Uninterruptible, monitored **Solar** Power Supply for Raspberry Pi

Title and Description:

A solar- UPS the other way round.
A Raspberry Pi may draw only as little energy as 6W, but it does it 24/24/365.

After one year of continuous operation, it sums up to 52KWh.

In this project the Raspberry Pi powered by a 40W solar panel, a simple PWM solar controller and 4x LiFePO 10Ah cells

At first sight, one may think 40W solar is overkill to power a 6 W Raspberry Pi?
Not at all, we will see later!

Regular solar chargers cut the load when the battery is too low. But we want to run the Raspberry Pi 24/24/365.
The prevent that, the solar charger is assisted by a 12V AC power supply that comes in operation when the battery is discharged. This is done by two 1A diodes.

To monitor the battery power supply, a cheap ESP8266 measures the battery voltage and forwards the value to a free on-line dashboard service

The solar power is sufficient during summer to power the Raspberry alone, only on upon a few heavily rainy days, we needed mains backup.
Now with winter approaching, we will need mains power more frequently, but anyhow even with a very small 40W panel and an extremely limited investment, the addition will be amortized in less than 3 years.

And -of course, should the mains power shut down for a short time, the battery takes over, providing UPS reliability.

Hardware Used:

- ESP8266 ($4)
- One resistor ($0.05)
- Two 1N4001 1a diodes ($0,20)
- 12V/5V buck controller to power the ESP ($1)
- Solar panel and PWM controller (50€)

Project Goals:

Being able to remotely monitor a solar charged battery without much cost, verifying the solar contribution.

This project uses an ESP and thinger.io to log values

The project intends to provide a “no frills” battery logging on a single ESP8366 device ($4).
You just don't need any additional hardware, nor computer, nor gateway, nor subscription.

Internet dashboard

For this sake we will use the freemium services of thinger.io-

You see a gorgeous dashboard worldwide on the Internet, can downloads historic data to Excel etc. Since solar operation is heavily dependent on the weather, the internet service OpenWeatherMaps is integrated in the dashboards.

Additionally, the devices have a Telnet console on the Raspberry Pi to control them and provide fall-back logs to a computer (optional, not required).

Implementation Steps:

Hardware.

Wire the solar panel to the solar controller the usual way, do not use the load output.

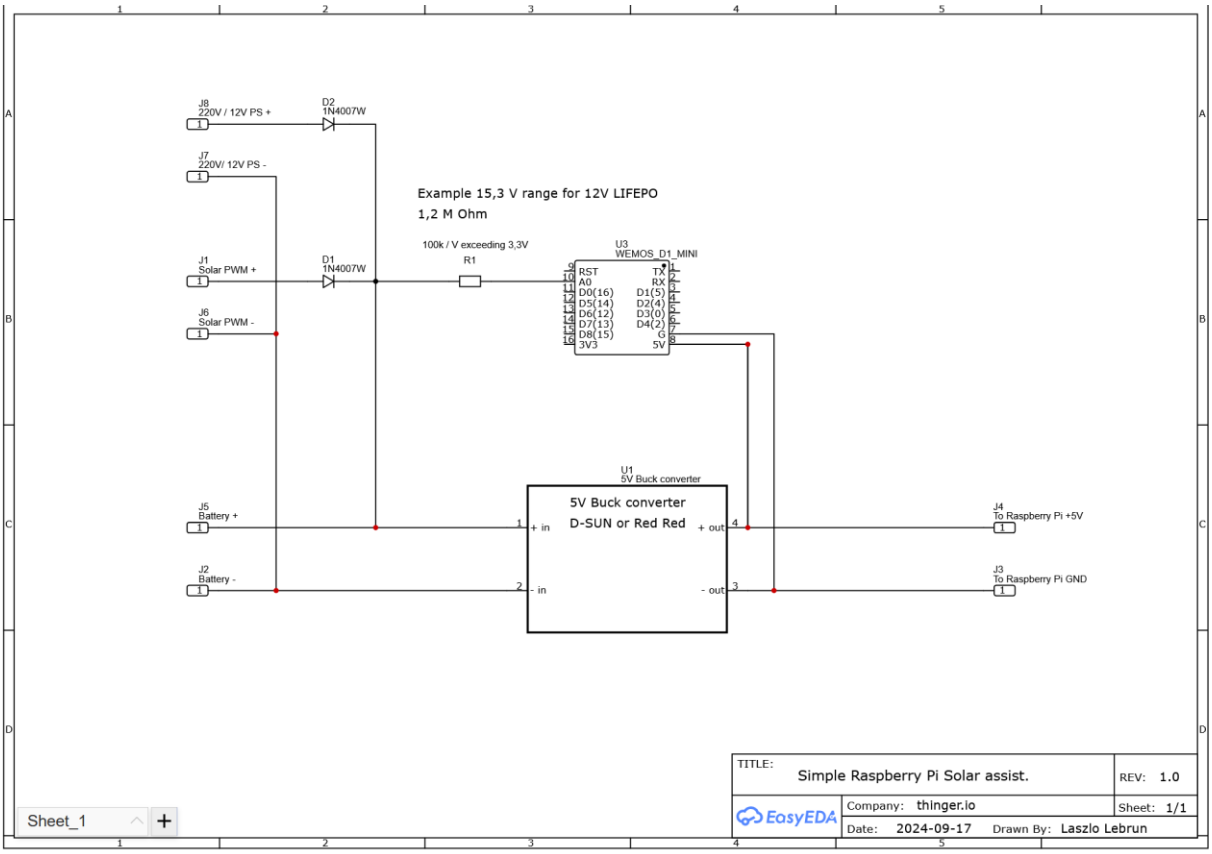


Figure 1 Schematic diagram

For monitoring, you just need a simple 100kΩ resistor to increase the A0 voltage range from 0..3,3V to 0..15,3V which is perfect to monitor a 12V LiFePo battery charged with a low cost PWM solar controller.
The buck converter will power the Raspberry Pi and the ESP8266 with uninterruptible 5V.

Software.

Get the code here:
https://github.com/rin67630/Victron_VE_on_Steroids/blob/main/Software/Binaries/RaspberryPi_Solar_UPS.bin

This is an already compiled binary, that you simply patch and upload using https://github.com/rin67630/ESP_Binary_patcher

Don't be afraid by the program complexity if you go to the sources: I am using a self-written generic multi-function software capable of many things, from which I use only a few features in that project.
If you want to compile yourself, let's use the content given in attachment as content for the config.h file:

Why not an MQTT-based open-source solution?

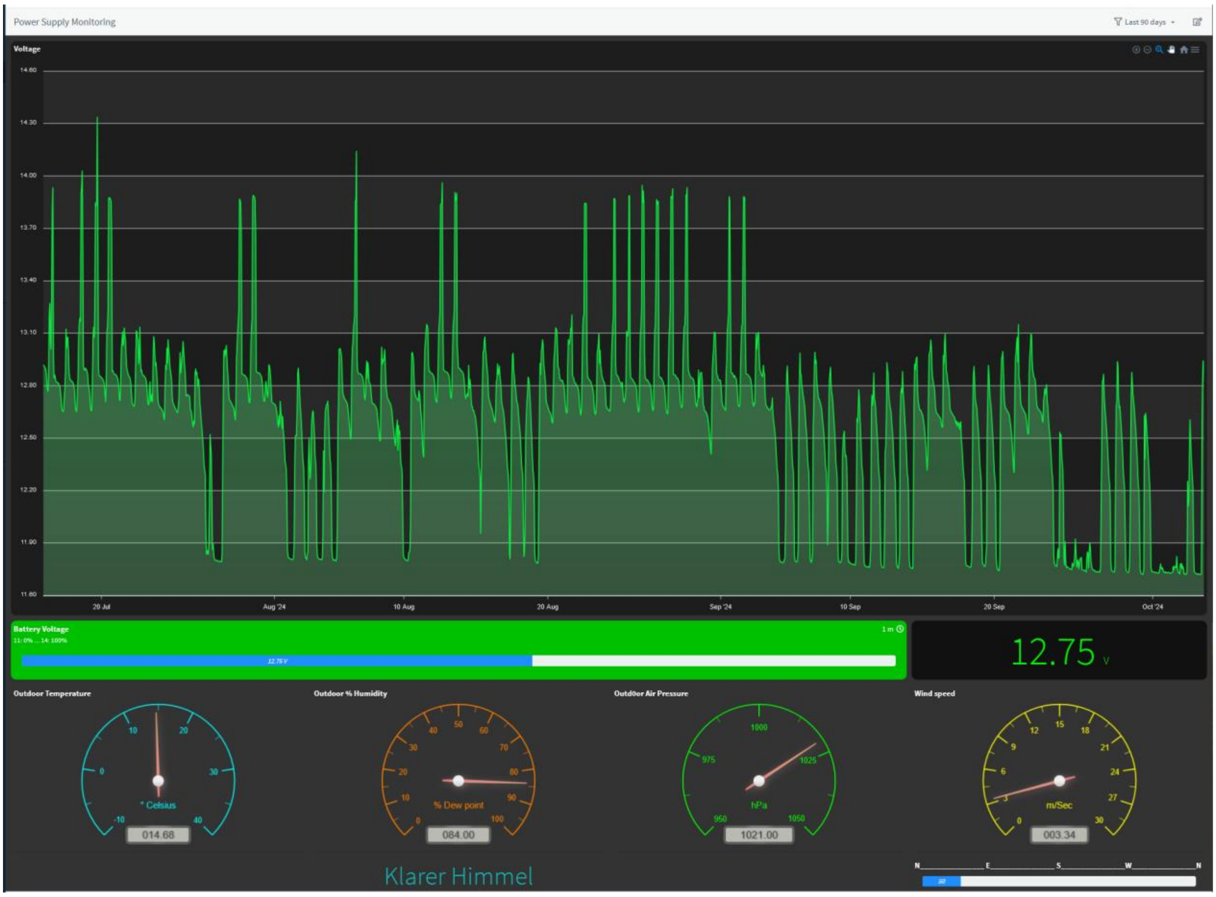
Open source solutions based on MQTT, Influx, Grafana exist and are popular, you may read here why I prefer thinger.io:
https://github.com/rin67630/Tasmota-thinger.io-bridge/blob/main/Documentation/Why_thinger.io-and_not_Open-Source.md

Dashboards

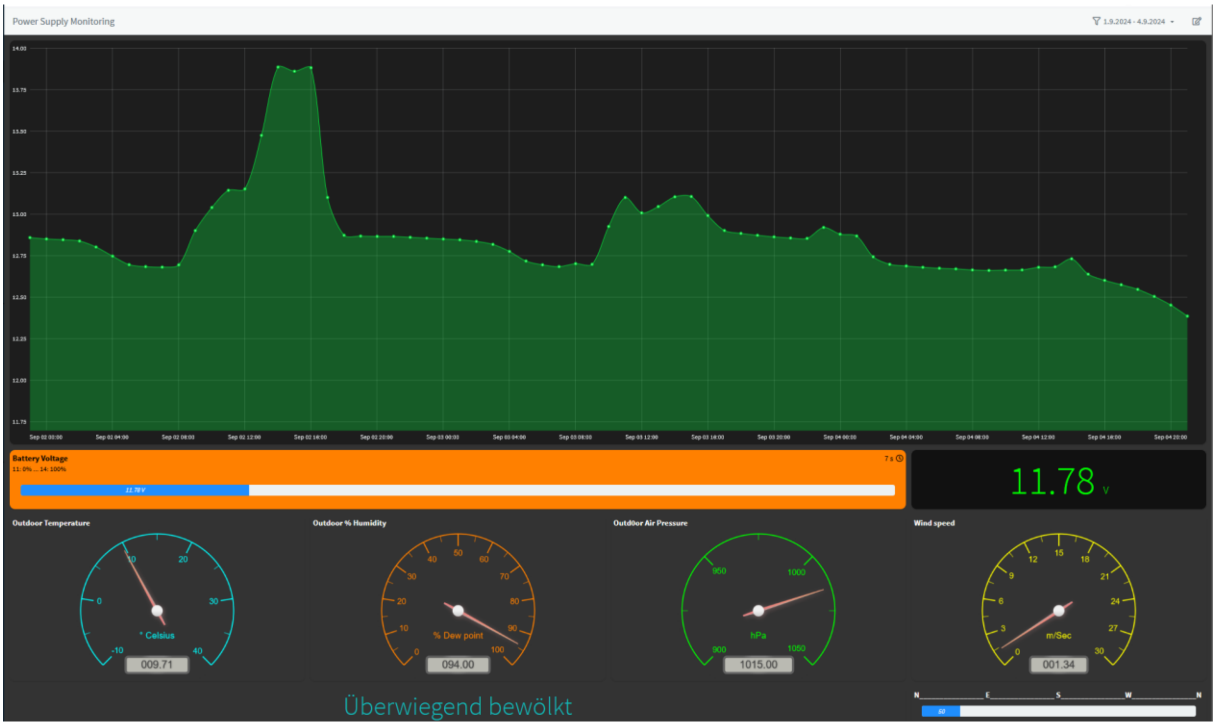
The dashboard's configuration is also provided as JSON file to be uploaded as Developer settings.

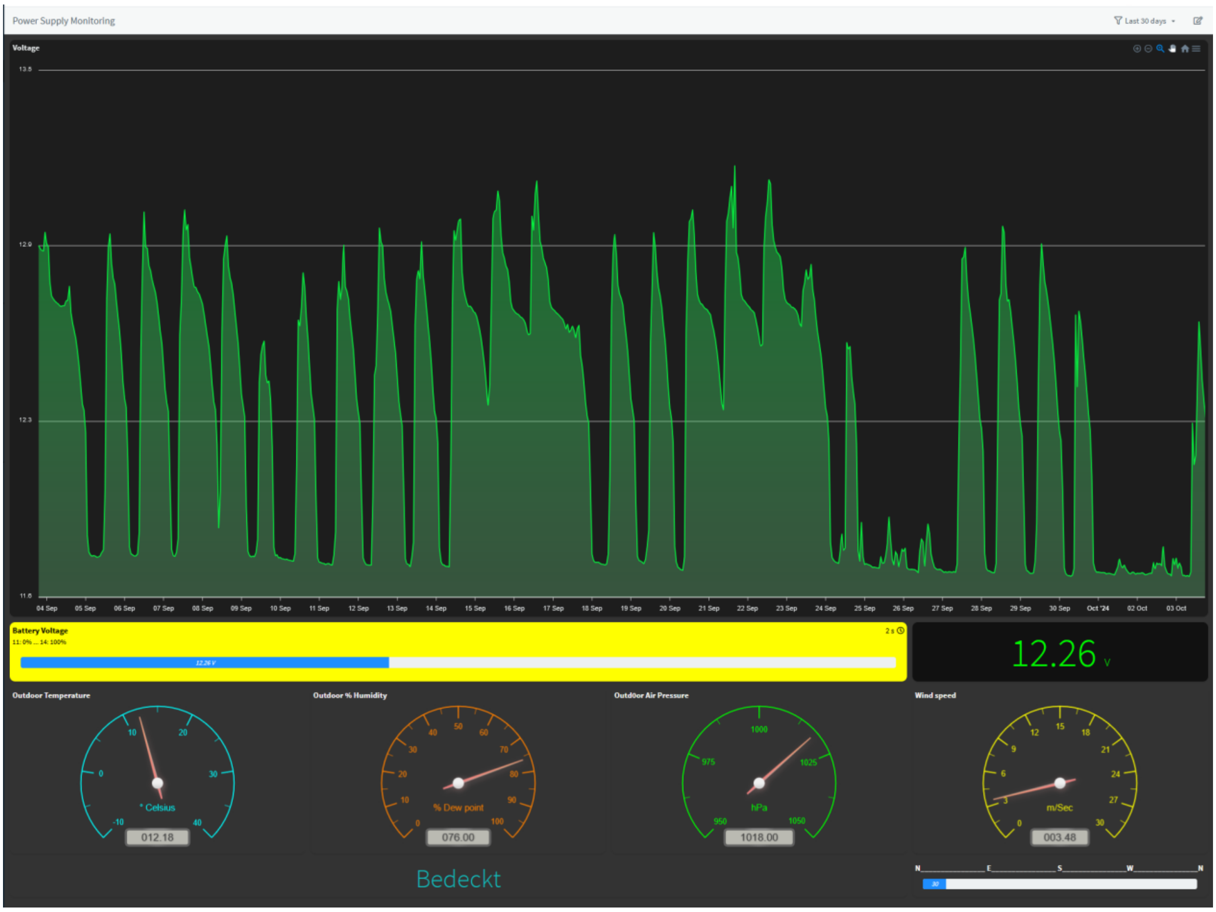
Screenshots or Videos:

90 days, with Apex Charts



4 days monitoring

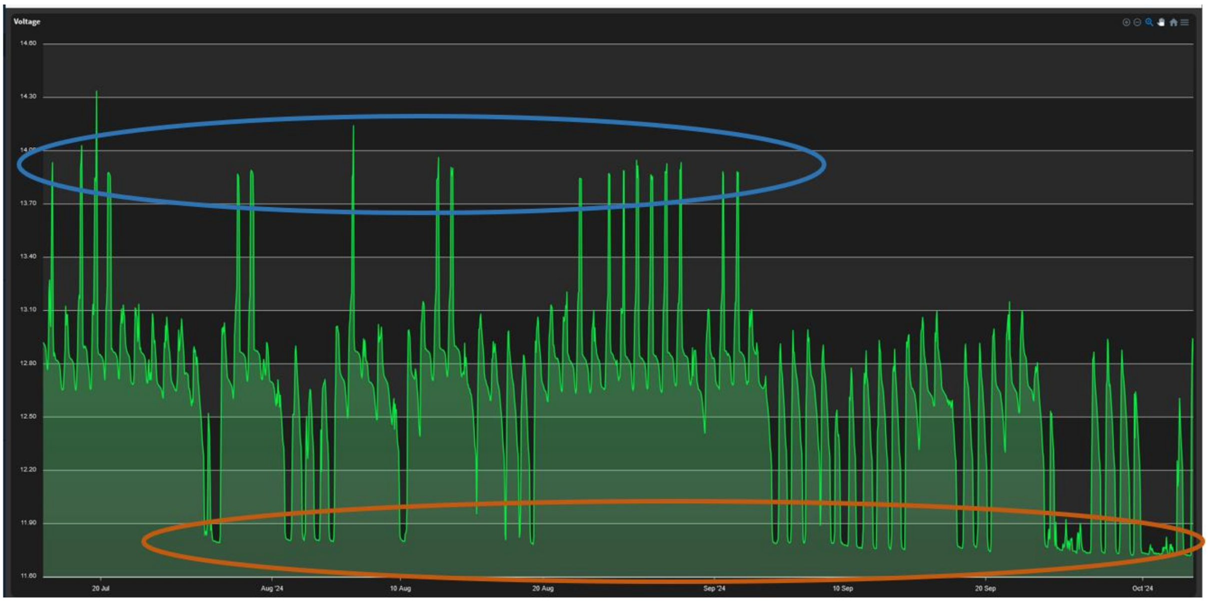




N.B: These high-res screenshots come from a 4k Monitor, on a lower resolution monitor, zoom back to get the widgets displayed correctly.

Conclusion

After ~90 days of solar operation in rather cloudy NRW, Germany, between July 15th and Oct 4th, the setup with the 40W solar panel, and 4x LiFePO 10Ah cells proved to be well balanced for a continuous operation of a Raspberry Pi running 24/24/365 and a SDR(Software Defined Radio) stick feeding continuously data to FR24.com.



The spikes (in the blue area) result from very sunny times, where the battery was fully charged and solar power is lost once the battery is completely charged.
The lower parts in the orange are times of over-casted weather, where (partial) mains backup was needed. These parts will increase in winter but, anyway, all the green surface is won energy.
Over 90% of the time the Raspbery Pi and it's SDR Radio was solar powered.
Even discharged to 11,7 V, the battery was able to take-over a simulated power outage of 1 hour, which practically never happens.

A full solar operation without mains back-up would have required a considerable larger panel of at least 400W and a battery of 100Ah, since long periods of over-casted short days in winter could not provide enough energy to power the Raspberry 24/24 <u>and</u> recharge the batteries within just 6 hours of daytime. With over-casted clouds, a solar panel just delivers between 5% and max 10% of its nominal power.

Table of Contents

Attachments

Config.h

<u>If you want to compile yourself your own version of the voltage logger</u>, then

- Create a Credentials.h file with your own credentials based on:

```
-    #define DEVICE_NAME               "DEVCNAME       "
-    #define WIFI_SSID                 "WIFISSID       "
-    #define WIFI_PASS                 "WIFIPASS               "
-    #define CLOUD_USERNAME            "CLOUDNAM       "
-    #define DEVICE_CREDENTIALS        "DEVCCRED       "
-    #define TZ_OFF                    "TZ_OFF  "        // Offset to GMT in secs (exactly 8  chars incl spaces)
-    #define DST_OFF                   "DST_OFF "        // Summer Offset in secs (exactly 8  chars incl spaces)
-    #define LONGTD                    "LONGTD  "        // Longitude             (exactly 8  chars incl spaces)
-    #define LATITD                    "LATITD  "        // Latitude              (exactly 8  chars incl spaces)
-    #define OPEN_WEATHER_MAP_APP_ID "enter here your own app-id at openweathermaps.org"
```

- place the following content in the Config.h file

```
// N.B. Compile sketch with following board settings:
//- for option 8266:            LolinWEMOSD1 (Clone)
//- for option TTGO:            TTGO T1
//- for option HELTEC LoRa:     HELTEC WiFi Lora32  (Not V2 !) from HELTEC, not generic
//----------------------- HARDWARE OPTIONS ---------------------------------
#define SCREEN_IS_NONE  // _NONE , _64x48 , _128x64, _TTGO, _HELTEC, _WEMOS32 _HW364A ;
#define SCREEN_IS_REVERSED // _IS_NORMAL, _IS_REVERSED To turn the display 180° if required
#define BRIGHTNESS 128     // PWM value for default brightness with TTGO 0=totally dark;255=totally shiny

//----------------------- SOFTWARE OPTIONS ---------------------------------
#define DASHBRD_IS_THINGER    // _NONE , _THINGER     // (Internet Dashboard)
#define GRACE_PAUSE           //Suspend Thinger processing for a grace pause, if the remote server takes too long to react, in
order to keep being reactive on menues
//#define STREAM_DATA         //Stream data to Thinger (instead of letting Thinger fetch the data when viewed)
#define TERM_IS_TELNET     // _TELNET, _SERIAL , _SOFTSER, _2SERIAL // defines where do Menus and Reports occur: _SERIAL and
D7_IS_VICTRON are mutually exclusive )
#define UDP_IS_NONE        // _NONE , _SEND , _RECEIVE // (UDP Inter-ESP Communication)
#define ESP_UDP_ADDR       "192.168.188.10"          // (IP of the receiving ESP)
#define ESP_UDP_PORT       4200  // (Port used to send/receive Values to other ESP)
#define COM_IS_NONE        // _NONE , _HOURLY          // Periodical reports to computer
#define COM_UDP_ADDR       "192.168.188.46"          // (IP of the receiving computer for night reports)
#define COM_UDP_PORT       4230                      // (Port used to send/receive Values to other computer)

//----------------------- MEASUREMENT OPTIONS--------------------------------
#define WEATHER_SOURCE_IS_OWM  // _NONE , _OWM , _BME680       // (Source of weather Information, URL in Credentials.h)
#define A0_IS_BATTERY          // _NONE , _BATTERY, _DOUBLEBATTERY , _HALFBATTERY, _PANEL, _POT, _SIMUL, _ACS712
#define A0_MAX              15500   // if A0 is used, define the value of the full range measure (mV or mA)
#define D0_IS_NONE          // _NONE ,

#define D5_IS_NONE          // _NONE , _RELAY2
#define D6_IS_NONE          // _NONE , _RELAY3, _VE_BLOCK, _VE_START_STOP
#define D8_IS_NONE          // _NONE , _DROK
#define D7_IS_NONE          // _NONE , _RELAY4; _VICTRON _DROK (DROK only ESP8266
//if _VICTRON or _DROK, MUST have a definition file see next two lines)
  //#include "MPPT_75_15.h" // Type of Victron controller (only relevant if D7_IS_VICTRON)
  //#include "J3806.h"      // Type of DC/DC controler (only relevant if D7_IS_DROK)
#define INA_IS_NONE        // _NONE , _226    when measures come from an INA226 power sensor  (Smart shunt scenario)
  //#include <SHUNT_1A_25mV.h> //Must be defined if INA_IS_226
#define ADS_IS_NONE         // _NONE , _1115   when measures come from an ADS1115 ADC converter (Drok type)
#define INA_VBUS_IS_FULL    // _FULL _HALF   (when high voltage schematic is used)
#define NUMBER_CELLS        4      // Number of cell in Battery
#define AH_CELLS            10    // AH of the battery
#define TYPE_IS_LIFEPO      // _LIFEPO, _LEAD, _LIION
#define POC_IS_TABLE        // _TABLE, _VICTRON   defines if POC is estmated from a table or comes from Victron
#define ESTIMATE_PANEL_POWER

//-------------------------- WiFi Options ------------------------------------
#define WIFI_REPEAT            500      //mS for retry
#define WIFI_MAXTRIES          5

#ifdef  ARDUINO_ARCH_ESP8266
#define WIFI_POWER             21.5  // from 0.5 to 21.5 full power (more current draw)
#else                                // ESP32
#define WIFI_POWER             WIFI_POWER_5dBm
#endif
/*
 Available ESP32 RF power parameters:
 WIFI_POWER_19_5dBm // 19.5dBm (19.5dBm output, highest supply current ~150mA)
 WIFI_POWER_19dBm // 19dBm
 WIFI_POWER_18_5dBm // 18.5dBm
 WIFI_POWER_17dBm // 17dBm
 WIFI_POWER_15dBm // 15dBm
 WIFI_POWER_13dBm // 13dBm
 WIFI_POWER_11dBm // 11dBm
 WIFI_POWER_8_5dBm // 8dBm
 WIFI_POWER_7dBm // 7dBm
 WIFI_POWER_5dBm // 5dBm
 WIFI_POWER_2dBm // 2dBm
 WIFI_POWER_MINUS_1dBm // -1dBm (For -1dBm output, lowest supply current ~120mA)
 Available ESP8266 RF power parameters: any value in 0.5 steps from
0    (for lowest RF power output, supply current ~ 70mA  to
20.5 (for highest RF power output, supply current ~ 80mA
 */
#define GRACE_PAUSE            //Suspend Network processing for a grace pause, if the remote server takes too long to react, in
order to keep being reactive on menues

//---------------------DO NOT EDIT until you know what you do ------------
#define SERIAL_SPEED      9600   // (Victron requires 19200)
```

```
#define LDR            A0
#define RELAY1         16    // D0  Relay or FET control 1
#define RELAY2         14    // D5  Relay or FET control 2
#define REDLED         15    // D8
#define BLULED         13    // D7 <-- Victron
#define GRNLED         12    // D6

#if defined(D7_IS_DROK) && defined(ARDUINO_ARCH_ESP32)
#error "D7_IS_DROK not possible on ESP32"
#endif

#if defined(D7_IS_DROK) && defined(ARDUINO_ARCH_ESP32)
#error "D7_IS_DROK not possible on ESP32"
#endif

#if defined (D7_IS_DROK) && not defined (buffsize)
#error "please include a hardware definition file e.g. MPPT_75_15.h"
#endif
```

Json Code for the dashboard to be pasted into the developer view:
https://github.com/rin67630/Victron_VE_on_Steroids/blob/main/Software/Binaries/RaspberryPi_Solar_UPS_Dashb.h

Go to your just created dashboard, enter configuration, then Settings, then Developer.
Remove everything in the {} Json field and paste there the content of the file downloaded.