**SmartTree Electrical System Design Documentation**

Author: Francis Chen, Allen Tang, Kevin Tian

Co-PM, Electrical Lead

July 20th 2016

Table of Content

1. Introduction
   1. Usage Overview
   2. Block Diagram
   3. Physical Arrangement
   4. Point to Point Schematic
2. Hardware
   1. Solar Panel
   2. Maximum Power Point Tracker (MPPT)
   3. Battery and Battery Management System (BMS)
   4. DC- DC Converters (36-12V) / (36-5V)
   5. Inverter and USB Port
   6. Screen
3. Software
   1. Architecture
   2. Battery Management System
      1. Battery Management API
      2. Error Handling Software
   3. Screen
      1. Overview
      2. External Libraries
      3. Functions
4. Energy Generation
5. Installation
   1. Weather Proof and Insulation
      1. Solar Panel
      2. Battery box
   2. Grounding
   3. Lightning Protection
   4. Battery/Control box construction

Appendix

Appendix 1: Schematic

Appendix 2:

1. Introduction

*This brief introduction is intended for illustrating the high level design. Refer to the later sections for more detailed design for each individual part.*

* 1. Usage Overview

The SmartTree is a solar powered charging station that resembles the shape of a tree and provides a unique outdoor gathering place for students without worrying about their electronic devices running out of power. The *8 solar panels* will power a small off-grid system with *12 lead acid batteries* for energy storage. With proper voltage conversion, the system will output *2 AC outlets* and *4 USB ports*, for personal electronics like laptop, tablet, phones, etc., In addition, a screen will be mounted on the trunk to display the energy generation from the sun and promote the usage of renewable energy.

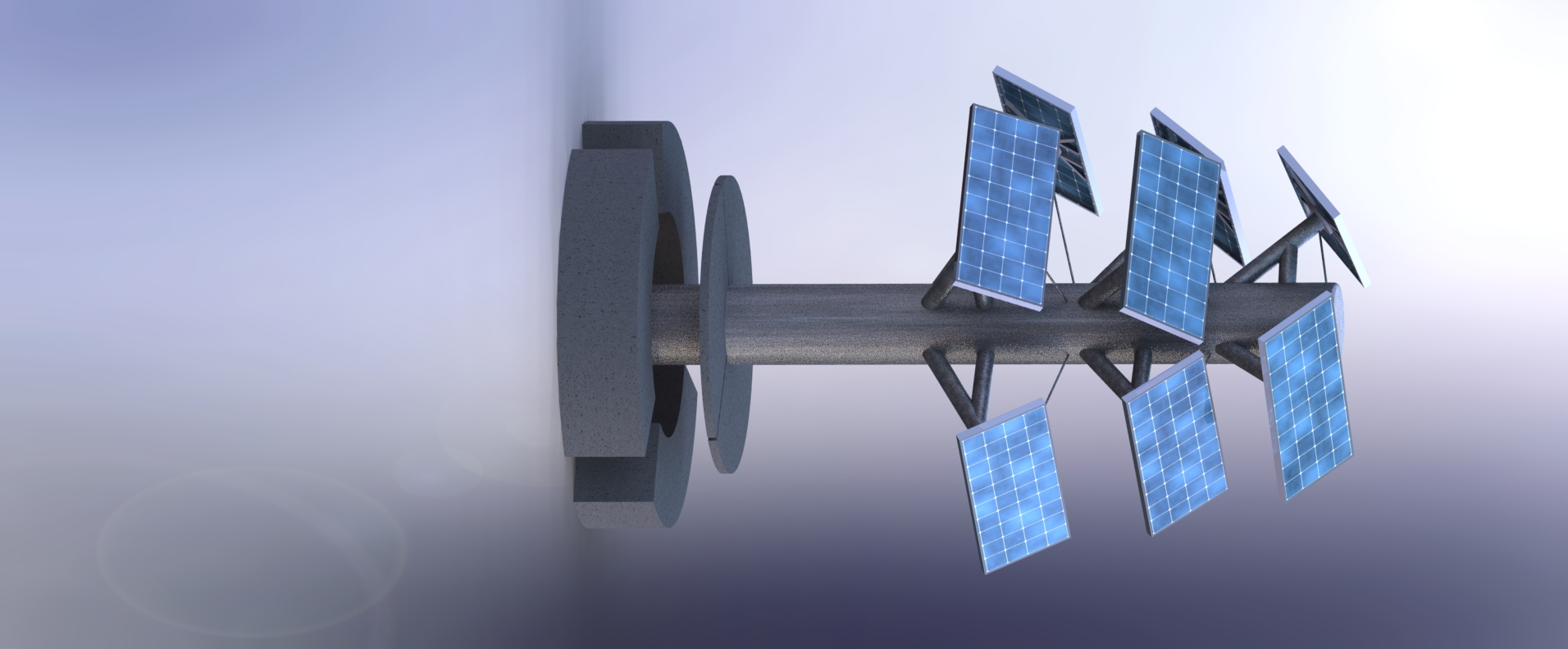


Figure 1.1 SmartTree CAD Model

* 1. Block Diagram

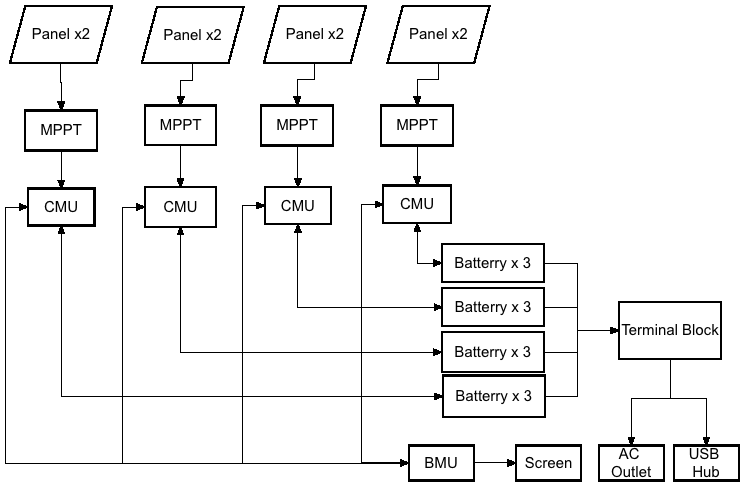


Figure 1.2 System Block Diagram

The overall system of the SmartTree consists of **8 solar panels,** each operating at **17V/75 W max power.** We paired up **every two of them** to form a subsystem, with **three serially connected 12V lead acid batteries** to store redundant energy and power the user output. Within every subsystem, a **maximum power point tracker (MPPT)** is implemented between the solar panels and the batteries to maximize the efficiency of the solar panel and regulate the voltage goes to the batteries. A **battery management system (BMS) consists of 4 CMUs and 1 BMU** is designed and implemented to monitor the state of the batteries, control the charging and discharging and displays energy savings on the screen. The four battery packs are all connected to a **36V terminal block** in parallel, which in turn powers **an inverter**, **a 36V-12V DC-DC converter** and **a 36V-5V DC-DC converter**. The inverter will provide the AC output, the 5V will power the USB ports, and the 12V will power the electronics within the system (i.e. BMS, Screen..etc.,).

* 1. Physical Arrangement



Figure 1.3 Physical Arrangements of Components

As shown in Figure 1.3, all the components except the solar panels, AC/USB outlets, and screen are housed within battery boxes. Solar panels are mounted at the end of the branches to collect sunlight. AC/USB outlets as well as the screen are mounted on the trunk, connecting with the battery boxes via wires running through the trunk. All the rest of the components, including battery, BMS, MPPT, inverter, DC-DC converters are all housed in five battery boxes, among which four houses batteries and one carries the control circuitry. Battery boxes are placed underground, underneath the table and benches. Individual components will be introduced in the next section.

* 1. Point to Point Schematic

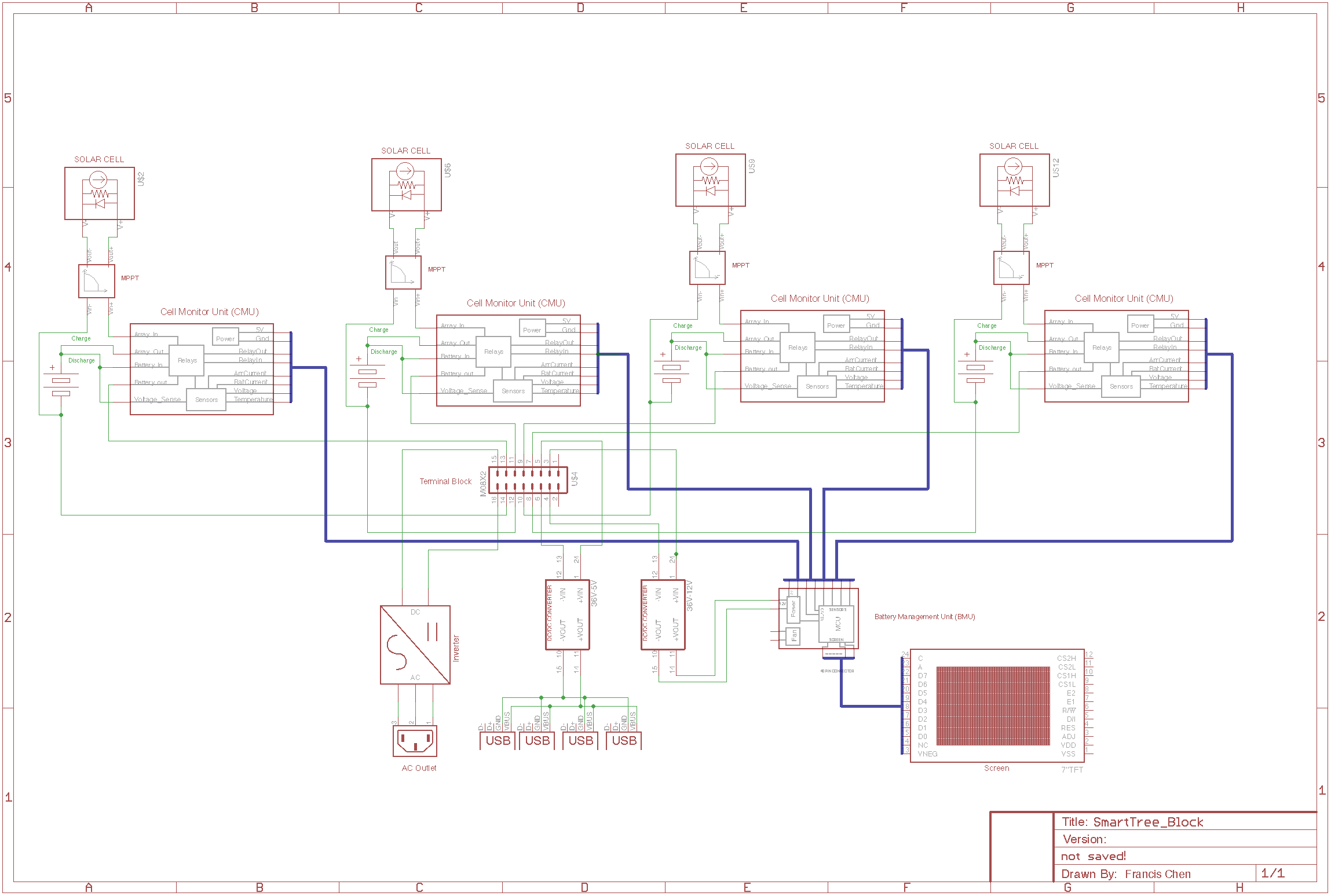


Figure 1.4 Point to Point Connections of SmartTree Electrical System

1. Hardware
   1. **Solar Panel**

We currently have 5 BP-275 solar panels with maximum power around 75W and 17V, and 3 VLS-85 solar panels, with maximum power around 85 W and 18V. Both models IV curve are shown below.

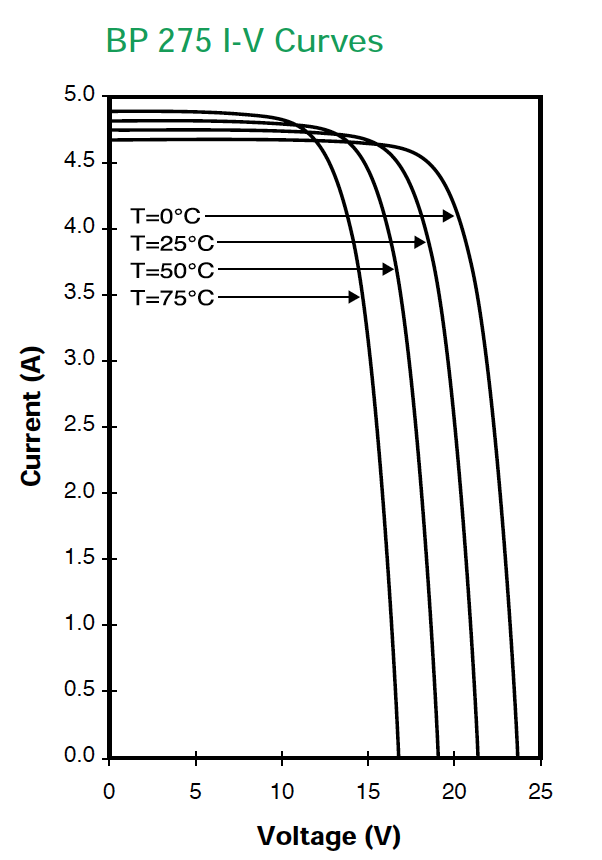
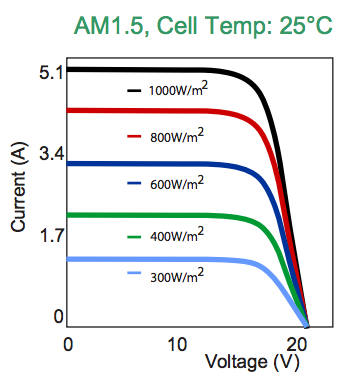


Figure 2.1 IV curves of solar panels (VLS-85W left, BP-275 right)

Note even though we have two different models of solar panels, the electrical and mechanical characteristics match rather well and we will be able to fit them into the system with minimal design changes.

* 1. **Maximum Power Point Tracker**

Our system right now implements a Genasun GVB-8-Pb-36V MPPT, which is designed for solar golf cart that runs on 36V lead acid batteries, fitting in our application perfectly.



This commercially designed MPPT meet the specification of our panel and batteries very well and is very easy to hook up, it has been tested out with the system.

Figure 2.2 Genasun MPPT

* 1. **Battery and Battery Management System**

As briefly discussed in the introduction, we have three serially connected 12V, 20Ah lead acid batteries for each subsystem, stacking up to 36V and we have the 4 battery packs from the 4 subsystems connected in parallel stacking up the current to around 8A (@0.1C).

Separated by the subsystem, we put every 3 batteries in a **battery box, resulting in 4 battery boxes** and the rest of the power electronics (inverter, DC-DC..etc.,) to a **central control boxes,** which accepts power input from all battery boxes. **Again, these 5 boxes are housed underneath the table of the SmartTree to prevent it from weathering conditions**. The interactions between these boxes are illustrated as below:

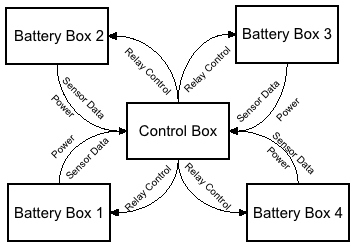


Figure 2.3 Battery Box Arrangements

We implemented a battery management system to monitor the power generation and the state of charge of the battery, which involves **4 Cell Monitoring Units (CMUs)** and **1 Battery Management Unit (BMU).**

**The CMU serves two functionalities:**

* ***Monitor state of the battery:*** The 4 CMUs located in the battery boxes are home designed PCBs that take the **voltage, input current, output current and temperature** measurement of the batteries and transmit the data to the BMU located in the control box.
* ***Control Charge and Discharge:*** The 4 CMUs accept control signals from BMU for two solid-state relays on the CMU that controls the charging and discharging process of the batteries.

**The BMU serves two functionalities:**

* ***Communicating with CMU*:** As mentioned above, the CMU constantly send back data to the BMU and the BMU is responsible for analyzing the voltage, current and temperature for each battery pack to decide if a fault condition has occurred and how to resolve by controlling the relays.
* ***Energy Calculation and Display*:** In addition, with the sensors data supplied by CMU, the BMU will integrate the data to get energy and power generation to display on a LCD screen for public awareness of renewable energy.

The schematic of the BMU and CMU are shown in the Appendix 1.1, essentially, BMU is a sensor aboard that has current/voltage/temperature sensors, and MOSFETs to drive relays. CMU is a microcontroller system (we’re using Arduino Due) for data processing.

* 1. **Inverter and USB Port**

We bought commercially available inverter for 36V system and USB port. Specifically, we’re using the NM300 36V DC to 110V AC inverter from Nimble and a 4 outputs Amazon Basics USB Port. These commercially available products will have proper fuses and protection, adding another layer of safety.





Figure 2.4 USB Ports and Inverter

* 1. **DC-DC Converters**

A RioRand DC-DC 36V to 5V 3A Step Down Converter converter is connected to the bus bar and used to power the USB hub. A Golf Cart Voltage Reducer Converter 36v to 12v DC-DC converter is connected to the bus bar and used to power the control electronics of the system, including the relays on the BMS and the screen.



Figure 2.5 DC-DC Converters

* 1. **Auxiliary Power**

In order to provide auxiliary power to micro-controllers in case of undesired system trip, auxiliary power is provided via an external 9 V battery pack.

* 1. **Screen**

To display the power generated by the system and raise awareness of renewable energy, a

7’’ TFT screen is implemented. In this case, a 7’’ TFT from SainSmart is used which comes with pre written library and a shield fitting to the Arduino Due footprint. 

Figure 2.6 SainSmart 7’’ TFT for Arduinod Due

3. Software

**3.2. Battery Management System (BMS)**

The BMS software running on the Arduino Due reads in sensor values from the four CMUs and monitors the state of the battery packs. Its primary functionality is to ensure safety and check failure conditions, which includes, over-temperature, over-current, over-voltage and under-voltage. The detailed software documentation below is to explain the monitoring and error handling functions of the software for future reference and debugging purposes.

3.2.1. Battery Management System API

Header File:bmucell.h

Implementation: bmucell.cpp

Dependencies <arduino.h>, "pins.h"

**Public Methods**

|  |
| --- |
| **static Cell::setup()**  Configures pins used by all Cells. Must be called before all methods except the constructor. |
| **Cell::Cell(unsigned int cellNumber)**  Constructs a Cell and configures pins used by the specific Cell indicated by cellNumber. The mapping of cellNumber to the required pins and the data acquisition procedure is implementation specific. |
| **void Cell::update()**  Updates all measurements for the specific Cell. Does not identify or handle error conditions. |
| **bool Cell::checkErrors()**  Identifies and handles error conditions for the specific Cell. This method should be called soon after **Cell::update()** is called, in order to use the most recent measurements.  The return value is true if an error condition is detected. |
| **void Cell::logData(Stream& stream)**  Logs the measurements (temperature, voltage, current in, and current out) to a specified Stream. |
| **void Cell::logErrors(Stream& stream, bool verbose)**  Logs the error condition to a specified Stream. If verbose is true, then the errors will be displayed in English; otherwise the error conditions will be logged hexidecimal form. A lookup table to determine error conditions is in Appendix n. |
| **double Cell::getTemperature()**  Gets the most recently read temperature, in degrees Celsius. |
| **double Cell::getVoltage()**  Gets the most recently read voltage, in volts. |
| **double Cell::getCurrentIn()**  Gets the most recently current entering the cell, in amperes. Note that this value is considered invalid when error conditions prevent charging, but in practice will be close to zero. |
| **double Cell::getCurrentOut()**  Gets the most recently current exiting the cell, in amperes. Note that this value is considered invalid when error conditions prevent discharging, but in practice will be close to zero. |

*Table 3.1 Public Methods for BMS*

*3.2.2 Error Handling Software*

The error handling code determines when the batteries are overloaded, overheated, or improperly charged, and takes corrective action to restore the system. The error handler ensures that the four measurements - temperature, input current, output current, and voltage - are within acceptable limits.

To reduce instability due to noise, the system makes multiple measurements before reacting. This is managed with an **error counter** for each variable. Every time a measurement is within the respective **error condition**, the error counter is incremented. Otherwise, the error counter is decremented until it reaches zero. When the error counter reaches an **error threshold** value (determined by frequency of measurement), the error flag is set and corrective action is taken.

To prevent rapid oscillation when values are close to the limit, the system uses **hysteresis** by requiring a stricter **recovery condition** to terminate the error. For the voltage and temperature errors only, a **recovery counter** is used in the same manner as the error counter to avoid spontaneous recovery due to noise.

For the current errors, a different approach is necessary because a current error will disconnect the path of the current, rendering future measurement impossible. To remedy this situation, the error handler will continually increment its recovery counter after the error has occurred. Once this counter reaches the **partial recovery threshold**, the system reconnects the current path, and the recovery counter continues to increment. Future cycles will measure the current, and if the value does not satisfy the recovery condition, the system will reset the counter and disconnect the circuit *immediately*. Otherwise, the cycle repeats until the counter reaches the **full recovery threshold**, which indicates that the recovery condition has been satisfied for a specified number of cycles. The error flag and recovery counter are then reset and operation continues as usual.

The following errors are recognized:

|  |  |  |  |
| --- | --- | --- | --- |
| **Error** | **Error Condition** | **Recovery Condition** | **Action** |
| High Temperature | T > 50°C | T < 45°C | Electrically isolate battery |
| High Voltage | V > 43.5 V | V < 43.0 V | Stop charging |
| Low Voltage | V < 36.0 V | V > 36.5 V | Stop discharging |
| High Current In | I > 6.6 A | I < 6.1 A | Stop charging |
| High Current Out | I > 6.6 A | I < 6.1 A | Stop discharging |

Table 3.2 Error Handling for BMS

In addition, there is an undercurrent input warning, where I < 1.0 A. This warninsg does not require a counter or a recovery value for hysteresis, since the warning takes no action except for displaying a warning on the screen.

**Error Lookup Table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| First Character | | | Second Character | | | | |
| Err. Code | Overcurrent Out | Undercurrent In (warning) | Err. Code | Temp. | Overvoltage | Undervoltage | Overcurrent In |
| 0 |  |  | 0 |  |  |  |  |
| 1 | ✓ |  | 1 | ✓ |  |  |  |
| 2 |  | ✓ | 2 |  | ✓ |  |  |
| 3 | ✓ | ✓ | 3 | ✓ | ✓ |  |  |
|  |  |  | 4 |  |  | ✓ |  |
|  |  |  | 5 | ✓ |  | ✓ |  |
|  |  |  | 6 |  | ✓ | ✓ |  |
|  |  |  | 7 | ✓ | ✓ | ✓ |  |
|  |  |  | 8 |  |  |  | ✓ |
|  |  |  | 9 | ✓ |  |  | ✓ |
|  |  |  | a |  | ✓ |  | ✓ |
|  |  |  | b | ✓ | ✓ |  | ✓ |
|  |  |  | c |  |  | ✓ | ✓ |
|  |  |  | b | ✓ |  | ✓ | ✓ |
|  |  |  | e |  | ✓ | ✓ | ✓ |
|  |  |  | f | ✓ | ✓ | ✓ | ✓ |

Table 3.3 Error Code generated

**3.3. Screen**

*3.3.1 Overview*

The Arduino code for the screen, “SmartTreeScreen.ino”, utilizes the pre-written libraries SPI, SdFat, UTFT, UTFT\_SdRaw, rtc\_clock, DueFlashStorage, efc, and flash\_efc. The additional libraries coded for this project are screen and TFT.

*3.3.2 External Libraries*

**Serial Peripheral Interface (SPI**)-a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances

**SdFat**-provides read/write access to FAT16/FAT32 file systems on SD/SDHC flash cards

**UTFT**- UTFT library for the TFT LCD screen

**UTFT\_SdRaw -** SD card library for TFT screen

**rtc\_clock**-real time clock for the microcontroller

**DueFlashStorage**-provides read/write access to the Arduino Due Flash Storage

**embedded flash controller (efc)**-

**flash\_efc**

*3.3.3 Functions*

**SDInit();**

Function: Initializes the SD card in the Arduino shield

Parameters: None

**LCDInit();**

Function: Initializes the LCD screen and loads the image stored in the SD card

Parameters: None

**RTCInit();**

Function: Initializes the RTC clock, setting the time and date.

Parameters: None

**UpdateBattery();**

Function: Updates the battery display icon on the screen corresponding to the amount of energy left in the storage system.

Parameters: None

**ErrorMessage(flag);**

Function:

Parameters:

Returns:

**UpdateEnergy(OldEnergy);**

Function: Updates the energy generated daily in the graph on the screen display

Parameters: OldEnergy: Daily energy generated by the system

**UpdateMemory(OldEnergy);**

Function: Saves the daily energy generated value in Arduino Due Flash Storage

Parameters:

OldEnergy: Daily energy generated by the system

**DisplayEnergy(PowerIn, OldEnergy, PowerOut);**

Function: Calculates the energy generated since the last interrupt and displays the cumulative energy generated in the current day, the current power input, the current power output, and the total energy generated since initialization.

Parameters:

1. PowerIn: Current power inputted into the battery system from the solar panels
2. OldEnergy: Daily energy generated by the system
3. PowerOut: Current power being withdrawn from the battery system

Returns: Updated daily energy generated by the system Usage: OldEnergy=DisplayEnergy(PowerIn, OldEnergy, PowerOut);

**datalog(PowerIn, PowerOut, OldEnergy, VoltageIn, CurrentIn);**

Function: Stores the system data into an SD file with a time and date stamp

Parameters:

1. PowerIn: Current power inputted into the battery system from the solar panels
2. PowerOut: Current power being withdrawn from the battery system
3. OldEnergy: Daily energy generated by the system
4. VoltageIn: Input voltage into the battery system
5. CurrentIn: Input current into the battery system

**GetDay();**

Function: Retrieves the current day from the RTC clock

Parameters: None

Returns: The current day of the month

Usage: NewDay=GetDay();

**ScreenInit();**

Function: Initializes all the components of the screen

Parameters: None

**ScreenUpdate();**

Function: Updates the battery level icon on the screen

Parameters: None

**DataRetrieve(VoltageIn, CurrentIn);**

Function: Calculates the power input and output and the energy generated, updates the display and stores the data into the SD card.

Parameters:

1. VoltageIn: Input voltage into the battery system
2. CurrentIn: Input current into the battery system

**DailyUpdate();**

Function: Updates the energy generated graph and resets the daily energy value at the start of a new day

Parameters: None

4. Energy Calculation

In order to calculate the energy consumption and generation, we make assumptions to the load condition as below:

- 4 USB charging port, operating @ 5V and 1.5A (assuming Dedicated Charging Port (DCP))

- 2 Laptop consuming 100 W of power each

- Inverter operates at 70% efficiency

- Assume usage only occurs during daylight

I. Assume Nominal Operating Condition:

**Therefore, at any given time, power input is greater than power output.**

II. Assume cloudy weather when solar cell is outputting 1/3 of its power.

**Assume 8 hours of daylight, the trees can be operated with 1/3 efficiency for roughly 5 days.**

III. Time to charge up the battery pack

Time needed to charge up battery = 3744Wh/(600-390)h = 17 hours.

**Assume 8 hours of day-light, the battery back only need roughly 2 days to charge back up.**

5. Installation

The installation of the electrical system requires the connection between the following components. (Picture explanation to come)

a. Solar panels to Battery Box MPPT Input - PV cable

b. Battery box battery output to control box switch box - Anderson

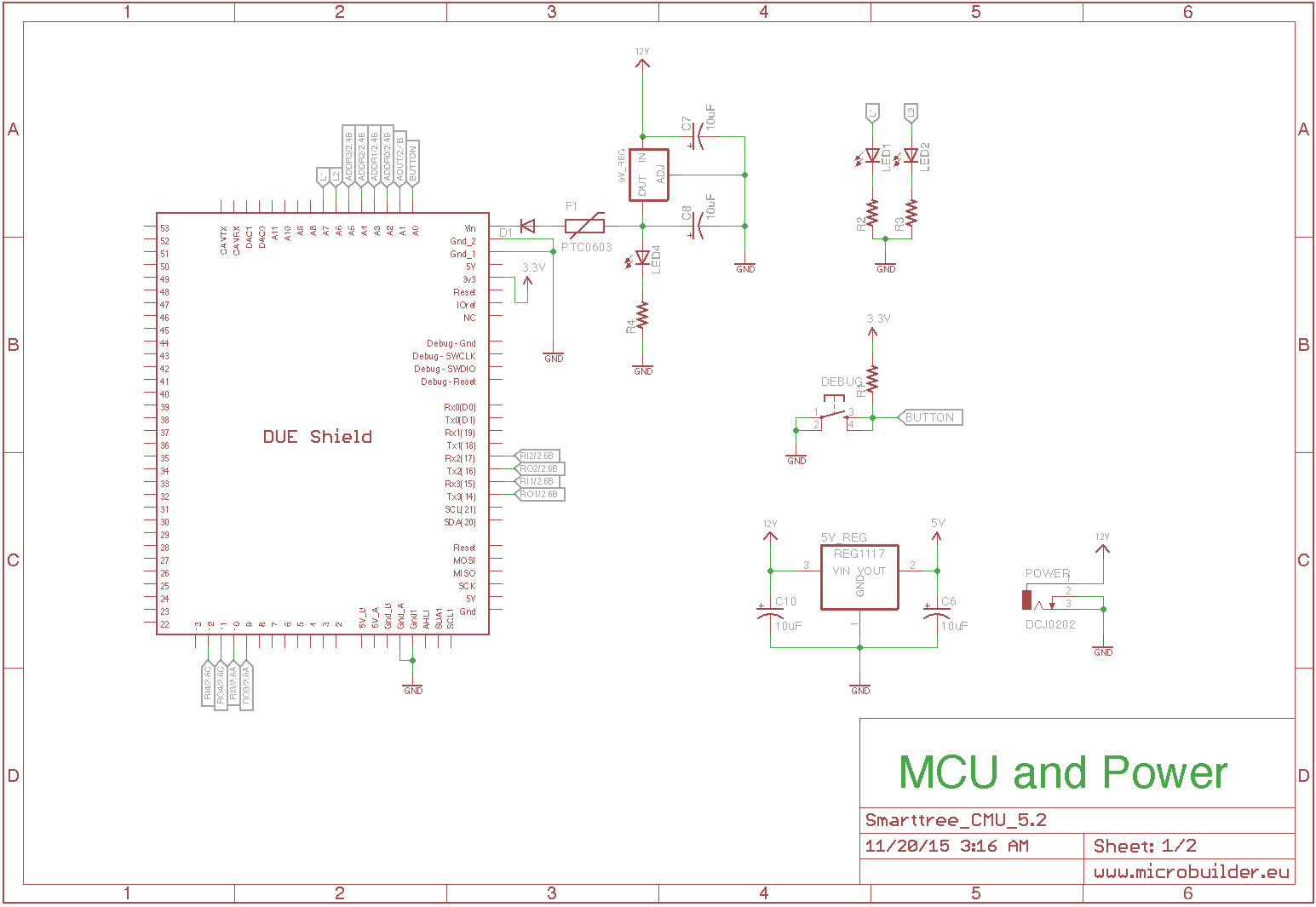
c. Battery box CMU signal output to control box BMU board – Ribbon cable

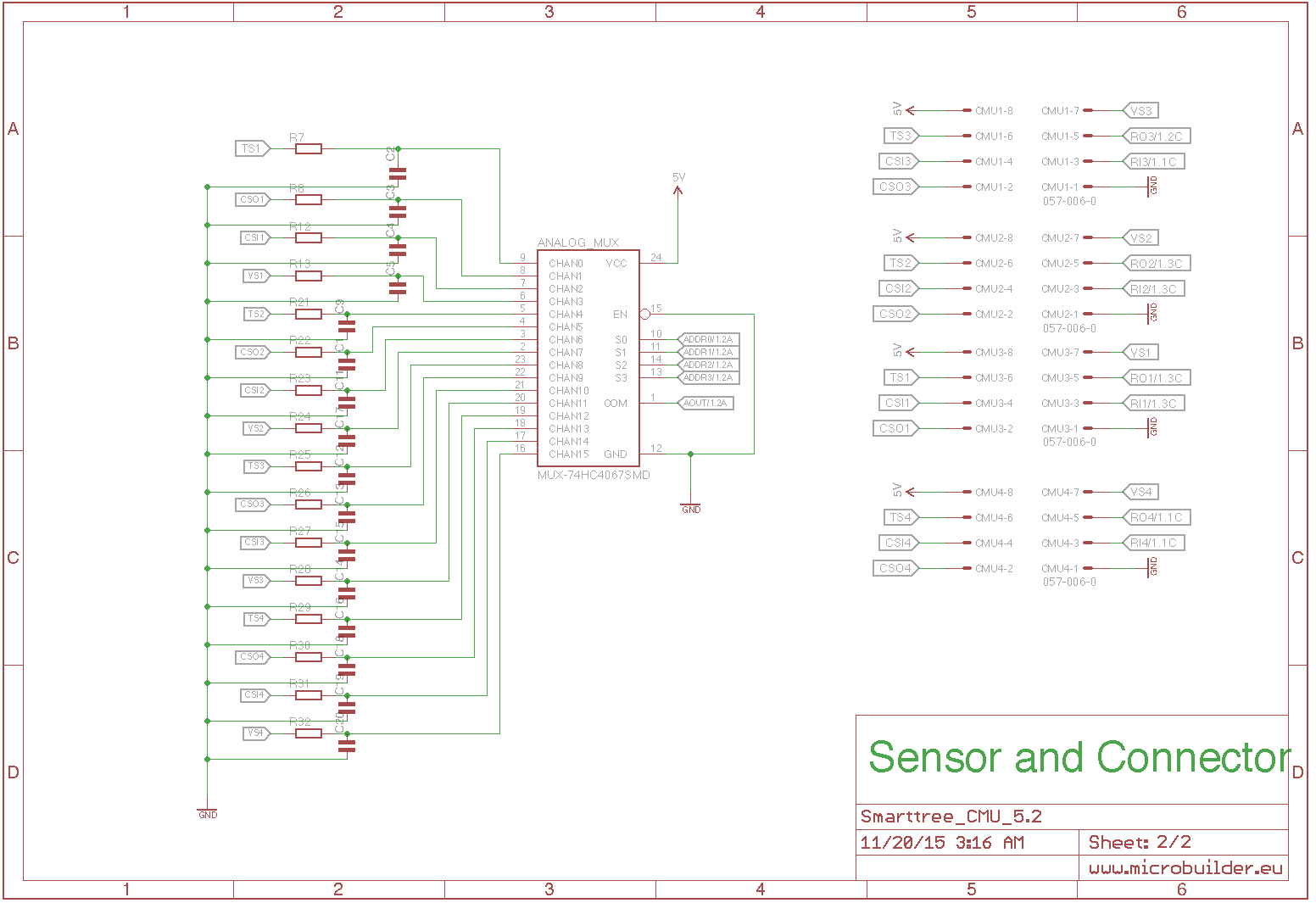
d. BMU board to Screen board – RS232 (DB9) cable

e. Control box 5v output to USB port on the trunk - USB extension cable.

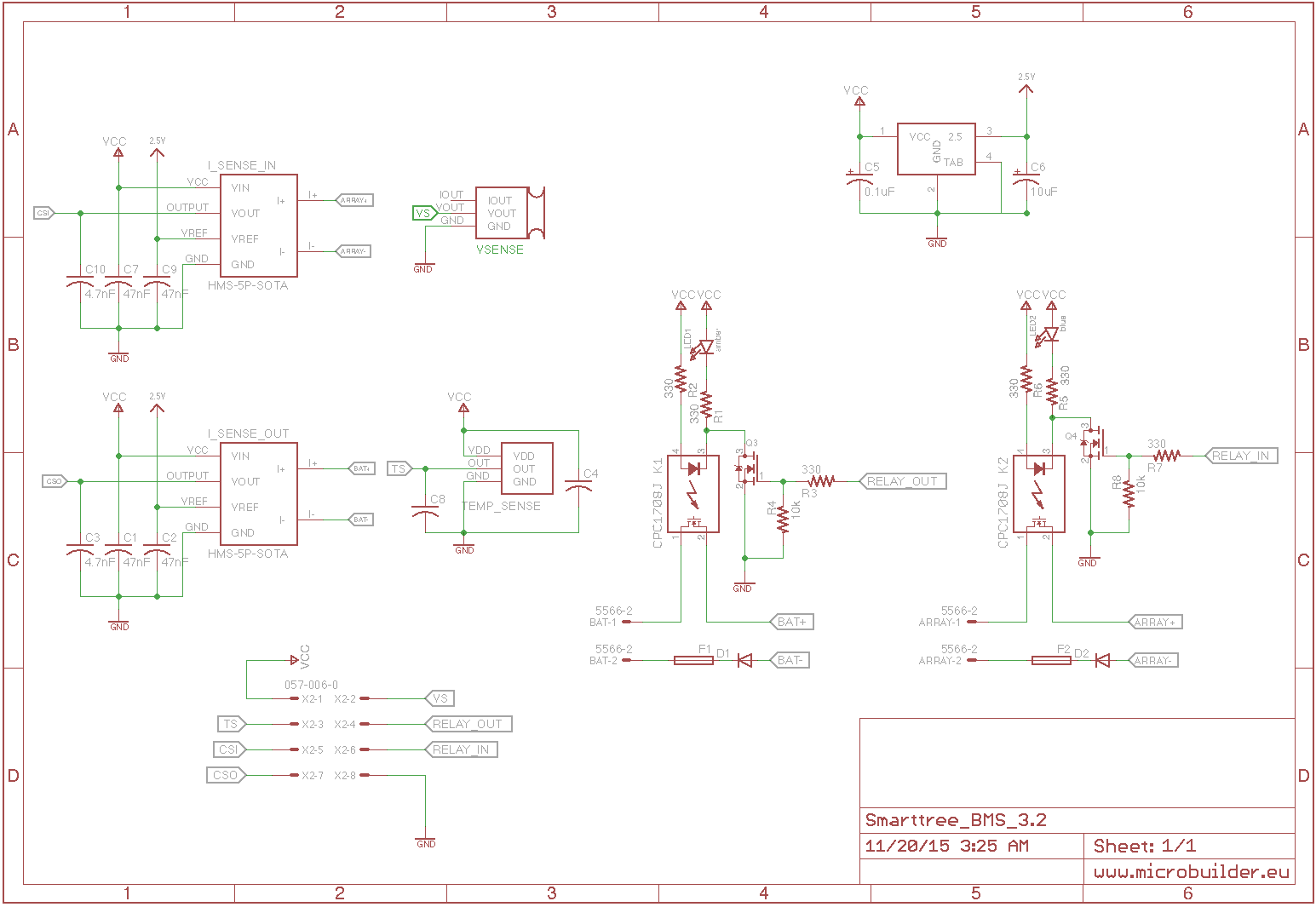
f. Control box inverter output to AC outlet - AC extension cable

Appendix 1: Schematic of Battery Management System (BMS)





**Schematic 1: Battery Management Unit (BMU)**



**Schematic 2: Cell Monitoring Unit (CMU)**