experience.
* Increase pad size even more for better heat dissipation. Make case with active cooling
* **Shorter wires between Mainboard & cellboard because high current introduces voltage drop, which is especially for the i2C a big problem**
* When designing new PCB, use logical sizes.

## Software

* Maybe design GUI from scratch. Do this in matlab since its extremely easy to get it to work!
* Add something like startup sequence based on testing.
* Remove hard coded values from code and make more modular like suggestions in appendices.
* Migrate code to VScode instead of using Arduino IDE.
* Simplify code and make function more universal/modular.

# Appendices

## BOM

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Id | Designator | Footprint | Quantity | Designation | Supplier and ref | Order code |
| 1 | U4,U1 | TO-263-5\_TabPin3 | 2 | LT3080xQ | nl.farnell.com | 4027648 |
| 2 | F1 | Fuse\_1812\_4532Metric\_Pad1.30x3.40mm\_HandSolder | 1 | 1812L200/12DR | nl.farnell.com | 2786590 |
| 3 | C6 | CP\_Radial\_D5.0mm\_P2.00mm | 1 | 10u |  |  |
| 4 | R13,R4,R9,R5 | R\_0805\_2012Metric\_Pad1.20x1.40mm\_HandSolder | 4 | 1k |  |  |
| 5 | C2 | CP\_Radial\_D5.0mm\_P2.00mm | 1 | 1u |  |  |
| 6 | R22 | R\_0805\_2012Metric\_Pad1.20x1.40mm\_HandSolder | 1 | 120 |  |  |
| 7 | R20,R14,R8 | R\_0805\_2012Metric\_Pad1.20x1.40mm\_HandSolder | 3 | 10k |  |  |
| 8 | J3 | PinHeader\_1x02\_P2.54mm\_Vertical | 1 | Conn\_01x02\_Pin | |  |
| 9 | R21 | R\_Axial\_Power\_L25.0mm\_W9.0mm\_P7.62mm\_Vertical | 1 | 0.5 |  |  |
| 10 | Q1 | TO-263-2 | 1 | IRFS4115 | nl.farnell.com | 2725984 |
| 11 | J2 | Molex\_SPOX\_5267-10A\_1x10\_P2.50mm\_Vertical | 1 | Conn\_01x10\_Pin | nl.farnell.com | 3050985 |
| 12 | J1 | TerminalBlock\_Phoenix\_MKDS-1,5-2-5.08\_1x02\_P5.08mm\_Horizontal | 1 | Screw\_Terminal\_01x02 | | |
| 13 | 12V,5V | LED\_1206\_3216Metric\_Pad1.42x1.75mm\_HandSolder | 2 | LED |  |  |
| 14 | R3 | R\_0805\_2012Metric\_Pad1.20x1.40mm\_HandSolder | 1 | 390 |  |  |
| 15 | J5 | PinHeader\_1x03\_P2.54mm\_Horizontal | 1 | Conn\_01x03\_Pin | |  |
| 16 | SW1 | SW\_DIP\_SPSTx02\_Slide\_9.78x7.26mm\_W7.62mm\_P2.54mm | 1 | BSE02GR | nl.farnell.com | 2864305 |
| 17 | R11,R12 | R\_0805\_2012Metric\_Pad1.20x1.40mm\_HandSolder | 2 | 10m |  |  |
| 18 | Q2 | SOIC-8\_3.9x4.9mm\_P1.27mm | 1 | FDS4897C | nl.farnell.com | 2453864 |
| 19 | R23,R6 | R\_0805\_2012Metric\_Pad1.20x1.40mm\_HandSolder | 2 | 1M2 |  |  |
| 20 | U2 | TSSOP-16\_4.4x5mm\_P0.65mm | 1 | AD5282 | nl.farnell.com | 4028680 |
| 21 | R2 | R\_0805\_2012Metric\_Pad1.20x1.40mm\_HandSolder | 1 | 2k7 |  |  |
| 22 | C1,C5,C3 | C\_0805\_2012Metric\_Pad1.18x1.45mm\_HandSolder | 3 | 100n |  |  |
| 23 | R1 | R\_0805\_2012Metric\_Pad1.20x1.40mm\_HandSolder | 1 | 1k2 |  |  |
| 24 | U6 | SOT-23 | 1 | MCP9700Ax-ETT | nl.farnell.com | 1605577 |
| 25 | R10 | R\_0805\_2012Metric\_Pad1.20x1.40mm\_HandSolder | 1 | 2k2 |  |  |
| 26 | U3 | SO-14\_3.9x8.65mm\_P1.27mm | 1 | TL084 | nl.farnell.com | 1564346 |
| 27 | R7 | R\_0805\_2012Metric\_Pad1.20x1.40mm\_HandSolder | 1 | 39k |  |  |

## Code

**Main code**

|  |
| --- |
| #include <Arduino.h>  #include <Wire.h>  // AD5282 Definitions  #define WIRE Wire  #define SLAVE\_ADDRESS\_1 47 // Address for Cell Board 1  2F  #define SLAVE\_ADDRESS\_2 44 // Address for Cell Board 2  2C  #define SLAVE\_ADDRESS\_3 46 // Address for Cell Board 3  2E  Maybe switch?  #define SLAVE\_ADDRESS\_4 45 // Address for Cell Board 4  2D  Maybe switch?  #define PIN\_SDA 48        // SDA connected to ESP32  #define PIN\_SCL 47        // SCL connected to ESP32  //  Maybe impliment hard current limit?  // MUX Definitions  #define EN\_PIN 45         // Enable pin on ESP32  #define A0\_PIN 11         // Address pin A0 on ESP32  #define A1\_PIN 12         // Address pin A1 on ESP32  #define A2\_PIN 13         // Address pin A2 on ESP32  #define A3\_PIN 10         // Address pin A3 on ESP32  #define DOUT\_PIN 9        // Common OUT for MUX  #define SAMPLE\_COUNT 128  // Bits for averaging  //  Cellboard Enable pins  #define Aenable 6         //  Change pin number  #define Benable 6         //  Change pin number  #define Cenable 6         //  Change pin number  #define Denable 6         //  Change pin number  // Constants for MCP9700 temperature sensor  const float V0C = 0.16;   // Voltage at 0°C in volts  const float TC = 0.01;    // Temperature coefficient in volts/°C (10 mV/°C)  void setup() {    // Initialize MUX pins    pinMode(EN\_PIN, OUTPUT);    pinMode(A0\_PIN, OUTPUT);    pinMode(A1\_PIN, OUTPUT);    pinMode(A2\_PIN, OUTPUT);    pinMode(A3\_PIN, OUTPUT);    pinMode(DOUT\_PIN, INPUT);    // Enable the multiplexer    digitalWrite(EN\_PIN, HIGH);      //  Begin communication over UART    Serial.begin(9600);    Serial.println("Enter command in the format 'va[voltage1]ca[current1]vb[voltage2]cb[current2]vc[voltage3]cc[current3]vd[voltage4]cd[current4]'.");    Serial.println("Example: va128ca10vb128cb10vc128cc10vd128cd10");    // Initialize I2C for AD5282    WIRE.setPins(PIN\_SDA,PIN\_SCL);    WIRE.begin(); // maybe set frequency? 100000    }  void loop() {    if (Serial.available())    {      String input = Serial.readStringUntil('\n');      input.trim(); // Remove extra spaces or newline characters      delay(100);      // Parse commands for all four boards      int vaIndex = input.indexOf("va");      int caIndex = input.indexOf("ca");      int vbIndex = input.indexOf("vb");      int cbIndex = input.indexOf("cb");      int vcIndex = input.indexOf("vc");      int ccIndex = input.indexOf("cc");      int vdIndex = input.indexOf("vd");      int cdIndex = input.indexOf("cd");      int ResetCom = input.indexOf("reset");      //  New sections for controlling power state for each cell      // int Astatus = input.indexOf("A");      // int Bstatus = input.indexOf("B");      // int Cstatus = input.indexOf("C");      // int Dstatus = input.indexOf("D");      // if (Astatus == 1)      // {      //   digitalWrite(Aenable,1);      // }      // if (Astatus == 0)      // {      //   digitalWrite(Aenable,0);      // }        if (vaIndex != -1 && caIndex != -1 && vbIndex != -1 && cbIndex != -1 && vcIndex != -1 && ccIndex != -1 && vdIndex != -1 && cdIndex != -1)      // This was within if statement, no idea why:   caIndex > vaIndex && vbIndex > caIndex && cbIndex > vbIndex && vcIndex > cbIndex && ccIndex > vcIndex && vdIndex > ccIndex && cdIndex > vdIndex      {          // Extract values for all four boards        int voltage1 = input.substring(vaIndex + 2, caIndex).toInt();        int current1 = input.substring(caIndex + 2, vbIndex).toInt();        int voltage2 = input.substring(vbIndex + 2, cbIndex).toInt();        int current2 = input.substring(cbIndex + 2, vcIndex).toInt();        int voltage3 = input.substring(vcIndex + 2, ccIndex).toInt();        int current3 = input.substring(ccIndex + 2, vdIndex).toInt();        int voltage4 = input.substring(vdIndex + 2).toInt();        int current4 = input.substring(cdIndex + 2).toInt();        // Validate values for all four cells        if (voltage1 >= 0 && voltage1 <= 255 && current1 >= 60 && current1 <= 255 &&            voltage2 >= 0 && voltage2 <= 255 && current2 >= 60 && current2 <= 255 &&            voltage3 >= 0 && voltage3 <= 255 && current3 >= 60 && current3 <= 255 &&            voltage4 >= 0 && voltage4 <= 255 && current4 >= 60 && current4 <= 255) {            Serial.println("-------------------------------------");            // Cell 1          Serial.println("Cell 1:");          setVoltage(SLAVE\_ADDRESS\_1, voltage1);          delay(100);          readMuxVoltageBoard1(voltage1);          setCurrent(SLAVE\_ADDRESS\_1, current1);          delay(100);          readMuxCurrentBoard1(current1);          // Cell 2          Serial.println("\nCell 2:");          setVoltage(SLAVE\_ADDRESS\_2, voltage2);          delay(100);          readMuxVoltageBoard2(voltage2);          setCurrent(SLAVE\_ADDRESS\_2, current2);          delay(100);          readMuxCurrentBoard2(current2);                  // Cell 3          Serial.println("\nCell 3:");          setVoltage(SLAVE\_ADDRESS\_3, voltage3);          delay(100);          readMuxVoltageBoard3(voltage3);          setCurrent(SLAVE\_ADDRESS\_3, current3);          delay(100);          readMuxCurrentBoard3(current3);          // Cell 4          Serial.println("\nCell 4:");          setVoltage(SLAVE\_ADDRESS\_4, voltage4);          delay(100);          readMuxVoltageBoard4(voltage4);          setCurrent(SLAVE\_ADDRESS\_4, current4);          delay(100);          readMuxCurrentBoard4(current4);          Serial.println("\nEnter next command:");          Serial.println("-------------------------------------");        }        else        {          //  User input is outside range (limited)          Serial.println("Invalid values. Voltage & current must be within range");        }      }      //  else if for reset function, currently no reset function      else      {        Serial.println("Invalid command format. Use 'va[voltage1]ca[current1]vb[voltage2]cb[current2]vc[voltage3]cc[current3]vd[voltage4]cd[current4]' or ResetCom[Value].");      }    }  }  // Function to set Voltage  void setVoltage(uint8\_t slaveAddress, uint8\_t value) {    //  Frame 1    WIRE.beginTransmission(slaveAddress);    //  Frame 2    WIRE.write(0); // Choose Wiper 1 (Voltage control)    //  Frame 3    WIRE.write(value);    WIRE.endTransmission();  }  // Function to set Current  void setCurrent(uint8\_t slaveAddress, uint8\_t value) {    WIRE.beginTransmission(slaveAddress);    WIRE.write(128); // Choose Wiper 2 (Current control)    WIRE.write(value);    WIRE.endTransmission();  }  //  Mux functions are hard coded. which isnt pretty!  // Function to read temperature from Cell 1  void readTemperatureBoard1() {    digitalWrite(A3\_PIN, LOW);    digitalWrite(A2\_PIN, LOW);    digitalWrite(A1\_PIN, LOW);    digitalWrite(A0\_PIN, HIGH);    delay(100);    int adcValue = analogRead(DOUT\_PIN);    float voltage = adcValue \* (3.3 / 4095.0);    float temp = 1.25 \* ((voltage - V0C) / TC);    Serial.print(temp, 2);    Serial.println(" °C");  }  // Function to read temperature from Cell 2  void readTemperatureBoard2() {    digitalWrite(A3\_PIN, LOW);    digitalWrite(A2\_PIN, HIGH);    digitalWrite(A1\_PIN, LOW);    digitalWrite(A0\_PIN, HIGH);    delay(100);    int adcValue = analogRead(DOUT\_PIN);    float voltage = adcValue \* (3.3 / 4095.0);    float temp = 1.25 \* ((voltage - V0C) / TC);    Serial.print(temp, 2);    Serial.println(" °C");  }  // Function to read temperature from Cell 3  void readTemperatureBoard3() {    digitalWrite(A3\_PIN, HIGH);    digitalWrite(A2\_PIN, HIGH);    digitalWrite(A1\_PIN, LOW);    digitalWrite(A0\_PIN, LOW);    delay(100);    int adcValue = analogRead(DOUT\_PIN);    float voltage = adcValue \* (3.3 / 4095.0);    float temp = 1.25 \* ((voltage - V0C) / TC);    Serial.print(temp, 2);    Serial.println(" °C");  }  // Function to read temperature from Cell 4  void readTemperatureBoard4() {    digitalWrite(A3\_PIN, HIGH);    digitalWrite(A2\_PIN, LOW);    digitalWrite(A1\_PIN, LOW);    digitalWrite(A0\_PIN, HIGH);    delay(100);    int adcValue = analogRead(DOUT\_PIN);    float voltage = adcValue \* (3.3 / 4095.0);    float temp = 1.25 \* ((voltage - V0C) / TC);    Serial.print(temp, 2);    Serial.println(" °C");  }  // Function to read and display voltage from Cell 1 MUX  void readMuxVoltageBoard1(int dPotStep) {    digitalWrite(A3\_PIN, LOW);    digitalWrite(A2\_PIN, LOW);    digitalWrite(A1\_PIN, HIGH);    digitalWrite(A0\_PIN, LOW);    delay(100);    long totalAdcValue = 0;    for (int i = 0; i < SAMPLE\_COUNT; i++) {      totalAdcValue += analogRead(DOUT\_PIN);      delay(1);    }    int averagedAdcValue = totalAdcValue / SAMPLE\_COUNT;    float voltage = ((((averagedAdcValue \* (3.3 / 4095)) \* 2) / 0.65) - 0.05);    Serial.print("dPot. step: ");    Serial.print(dPotStep);    Serial.print("   |  ");    Serial.print(voltage, 2);    Serial.println(" V");  }  // Function to read and display voltage from Cell 2 MUX  void readMuxVoltageBoard2(int dPotStep) {    digitalWrite(A3\_PIN, LOW);    digitalWrite(A2\_PIN, HIGH);    digitalWrite(A1\_PIN, LOW);    digitalWrite(A0\_PIN, LOW);    delay(100);    long totalAdcValue = 0;    for (int i = 0; i < SAMPLE\_COUNT; i++) {      totalAdcValue += analogRead(DOUT\_PIN);      delay(1);    }    int averagedAdcValue = totalAdcValue / SAMPLE\_COUNT;    float voltage = ((((averagedAdcValue \* (3.3 / 4095)) \* 2) / 0.65) - 0.05);    Serial.print("dPot. step: ");    Serial.print(dPotStep);    Serial.print("   |  ");    Serial.print(voltage, 2);    Serial.println(" V");  }  // Function to read and display voltage from Cell 3 MUX  void readMuxVoltageBoard3(int dPotStep) {    digitalWrite(A3\_PIN, LOW);    digitalWrite(A2\_PIN, HIGH);    digitalWrite(A1\_PIN, HIGH);    digitalWrite(A0\_PIN, HIGH);    delay(100);    long totalAdcValue = 0;    for (int i = 0; i < SAMPLE\_COUNT; i++) {      totalAdcValue += analogRead(DOUT\_PIN);      delay(1);    }    int averagedAdcValue = totalAdcValue / SAMPLE\_COUNT;    float voltage = ((((averagedAdcValue \* (3.3 / 4095)) \* 2) / 0.65) - 0.05);    Serial.print("dPot. step: ");    Serial.print(dPotStep);    Serial.print("   |  ");    Serial.print(voltage, 2);    Serial.println(" V");  }  // Function to read and display voltage from Cell 4 MUX  void readMuxVoltageBoard4(int dPotStep) {    digitalWrite(A3\_PIN, HIGH);    digitalWrite(A2\_PIN, LOW);    digitalWrite(A1\_PIN, HIGH);    digitalWrite(A0\_PIN, LOW);    delay(100);    long totalAdcValue = 0;    for (int i = 0; i < SAMPLE\_COUNT; i++) {      totalAdcValue += analogRead(DOUT\_PIN);      delay(1);    }    int averagedAdcValue = totalAdcValue / SAMPLE\_COUNT;    float voltage = ((((averagedAdcValue \* (3.3 / 4095)) \* 2) / 0.65) - 0.05);    Serial.print("dPot. step: ");    Serial.print(dPotStep);    Serial.print("   |  ");    Serial.print(voltage, 2);    Serial.println(" V");  }  // Function to read and display current from Cell 1 MUX  void readMuxCurrentBoard1(int dPotStep) {    digitalWrite(A3\_PIN, LOW);    digitalWrite(A2\_PIN, LOW);    digitalWrite(A1\_PIN, HIGH);    digitalWrite(A0\_PIN, HIGH);    delay(100);    float currentSum = 0;    for (int i = 0; i < SAMPLE\_COUNT; i++) {      int adcValue = analogRead(DOUT\_PIN);      float current = ((((adcValue \* (3.3 / 4095)) \* 2) / 3.3) / 0.5);      currentSum += current;    }    float currentAverage = currentSum / SAMPLE\_COUNT;    Serial.print("dPot. step: ");    Serial.print(dPotStep);    Serial.print("    |  ");    Serial.print(currentAverage, 2);    Serial.print(" A  |   ");    readTemperatureBoard1();  }  // Function to read and display current from Cell 2 MUX  void readMuxCurrentBoard2(int dPotStep) {    digitalWrite(A3\_PIN, LOW);    digitalWrite(A2\_PIN, LOW);    digitalWrite(A1\_PIN, LOW);    digitalWrite(A0\_PIN, LOW);    delay(100);    float currentSum = 0;    for (int i = 0; i < SAMPLE\_COUNT; i++) {      int adcValue = analogRead(DOUT\_PIN);      float current = ((((adcValue \* (3.3 / 4095)) \* 2) / 3.3) / 0.5);      currentSum += current;    }    float currentAverage = currentSum / SAMPLE\_COUNT;    Serial.print("dPot. step: ");    Serial.print(dPotStep);    Serial.print("    |  ");    Serial.print(currentAverage, 2);    Serial.print(" A  |   ");    readTemperatureBoard2();  }  // Function to read and display current from Cell 3 MUX  void readMuxCurrentBoard3(int dPotStep) {    digitalWrite(A3\_PIN, LOW);    digitalWrite(A2\_PIN, HIGH);    digitalWrite(A1\_PIN, HIGH);    digitalWrite(A0\_PIN, LOW);    delay(100);    float currentSum = 0;    for (int i = 0; i < SAMPLE\_COUNT; i++) {      int adcValue = analogRead(DOUT\_PIN);      float current = ((((adcValue \* (3.3 / 4095)) \* 2) / 3.3) / 0.5);      currentSum += current;    }    float currentAverage = currentSum / SAMPLE\_COUNT;    Serial.print("dPot. step: ");    Serial.print(dPotStep);    Serial.print("    |  ");    Serial.print(currentAverage, 2);    Serial.print(" A  |   ");    readTemperatureBoard3();  }  // Function to read and display current from Cell 4 MUX  void readMuxCurrentBoard4(int dPotStep) {    digitalWrite(A3\_PIN, HIGH);    digitalWrite(A2\_PIN, LOW);    digitalWrite(A1\_PIN, HIGH);    digitalWrite(A0\_PIN, HIGH);    delay(100);    float currentSum = 0;    for (int i = 0; i < SAMPLE\_COUNT; i++) {      int adcValue = analogRead(DOUT\_PIN);      float current = ((((adcValue \* (3.3 / 4095)) \* 2) / 3.3) / 0.5);      currentSum += current;    }    float currentAverage = currentSum / SAMPLE\_COUNT;    Serial.print("dPot. step: ");    Serial.print(dPotStep);    Serial.print("    |  ");    Serial.print(currentAverage, 2);    Serial.print(" A  |   ");    readTemperatureBoard4();  } |

**I2C scanner**

|  |
| --- |
| // SPDX-FileCopyrightText: 2023 Carter Nelson for Adafruit Industries  //  // SPDX-License-Identifier: MIT  // --------------------------------------  // i2c\_scanner  //  // Modified from https://playground.arduino.cc/Main/I2cScanner/  // --------------------------------------  #include <Wire.h>  // Set I2C bus to use: Wire, Wire1, etc.  #define WIRE Wire  #define PIN\_SDA 48       // SDA connected to ESP32  #define PIN\_SCL 47       // SCL connected to ESP32  void setup() {    WIRE.setPins(PIN\_SDA, PIN\_SCL);    WIRE.begin();    Serial.begin(9600);    while (!Serial)       delay(10);    Serial.println("\nI2C Scanner");  }  void loop() {    byte error, address;    int nDevices;    Serial.println("Scanning...");    nDevices = 0;    for(address = 0; address < 127; address++ )    {      // The i2c\_scanner uses the return value of      // the Write.endTransmisstion to see if      // a device did acknowledge to the address.      WIRE.beginTransmission(address);      error = WIRE.endTransmission();      if (error == 0)      {        Serial.print("I2C device found at address 0x");        if (address<64)          Serial.print("0");        Serial.print(address,HEX);        Serial.println("  !");        nDevices++;      }      else if (error==4)      {        Serial.print("Unknown error at address 0x");        if (address<64)          Serial.print("0");        Serial.println(address,HEX);      }      delay(10);    }    if (nDevices == 0)      Serial.println("No I2C devices found\n");    else      Serial.println("done\n");    delay(5000);           // wait 5 seconds for next scan  } |

## Sims

|  |
| --- |
| LTspice simulation without implementing new voltage divider (current regulator) |