Solar generator

M.T. Konstapel

2021-11-25

Abstract

This document describes the design of an off-grid electrical power system. It is intended for a tiny house I've build. The design philosophy is that a discharged battery should be charged within a days worth of sunshine, also in late autumn and early spring. The build document is more intended as an inspiration for your own build rather than a step-by-step instruction guide. But it is detailed enough, should you want to build more or less the same system as I build.

1 The specifications

1.1 Energy storage

First, I have to figure out the specifications for the system. To start, it is important to know how much energy is needed every day. One regularly guest uses a medical device at night. It is important that there is enough stored energy to power this device. Then there are four 5 Watt lights which will be on for about six hours per day. And finally the power consumption of the solar generator itself. That last number is not known yet, but I already have an idea what inverter I want to use. So lets use the power consumption of that inverter:

Device	**Watt**	**Hours/day**	**Wh/day**
Medical device	15	8	120
Lights	20	6	120
Solar generator	8	24	192
		Total	**432**

Table 1: energy consumption per day.

Just to be on the safe side and also because it must be possible to charge a phone or use a laptop, I use a factor of 1.5 to calculate the needed energy storage capacity: $432 \cdot 1.5 = 648Wh$.

1.2 Energy harvesting

The energy that is needed every day must be the same as the energy harvested by the solar panel. Otherwise, the next day there will be no energy left in the battery. The solar panel must therefore be able to generate 648Wh of energy every day, preferable during the whole holiday season.

And therein lies the challenge: table 2 shows the hours of daylight per day for each month. January has only half the hours per day as July. And on top of that, the angle of the sun is lower in the winter. The amount of energy per m^2 is significantly lower in winter compared to summer. And not every day is sunny, which makes the problem even more difficult.

Month	**Hours of daylight**
January	8
February	10
March	12
April	14
May	16
June	17
July	16.5
August	14.5
September	12.5
October	10.5
November	8.5
December	7.5

Table 2: hours of daylight at 53 degrees latitude.

A compromise has to be made. Lets design a system that can generate enough energy to charge the battery on a sunny day from March to October, which is the normal holiday season in the Netherlands. First, we need to know the amount of energy we can expect from the sun per m^2 in every month. Than we take the month with the least amount of energy per m^2 as base for further calculations.

Luckily, there is a good data source on the internet[1] which has data collected over a 22 year period. Table 3 shows the numbers for the north of the Netherlands for a panel facing south with various panel angles. October seems to be the month with the lowest energy per m^2 per day. The optimal summer angle of 52 ° is, with $2.18kWh/m^2/day$, the worst performing angle for October from the three different angles.

A typical solar panel has an efficiency of about 20%. So it can harvest $2.18kWh/m^2/day \cdot 20\% = 436Wh/m^2/day$. For the total amount energy of $648Wh/day$ we need a panel of $\frac{648Wh/day}{436Wh/m^2/day} = 1.5m^2$.

Solar panels are typically specified in Watts, measured at an irradiance of $1000W/m^2$. That means that a solar panel with an efficiency of 20% and an area of $1.5m^2$ has a defined specified output power of $1000W/m^2 \cdot 20\% \cdot 1.5m^2 = 300W$. For the project I need a solar panel with a specified output power of at least 300W.

1.3 Emergency energy

In the Netherlands, it can be cloudy for days. In summer, this is not a big deal. As the system is designed to work during the darker October days, there should still be enough light to charge the battery. But later in the season, it

[1] http://www.solarelectricityhandbook.com/solar-irradiance.html

Month	**22 ° angle** **optimal winter** $kWh/m^2/day$	**37 ° angle** **optimal year round** $kWh/m^2/day$	**52 ° angle** **optimal summer** $kWh/m^2/day$
January	1.18	1.16	1.08
February	2.08	2.09	2.00
March	2.92	3.07	3.07
April	3.87	4.24	4.43
May	4.28	4.83	5.25
June	3.98	4.54	5.03
July	4.02	4.55	5.01
August	3.94	4.41	4.67
September	3.08	3.28	3.33
October	2.21	2.25	2.18
November	1.38	1.37	1.28
December	0.97	0.94	0.87

Table 3: averge solar insolation figures

can become a problem. That is why a backup is needed. With a mains powered battery charger, the battery can be charged using a petrol or diesel powered generator.

1.4 Outputs

As almost all devices work on 230V mains, it makes sense to install an inverter to generate 230V mains power from the battery. For normal use, not much power is needed, so an inverter with an output power of 300 Watts should be enough. But to make the solar generator more versatile I choose to install a 1000W inverter. This way, I can occasionally use the solar generator for powering power tools or use it as an emergency generator, powering my fridge and freezer in case of a mains power outage.

Besides a mains outlet, I want to install a 12VDC socket and two 5VDC USB sockets. This makes the solar generator even more versatile.

1.5 Portable

The system should be portable so it can also be used outside the tiny house it is designed for. Alternative uses are, for example, as a portable generator for powering an irrigation pump or as an emergency generator in case of a mains power outage.

1.6 Summary

We now know the main specifications for the solar generator. The next step is to design the circuit using the appropriate components.

- 300W solar panel
- 648Wh battery

- Mains emergency charger
- 1000W inverter for mains output
- 12V output
- 5V USB output

2 The design

2.1 Battery

Because I specified a 12V output, it is obvious to use a 12V battery. This way the 12V output can be directly connected to the battery without the need for a DC/DC converter.

Traditionally, off-grid solar uses lead acid batteries, because that's all that was available. But nowadays, lithium batteries are available for reasonable prices. The benefits are plentiful: they are lightweight, last a long time and can withstand a broad temperature range without deteriorating. That last feature is important as the summers in the Netherlands are getting hotter. 35 degrees Celsius is not uncommon. And at high temperatures lead acid batteries deteriorate fast. It can shorten the lifetime by as much as 50%! Lithium batteries do not have this disadvantage and are the better choice, even when they are more expensive. In the long run, it is probably cheaper to use a lithium battery.

The battery I choose is a Liontron 12.8V 55Ah (704Wh) LiFePO4 battery.

2.2 Solar charger

There are two types of solar chargers: PWM chargers and MPPT charger. The first type is a basic and therefor crude charger. It has a low efficiency wasting a lot of solar power. But it is very cheap to produce, so still popular in some applications. And than there is the MPPT charger or **M**aximum **P**ower **P**oint **T**racker charger. It can extract every Watt from the solar panel by constantly choosing the optimum power point of the solar panel. This is a more sophisticated process, but it comes at a cost: it is more expensive to produce than a PWM charger. But nowadays almost anyone with at least two brain cells uses this type of solar charger. I've got three, so I choose wisely.

The chosen battery can be charged with a current of up to 100A, but the recommended maximum charge current is 25A. I can choose between 20A solar chargers and 30A solar chargers. To be on the safe side, I choose a solar charger with a maximum charge current of 20A. The battery has a nominal voltage of 12.8V, so the maximum charge power is $20A \cdot 12.8V = 256W$. I specified a solar panel of 300W, which is higher than the maximum charge power, which seems wasteful and potentially can cause problems, right? Wrong. Solar panels are specified at an irradiance of $1000W/m^2$. In the Netherlands at least, that's not a realistic figure. The 300W is never reached, so a solar charger with a maximum charge power of 256W is more than enough.

The solar charger I choose is an EPEVER XTRA2210N-XDS2 charger.

2.3 Mains emergency charger

The mains charger has to charge the battery as fast as possible, but still within the recommended maximum charge current of the battery, which is 25A. I can choose between a 20A and a 30A charger model. So is has to be the 20A model.

The mains emergency charger I choose is a Xenteq LBC 512-20S charger.

2.4 Inverter

The inverter has to be able to generate 1000W of power at 230V from a 12V source. There is a lot to choose from. So let's narrow the search a bit. I want the inverter to produce a pure sine wave rather than a modified sine wave. A lot of loads will work on a modified sine wave, but not all. And I want to power them all! It also has to be reliable (obvious), so no cheap Chinese boxes.

The inverter I choose is a Xenteq PPI 1000-212C.

2.5 Solar panel

The solar charger can handle a maximum input power of 390W and a maximum input voltage of 92V. I specified a solar panel of at least 300W so it has to be a panel with a specified output power of between 300W and 390W. There are a lot of solar panels out there, so choosing one was a matter of random picking than. From a reputable source, of course.

The solar panel I choose is a JA Solar JAM60S10/345.

2.6 Controls

It is important that the solar generator can be switched off, as the inverter has a relative high power consumption of 8W, even when there are no loads attached to it. In winter, when there is little sun, there could be days when the power consumption of the solar generator itself is higher than the power harvested from the solar panel. The solar charger itself must never be switched of, as I want the battery to be charged at all times.

The inverter can be switched of via a control port. A special device (Xenteq PPR-2) attached to this port can switch on and off the inverter remotely. A DTSP switch mounted on the front panel of the solar generator is connected to the PPR-2. The solar generator can now be switched off by the user.

This same switch also switches off the 5V USB charge ports, as these also use some power when not in use.

2.7 Protection

The solar charger can detect a low battery voltage. When this occurs, the LOAD output of the charger is switched off. By connecting the LOAD output to the PPR-2, the inverter is switched off when the battery is almost discharged, therefor protecting the battery from a deep discharge.

3 Programming and monitoring

The designed system can be used just as is, but in order to protect the battery from over-discharging it is advisable to build a proper protection circuit. The principle is very simple: determine the amount of charge in the battery and when the battery is almost, but not fully discharged the load has to be switched off. Easy, right?

3.1 Battery

The hardware design depends on the solar charger to detect a low battery voltage and than switching of the inverter to protect the battery. The voltage of a *lead acid* battery is a good measure of the **S**tate **o**f **C**harge (SoC). And when combined with the load of the battery it can reliably be used by a protection circuit to switch of the battery. If the load is connected to the LOAD terminals of the charger, the solar charger knows both the battery voltage and the battery current. But the inverter is connected directly to the battery, so the current cannot be measured by the solar charger. But even if the charger knows the battery current as well as the battery voltage, it cannot reliably calculate the SoC of a *lithium battery*, as the voltage of the battery is not a measure of the SoC. We need some other way to determine the SoC of the battery.

The battery has a build in Bluetooth LE server. A special app can be installed on a smartphone to read the SoC of the battery. The app is nice for a normal user, but useless for a stand alone protection circuit. Unfortunately, the manufacturer does not provide any documentation regarding the protocol used. Luckily the makers community do shares their knowledge.

The battery uses a generic communication protocol, used on many Chinese BMS boards. It communicates over a serial bus connected to the micro controller of the BMS. The Liontron battery has a serial to Bluetooth LE converter connected to it. That makes it a two part problem: first I have to know the serial protocol and than I have to figure out the protocol used by the BLE module (Figure 1).

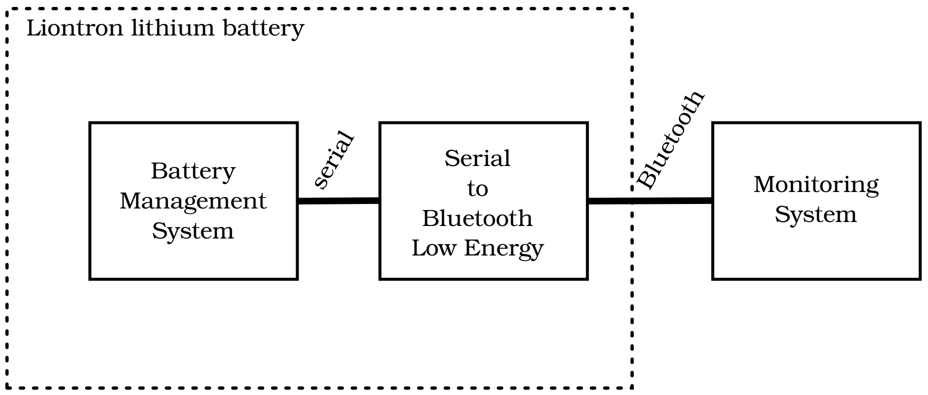


Figure 1: two protocols to figure out

3.1.1 Serial protocol of the BMS

The serial protocol used is well described on the website of a German guy named Jake[2]. Every packet begins with a **start byte** (0xDD) and ends with a **stop byte** (0x77). The byte after the start byte is the **command byte** which can either be 0xA5 (request to read) or 0x5A (request to write), than two bytes for the ID of the data we want to read or write. And last a two byte checksum (Table 4). For example to read the hardware info send: DD A5 05 00 FF FB 77.

pos	size	type	value
0	1	start byte	0xDD
1	1	command	0xA5 (read) 0x5A (write)
2	2	ID	0x0300 (BasicInfo) 0x0400 (CellVoltages) 0x0500 (HardwareInfo)
4	2	checksum	0xFFFD (BasicInfo) 0xFFFC (CellVoltages) 0xFFFB (HardwareInfo)
6	1	stop byte	0x77

Table 4: request packet

The BMS controller response by sending back a packet. Table 5 shows the three different packets which can be received. The first byte is always 0xDD and the second and third bytes echo the command we send to the BMS. Than one byte with the length of the packet followed by the data itself. Than a two byte checksum and the stop byte, which is always 0x77.

Calculating the cecksum is basically taking the sum over every relevant byte and then subtract that from 2^{16} (65536). The result should match the checksum in the packet.

Now lets implement this in Python:

```
def verify(packet):
    data = packet[2:-3]
    check = packet[-3:-1]
```

Here **data** contains all bytes that go into the checksum calculation and **check** is the 2 byte checksum of the current package. The last bit is always 0x77 (the stop bit), so cut it off.

```
crc=0x10000
for i in data:
    crc = crc-int(i)
crc_b = crc.to_bytes(2, byteorder='big')
return check == crc_b
```

It returns **True** if both the calculated and received checksums match.

[2]https://blog.ja-ke.tech/2020/02/07/ltt-power-bms-chinese-protocol.html

Basic Info			
pos	**size**	**type**	**value**
0	1	start byte	0xDD
1	2	ID	0x0300
3	1	length	number of data bytes
4	2	voltage	$raw * 0.01$ V
6	2	current	$raw * 0.01$ A
8	2	remain_cap	$raw * 0.01$ Ah
10	2	typ_cap	$raw * 0.01$ Ah
12	2	cycles	counts
14	2	prod_date	date
16	4	balance_status	32 bit flags
20	2	prot_status	see table 6
22	1	software_version	version
23	1	remain_cap_percent	%
24	1	fet_status	see table 6
25	1	cell_count	number of cells
26	1	ntc_count	number of ntcs
27	2	temp	$(raw * 0.1) - 273.1$ °C
...	2	repeat for every ntc	
4+length	2	checksum	calculate
last	1	stop byte	0x77

Cell Voltages			
pos	**size**	**type**	**value**
0	1	start byte	0xDD
1	2	ID	0x0400
3	1	length	number of data bytes
4	2	voltage	cell voltage
...	2	repeat for every cell	
4+length	2	checksum	calculate
last	1	stop byte	0x77

Hardware Info			
pos	**size**	**type**	**value**
0	1	start byte	0xDD
1	2	ID	0x0500
3	1	length	number of data bytes
4	length	version	depends on BMS
4+length	2	checksum	calculate
last	1	stop byte	0x77

Table 5: response packets

3.1.2 Bluetooth communication

Now we know the protocol to talk to the BMS, we need to know how to talk to the Bluetooth LE transceiver build inside the battery. Again, the makers

Protection List (prot_status)			
pos	**size**	**type**	**value**
0	3b	reserved	b3
0:3	1b	fet_lock	bit
0:4	1b	ic_error	bit
0:5	1b	ocp_short	bit
0:6	1b	ocp_discharge	bit
0:7	1b	ocp_charge	bit
1	1b	utp_discharge	bit
1:1	1b	otp_discharge	bit
1:2	1b	utp_charge	bit
1:3	1b	otp_charge	bit
1:4	1b	uvp_pack	bit
1:5	1b	ovp_pack	bit
1:6	1b	uvp_cell	bit
1:74	1b	ovp_cell	bit

Fet Bits (fet_status)			
pos	**size**	**type**	**value**
0	6b	reserved	b6
0:6	6b	discharge	bit
0:7	6b	charge	bit

Table 6: extra bits

community was very helpful. This time, a German chap who calls himself *sw-home*[3] helped me out. He published some Python code showcasing how to implement Bluetooth communication with a Liontron battery on a Raspberry Pi. The code uses a Python module called *gatt*.

It is very straightforward: first make sure the computer trusts the Bluetooth module inside the battery. This can be done with the Linux utility *bluetoothctl*.

```
pi@rpi:~>bluetoothctl
Agent registered
[bluetooth]# power on
Changing power on succeeded
[bluetooth]# scan on
[CHG] Controller AB:CD:EF:12:34:56 Discovering: yes
[NEW] Device 12:34:56:78:90:AB (the battery)
[bluetooth]# trust 12:34:56:78:90:AB
[bluetooth]# exit
pi@rpi:~>
```

Than we can connect to the device and start the event loop with the following Python code:

```
import gatt

manager = gatt.DeviceManager(adapter_name='hci0')

device = BluetoothDevice(mac_address='12:34:56:78:90:AB', manager=manager)
device.connect()
```

[3]https://github.com/sw-home/FHEM-BluetoothSmartBMS

```
    manager.run()
```

If all went according to plan, we are connected to the battery. If, for any chance, you have opened the battery you can see that the connected-LED is now on. The next step is to start the event loop and make sure that all the events are doing something useful. The gatt module takes care of everything, so we don't have to worry about anything.

```
class BluetoothDevice(gatt.Device):
    def connect_succeeded(self):
        super().connect_succeeded()
        print("[%s] Connected" % (self.mac_address))

    def services_resolved(self):
        super().services_resolved()

        device_information_service = next(
            s for s in self.services
            if s.uuid == '0000ff00-0000-1000-8000-00805f9b34fb')

        self.bms_read_characteristic = next(
            c for c in device_information_service.characteristics
            if c.uuid == '0000ff01-0000-1000-8000-00805f9b34fb')

        self.bms_write_characteristic = next(
            c for c in device_information_service.characteristics
            if c.uuid == '0000ff02-0000-1000-8000-00805f9b34fb')

        print("BMS found")
        self.bms_read_characteristic.enable_notifications()

    def characteristic_enable_notifications_succeeded(self, characteristic):
        super().characteristic_enable_notifications_succeeded(characteristic)
        print("BMS request generic data")
        self.bms_write_characteristic.write_value(
                                bytes([0xDD,0xA5,0x03,0x00,0xFF,0xFD,0x77]));

    def characteristic_value_updated(self, characteristic, value):
        print("BMS answering")
```

For now, the most important function is *services_resolved()*. Here, the Bluetooth LE characteristics for the battery are defined. These are the values we have to use in order to communicate with the device. After this it is only a matter of calling *bms_write_characteristic.write_value()* with the commands for the serial communication, described in the *Serial protocol of the BMS* section and waiting for a response, which triggers a *characteristic_value_updated()* event. In this function we can read the serial bytes from the BMS.

The example code is only a small part of the program. The whole source code is available from my website.

3.2 Solar charger

As the solar charger is made for lead acid batteries, it cannot be used with lithium batteries as-is. It has to be programmed to work properly. From the manufacturers website, a special program can be downloaded to do this. It is only available for Windows, so that's a bummer! But it has to be done. After that, we can connect the solar charger to a proper Linux-based system an read all the registers via the RS-485 port. It is also possible to switch the LOAD output on and off. And that's important, as this gives us the opportunity to switch the system on and off without building a special piece of hardware for this purpose.

3.2.1 Modbus protocol

The solar charger communicates via the Modbus protocol. A very ancient, but still widely used protocol. The physical interface on the charger is a RS-485 serial bus. The manufacturer sells a special RS-485 to USB converter for the solar charger. It is not supported on Linux by default, but luckily, a driver is available[4]. A detailed description about compiling and installing this driver can be found in the source code directory of this project.

After installing this driver, Linux detects the cable without any problems and it can be used together with the MinimalModbus module[5] for Python.

A complete description of all the Modbus registers of the solar charger is included in the archive of this project.

3.3 Mains emergency charger

The mains charger has to be set to lithium. See the manual for more information.

3.4 Stitching it all together

The main reason to go to all this effort is to be able to switch off the system when the battery is nearly empty. This prevents damaging the battery.

Via the Bluetooth connection we can detect if the level of the battery is below 5%. The LOAD output of the solar charger is than switched off via the Modbus interface. Because the LOAD output is connected to the PPR-2 (the controller for the inverter), the inverter is also switched off. At the same time, the 5V USB ports and the 12V output are also powered down. This prevents the battery from discharging to much. The battery is still connected to both the solar charger and the mains charger. The battery can therefore still be charged. If the charge is back up to 10%, the LOAD is switched on again and the system is back in service.

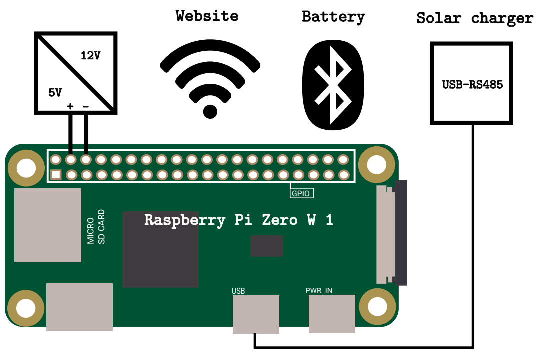
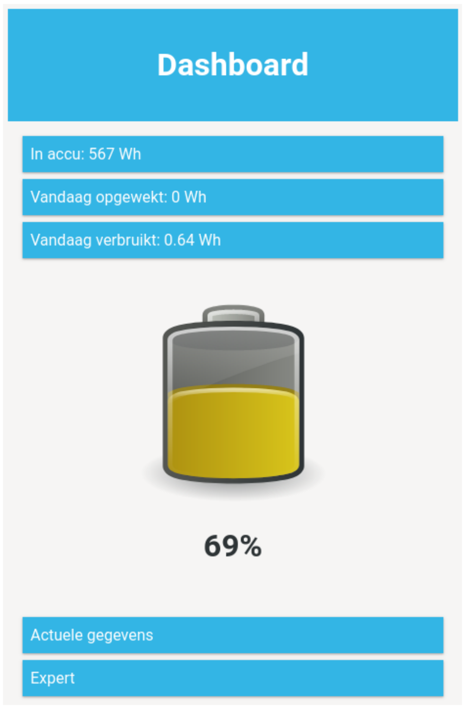


Figure 2: Raspberry pi controller

[4]https://github.com/kasbert/epsolar-tracer/
[5]https://minimalmodbus.readthedocs.io/en/master/readme.html

All the communication with the Bluetooth battery, the solar chargers' modbus and the WiFi is done with a Raspberry Pi Zero 1 W (see figure 2). This is probably a bit of an overkill as I am sure it can be done with an ESP32. But I am familiar with the Raspberry Pi and that makes the software development a lot easier and faster. The software is a bundle of Python utilities, for each task a separate one. All according to the Unix tradition. It is hold together by a Bash script. Every minute or so, the registers from the battery and the solar charger are read and interpreted. The results are saved in JSON files. These JSON files are read by another program, which extracts the information and appends this to a csv file. At the same time, the most important information is published on the website. The generated csv files are used to plot some nice diagrams for the website using *gnuplot*. And every day at midnight all the data from that day is analyzed and the results are appended to another csv file, which hold track of all the daily data. This data can be downloaded from the website for further analysis.



Dashboard

In accu: 567 Wh

Vandaag opgewekt: 0 Wh

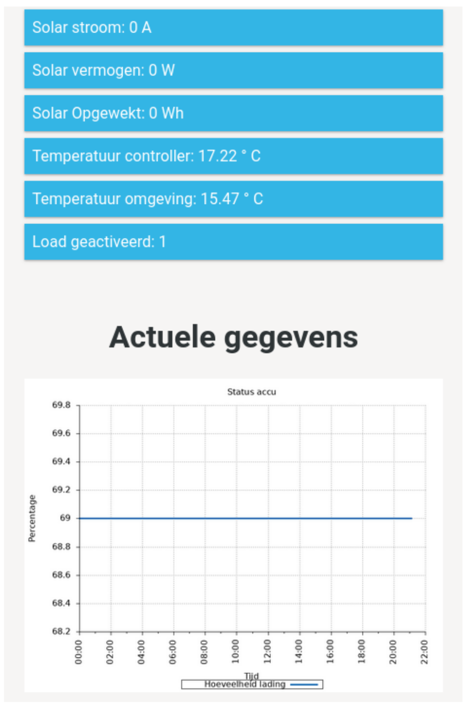
Vandaag verbruikt: 0.64 Wh

69%

Actuele gegevens

Expert

(a) Main page



Solar stroom: 0 A

Solar vermogen: 0 W

Solar Opgewekt: 0 Wh

Temperatuur controller: 17.22 ° C

Temperatuur omgeving: 15.47 ° C

Load geactiveerd: 1

Actuele gegevens

(b) Expert page

Figure 3: website solar generator

4 Construction

4.1 Housing

I want the solar generator to be portable. This way I can use it in the tiny house as well as for other applications. A flight cage is perfectly suited for the job. I modified a second hand flight case so it can accommodate all the parts needed. Figure 4 shows the already modified flight case.

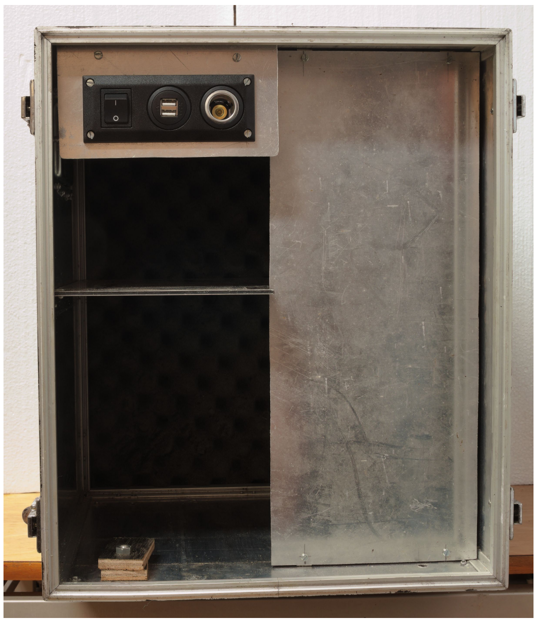


Figure 4: modified flight case

4.2 Mounting the battery

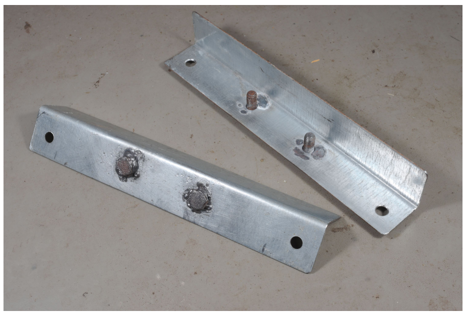
The lithium battery comes without mounting holes or other means to secure it in place. There are, however, four holes on the top cover of the battery.

I fabricated two mounting brackets and a mounting plate as can be seen in figure 5.

The brackets have pins which fits in the hole on the top of the battery. See figure 6.

After placing the battery on the mounting plate, the brackets are bolted onto the mounting plate with four long M8 bolts. See figure 7.

The complete assembly can be placed on the bottom of the flight case. With three bolts, the mounting plate is secured to the case. Because the battery is placed on the bottom of the flight case, the center of mass is low, making the cage easy to move around.


(a) Bracket

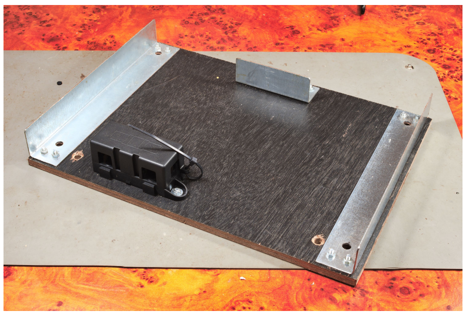
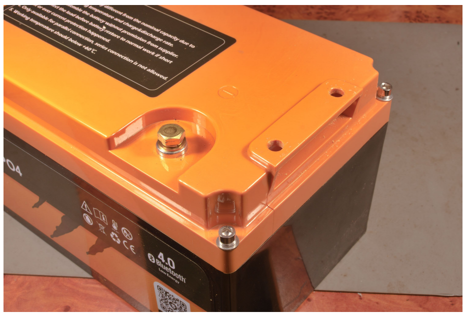

(b) Plate

Figure 5: battery mounting solution


(a) Mounting holes

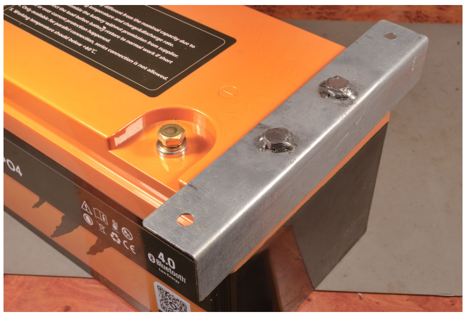
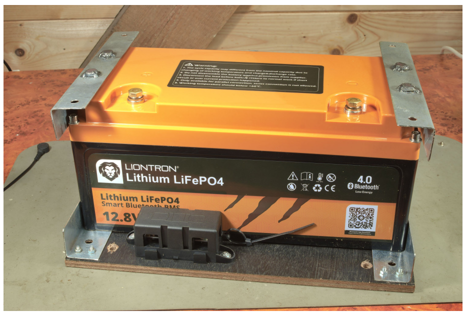
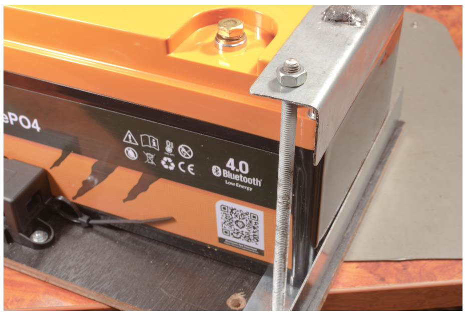

(b) Bracket in place

Figure 6: installing bracket



(a) Battery installed



(b) Bolted down

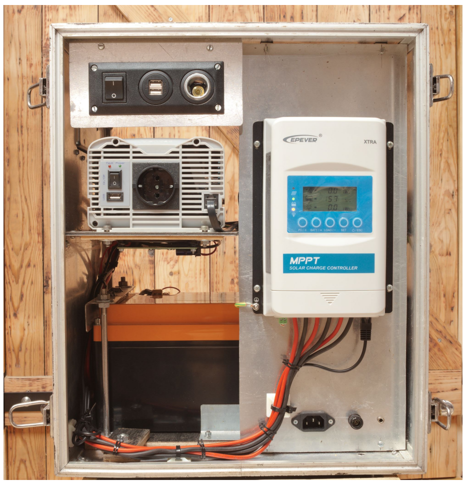
Figure 7: battery mounted

4.3 The final product

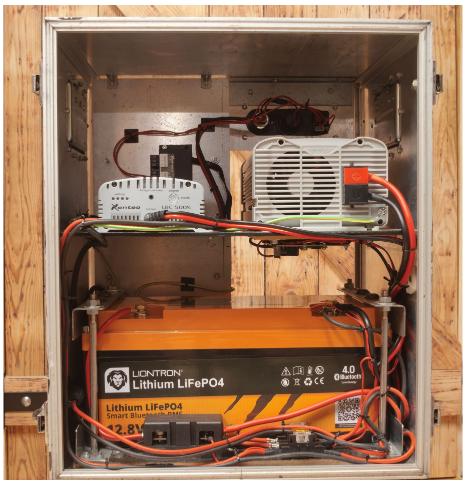
Everything fits nicely in the flight case. All cables are tied together and secured to the housing with self adhesive cable mounts. This prevents the cables from coming loose when the flight case is transported. On the front (figure 8a) the mains connector for the mains charger is clearly visible. Next to it are two 4 mm connectors for connecting the case to the protective earth.

In figure 9 you can see the PPR-2 controller for the inverter screwed to the backside of the front panel. The backside of the control panel and its wire harness is also visible.

Figure 10 give a peek on the Raspberry Pi Zero W 1 controller board. The controller is mounted on a large perforated prototyping board. This way, there is enough room for future expansions.



(a) Front side



(b) Back side

Figure 8: the fully build solar generator

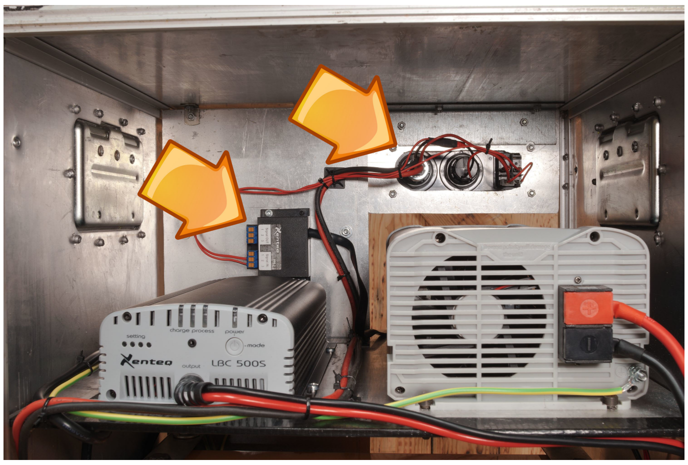


Figure 9: PPR-2 controller

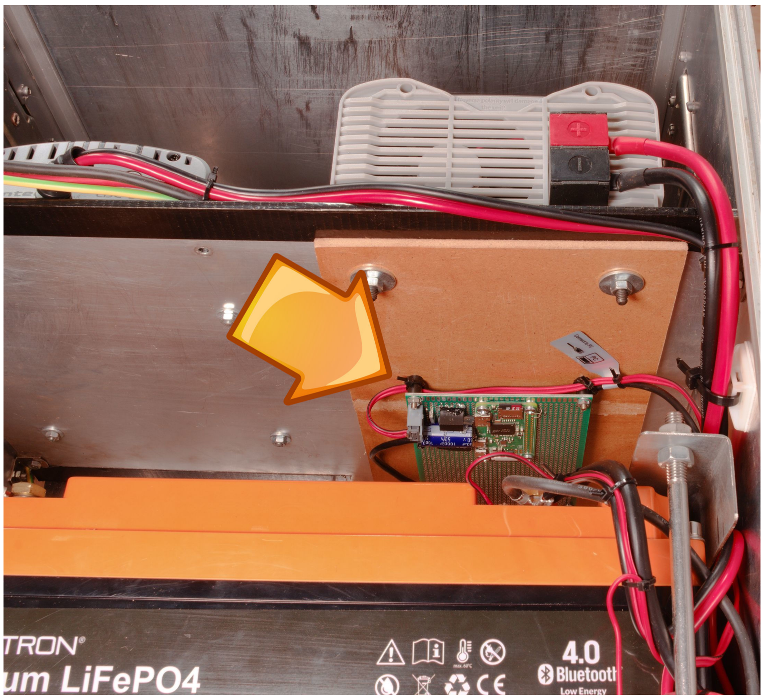
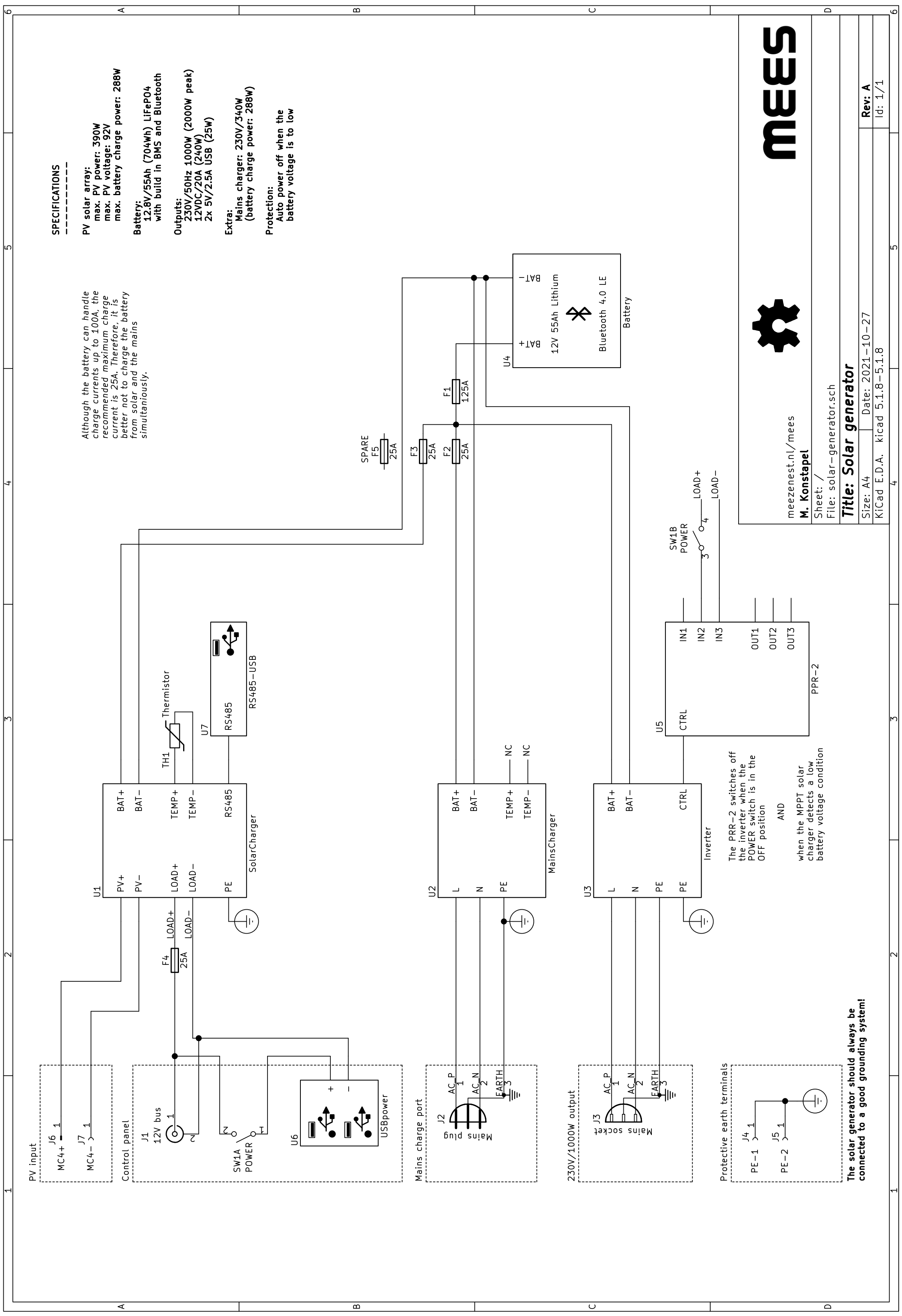


Figure 10: Raspberry Pi controller

A Full schematic and bill of material

#	Reference	Description	Value	Manufacturer	Ordering #
1	F1	Main fuse	125A	Ripca	MEGA-fuse holder with 125A fuse
4	F2 F3 F4 F5	Secondary fuses	25A	Tru Components	TC-9070404 with 25A fuse
1	J1	Front panel - 12VDC connector	12V bus	ProCar	67326010
1	J2	Mains connector charger	Mains plug	Tru Components	781/SW
1	J3	Mains power outlet	Mains socket	-	Part of U3
1	J4	Protective earth terminal	PE-1	Schnepp	BU 4000
1	J5	Protective earth screw terminal	PE-2	Monacor	BP-360/SW
1	J6	Solar connector	MC4+	MultiContact	MC4+
1	J7	Solar connector	MC4-	MultiContact	MC4-
1	SW1	Front panel - power switch	POWER	-	Part of J1
1	TH1	Temperature sensor solar charger	Thermistor	EPEVER	included with solar charger
1	U1	MPPT solar charger	SolarCharger	EPEVER	XTRA2210N-XDS2
1	U2	Mains powered battery charger	MainsCharger	Xenteq	LBC 512-20S
1	U3	12VDC to 230VAC inverter	Inverter	Xenteq	PPI 1000-212C
1	U4	Lithium battery	Battery	Liontron	LiFePO4 Smart BMS 12.8V 55Ah
1	U5	Controller for inverter	PPR-2	Xenteq	PPR-2
1	U6	Front panel - USB power connector	USBpower	-	Part of J1
1	U7	Programming cable solar charger	RS485-USB	EPEVER	PCUSB

Figure 11: bill of material devices



SPECIFICATIONS
PV solar array:
max. PV power: 390W
max. PV voltage: 92V
max. battery charge power: 288W
Battery:
12.8V/55Ah (704Wh) LiFePO4
with build in BMS and Bluetooth
Outputs:
230V/50Hz 1000W (2000W peak)
12VDC/20A (240W)
2x 5V/2.5A USB (25W)
Extra:
Mains charger: 230V/340W
(battery charge power: 288W)
Protection:
Auto power off when the
battery voltage is to low
Although the battery can handle charge currents up to 100A, the recommended maximum charge current is 25A. Therefore, it is better not to charge the battery from solar and the mains simultaniously.
The PRR-2 switches off the inverter when the POWER switch is in the OFF position
AND
when the MPPT solar charger detects a low battery voltage condition
The solar generator should always be connected to a good grounding system!
U1 SolarCharger
U2 MainsCharger
U3 Inverter
U4 Battery 12V 55Ah Lithium Bluetooth 4.0 LE
U5 PPR-2
U6 USBpower
U7 RS485-USB
TH1 Thermistor
F1 125A
F2 25A
F3 25A
F4 25A
SPARE F5 25A
SW1A POWER
SW1B POWER
J1 12V bus
J2 Mains plug
J3 Mains socket
J4 PE-1
J5 PE-2
J6 MC4+
J7 MC4-
PV input
Control panel
Mains charge port
230V/1000W output
Protective earth terminals
meezenest.nl/mees
M. Konstapel
Sheet: /
File: solar-generator.sch
Title: Solar generator
Size: A4
Date: 2021-10-27
Rev: A
KiCad E.D.A. kicad 5.1.8-5.1.8
Id: 1/1

Figure 12: full schematic of the solar generator

B Cable diagrams and bill of material

Id	Description	Qty	Unit	Designators
1	Cable, 1 x 0.75 mm²	1,1	m	W10, W17, W18, W20, W21, W22, W23
2	Cable, 1 x 10 mm²	1,6	m	W1, W4, W5
3	Cable, 1 x 2.5 mm²	1,2	m	W11, W12, W13
4	Cable, 1 x 6 mm²	3,8	m	W15, W16, W19, W2, W3, W7, W8
5	Cable, 2 x 4 mm²	0,4	m	W14
6	Cable, 2 x 6 mm²	0,7	m	W6
7	Cable, 3 x 0.75 mm², attached to device	1	m	W9
8	Connector, Cable lug M4, 0.75 mm²	1		
10	Connector, Cable lug M4, 2.5 mm²	5		
11	Connector, Cable lug M6, 10 mm²	2		
12	Connector, Cable lug M8, 10 mm²	4		
13	Connector, Cable lug M8, 6 mm²	4		
15	Connector, Crimp ferrules, 0.75 mm²	2		
16	Connector, Crimp ferrules, 6 mm²	7		
18	Connector, Faston 4.8, 0.75 mm²	3		
19	Connector, Faston 6.3, 0.75 mm²	6		
21	Connector, Faston 6.3, 6 mm²	8		
22	Connector, MC4, set of two	1		J6_7

Figure 13: bill of material cables

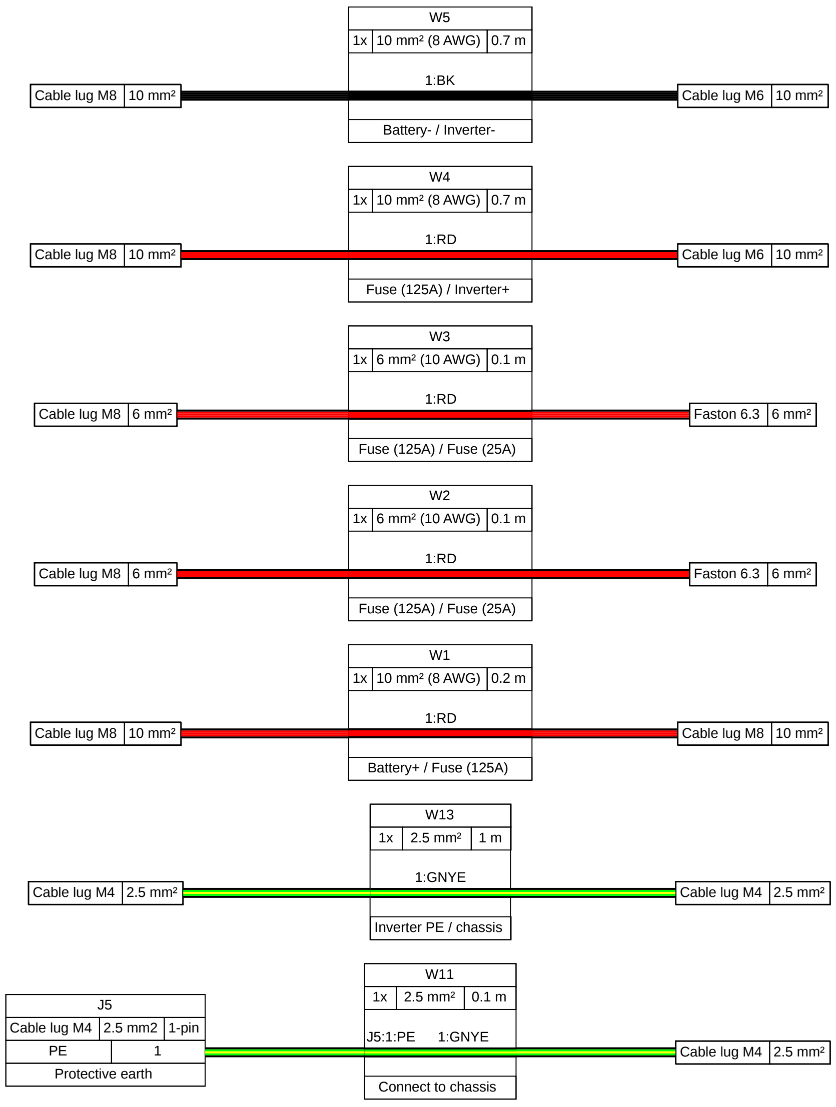


Figure 14: Cables battery

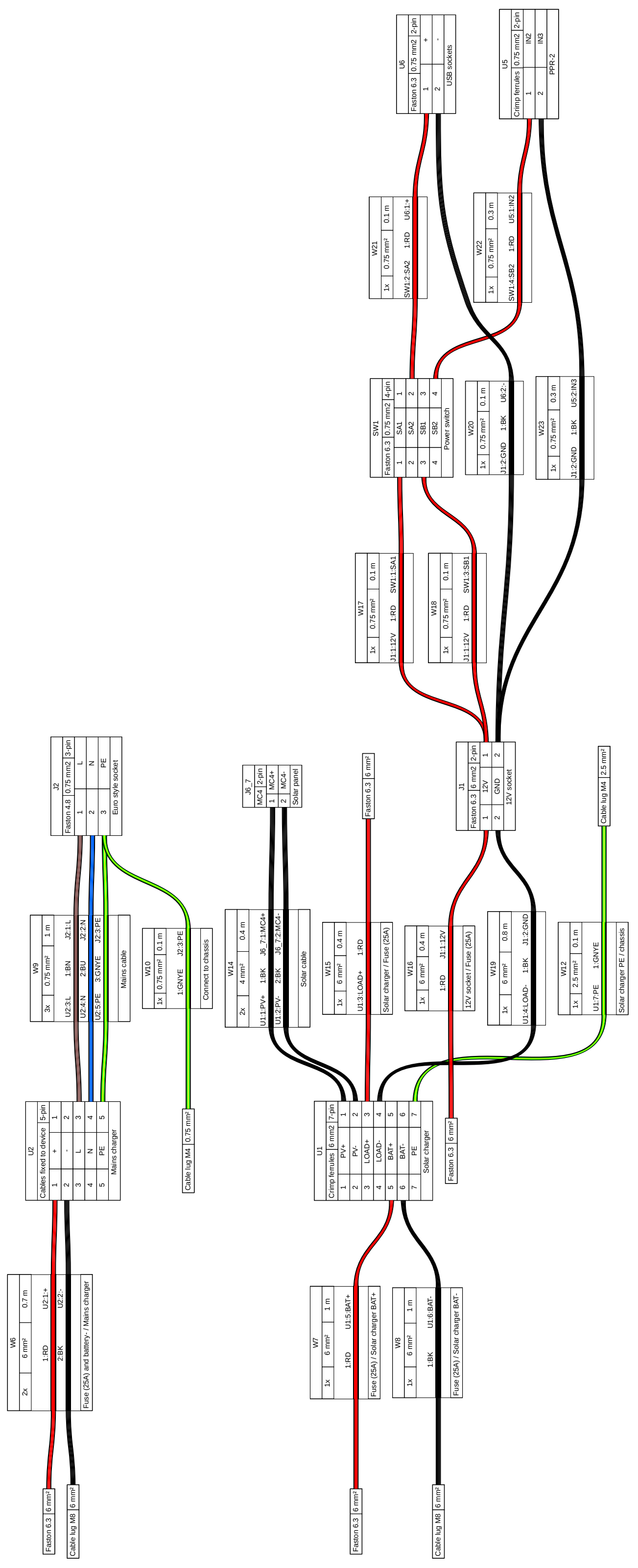


Figure 15: Cables devices

C Open source hardware

All the design files are available on my website: https://www.meezenest.nl/mees