LoRa Beacon
HW/SW Installation manual

Issue 1.0
Issue date: 07/06/2023

Author: Michele Fucito, *I8FUC*
Translation: Alfredo Vania *IZ7BOJ*

CONTENT INDEX

FIGURES INDEX

ACRONYMS

ACRONYM	Explanation
AP	Access Point
APRS	Automatic Position Reporting System
APRS-IS	APRS Internet Service
BT	Bluetooth
BW	Band Width
CPU	Central Processing Unit
CR	Coding Rate in case of Lora Protocol or Carriage Return
DHCP	Dynamic Host Configuration Protocol
EEPROM	Electrically Erasable Programmable Read-Only Memory
FRAM	Ferroelectric RAM
GPS	Glopal Positioning System
GUI	Graphic User Interface
HW	Hardware
IoT	Internet of Things
IP	Internet Protocol
KISS	Keep It simply Stupid protocol
LAN	Local Area Network
LF	Line Feed
LoRa	“Long Range” Radio Communication Technique
M2M	Machine to Machine
MQTT	Message Queuing Telemetry Transport Protocol
OLED	Organic Led (display technology)
OTA	Over The Air update
PCB	Printed Circuit Board
PPM	Parts Per Million
PPS	Pulse Per Second
RAM	Random Access Memory
RTC	Real Time Clock
RX	Receive
SDR	Software Defined Radio
SF	Spreading Factor
SMD	Surface Mounted Devices
SNR	Signal to Noise Ratio
SPI	Serial Peripheral Interface
SW	Software
TCXO	Temperature Compensated Crystal Oscillator
TFT	Thin-film transistor (display technology)
TX	Transmit
uC	MicroController

1 Introduction

This document describes the HW assembly and SW installation methods of the devices based on the LoRa_Beacon project developed in the context of the SARIMESH experimentation (ref. http://www.sarmesh.net).

The goal of this project is to generate an HW/SW platform that can be used for experimentation activities **relating to LoRa technology applied to radio amateur** and therefore not necessarily using the guidelines and implementations mandatory in the IoT (Internet of Things) applications.

It is worth remembering that LoRa technology was born mainly to allow the creation of "sensor" type devices or for M2M (Machine-To-Machine communication). This type of applications is characterized by a limited communication need in terms of quantity of transmitted data and by a maximum independence from external power sources, trying to use batteries to power the devices with an autonomy target of the order of years.

As a result of these functional requirements, the LoRa radio protocol has aimed to reduce the transmission power of the devices according to the available **"radio link budget"**; the key to achieve this objective is the use of a spread spectrum transmission method which allows to obtain a significant "process gain".

Therefore, in the context of the IoT, the reduction of energy consumption assumes enormous importance, considering their use as remote sensors not powered by external energy sources.

In amateur radio use, this aspect obviously takes a back seat, while the aspects of maximizing performance in terms of "available radio link budget" take on particular importance.
It was therefore decided in our experimentation to neglect, at least initially, the aspects of minimization of energy consumption, which are inessential for the purposes of our uses.

The target user has therefore been thought to be the average radio amateur eager to experiment with this new HW/SW technology with little expense and without particular skills.

The chosen solution is based on the reuse of HW and SW "functional blocks" already available on the market and on the internet.

2 Needed Hardware

The HW platform can be:
- SARIMESH Custom HW
- Classic "Chinese cards", currently widely used for LoRa applications and which still almost always mount first generation LoRa chips (es: Ttgo T-Beam)

The SARIMESH HW is based on the concept of mother and daughter boards, which are small printed circuits that can be purchased on the classic e-commerce platforms at very convenient prices.

The daughter boards have been designed with different "footprints" which cover the most popular lora modules available on the market.

The "Pin in Hole" technology has been used as much as possible, limiting the use of SMD components only to exceptional cases. Also this choice was motivated to make the assembly of the circuits as easy as possible, without any particular manual and assembly skill.

Unfortunately, the support of different HW platforms needs an economic commitment to obtain samples and a lot of time to carry out the non-regression tests.

With version 4.x we therefore decided to delegate the complete test of the SW on the TTGO boards due to the evidence that there are many different versions of this HW that are often quite different from each other. This does not mean that the SW cannot run on these devices. We tested these SW as best as possible on the samples we have available in our shacks, leaving the test on each specific device to the interested people, who will eventually be able to share their experiences via the GITHUB system.

The user can also build his own Hardware. In this case, it's possible to modify by GUI all the parameters related to the HW architecture, as the I/O pins used by the ESP32 for interfacing to various chips installed on the board.

A 3D-printable enclosure for both tracker and iGate in "Sarimesh HW version", is also available on GitHub.

If a TTGO T-Beam board is used, a lot of cases or 3D printing files are already available on the internet.

Figure 2-1 shows the tracker inside its 3D printed case.

Figure 2-1 : Tracker Sarimesh in a 3D printed case

At the end of this guide, the reader can also find a list of useful links for purchasing the HW online at good prices.

3 Needed Software

The source dode is available on Github, at this link:
https://github.com/IZ7BOJ/ESP32_LoRa_APRS_SARIMESH

From the SW point of view we tried to use a similar approach, using **"SW functional blocks"** already available in the open source environment, limiting new developments to the bare essentials.

A particular note in this regard is the programming style used: being the target a typical radio amateur, we tried to avoid "courtly" programming style and to write code easily understandable even by people who are not professional programmers. The goal was and remains to encourage even the inexperienced radio amaterur to experiment by his hand. The use of particularly advanced programming techniques is the exact opposite of what is needed to lower the entry threshold to this type of applications.

The SW aspect is obviously closely linked to the HW choice of a specific type of processor used: the choice fell on the use of the **ESP32 processor** which in the field of microcontrollers represents an optimal solution as it provides an extremely low cost platform, very powerful and capable of operating in "multiprocessing" mode in "real time", supported by a very light and efficient operating system (FreeRTOS) and supported by **PlatformIO,** which became one of the most used platform for this type of application and, moreover, extremely more advanced than Arduino, which was used during the early stages of development.

A very important aspect taken into account during the development, is the device management**:** also in this case, we have tried to avoid potential users having to necessarily go through the complete installation of the SW development environment (process typically required by projects based on microcontrollers) by providing a simple, easily customizable graphical interface which allows to set many of the available functions and enjoy the experimentation.

In the document, some notes for the assembly of a prototype in the two currently available HW versions are presented, as well as the description of the SW image loading and its configuration.

At the end of the document, there is also a HW/SW "debugging" chapter, very useful if the user wants to join the development group.

4 Compatibility with other existing LoRa APRS Standard

The Software support both AX.25 encapsulation and the Austrian LoRa APRS implementations, which are a world-wide standard. We will call this mode “OE_Style” from now on; the SW can always decode APRS traffic in both AX.25 or OE_Style encapsulation; the outgoing LoRa traffic can be set by GUI in one way or another.

This feature allows you to use in the same LoRa APRS network both SARIMESH and OE_Style devices.

5 Features introduced starting from Versions 4.1.x

With the SW 4.x version, new interesting features have been introduced:

1) Use standard APRS compression for the coordinates of transmitted packets, which can be enabled or not from the GUI. During receiving, the SW can always decode both compressed and uncompressed location.
2) "Agile Beaconing”: it’s an algorythm for the optimization of beacon transmitting rate considering the trajectory of the moving tracker.

3) “Black Zones”: they allow to exclude a set of geographical "zones" from sending beacons, mainly for privacy purposes or to mitigate any radio congestion problems in particular areas such as eg. a town centre.
4) Support of the “syslog” protocol towards a remote server for the purpose of collecting, mainly in the testing and diagnostic phase, a series of information useful for tracing the behavior of a device; this function obviously requires that the device under test is able to reach the internet directly, e.g. via a local wifi network (e.g. via a mobile phone operating in hotspot mode).
5) Support of MQTT protocol: It sends data to a remote server (broker) again via internet connectivity. This function can be used in two ways:
    a. in a first way, the system allows you to perform remote operation on a LoRa Sarimesh device connected on the internet. e.g. it is possible to access some functions remotely or even perform operations on the device (e.g. restart the device or trigger other ad hoc functions). It’s worth to be noted that this mode does not need any intervention or interaction in the local internet access device (router).
    b. A second way of using the MQTT protocol is the classic one used in IoT applications, which allows local data to be sent to remote servers such as e.g. measurement of temperature, humidity or other types of parameters.
6) Use of front button for quick functions: keeping the key pressed, different options are presented on the display in sequence at intervals of few seconds. Releasing the key when, the function shown on the display will be selected. A trivial example is the manual beacon transmission: by pressing the key, the first option is exactly “Send Beacon”. By releasing the key the beacon will be sent. The functions are:
    a. Send Beacon
    b. Turn On/Off the Wi-Fi upstream connection
    c. Restart device
7) Use of front button for loading of presets during boot phase. When the device is switched on, after few seconds the user can see the permanent switching ON of the green and red LEDs (in case of SARIMESH HW), during which a pressure of the key on the device is interpreted as a request to reset to the factory defaults. In "factory default", the device will operate in tracker mode with standard parameters which have to be changed immediately for concrete use, but they immediately allow you to have a device capable of functioning giving signs of life and enabling the user to set his own personal parameters directly from the GUI interface accessible for example via a mobile phone or a PC.
   If the SW in question is installed on HW devices other than the Sarimesh one, obviously the presence or absence of a small key and some LEDs must be taken into account.
   In the case of TTGO type cards it is easy as there is a small key and a blue led which is mapped to operate as both the two red and green leds of the HW Sarimesh.
   In the case of HELTEC boards that do not have a key functionally intended for a use other than a simple reset, unfortunately it will be necessary to restore the factory defaults by reloading a complete SW image from scratch.

6 SARIMESH HW assembly notes

Currently there are two HW variants of the LoRa_Beacon PCBs: the two variants differ only in the type of physical/functional plug-in modules supported and in the physical dimensions, but the basic SW functions are mantained in both cases.

6.1 LoRa_Beacon_2020_vr4_1 version (alias mini-Tracker)

The following figures represent the wiring diagram and the Tracker PCB, which is characterized by being **very small in size and intended for mobile use** .

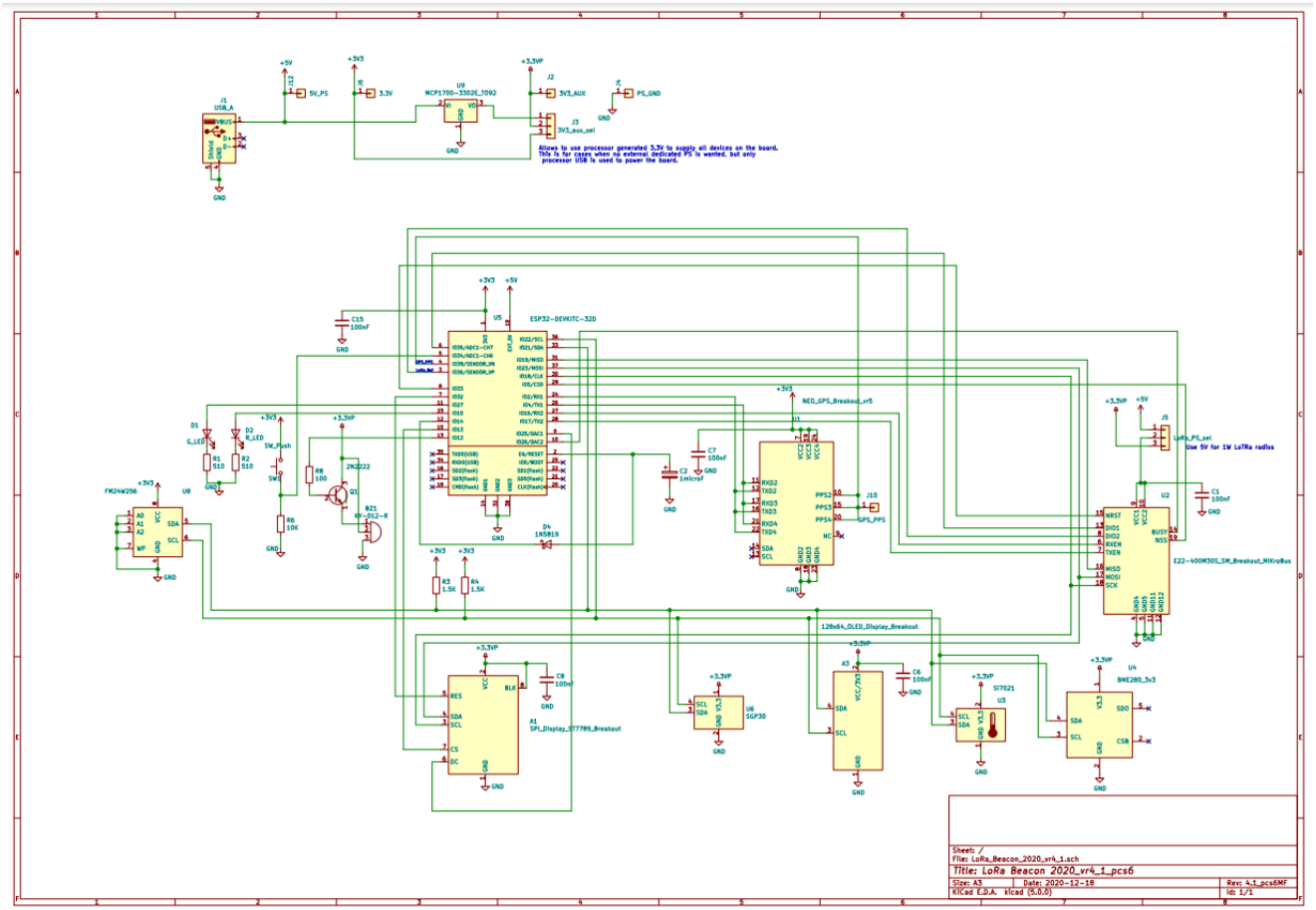

Figure 6-1 : LoRa_Beacon_2020_vr4.1_pcs6 version schematic diagram

This PCB is designed to be powered via a USB Type “A” cable (therefore at 5V) using common sources such as a mobile phone power supply, a car USB socket or any power bank designed to power a classic mobile phone.

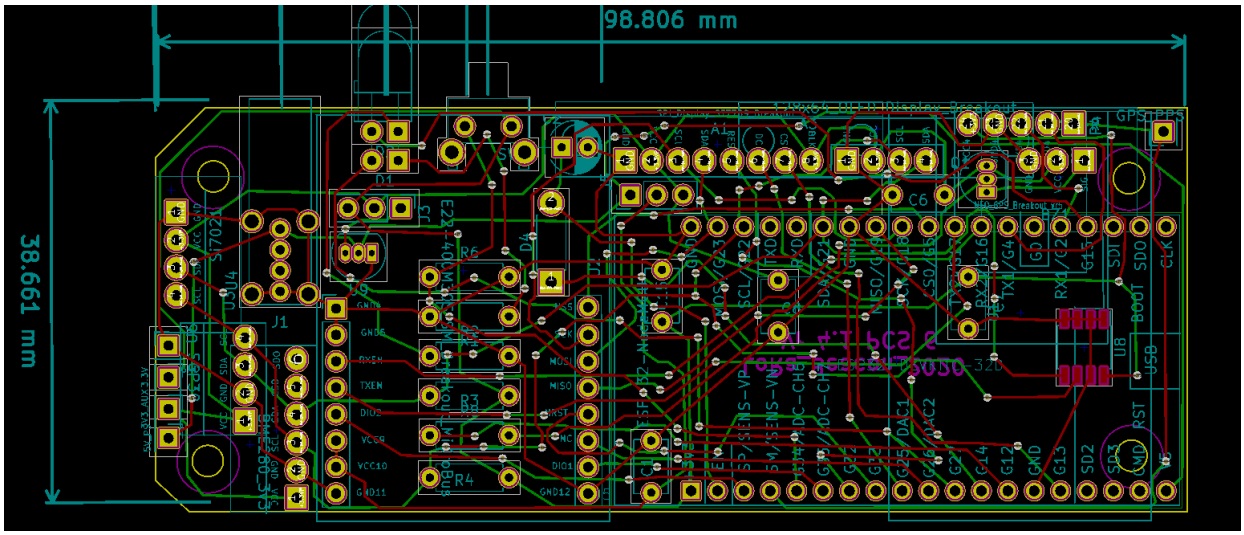

Figure 6-2 : LoRa_beacon_2020_Vr_4.1_pcs6 layout

The part list is reported here below:

```
 1      A1 - SPI_Display_ST7789_Breakout : APRS_Mini_Tracker:SPI_Display_ST7789_breakout
 2      A3 - 128x64_OLED_Display_Breakout : APRS_Mini_Tracker:128x64_OLED_Vert_Display_breakout
 3     BZ1 -         KY-012-R : APRS_Mini_Tracker:KY-012-R_breakout
 4      C1 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 5      C2 -          1microF : Capacitors_THT:CP_Radial_D5.0mm_P2.50mm
 6      C6 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 7      C7 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 8      C8 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 9     C15 -            100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
10      D1 -            G_LED : APRS_Mini_Tracker:LED_D5.0mm_Horizontal_O3.81mm_Z5.0mm
11      D2 -            R_LED : APRS_Mini_Tracker:LED_D5.0mm_Horizontal_O3.81mm_Z5.0mm
12      D4 -           1N5819 : Diodes_THT:D_DO-41_SOD81_P7.62mm_Horizontal
13      J1 -            USB_A : USB_A:USB_A_Vertical
14      J2 -          3V3_AUX : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
15      J3 -      3V3_aux_sel : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
16      J4 -           PS_GND : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
17      J5 -      LoRa_PS_sel : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
18      J6 -             3.3V : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
19     J10 -          GPS_PPS : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
20     J12 -            5V_PS : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
21      Q1 -           2N2222 : TO_SOT_Packages_THT:TO-92_Inline_Narrow_Oval
22      R1 -              510 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
23      R2 -              510 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
24      R3 -             1.5K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
25      R4 -             1.5K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
26      R6 -              10K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
27      R8 -              100 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
28     SW1 -          SW_Push : APRS_Mini_Tracker:Button_Angled
29      U1 - NEO_GPS_Breakout_vr5 : APRS_Mini_Tracker:NEO_GPS_Breakout_vr7
30      U2 - E22-400M30S_SM_Breakout_MiKroBus : APRS_Mini_Tracker:E22-M40030S_SM3_MiKroBus_Breakout_STACKED
31      U3 -           Si7021 : APRS_Mini_Tracker:SI-7021_breakout
32      U4 -       BME280_3v3 : APRS_Mini_Tracker:BME-280_breakout
33      U5 - ESP32-DEVKITC-32D : APRS_Mini_Tracker:MODULE_ESP32-DEVKITC-32D_STACKED
34      U6 -            SGP30 : APRS_Mini_Tracker:SGP30_breakout
35      U8 -         FM24W256 : Housings_SOIC:SOIC-8_3.9x4.9mm_Pitch1.27mm
36      U9 - MCP1700-3302E_TO92 : TO_SOT_Packages_THT:TO-92_Inline_Narrow_Oval
```

Figure 6-3 : LoRa_Beacon_2020_vr_4.1_pcs6 part list

The figure below provides a more readable view of the placement of the various components on the PCB.

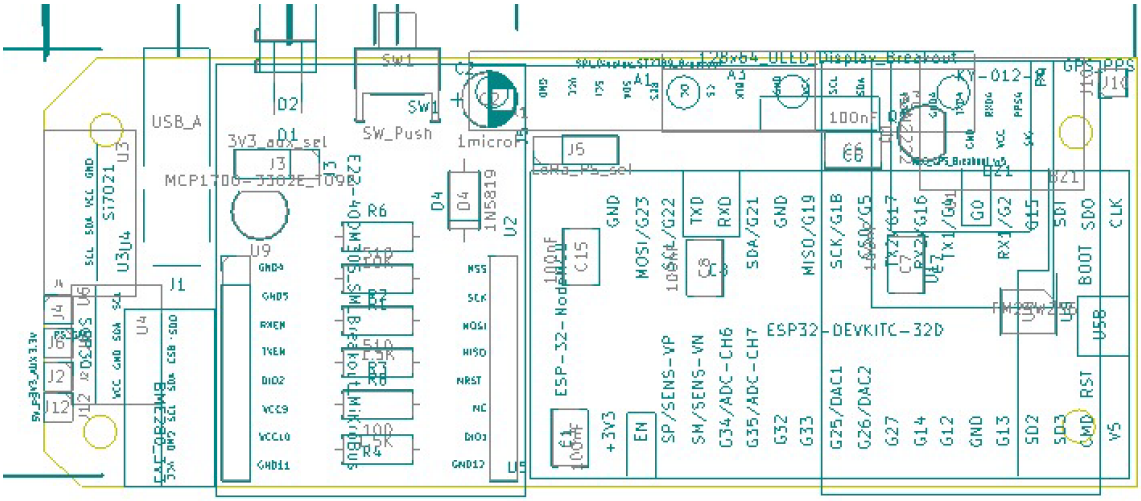

Figure 6-4 : LoRa_Beacon Vr. 4.1-pcs6 : main components positioning

The following figure provides a detail of the assembly of the single SMD chip present on the circuit; pin 1 corresponds to a small hole on the plastic body of the U8 device.

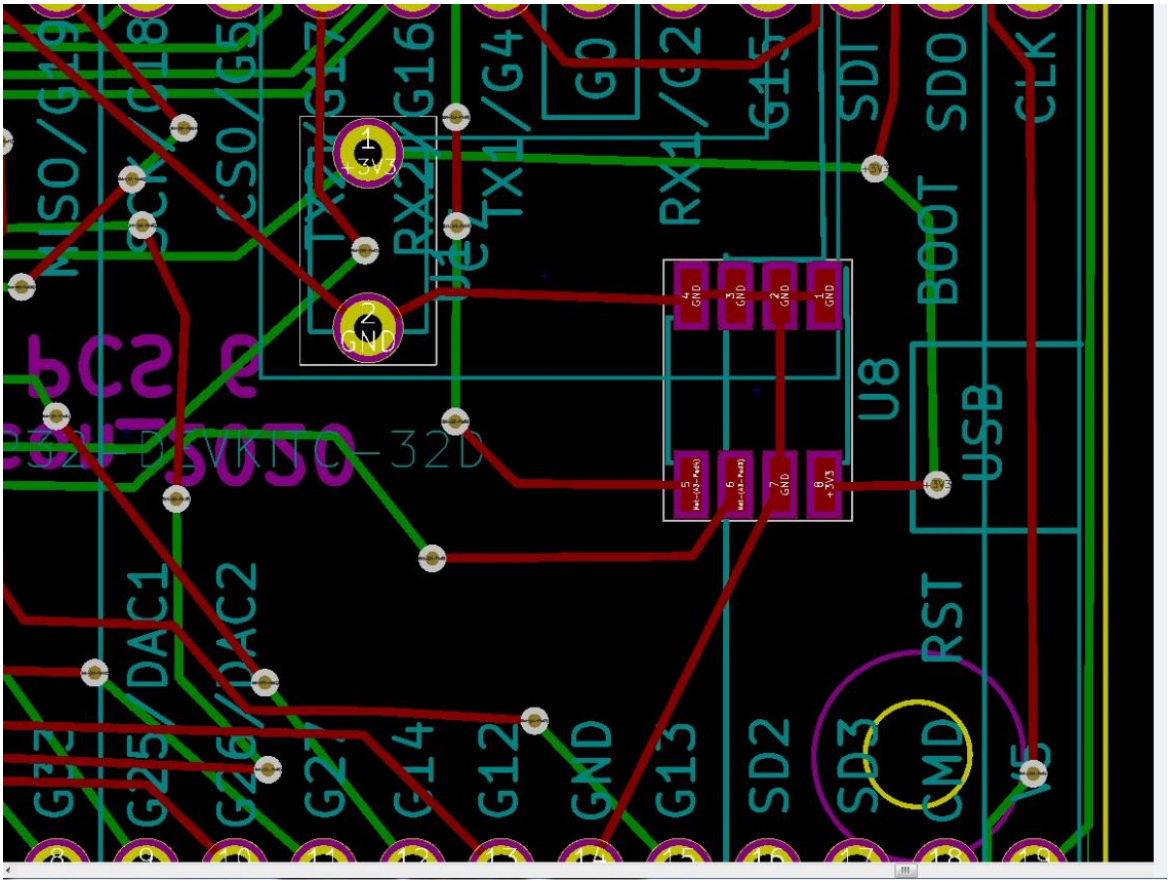

Figure 6-5 : LoRa_Beacon Vr. 4.1_pcs6 : U8 SMD FRAM assembly detail

The following photo shows a view of the component side of the Tracker PCB.

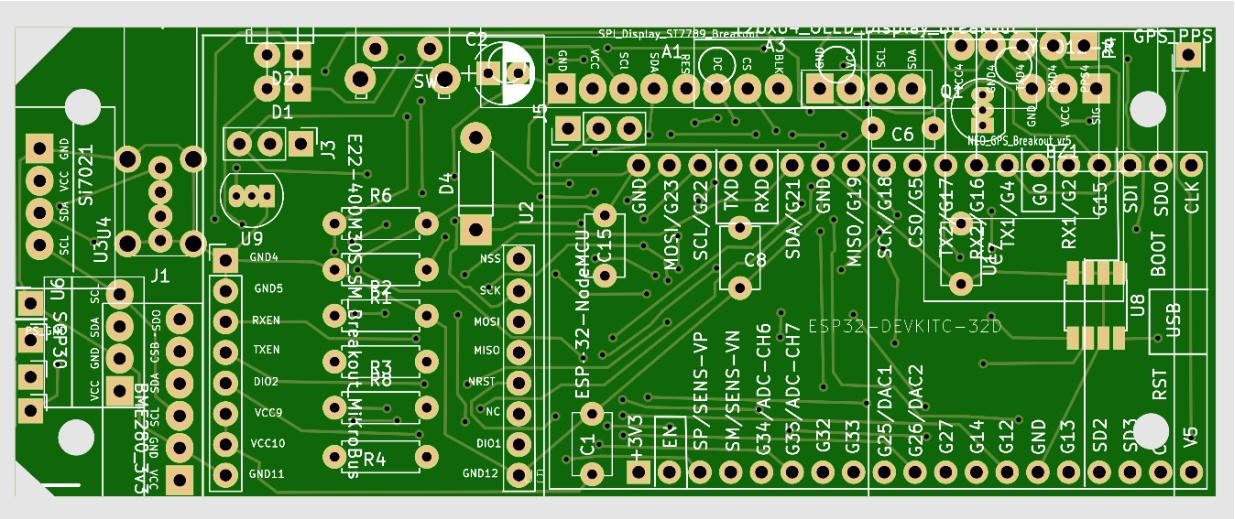

Figure 6-6 : LoRa_Beacon Vr 4.1_pcs6 top side photo of the printed circuit board

Figure 6-7 shows a Tracker completely assembled in one of its configuration.

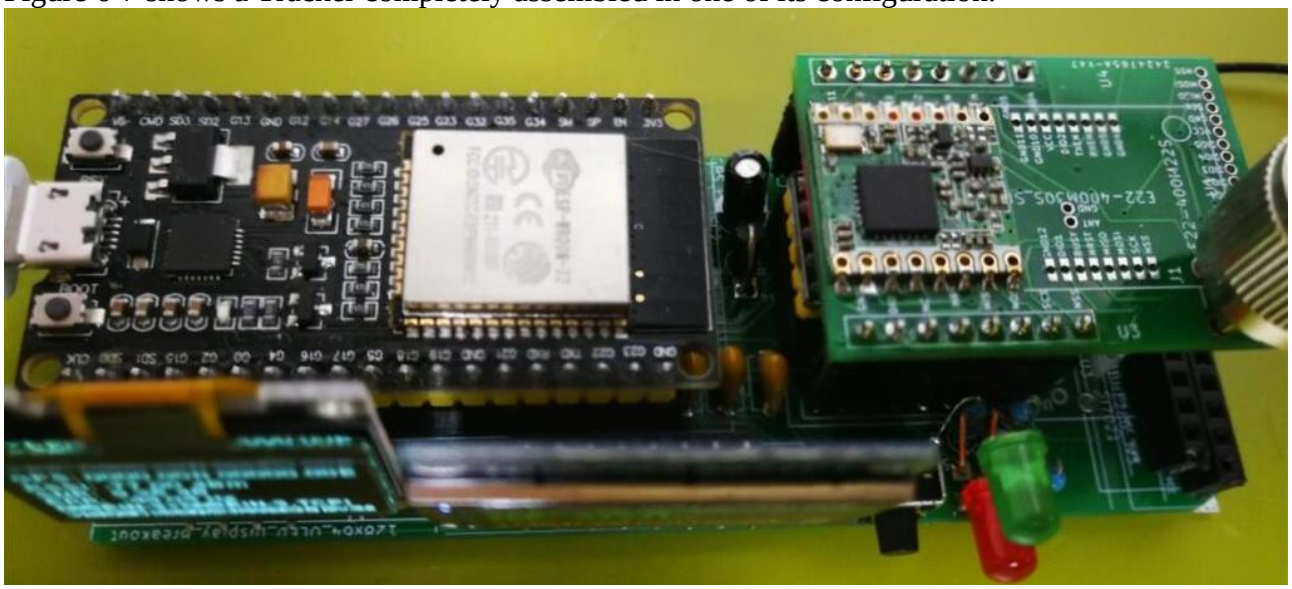

Figure 6-7 : tracker completely assembled

Figure 6-8 shows a comparison between the Sarimesh tracker and T-Beam board.

Figure 6-8 : Sarimesh HW compared to T-Beam

6.2 LoRa_Beacon_2020_vr3_pcs4 version (aka iGate)

The following figures represent the wiring diagram and the PCB of Igate, which is characterized by being larger than the Tracker version and by having the provision for additional plug-in modules which allow more advanced experimentation compared to the Tracker.

Being this version designd to be used in a fixed, a 12V power supply is foreseen.

Unlike the tracker, the iGate has the possibility of connecting to the internet both in WiFi mode and via a LAN interface, if a dedicated plug-in module is installed. Moreover, it can hold an RTC (Real Time Clock) module for keeping local time even in the absence of a connection to a GPS device or the presence of internet connectivity.

The PCBs keep the complete interchangeability of the common modules.

In the electrical diagram, the similarities are obviously very high even if the numbering and naming of the various components are different.

Details of this version follow.

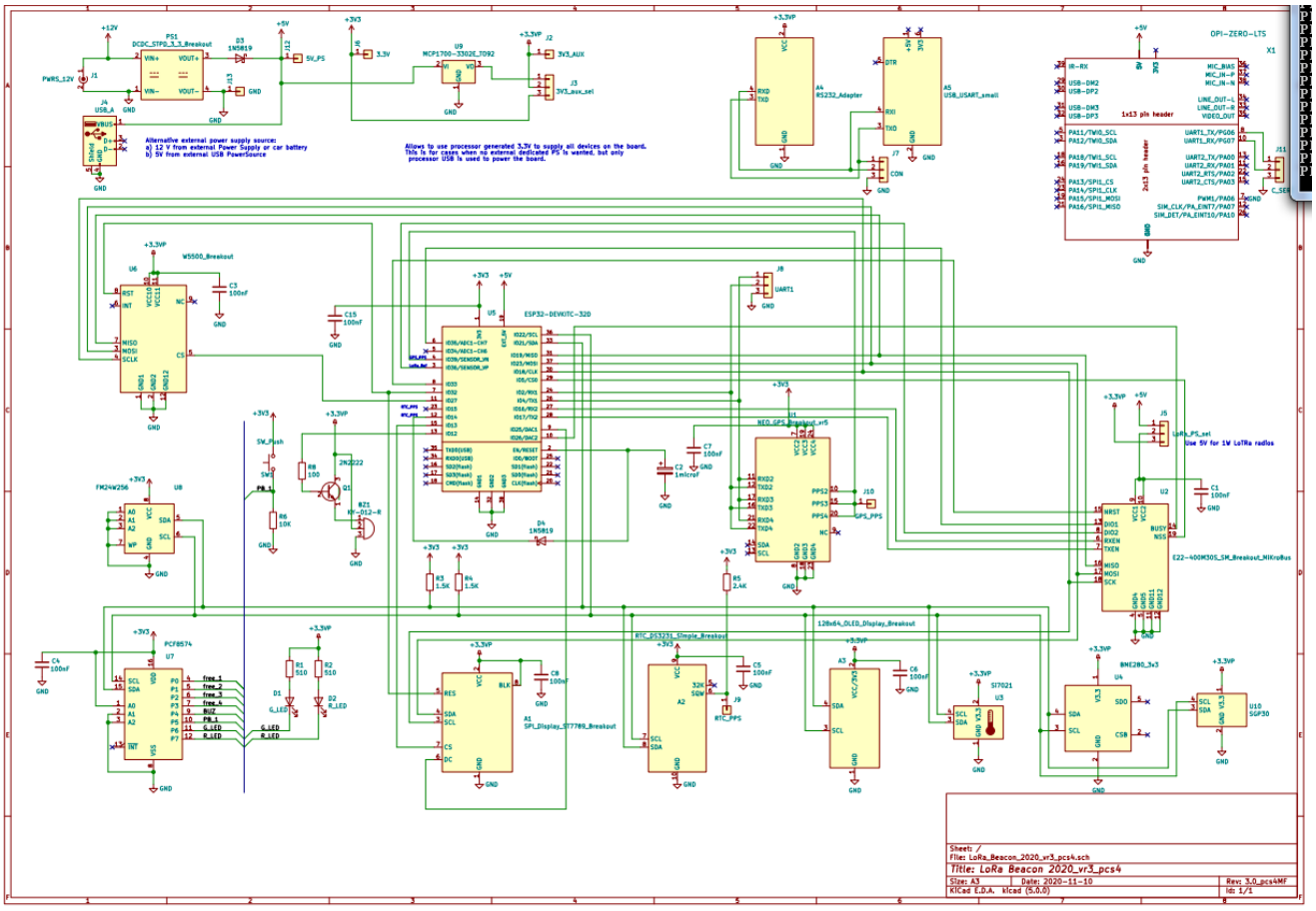

Figure 6-9 : LoRa_Beacon_2020_vr3_pcs4 : wiring diagram

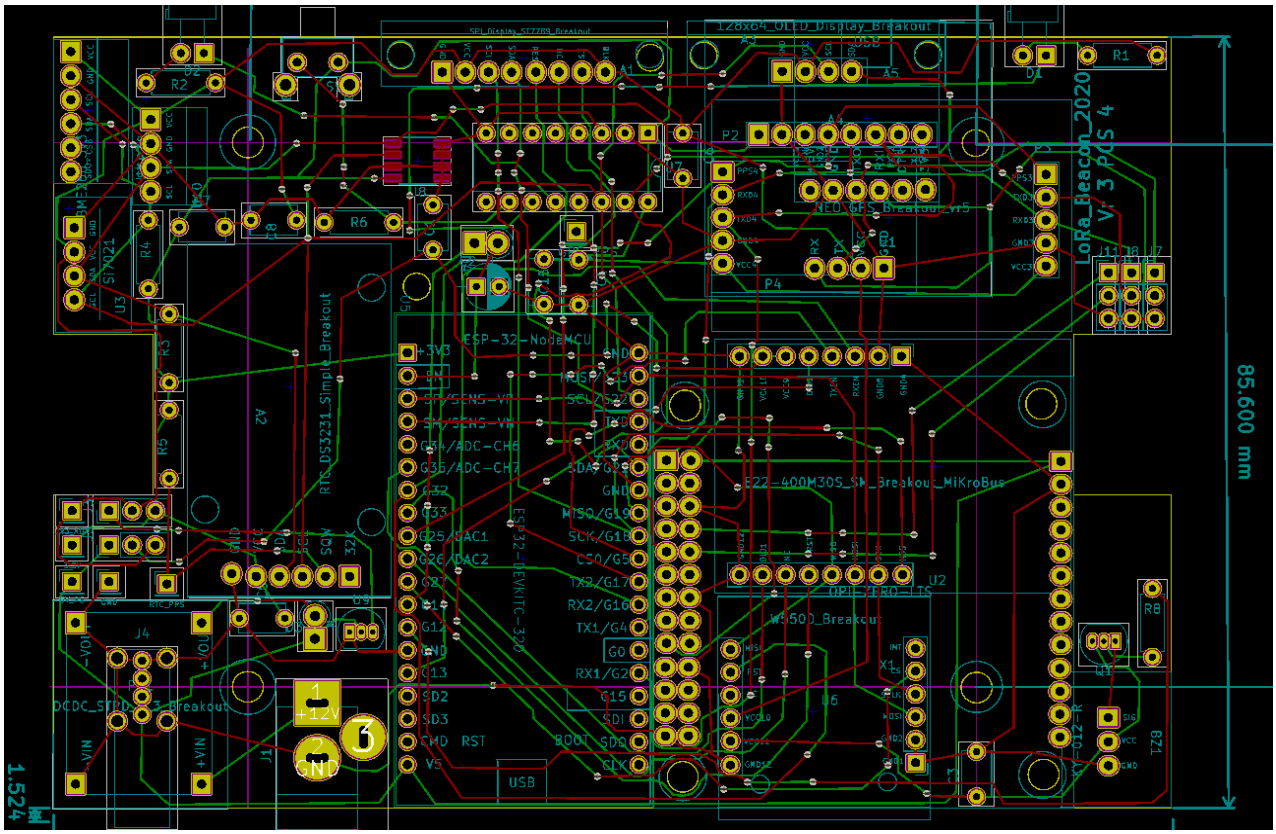

Figure 6-10 : LoRa_Beacon_2020_vr3_pcs4 : Carrier PCB layout

Here below, the part list of the iGate version is reported below:

```
 1      A1 - SPI_Display_ST7789_Breakout : APRS_Mini_Tracker:SPI_Display_ST7789_breakout
 2      A2 - RTC_DS3231_Simple_Breakout : APRS_Mini_Tracker:RTC_DS3231_breakout_simpl
 3      A3 - 128x64_OLED_Display_Breakout : APRS_Mini_Tracker:128x64_OLED_Vert_Display_breakout
 4      A4 -    RS232_Adapter : APRS_Mini_Tracker:RS232_adapter
 5      A5 -  USB_USART_small : APRS_Mini_Tracker:USB_USART_small
 6     BZ1 -         KY-012-R : APRS_Mini_Tracker:KY-012-R_breakout
 7      C1 -           100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
 8      C2 -          1microF : Capacitors_THT:CP_Radial_D5.0mm_P2.50mm
 9      C3 -           100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
10      C4 -           100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
11      C5 -           100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
12      C6 -           100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
13      C7 -           100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
14      C8 -           100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
15     C15 -           100nF : Capacitors_THT:C_Disc_D5.1mm_W3.2mm_P5.00mm
16      D1 -           G_LED : APRS_Mini_Tracker:LED_D5.0mm_Horizontal_O3.81mm_Z5.0mm
17      D2 -           R_LED : APRS_Mini_Tracker:LED_D5.0mm_Horizontal_O3.81mm_Z5.0mm
18      D3 -          1N5819 : Diodes_THT:D_DO-41_SOD81_P2.54mm_Vertical_KathodeUp
19      D4 -          1N5819 : Diodes_THT:D_DO-41_SOD81_P2.54mm_Vertical_KathodeUp
20      J1 -         PWRS_12V : Connectors:JACK_ALIM
21      J2 -          3V3_AUX : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
22      J3 -      3V3_aux_sel : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
23      J4 -            USB_A : USB_A:USB_A_Vertical
24      J5 -      LoRa_PS_sel : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
25      J6 -             3.3V : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
26      J7 -              CON : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
27      J8 -            UART1 : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
28      J9 -          RTC_PPS : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
29     J10 -          GPS_PPS : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
30     J11 -            C_SER : Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
31     J12 -            5V_PS : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
32     J13 -              GND : APRS_Mini_Tracker:Pin_Header_Straight_1x01_Pitch2.54mm_local
33     PS1 - DCDC_STPD_3_3_Breakout : APRS_Mini_Tracker:DCDC_STPD_3_3_TOP_Breakout
34      Q1 -           2N2222 : TO_SOT_Packages_THT:TO-92_Inline_Narrow_Oval
35      R1 -              510 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
36      R2 -              510 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
37      R3 -             1.5K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
38      R4 -             1.5K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
39      R5 -             2.4K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
40      R6 -              10K : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
41      R8 -              100 : Resistors_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
42     SW1 -          SW_Push : APRS_Mini_Tracker:Button_Angled
43      U1 - NEO_GPS_Breakout_vr5 : APRS_Mini_Tracker:NEO_GPS_Breakout_vr5
44      U2 - E22-400M30S_SM_Breakout_MiKroBus : APRS_Mini_Tracker:E22-M40030S_SM3_MiKroBus_Breakout
45      U3 -           Si7021 : APRS_Mini_Tracker:SI-7021_breakout
46      U4 -       BME280_3v3 : APRS_Mini_Tracker:BME-280_breakout
47      U5 - ESP32-DEVKITC-32D : APRS_Mini_Tracker:MODULE_ESP32-DEVKITC-32D
48      U6 -   W5500_Breakout : APRS_Mini_Tracker:W5500_Breakout
49      U7 -          PCF8574 : APRS_Mini_Tracker:PCF8574_DIP
50      U8 -         FM24W256 : Housings_SOIC:SOIC-8_3.9x4.9mm_Pitch1.27mm
51      U9 - MCP1700-3302E_TO92 : TO_SOT_Packages_THT:TO-92_Inline_Narrow_Oval
52     U10 -            SGP30 : APRS_Mini_Tracker:SGP30_breakout
53      X1 -     OPI-ZERO-LTS : APRS_Mini_Tracker:Controller_Vr1
```

Figure 6-11 : LoRa_Beacon_2020_vr3_pcs4 : part list

Here below, there is a view of the placement of the components:

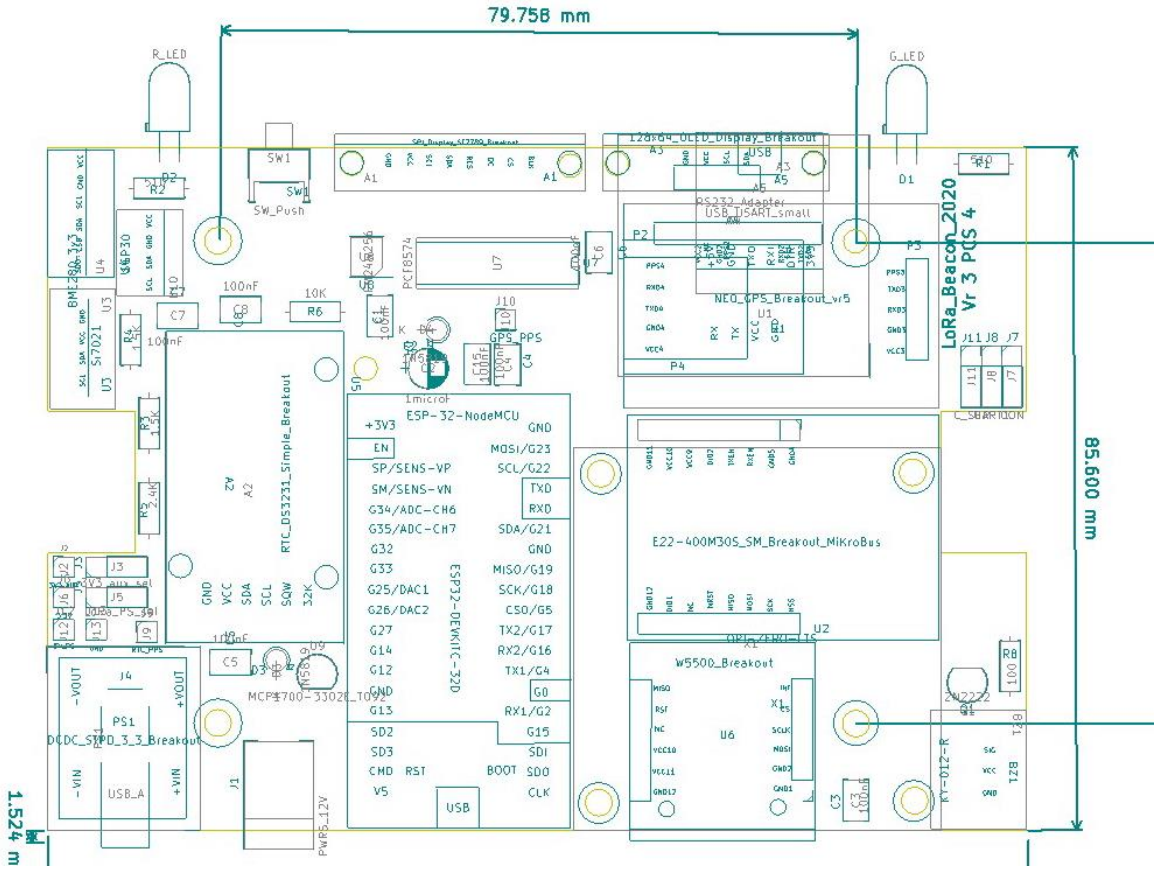

Figure 6-12 : LoRa_Beacon_2020_vr3_pcs4 : main components placement

Here below, the assembly details of the single SMD U8 component are reported.

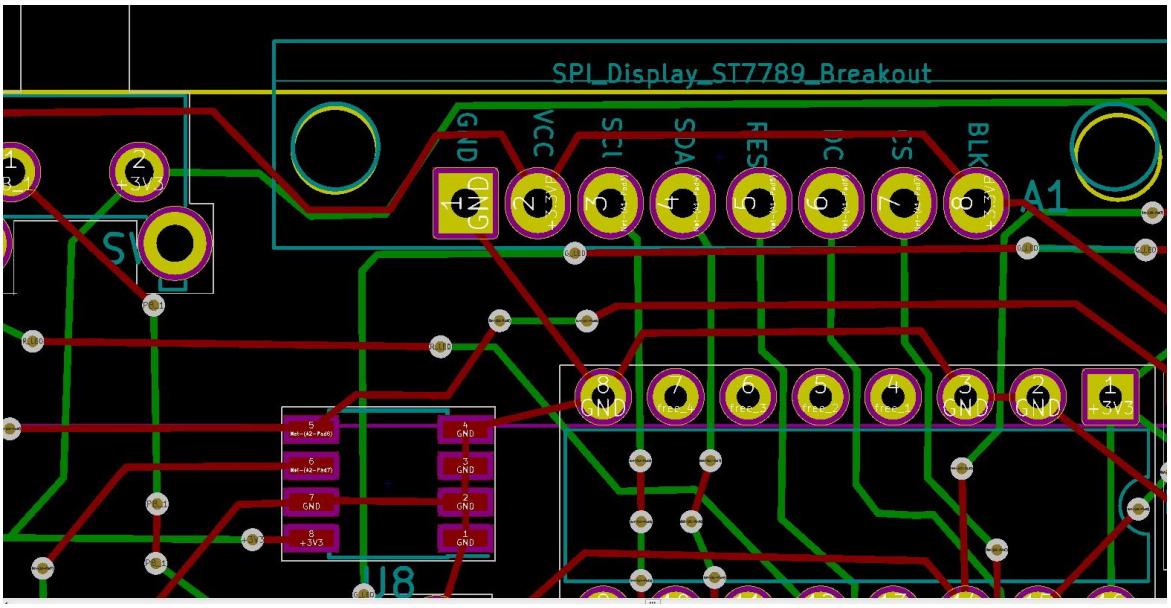

Figure 6-13 : LoRa_Beacon_2020_vr3_pcs4 : U8 SMD FRAM positioning detail

The following figure is a photo of the component side of the PCB of this version

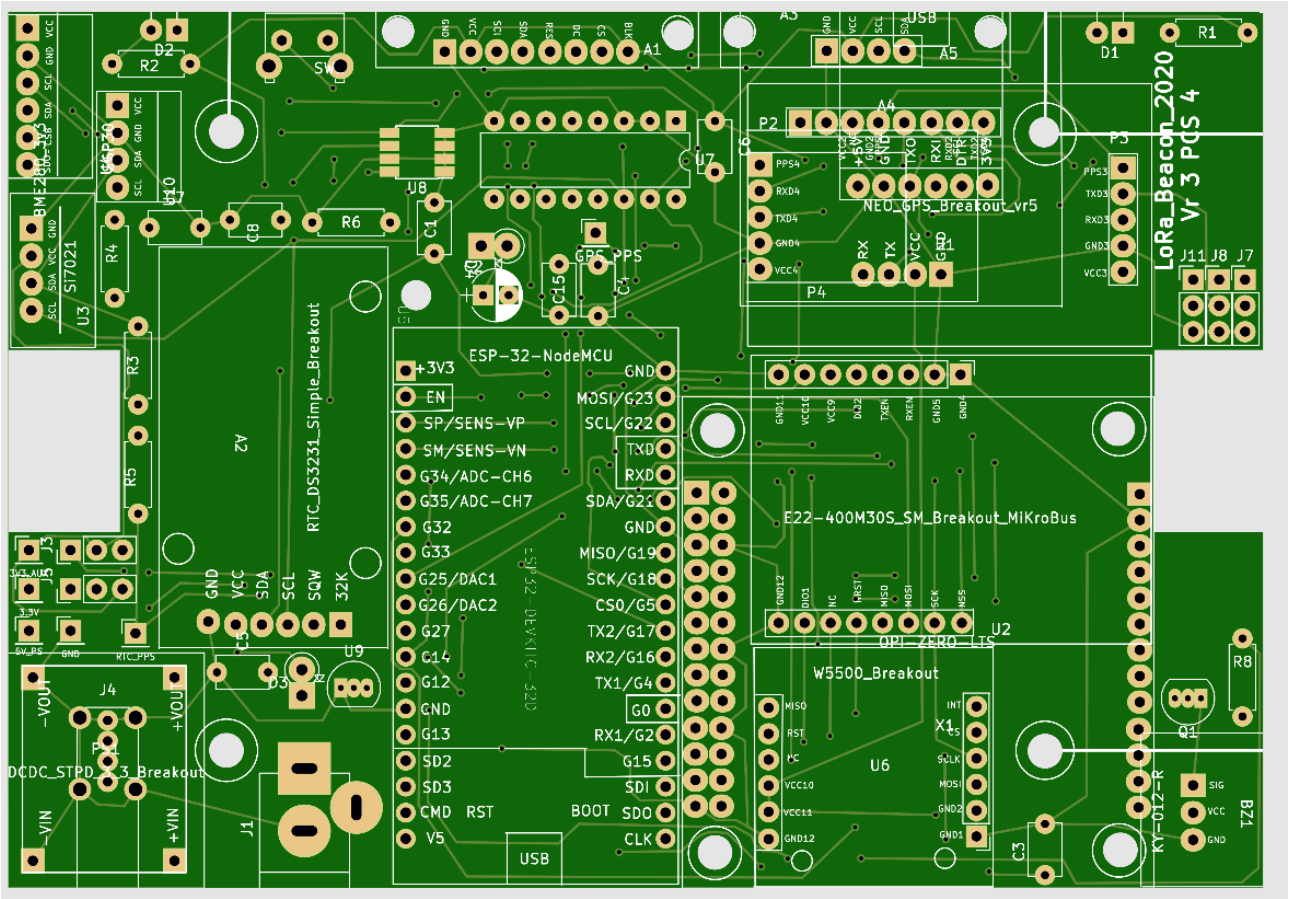

Figure 6-14 : LoRa_Beacon_Vr 3 pcs 4 : PCB top side view

Figure 6-15 shows an iGate completely assembled in one of the possible configuration.

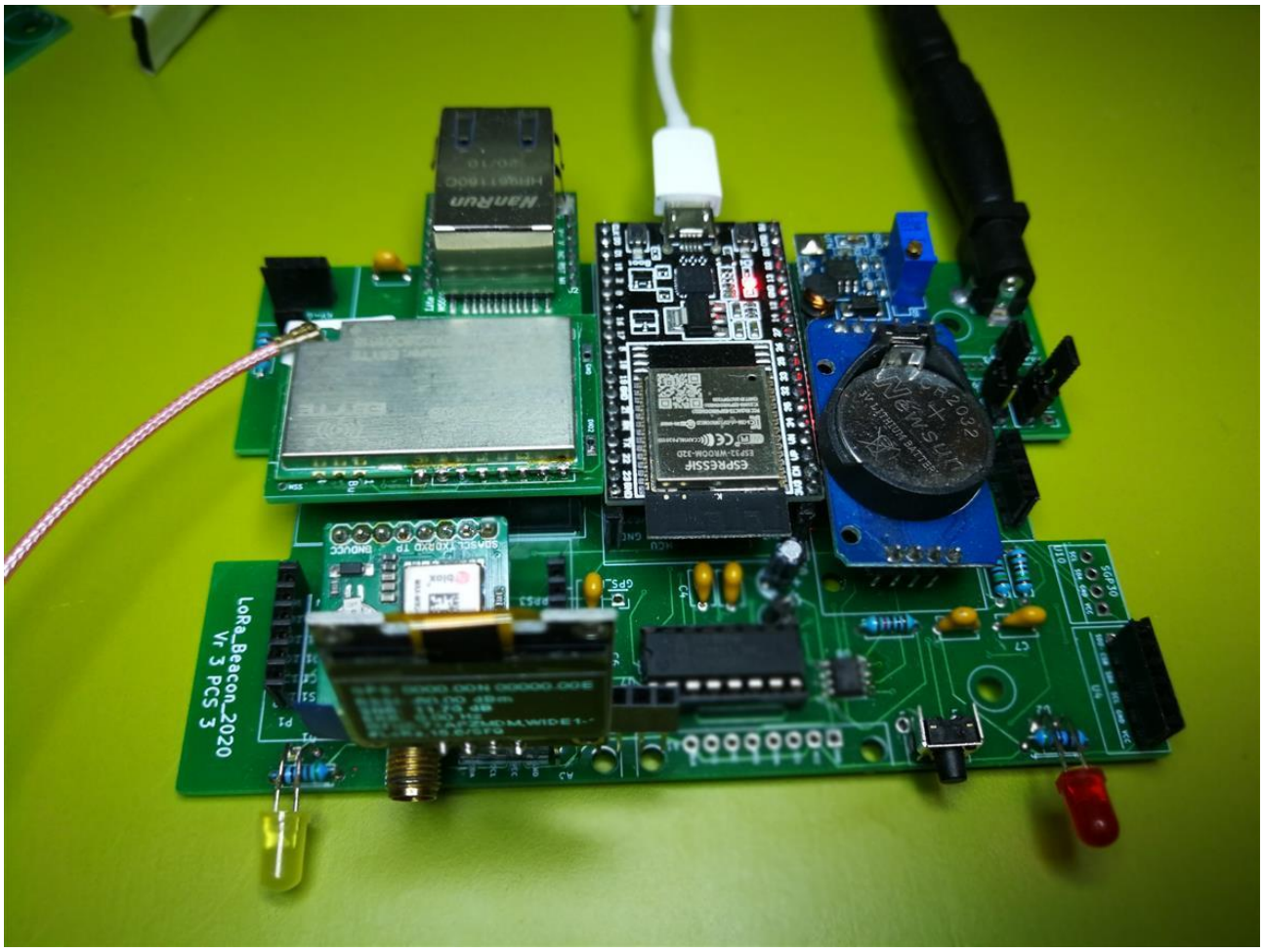

Figure 6-15 : iGate completely assembled

6.3 Carrier for LoRa radio modules (LoRa carrier)

Unfortunately, the LoRa radio modules available on the market have a considerable diversity of pin layout and physical format, not always compatible with a 2.54mm pin pitch that characterizes the majority of the other modules required for our implementation.

For these reasons, we designed a PCB with the aim to adapt the pinout of the various LoRa modules on the market to the standard 2.54mm receptacles present on the main carrier.

We have developed different carriers, but in this paragraph is reported the more useful, at least initially.

The following figures represent the wiring diagram and the PCB of this LoRa carrier version:

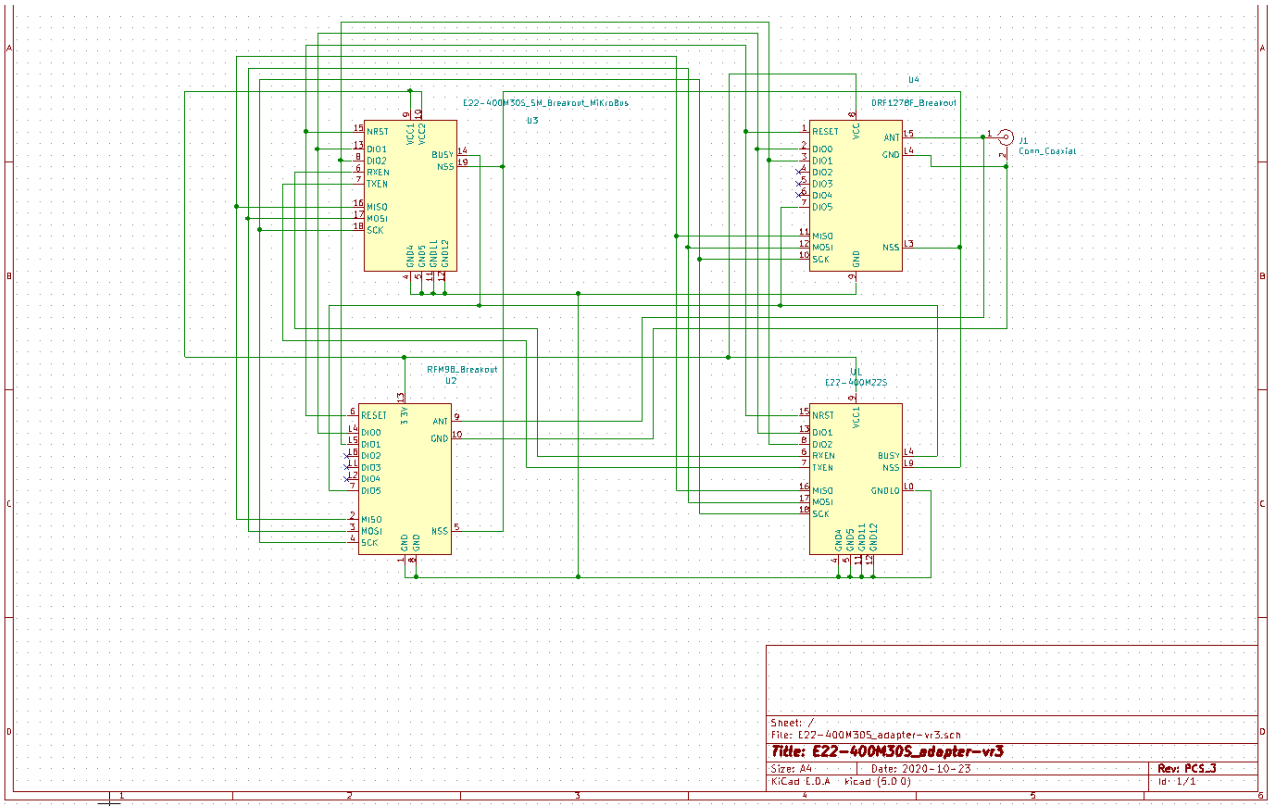

Figure 6-16 : LoRa carrier wiring diagram

As can be seen, the PCB has many different fingerprints:

- Ebyte E22-400M30S and similar
- Ebyte E22-400M22S and similar
- RFM98 and similar
- DRF1278F and similar

Obviously, only one of the fingerprints will be used at a time; to mount different LoRa chips, different examples of carrier + LoRa chips will therefore be built.

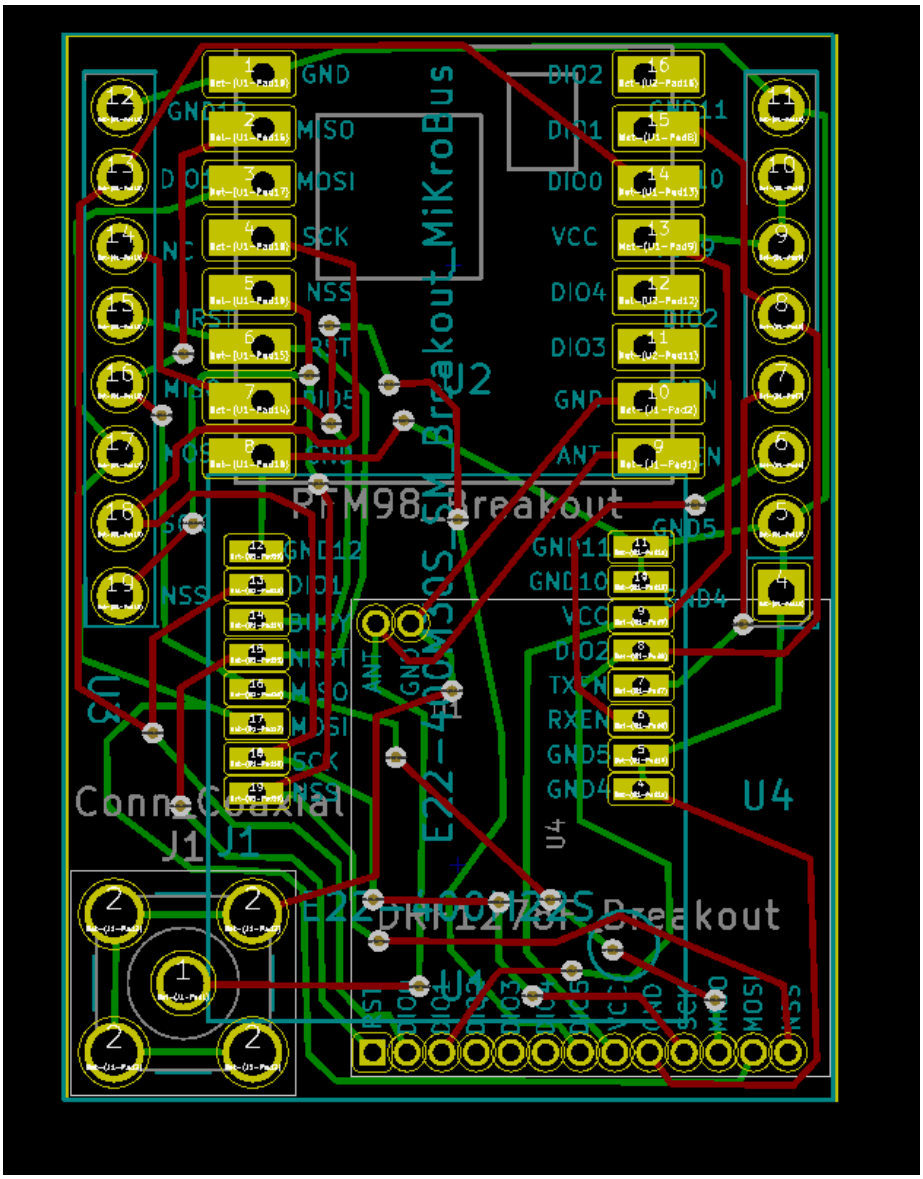

Figure 6-17 : LoRa carrier PCB layout

The figure below, a photo of the top side of the PCB is reported.

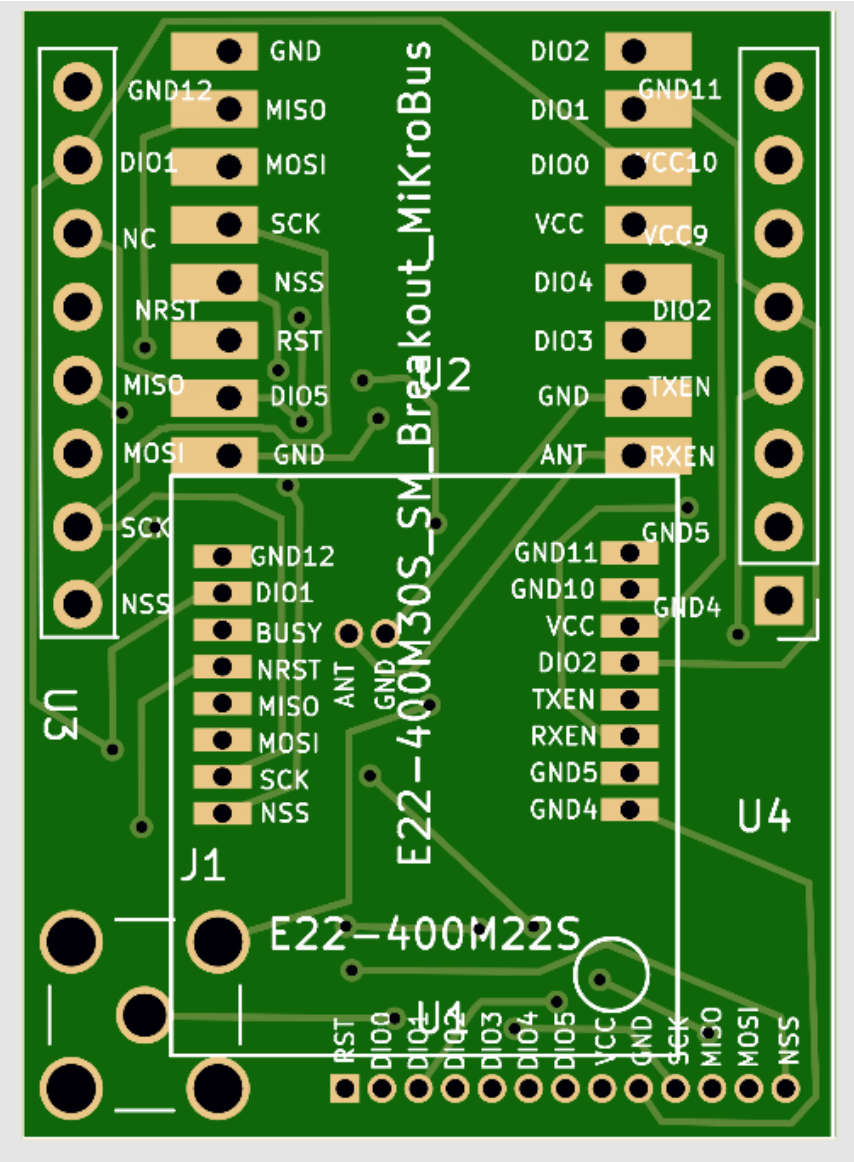

Figure 6-18 : LoRa_Carrier PCB top view

One of the most popular LoRa module is the E22-400M30S, which already has a pin spacing of 2.54 mm even if with an SMD mounting layout.

Also in this case, the carrier will allow the radio module to be mounted in an optimal way. The following figures illustrate how this module is mounted.

Figure 6-19 : LoRa module mounted on LoRa carrier PCB – view1

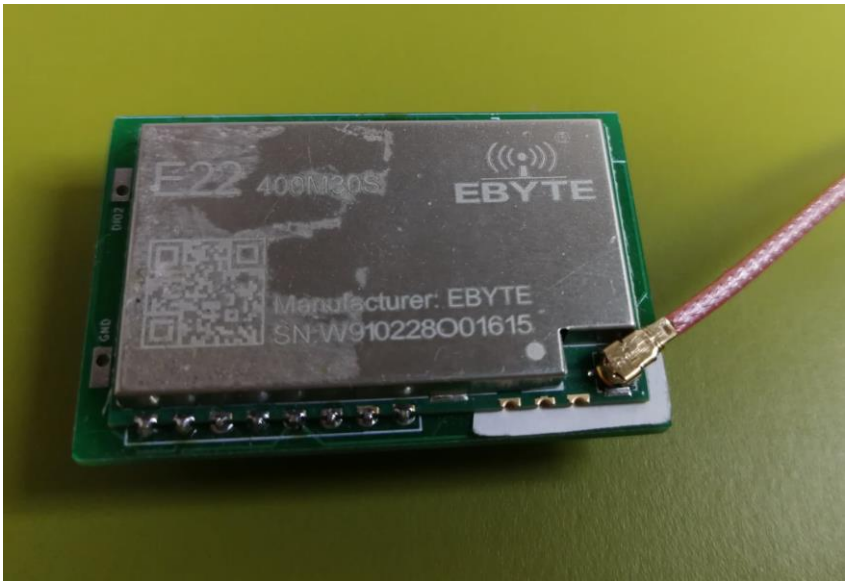

Figure 6-20 : LoRa module mounted on LoRa carrier PCB – view2

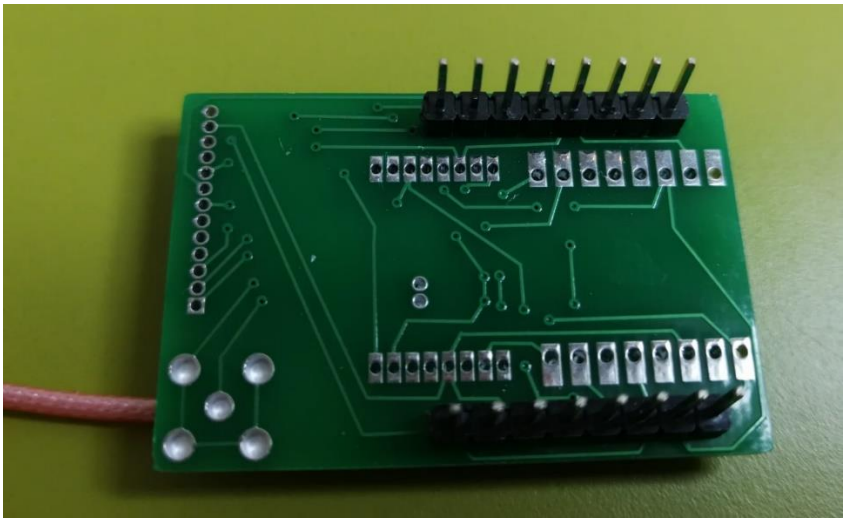

Figure 6-21 : LoRa module mounted on LoRa carrier PCB - bottom view

6.4 Assembly notes related to both PCB carrier versions.

On both main carrier PCB versions there are some provisions for optional plug-in modules.

The first is for the display modules: both versions have the footprints and connections compatible for both color and monochrome display, which can be used alternatively or even simultaneously through an appropriate customization of the SW; in the standard version of the SW, a color TFT module with SPI interface and an OLED module with I2C interface are simultaneously supported, however showing the same type of content.

A further note always concerns the color display module which, depending on the version purchased, may or may not have the pins already soldered on the relative PCB: for an optimal assembly it is necessary to use pins with a 90° bend; if straight pins are already soldered on the module, it is necessary to remove them and replace them with pins bent at 90°; to simplify the work of desoldering the original pins, it is better to cut the plastic support that binds the original pins with a pair of nippers before starting the desoldering process.

All the modules can be plugged on the Carrier Board using a row of standard 2.54mm pitch connectors.

This allows in the future to be able to easily replace or interchange the modules without complicated desoldering operations which would inevitably ruin the circuits.

For both PCB versions the sensor modules are optional and currently not supported in the base SW.

For the iGate version, the LAN module is required only if you want to use a cable LAN connection instead of wifi as internet connection method; by default this mode is not enabled.

With the 4.x version of the SW, the support of the LAN mode is removed for the simple reason that no need for this feature has been seen in the experiments carried out so far.

The GPS module is mandatory for the tracker version while it is optional for the iGate version.

The RTC module for the iGate version is supported by SW but is still optional.

The GPS module has been provided on both carrier PCBs with a footprint that can be adapted to different types of modules available on the usual internet purchase portals; for further details on compatible modules, contact the writer by e-mail

Note: the GPS module must have PPS OUT pin.

The iGate PCB presents other customization possibilities in terms of modules that can be equipped which at this stage are not yet documented as they have not yet been sufficiently tested in terms of SW support.

The iGate version has two power options: 5V connection through a USB cable or via a 12V connection received from a jack connector; the default version uses the 12V power supply and therefore requires the PS1 DC/DC module and its power connector.

As a general note, it is recommended to test the circuit by gradually connecting the various modules, starting with the ESP32 processor and continuing with the display, the GPS and finally the LoRa module.

In order to possibly allow trobleshouting actions, simple "test segments" will be soon uploaded on github, i.e. small programs which, when suitably loaded, via the Arduino SW development environment or, in version 4.x, PlatformIO on the ESP32 processor can allow you to test the various functional blocks individually or in small groups.

A Troubleshooting guide is also available as appendix of this document.

Finally, there are some configuration that can be made using appropriate jumpers with 2.54mm format which are documented below:

6.4.1 Mini-tracker PCB version: jumpers and their meaning

- **J3**: 3V3_aux_sel default connect pins 1-2
- **J5**: LoRa_PS_sel use position 1-2 for LoRa module E22-400M30S operating at 5V; position 2-3 for LoRa modules working at 3.3V

6.4.2 iGate PCB version: jumpers and their meaning

- **J3**: 3V3_aux_sel default connect pins 1-2
- **J5**: LoRa_PS_sel use position 1-2 for LoRa module E22-400M30S operating at 5V; position 2-3 for LoRa modules working at 3.3V
- **J7**: for future expansions, no jumpers to connect
- **J8**: for future expansions, no jumpers to connect
- **J11**: for future expansions, no jumper to connect

6.5 Plastic Enclosures

3D case projects are available on Github repository for both Tracker and Igate.
They are a “baseline” and the user can easily customize the projects following his needs, or can adapt commercial boxes.
The source files can be opened by OpenScad, a freeware software freely downloadable from the internet: https://openscad.org/
Here below, a pictureof tracker case viewed in OpenScad environment, only for example.

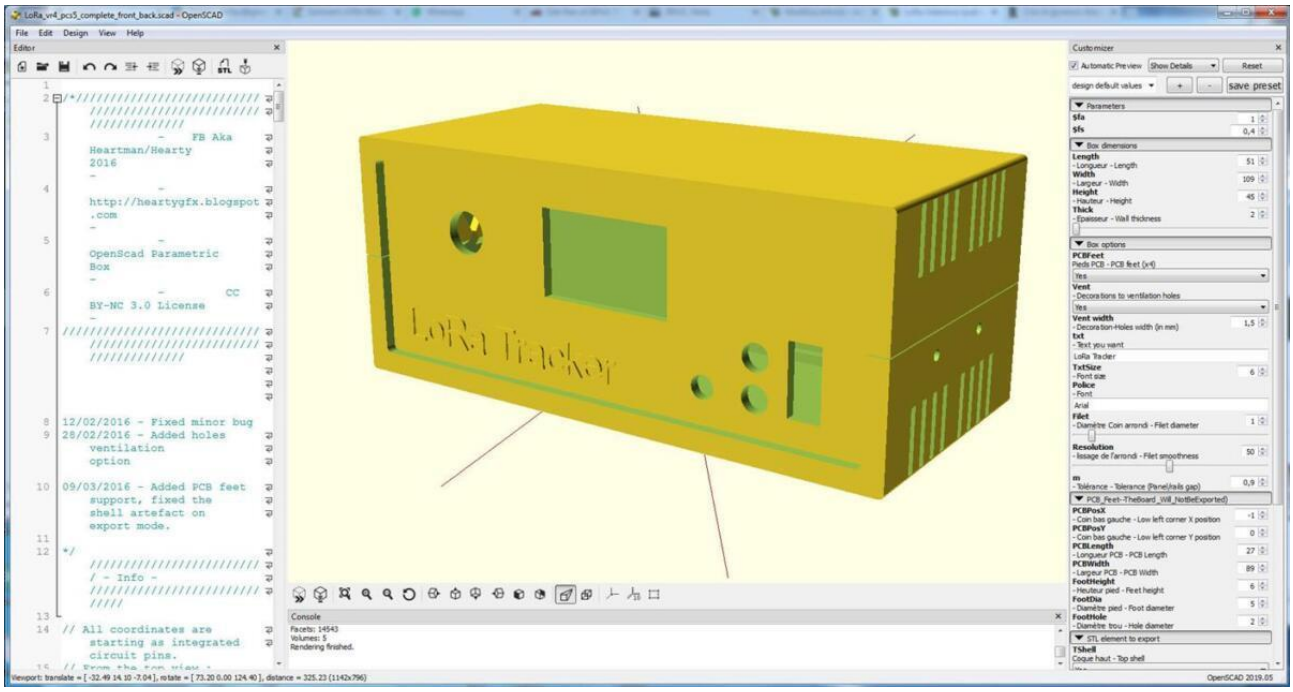

Figure 6-22 : Tracker Case viewed in OpenSCAD

Note: the dimensions of the case are parametric, so that the case can be easily adapted to other projects. The correct Height, Length and Width are visible in Figure 6-22 and are reported below:

- Length: 51
- Width: 109
- Height: 45

Once the project is closed in OpenScad, it can be viewed in “Ultimaker Cura”, which is a free, easy-to-use 3D printing software downloadble here: https://ultimaker.com/software/ultimaker-cura/ . With this Software, it’s possible to do final “slicing” and the generation of the file wich can be processed by the 3D printer.
Here below, a view of the same case in Ultimaker Cura is shown:

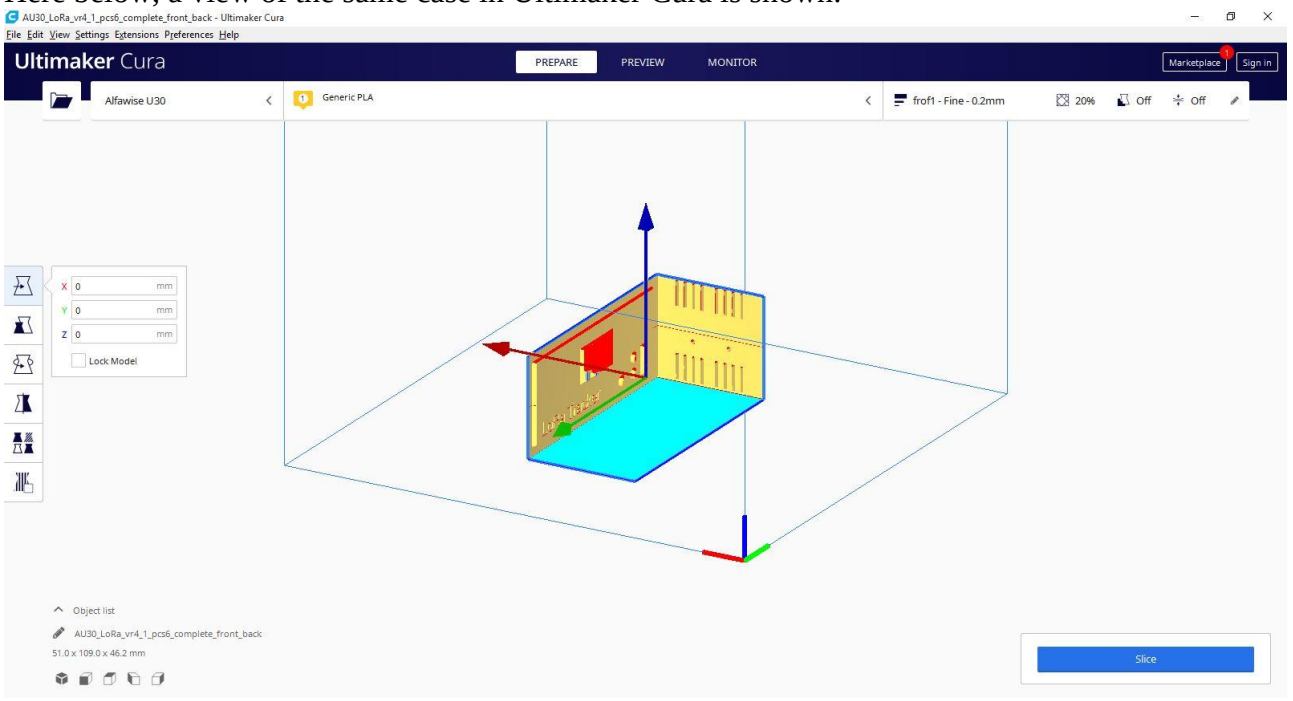

Figure 6-23 : Tracker Case viewed in Ultimaker Cura

7 Initial SW installation

The micontroller used in the LoRa_Beacon project is the ESP32. It is a device with a very high integration scale uC with a significant amount of functions that are not described here for the sake of brevity; on the internet you can find a lot of documentation about it.

The processor can be purchased in the form of modules with standard 2.54mm pinout which contain, beyond the microcontroller, a series of other components including a flash memory which will be loaded with the SW program, a RAM memory to be used as working memory and a USB interface used for debug, direct monitoring and initial SW loading.

The following figure shows the processor module and its physical interfaces. It is worth noting that there are several variants of this module on the market that differ in the number of pins and in the processor chip used; our project requires to use the 38-pin module version with on-board (printed) WiFi antenna, as clearly shown in the following figures.

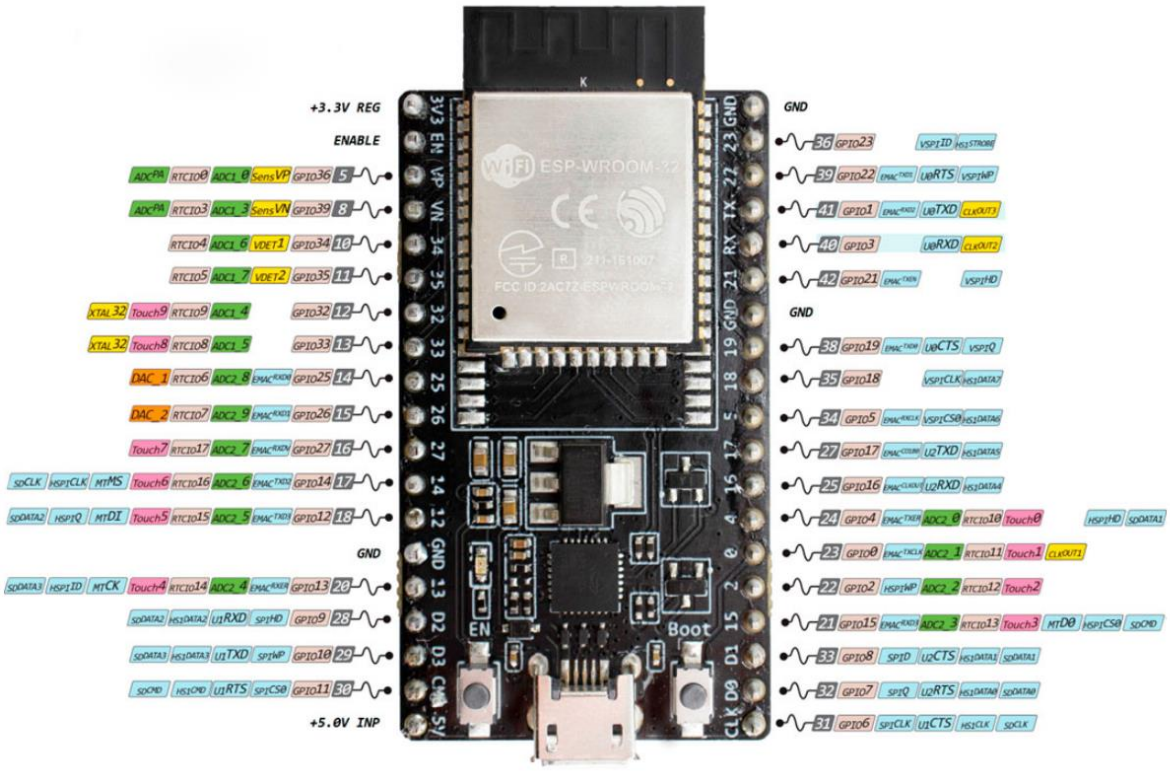

Figure 7-1 : ESP32 processor module with related interfaces

At the time of purchase, the processor comes already equipped with a boot loader that allows the loading of the user SW via USB Interface and a PC equipped with a host GUI.

There are several possible development environments; the one used initially during the early stages of LoRa_Beacon project is based on the SW Arduino IDE platform; with the SW 4.x version the reference development environment has been migrated to the PlatformIO platform which is extremely more performing and user-friendly as setup and management; familiarization with this new platform involves a certain "learning curve" for users accustomed to Ardino which, however, will prove to be quite short and will significantly simplify the work for subsequent developments.

Only the first SW loading method is reported in this document: the big advantage of this method is that it allows you to load the SW onto the HW target without necessarily having to set up any SW development environment (eg Arduino or PlatformIO) and without the need for any SW programming experience.

Obviously the second mode, based on the use of a PlatformIO development platform, is essential if you want to make changes to the SW or set particular options that cannot yet be managed directly via the device's graphic GUI interface.

7.1 First SW loading

The SW loading on the ESP32 processor requires the a PC equipped with a Windows or Linux operating system and the proper host SW.

It is required to make the connection between the PC and the processor using a USB cable headed with appropriate connectors.

Upon connection via the USB cable, the processor will be powered via the cable itself and usually this operation will force Windows to automatically install the required serial port drivers.

To discover the identity of the serial port associated to the ESP32 module, simply explore the list of PC devices (via the control panel); on the Linux platform, recognition of the new interface is usually automatic.

In this phase the ESP32 module can be programmed even if it is not connected to the circuit on which it is to be used.

To download the programming tool from the internet, you can use the following URL: https://www.espressif.com/en/support/download/other-tools

The following figure shows the download page of the tool and indicates the version to download.

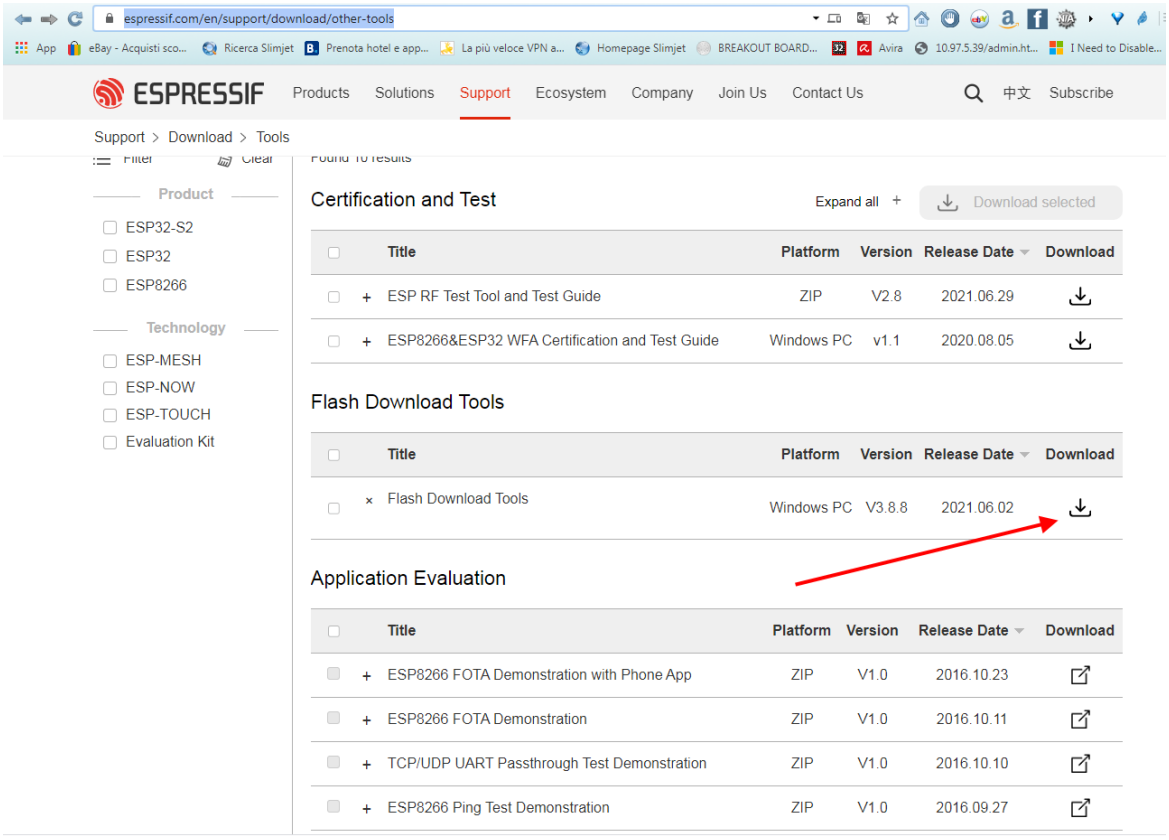

Figure 7-2 : Download page of the ESP32 programming tool

On the site http://iot-bits.com/esp32/esp32-flash-download-tool-tutorial/ it is also possible to find a small tutorial for using the tool; in this paragraph we will describe only the tool under a Windows environment.

Once the Flash Download Tool has been downloaded on the PC, expand the relative archive and launch the “flash_download_tool_3.8.8.exe” file; then select from the small panel that appears "DOWNLOAD TOOL MODE" the values **"chip_Type = ESP32" and "WorkMode=develop"** ; a small panel will appear to be set as in the following figure:

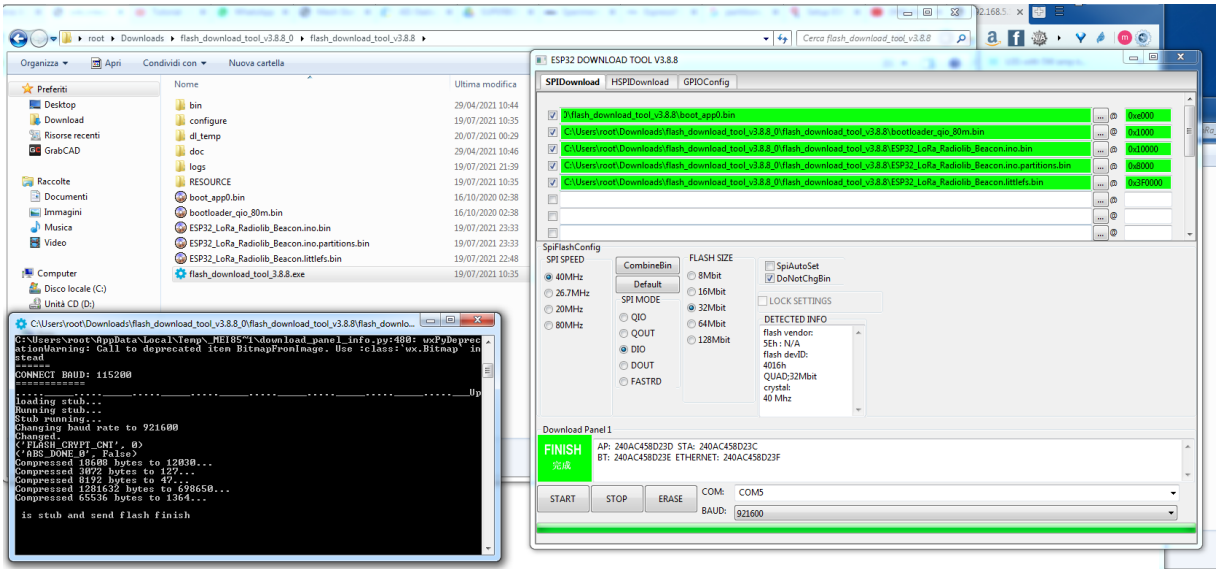

Figure 7-3 : Use of SW Download tool on ESP32 processor

The lines in green are the various SW modules that make up the complete image of the SW: these are 5 load modules that correspond to as many sections of the flash.

The SW modules which compose the complete image are the lines in green of Figure 7-3 and Figure 7-4 are listed here below:

- Processor startup boot_app
- Boot_loader for loading the operating SW
- Application SW image
- Flash partition_table
- LittleFS partition

For simplicity, the image is available on github as a .zip file. It has to be expanded in the saame directory of the SW download tool; **for each section it is necessary to set manually in the right part of the corresponding line the hexadecimal address where that small piece of SW must be loaded**.

These addresses depend by the flash configuration established during the SW development phase and are reported in a suitable readme.txt file present in the same zip file.

Pay close attention to the setup of these values, otherwise the processor will not restart correctly. In case of error it is sufficient to repeat the flashing operation introducing the correct values.

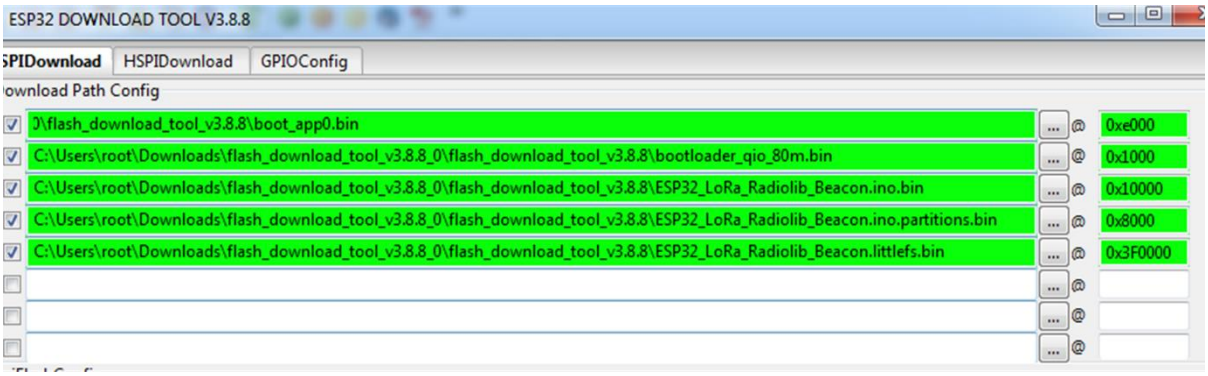

Figure 7-4 : modules do be loaded and related flash addresses

The serial port associated to the ESP32 interface (visible in Windows Control Panel) must be inserted in the COM window; for the other parameters, set them as in the figure. Set the BAUD value to 921600 ; if that doesn't work, try 115200.

Make sure that no other related window is open at this stage, e.g. to the IDE or other application that uses the same serial port.

By pressing the START key, the tool will load the indicated SW image onto the processor: in the command window of the tool, it is possible to follow the progress of the SW loading; if you do not notice the start of the green line on the lower edge of the main window, check the entered values and if necessary try to modify the BAUD value.

Once the SW image is uploaded mount the processor module on the main PCB and check the display for signs of life :).

Figure 7-5 : mesages displayed during first boot

With the SW 4.x version, after the first loading on a brand new device, a default configuration is automatically loaded with a series of preset parameters in order to have a device already able to operate as a tracker. Obviously, for a concrete use it will be necessary to modify this configuration by replacing the default parameters with user parameters such as eg. the callsign of the device, the reference position (latitude/longitude), the Wi-Fi credentials (essential for the iGate type of operation) and the credentials of http://aprs.fi for forwarding the spots to APRS-IS. For customization, refer to the following sections of this document.

If the SW is loaded on a device previously used with the same or a different SW version, the old configuration of the device contained in the FRAM memory or in the EEPROM of the processor will be maintained: this implies that if the new version of the SW is compatible with the previous one, the device will not need any adjustments to the configuration itself, while if the new version of SW is not compatible with the previous one, a punctual check and an adaptation may be necessary or a restoration of the default configuration could be necessary. Compatibility between versions will be highlighted in the release notes of the various SW versions that will follow.

With version 4.x , dedicated compatibility checks are implemented and possibly indications are shown on the display to guide the setup.

Still in SW vr. 4.x the restore mode to the default configuration has been simplified: now **when the device starts up and before any new message appears on the display there is a phase in which the two red and green LEDs will lighted simultaneously for a few seconds... pressing the small key on the device in this phase restores the default conditions** and completely cancels the old configuration of the device.

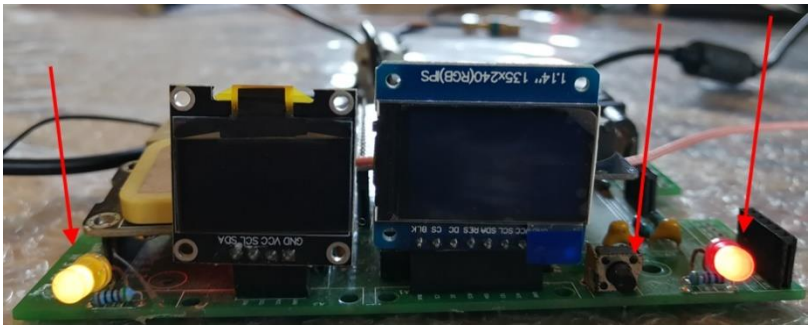

Figure 7-6 : Led During boot phase

After the default configuration is loaded, a new WiFi network called “ESP32-<mac address>” will appear. An example is reported in Figure 7-7. Select this network on the PC (leave DHCP enabled and ESP23 will automatically assign an IP address to the PC**).**

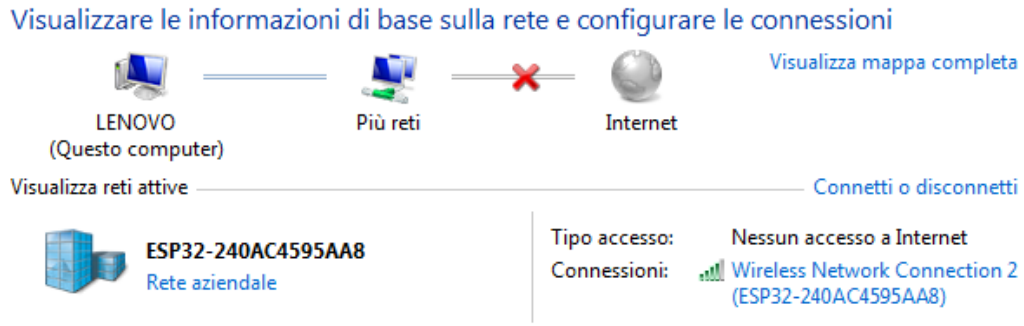

Figure 7-7 : example of Tracker SSID displayed in Windows WiFi management panel

At this point, open a browser such as chrome or firefox and type the following address in the URL bar: http://192.168.5.1 .

If everything is OK, a page will appear as in the figure below:

Figure 7-8 : GUI welcome screen at first access

The values in the windows will obviously depend on the SW loaded and the device used, so the image is for example only. By pressing the "continue" key, the main GUI screen will appear, as in the following figure:

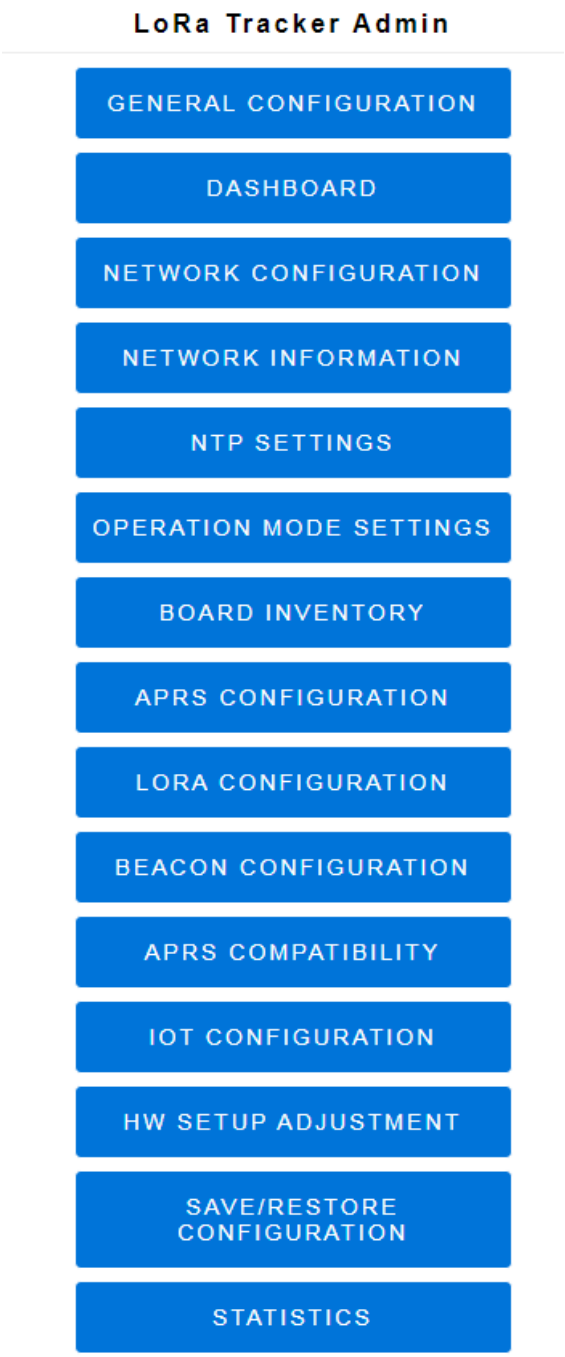

Figure 7-9 : GUI Main page

7.2 Initial setup of the LoRa_Beacon device (any version)

In its default configuration, the SW is not able to operate properly as it **is necessary to insert some specific parameters**, therefore a "configuration of the installed SW" phase is required .

To illustrate the configuration, first of all it’s necessary to introduce the concept of " **operating modes** ".

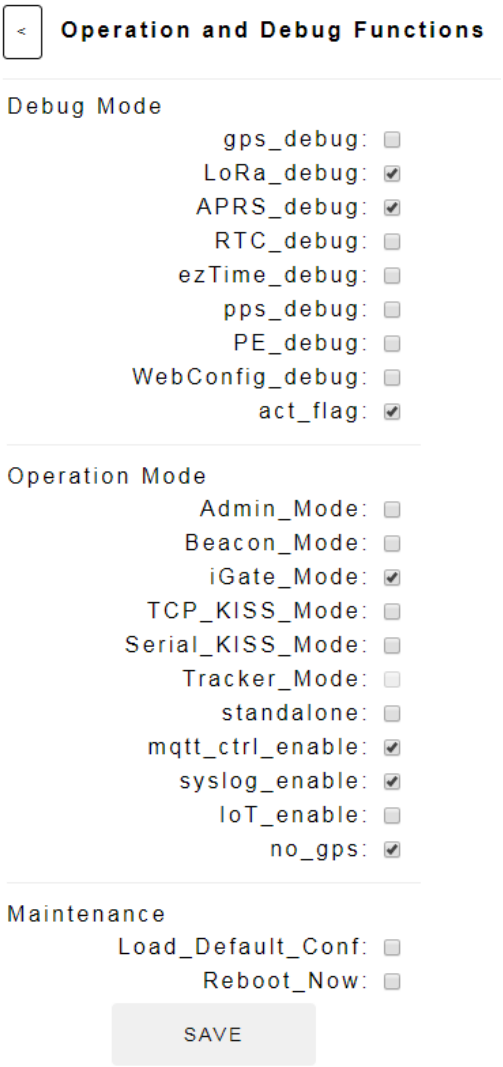

Figure 7-10 : GUI "Operation Mode settings" page

The device can behave in different ways; these different operating modes are generally such that it is not possible to switch from one mode to another without rebooting the SW.

Some operations such as replacing the SW and saving or restoring the operating configuration, will require the freezing of some functions (e.g. the LoRa part and the GPS part). This is the so called "**Admin Mode**".

Clicking on **"OPERATION MODE SETTINGS"** button of the main page, the menu of fig Figure 7-10 will appear. It is the **first section to be configured** after the initial loading of the SW.

A first section of the screen allows you to set a series of debug modes that can be used to track and fix operating anomalies or to set specific device equipment conditions. This section can initially be left in its default condition.

The second section of the screen allows you to set the operating mode for the device; various operating modes are possible which will be individually illustrated below.

The third section allows you to manually reboot the device without removing and reconnecting the power supply to the device, or also restore the default configuration for the device.

The "iGate_Mode" mode prepares the device to operate in APRS LoRa iGate mode and requires all the parameters present in the "APRS CONFIGURATION" screen.

If this item is not selected, the "Tracker APRS" mode is automatically selected, which can also be customized using the same APRS Configuration screen.

The "**BT_KISS_Mode**" , "**Serial_KISS_Mode**" modes allow the device to be operated like a classic KISS TNC for interfacing with SW such as APRX or direwolf.

The "**Syslog Enable**" item allows you to enable the automatic transfer of log messages to an external syslog server. The function will be documented later.

The "**mqtt_ctrl_enable**" item enables the remote control functions of the device via the mqtt protocol. The function will be documented later.

The "**IoT_enable**" item is used to enable a series of IoT-type functions related to the sensors present on the device and is still to be documented.

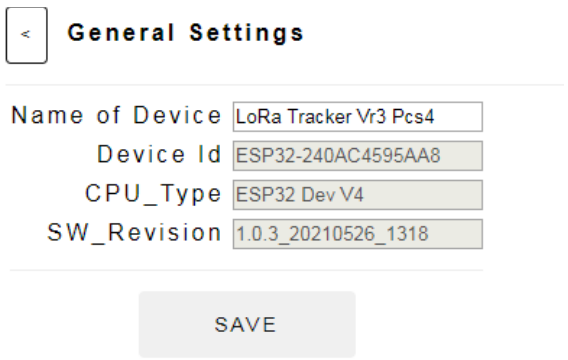

Figure 7-11 : GUI "General Configuration" page

The "**no_gps**" item is used to declare that the device has no GPS or must not assume the presence of GPS on board: it is used, for example, in iGate mode. In this case, the position will be the one declared in the lat/lon fields of the GUI.

A further section to be configured initially is represented by the "**GENERAL CONFIGURATION**" item of the main GUI screen.

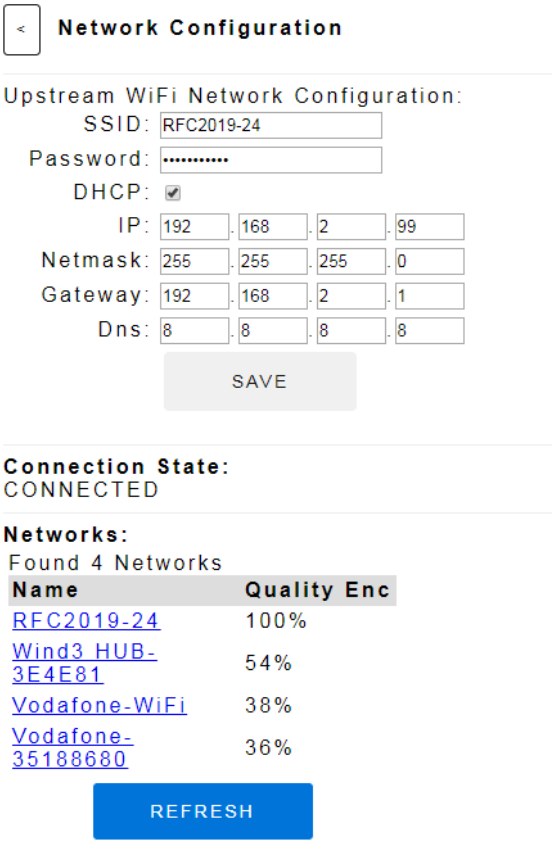

Figure 7-12 : GUI "Network Configuration" page

This section contains :

- SW version
- CPU Type and identity of the device
- Mnemonic device name, used to identify the device and is therefore freely settable by the user

A further section to be set initially is the " **NETWORK CONFIGURATION".**

The device implements two types of WiFi functionality, which can be used for different needs:

- WiFi Access Point "AP mode"
- "WiFi Station" mode

The first mode has been described in the previous paragraph.

In "Station WiFi" mode, on the other hand, the device will autonomously scan and show the available WiFi networks. Note that the ESP32 will work only on the 2.4 Ghz band.

The screen of Figure 7-12 is used to select the WiFi network to connect the device to in Station mode, and allows the connection of the device to the local network and then to the internet.

It is also possible to choose between dynamic (DHCP) or static IP address assignment.

If you access this screen for the first time, the "Network" section may appear empty, but waiting a few seconds, it will automatically be populated with the list of available 2.4Ghz WiFi networks: by selecting one of the networks, the relative password can be set.

Once the setup of the various boxes is finished, it will be necessary to save the configuration with the "SAVE" key.

The "REFRESH" key is used to reacquire the list of available WiFi networks.

To find out the status of the WiFi network connection, see the "**NETWORK INFORMATION**" section of the main page of the GUI: this page will provide the values currently in use by the device for its connection to the local network and to the internet.

It should be noted that the two functions indicated cannot be used simultaneously due to the SW limitations present in the basic SW of the ESP32 processor; therefore, to access the device via WiFi, you will have to use either the AP mode (and therefore the fixed address 192.168.5.1) if the device is set in “stand-alone” mode , or, if the “stand-alone” mode is unchecked and the connection upstream is active, through the address that the device will acquire from the WiFi Access Point of the selected network.

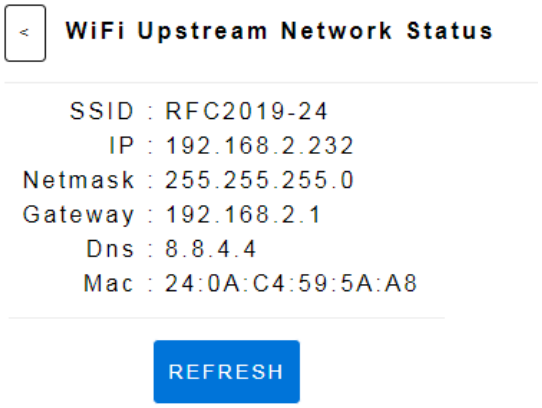

Figure 7-13 : GUI "Network Information" page

The WiFi upstream can be also set by a fairly simple mechanism that uses the front button of the device: holding down the key for a certain time, messages will appear on the display indicating the function to be selected. Releasing the key while a certain function is indicated, will activate the function itself. At that point, it will be necessary to activate the new mode restart the device.

The device restart function can be performed either by switching the device off and on again, or by keeping the key pressed until the wording "Rebot Now" appears.

To complete the configuration of the device's network parameters, it is necessary to specify a "Network Time Server" which eventually be used for synchronizing the local time of the device which will be used for the timestamping of the logs and messages.

For this purpose, the "**NTP SETTINGS**" section of the main page shown in Figure 7-14 will be used.

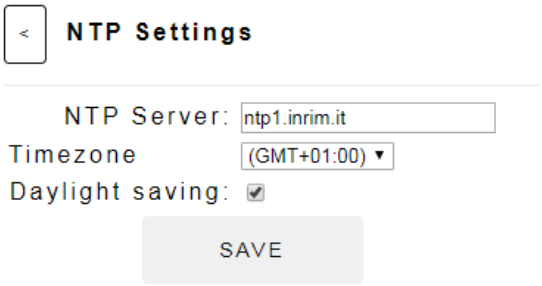

Figure 7-14 : GUI "NTP Settings" page

The NTP server value must be set with the name of the server you intend to use while the Timezone box must be selected based on the geographical time zone.

The Daylight Setting box is used to indicate if daylight saving time is in use.

Setting up an NTP server is important as it allows you to correctly set the time of the device especially in the absence of a GPS module as typically occurs when used as an iGate.

In fact, the device is designed to link its local clock to a GPS module, if present, or in its absence to an NTP server reachable via the internet.

To find out the connection status of the device and its main operating characteristics, it is possible to use the "**DASHBOARD**" screen on the main page of the GUI.

This screen, which will be described in detail below, will show also the local time in the first lines.

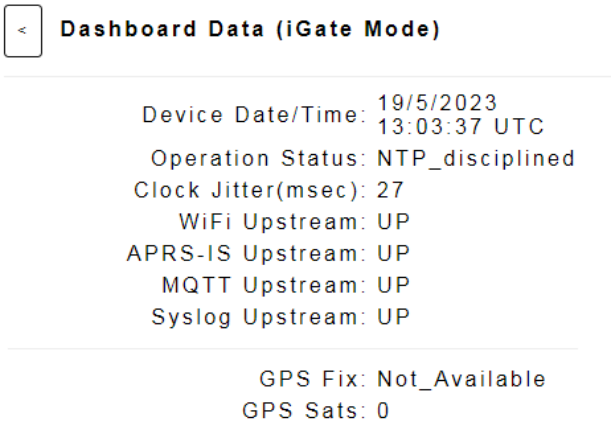

Figure 7-15 : GUI "Dashboard" page

A very important section to consult is the " **BOARD INVENTORY** " page accessible from the GUI main page.

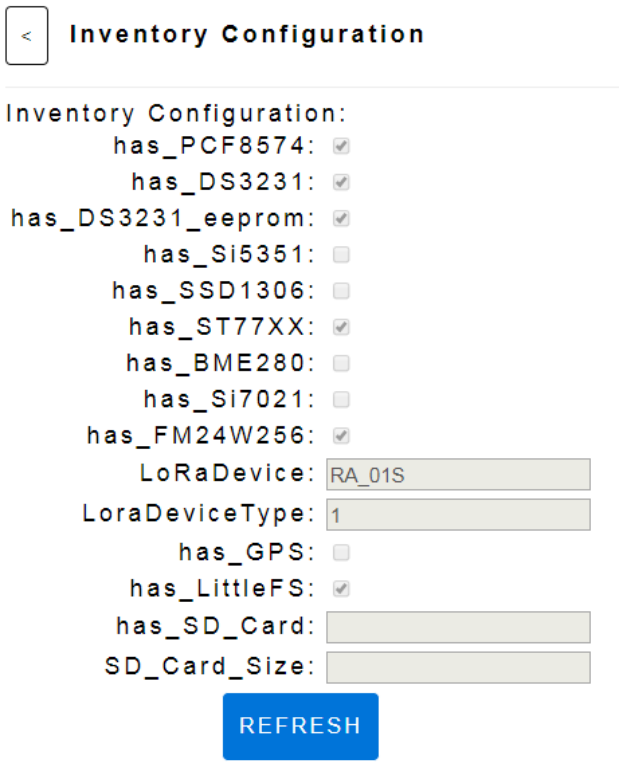

Figure 7-16 : GUI "Board Inventory" page

As a consequence of its modularity, the device may have a series of optional plug-in modules: most of these, will be automatically recognized during the start-up phase: the result of this phase is summarized in the screen above.

A very important feature is the one indicated as "**has_FM24W256**": this is the FRAM memory dedicated to the device configuration data; the absence of recognition of this device prevents the saving of the configuration data in FRAM (however the on-board flash of the ESP32 will be used).

A particular mention to the "act_flag" flag, or activity flag: it is a flag to always be left activated.

7.3 LoRa subsystem setup (any version)

Through the **“LORA CONFIGURATION"** section from the main page of the GUI, you access the configuration of the LoRa subsystem.

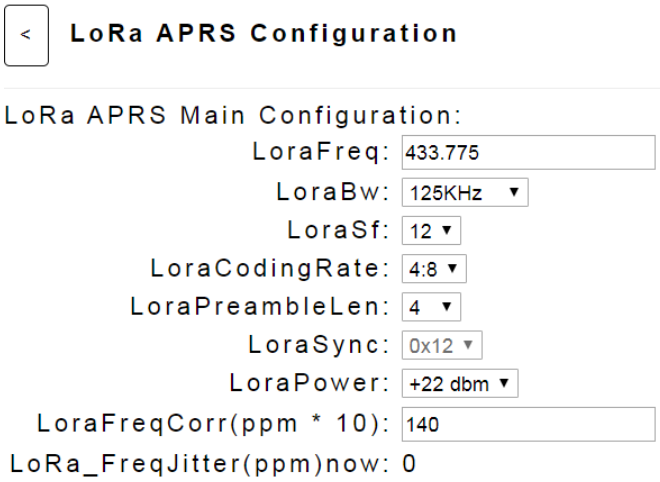

Figure 7-17 : GUI "APRS Configuration" page

As said before, different types of LoRa radio modules can be equipped on the device: to find out the currently installed LoRa module, it is possible to consult the "**Inventory configuration**" page.

The LoRa parameters will depend also by the installed LoRa module.

In the LoRa configuration page only a subset of “high-level” parameters described in the module programming manual can be configured; the low-level parameters are not shown because they are already optimized for the HW Platform and they should not be changed by the end-user.

Many of the parameters are intuitive and their setting depends on the type of experimentation you intend to do.

In principle, for two devices to communicate with each other, the LoRa parameters of the two devices must coincide.

The only parameters that can differ are the transmission power value and the frequency correction value.

This last parameter can typically be left at zero for devices that mount high frequency precision LoRa modules (e.g. the E22_400M30S model or other second generation modules equipped with TCXO) while it will be set experimentally for first generation LoRa modules which do not have good frequency accuracy or for second generation ones without TCXO. **For this purpose, the “FreqJitter” value measured in real time for the packets received and shown in the last line of the screen can be used as a suggestion.**

As a verification of the correct frequency alignment, the transmitted signal can be observed with SDR receiver to evaluate the offset value in ppm (parts-per-million) to be set.

7.4 APRS subsystem setup (any version)

By selecting the " **APRS CONFIGURATION** " section from the GUI main page, you access the page that allows you to configure the parameters for the APRS service.

As it is now clear, the device can operate in two different APRS modes:

- iGate/repeater mode;
- Tracker mode.

The parameters to be configured for the two modes are different even if similar. The Figure 7-18 covers both modes.

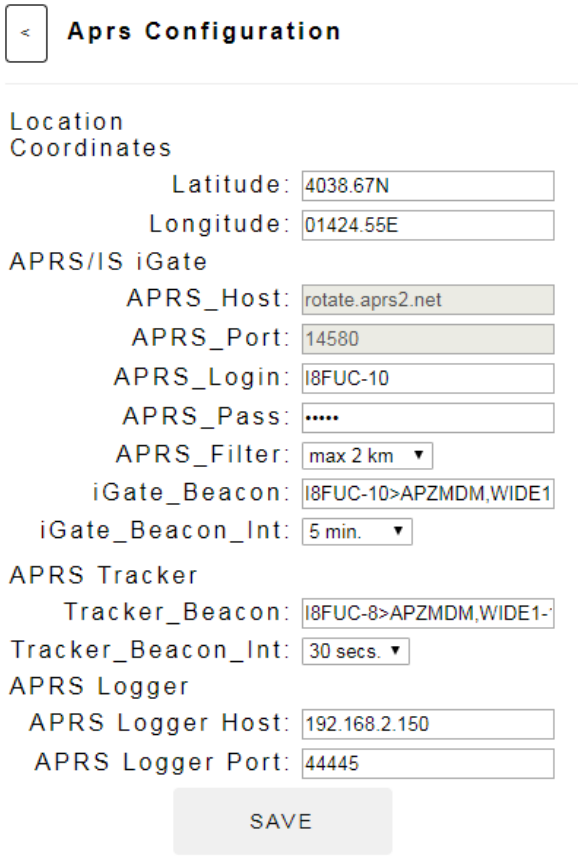

Figure 7-18 : GUI "APRS Configuration" page

7.4.1 APRS subsystem setup for iGate and Tracker mode

For the iGate mode, the device must first of all be able to connect to the Internet and to a suitable ARPS-IS server.

If the internet gateway in use in the Wi-Fi network is protected by a Firewall, it must be configured to allow outgoing traffic on the IP port used by the APRS-IS server and which will depend on the chosen APRS-IS server.

Finally, a Login and a Password will be needed for connecting to aprs server. The password is a numeric code that can be calculated starting from the Callsign at this link: https://www.iz3mez.it/aprs-passcode/

The Igate position can be automatically acquired , if there is a GPS module installed on the board, which has reached a 3D fix; otherwise it is possible to manually enter the position to be reported.

To determine the value of the coordinates to be reported, the native APRS format must be used; that is to say:

- latitude: 2 digits for degrees + 4 digits with dot after the first two digits for first and second seconds as required by APRS followed by the letters N/S for north or south.
- longitude: 3 digits for degrees + 4 digits with dot after the first two digits for first and second seconds as required by APRS followed by the letters E/W for north or south.

Again for the iGate functionality, a parameter must be inserted that allows filtering the beacons to be transmitted on the LoRa RF network and coming from APRS-IS: the only mode supported by the GUI is the "maximum distance from the iGate position".

Note: the LoRA Channel could easily saturate if wrong settings are used! Keep tx rate as low as possible and limit the distance range to the necessary

At this point, is possible to enter the required parameters to create the beacon string to be transmitted to the LoRa network and to the APRS/IS network.

This string will be automatically composed on the basis of the parameters previously entered in the GUI , as illustrated in the example below:

parameters entered in the iGate_Beacon or Tracker_Beacon field:
I1XYZ-10, WIDE1-1, LoRa, &, L

resulting string beacon (location field in plain text):
I1XYZ-10 >APLS01, **WIDE1-1** :!GPS_LAT **L** GPS_LON **&** LoRa -32B125S12C8P10

resulting string beacon (compressed location field):
I1XYZ-10 >APLS01, **WIDE1-1** :! **L** 9 w?#R-GG **&** !!G LoRa -32B125S12C8P10

The "**GPS_LAT**" and "**GPS_LON**" strings indicate respectively the latitude and longitude acquired by the GPS module, or the values entered in the latitude/longitude fields of the screen in the "APRS Configuration" if no GPS module is installed.

The beacon format is compliant to the actual APRS specifications. The only fields that can be set by the user are the red ones.

The string "*APLS01"* is the "destination" field of APRS packet and identifies the SARIMESH Device. The list of all sources is available at http://www.aprs.org/aprs11/tocalls.txt .

The field following the symbol " **!** " represents the location of the device: if it begins with a numeric character, it is assumed that the location is in uncompressed format, otherwise it is assumed that the 13 characters that follow represent the location in a compressed format; the compression algorithm is always described in the APRS protocol specification.

With the SW 4.x version, the software decode locations in both formats, while in transmission it is possible to indicate the method to use; the use of the compressed mode allows to reduce the length of the location field to the advantage of a higher transmission speed of the single packet.

The characters "L" and "&" present in the above example can be replaced with other characters in order to modify the symbol with which the APRS icon will appear on aprs.fi and similar websites; for a complete list of icons, it is possible to consult the following URL: https://www.iz3mez.it/aprs-server/simboli-aprs/ (thanks to Giovanni IZ0CZW for the report). It is suggested not to change the character L as for standard use it indicates a LoRa device.

The last part of the beacon is automatically generated by the SW and summarizes the working conditions of the device; in particular the meaning is the following:

- working conditions (example):
-32B125S12C8P10

- meaning of the fields if present:
"-" indicates that the beacon was generated in a blacklisted area (for testing only)
32 Sequence Number "module 100" for packet (is incremented each beacon package generated)
B125 Bandwidth used for transmission in Khz (without decimals)
S12 Spreading Factor used for transmission
C8 Coding Rate used for transmission
P10 Preamble Length (number of symbols)

In the generated beacon strings, there may also be two other fields enclosed by parentheses containing one or more subfields which reports the first and last node crossed by the APRS packet; this easily allows to have an indication (in particular for the second case) of the receiving condition of the packet by the node indicated in the first subfield of the parenthesis: the meaning can be deduced from the following example:

- device field crossed:
(IQ8SO-10 -120 -16 186)

- subfield decoding:
IQ8SO-10 callsign of the crossed node
-120 signal strength of the spot received from the crossed node in db without decimals
-16 SNR of the spot received from the crossed node in db without decimals
186 frequency shift measured by the crossed node receiver in Hz

These optional fields are included in the comment part of the beacon and are clearly NOT APRS standard. They can be partially or completely omitted in order to have packets issued in compliance with the APRS standard.

In order to optimize the channel occupation, the device will transmit the optional fields only every five packets of a sequence of one hundred packets. For all other packets in a sequence of one hundred, only the sequence number will be transmitted. The function of the sequence number is to be able to highlight packet loss situations; it is obviously useful only in the experimentation phase to deepen the analysis of the behavior of the LoRa system.

Finally, it is possible to specify the time between two beacon if the "Agile beaconing" function is not used.

In "tracker mode", the setup of this screen is very easy; the only required parameters are the string to be sent as a beacon (according to the format explained for the iGate case) and the period of the beacon in sec.

7.4.2 APRS subsystem setup for connection to a service server

The device is able to connect to a server dedicated to collect data relatedto the APRS spots received from the LoRa network; this interface is based on a simple UDP protocol and allows to show spots on a geolocated map.

The documentation of this interface will be released at a later time.

7.5 APRS Compatibility Setup (any version)

In the APRS applications, the transmitted packets must be "encapsulated" to be recognizable as correctly transmitted and received entities; for this purpose, different methods are possible.

In the SARIMESH implementation, the preferred encapsulation is the standard AX.25 which has the best compatibility with other data transmission applications in use (eg TNC KISS mode).

There are other implementations of LoRa APRS with different type of encapsulation: in particular, the mode widespread in the Austrian and German implementations is also supported, which we briefly indicate as the "OE_Style" mode.

The SW is able to receive packets in both encapsulation methods, and performs adaptations necessary for the compatibility of the transported data; in transmission it is necessary to specify which type of encapsulation to use (see the Figure 7-19).

In the compatibility screen it is possible to specify which non-standard options you intend to use and which not.

In particular, the following **options are present (which only influence the transmission behavior of the devices):**

- **payload encapsulation** : already described above;
- **payload style configuration** : enable or disable the additional fields described in the previous paragraph (working conditions and parameters of the crossed nodes);
- **repeater operation configuration** : enable the digipeater function or not (available only in iGate mode);
- **location compression configuration** : enable compression of location fields;

- **TX Agile Beaconing configuration** : enable adaptive beaconing function (see below);
- **Beacon BlackList configuration** : enable blacklist for protected areas;
- **LoRa sync word configuration** : set the sync word value to be used in LoRa packets.

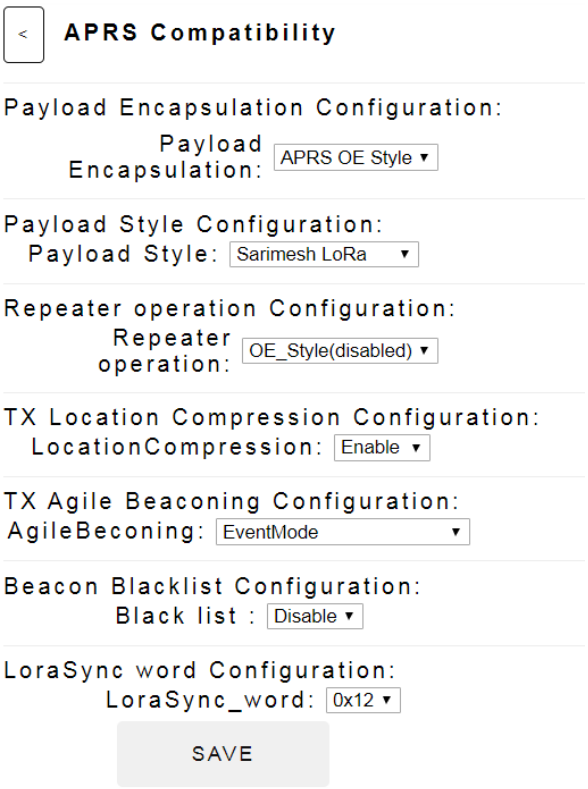

Figure 7-19 : GUI "APRS Compatibility" page

Let’s spend some words about the beacon interval algorythm.

The function of beacons is obviously to report the status of the device that transmits the beacon: the nature of this data can be very different and generally includes a "location" as the main data, (i.e. an identifiable position or via geo coordinates or through a QRA locator); in some cases it is also possible to avoid transmitting this basic data because it is not subject to change (as for example for a weather station or for a fixed iGate); in these cases the data is generally transmitted at bigger time intervals, avoiding the transmission of data that does not change over time.

In the classic use of APRS, the main aim is to highlight the movement of the device over a territory, so it makes sense to transmit beacons only in case of position change, in the case of LoRa experiments the "radio mapping" aspect of the territory becomes interesting and important in order to evaluate the radio coverage reached thanks to LoRa protocol.

In this last situation it therefore becomes interesting to be able to issue beacons with a logic linked precisely to the use of the collected data... for example. use the change of position or the path followed by the mobile device as a reference basis for the emission of the spots: a

example is to detect the radio coverage map of a certain area or the level of coverage of a certain route (eg a mountain route or a particular road).

The "Agile Beaconing" function aims to dose the beacon emission times according to various criteria in order to obtain the above objective.

In the specific case, the data acquired in real time by the GPS (in particular position, direction of travel, time, speed and altitude) are used to appropriately dose the emission times of the beacons.

There are two main sub-modalities: modalities based simply on the distance traveled or modalities based on "events", meaning by this term the sets of predefined and significant conditions of the path being followed.

It is then possible to completely disable the Agile function, in this case falling back to the basic beacon emission mode, i.e. the constant time mode; in any case, even if the "Agile" mode is active, the time parameter indicated in the APRS function setup screen remains active so that beacons can be sent at worst every certain time if there are no events.

It should be noted that given the nature of positioning data derived from GPS, it is normal for such data to be affected by an error in particular for the position which can be evaluated in a few meters and which depends on the GPS signal reception conditions: to avoid excessive sending of beacons due to these errors it is however avoided to transmit position data that differ by a few meters.

A similar limitation exists for the time interval between beacons: in this case, to avoid situations of congestion, transmissions of beacons that are too close in time to previous beacons are avoided;.

In any case, it is always possible to immediately send a beacon manually by pressing the key on the devices.

A further function linked to beaconing is to avoid issuing beacons when the mobile phone is in particular areas from which you want to avoid transmitting data for privacy or technical reasons, e.g. to radio congestion problems.

For this purpose it is possible to define a set of areas, identifiable by means of a center and a radius, such that if one is within one of these areas the beacon is suppressed.

This function can be enabled via the compatibility screen.

All the characteristic parameters of the beaconing function are currently NOT modifiable from the GUI but require modification of the parameters present in appropriate tables in the source code; in the near future, if the function proves to be of real interest, a suitable screen can also be implemented for this function to allow the modification of the various parameters from the GUI.

7.6 Dashboard screen (any version)

The Dashboard allows to have a general overview of the device working conditions, from an operational point of view and from a configuration point of view.

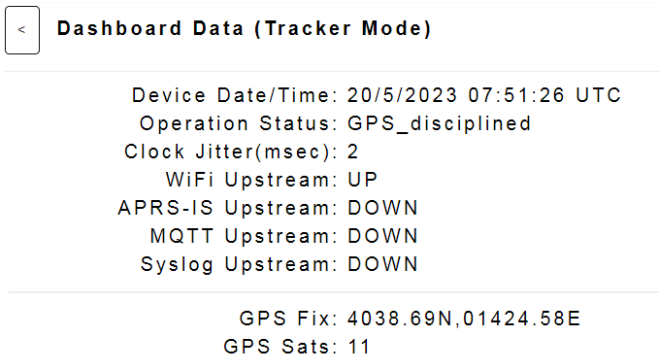

Figure 7-20 : GUI "Dashboard" page showing GPS fix

AS visible in figure Figure 7-20, the first section shows the date and time of the device and indicates a device linked to a GPS. In Figure 7-21 the dashboard indicates the time acquired by NTP protocol. Moreover, it's possible to see if the device is operating in a standalone mode or not.

The Clock Jitter value in msec is also reported, i.e. an indication of how much the local time varies over time in relation to the type of synchronization in progress.

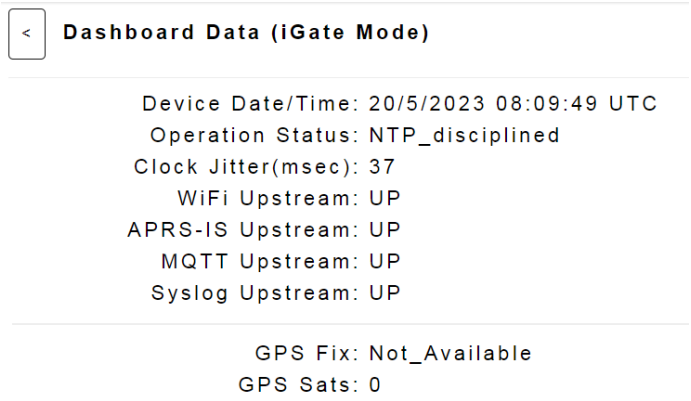

Figure 7-21 : GUI "Dasboard" page showing time acquired by NTP

The status of the upstream WiFi connectivity, the status of the connection to the APRS-IS system and the status of the MQTT and syslog subsystems present on the device are then reported; for basic use of the devices in tracker mode, no functions are required to be active, while for use as an iGate, the only function that must necessarily be active is the WiFi connection and the connection to APRS-IS.

This section also reports the status of the GPS subsystem and in particular the current FIX value and the number of satellites on which it is based.

This section also lets you know in which operating mode the device is; from the example above, the device is in tracker mode, while the Figure 7-21 shows an example of a device that operates in iGate mode. In this case eg. it can be seen that the GPS is not active and therefore the local time is derived from the NTP internet protocol; as a result it is highlighted that the local clock has a jitter of a few tens of msec as is normal for the time derived from the internet.

The Jitter value is important for some features, described below, that require a pseudo-synchronous timebase for their operation.

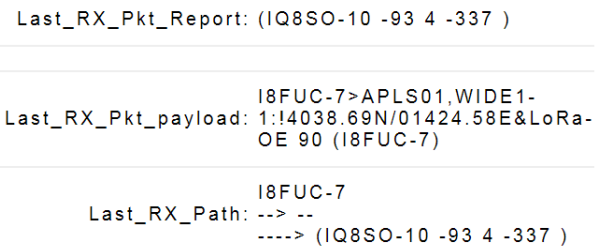

Figure 7-22 : Dashboard showing "last packet" information

A second section of the DashBoard, shown in Figure 7-22, summarizes the status of the receiving part of the device.

In particular, this section reports only the last packet received and for this packet it shows:

- the "reception report" that the node will eventually attach to a digipeated packet or to a packet transmit to APRS-IS: this record contains the identity of the node that is receiving the packet, the signal level in dBm, signal/noise ratio in dB and frequency offset in Hz.
- payload contained in the received packet: if the packet is received as "CRC Errored" it means that the packet, despite having been recovered, is affected by uncorrectable errors which may

contain incorrect characters or unprintable symbols; the strategy is to report the recovered content in any case for any subsequent actions.

- path of the received packet: only the identifiable components of the path are reported, i.e. usually only the original source of the received packet (regardless of any repetitions that the packet has been subjected to) and the last node the packet passed through.

The next section of the Dashboard, shown in Figure 7-23, will report a series of counters related to the traffic of packets passing through it; the meaning of the various counters is quite intuitive based on its name.

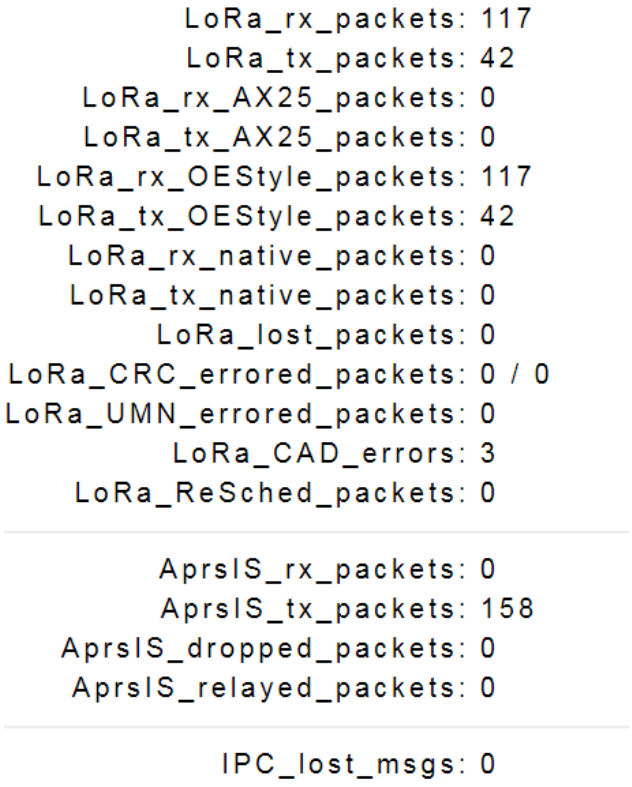

Figure 7-23 : GUI Dashboard statistics – part 1

In particular, the packets are listed according to their encapsulation (AX_25 or OE_Style), according to the direction in which they have been seen (TX or RX from the point of view of the node), according to the type of recognized payload (APRS or native format, described in 7.10); the lost packets are then reported (due to unavailability of the channel being transmitted) and the packets received but revealed to be CRC_Errored, ie affected by uncorrectable transmission errors.

The packets that failed to transmit due to the occupation of the radio channel revealed by the CAD (Channel Activity Detection) mechanism are then reported; a further parameter is the number of packets rescheduled in the transmission phase again due to radio channel occupation problems.

From APRS-IS side (APRS connection to and from the internet) the packets received and transmitted are reported as well as those repeated towards the LoRa network or suppressed as they do not comply with the repetition policies towards the LoRa network side.

The last parameter reported is the number of messages lost in the interprocess-communication (IPC) system used internally by the SW to make the various components of the SW interact with each other: this parameter is an indicator of SW congestion.

It should be noted that the counter of messages received from the APRS-IS side takes into account the filtering applied in that direction by the node: in particular, the only filter applied is the distance filter (intended as distance between the transmitted beacon and the iGate position).

The last section of the Dashboard shows a series of data relating to the node in its entirety; in particular the first parameter shows the moving average of the transmission time values (OnAirTime in msec) of the transmitted packets, the DutyCycle in % of channel occupation or how much of the transmission time of the node has been actually used by the node, and a further parameter (heuristic) which gives an approximate indication of the level of congestion of the radio channel as perceived by the node).

Two data relating to the LoRa level follow: the value of "ExstimatedNoiseLevel" (ENL in dbm) evaluated for the site where the node is installed and the moving average of the Frequency Difference values measured on the received packets.

The ENL value is a heuristic estimate of the noise level in dbm of the site calculated considering only packets received with a negative signal-to-noise ratio (SNR): therefore this parameter will be present only if the node has received packets with a negative SNR. It represents the dbm value (evaluated by the lora chip in its internal operation) of the signal intensity at the LoRa chipset terminals regardless of the "process gain" values that the LoRa protocol allows to obtain.

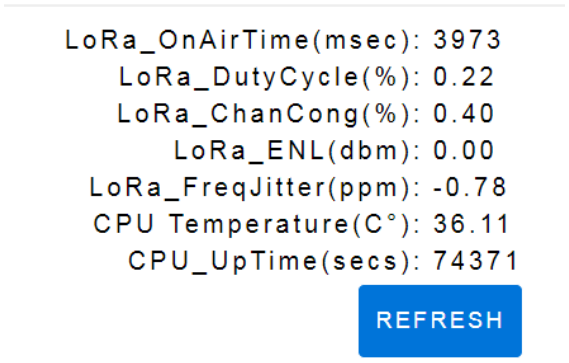

Figure 7-24 : GUI Dashboard statistics - part 2

The indicated Jitter value (in parts-per-million) is the moving average of the frequency difference values divided by the value of the transmission frequency that the node was able to measure on the received packets regardless of where these packets were transmitted : therefore it represents a form of estimation of how much the current node could be "retuned in a frequency" to operate better in the LoRa network in which it operates.; this value is presented in the node's LoRa setup screen to be effectively used as a correction value to be applied to the LoRa frequency setup.

The last values presented are an estimate of the temperature value of the node CPU and the value of the operating time (in sec.) since the last reboot of the node itself.

7.7 HW Setup Configuration

With the 4.x version of the SW, the use of the SW on HW platforms other than the Sarimesh one is no longer officially supported, substantially not so much for a functional problem as mainly because the number of platforms available on the internet and potentially compatible with the SW has increased considerably, so there is the economic problem of having physical prototypes and a lot of time available for carrying out the necessary non-regression tests.

In any case, since the SW is very modular and based on decidedly standard HW components, it is easy to adapt the code to run on almost all boards based on the ESP32 processor and which have the necessary HW components (e.g. a GPS module and a LoRa chipset) .

If the SW is used on HW platforms other than SARIMESH platforms, in general it may be necessary to appropriately set some features of the SW in order to adapt to the different characteristics of the HW; in particular what may differ, within a certain type of HW card, is the value of some pins of the processor to which the peripherals present on the card are connected.

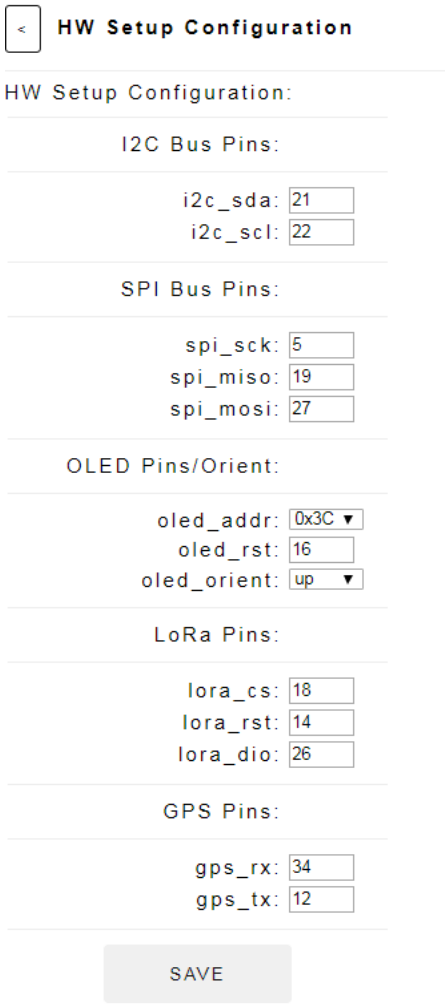

Figure 7-25 : GUI "HW Setup Adjustment" page

The screen shown in Figure 7-25 allows you to set the pins of the ESP32 processor to be used for the following functions:

- I2C bus
- SPI bus
- OLED pins, addr and orientation
- LoRa specific pins
- GPS specific pins

The values to be set can be found in the hardware documentation of the devices used.

For the SARIMESH HW this page shows the values used but not modifiable as they are invariant.

7.8 Syslog logging setup (any version)

This feature was already introduced in SW Vr. 3.x although it has not been documented so far.

The function consists in the possibility of sending event or error type messages to an external server in the cloud by exploiting the syslog protocol widely used in internet applications.

The function does not yet have a graphical interface and has been implemented as a "proof of concept" in order to acquire operating data or error conditions of an experimental network.

The function obviously requires that the nodes have internet connectivity.

A typical use is, for example, to coordinate and control the operation of nodes involved in an experimentation phase such as the one described in one of the following paragraphs of this document.

If there is anyone interested in using this feature, it is possible to consult the source code of the SW or contact the author of the SW.

7.9 MQTT subsystem setup (any version)

This feature was already introduced in SW Vr. 3.x although it has not been documented so far .

The function consists in the possibility of exploiting the MQTT protocol, very widespread in the IoT sector, to interact with an appropriate proxy server located on the internet cloud.

The typical use of MQTT is to support IoT functions such as e.g. the collection and display of data such as temperature, pressure, etc. from one or more remote nodes allowing for an easy and flexible organization and presentation of data.

A second use, less known but extremely interesting, is to operate as a proxy to carry out remote maintenance operations on a certain node without absolutely requiring any reconfiguration of the internet gateway of the remote node to be controlled.

In the "proof of concept" created in Sarimesh SW, this second method is implemented and used in order to acquire status data of a generic remote node and perform remote actions such as the remote reboot.

The MQTT function being very experimental is only supported at the source code level of the SW and no graphical interface for its management has yet been created; if there is interest in discovering and trying to use this function it is possible to consult the source code and/or contact the author of the SW for support.

7.10 Native Beaconing Subsystem Setup (any version)

The Sarimesh SW Sarimesh contains, starting from version 3.x , an undocumented feature that we define "Native Beaconing": it is an experimental function that operates the node in different modes with a static "time slicing" type approach.

First of all, this function assumes that a node operates in a GPS_Disciplined or even NTP_discipled mode in such a way that the local time values are sufficiently "precise" , that is having tolerances within a maximum of a few tens of msec.

In practice, the operating time of the node is managed dynamically on the basis of a certain "usage schedule" consisting of a certain time base (e.g. 3 minutes) aligned in a certain way (e.g. on a

time base of UTC clock): this program specifies how to operate in the subintervals of this program duration time (slice time).

In each time slice, the node will be able to operate in a completely different settings, i.e. with different frequency values, emission mode and power as well as packet encapsulation typology and relative payload.

An example that has been used for about a year in a particular experiment carried out on a 120 km radio section and documented on the Sarimesh.net website (http://www.sarimesh.net/lora-beacon-propagation-test/ and http ://www.sarimesh.net/lora-propagation-test-bed-2/).

The experiment was based on a pair of nodes operated in Native mode for 100 sec for simulate very marginal conditions of the link (SF12 / BW 10.4 Khz) and for the remaining time (80 seconds) in APRS mode with the standard parameters (SF12 / BW 125 Khz) on a different frequency.

The objective was to acquire and compare the typical parameters of the connection in the two different operating modes.

For setting the mixed operating mode, a dedicated page has been prepared in the GUI which is illustrated in Figure 7-26.

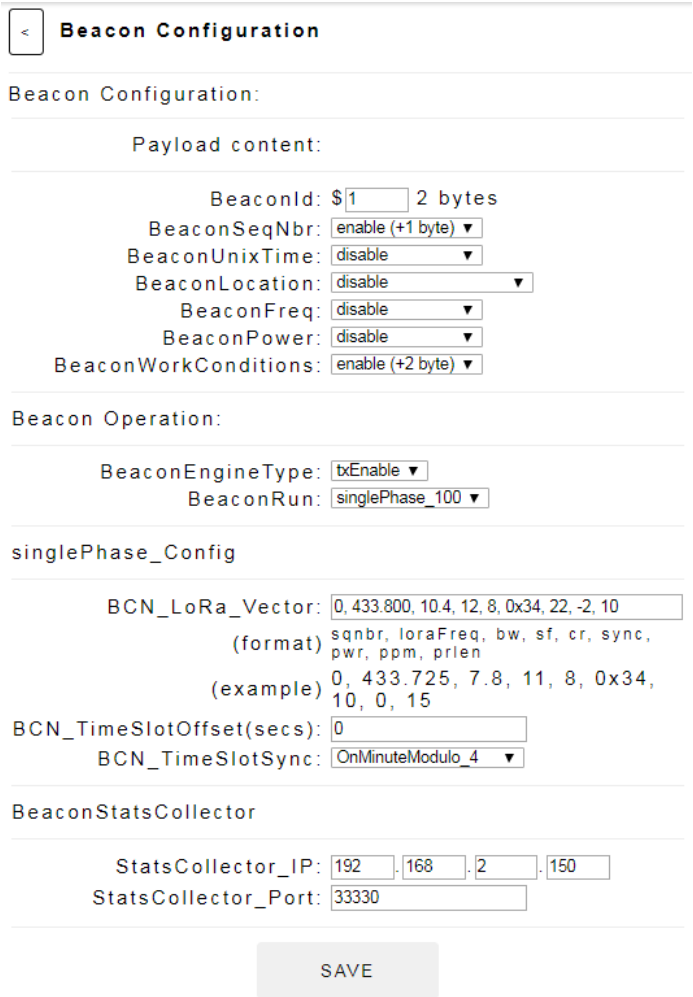

Figure 7-26 : GUI "Beacon Configuration" page

To activate the mixed operation function it is sufficient to activate the Native Beacon function in the "Operation Mode" page.

A "native" default payload has been defined in the GUI. It contains all the data selected in the first section of the screen and arranged in very few bytes; to identify a node, a byte with a numerical code is simply used; the maximum number of active nodes in a given experiment depends on the slicing program defined below.

The coding of the various parameters is defined in the SW and for those interested just go and consult the sources of the SW.

The "Beacon Operation" section follows which specifies the type of program and whether the various phases are reception-only or bi-directional.

The "BeaconRun" item defines the various types of use of the program time and can assume SinglePhase values (of varying duration from 18 sec to 100 sec) or tuning mode in which the beacon automatically follows a sequence of transmission/reception phases with different frequency values to find out any frequency shift).

Then follows a section that specifies how the Native Beacon phase takes place: in practice it synthetically specifies the working conditions at the LoRa level to be used and the time offset with respect to the program time from which to start with the native beacon phase.

The APRS type phase is defined to complement the program duration time and uses the LoRa / APRS parameters of the node as described in the relevant screens.

To ensure the coexistence of several nodes in an experiment phase, the "Identity of the node" parameter is exploited with the simple logic that in a certain time interval only a certain node can transmit; in particular, the NodeId parameter defined above is used as an index to activate the transmission phase of only one of the nodes in a fraction of the time assigned to the "native beacon" phase of the program; this operating mode is only available by selecting a duration of the native beacon phase of 100 sec.; for all other values a single pair of nodes is always assumed as the target of the experiment.

The function described above is, as can be seen, still very experimental and represents only a "proof of concept" to be possibly better defined and exploited in the future.

8 Remote Debugging Interface

The devices of the LoRa Beacon family are equipped with a remote debug interface that allows access to a series of features designed for advanced use of the devices or to access the functions for monitoring operation in real time without requiring connection to a computer via USB or serial and therefore also usable remotely for devices such as located in a point connected via the Internet.

Access to the Remote Debug interface uses the telnet protocol equipped with a simple password-based security system.

Access to this functionality can take place either using a classic Telnet client available for all existing computer platforms, or via a web application using any internet browser.

Regardless of how you access the Remote Debug interface, a series of commands are available that correspond to as many monitoring or debugging functions.

The two access modes are described below, followed by the debugging commands available.

8.1 Access to Remote Debug IF via Telnet Client

To use this access method, simply obtain any Telnet client such as the classic "putty" downloadable from the following URL: http://putty.org . In Figure 8-1 you can see the download page.

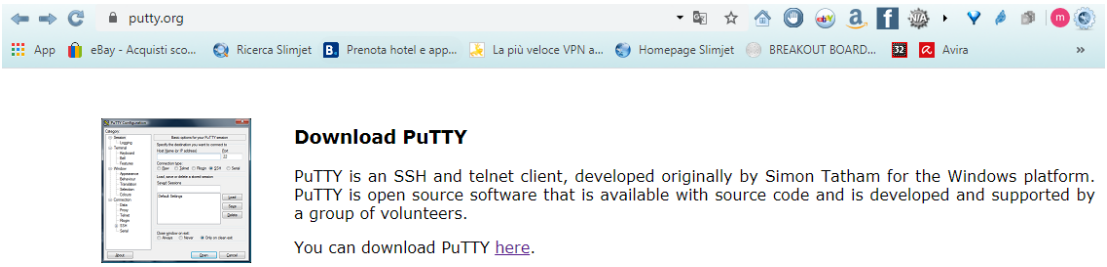

Figure 8-1 : Putty Download page

Once you have downloaded the putty.exe file corresponding to your PC platform that you intend to use, simply start the file without the need for installation.

From the screen of Figure 8-2, select the following options in addition to the default ones:

- Session: Other --> Telnet
- Host Name: LAN IP address of the device

The device IP could be:

- 192.168.5.1 in case of device in “stand-alone”
- The static IP decided by the user, if DHCP is not enabled
- IP assigned by Wifi router if DHCP is left enabled on the ESP32 (see your WiFi router mangement interface to identify the IP assigned to ESP32)

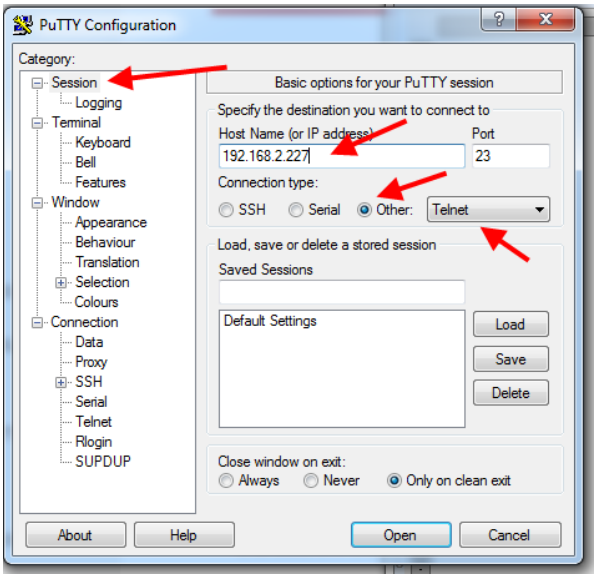

Figure 8-2 : Putty Setup - 1

Follow Figure 8-3 for the correct settings of Carriage Return and Line Feed:

- Terminal: Implicit CR in every LF

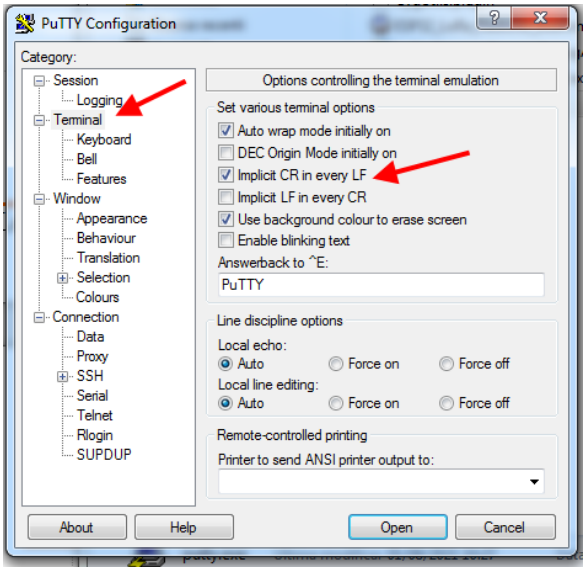

Figure 8-3 : Putty Setup - 2

At this point click on “Open”. If the connection is correctly esablished, a screen similar to Figure 8-4 will appear asking for a password: default is “esp32”.

Figure 8-4 : Initial Telnet screen

By entering the password, the device debug console command interface shown in Figure 8-5 will appear.

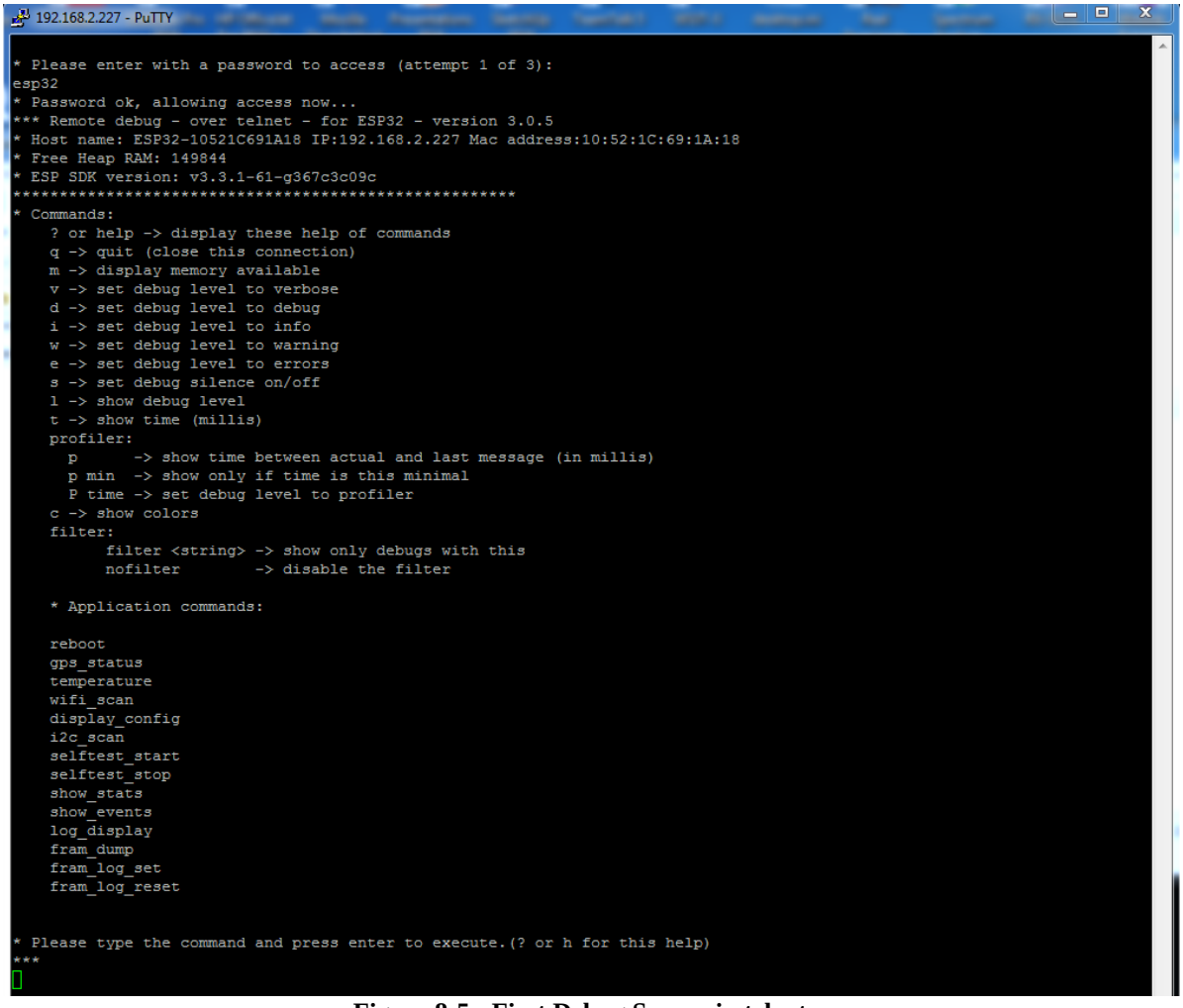

Figure 8-5 : First Debug Screen in telnet

By default, the console reports a series of diagnostic information produced by the device SW, based on the enabled debug functions.

The initial screen, after login, lists the available commands and their meaning briefly; some commands are called "Application Commands" and are used to activate specific functions of the SW.

Some of the available commands represent the exact equivalent of what can be viewed on the serial console via USB typically accessible from within the Arduino environment.

Other commands are additional functions available only through this Remote Debug interface and are described in detail below.

8.1.1 "gps_status" command

The "gps_status" command returns detailed information on the GPS status: in particular, it provides the following parameters:

- List of satellites currently used for the fix: number of satellites used, PRN, elevation, azimuth and SNR values for each satellite used for the fix
- parameters related to the fix: latitude, longitude, age of the fix, date, time, height, speed, quality parameters of the fix

An output example is in Figure 8-6.

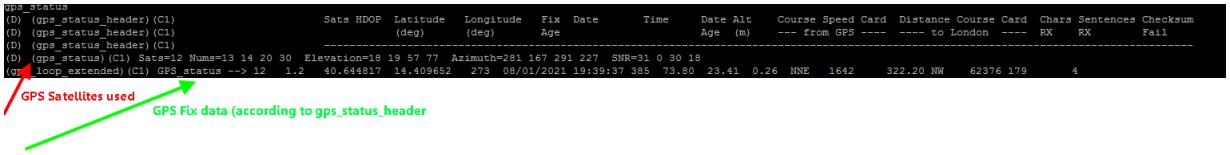

Figure 8-6 : output of "gps_status" command

8.1.2 "temperature" command

Provides an estimate of the temperature value in °C at the level of the ESP32 chip inside the device; this value should be used as a general indication of the temperature value as its determination is based on an undocumented feature of the ESP32 chip.

8.1.3 “wifi_scan” command

Scan the 2.4Ghz WiFi band and report the list of networks found and the relative signal levels and the type of encryption.

A typical output is shown in Figure 8-7.

```
wifi_scan
(do_wifi_scan)(C1) wifi_scan start...scan done: 11 networks found
1: RFC2019-24 (-55)*
2: MikroTik-66BABF-24 (-59)*
3: SARIMESH (-59)*
4: ESP32-240AC4595AA8 (-73)
5: TIM-24138604 (-83)*
6: casa Izzo (-89)*
7: Vodafone-35188680 (-90)*
8: Vodafone-WiFi (-91)
9: FASTWEB-SK5JE6 (-91)*
10: FASTWEB-VJX4Y7 (-92)*
11: WOW FI - FASTWEB (-93)*
```

Figure 8-7 : "wifi_scan" command output

8.1.4 "display_config" command

This command displays in text form the current configuration of the device in use.
The configuration consists of a list of lines of the “key = value” type.
Figure 8-8 shows only a small subset of parameters for sake of brevity.

```
display_config
(DisplayConfig)(C1) Display active Configuration data
ESP_config.DeviceName  = LoRa Tracker Vr4.1
ESP_config.DeviceId  = Not Available
ESP_config.cpu_type  = ESP32 Dev V4
ESP_config.dhcp        = 1
ESP_config.daylight    = 1
ESP_config.Update_Time = 0
ESP_config.timezone    = 10
ESP_config.IP          = 192.168.2.60
ESP_config.Netmask     = 255.255.255.0
ESP_config.GatewayIP   = 192.168.2.1
ESP_config.DnsIP       = 8.8.8.8
ESP_config.ssid        = RFC2019-24
ESP_config.password    = [redacted]
ESP_config.ntpServer   = ntp1.inrim.it
ESP_config.gps_debug  = 0
ESP_config.LoRa_debug  = 0
ESP_config.RTC_debug  = 0
ESP_config.ezTime_debug  = 0
ESP_config.pps_debug  = 0
ESP_config.PE_debug  = 0
ESP_config.BT_KISS_Mode  = 0
ESP_config.Serial_KISS_Mode  = 0
ESP_config.Tracker_Mode  = 1
```

Figure 8-8 : "display_config" command output

The nomenclature used for the keys should make it easy to identify the meaning of the listed parameters.

8.1.5 "show_stats" command

This command reports a list of statistics for the LoRa part.

In particular, it reports:
- SW version in use on the device
- Number of LoRa packets received and transmitted
- Number of LoRa packets lost (or suppressed in transmission) due to radio channel congestion
- Number of LoRa packets received with CRC value at the wrong radio level (generally due to transmission errors on the radio channel or access conflict to the radio channel)
- Waiting time for the transmission of LoRa packets

- Temperature of the ESP32 CPU
- "Uptime" time value of the CPU since the last reboot.

Figure 8-9 shows an example of this command ouput.

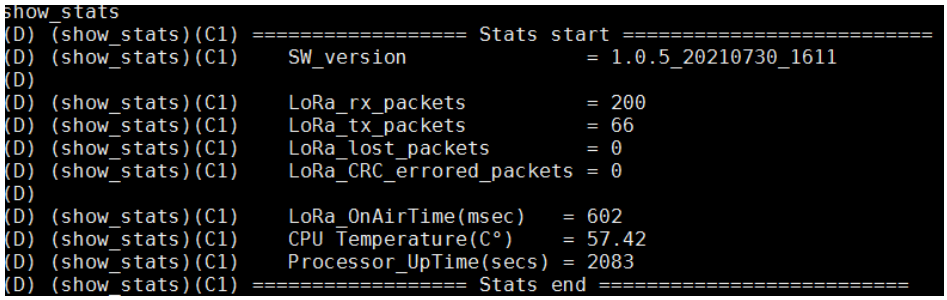

Figure 8-9 : "show_stats" command output

8.1.6 “show_events” command

The device, in the version based on HW LoRa Beacon , has an on-board FRAM memory used for keeping not only the device configuration data, but also a series of data collected in real time related to particular events and to the reception of LoRa spots.

The "show_events" command lists only the "events" part of the log kept in the FRAM; the reported events are typically linked to management actions that imply configuration changes and processor restarts.

The events are “timestamped” with the real time in which they were recorded and therefore they “survive” to device restarts and can be used for troubleshooting even after eventual crash of SW.

Figure 8-10 shows an example of output for this command.

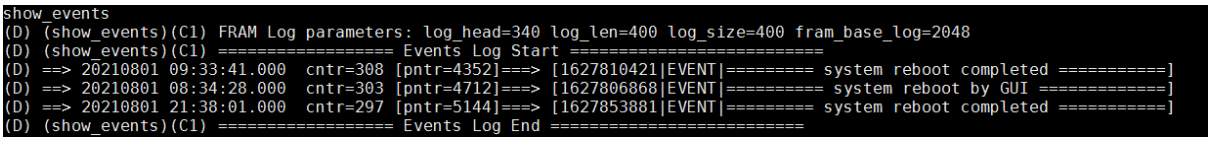

Figure 8-10 : "show_events" command output

8.1.7 “log_display” command

The "log_display" command shows the complete list of the "circular tail" which shows all the events and spots recorded in real time during device operation.

Using a “circular queue” to hold this data allows you to keep track of the last 400 recorded events or spots, discarding the oldest.

The format of the recorded "spots" is different for the case of use as an iGate or as a Tracker:

- in the first case (use as iGate) each spot shows the name of the source station, the position, the signal level and SNR of the received spot.

- in the second case (use as a tracker) the spot shows the physical position in which the tracker was at the time of receiving a spot, the frequency, the signal level and SNR of the received spot.

```
(D) ==> 20210801 20:34:19.003  cntr=16 [pntr=29336]===> [1627850059|40.6448326|14.4096670|433.725|-89.00|11.25|0.00] d=23
(D) ==> 20210801 20:34:49.003  cntr=15 [pntr=29408]===> [1627850089|40.6448326|14.4096670|433.725|-48.00|11.25|0.00] d=23
(D) ==> 20210801 20:34:50.003  cntr=14 [pntr=29480]===> [1627850090|40.6448326|14.4096670|433.725|-88.00|11.25|0.00] d=23
(D) ==> 20210801 20:35:20.003  cntr=13 [pntr=29552]===> [1627850120|40.6448326|14.4095001|433.725|-48.00|11.25|0.00] d=18
(D) ==> 20210801 20:35:21.003  cntr=12 [pntr=29624]===> [1627850121|40.6448326|14.4095001|433.725|-87.00|11.00|0.00] d=18
(D) ==> 20210801 20:35:52.003  cntr=11 [pntr=29696]===> [1627850152|40.6448326|14.4096670|433.725|-48.00|11.00|0.00] d=23
(D) ==> 20210801 20:35:52.003  cntr=10 [pntr=29768]===> [1627850152|40.6448326|14.4096670|433.725|-87.00|11.25|0.00] d=23
(D) ==> 20210801 20:36:23.003  cntr=9 [pntr=29840]===> [1627850183|40.6448326|14.4096670|433.725|-48.00|11.25|0.00] d=23
(D) ==> 20210801 20:36:23.003  cntr=8 [pntr=29912]===> [1627850183|40.6448326|14.4096670|433.725|-88.00|11.25|0.00] d=23
(D) ==> 20210801 20:36:54.003  cntr=7 [pntr=29984]===> [1627850214|40.6448326|14.4096670|433.725|-50.00|11.25|0.00] d=23
(D) ==> 20210801 20:36:54.003  cntr=6 [pntr=30056]===> [1627850214|40.6448326|14.4096670|433.725|-90.00|10.75|0.00] d=23
(D) ==> 20210801 20:37:21.003  cntr=5 [pntr=30128]===> [1627850241|40.6448326|14.4096670|433.725|-48.00|11.25|0.00] d=23
(D) ==> 20210801 20:37:22.003  cntr=4 [pntr=30200]===> [1627850242|40.6448326|14.4096670|433.725|-88.00|10.75|0.00] d=23
(D) ==> 20210801 20:37:25.003  cntr=3 [pntr=30272]===> [1627850245|40.6448326|14.4096670|433.725|-48.00|11.50|0.00] d=23
(D) ==> 20210801 20:37:26.003  cntr=2 [pntr=30344]===> [1627850246|40.6448326|14.4096670|433.725|-87.00|11.00|0.00] d=23
(D) ==> 20210801 20:37:41.003  cntr=1 [pntr=30416]===> [1627850261|40.6448326|14.4096670|433.725|-87.00|11.25|0.00] d=23
(D) (log_display)(C1) ================== Log End ========================
```

Figure 8-11 : oldest data of "log_display" output

```
(D) (log_display)(C1) FRAM Log parameters: log_head=395 log_len=400 log_size=400 fram_base_log=2048
(D) (log_display)(C1) ================== Log Start ========================
(D) ==> 19700101 00:15:10.093  cntr=399 [pntr=30560]===> [910|40.6435013|14.4095001|433.725|-88.00|10.75|0.00] d=3341105
(D) ==> 19700101 00:15:29.093  cntr=398 [pntr=30632]===> [929|40.6435013|14.4095001|433.725|-102.00|4.75|0.00] d=3341105
(D) ==> 19700101 00:15:30.093  cntr=397 [pntr=30704]===> [930|40.6435013|14.4095001|433.725|-41.00|11.50|0.00] d=3341105
(D) ==> 19700101 00:15:33.093  cntr=396 [pntr=30776]===> [933|40.6435013|14.4095001|433.725|-101.00|3.25|0.00] d=3341105
(D) ==> 19700101 00:15:41.093  cntr=395 [pntr=2048]===> [941|40.6445007|14.4090004|433.725|-96.00|8.00|0.00] d=3882434
(D) ==> 19700101 00:15:42.093  cntr=394 [pntr=2120]===> [942|40.6445007|14.4090004|433.725|-32.00|12.00|0.00] d=3882434
(D) ==> 19700101 00:15:43.093  cntr=393 [pntr=2192]===> [943|40.6445007|14.4090004|433.725|-96.00|8.50|0.00] d=3882434
(D) ==> 19700101 00:16:06.093  cntr=392 [pntr=2264]===> [966|40.6445007|14.4090004|433.725|-86.00|11.25|0.00] d=3882434
(D) ==> 19700101 00:16:11.093  cntr=391 [pntr=2336]===> [971|40.6446686|14.4095001|433.725|-39.00|10.50|0.00] d=8415512
(D) ==> 19700101 00:16:12.093  cntr=390 [pntr=2408]===> [972|40.6446686|14.4095001|433.725|-79.00|12.00|0.00] d=8415512
(D) ==> 19700101 00:16:13.093  cntr=389 [pntr=2480]===> [973|40.6446686|14.4095001|433.725|-42.00|12.00|0.00] d=8415512
(D) ==> 19700101 00:00:30.093  cntr=388 [pntr=2552]===> [30|EVENT|========== system reboot completed ===========] d=4749550
(D) ==> 19700101 00:00:39.093  cntr=387 [pntr=2624]===> [39|0.0000000|0.0000000|433.725|-49.00|11.75|0.00] d=4749550
(D) ==> 19700101 00:00:39.093  cntr=386 [pntr=2696]===> [39|0.0000000|0.0000000|433.725|-86.00|11.75|0.00] d=4749550
(D) ==> 19700101 00:00:44.093  cntr=385 [pntr=2768]===> [44|0.0000000|0.0000000|433.725|-88.00|11.25|0.00] d=4749550
(D) ==> 19700101 00:01:10.093  cntr=384 [pntr=2840]===> [70|40.6448326|14.4096670|433.725|-49.00|11.25|0.00] d=5323454
(D) ==> 19700101 00:01:10.093  cntr=383 [pntr=2912]===> [70|40.6448326|14.4096670|433.725|-87.00|11.75|0.00] d=5323454
(D) ==> 19700101 00:01:41.093  cntr=382 [pntr=2984]===> [101|40.6448326|14.4096670|433.725|-47.00|11.25|0.00] d=10600898
(D) ==> 19700101 00:01:41.093  cntr=381 [pntr=3056]===> [101|40.6448326|14.4096670|433.725|-88.00|11.25|0.00] d=10600898
(D) ==> 19700101 00:01:43.093  cntr=380 [pntr=3128]===> [103|40.6448326|14.4096670|433.725|-49.00|12.00|0.00] d=10600898
```

Figure 8-12 : newest data of "log_display" command output

Figure 8-11 and Figure 8-12 show a log example for an iGate, while in Figure 8-13 in case of a tracker.

The reason for the different information content derives from the impossibility, in the tracker case, to identify the source of the LoRa message.

Each spot also contains a field d (distance) whose meaning is left to a future documentation phase; currently it is to be considered a non-significant field.

```
(D) ==> 20210731 16:59:30.690  cntr=32 [pntr=9680]===> [1627750770|41.2881660|13.2609997|IZ0CZW-9|-118.00|-8.00|0.00] d=121756
(D) ==> 20210731 17:00:01.690  cntr=31 [pntr=9752]===> [1627750801|41.2905006|13.2644997|IZ0CZW-9|-114.50|-6.50|0.00] d=121684
(D) ==> 20210731 17:01:03.690  cntr=30 [pntr=9824]===> [1627750863|40.6445007|14.4091663|I8FUC-10|-62.00|11.50|0.00] d=3682
(D) ==> 20210731 17:01:32.690  cntr=29 [pntr=9896]===> [1627750892|40.6119995|14.4008331|IQ8SO-10|-62.00|11.75|0.00] d=0
(D) ==> 20210731 17:03:03.690  cntr=28 [pntr=9968]===> [1627750983|40.6445007|14.4091663|I8FUC-10|-62.00|11.75|0.00] d=3682
(D) ==> 20210731 17:03:08.690  cntr=27 [pntr=10040]===> [1627750988|41.3068352|13.2873335|IZ0CZW-9|-110.00|-4.00|0.00] d=121325
(D) ==> 20210731 17:05:12.690  cntr=26 [pntr=10112]===> [1627751112|41.3193321|13.3013334|IZ0CZW-9|-98.00|3.25|0.00] d=121311
(D) ==> 20210731 17:05:43.690  cntr=25 [pntr=10184]===> [1627751143|41.3224983|13.3056669|IZ0CZW-9|-101.00|1.50|0.00] d=121262
(D) ==> 20210731 17:06:14.690  cntr=24 [pntr=10256]===> [1627751174|41.3268318|13.3078337|IZ0CZW-9|-105.75|-1.75|0.00] d=121437
(D) ==> 20210731 17:06:34.690  cntr=23 [pntr=10328]===> [1627751194|40.6119995|14.4008331|IQ8SO-10|-62.00|11.75|0.00] d=0
(D) ==> 20210731 17:07:05.690  cntr=22 [pntr=10400]===> [1627751225|40.6445007|14.4091663|I8FUC-10|-62.00|11.75|0.00] d=3682
(D) ==> 20210731 17:09:06.690  cntr=21 [pntr=10472]===> [1627751346|40.6445007|14.4091663|I8FUC-10|-62.00|11.75|0.00] d=3682
(D) ==> 20210731 17:11:07.690  cntr=20 [pntr=10544]===> [1627751467|40.6445007|14.4091663|I8FUC-10|-62.00|11.50|0.00] d=3682
(D) ==> 20210731 17:11:35.690  cntr=19 [pntr=10616]===> [1627751495|40.6119995|14.4008331|IQ8SO-10|-64.00|11.00|0.00] d=0
(D) ==> 20210731 17:13:09.690  cntr=18 [pntr=10688]===> [1627751589|40.6445007|14.4091663|I8FUC-10|-62.00|11.50|0.00] d=3682
(D) ==> 20210731 17:15:10.690  cntr=17 [pntr=10760]===> [1627751710|40.6445007|14.4091663|I8FUC-10|-62.00|11.50|0.00] d=3682
(D) ==> 20210731 17:16:36.690  cntr=16 [pntr=10832]===> [1627751796|40.6119995|14.4008331|IQ8SO-10|-62.00|12.00|0.00] d=0
(D) ==> 20210731 17:19:12.690  cntr=15 [pntr=10904]===> [1627751952|40.6445007|14.4091663|I8FUC-10|-62.00|11.75|0.00] d=3682
(D) ==> 20210731 17:21:13.690  cntr=14 [pntr=10976]===> [1627752073|40.6445007|14.4091663|I8FUC-10|-62.00|12.00|0.00] d=3682
(D) ==> 20210731 17:21:17.690  cntr=13 [pntr=11048]===> [1627752077|41.3328323|13.3161669|IZ0CZW-9|-121.50|-9.50|0.00] d=121345
```

Figure 8-13 : example of log in case of a tracker

8.2 Access to Remote Debug IF via Web App

To use this access method, a particular application called Remote Debug WEB App must be installed on your PC and can be downloaded from the following URL: https://github.com/JoaoLopesF/RemoteDebugApp#installing

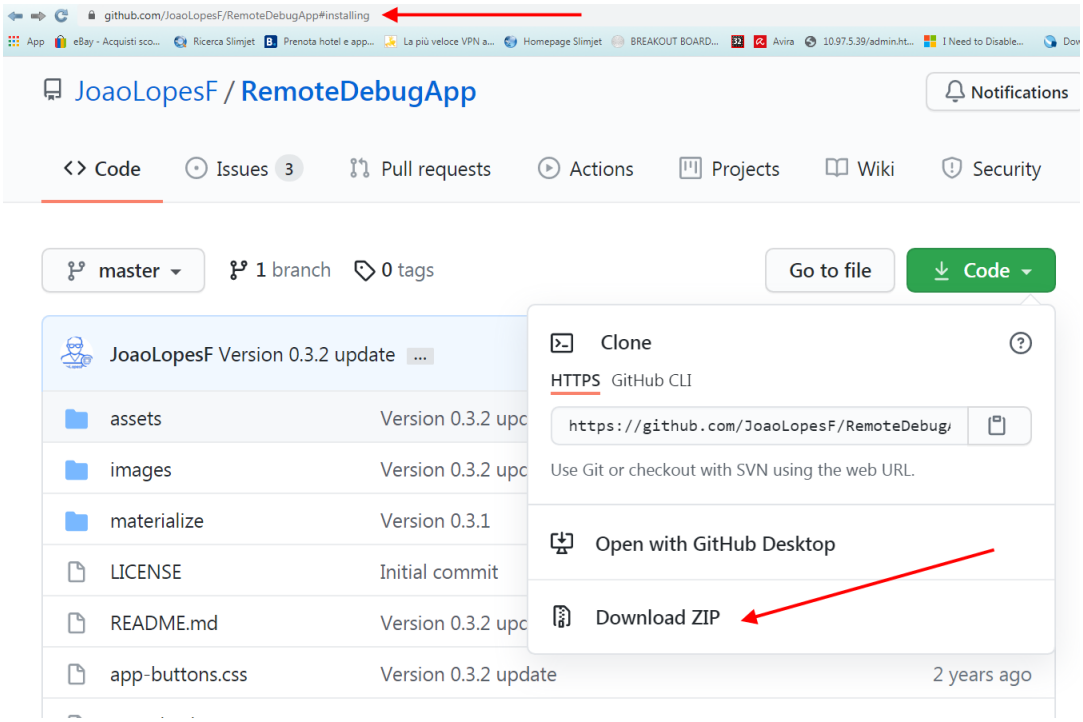

Figure 8-14 : Remote Debug Web app download page

Figure 8-14 shows the download page; to download the App it is necessary to select the "Code" box and from the subsequent selection window that appears, choose the "Download" option.

The .zip archive thus obtained must be expanded on the PC you are using in a directory chosen by the user; then with the local "explorer" tool, select the index.html file in the directory where the archive was expanded and open it with a browser such as chrome or firefox.

Figure 8-15 shows the appearance of the resulting chrome browser window.

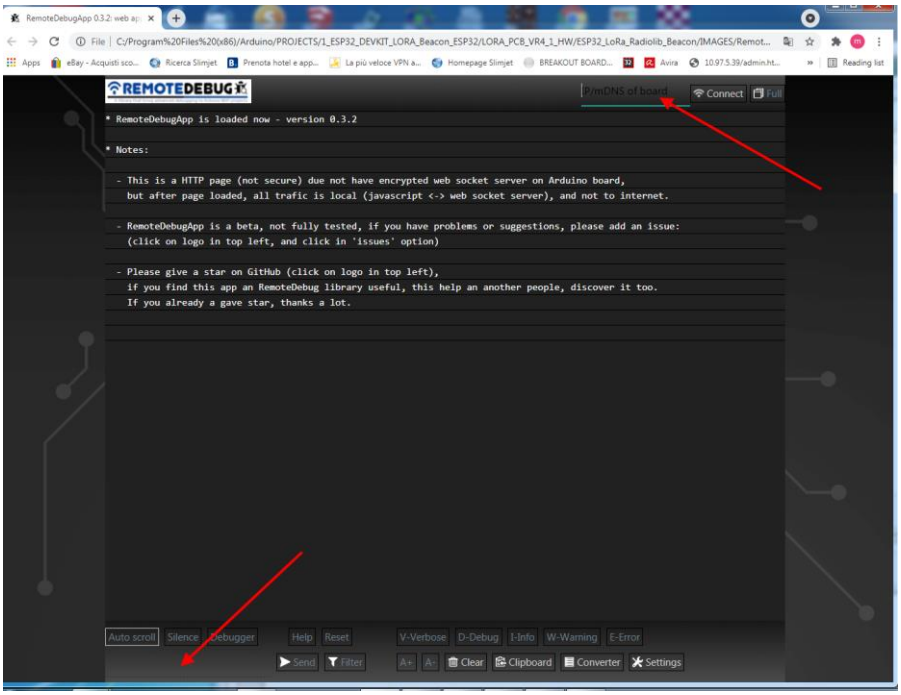

Figure 8-15 : Remote Debug first screen appearance

As you can see, it is a classic web interface that initially requires entering the IP address of the device to be monitored.

It is then necessary to enter the access password (by default esp32) in the input window.

After entering the password, you will find a window similar to the one obtained using the putty application described in the previous paragraph.

At this point, all the functions and considerations made in the previous paragraph apply exactly.

Using this interface, some additional functions are available, among which perhaps the most interesting is the “Content Filter", used to display only particular messages; this function is therefore very useful in case of different debug functions enabled on the device.

An example of filter application is shown in Figure 8-17.

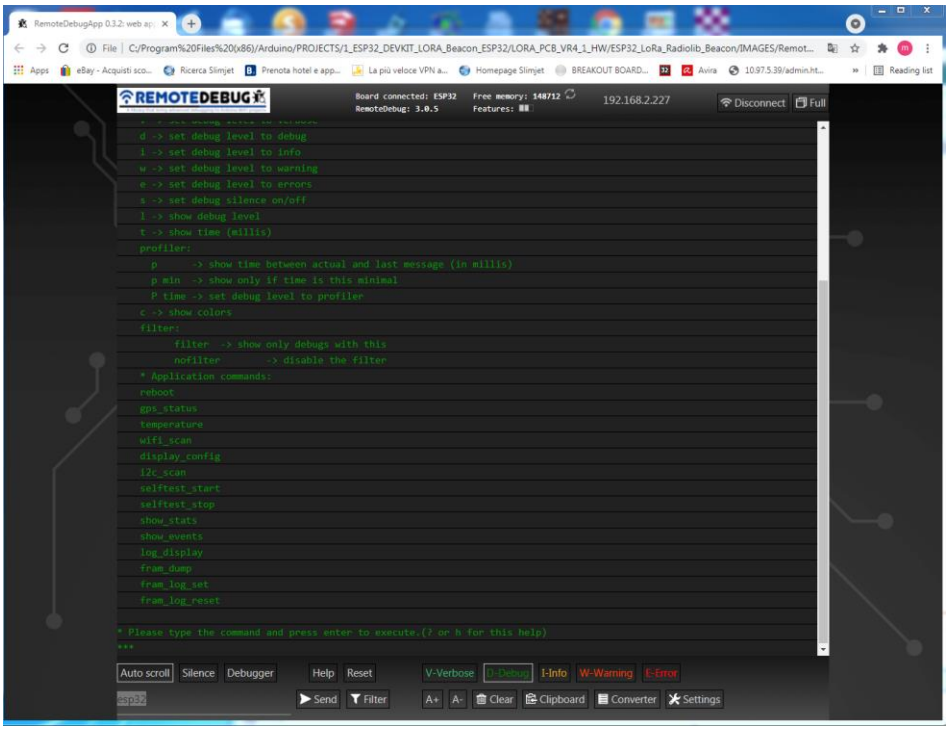

Figure 8-16 : Initial Remote debug screen

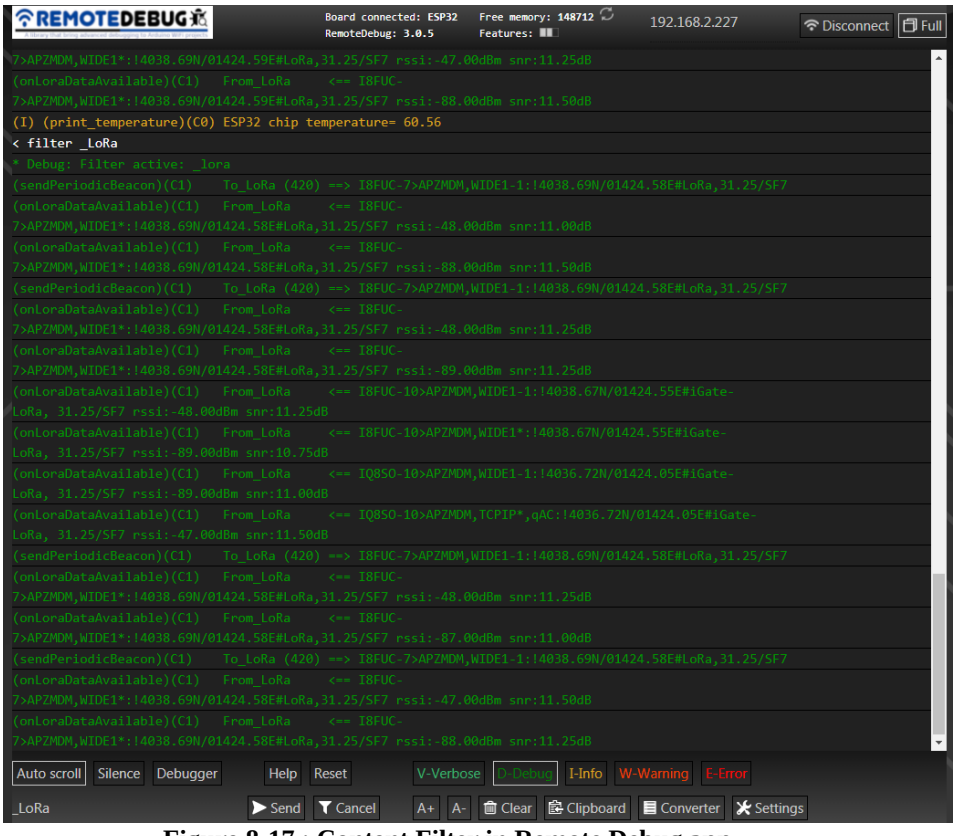

Figure 8-17 : Content Filter in Remote Debug app

9 Porting of the LoRa Beacon SW to other HW platforms

The SW of the LoRa Beacon project has been designed to be used not only on the LoRa Beacon HW platform described above, but also on similar HW devices which use the same type of microcontroller (ESP32) and which include a set of similar HW functions , such as e.g. a GPS module and/or a LoRa module.

At present, the SW has been ported to two particular cards which represent significant examples of "all-in-one" devices available on the usual Chinese portals.

The "porting" operation consists of the following operations:

- properly configure the Arduino environment or PlatformIO for SW development by appropriately selecting the HW platform to support
- analyze the HW structure of the device and appropriately set the SW configuration file "master_config.h".
- carry out a "build" of the SW by connecting directly to the target via USB
- carry out the necessary "non-regression" tests to verify the correct functioning of the HW-SW set, only for the functions that can be supported by the HW in use
- verify that the device functionally behaves as expected.

Once the porting work has been successfully completed, it is possible to generate a complete image of the SW that can be loaded onto the card using the classic ESP32 microcontroller downlad tool, without using the Arduino or PlatformIO development environment.

Building images is very useful for those who intend to use simply the SW without necessarily making changes to it, but using the configuration interface provided for setting up the device and for carrying out any experiments with it.

The following paragraphs show the settings to be used for the download tool in relation to the different types of boards to be supported.

9.1 SW installation on TTGO T-beam-V1-2019 device

This card is the first version of a long series of devices made by the same Vendor.

It’s worth to be noted that although these devices share the same commercial name “T-Beam”, they may have a HW equipment that is also quite different from the original version. Doing a search on the internet it is possible to learn more about this topic (https://github.com/Xinyuan-LilyGo/TTGO-LoRa-Series).

The following figure shows the details of the device used for the test.

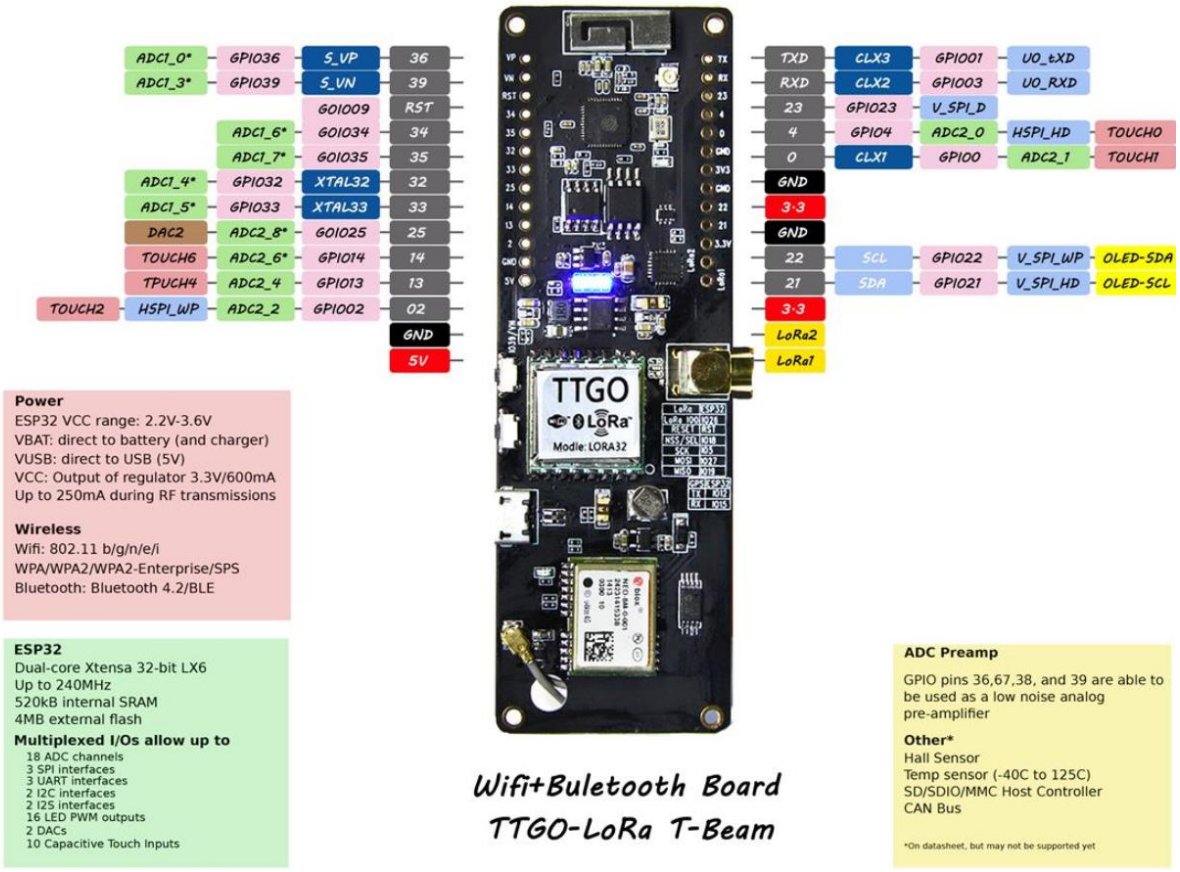

Figure 9-1 : TTGO T-Beam V1 board used for the tests

The following procedure obviously refers to the specific device indicated for obvious reasons of availability of the HW on which to carry out the test.

This board is not able to support the FRAM memory present in the LoRa Beacon project, therefore all the SW functions that require this component are not supported, with the exception of the device configuration part only.

Instead of the FRAMs, a portion of the Flash memory is used to store the device configuration; from a functional point of view there are only minor differences compared to the case of using the FRAM.

As first consequence, the permanent logs and therefore the reconfiguration/reboot events and the log of the received spots are not available on this board.

This card supports only (in the tested version) first generation (sx127x) LoRa chips, therefore with a maximum output power of about 100mWatt, with sensitivity of the radio receiving part not improved and without TCXO support for the stabilization of the emitted frequency.

As for the display, the only supported display is the 0.96” monochrome one with I2C interface.

Figure 9-2 illustrates the connections of the display to the circuit board.

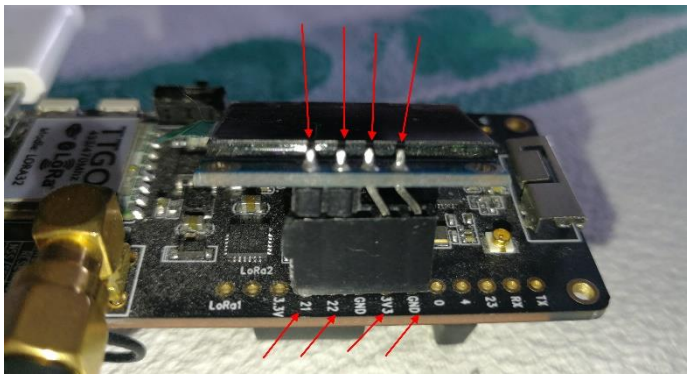

Figure 9-2 : TTGO T-Beam V1 display connections

The functions of the red and green LEDs are not supported but it is possible to map them to the blue LED present on the devices.

For the SW installation using the download tool presented in the previous chapters, exactly the same considerations apply with the only exception of the serial port number, which can be different, and the hexadecimal addresses for loading modules.

The figure shows the settings to use:

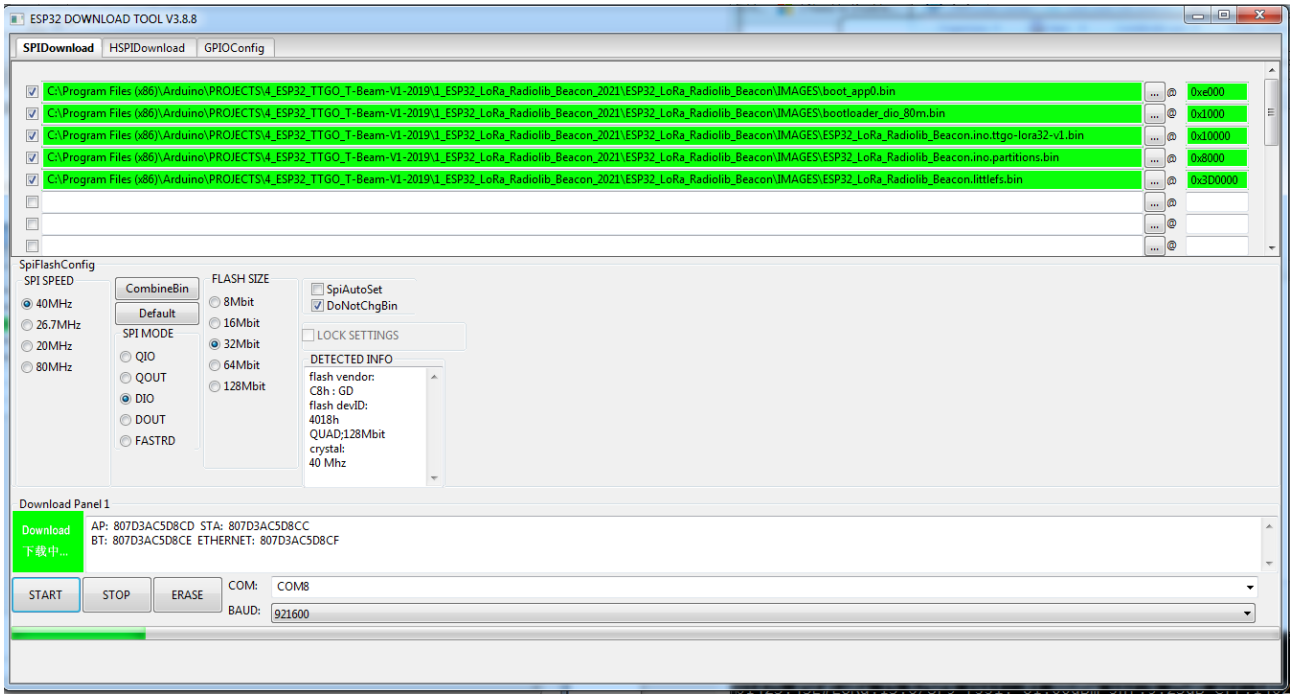

Figure 9-3 : settings to be used for TTGO T-Beam V1 SW upload

The SW image is available on github as .zip archive which contains the 5 SW modules to be loaded onto the device.

Figure 9-4 is an example of the display content.

Figure 9-4 : Example of display content for TTGO T-Beam

Note the presence of the " **ERR** " field which reports the frequency error between RT/TX due to the lack of TCXO on this type of HW; this offset can be corrected using the "ppm" field in the setup screen of the LoRa section.

This loading procedure is only required for the first loading of the SW on the card; for subsequent SW updates it is possible to use the OTA update interface available on the GUI.

9.2 SW installation on Heltec_wifi_lora32 device

This card is very similar to the device described in the previous paragraph, but lacks the GPS module; a possible use is as an iGate as it is not essential to have GPS functionality in this application.

For the rest, it shares almost all the limitations mentioned in the previous paragraph.

Figure 9-5 shows the card we're talking about; there is a very similar card available under the commercial name TTGO WiFi Lora 32 which is practically identical to the one shown below.

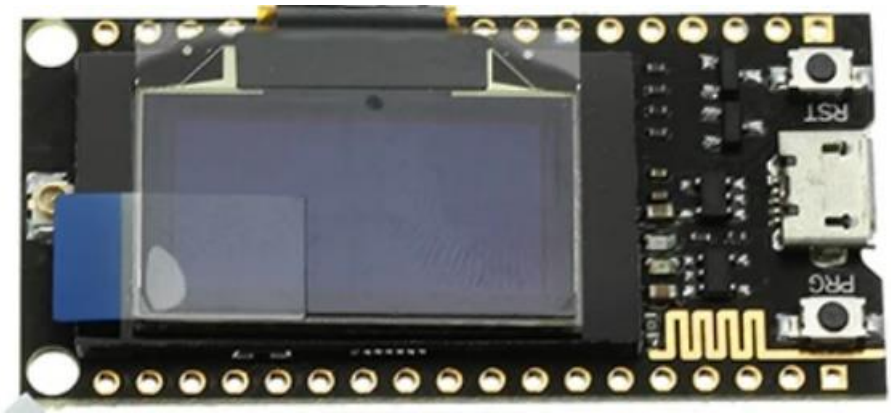

Figure 9-5 : Heltec_wifi_lora32 board

The following figure represents the pinout of the device used for the test.

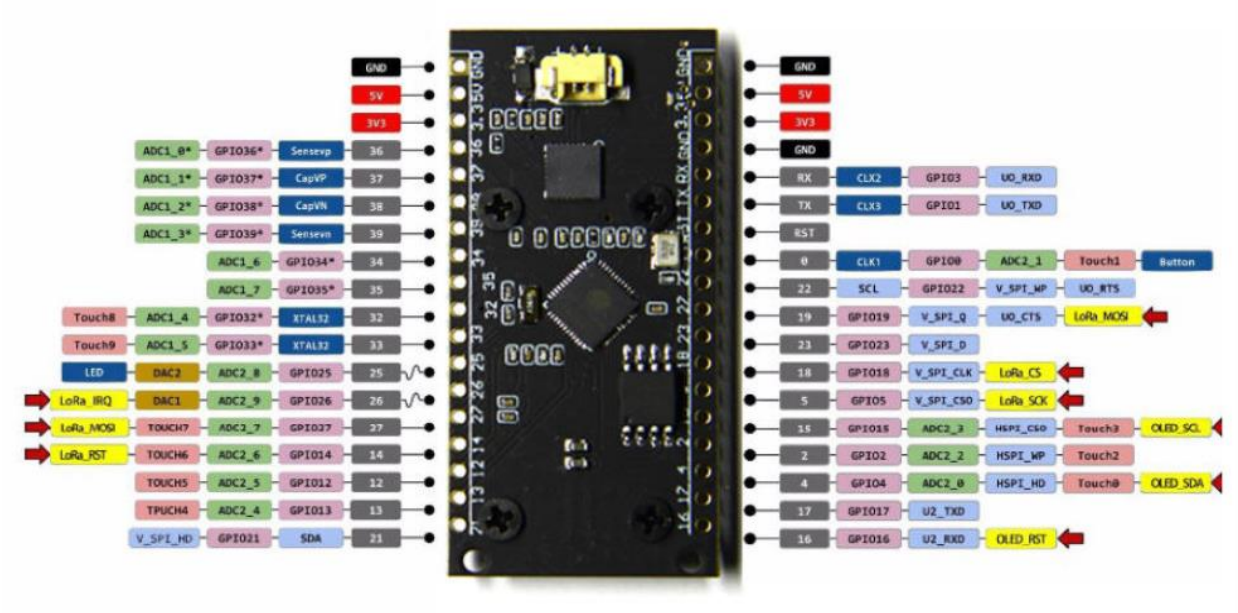

Figure 9-6 : Pin Layout for Heltec WiFi LoRa 32

Figure 9-7 shows the settings to be used for the Sw loading tool:

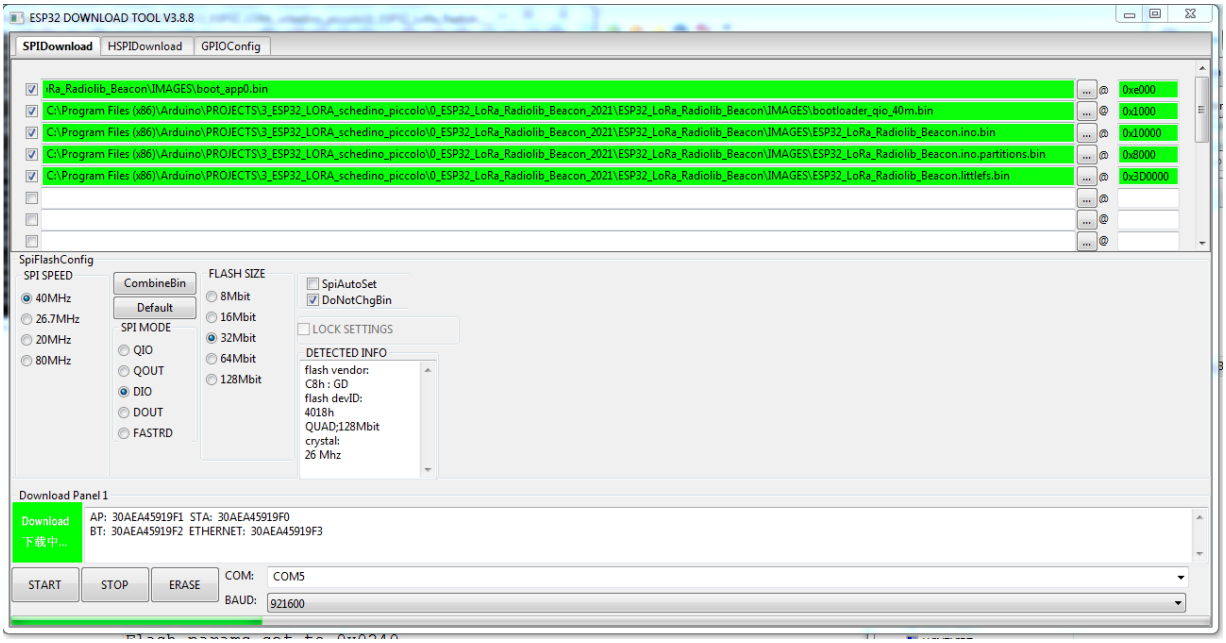

Figure 9-7 : SW upload Settings for Heltec WiFi LoRa 32

Figure 9-8 shows the general properties screen of this version.

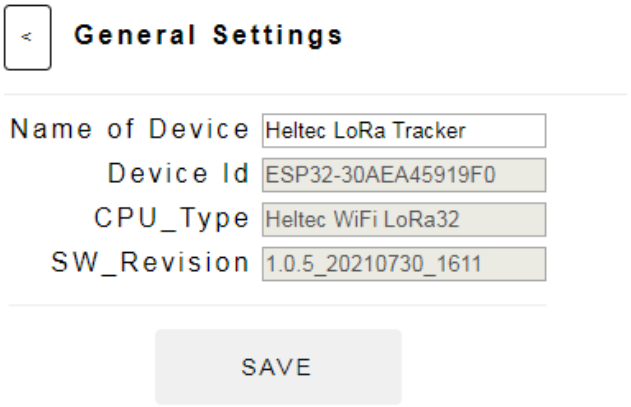

Figure 9-8 : "General Settings" page for Heltec WiFi LoRa 32

10 SW installation via OTA (Over-The-Air)

The LoRa Beacon SW provides a way to update the SW without requiring any physical connection to the target device.

This mode, called OTA (Over-The-Air), takes advantage of the WiFi Station functionality which will be active on the device once the Sarimesh SW is loaded.

The availability of this function allows, moreover, to update the SW of a remote device without needing to be on-site.

The SW update does not impact the configuration data which are therefore preserved between a SW update. However, it remains to be checked after each update.

Therefore, in order to be able to update the SW, it is necessary to have a single SW module to download onto a computer equipped with a browser of the chrome or firefox family.

The computer to be used for the SW update must be able to reach the device to be updated via WiFi (and possibly via an internet connection): therefore the device to be updated must in turn be connected to an access point from which It is possible to reach (also via internet) the PC that you intend to use for the SW update.

Actually, the OTA is not supported if the device is in “stand-alone” (wifi AP) mode.

Therefore the first step to be able to carry out the SW update is to verify the connectivity between the PC and the device to be updated, using for example the the classic “ping” tool available on any PC.

Once the connectivity has been verified, it is necessary to place the device to be updated in a particular operating condition called "Admin Mode" which is required in order to temporarily disable some unnecessary functions in the SW update phase.

Note: the LoRa and GPS stages will not work while in Admin Mode!

For this purpose it is possible to access the “OPERATION MODE SETTINGS” screen and select the Admin_Mode box, thus saving the page.

The remote device will now reboot and the display will show “Admin_Mode”.

It should be noted that the automatic reboot is only available using devices based on the LoRa Beacon HW platform; in case of different platforms it will be necessary to manually restart the remote device (and therefore a person near the device or some other mechanism will be needed to restart the device remotely eg by clicking the power).

In these conditions, the device will appear to be temporarily non-functional at the LoRa Radio level as this functionality will, as explained, be temporarily unavailable.

The activation of the Admin_Mode will bring up a new graphic interface dedicated to the SW update: to access this interface it will be necessary to point with another browser window to the address of the remote device, using a port number equal to 88 as in the example of Figure 10-1.

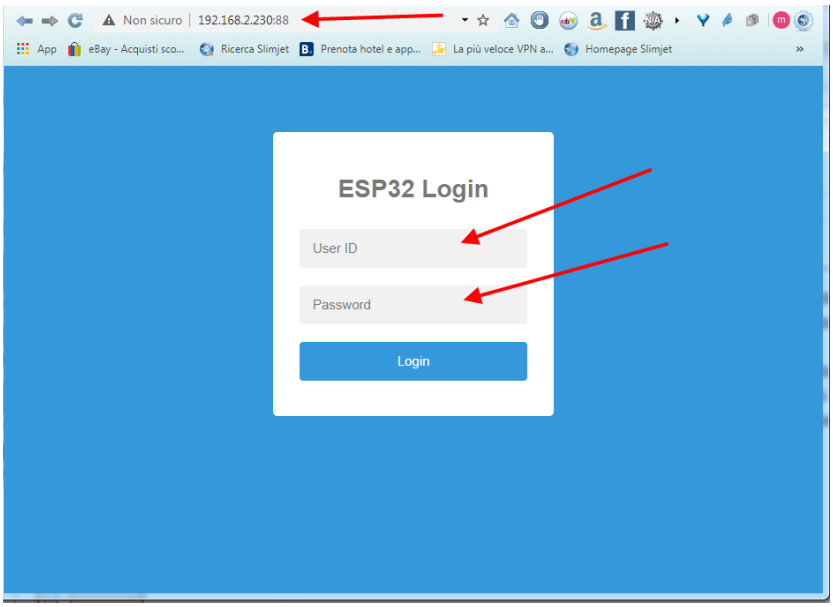

Figure 10-1 : OTA web page

At this point the following values will be entered as credentials (by default):

- User ID: admin
- Password: adminota

A new screen will appear asking you to select the new SW update image from your PC: select the file received as an update, whose name will typically end in .bin.

An example is reported in Figure 10-2.

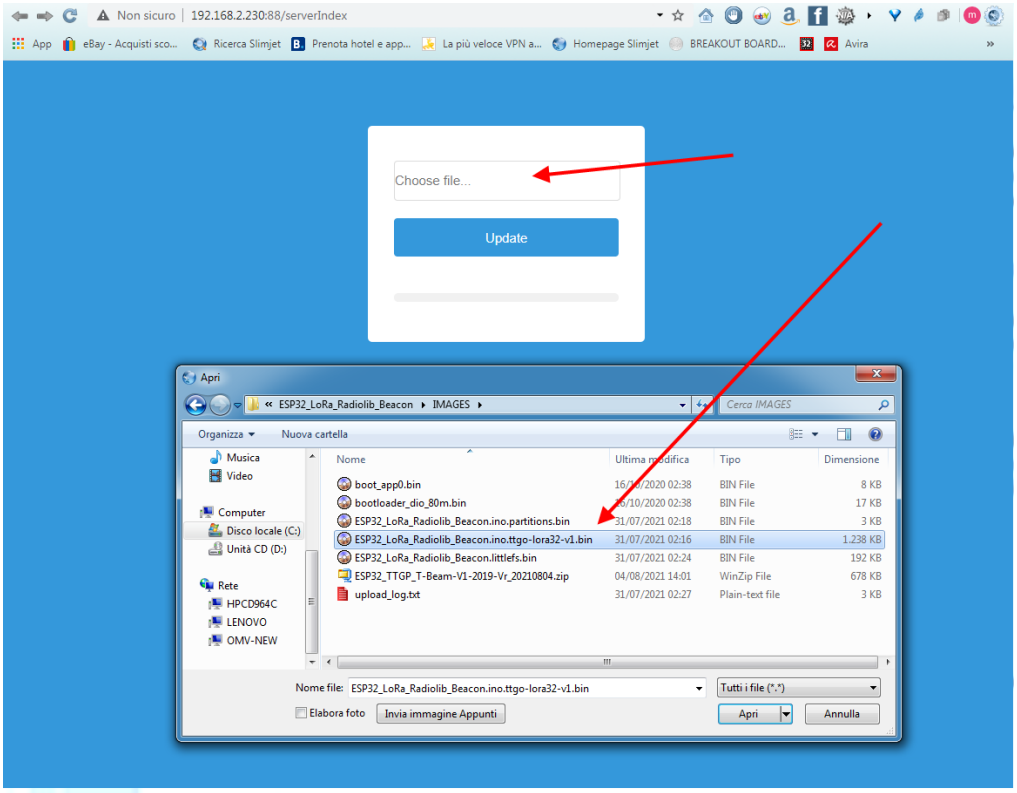

Figure 10-2 : Selection of SW image to be loaded

Then start loading the SW package and wait for the operation to complete, as reported in Figure 10-3

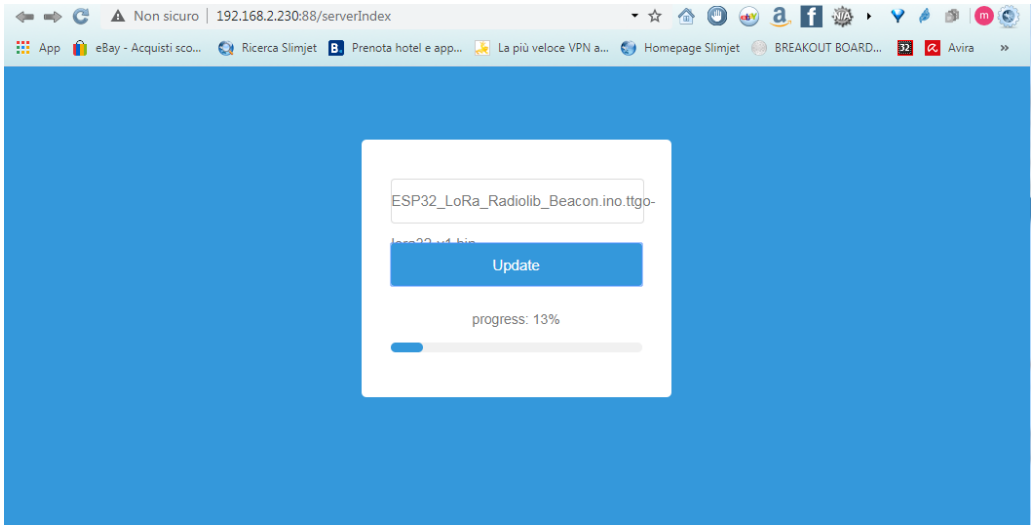

Figure 10-3 : Progress bar during SW upload

Once the new update image has been loaded, the remote device will reboot and reappear with its graphical interface, according to the new loaded SW; by accessing the "General Configuration" page, it will be possible to verify that the SW version of the device is the one expected as a result of the loading performed.

At this point the device is updated so it is possible to exit the "Admin_Mode" via the "Operation_Mode_settings" page; the device will perform a new reboot and will appear in the selected operating mode.

After each SW update, it is a good idea to check that the configuration of the device is correct as, depending on the changes introduced in the new SW version, minor adjustments to the configuration may be necessary; such adjustments should possibly be documented in the "Release Note" which should accompany each new SW release.

11 Saving and Loading of the configuration via OTA

The SW LoRa Beacon has been designed to be easily used for experimenting the LoRa radio protocol and not to be a pure SW development exercise, it is aimed as an instrument for the purpose of evaluation the LoRa technology in non-standard uses (according to the objectives for which it was born), or rather for typical use in the amateur radio world.

So the goal was to provide the ability to easily change a series of operating parameters without necessarily having to have SW development experience.

The method chosen was therefore to parameterize all the functional elements and make them modifiable through a graphical interface; the set of parameters that characterize a given setting of each device is indicated hereafter with the term "configuration" and can be viewed as a set of attributes and corresponding values.

The configuration of a device is kept in a non-volatile memory which can be a FRAM, in case of LoRa Beacon HW, or a fraction of the flash memory for those devices without FRAM.

The format chosen for the configuration file is the JSON standard which is easily readable and manageable both with a simple editor and with one of the many existing tools for this purpose. These tools can also easily compare multiple configuration files to highlight any differences between them.

The ability to save and reload a certain configuration allows you to easily keep track of the test conditions in which a tests has been performed.

All the configuration save and restore operations require that the device be put in "Admin_Mode" as for the SW update operations.

To access the configuration save/restore functions, the “ **SAVE/RESTORE CONFIGURATION** ” page is available.

Figure 11-1 shows the content of the “Save/Restore” screen: as you can see, there is a list of config files that can be loaded manually. The File /WebConf.conf always represents the current configuration file.

Figure 11-1 : "Save/Restore" GUI page

To save the current configuration of the device, simply select the “Save” button: a file will be created and downloaded directly to the PC, as shown in the following figure:

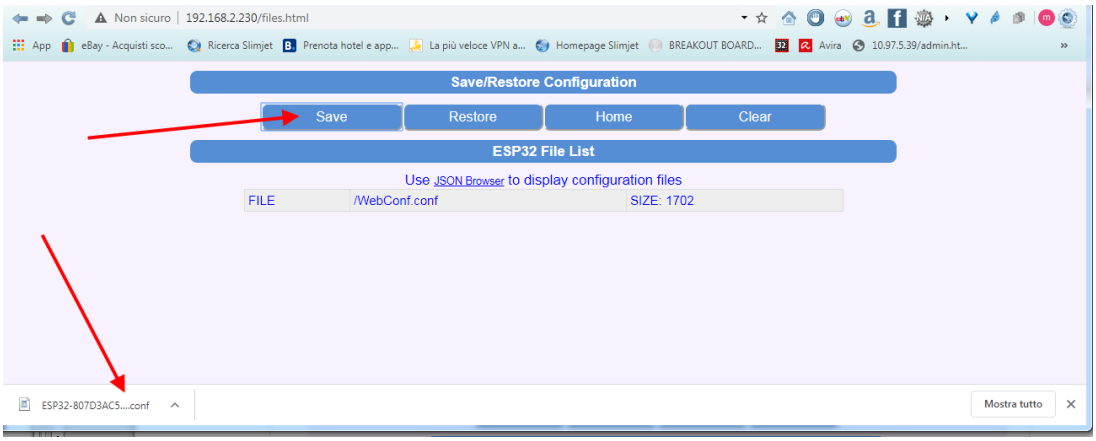

Figure 11-2 : Saving of the device configuration into a PC

The downloaded file will have a name like “ESP32-<MAC address>.conf” and will be saved in a location depending on the browser in use.

To restore the configuration from a file, simply select the "Restore" button and select the file you wish to load, then select the “Upload File” button, as depicted in Figure 11-3.

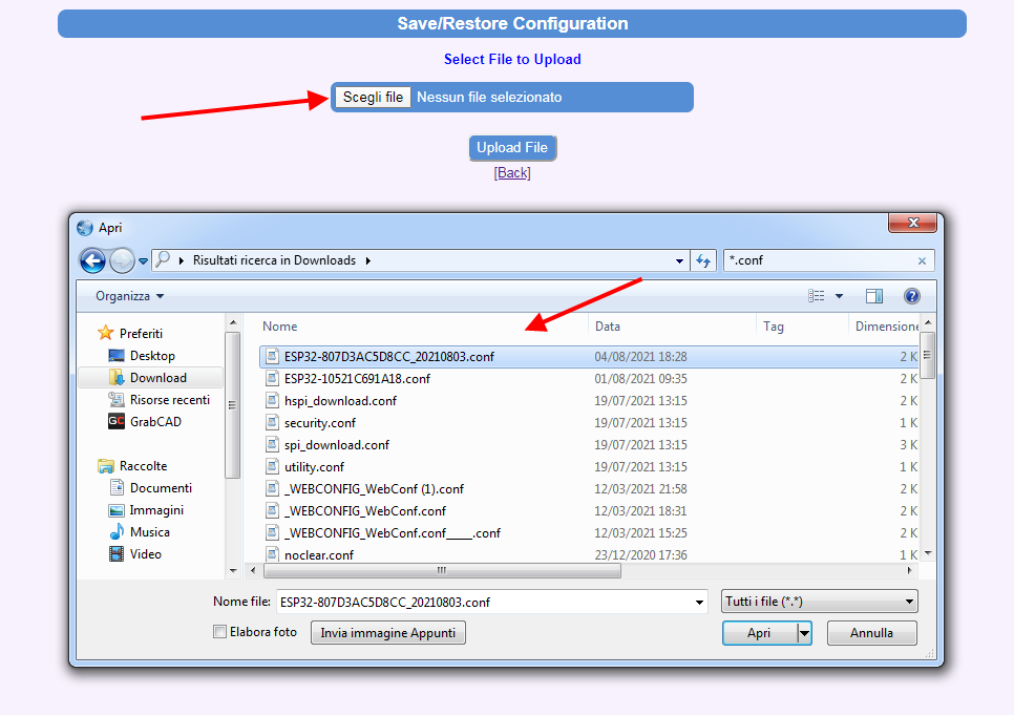

Figure 11-3 : Restoring configuration from a file

If the loading is successfully completed, you will see a screen like Figure 11-4:

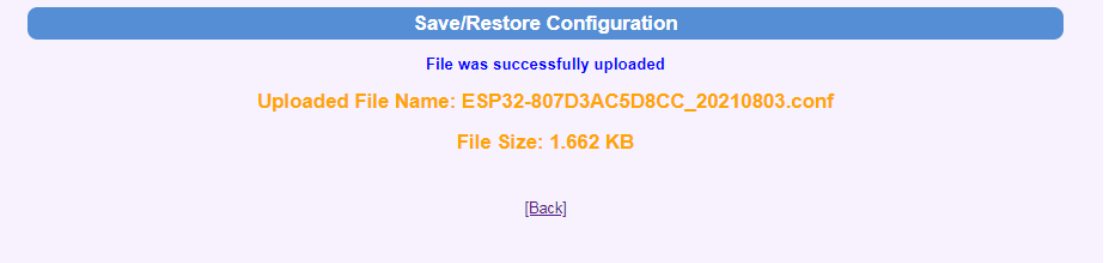

Figure 11-4 : Successful completion of config restore

By selecting the "Back" key, the following screen will appear which highlights the presence of the loaded file among the files present on the device.

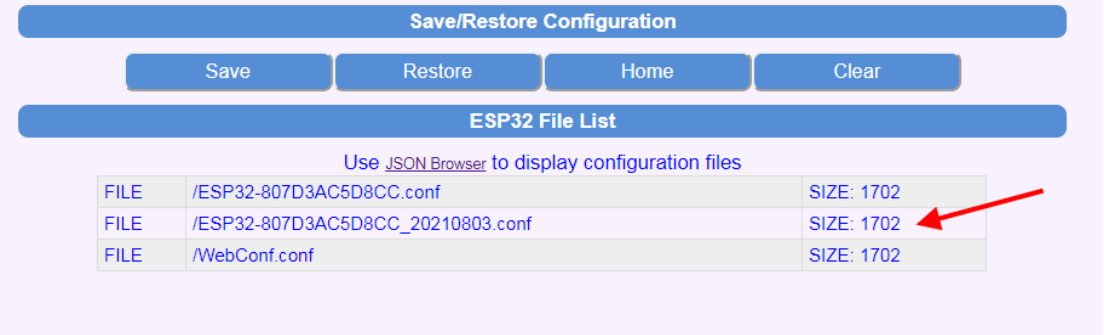

Figure 11-5 : Configuration file list after restore

To make the new configuration effective, it is necessary to exit the "admin_Mode" mode and restart the device.

If you want to inspect and/or possibly modify the configuration file, you can use, for example, the tool https://jsonformatter.org/json-pretty-print , which isfreely accessible on the internet via any browser such as chrome or firefox.

In Figure 11-6, all the step for opening a config file are shown.

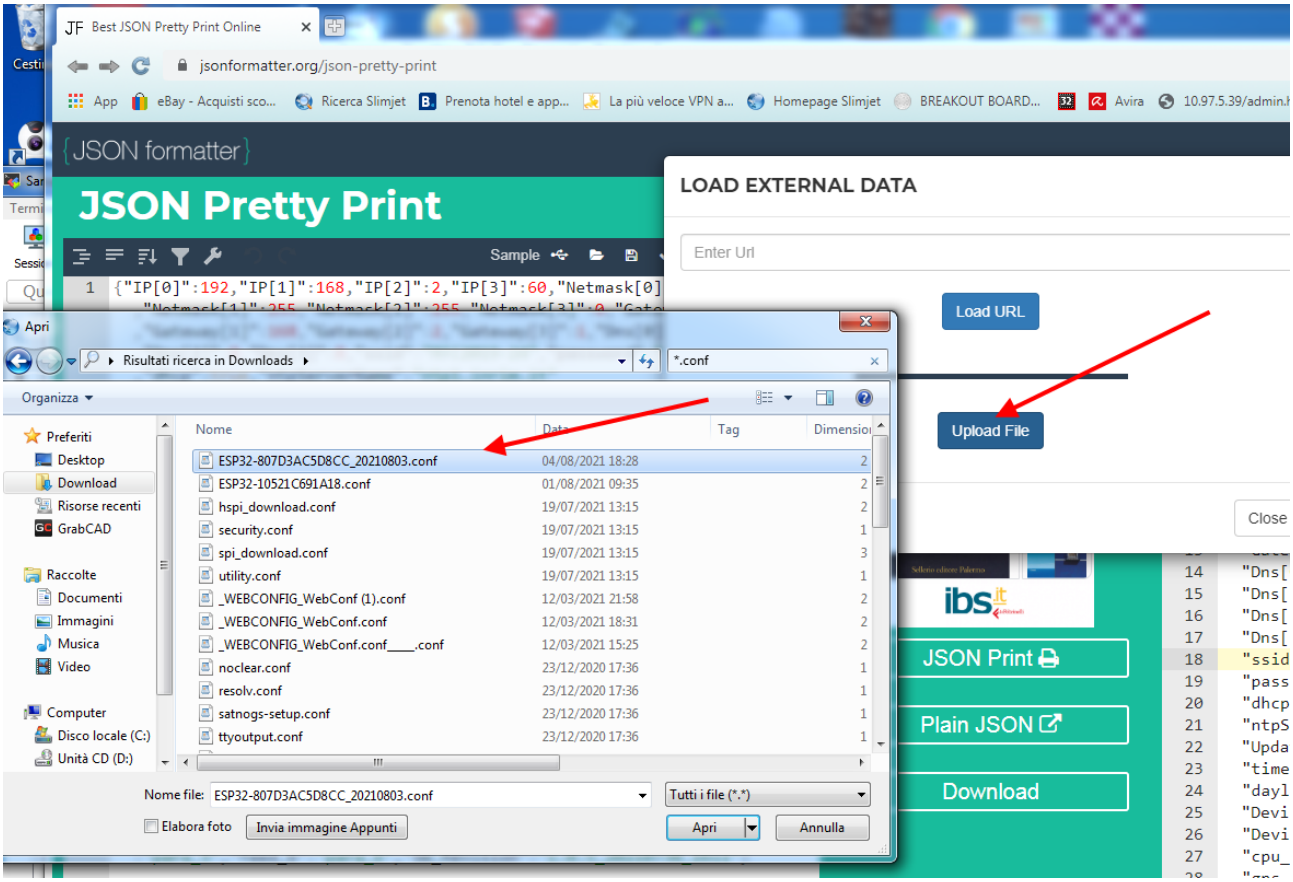

Figure 11-6 : Opening config file in JSON Pretty Print

The following figure is an example this tool while is viewing the contents of a configuration file in “plain text”:

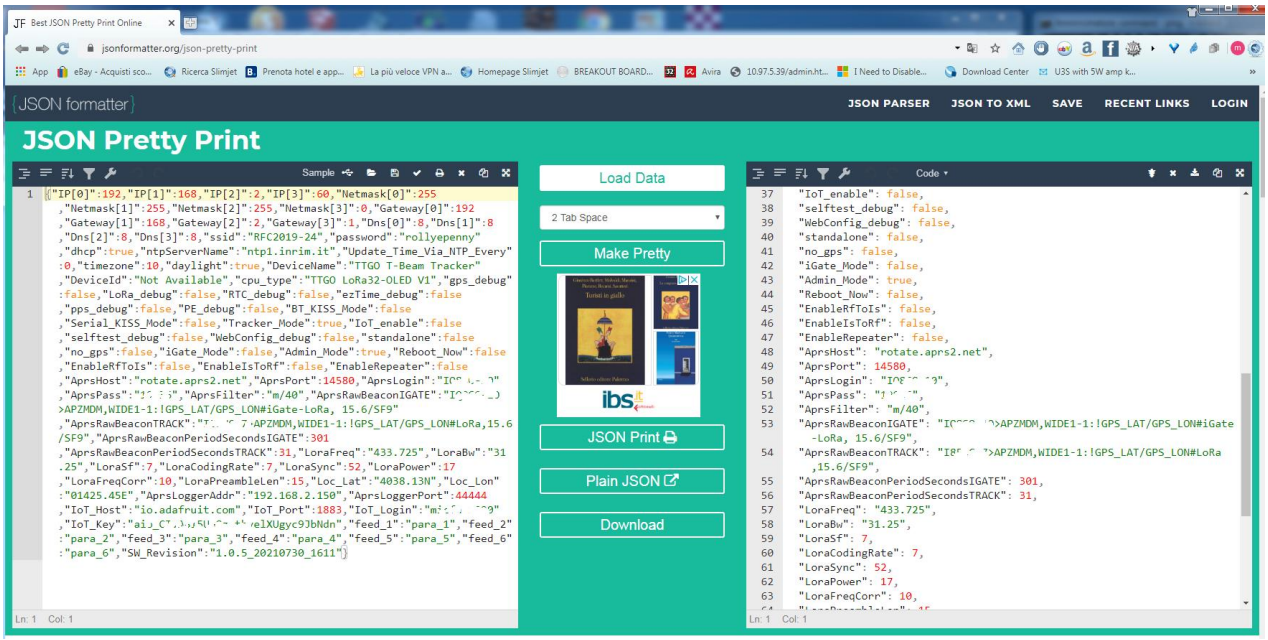

Figure 11-7 : JSON Pretty Print while displaying config file

12 Suggestions for the components procurement

All the components necessary for the assembly of both versions of LoRa Beacon can be easily purchased on the classic Asian portals; it is important to highlight that although it is generally very easy to find what is necessary, **the experimenter must be careful in purchasing components that are apparently identical to the recommended ones, but much cheaper:** unfortunately it is a classic trap as, as is well known in practice there is no guarantee that these cheaper components comply with what is declared or even shown in the photo.

The advice is to look for the deal, but making sure that it is really a deal!

As mentioned, all the components are easily available; the only components obviously not available on these portals are the printed circuits which can be requested at the following address info@sarimesh.net : the price of these PCBs is however very low as it only cover manufacturing costs and shipping.

Below is a purely indicative list, at the date, of the components required for the assembly of the two versions of LoRa Beacon: the list is unique for the two versions so, based on the version you want to assemble, it will be necessary to purchase only a subset of components. **Of course, no responsibility is assumed for any errors contained in this list, nor for the reliability of the sellers indicated.**

Shipping times from China have recently dropped a lot thanks to the new regulation on purchases made on online portals; however, it must be considered that all the prices seen on the portals must then be increased by the value of VAT (22%) when the orders are concluded.

The following list is updated to 05/08/2021.

- ESP32-WROOM-32D microcontroller 38 pin version with molded antenna
- I2C 0.96" display
- SPI IPS Display 1.14" 135x240 LCD Module
- FRAM FM24W256
- | GPS modules NEO-6M NEO-7M NEO-8M
- Port Expander PCF8574
- LoRa module 1 Watt SX1268 Ebyte E22-400M30S
- LoRa Module 100milliWatt SX1268 E22-400M22S
- LoRa module 100 milliWatt SX1278 RFM98W
- Buzzer KY-012
- Vertical USB connector for PCB
- Assortment of diodes
- 2N2222 transistors
- Assortment of 5 mm red and green LEDs
- MCP1700
- 6x6x12 vertical button
- Contact strips 2.54 mm high 7.1 mm female
- pin strips 2.54 mm male right angle type R1
- DS3231 RTC
- IPX-SMA cable 15 cm
- Straight male 2.54 mm pin strips
- Assortment of ceramic capacitors
- Resistance assortment
- 16-pin DIP socket
- Antenna with magnet and SMA 433 Mhz attack
- USB extension cable
- 12V power connector
- Active external GPS antenna
- DC/DC Step down conv.
- Assortment of electrolytic capacitors