Operators manual for the
FF-ESP32-OpenMPPT v1.3

Document revision: 27.10.22

Up to 100 Watt peak of solar power charging and up to 8 Ampere consumer current from a 12 Volt battery system.

This document has been written by Elektra Wagenrad (onelektra AT gmx DOT net)**,**
with kind funding support by www.apc.org

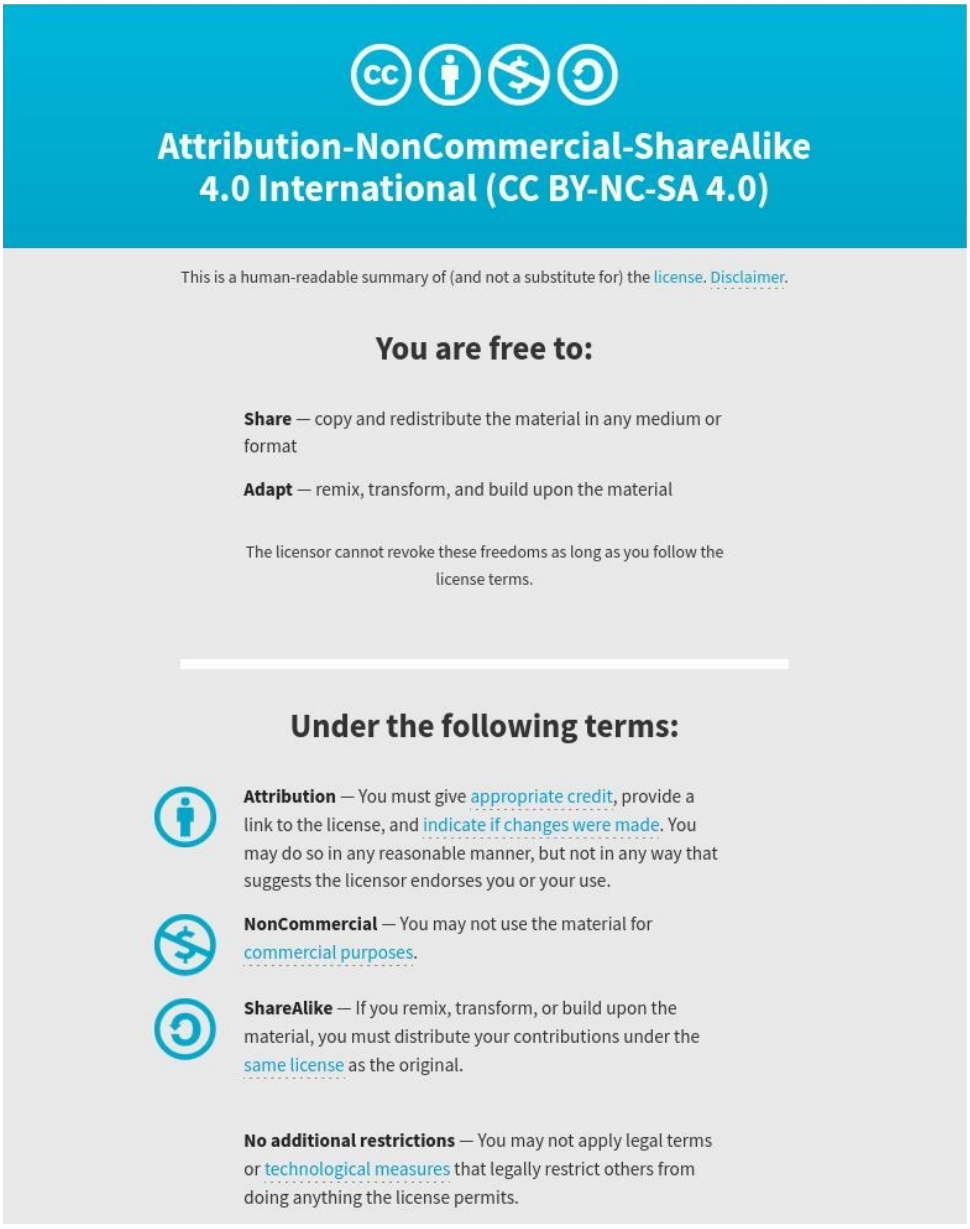


Table of Contents

Introduction

The *FF-ESP32-OpenMPPT* device is an *open-hardware* and *open-software* battery charge controller for photovoltaic solar modules with additional advanced features:

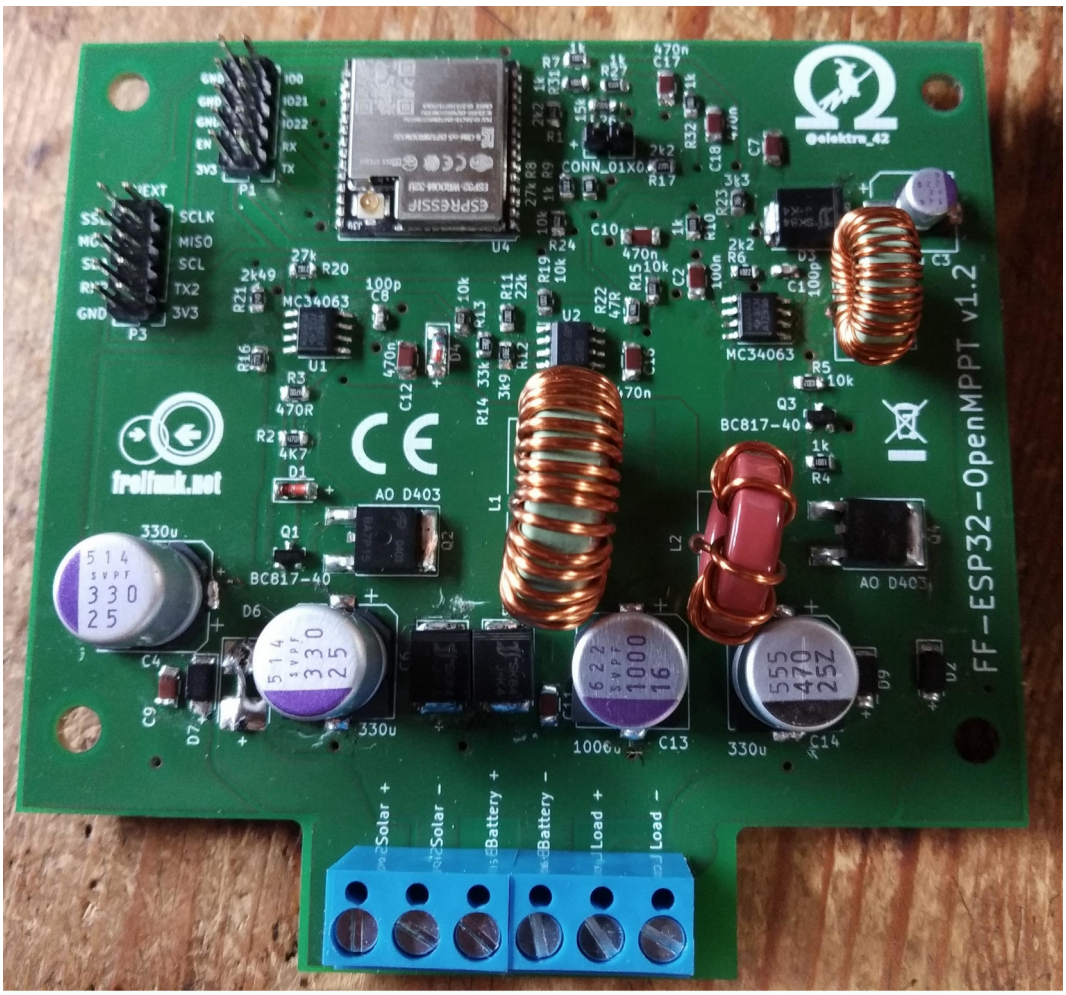
1/ Maximum power point tracking increases the amount of energy harvested from the solar module by up to 30%. An intelligent high efficiency (94%) DC step-down converter draws the greatest amount of energy available from the solar module at its maximum power point and transforms it down to the best battery charge voltage level for maximum charge current.

2/ WiFi networking according to 802.11n with Accesspoint, Client and Accesspoint-Client mode, supporting IP-forwarding and routing. Therefore the device can serve as an energy-autonomous wireless relay and/or sensor node in sensor networks or general purpose autonomous wireless infrastructure.

3/ Firmware based on Free-RTOS/Nodemcu. Programmable and extendable software in Lua, a high level programming language

4/ Support for additional external hardware like displays, environmental sensors and so on via GPIOs, SPI, i2c, serial port.

5/ Battery temperature monitoring and temperature controlled charging according to IU-charging protocol. IU-protocol: Charge with maximum current until charge end voltage is reached. Reduce



charge current until self-discharge current is reached. Maintain current supply on self-discharge level to take care of self-discharge current.

6/ Telemetry support via MQTT protocol.

7/ Built-in command line interface via Telnet protocol or serial terminal.

8/ Up-/download and execution of software modifications/extensions/updates by developers and operators via serial link, Telnet and FTP.

9/ State of charge estimation.

10/ Watchdog function.

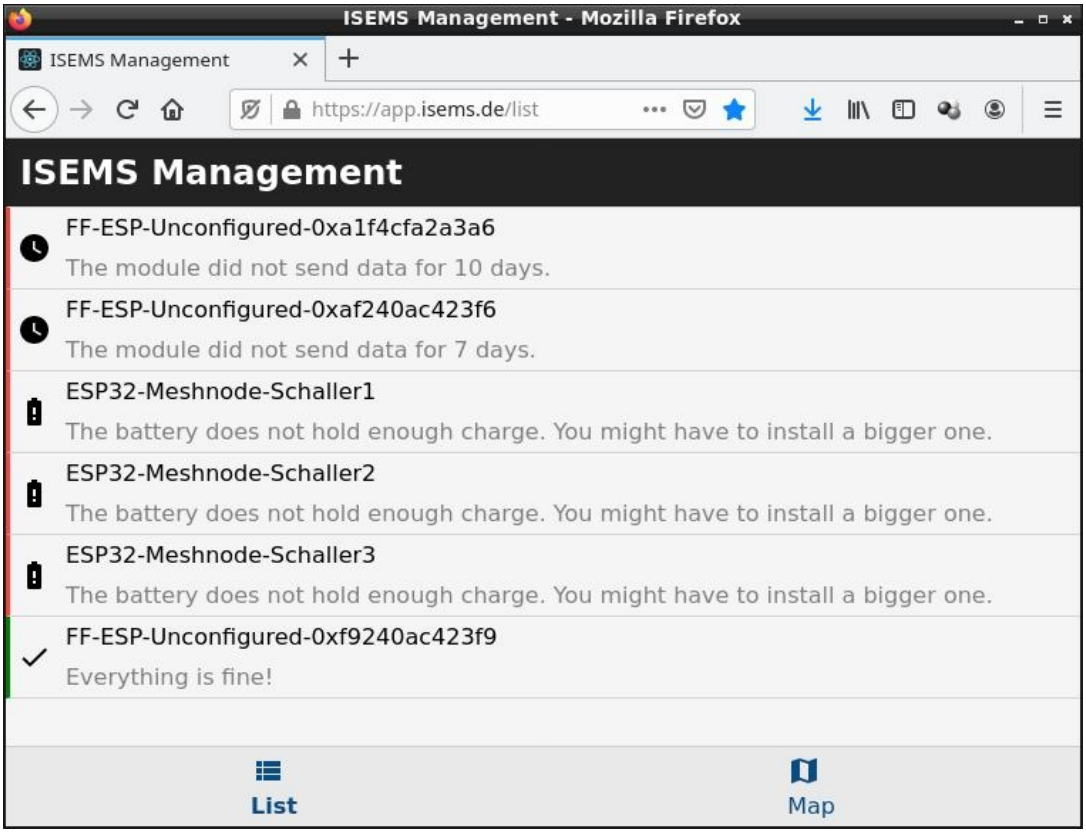
11/ Remote control of load.

12/ Low voltage disconnect and deep sleep mode.

13/ Regulated fully analog charging without software operation to recover from deep discharged battery state.

14/ Internal HTTP server with status, command and configuration page.

15/ ISEMS (Independent solar energy mesh system) web app support at https://app.isems.de without registration.



The ISEMS Web app start page – nodes with issues appear at the top.

16/ Free online Graphana- and Kotori-based telemetry dash board, kindly provided by https://isems.mqtthub.net.

Mqtthub.net showing the battery voltage graph over the last 96 hours.

The FF-ESP32 is designed and programmed to charge 12 Volt lead-acid batteries (AGM-, Gel-, solar-, starting-, lighting-, traction-) from a single PV solar module or a combination of multiple PV solar modules with a total power rating of up to 100 Watt peak and up to 25 Volt open circuit voltage. Since the device is open to developers in the hard- and software domain, it can support many applications.

Applications

- Least cost, low power energy-autonomous WiFi relay station or accesspoint
- Watchdog and solar power supply for wireless relays with external hardware (FM, HAM, WiFi, HF, VHF, UHF)
- Energy autonomous street lighting

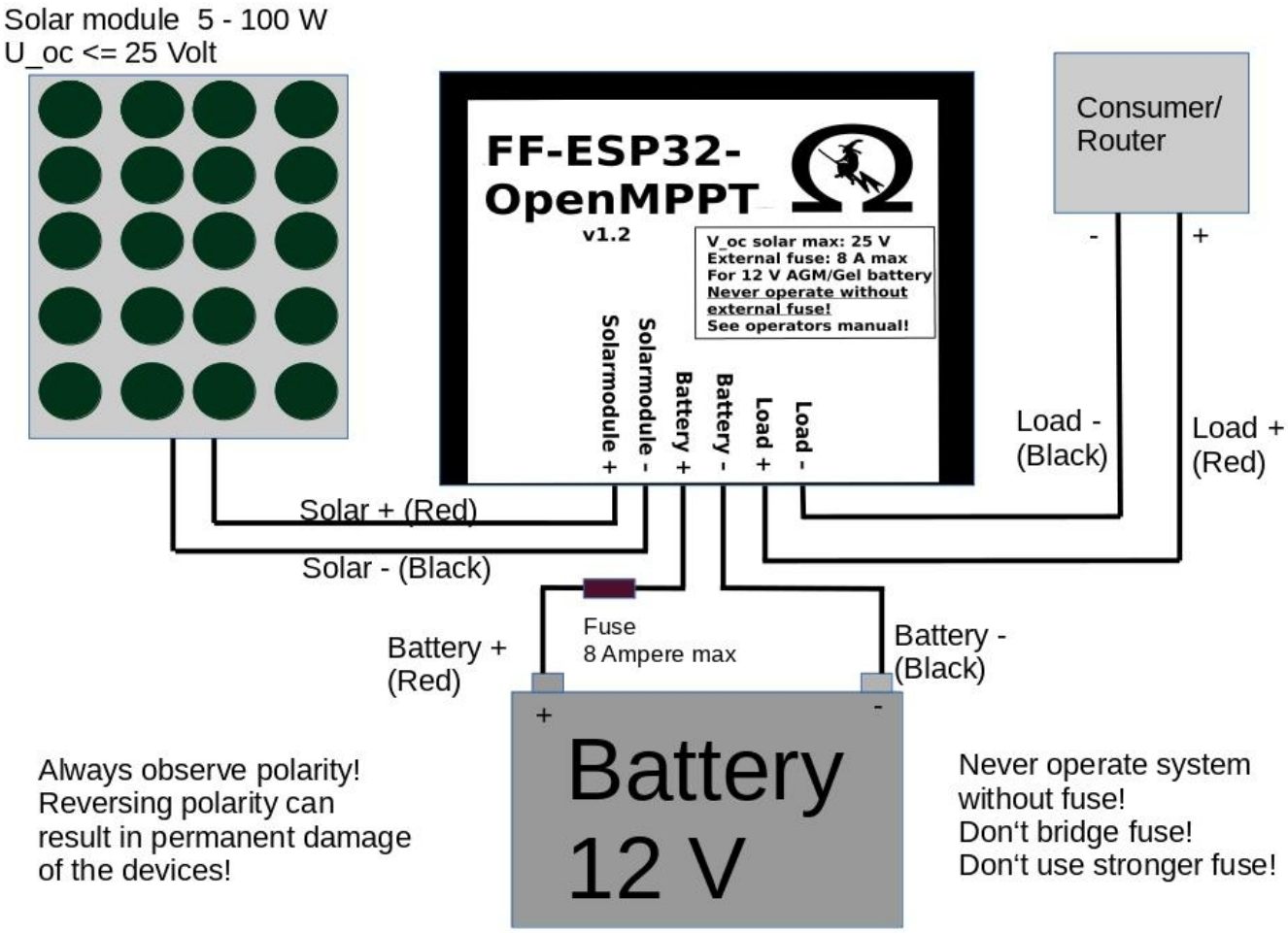
- Low cost and high efficient small scale solar system for lighting and computing
- Remote battery monitoring
- Environment monitoring sensor networks
- Energy autonomous, general purpose low-cost networking infrastructure
- Replacement for ineffective PWM solar chargers for shacks, campers, boats
- Ad-hoc communication infrastructure for disaster recovery
- Solar mobile phone charging systems
- Energy autonomous irrigation systems

and many more…

FF-ESP32-OpenMPPT developer & evaluation kit

* FF-ESP32-OpenMPPT system board
* FTDI USB to serial TTL dongle
* Analog temperature sensor
* Modular housing box & 2 screws 2.9 x 4.5mm
* WiFi Antenna (SMA)
* Wifi Pigtail (U.FL to SMA)
* FKS clip-on fuse holder and fuse 7.5 Ampere
* OLED I²C display module
* Set of jumper cables

Quick installation guide



Tools and installation materials

1 or more solar panels with at total capacity of up to 100 Watt.
A fuse holder and fuse, rated 8 Ampere or less.
The OpenMPPT ;)
A 12 Volt AGM battery of adequate capacity.
Use cable with 1.5mm² or 2.5mm² wire cross section. Preferably red (positive polarity) and black (negative polarity) according to international DC color coding.
Cable lugs that fit the cable wire cross section and the battery terminals.
Battery terminal grease if the system uses non-maintentance-free batteries.
Some basic tools: Flat head screwdriver, plier, side-cutter.
A multimeter.

Step by step installation guide

1/ Cut the cables to the required length.

2/ Remove approximately 5 mm of the cable insulation of the red and black cables from each cable end.

3/ Twist the uninsulated 5 mm end of the individual wires, so they stay together.

4/ Connect the cables for the load (if any) to the clamps labeled „Load +“ and “Load -“ of the OpenMPPT. Use the red cable for “Load +“. Use the black cable to “Load -“. But don‘t plug in the load(s) just yet. Observe polarity! If you have more than one consumer, you probably need some kind of distribution panel and a fuse block. If you have a cable with a DC plug for the device, connect it to he “Load“ output, but don‘t plug it into the consumer device yet. We will do that later, after checking polarity with the multimeter.

5/ Check that there are no open ends of the red and black “Load“ cables (if any) that touch each other.

6/ Make the connection “Solarmodule -“ between the solar module negative terminal and the OpenMPPT. Use black cable.

7/ Make the connection “Battery -“ between the battery negative terminal and the OpenMPPT. Use black cable according to the DC color scheme, if available. Crimp the cable lug fitting your battery terminals with the plier. Connect the black cable with the cable lug to the black terminal (-)

8/ Connect red battery cable to the OpenMPPT port labeled “Battery +“. Crimp the cable lug fitting your battery terminals with the plier. If positive and negative cables are of the same color, mark or label the positive cable on each end. (I have the habit of tying a node at the end of each positive cable.)

9/ Cut the positive (red) battery cable approximately 20 cm away from the end with the cable lug. Bridge the cut with the fuse holder. Insert the fuse.

10/ If you use batteries that produce corrosive gas in normal operation (those that are not GEL, AGM or at least maintenance-free type) it is a good idea to wrap the circuit breaker with tape, to protect them from corroding. The positive terminals attract the gas. Often in old cars you see that the positive terminal of the battery has a layer of blueish-gray looking corrosion on the terminal, while the negative terminal is clean.

10/ Check all connections you have made so far for firm connection and polarity.

11/ Connect the positive (red) battery cable to the positive terminal of the battery.

12/ Make the connection “Solarmodule +“ between the positive terminal of the solar module and the OpenMPPT.

13/ Set up the Multimeter for DC voltage measurements. Check the polarity of the voltage on distributor / at the DC plug (if any) before connecting any consumers. If your device expects positive polarity in the center of the plug, touch the center of the plug with the red probe and the outside of the plug with the black probe. If it shows the DC voltage reading without negative prefix, the polarity is not reversed.

14/ Connect the load.

15/ Connect the battery temperature sensor to the 2-pin battery header.

16/ Attach the battery sensor to the battery housing with good quality adhesive tape.

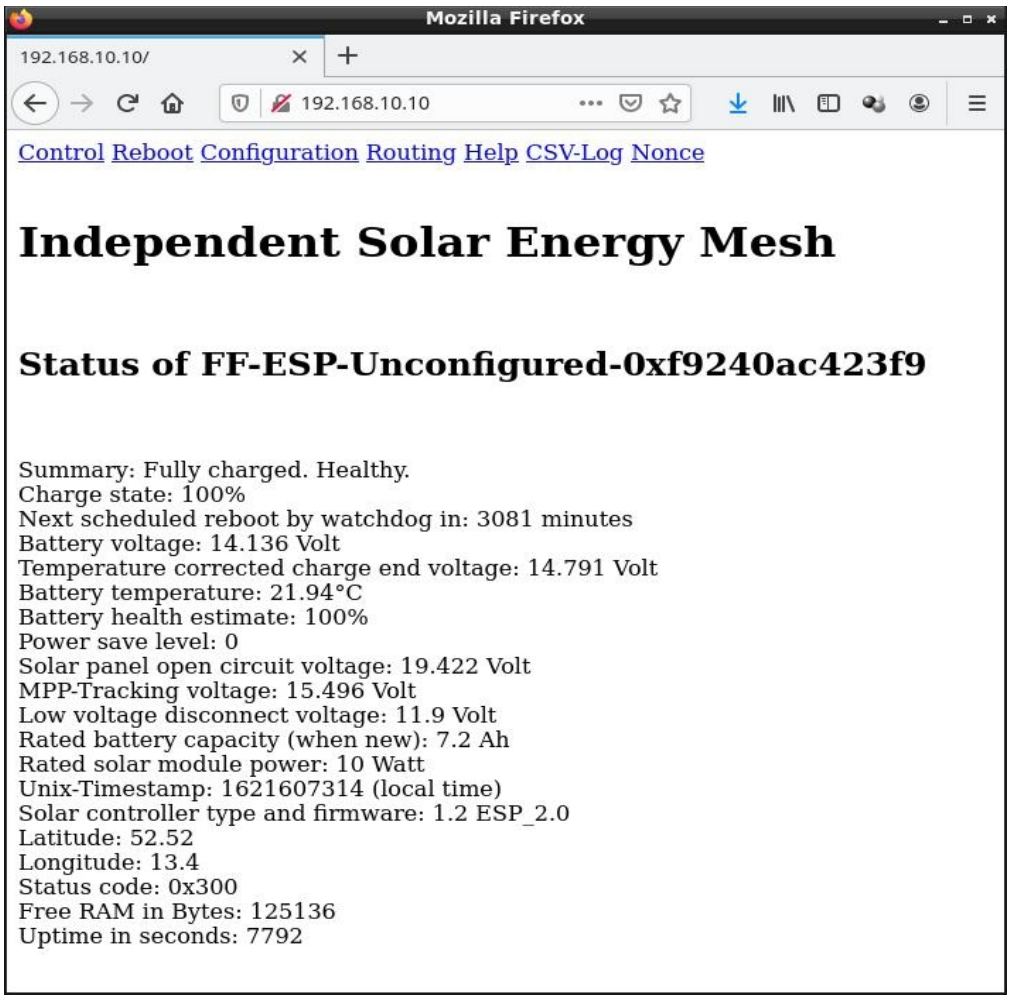
17/ Carefully attach the tiny U.FL antenna cable & plug to the U.FL socket on the ESP32 module, if it isn’t attached already. Connect a WiFi antenna to the SMA socket.

NOTE: If you frequently disconnect/reconnect the U.FL plug to the U.FL socket, the U.FL plug might lose the tight grip that it has when new. U.FL is not designed to be frequently plugged/unplugged. This can be fixed by carefully tightening the U.FL plug with a small plyer. If this doesn’t work out, replace the U.FL pigtail.

18/ Take a WiFi enabled device (PC/Notebook/Tablet/Smartphone). Search for new WiFi networks and connect to the AP “esp32-isems-ap", the WiFi key is "12345678"

19/ Open the URL http://192.168.10.10 in your browser. Note that the protocol is http not https. If your browser complains about the security risk of connecting to a http server – please ignore and accept the „risk“.

20/ You are greeted by the status page.



21/ To customize the settings, select “Configuration“ from the menu at the top of the page. You will be asked for username and password credentials.

Username: **root**
Default password: **pass123**

Note: The same password is used for Web administration, Telnet login and FTP.

Initial software setup and customization

While you are in the configuration menu, it might be a good idea to customize some settings. Settings you probably want to change:

- Administrator password
- WiFi modes, WiFi AP and/or Client network name and WiFi key
- Set a unique device name
- Battery capacity in Amperehours
- Average consumer current
- Rated peak power of solar modules
- Geolocation with latitude and longitude
- By default MQTT support is turned off. If the FF-ESP32-OpenMPPT is connected to the Internet, I highly recommed to turn MQTT support on. Two MQTT brokers are pre-configured, so you can monitor the device operation online. In the section **MQTT-Telemetry configuration** of the configuration page, set **Enable MQTT?** to "**true".**

Finally, click on the „**Submit**" button at the bottom of the page.

Once you have made and saved the changes, the device has to be rebooted in order to apply them.

All settings are stored in the file **config.lua** in the ESP32 flash memory. In case you break something in the software, you can always up/download or delete the content of the flash memory with the **nodemcu-tool** and the serial cable. However, this requires physical access to the device.

WiFi modes

The FF-ESP32 WiFi interface supports 4 modes:

1 = STATION

In this mode, the FF-ESP32 can connect to a WiFi hotspot as a WiFi client. If the WiFi hotspot offers Internet access, the FF-ESP32 is also connected to the Internet.

WiFi Station mode (or WiFi client) mode is the „normal“ WiFi mode that probably everyone knows, since it is used by smartphones, tablets, notebooks or PC to connect to the Internet via WiFi. Station mode requires a WiFi hotspot (or accesspoint) in range and the FF-ESP32 device connects to the hotspot as a WiFi client. This is also the most power saving mode when running the FF-ESP32 WiFi interface.

2 = SOFTAP

In this mode, the FF-ESP32 WiFi interface is in the opposite role of a WiFi client: It acts as a WiFi hotspot. Up to 8 smartphones, tablets, notebooks or PCs can connect to the FF-ESP32 via WiFi. By default, the FF-ESP32 configuration file is set up to offer an WiFi hotspot with the WiFi name (SSID) “**esp32-isems-ap“** and the WiFi key is **12345678**, so anyone can connect to the FF-ESP32 via WiFi. By default, WiFi clients of the FF-ESP32s accesspoint automatically get an IPv4 address in the range of **192.168.10.0/24** and the FF-ESP32 device itself is available under the IP address **192.168.10.10** for http, telnet and ftp access.

Note: In this mode the FF-ESP32 doesn’t offer Internet access to the WiFi clients, only a Wireless local area network (WLAN). This is also the most power consuming mode of the FF-ESP32 WiFi interface.

3 = STATIONAP

In this mode, the FF-ESP32 WiFi interface creates a WiFi hotspot like in mode 2 and it is also able to connect to another hotspot like in mode 1 at the same time. The local WiFi hotspot created by the FF-ESP32 and the remote hotspot that the FF-ESP32 connects to as a client must be on the same channel. The power consumption is the same as in mode 2 of the FF-ESP32 WiFi interface.

Note: In this mode the FF-ESP32 can offer Internet access to the WiFi clients of its own WiFi hotspot. In order to make this work, a static IPv4 routing entry must be added to the routing table in the remote WiFi hotspot that the FF-ESP32 is connecting to as a WiFi client.

Here is an example, using example addresses and a Linux based firmware like OpenWRT on the remote hotspot (you might guess that I’m a big fan of OpenWRT):

The remote OpenWRT WiFi hotspot gives out IPv4 adresses to its hotspot clients in the range of **192.168.0.0/24**

The FF-ESP32 WiFi client interface receives an IPv4 address of **192.168.0.101** from the remote OpenWRT WiFi hotspot. (This is just an example address and will likely be different on your side!)

If you are wondering now how to figure out the IPv4 address of the FF-ESP32 device, check out the log of the DHCP leases in the remote hotspot. Look for the client IP for the device with the name „**espressif**“. On the OpenWRT command line, this can be done with the command:

cat /tmp/dhcp.leases

You will find something like:

*1645556115 24:0a:c4:23:f9:24 192.168.0.101 espressif **

The required routing entry can be added into the OpenWRT device with the “**ip r**“ command on the command line:

ip r add 192.168.10.0 via 192.168.0.101

Now the WiFi clients of the local FF-ESP32 WiFi hotspot can reach the Internet. One can check this by simply running the command “**ip r**“. In the routing table, you will see an entry like this:

192.168.10.0 via 192.168.0.101 dev br-lan

4 = NULLMODE

Nullmode means that the WiFi interface is turned off. In this mode, configuration settings can only be changed by uploading a new and manually edited configuration file „config.lua“ using the serial port and nodemcu-tools. Disabling WiFi altogether is therefore only recommended if this is really what you want.

What if the IPv4 address of the FF-ESP32 changes from time to time?

The OpenWRT DHCP server tries to give out the same IP address to the same client devices according to their MAC address. However, this is not guaranteed, if many clients are connecting to the OpenWRT hotspot and the IP address is already occupied, because the FF-ESP32 was offline for a while. If the FF-ESP32 IP address changes, the DHCP server in OpenWRT can be configured to always give the same IPv4 address to the FF-ESP32 (or any other DHCP client device, for that matter).

On the OpenWRT command line, execute:

uci add dhcp host
uci set dhcp.@host[-1].name="FF-ESP32"
uci set dhcp.@host[-1].dns='1'
uci set dhcp.@host[-1].mac="00:11:22:33:44:55"
uci set dhcp.@host[-1].ip="192.168.0.101"
uci commit dhcp
/etc/init.d/dnsmasq restart

I’m using the IPv4 addresses of the forementioned example. The MAC address should match the WiFi client MAC address of your FF-ESP32 device, of course.

WiFi performance versus power saving tweaks

Operating a WiFi transceiver (**trans**mitter & re**ceiver**) consumes a considerable amount of energy. Transmitting WiFi signals consumes more energy than receiving them, but the energy consumption of an active WiFi receiver is considerable for any energy-autonomous or battery-powered embedded system. In WiFi mode 1 = STATION the ESP WiFi driver tries to put the client interface into sleep mode most of the time by default. The receiver only wakes up to check for Accesspoint beacon signals that are sent at an interval of typically 100 ms. If the AP wants to send data to the FF-ESP32, the AP tells the FF-ESP32 in its beacons to wake up.

If the FF-ESP32 is configured to operate as a WiFi AP (Mode 2 = SOFTAP or Mode 3 = STATIONAP) instead of Mode 1 = STATION, sleeping of the WiFi receiver is disabled. The FF-ESP32 draws about 0.35 Watt more power in AP mode compared to client-only mode with power

saving turned on. The added power consumption of the entire system is mostly the result of the WiFi receiver in AP mode being active all the time to receive client data. It is also transmitting beacons and talking/responding to clients.

In SoftAP+Client mode, power saving is turned off for both interfaces.

Power saving is great for the FF-ESP32 to preserve energy in WiFi client-only mode. In client mode, power saving is enabled by default. If quicker networking response and performance is required at the expense of increased energy consumption, power saving can be turned off. This is nice, for example, if you want to work on the device via a more responsive Telnet shell.

Enabling / disabling WiFi power save mode

The FF-ESP32 firmware supports the global Lua function **wifi.setps()**

Calling the function with the parameter **wifi.PS_NONE** disables power saving mode for the client interface:

wifi.setps(wifi.PS_NONE)

Calling the function with the parameter **PS_MIN_MODEM** enables power saving mode of the ESP client interface again:

wifi.setps(wifi.PS_MIN_MODEM)

The function **wifi.setps()** can be called from a Lua program or manually via serial console or via the Telnet Lua console (like all other global lua functions in the firmware) .

An even more aggressive power saving parameter **wifi.PS_MAX_MODEM** is available, but this setting basically makes networking access so slow and sluggish that I consider it unusable.

Power saving is easily noticeable when pinging the FF-ESP32 via its WiFi-client interface. A considerable round-trip-time is the result. To make matters worse, the ESP WiFi powersaving implementation ramps up the response time as the process continues:

ping 10.36.158.61

PING 10.36.158.61 (10.36.158.61) 56(84) Bytes Data.

64 Bytes von 10.36.158.61: icmp_seq=1 ttl=255 Zeit=0.977 ms

64 Bytes von 10.36.158.61: icmp_seq=2 ttl=255 Zeit=1.00 ms

64 Bytes von 10.36.158.61: icmp_seq=3 ttl=255 Zeit=16.2 ms

64 Bytes von 10.36.158.61: icmp_seq=4 ttl=255 Zeit=38.7 ms

64 Bytes von 10.36.158.61: icmp_seq=5 ttl=255 Zeit=60.4 ms

64 Bytes von 10.36.158.61: icmp_seq=6 ttl=255 Zeit=87.6 ms

64 Bytes von 10.36.158.61: icmp_seq=7 ttl=255 Zeit=104 ms

[...]

64 Bytes von 10.36.158.61: icmp_seq=39 ttl=255 Zeit=842 ms

64 Bytes von 10.36.158.61: icmp_seq=40 ttl=255 Zeit=871 ms

64 Bytes von 10.36.158.61: icmp_seq=41 ttl=255 Zeit=888 ms

64 Bytes von 10.36.158.61: icmp_seq=42 ttl=255 Zeit=914 ms

64 Bytes von 10.36.158.61: icmp_seq=43 ttl=255 Zeit=1.37 ms

It ramps up to almost one second response time, goes back to short response time and ramps up again. If round trip time in power save mode would be shown in a graph, it would look like a sawtooth.

A client-only interface with power save active is suitable for monitoring the system parameters and sensors connected to it, sending and receiving MQTT data, occasional use of the device HTTP status page and so on. The power saving effect is considerable. If better performance is required power save can be turned off at the expense of additional energy consumption.

Once power saving for the client interface is turned off, the performance and traffic forwarding performance is great for such a small device:

ping 10.36.158.61

PING 10.36.158.61 (10.36.158.61) 56(84) Bytes Daten.

64 Bytes von 10.36.158.61: icmp_seq=1 ttl=255 Zeit=1.48 ms

64 Bytes von 10.36.158.61: icmp_seq=2 ttl=255 Zeit=3.29 ms

64 Bytes von 10.36.158.61: icmp_seq=3 ttl=255 Zeit=1.43 ms

64 Bytes von 10.36.158.61: icmp_seq=4 ttl=255 Zeit=3.23 ms

64 Bytes von 10.36.158.61: icmp_seq=5 ttl=255 Zeit=3.48 ms

64 Bytes von 10.36.158.61: icmp_seq=6 ttl=255 Zeit=1.26 ms

64 Bytes von 10.36.158.61: icmp_seq=7 ttl=255 Zeit=1.41 ms

64 Bytes von 10.36.158.61: icmp_seq=8 ttl=255 Zeit=1.31 ms

The ramping-up effect of the network response time is gone as well.

Making a backup using FTP (File transfer protocol)

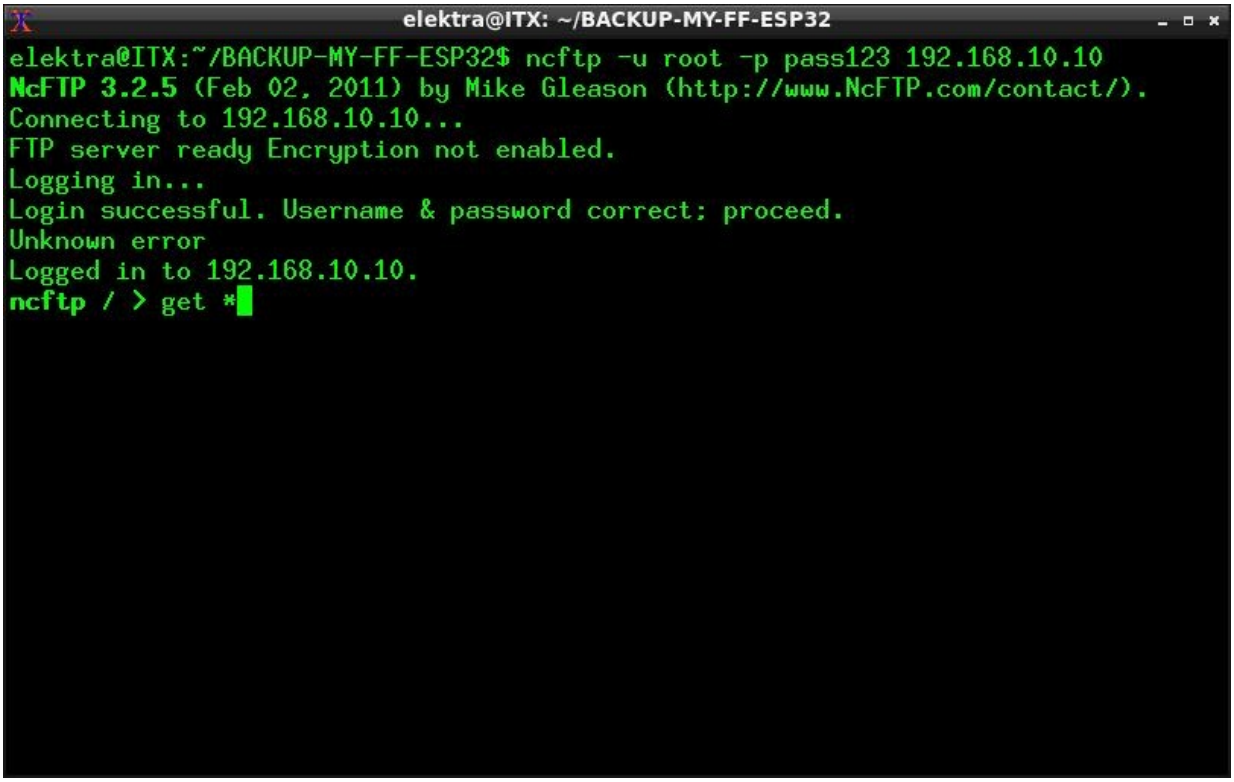
It is a good idea to make a backup of all the files in the ESP32s flash memory on your computer. The files **board.lua** and **VrefCal** contain individual calibration data for each FF-ESP32 device. A backup copy of **config.lua** will remind you of the password and other settings.

In order to make a backup, one can connect to the FTP server of the FF-ESP32. Unfortunately, mainstream web browsers like Firefox and Chrome have deliberately removed support for the File Transfer Protocol, because they consider it an insecure legacy. Hence, if you click on ftp://192.168.10.10 an external program should start.

If not, one has to be installed. I use **ncftp** (a command line program for Linux) or bare bone **ftp** (yet another command line program).

Note: The FTP server of the FF-ESP32 only supports passive mode FTP. On my Linux machine, simply invoking the **pftp** command instead of **ftp** turns on passive mode.

Here, I am executing the program inside the local folder **BACKUP-MY-FF-ESP32**. The **ncftp** message “Unknown error“ can safely be ignored. **get *** simply downloads everything from the server into my local folder after pressing *Enter*. The command line of ncftp understands the **passive** command to turn passive FTP mode on and off.



If you want to upload a file to the FF-ESP32, you can use the **put <local file>** command. Once you are done, type **quit** and press *Enter.*

Telnet use

Telnet is an ancient and basic remote login shell protocol for the remote execution of commands **without transport encryption**, so username/password credentials and commands can be sniffed – if the network is not safe.

There are two telnet user names / user accounts available: **root** and **lua**.

The user account „**lua**“ logs in to a Lua programming shell, the same Lua programming shell that is accessible via serial port. The main difference between serial terminal access and wireless telnet access is that the serial terminal is more robust and doesn’t need a working WiFi connection for debugging.

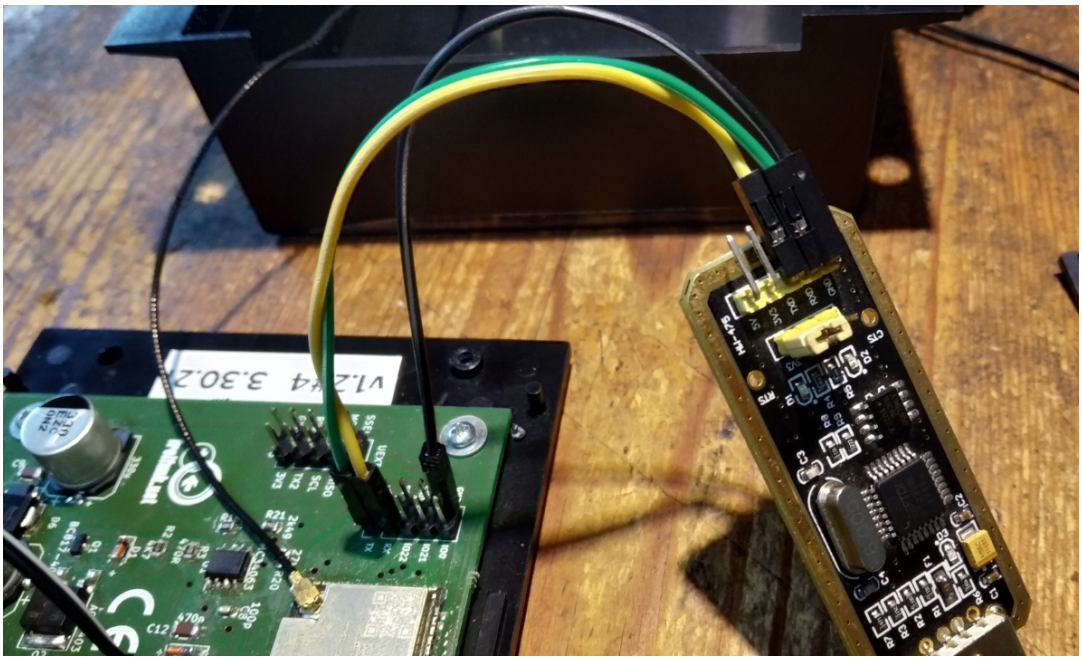
The user account „**root**“ logs in to a minimalistic Unix-like shell. The following commands exist: *cat config console cp df dump exit free help ls mv ota pkg print reboot rehash rm route uptime wget wifiscan*

Serial port connection for programming, flashing, software development and recovery.

Setting up the serial port connection

Hardware

The serial port connection from your PC or Laptop to the FF-ESP32 requires a USB to TTL serial port adapter with *3.3 V logic level* and the following connections between the USB to TTL device and the pin header *P1* of the FF-ESP32:



Wiring is simple:

TX to RX
RX to TX
GND to GND

Note: It is not necessary or recommended to connect the 3.3 Volt pin of the FF-ESP32 with the 3.3 Volt pin of the USB-TTL adapter. The USB-TTL adapter is USB-powered from your PC/Laptop and the FF-ESP32 should be powered from a 12 Volt supply or a 12 V battery. Powering the FF-ESP32 µC from the 3.3 Volt rail provided by USB-TTL adapter might result in three problems:

- *The USB-TTL adapter might randomly hang upon initialization, because it tries to charge the 3.3 Volt rail capacitors of the FF-ESP32 and can‘t deliver enough current.*

- *To power-cycle the FF-ESP32 board, both the 12 Volt supply and the 3.3 Volt connection have to be disconnected.*

- *In order to preserve power, the 3.3 Volt rail of the OpenMPPT is actually tuned to 3.05 Volt. This is also the reference voltage for analog measurements. If the USB-TTL adapter actually manages to feed the rail with 3.3 Volt, some analog readings are off and you can not recalibrate VrefCal, in case you want to recalibrate (which is actually only necessary after manufacturing or if you accidently deleted VrefCal file in the flash memory).*

Hence, in total only 3 wires are needed: ***TX, RX, GND***

A serial link can also been made the same way between a WiFi router with a 3.3 Volt serial port or a single board computer like Raspberry-Pi, Banana-Pi, Odroid or the like. Again: *Only 3 wires are needed:* ***TX, RX, GND***

*Note: There are USB-TTL dongles for 3.3 Volt and 5 Volt TTL (**T**ransistor-**T**ransistor-**L**ogic) levels. 5 Volt TTL levels are incompatible with the FF-ESP32 and may damage the ESP32 chip. Connecting the 3.3 Volt rail pins of the FF-ESP32 to a 5 Volt source (or higher) will certainly destroy the ESP32 chip.*

There are very cheap knock-off USB-TTL dongles on the market with unlabeled USB-TTL chips. They typically cost about 2 US-Dollars or less and the chip is referred to as a PL2303 or PL2303 clone or even an FTDI chip. The quality is extremely bad and it is impossible to get a usable serial connection with them. USB-TTL adapters with genuine FTDI chips are a good choice and cost around 7-10 Dollars.

Software

Via the serial connection, the FF-ESP32 can be debugged, monitored, flashed, reprogrammed or files (including LUA code) deleted, uploaded or downloaded. The serial connection is also the path to fix/debug software issues with the FF-ESP32 if you can‘t access the device via WiFi.

1. Command line access to the LUA shell

In order to interact with the LUA console / command line interface, a generic serial terminal software like *minicom*, *picocom*, *putty* or similar is required. By the way: The LUA console of the FF-ESP32 can also be accessed via WiFi, using the Telnet protocol. This is covered in a seperate chapter.

The serial port parameters are 115200 baud, 8N1. If your connection doesn‘t work, you might have reversed TX and RX.

The terminal needs to know the address of the USB-TTL adapter. If you are using Linux, the first USB-TTL adapter shows up as */dev/TTYUSB0*.

2. Uploading, removing, listing, downloading files (namely LUA code)

My favorite tool for working with the FF-ESP32 is **nodemcu-tool**, which is like a swiss army knife for working with the FF-ESP32. It has a serial terminal option and it can also to delete, upload, list or download files.

Example with nodemcu-tool

In this example, a modified file config.lua is uploaded via serial port first:

nodemcu-tool upload config.lua

Logging into the LUA command line interface via serial port:

nodemcu-tool terminal

Next, a specific nodemcu LUA function is executed. *node.restart()* reboots the ESP32 chip.

node.restart()

Boot messages will start rushing over the screen. The device will then wait for 5 seconds before starting the main program **is.lua**

In case the boot process has to be interrupted, simply type ***stop()*** and press ENTER in less than 5 seconds.

3. Flashing the firmware

Flashing the firmware requires either esptool.py (Linux, Windows, MAC) or NodeMCU PyFlasher (Windows, MAC).

All software options to perform the tasks of firmware flashing or up-/downloading code to the FF-ESP32 from a PC are listed here.

Instructions for flashing the ESP32 from a PC / Laptop

Dimensioning cable cross section and length, taking cable losses into account.

Two considerations are important for selecting the appropriate cable cross section: Losses and safety.

Safety first

Any wire can only handle a limited amount of current before it starts to glow and burn. Before this happens, the circuit breaker (fuse) must kick in.

Cross section	Maximum current (A)	Resistance (Ohm) per meter
0.5 mm^2	7.1 A	0.032
0.75 mm^2	9.1 A	0.024

1.0 mm^2	11.5 A	0.016

1.5 mm^2	16.1 A	0.011

2.50 mm^2	22 A	0.0064

Hence, 1.5mm² or 2.5mm² wire cross section is fine safety-wise for the current level of the FF-ESP32-OpenMPPT system. It is also the right choice to keep wire resistance losses low. Interestingly, the highest current in the entire circuit can occur in the load circuit. One has to take into account that the DC/DC Step-Down MPP tracking circuit produces power and additional power can be drawn from the battery. Both power sources can add their currents together during the day. So, for example 7.5 Ampere current from the MPP tracking circuit can add to 7.5 Ampere current drawn from the battery. Hence, all loads together or a short can draw ~15 Ampere during the day before the fuse kicks in, despite the battery connection wire at the positive terminal being fused with a 7.5 Ampere fuse.

Therefore, safety-wise it is a good idea to add a fuse into the load circuit and not be on the cheap side with regards to wire cross section. The consumers should be fused anyway.

Calculating wire losses that affect efficiency

As shown in the previous section, 0.75mm² wire cross section is just enough for the battery connection for safety reasons, if a 7.5 Ampere fuse is used as circuit breaker. However, a solar system ought to be efficient. The energy that one deals with is limited, hence efficiency is a good idea. Resistance in wires wastes energy by producing heat. Thinner wires have greater resistance.

Note: One can always use a lower rated fuse, if smaller solar module capacity and smaller consumers are used that produce / draw lower currents.

Example calculation:

From the previous table, one can see that 1 mm² of copper wire has a specific resistance of 0.016 Ohm per meter. Hence 10 meter of 1 mm² of copper wire has a resistance of

0.16 Ohm = 0.016 per meter * 10 meters

The actual power loss is dependent on the current floating through the wire, resulting in a voltage drop. According to Ohms law, at 1 Ampere current the voltage loss is 0.16 Volt.

Ohms Law

U(Volt) = I (Ampere) * R (Ohm)

0.16 V = 1 A * 0.16 Ohm

The resistance in the wire acts as a consumer that converts electric energy into heat.

Power (loss)

The formula to calculate the consumed power in the wire resistance:

P(Watt) = U(Volt) * I (Ampere)

0.16 Watt = 0.16 V * 1 A

An energy loss of 0.16 Watt is not that much. However, the calculation was just for 1 Ampere.

At 5 Ampere, the voltage drop in the wire resistance is 0.8 Volt. If we enter the new values in the Power loss calculation:

P(Watt) = U(Volt) * I (Ampere)

4 Watt = 0.8 V * 5 A

4 Watt or 25 times the loss at 1 Ampere. The wire resistance loss grows to the square of the current in Ampere.

The formula to calculate power straight from resistance and current is:

$P(Watt) = I^2$ (Ampere) * R (Ohm)

If 2.5 mm^2 wire is used instead, the wire resistance is 0.064 Ohm = 0.0064 Ohm/meter * 10 meters

Hence the power loss is:

1.6 Watt = 5^2 * 0.064 Ohm

Dimensioning solar panel and battery size

Calculating power consumption

Add up all consumer power requirements and take into account how long they are in operation.

Example 1:

The FF-ESP32 board is in operation for 24 hours a day in AP+Client mode without a consumer attached, the instantaneous consumption is about 0.6 – 0.7 Watt, times 24 hours. So ~17 Watthours (Wh) per day.

Example 2:

The FF-ESP32 board is in operation for 24 hours a day in Client-only mode with a consumer (a Libremesh WiFi router) attached, that draws 7 Watts via a DC/DC step-up converter and also runs 24 hours a day. In order to save power, the Librerouter is not powered via its Power over Ethernet feature, since the thin copper wires in Ethernet cable will cause a considerable energy loss. Instead, cable with 1.5mm² copper cross section, connected to the DC socket, keeps cable losses at a minimum.

The instantaneous consumption of the FF-ESP32 in Client-only mode is about 0.3 – 0.4 Watt, times 24 hours. So ~10 Watthours (Wh) per day in addition to 168 Watthours (7 Watt times 24 hours) for the WiFi router, so the system will draw a total of 178 Watthours per day.

Example 3:

The FF-ESP32 is used to power applications at home. 3 LED-lights with 3 Watt each that are in use for 4 hours a day. The system also powers a notebook computer that is used for 8 hours a day, powered by an efficient DC/DC laptop supply that converts the 12 V DC power from the battery source to 19 Volt. The DC/DC laptop supply is highly efficient (about 95 % efficiency). The instantaneous power consumption is 15 Watt on average. *The DC/DC supply is a great efficiency improvement in comparison to an DC to AC inverter producing 230 V AC (80 % efficiency) followed by a standard mains AC to DC laptop power supply (80 % efficiency), resulting in a total efficiency of 64 %.*

36 Wh are consumed for lighting and 120 Wh are used for computer use per day, so 156 Wh per day in total.

How many Amperehours battery size?

Next, the battery size considering backup time is calculated.

Note: Calculating for a backup time of less than 2-3 days is a design failure, resulting in multiple problems and premature battery damage.

Example 1:

We take into account that, for example, there is no energy coming in at all for five days. So we need to store 85 Wh = 17 Wh per day times 5 days.

If 5 consecutive days without solar power happen just once a year, it is ok to accept that the battery discharges to 100% depth of discharge (DoD).

7 Ah = 85 Wh / 12 Volt

So a 7.2 Ah 12 Volt AGM battery – a standard size, very common in small UPS devices and relatively cheap – will do the job.

If we care for 30 % charge reserve (better!):

9.1 Ah = 7 Ah times 1.3

A calculated backup time of 5 days also means that the system will discharge the battery by less than 10 % at night on average. This low DoD level allows the battery to work for thousands of charge-discharge cycles before wearing out.

Example 2:

5 days backup time add up to 890 Wh consumption.

74.2 Ah = 890 Wh / 12 Volt

Example 3:

5 days backup time add up to 780 Wh consumption.

65 Ah = 780 Wh / 12 Volt

Battery capacity limits solar panel size.

The capacity of the battery limits the size of the solar panel. The critical factor is the maximum **initial charge current.** With a 100 Watt solar module, one can assume a maximum charging current of just over 6 A at 12.5 Volt, which means 75 Watt.

The reason why we are not getting 100 Watt is that hot solar modules have an efficiency of about 80 % and the FF-ESP32 tracker has an efficiency of about 95 %. Hence the total loss (not including cable losses) is 25 %. Actually, the solar industry is cheating a bit when rating the peak maximum power of solar panels. The trick of the industry is to quickly test the solar panel when they are cold (25 °C) in a machine called solar flasher. The measurement protocol in the factory is so fast that the solar module doesn't heat up. However, in actual operation the solar cells heat up to 70 degrees Celsius or more. With increasing cell temperature, the efficiency of mono-crystalline solar cells ***decreases*** *by 0,4 % per degree (0.45 % for poly-crystalline cells).*

So a little over 30 Ampere hours (5 times 6 Ampere) would be the minimum for a 100 Watt solar panel. I usually take about ten times the maximum charging current as the capacity in Ah, if possible. Here is why:

- The slower the charging and discharging, the longer the battery life.
- The lower the discharge current, the greater the effective capacity. The nominal capacity is usually given for a discharge current that discharges the fully charged battery within 20 hours.
- It also follows that the slower the charging and discharging process, the higher the charging / discharging efficiency.
- The service life of the battery depends to a large extent on the depth of discharge and the number of cycles, followed by the charging and discharging rate.

In general, a reduction to 60% of the capacity is considered to be the end of the service life. A smaller battery wears out quicker to the point when it is dead.

According to the data sheets of the 7.2 Ah battery from **Example 1**, the initial charging current for AGM batteries must not exceed 0.3 CA. 1 CA is the nominal Amperehour capacity of the battery, e.g. 7.2 Ah. The initial charge current limit for an 7.2 Ah battery **when new** is therefore

2.16 A = 7.2 Ah x 0.3

If the initial charge current limit is exceeded, the battery will be damaged prematurely.

Over time the battery will inevitably lose capacity due to wear and tear. If the battery just fits the maximum charge current bill when new, the system will run into problems soon. At one point the battery will be a 4 Ah battery internally, since the active material inside has dropped to 60% of the nominal capacity.

If we recalculate the initial charging current for this point, there is now only a maximum of 1.2 A initial charging current allowed. The weakened battery will then be overloaded and stressed by the charging current, resulting in even quicker wear and tear. Since the capacity has already dropped, the discharging cycles have a deeper depth of discharge (DoD) every night, which results in even quicker wear and tear. Both effects generate a vicious circle. A replacement will soon be due. Therefore, the wear of the battery should be taken into account. If the battery is sized generously – at least five times of the maximum charge current, a standard AGM battery from a quality brand lasts at least five years.

A standard Kung-Long or Panasonic AGM has a service life of 1300 cycles at 30% depth of discharge before the capacity of the battery has dropped to 60% of the new capacity.

Now, if the system is designed to discharge the battery to 70 % State of Charge = 30 % Depth of Discharge every night, already when the battery is new, it will drop to 60 % capacity in three years.

The projected service life - if wear is taken into account - is ten years with a longlife type and five years with a standard type. If the battery is undersized, even a long-life type will quickly reach the end of its service life. If the battery is sized generously, it may last even longer before replacement.

Does it help to ridiculously oversize the battery?

A large storage capacity is great, but within limits. The limiting factor is the shelf life, the aging of the battery starting from the day of manufacture, even without use. Even an unused battery that is carefully charged once a year for maintenance will break down at some point. The projected service life of normal types is five years, that of long-life types is ten years. My personal experience is that buying quality pays off.

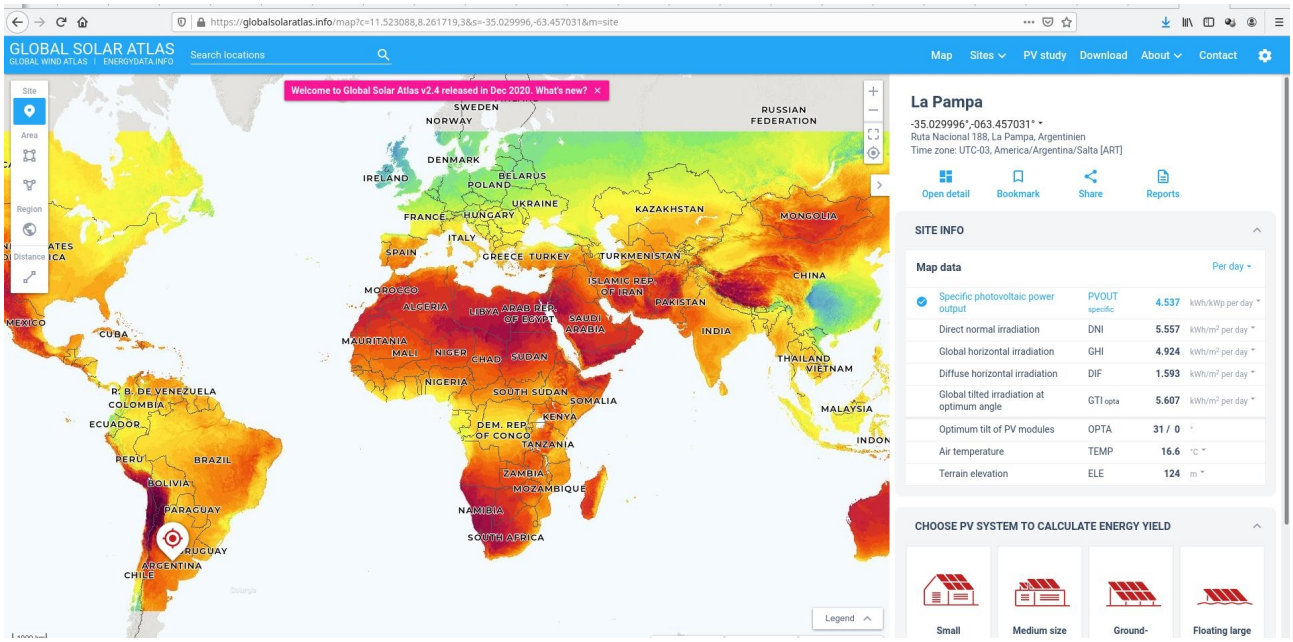
How many Watt solar power?

The required amount of solar power has to be calculated for the month of the year with the lowest sun irradiation. A good resource for sun irradiation is:

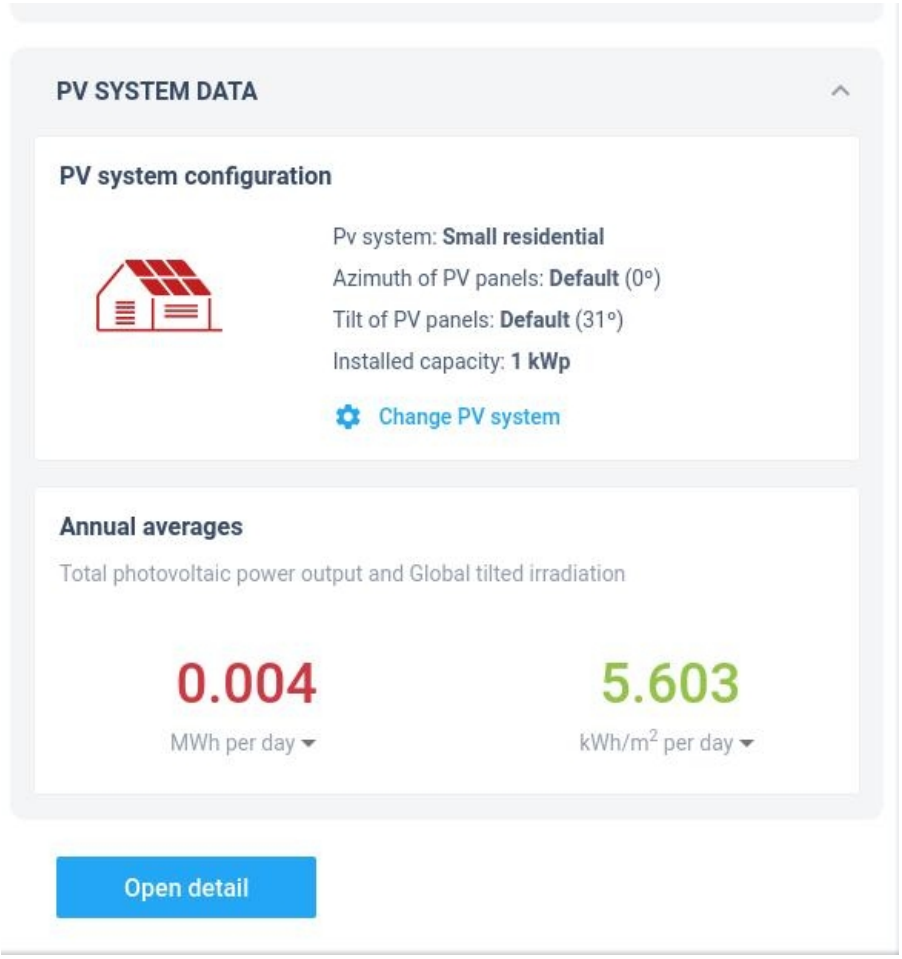
https://globalsolaratlas.info/

Copyright disclaimer: Data/information/maps/screenshots in this document are obtained from the "Global Solar Atlas 2.0, a free, web-based application that is developed and operated by the company Solargis s.r.o. on behalf of the World Bank Group, utilizing Solargis data, with funding provided by the Energy Sector Management Assistance Program (ESMAP). For additional information: https://globalsolaratlas.info

I'm using the base of **Example 2** – installing a Libremesh router in La Pampa, Argentina

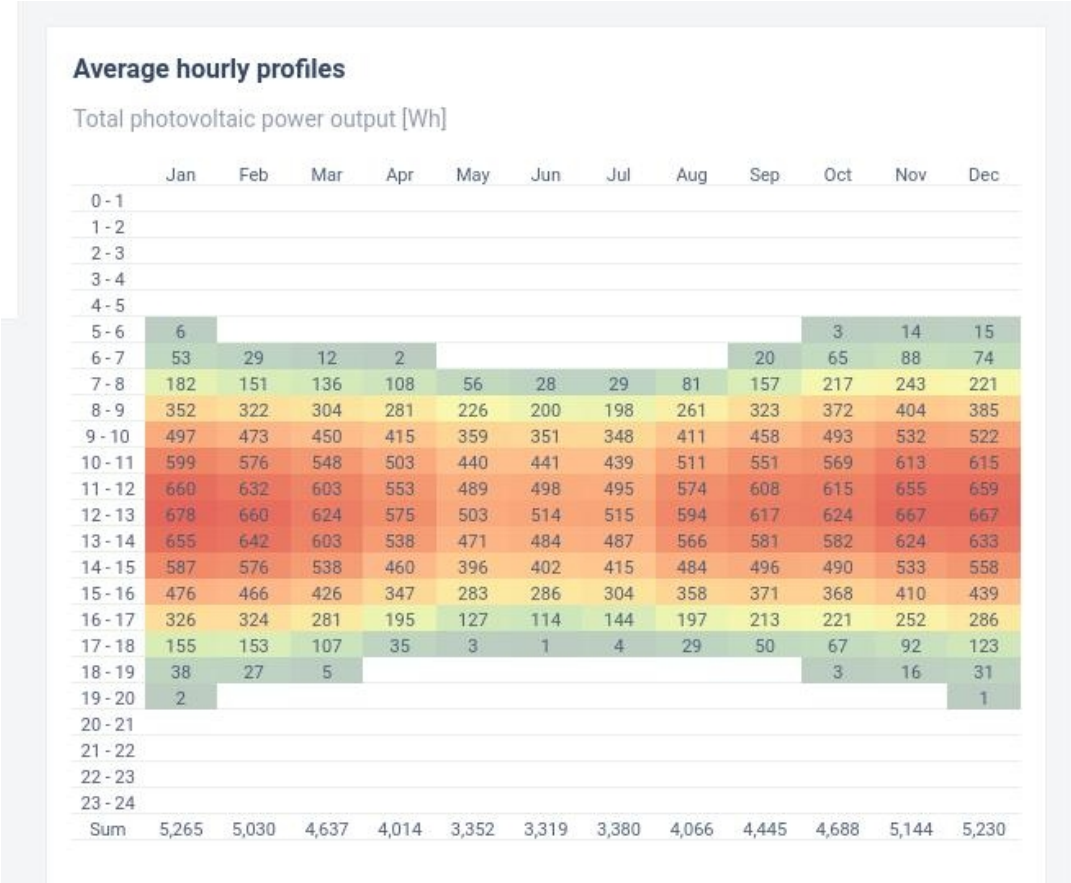


After picking the location, choose **Small residential** system.



Select “**Open detail**“.

This is what we are looking for:



Average hourly profiles

Total photovoltaic power output [Wh]

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0 - 1												
1 - 2												
2 - 3												
3 - 4												
4 - 5												
5 - 6	6									3	14	15
6 - 7	53	29	12	2					20	65	88	74
7 - 8	182	151	136	108	56	28	29	81	157	217	243	221
8 - 9	352	322	304	281	226	200	198	261	323	372	404	385
9 - 10	497	473	450	415	359	351	348	411	458	493	532	522
10 - 11	599	576	548	503	440	441	439	511	551	569	613	615
11 - 12	660	632	603	553	489	498	495	574	608	615	655	659
12 - 13	678	660	624	575	503	514	515	594	617	624	667	667
13 - 14	655	642	603	538	471	484	487	566	581	582	624	633
14 - 15	587	576	538	460	396	402	415	484	496	490	533	558
15 - 16	476	466	426	347	283	286	304	358	371	368	410	439
16 - 17	326	324	281	195	127	114	144	197	213	221	252	286
17 - 18	155	153	107	35	3	1	4	29	50	67	92	123
18 - 19	38	27	5							3	16	31
19 - 20	2											1
20 - 21												
21 - 22												
22 - 23												
23 - 24												
Sum	5,265	5,030	4,637	4,014	3,352	3,319	3,380	4,066	4,445	4,688	5,144	5,230

June is the month in La Pampa with the smallest irradiation of 3319 Watthours an average per day, if 1000 W_peak of solar power is installed. **Example 2** requires 178 Wh per day.

Required Solar power W_peak = (178 Wh / 3319 Wh) * 1000

Required Solar power W_peak = 53.6 Watt

It is unclear if the number contains the drop in efficiency due to temperature. There are also other losses like charge/discharge loss, cable loss, loss in the main fuse, dirt on the solar module, the actual mounting angle and so on. It should be also possible that a system starting with a discharged battery produces some energy surplus to get back to 100 % charge after a reasonable amount of time, even in the month with the lowest irradiation. Hence, I‘ll add 40 % safety margin. I‘m assuming that there is no shade on the panel from close objects in front of the panel.

Required Solar power W_peak = 53.6 Watt * 1.4

Required Solar power W_peak = 75.1 Watt

At 75 Watt peak the expected maximum charge current is 4.75 Ampere (25 % loss due to temperature and the DC/DC conversion loss of the MPP circuit). I have calculated 74 Ah battery capacity for 5 days of no sun at all. 4.75 Ampere maximum charge current is a maximum initial charge current of 0.064 CA, which is excellent.

CA_charge_max = 4.75 A / 74 Ah

CA_charge_max = 0.064

Same goes for the CA_discharge:

Instantaneous consumer current = 7.3 Watt / 12 V

Instantaneous consumer current = 0.61 A

CA_discharge = 0.61 A / 74 Ah

CA_discharge = 0.0082

Less than 1 % discharge per hour. It also wouldn‘t harm at all if a 100 W_peak panel is used instead of a 75 Watt panel.

Things to observe during hardware installation and operation of the solar system

- **Never operate** powerful batteries **without a circuit breaker (fuse)** in the cable that is connected to the positive terminal. Batteries do produce immense currents when shorted. This can result in burning cables that could possibly ignite a fire. Therefore an adequately rated master fuse must be present as a circuit breaker in the circuit close to the positive terminal at all times. If you accidently short cables or reverse the DC polarity, a properly rated fuse will interrupt the current and prevent/limit the damage to the FF-ESP32 and connected devices.

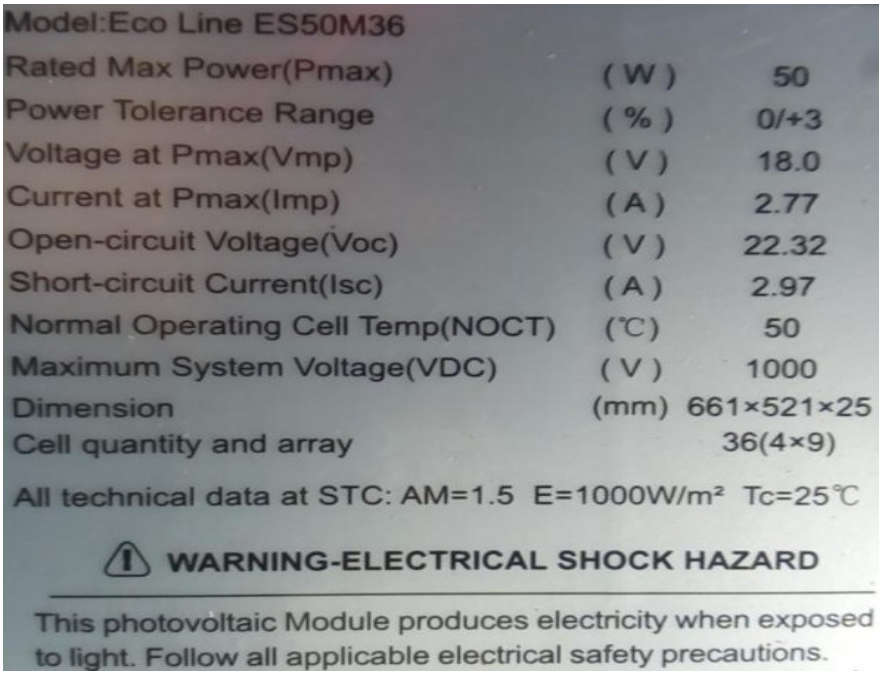
AGM 12 Volt battery with circuit breaker (blue) with fuse (red). Since both wires are black, the positive wire is marked with a node.

Safety precaution warnings on a 12 Volt AGM battery.

- Do not exceed a fuse rating of 8 Ampere.
- Do not “repair“ a broken fuse by bridging it with tin foil or wire.
- Do not replace the 8 Ampere fuse with a stronger one. You can always replace it with a smaller one, if it is sufficient for the expected current.
- If you are running a small system, it might be good idea to use a smaller rated fuse. For a 50 Watt module and consumers drawing less then 50 Watt, a master fuse rated at 4 Ampere is sufficient.

- Be careful when working on powerful batteries with metal tools. Batteries can explode if there is a massive short between their terminals. The worst possible battery accident in a workshop is a massive wrench that accidently touches both poles of a car or truck battery, causing a massive short. The wrench can rapidly heat up (glow!) and can weld with the battery terminals. If things have gone wrong up to this point – you realize that you can‘t touch the tool because it will burn your hand, while the tool causing the short cut is welding itself to the battery – **run!**

- The FF-ESP32 is designed to operate with solar panels with less than 25 Volt **Voc** (Open circuit voltage). Do not exceed 25 Volt open circuit voltage at the input. Solar modules with higher **Voc** voltage are not suitable for the FF-ESP32.



The label on the back side of a typical 50 W solar panel – suitable for operation with the FF-ESP32

- Maximum load current for the consumers is 8 Ampere.

- If the wires going to the load (consumers) are thinner than 1.5mm², they should be fused with a smaller fuse as well.

- The screw terminal connectors of the OpenMPPT allow up to 2.5mm² wire cross section.

- If you use the OpenMPPT with the black housing option, assemble it upright, so fresh air comes in from the connector side and leaves the housing at ventilation holes at the top.

- Do not cover the ventilation holes.

- Do not operate the device in excessively hot environment above 55 degrees Celsius to prevent overheating.

- Do not expose the device to direct sunlight. It should operate in the shade. A good place is behind the solar module, without direct contact to the back side of the solar module.

- If you have more than one solar panel, connect them in parallel i.e. connect both positive (+) and (+) terminals of the solar modules with each other. Connect both negative (–) and (–) terminals of the solar modules with each other.

- Both solar modules should have the same or similar specifications for U_mpp (voltage at maximum power point) and U_oc (idle voltage at open circuit). Otherwise the device can match only the maximum power point of one module and you are wasting energy from the second one.

- Do not exceed maximum peak solar power of 100 Watt in total. So one 100 Watt panel or two 50 Watt solar modules in parallel are the limit.

- There is no minimum amount of peak solar power, but less than 5 Watt peak solar power is probably too small for most applications.

- The OpenMPPT board consumes ~0.3 Watt in WiFi client mode and ~0.65 Watt in AP and AP-Client mode.

FAQ – Frequently Asked Questions

Can I host the ISEMS app dashboard myself?

Yes, of course. ISEMS is a dedicated open-source project, initially funded by the Prototype fund. The sources are here: ISEMS-Ansible

The repository contains ansible configuration management code for easier deployment of the ISEMS app(s).

The idea is that you use these scripts to install the isems-app (the static page app that is used to display data) and the isems-data-collector (the python application that collects data and makes it available as an api). You can even install it on a Raspberry Pi in your network.

Can I host a service like Mqtthub.net myself?

Yes, that should be possible as the system is a combination of open-source programs. MQTT for telemetry, Kotori as database and Graphana for the amazing looking graphs. Note that I am not involved with Mqtthub. Mqtthub is hosted by Andreas Motl, the main developer of Kotori. Andreas was kind enough to send MQTT patches for the FF-ESP32 firmware and added support for uploading ISEMS data.

Would it make sense to connect two solar panels – i.e. two 50 Watt panels with the same specifications – in series in order to reduce cable loss?

The FF-ESP32-OpenMPPT is designed for nominal input voltages up to 25 Volt. Putting the two panels in series would double the solar input voltage. Doing so would damage the FF-ESP32-OpenMPPT. *The absolute maximum input voltage of the FF-ESP32-OpenMPPT is 27 Volt DC.*

Besides: There is a general disadvantage of putting multiple solar modules in series. Lets assume our two modules are not exactly equal. One is good for 48 Watt peak, the other one for 52 Watt peak. If they are connected in series, the maximum power of both modules is 96 Watt peak. Why? A chain is as strong as its weakest link. The weakest panel determines the total output current.

There is yet another reason to not put solar modules in series on small solar systems in combination with the FF-ESP32-MPPT: Partial shading of solar modules. If the modules are in series, the total output of both modules is affected if there is a shade of any one of the two.

Would there be a problem using a car cigarette lighter power supply or other mains supply as charge source instead of a solar panel?

The FF-ESP32-OpenMPPT is not designed and programmed to do that. The Maximum Power Point Tracking hardware and software of the FF-ESP32-OpenMPPT is designed and programmed to handle a photovoltaic power source with the maximum power point characteristics of (silicon-based) solar panels only. If your application is a solar and mains powered UPS system, take a look at the next question.

How can we use the FF-ESP32-OpenMPPT together with intermittent mains power?

The simple solution is to add a current and voltage regulated power supply with 13.8 to 14.2 Volt output. The system requires a battery and a solar panel. The FF-ESP32-OpenMPPT and the mains supply are connected to the battery terminals in parallel. The consumer / load is connected to the FF-ESP32-OpenMPPT load output.

Note: The mains supply needs to have a regulated current limit build in (so not a unregulated supply with a simple circuit breaker or overcurrent turn off feature). Suitable power supplies are automatic battery chargers or adjustable laboratory power supplies. However: Some automatic battery chargers lack an autostart feature, hence they need manual user interaction to start/continue charging every time the power comes back. These are basically useless for this type of application.

So I‘d recommend a cheap laboratory power supply with adjustable current and voltage output.

The battery must be powerful enough to take the currents from both supplies added together. The maximum initial charge current limit of the battery specification data sheet must be observed.

This way, the power for the system comes from the sun during the day and from the mains supply at night. If there is an outage, it will come from battery and solar alone. The result is an excellent solar backed up and energy saving UPS. If you skip the solar panel, the FF-ESP32-OpenMPPT will operate as a low voltage disconnect switch or Wifi remote switch for the load plus battery monitoring device. But there is no point in using the FF-ESP32-OpenMPPT without solar panels, since the Solar Maxium Power Point Tracking function is one of its main features ;)

What are the units for the 'Status' value in shown in Graphana at https://isems.mqtthub.net/grafana/d/UsLo8SsZk/isems-testdrive? It seems to range from 300-900.

Status value is a Hex code that encodes system status and errors in individual bits. The ISEMS app does understand and analyze it for reports in the node status messages and warnings. Mqtthub.net doesn‘t have the software to analyze the hex values, so the grap does just show the value as such and its changes over time. This can still be useful for debugging, as you can see the value change over time. If you want to learn more about the Hex code, take a look at the ISEMS hex bit code section in the ISEMS documentation: https://www.isems.de/documentation/codes/

Adding hardware extensions and sensors

The FF-ESP32-OpenMPPT features 14 exposed GPIO pins on three pin headers that can be used for adding extensions.

Pin header P1 is used for debugging via serial terminal, programming and firmware flashing.

It features 2 additional GPIOs #21 and #22:

Ground	**GND**	**IO0**	GPIO0 Pulling this pin to Ground at boot time enables firmware flashing mode. ADC input port: ADC2_CH1
Ground	**GND**	**IO21**	GPIO 21. Input/Output
Ground	**GND**	**IO22**	GPIO 22. Input/Output
Enable pin. Pulling this pin down to Ground will turn off the ESP32 chip.	**EN**	**RX**	First serial port RX pin. 3.3V TTL level. UART0_RX, GPIO3
3.3 Volt rail (nominal, actually 3.05 V)	**3V3**	**TX**	First serial port TX pin. 3.3V TTL level. UART0_TX, GPIO1

TempSens P2 is prepared and used as an analog input with 2k2 Ohm pull-up resistor to Vcc rail for the analog temperature sensors KTY81-210 or KTY81-220.

GPIO32, ADC1_CH4, Input/Output	**+ TempSens**	**GND**	Ground

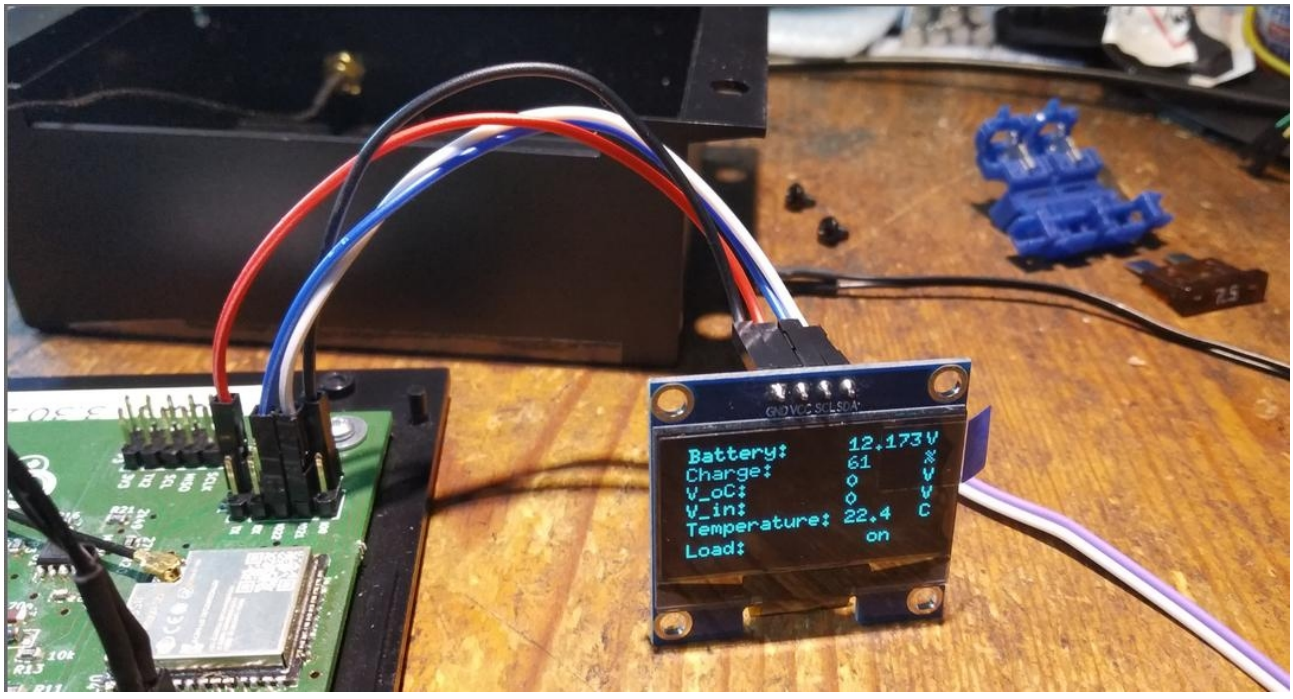
Pin header P3 uses the same pin layout as the ‘Olimex‘ brand UEXT header. All pins, except the power pins 3V3 and GND, are GPIOs that can be used for other purposes, as well.

SPI Slave select pin. GPIO5, Input/Output	**SSEL**	**SCLK**	SPI clock pin GPIO18, Input/Output
SPI Master out, Slave in. GPIO23, Input/Output	**MOSI**	**MISO**	SPI Master in, Slave out. GPIO19, Input/Output
I²C SDA pin. GPIO15, Input/Output	**SDA**	**SCL**	I²C SCL pin, GPIO2, Input/Output, ADC2_CH2
Second serial port RX pin. 3.3V logic level. GPIO16, Input/Output	**RX2**	**TX2**	2nd serial port TX pin. 3.3V logic level. GPIO17, Input/Output
Ground	**GND**	**3V3**	3.3 Volt rail (nominal, actually 3.05 V)

Adding a SSD1306 I²C OLED display

The FF-ESP32 firmware is pre-configured to connect to a generic 128x64 pixel OLED display module with SSD1306 controller via I²C interface. Connecting the display is simple.

Note: DC power has a polarity. Accidently reversing the positive and negative terminal (3.3 Volt Vcc and Ground) will damage the display immediately.



Disconnect power (battery and solar) from the FF-ESP32-OpenMPPT.

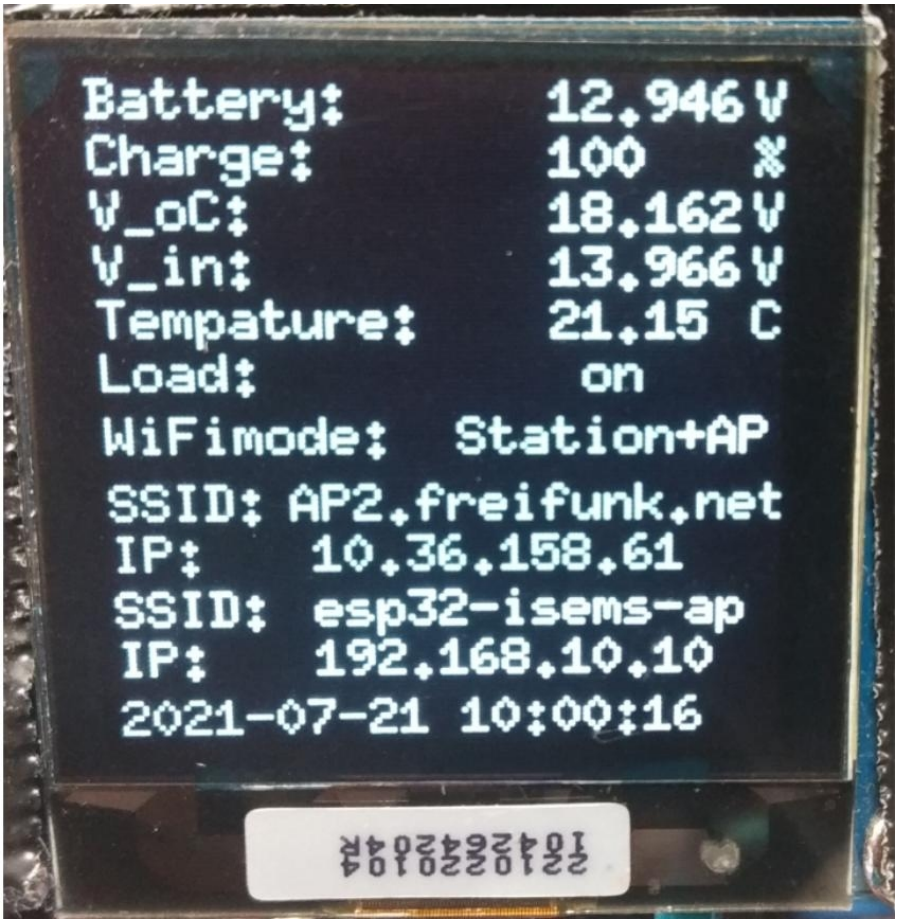
The wiring diagram is as follows:

Display FF-ESP32 connector **P1**
GND ———— GND (black)
VCC ———— 3V3 (red)
SCL ———— IO22 (blue)
SDA ———— IO21 (white)

Your color scheme may vary. Use whatever wire colors you have at hand. Sticking to the color scheme of black cable for Ground and red cable for Vcc/3V3 is recommended, though.

Note: The display contains 8192 small organic LEDs that can be turned on or off individually. Unlike non-organic LED light sources they are susceptible to wear and have a limited life-time. If you keep certain pixels shine for a long time, you will notice that their intensity gets dimmer than the others over time. Energy consumption of the display (albeit very low) also depends on the number of LEDs that are shining at a time.

Adding a SSD1327 SPI OLED display with 128x128 pixels



Support for this 1.5“ SPI OLED „Waveshare“ display module is build in, too, but it is not activated by default.

Software preperation

Two software changes are necessary to change firmware support from the I²C SSD1306-based display to the larger SPI SSD1327-based 128x128 display.

1/ Edit the file *init.lua*, add two dashes at the beginning of the first line, and remove the two dashes from the second line. Like so:

```
--dofile"SSD1306.lua"
dofile"SSD1327-waveshare-128x128.lua"
```

Leave everything else in *init.lua* untouched and upload the modified file to the FF-ESP32.

2/ Rename the file *display128x128.lua* to *display.lua* and load it up to the FF-ESP32. This will replace the previous version for 128x64 pixels display size.

Hardware connection

Disconnect power (battery and solar) from the FF-ESP32-OpenMPPT.

The Waveshare 1.5 inch OLED display module is shipped with a color-encoded cable assembly.

The SPI bus is available on pin header P3 of the FF-ESP32-OpenMPPT. The wiring diagram is as follows:

Display FF-ESP32 connector **P3**
VCC ———— 3V3 (red)
GND ———— GND (black)
DIN ———— MOSI (blue)
CLK ———— SCLK (yellow)

CS ———— SSEL (orange)
DC ———— RX2 (green)
RST ———— TX2 (white)

Ready to go…

Direction, tilting angle and other considerations when mounting solar panels

Direction

Choosing the right direction for the solar panel assembly is the simple part. In the northern and southern hemisphere, the flat side of the solar panels should face straight at the sun at noon time, when the sun stands at the highest point of its apex. Hence the solar panel(s) should look to the north in the south and to the south in the north.

Tilting angle

When considering tilting angle, matters are getting a bit more complex. A solar calculator like https://globalsolaratlas.info/ may suggest the optimal tilting angle for the entire year. The optimal tilting angle changes throughout the year, if the location of the solar installation is not at or close to the equator. A solar calculator will typically suggest a flat tilting angle for the summer and a more steep tilting angle for the winter time. Hence, in order to achieve the maximum power output of the solar system throughout the year, we would have to change the tilting angle a few times. However, if the application of the solar system is to provide steady power for a wireless infrastructure unit throughout the year without any interruptions, the optimal tilting angle is the optimal tilting angle at winter time. Throughout summer, spring and autumn, a properly designed system has a decent surplus of power for these three seasons. In a climate zone like central Europe, for example, the amount of solar energy available in winter is 10 times less than in summer.

Cleaning and self-cleaning

It the system is mounted at a remote, hard-to-get-there place, considering self-cleaning may be critical. Dust, leaves, bird poo and other dirt are best washed off naturally on their own if solar panels are mounted at a steep tilting angle.

Snow piling up in front of the panel

If stuff slides down from the lower edge of the solar panel, there should be enough free space in front and below, so whatever slides down doesn‘t pile up in front of the solar panel.

Partial shading from obstructing objects – a real bummer!

Basically, obstructing objects in front of solar panels are a no-go. The shade of a tower, a pole, leaves, trees, etc moving over the front of the solar panel during the day is a real-show-stopper.

A chain of solar cells can only produce as much current (in Ampere) in total as the cell with the most shade on it, since a chain is only as strong as the weakest limb. Shade on one or more solar cells in a solar panel are weakening the energy production of the entire solar string considerably. If just one solar cell in a solar panel has 90% shade on it, the entire solar panel will deliver only ~10% of the current that it would otherwise produce. It doesn‘t help at all if all the other solar cells are 100% exposed to the sun. The power output will be 10% of what the panel would produce without the shade on the single cell.

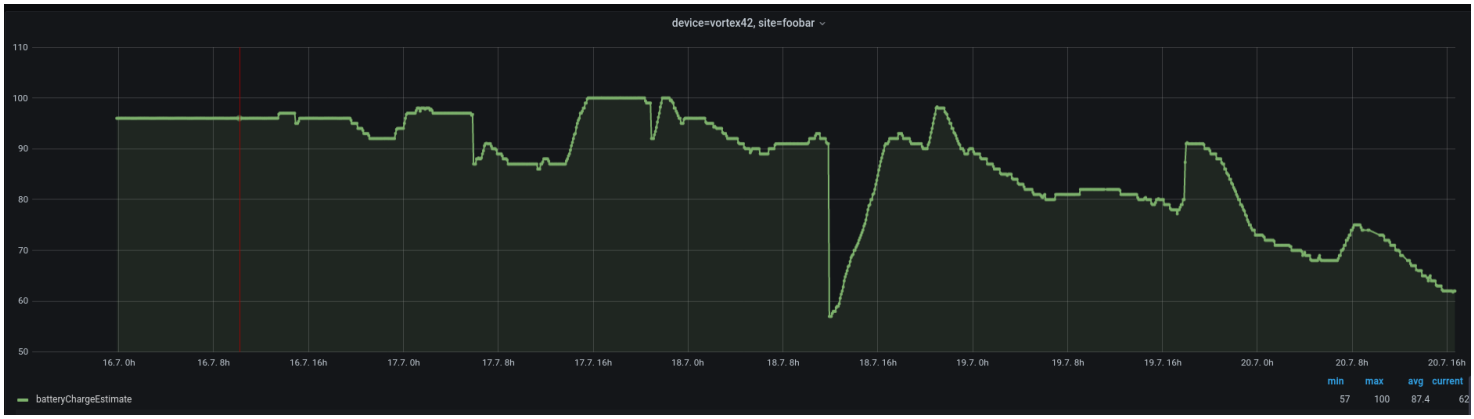
Therefore, when considering where to mount a solar module, shade from obstructing objects should be avoided as far as possible.

It is important to keep the battery and temperature sensor cool!

Lead batteries should not be charged above +40 degrees Celsius in order to avoid quick premature damage due to loss of electrolyte. The FF-ESP32-OpenMPPT stops charging the battery if the temperature sensor detects a battery temperature above +40 degrees. The same applies to batteries with other active materials (Lithium for example), by the way.

1) The temperature sensor should be attached to the battery with good quality adhesive tape.
2) The battery and the temperature sensor should be located at a cool place in the shade and not exposed to the sun.

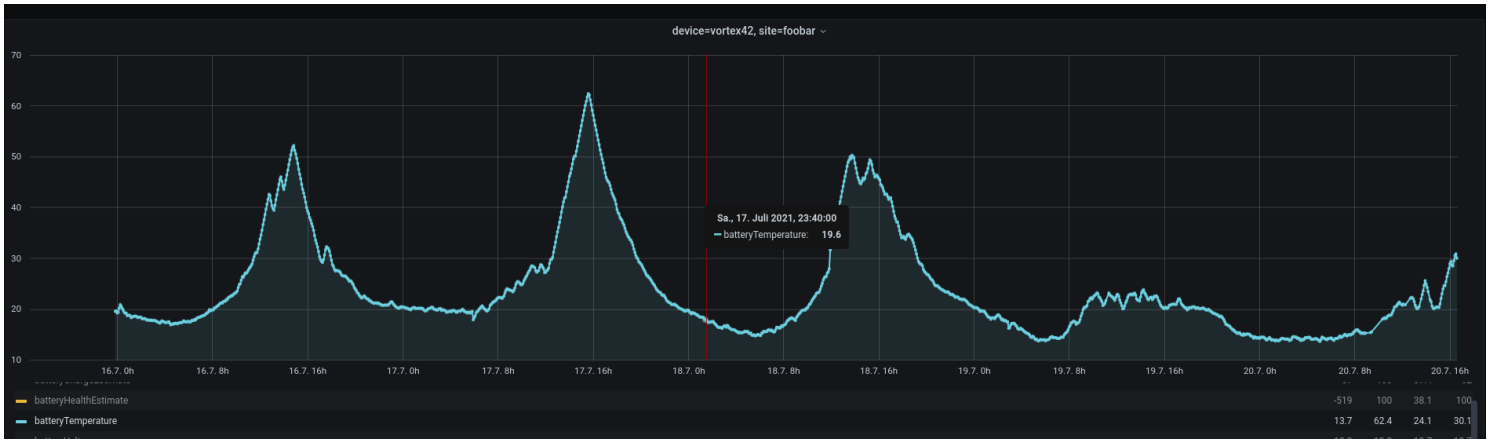
Here is an example of what happens if battery and / or temperature sensor are exposed to intense heat. Even though the sun is providing plenty of power in July in Central Europe, the system keeps discharging the battery over time.



After a few days in the hot summer the battery charge has dropped to 62 %.

By the way: The sharp drop in charge estimate at 16.7. at 12:00 h is caused by a system reboot. The battery charge estimation algorithm takes a while to provide a good estimate after a cold start.

The problem is excessive heat, detected by the temperature sensor. The following graph from the temperature sensor shows more than 60 degrees Celsius on 17.7. around 16h and three days with more than 40 degrees Celsius each.



The charging algorithm stops the charging process to protect the battery. The following graph shows the targeted temperature controlled charge end voltage, which is inverse to the temperature.

The algorithm avoids charging and discharging while the battery is overheated. The power output of the FF-ESP32 is regulated down to 13.1 Volt. At this voltage level, neither charging or discharging of the battery occurs, given that the instantaneous power provided by the solar module(s) is sufficient to power the consumers.

Despite the battery being protected from damage by overheating, it would probably still discharge continuously during the following days at such high temperatures in this example. At the point when the battery voltage drops below 11.9 Volt (30% state of charge – SoC), the low-voltage-disconnect routine disables the load and the FF-ESP32 goes into standby mode.

Low voltage disconnect and system recovery from a drained battery

If the battery voltage drops below 11.9 Volt – about 30 % state of charge in a typical solar system application – the FF-ESP32 disables the load and the µC puts itself in standby mode for 60 seconds. After 60 seconds the µC will wake up and check the battery voltage. If the battery is still below 11.9 Volt, it will go back to sleep again.

While this sleeping loop routine is going on, the analog part of the Maximum Power Point Tracking circuit will operate with a fixed low MPP point and continue to harvest energy from the solar panel(s) until the battery voltage exceeds 11.9 Volt again.

Above 11.9 Volt battery voltage, the µC takes over control of the analog MPPT again, sets the Maximum Power Point Tracking value and restarts the WiFi radio (if it is enabled in the configuration file config.lua).

As soon as the battery voltage level is equal or greater 12.3 Volt, the low voltage disconnect output is enabled again.

Routing

Adding routing entries

Example:

net.route.add{dest='10.250.250.0',prefixlen=24,nexthop='192.168.8.1',iface=0}

Destination (dest) and nexthop are provided as IP address string each (note the straight apostrophes at the beginning and the end required for entering strings in Lua), the prefixlen is provided as an integer (hence no apostrophes) or predefined names.

The network interface can be provided either as integer

0 = WiFi station interface

1 = WiFi soft AP interface

or this way:

net.IF_WIFI_STA

net.IF_WIFI_AP

Example:

net.route.add{dest='10.250.250.0',prefixlen=24,nexthop='192.168.8.1',iface=net.IF_WIFI_STA}

does the same as the first example.

Deleting a routing entry

Example:

net.route.delete{dest='10.250.250.0',prefixlen=24,nexthop='192.168.8.1',iface=net.IF_WIFI_STA}

Showing the content of the routing table

Executing the Lua program **ipr.lua** shows all routing entries in the routing table.

Syntax:

dofile"ipr.lua"

Example:

> dofile"ipr.lua"

Output:

dest=10.250.251.0 nexthop=192.168.8.1 prefixlen=24 iface=0

dest=10.250.250.0 nexthop=192.168.8.1 prefixlen=24 iface=0

ESP-Tree 'Mesh' protocol

Full featured mesh networking functionality requires that the WiFi radio is able of multipoint-to-multipoint communication. This requires the proper implementation of 802.11 ad-hoc mode or 802.11s – IEEE’s official WiFi mesh standard. Unfortunately, the ESP32 WiFi driver does not implement ad-hoc or 802.11s. However, since the ESP32 WiFi can do Software-Accesspoint and Client mode at the same time, a network topology that looks like a tree with branches and sub-branches and so on can be implemented.

Actually, large sections of real-life mesh networks in community networks actually resemble tree structures. So, while not being the real deal, the ESP-Tree 'Mesh' protocol can be useful to implement relatively large networks with the ESP32 that can be used to provide energy-autonomous Internet access at very low cost and power consumption.

If this doesn’t satisfy the demand, a more powerful WiFi device (OpenWRT based router) can be powered with solar energy from the FF-ESP32.

[This chapter is a work in progress since the ESP-Tree protocol is not yet ready for productive use. Please head on to the next page ;]

Flashing a new or customized FreeRTOS image /NodeMCU operating system

Instructions for flashing the ESP32 from a PC/Laptop

- Install **esptool** on your PC/Laptop. In most Linux distributions it is available from the package management system.
- Disconnect the FF-ESP32-OpenMPPT from all power sources for at least 45 seconds.
- Wire pin **IO0** on Pin header P1 to one of the **GND** pins with a female/female jumper cable.
- Make these connections between the USB to serial dongle and the FF-ESP32-OpenMPPT board:

  TX to RX

  RX to TX

  GND to GND

- Power the FF-ESP32-OpenMPPT board up again. Do not connect the 3.3 Volt pin of the USB TTL dongle to power the 3.3 Volt rail of the FF-ESP32-OpenMPPT board. This can cause trouble, since the USB TTL dongle might enter a brown-out (non-responing) state while trying to charge the powerful internal capacitors of the FF-ESP32-OpenMPPT.
- Change into the directory where all the necessary firmware **.bin** files (**bootloader.bin, NodeMCU.bin, partitions.bin**) are located.
- Copy and paste the following line to the command line of your terminal program:

```
esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 115200
--before default_reset --after hard_reset write_flash -z --
flash_mode dio --flash_freq 40m --flash_size detect 0x1000
bootloader.bin 0x10000 NodeMCU.bin 0x8000 partitions.bin
```

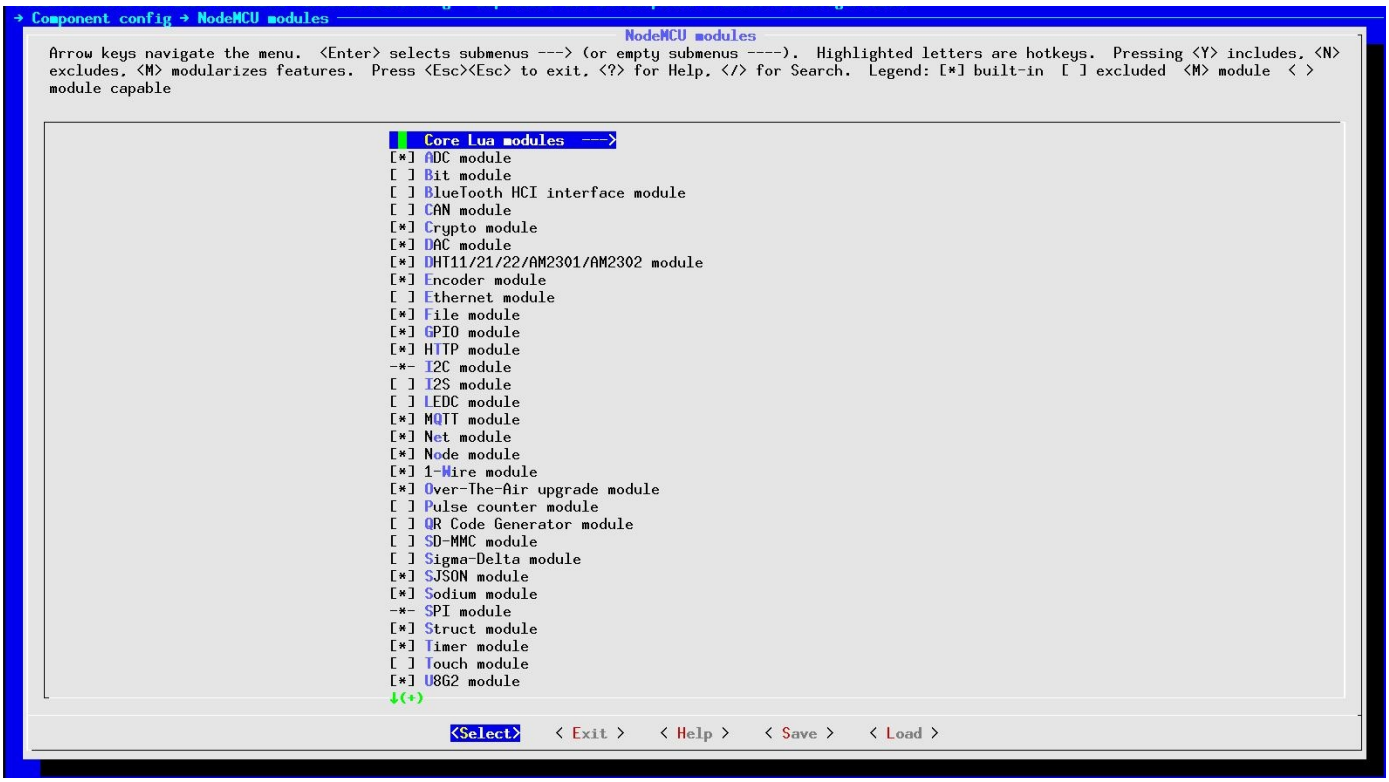
- Press **Enter**
- After the process is completed, remove the jumper wire from pin **IO0** to one of the **GND** pins.
- Power-cycle or reset the ESP32 chip. To do a power-cycle, disconnect the FF-ESP32-OpenMPPT from all power sources for at least 45 seconds **or** briefly connect the **EN**(**able**) pin to one of the **GND** pins to reset the ESP32. After removing the connection between **EN**(**able**) and **GND**, the device will reboot.

  The second procedure is quicker, more reliable and therefore recommended ;)

Building a new or customized FreeRTOS image /NodeMCU operating system

So far, this document has been trying to show how to use and install the FF-ESP32-OpenMPPT device, customize settings, add or change LUA programs and so on. For most users this might be already more than they ever need. However, if one wants to customize the system for a specific application, one might need to dig deeper and head on into the basement of its operating system. Because the RAM / storage resources of the ESP32 microcontroller are limited, not all available NodeMCU modules are included in the FF-ESP32-OpenMPPT firmware.

For example: A certain FreeRTOS/NodeMCU-firmware module required for a certain type of environmental sensor in your hardware project might not be included in the default FreeRTOS/NodeMCU-firmware that the FF-ESP32-OpenMPPT has been shipped with. If the required module/s is/are already available in the nodemcu-firmware system, one can select it from the ncurses-based menu system of the nodemcu-firmware environment and rebuild /reflash the firmware files. The firmware build configuration menu can be started from a terminal interface inside the nodemcu-firmware folder with the usual “make menuconfig“ command. Here is how it looks like:



If you notice that one or more NodeMCU modules are missing in the default firmware you might want to ask someone that has the nodemcu-firmware build environment installed to make a custom firmware with the additional module(s) for you, so you can flash it with esptool.

It might also be worth your time to set up the NodeMCU build environment locally. I recommend to take this route, because it gives you the greatest level of flexibility, insights and options. If you are an experienced developer, this is a piece of cake and not intimidating at all, of course ;)

The starting point for installing the nodemcu-firmware build environment (and its depencencies ;) is the NodeMCU documentation here:

https://nodemcu.readthedocs.io/en/dev-esp32/build/

Note: In order to build a custom FF-ESP32-firmware with IPv4 routing table support use these GIT repositories and branches as the starting point:

https://github.com/dh1df/esp-idf/tree/routing

https://github.com/dh1df/nodemcu-firmware/tree/dev-esp32

https://github.com/dh1df/esp-lwip/tree/routing

by Martin Schaller

or:

https://github.com/ISEMS/esp-idf/tree/routing

https://github.com/ISEMS/nodemcu-firmware/tree/dev-esp32

https://github.com/ISEMS/esp-lwip/tree/routing