

# Superviseur solaire

J. Lemaire

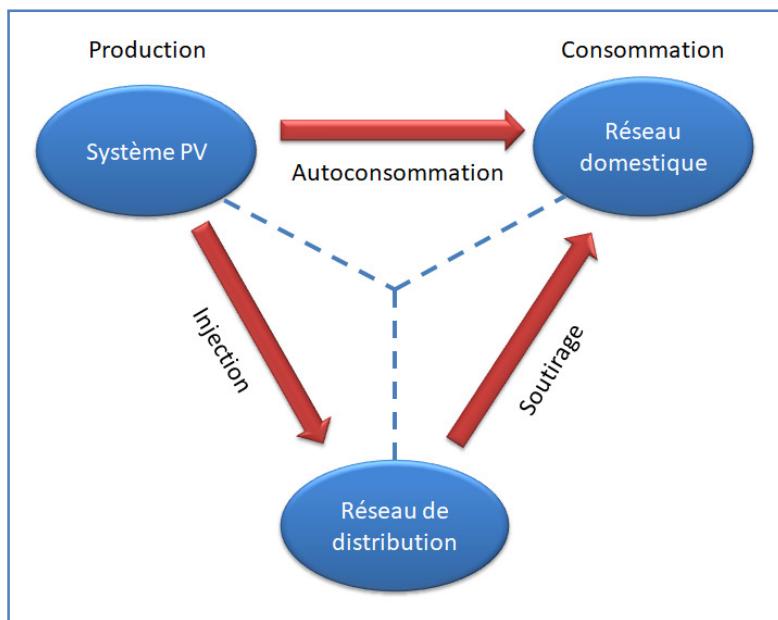




# 1 Objectifs

Ce petit projet était au départ destiné à tester un certain nombre de fonctionnalités, en vue de réaliser un routeur statique, dans une installation photovoltaïque en autoconsommation, fonctionnant en mode 0-injection.

Les tests étant faits, il a été tentant de les mettre à profit pour réaliser un petit système de supervision solaire dans le cas d'une installation photovoltaïque en autoconsommation. On s'intéresse donc au flux suivants (puissance ou énergie) :



Comme à tout instant :

- **Autoconsommation** = min (**Production**, **Consommation**) ;
- **Injection** = **Production** – **Autoconsommation** ;
- **Soutirage** = **Consommation** – **Autoconsommation** ;

il suffit de connaître la **Production** et la **Consommation** pour pouvoir calculer toutes les autres quantités et on notera que le **Soutirage** est nul si l'**Injection** est positive et vice versa.

Ainsi :

- **Consommation** = **Production** - **Injection** + **Soutirage**

On pourra donc tout calculer dans le cas d'un onduleur de chaîne, capable d'évaluer la **Production** avec un compteur de puissance/énergie sur la branche verticale, capable d'évaluer l'**Injection/Soutirage**.

On notera aussi que ces données sont souvent disponibles via une API avec un protocole standard de communication. C'est le cas par exemple avec un onduleur Fronius Primo GEN24 Plus et un compteur SmartMeter positionné « au point d'injection »<sup>1</sup> où la carte de Pilot de l'onduleur peut fournir ces données dans un format JSON, via un serveur Web interne, accessible en Wifi.

---

<sup>1</sup> La branche verticale dans le schéma précédent.

On notera que ces données temps réel sont datées. Là encore la date et l'heure peuvent être obtenues de l'API, mais il est plus simple et plus sûr de les initialiser en utilisant un serveur NTP<sup>2</sup>, puis de les mettre à jour en utilisant seulement un système interne au superviseur, qui sera basé sur un microcontrôleur offrant ces possibilités.

En résumé, Le système de supervision ici réalisé aura les fonctionnalités suivantes :

- obtention des données de **Production** et d'**Injection**, via un accès Wifi au serveur de ces données ;
- obtention de l'heure courante pour les dater, avec initialisation par appel un serveur NTP dans le setup ;
- affichage en temps réel des puissances de **Consommation** et de **Production** ;
- affichage de la date, de l'heure et de la tension de la batterie ;
- enregistrement de ces données sur une carte SD, en estimant régulièrement (toutes les 5mn par exemple) les énergies moyennes correspondantes ;
- utilisation d'un buffer circulaire pour stocker ces données quand la carte SD n'est pas accessible ;
- affichage de données journalières, énergies de **Consommation** et de **Production**, en intégrant les données de la carte SD ;
- protection contre les coupures d'alimentation.

Comme annoncé au départ, ce projet a permis de tester les techniques suivantes sur un ESP32 et un onduleur Fronius Primo GEN24 Plus :

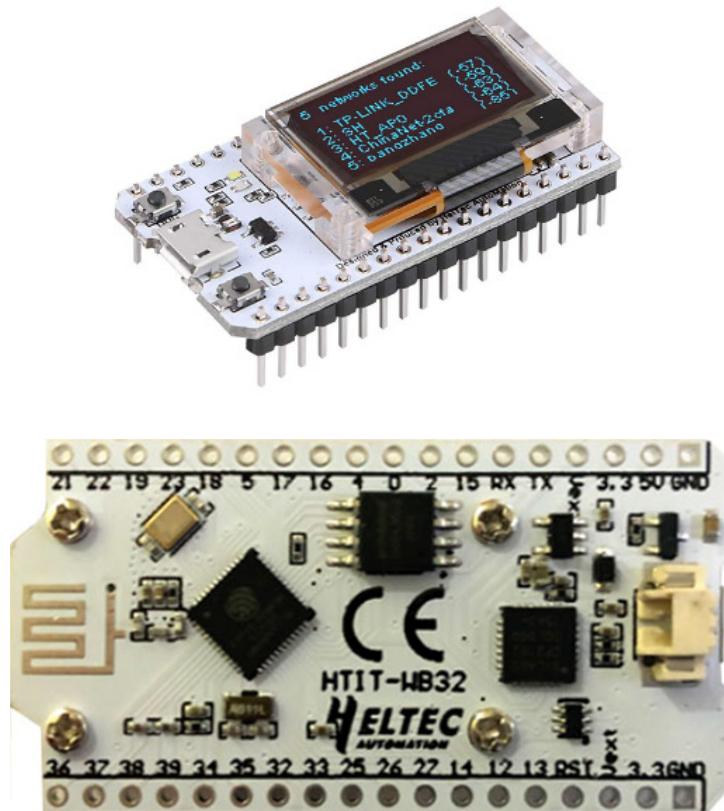
- affichage OLED ;
- client Wifi ;
- bibliothèque Arduino\_JSON pour décoder les données suivantes ;
- activation et utilisation de SOLAR.API sur l'onduleur pour obtenir les puissances de consommation et de production ;
- utilisation de la bibliothèque POSIX time pour maintenir la date et l'heure avec initialisation par appel un server NTP ;
- gestion de la batterie avec affichage de la tension ;
- enregistrement des données courantes dans un buffer circulaire ;
- sauvegardes de moyennes sur la carte SD ;
- boutons poussoirs avec traitement par interruption pour commander les affichages ;
- boîtier réalisé avec imprimante 3D.

---

<sup>2</sup> Network Time Protocol.

## 2 Composants

### 2.1 Carte microcontrôleur<sup>3</sup> <sup>4</sup>



#### 2.1.1 Caractéristiques

Ici, la carte **Heltec Wifi Kit 32 version 2.1<sup>5</sup>** a été choisie pour les raisons suivantes :

- je l'avais sous la main !
- excellent rapport qualité/prix ;
- processeur ESP32 DOWDQ6 dual core 32-bit pouvant fonctionner à 240MHz au maximum ;
- 520 kB de mémoire SRAM ;
- 8 MB de mémoire flash SPI ;
- Wifi avec antenne intégrée 2,4GHz ;
- écran OLED 0,96 " 128x64 ;
- 2 interfaces I2C ;
- 4 interfaces SPI ;
- 2 interfaces UART ;
- 43 GPIO programmables ;
- interface USB (CP2102) pour flashage, monitoring et alimentation (5V) ;
- connecteur SH1.25 pour alimentation par batterie LiPo 3,7V ;
- circuit SH1.25-2 pour la gestion de la batterie :

<sup>3</sup> <https://heltec.org/project/wifi-kit-32/>

<sup>4</sup> [https://docs.heltec.cn/en/arduino/esp32\\_arduino/wifi\\_kit\\_32/hardware\\_update\\_log.html](https://docs.heltec.cn/en/arduino/esp32_arduino/wifi_kit_32/hardware_update_log.html)

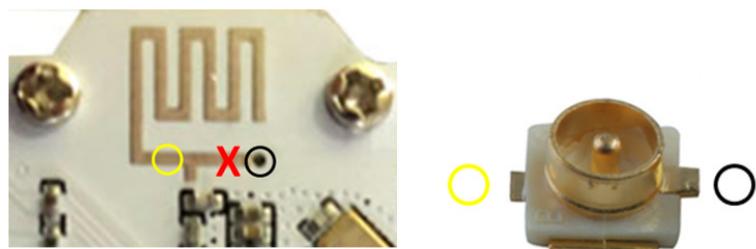
<sup>5</sup> Le produit, épuisé, peut être remplacé par sa version 3 (<https://heltec.org/project/wifi-kit-32-v3/>), dont la fonctionnalité LORA ne sera pas utilisée ici.

- LTH7R pour le contrôle de la charge (à partir de l'alimentation USB) et de la décharge ;
- protection contre la surcharge ;
- sélection automatique de l'alimentation USB si double alimentation ;
- tension sur PIN 37 ;
- programmation possible avec l'IDE Arduino ;
- nombreuses bibliothèques et tutoriels disponibles.

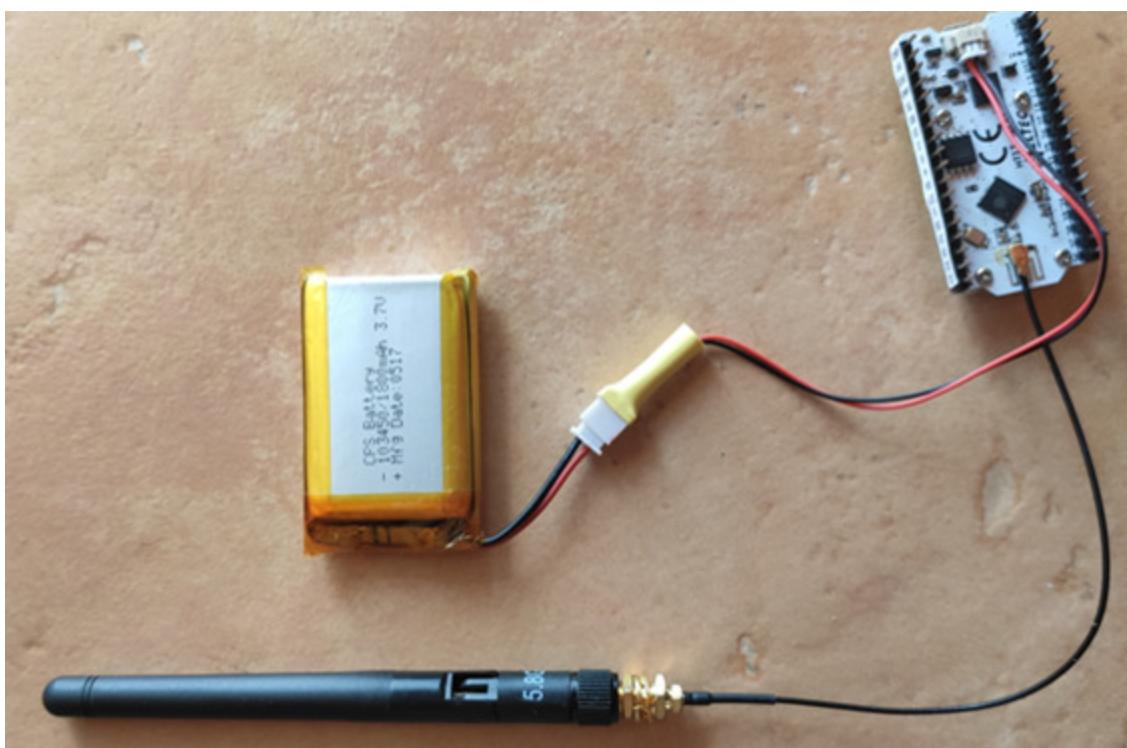
## 2.1.2 Ajout d'une antenne externe

Bien qu'améliorée dans cette version l'antenne Wifi intégrée n'a pas permis de se connecter de manière sûre au point d'accès Wifi utilisé ici, lorsque la carte était un peu éloignée de ce dernier.

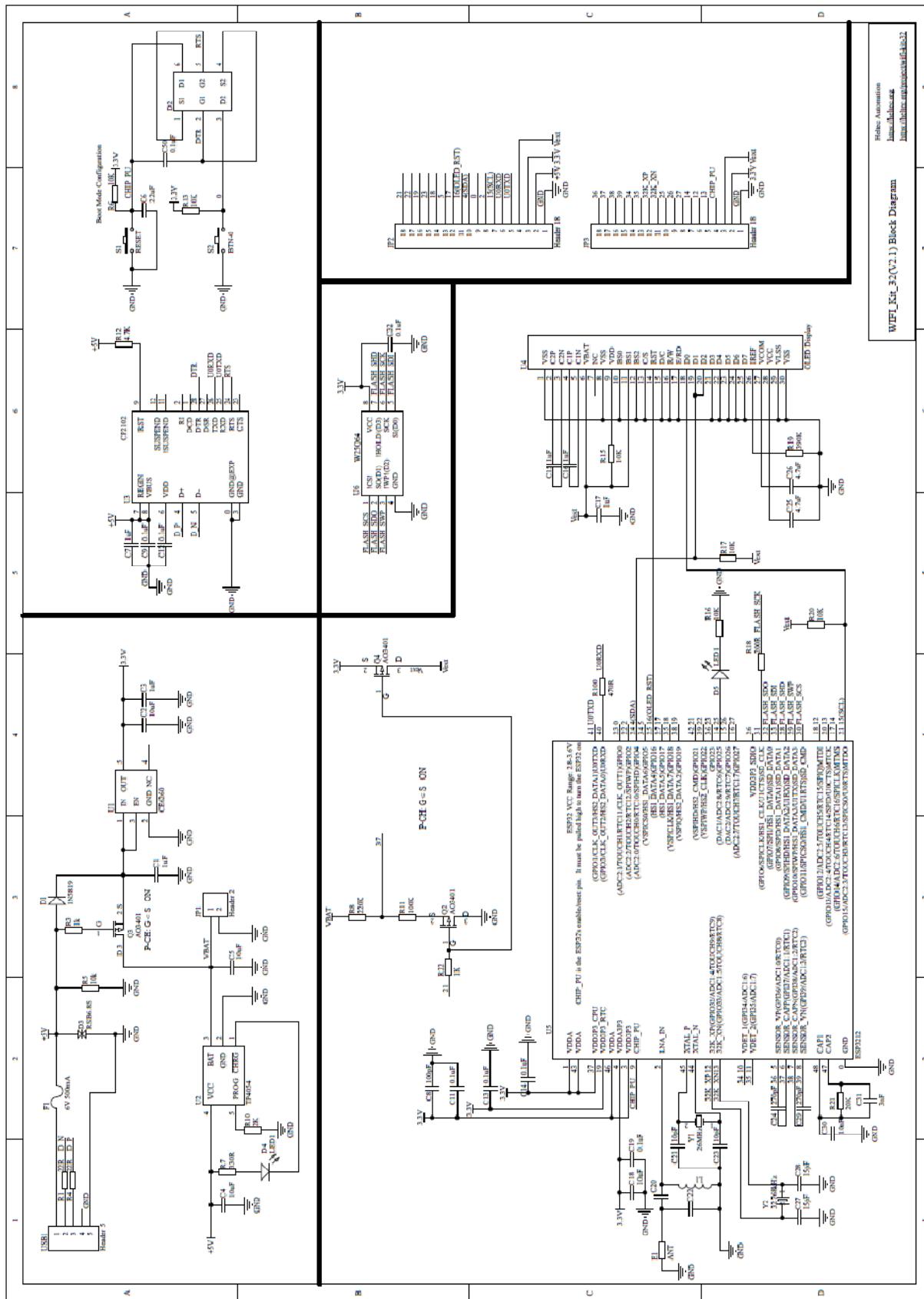
On l'a donc remplacée par une antenne externe de 5,8GHz (meilleure caractéristique observée), branchée sur un connecteur IPEX U.FL mâle soudé sur la carte, après avoir sectionné la branche de l'antenne entre le signal et la masse :



Un câble d'antenne coaxial avec un connecteur U.FL femelle et un connecteur SMA femelle à ses extrémités est ici utilisé pour brancher l'antenne :

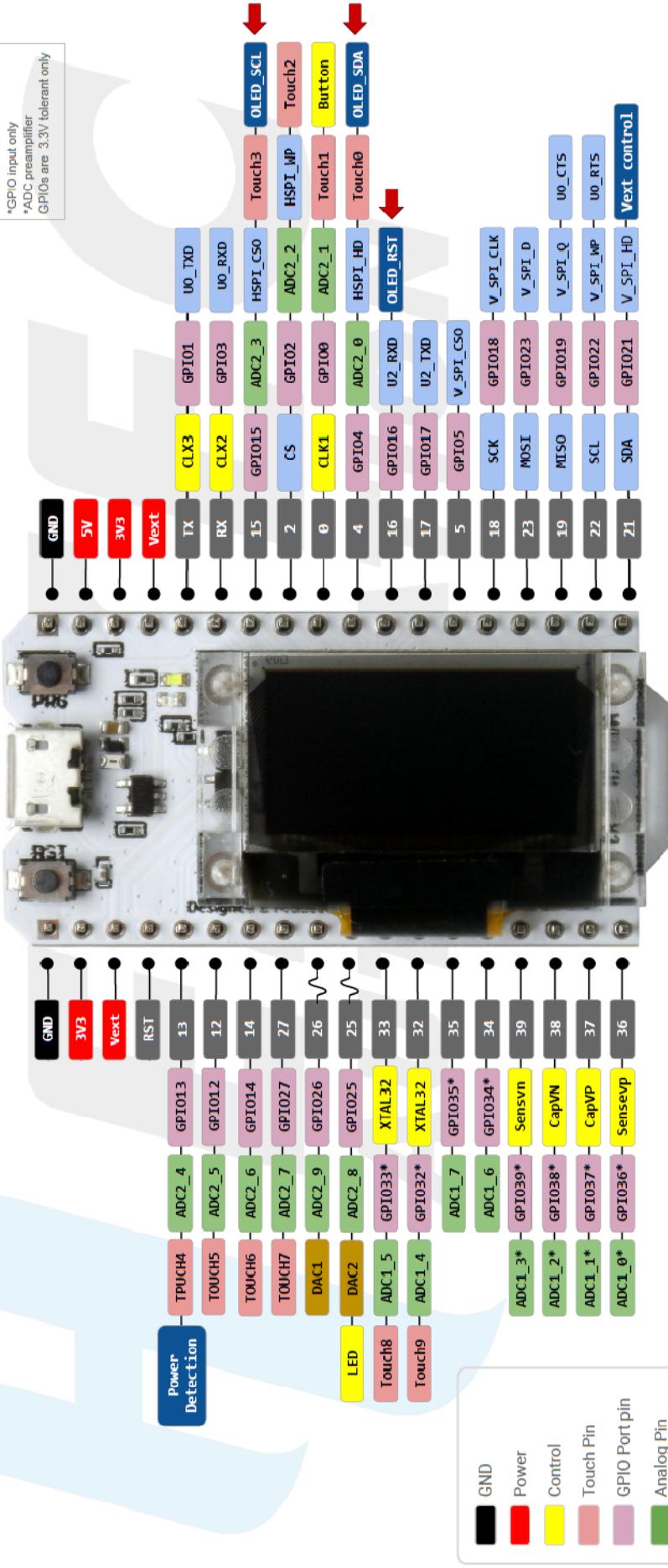


### 2.1.3 Schémas



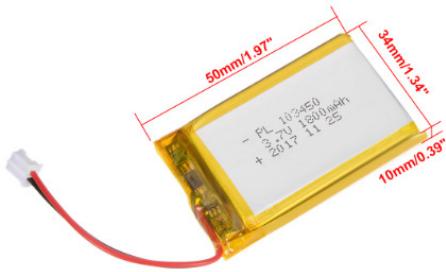
NEW WIFI Kit 32 Pinout Diagram

Pins with this arrow are used by on-board OLED, they must not be used for other purpose unless you know what you are doing!



\*GPIO input only  
\*ADC preamplifier  
GPIOs are 3.3V tolerant only

## 2.2 Batterie



- Lipo 3,7V 1800mAh.
- 103450 : 10x34x50mm.
- Circuit de protection contre la surcharge et la décharge excessive
- Connecteur JST PH2

## 2.3 Lecteur de carte micro SD<sup>6</sup> <sup>7</sup>



- SPI.
- Format Fat32.
- Broches utilisées :
  - MISO = 19 ;
  - MOSI = 23 ;
  - CLK (SCK) = 18 ;
  - CS = 5 ;
  - GND = GND ;
  - 3V3 = 3V3.

## 2.4 Carte micro SD<sup>8</sup>



- 32 GB.
- SDHC.
- Classe 10 (10 Mb/s).

<sup>6</sup> <https://eckstein-shop.de/MicroSDBreakoutBoardfC3BCrSD2FTFKarteC3BCrArduino32C3V6Pin>

<sup>7</sup> <https://randomnerdtutorials.com/esp32-microsd-card-arduino/>

<sup>8</sup> <https://eckstein-shop.de/32GBMicroSDCardClass10SpeicherkartemitAdapter>

## 2.5 Antenne RF



- 5,8 GHz.
- 3 dB.
- Omnidirectionnelle.
- Connecteur SMA male.

## 2.6 Câble coaxial RF



- $50\Omega$ .
- 15 cm.
- Connecteur SMA femelle.
- Connecteur U.FL femelle

## 2.7 Boutons-poussoir miniatures<sup>9</sup>

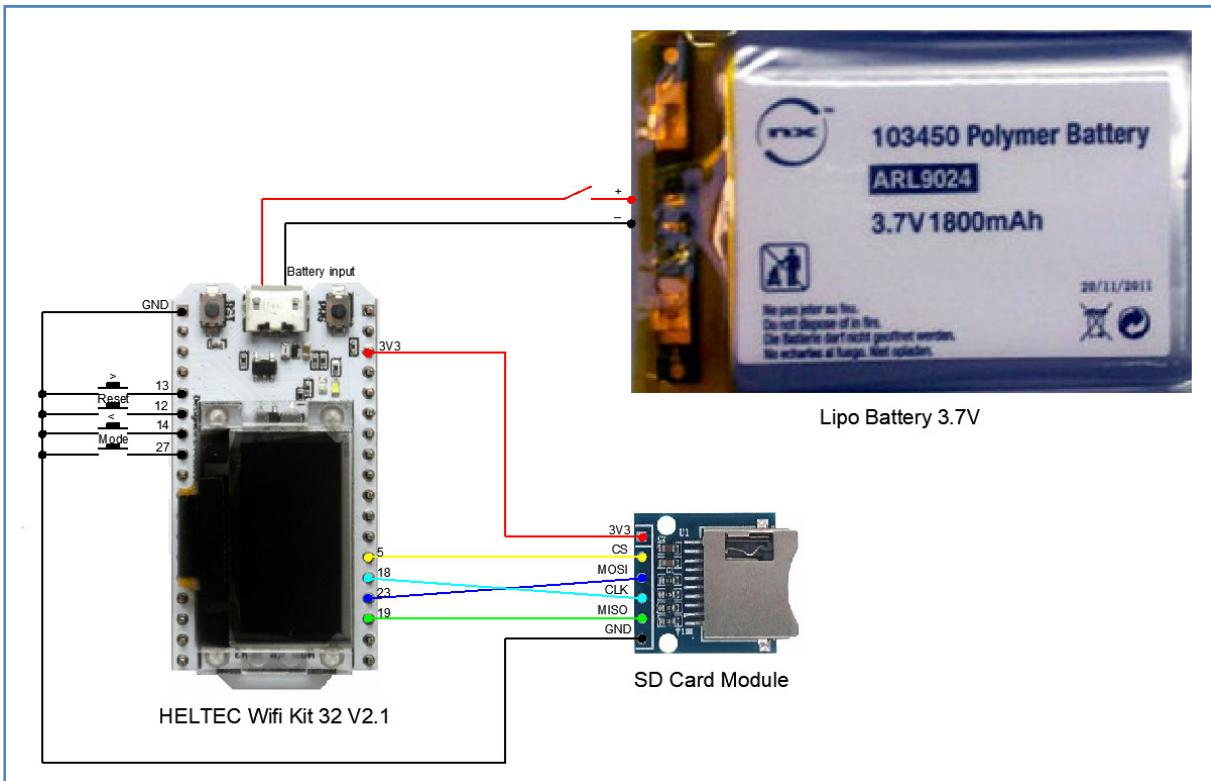


- 6x6x6mm

---

<sup>9</sup> <https://composant-electronique.fr/bouton-poussoir-miniature-microswitch-sw100664-060-160>

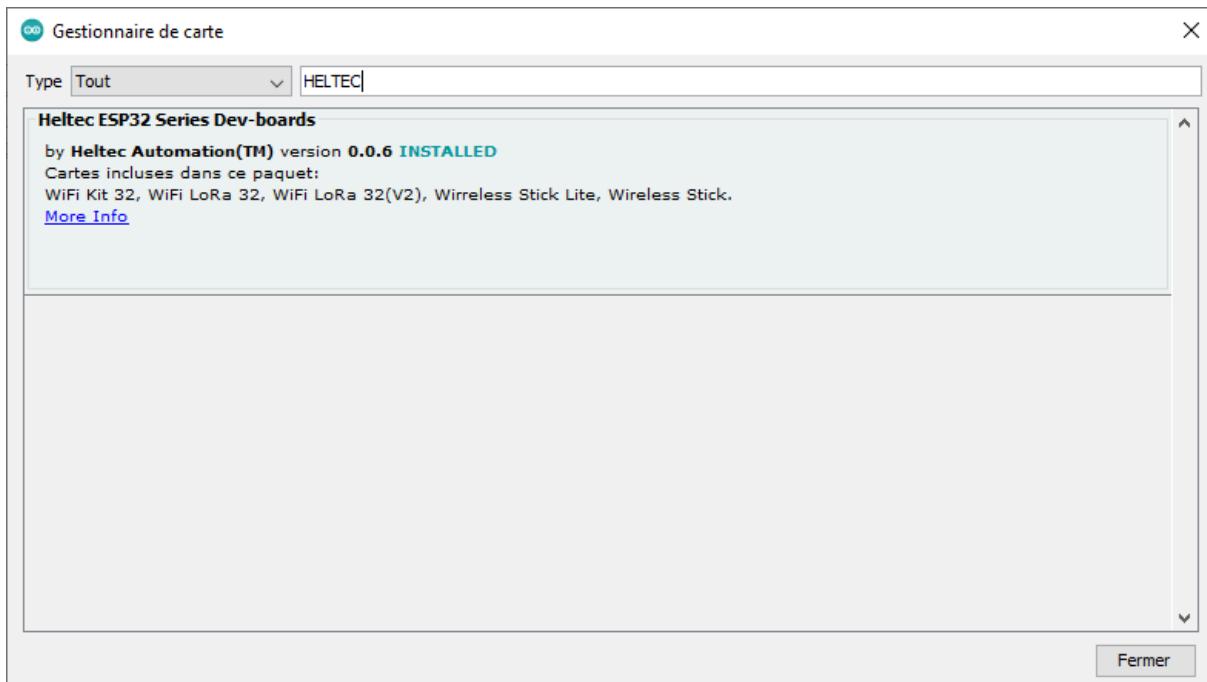
### 3 Schéma de câblage



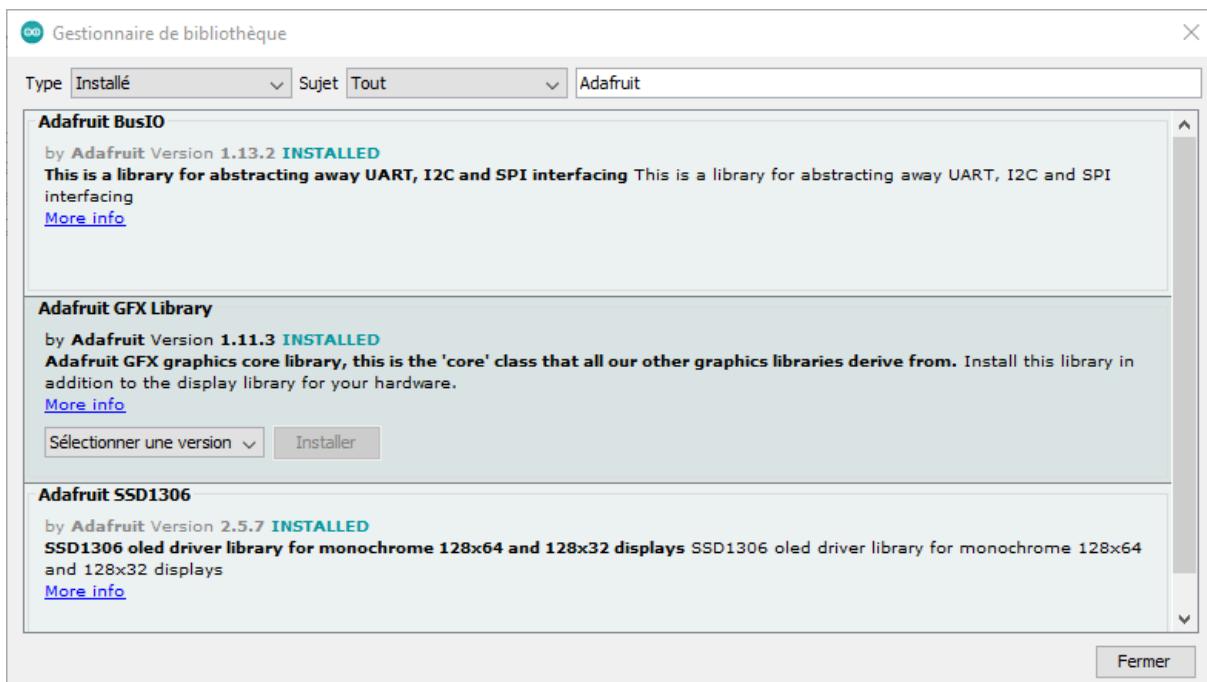
# 4 Programmes

## 4.1 Préparation de l'IDE Arduino

Dans l'IDE Arduino, il faut commencer par ajouter les cartes de développement HELTEC à base d'ESP32<sup>10</sup> :



Avec le Gestionnaire de bibliothèque, on sera amené à installer des bibliothèques spécifiques, notamment d'Adafruit :



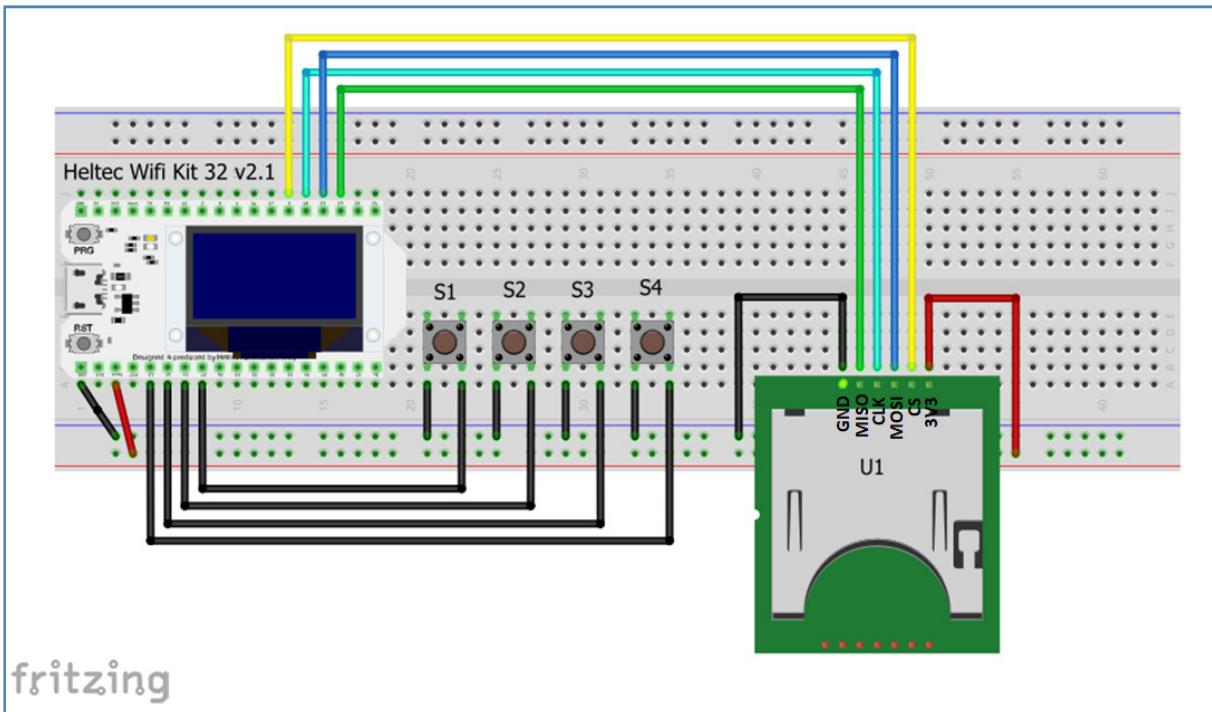
Mais ceci sera précisé dans le paragraphe suivant.

<sup>10</sup> [https://docs.heltec.org/en/node/esp32/quick\\_start.html](https://docs.heltec.org/en/node/esp32/quick_start.html)

## 4.2 Tests

## 4.2.1 Circuit de test

Pour faciliter la compréhension, toutes les fonctionnalités ici mises en œuvre ont été testées séparément sur une platine d'essai<sup>11</sup> :



A noter qu'une batterie Lipo de 3,7V et une antenne externe de 5,6GHz sont aussi connectées à la carte Heltec.

## 4.2.2 Led

C'est le « Hello world » des microcontrôleur et ici **LED\_BUILTIN** = 25.

```
// TestLed.ino

/*
  Led blinking with a 2s period.
  Adapté de : http://www.arduino.cc/en/Tutorial/Blink
*/

void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

<sup>11</sup> <https://forum.fritzing.org/t/heltec-new-wifi-kit-32-v2/11996>

## 4.2.3 Ecran OLED

La carte Heltec Wifi Kit 32 v2.1 intègre un écran OLED monochrome 128x64 SSD1306 connecté en I2C, à l'adresse 0x3C.

Comme dans l'exemple d'Henri Bachetti<sup>12</sup>, la bibliothèque générique **Adafruit\_SSD1306**<sup>13</sup> est ici utilisée. Celle-ci s'appuie sur la bibliothèque **Adafruit\_GFX**<sup>14</sup> qui en particulier propose différentes polices d'affichage, dont **FreeSerif9pt7b** qui offre des lettres plus petites.

Pour la liaison I2C, l'objet **Wire1** de la bibliothèque **Wire** est ici employé, de manière pouvoir préciser les broches SDA (SDA\_OLED) et SCL (SCL\_OLED)<sup>15</sup>.

```
// TestOLED.ino

/*
  OLED display test using 2 fonts.

  References :
  - Henri Bachetti : https://riton-duino.blogspot.com/2021/06/esp32-heltec-wifi-kit-32.html
  - Library Adafruit_SSD1306 : https://github.com/adafruit/Adafruit_SSD1306
  - Library Adafruit_GFX : https://github.com/adafruit/Adafruit-GFX-Library
*/

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Fonts/FreeSerif9pt7b.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define SCREEN_ADDRESS 0x3C

// Declaration for an SSD1306 display connected to I2C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire1, RST_OLED);

void setup()
{
    // Init OLED
    Wire1.begin(SDA_OLED, SCL_OLED);
    if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) for(;;);
    display.setTextColor(WHITE);
}

void displayPower()
{
    display.clearDisplay();
    display.setFont();
    display.setCursor(0, 3);
    display.println("04/09/22:08h15 98%");
    display.drawRect(92,0, 32, 14, 1);
    display.setFont(&FreeSerif9pt7b);
    display.setCursor(0, 35);
    display.println("c 3400W");
    display.println("p 4021W");
    display.display();
}

void displayEnergy()
{
    display.clearDisplay();
    display.setFont();
    display.setCursor(0, 3);
    display.println("04/09/22");
    display.setFont(&FreeSerif9pt7b);
    display.setCursor(0, 35);
    display.println("c 12.1kWh 100%");
}
```

<sup>12</sup> <https://riton-duino.blogspot.com/2021/06/esp32-heltec-wifi-kit-32.html>

<sup>13</sup> [https://github.com/adafruit/Adafruit\\_SSD1306](https://github.com/adafruit/Adafruit_SSD1306)

<sup>14</sup> <https://github.com/adafruit/Adafruit-GFX-Library>

<sup>15</sup> Ces paramètres sont connus dès qu'on sélectionne la carte **Wifi Kit 32** dans l'IDE Arduino.

```

    display.println("p 4.3kWh 100%");
    display.display();
}

void loop()
{
    displayPower();
    delay(2000);
    displayEnergy();
    delay(2000);
}

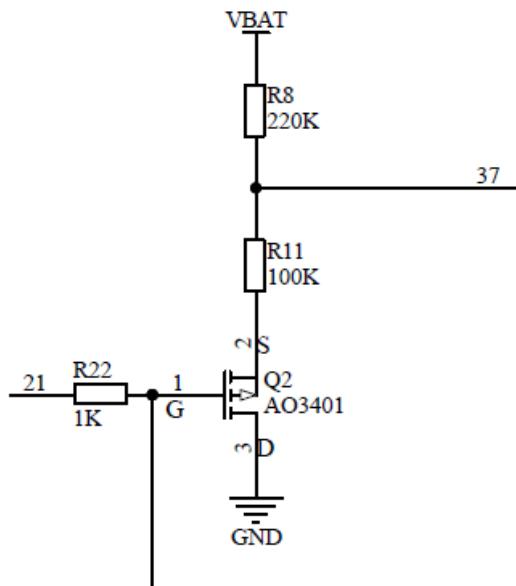
```



#### 4.2.4 Tension batterie

On observe d'abord que la carte Heltec peut bien être alimentée par une batterie (4,7V) et/ou par une connexion USB (5V). De plus, si ces 2 sources sont présentes, la batterie est rechargée si nécessaire. Elle peut donc être utilisée pour continuer alimenter la carte, en cas de coupure secteur.

Dans cette version de la carte Heltec Wifi Kit 32, la tension de la batterie est disponible sur la broche n° 37 et on peut donc la mesurer en appelant **analogRead<sup>16</sup>**.



A noter aussi l'utilisation de la bibliothèque **elapsedMillis<sup>17</sup>** pour définir un compteur de temps à rebours ; cette technique sera systématique dans les tests suivants.

```

// TestBatt.ino

/*
Periodic display of the battery voltage.

References :
- https://github.com/3KUdelta/heltec\_wifi\_kit\_32\_batt\_monitor/blob/master/Wifi%20Kit%2032%20Batt%20Watch.ino

```

<sup>16</sup> <https://randomnerdtutorials.com/esp32-adc-analog-read-arduino-ide/> par exemple.

<sup>17</sup> <https://github.com/pfeerick/elapsedMillis>

```

- https://github.com/pfeerick/elapsedMillis
*/
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#include <elapsedMillis.h>

// Screen constants
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define SCREEN_ADDRESS 0x3C

// Battery constants
#define VBAT_PIN 37           // GPIO connected to VBAT in Heltec Wifi kit 32 V2.1
#define KV 1.7E-3              // Coefficient to convert in Volts the analog reads

// Period (ms) to display the voltage
#define DISPLAY_VOLTAGE_PERIOD 5000    // 5s

// Declaration for an SSD1306 display connected to I2C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire1, RST_OLED);

// Timer to display voltage
elapsedMillis tmrDisplayVoltage;

void setup()
{
    // Init OLED
    Wire1.begin(SDA_OLED, SCL_OLED);
    if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) for(;;);
    display.setTextColor(WHITE);
}

void loop()
{
    if (tmrDisplayVoltage > DISPLAY_VOLTAGE_PERIOD)
    {
        // Reset timer
        tmrDisplayVoltage = 0;

        // Get voltage
        float voltage = KV*analogRead(VBAT_PIN);

        // OLED display
        display.clearDisplay();
        display.drawRect(93,0, 32, 13, 1);
        display.setCursor(97, 3);
        display.print(voltage, 1);
        display.println('V');
        display.display();
    }
}

```



## 4.2.5 Wifi

C'est l'un des principaux avantages du microcontrôleur ESP32.

On teste ici la connexion Wifi comme station au point d'accès Wifi du réseau local (Livebox).

```

// TestWifi.ino

/*
Test Wifi station

```

```

Reference :
- https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/
*/

#include <WiFi.h>

const char* ssid = "Livebox-486c";
const char* password = "FAAD971F372AA5CCE13D25969E";

void setup()
{
    Serial.begin(115200);
    delay(1000);

    //WiFi.mode(WIFI_STA); //Optional
    WiFi.begin(ssid, password);
    Serial.println("\nConnecting");

    while(WiFi.status() != WL_CONNECTED){
        Serial.print(".");
        delay(100);
    }

    Serial.println("\nConnected to the WiFi network");
    Serial.print("Local ESP32 IP: ");
    Serial.println(WiFi.localIP());
}

void loop(){}

```

```

Connecting
...
Connected to the WiFi network
Local ESP32 IP: 192.168.1.26

```

## 4.2.6 Solar API

La fonctionnalité Wifi testée précédemment va d'abord être utilisée pour accéder à l'API<sup>18</sup> **Solar API V1**<sup>19</sup>, activable dans l'onduleur avec le menu **Communication/Solar API** :

### Solar API

#### Remarque

The Solar API is an Ethernet-based, open JSON interface. If enabled, IOT devices in the local network may access inverter information without authentication. For security reasons the interface is disabled by default and should not be enabled if it is not required for a 3rd party application (e.g. EV charger, smart home solutions, etc.).

For monitoring Fronius recommends to use [Solar.web](#) instead, which provides secure access to inverter status and production information.



Activate communication via Solar API

Dans le cas d'un GEN24, Solar API expose ainsi plusieurs scripts CGI<sup>20</sup>, dans la page **solar\_api** du serveur intégré de sa carte Pilot :

<sup>18</sup> Application Programming Interface.

<sup>19</sup> <https://www.fronius.com/en/solar-energy/installers-partners/technical-data/all-products/system-monitoring/open-interfaces/fronius-solar-api-json->

- GetAPIVersion.cgi
- GetInverterRealtimeData.cgi
- GetMeterRealtimeData.cgi
- **Erreur ! Référence de lien hypertexte non valide.**

Le second nécessite des paramètres :

- Scope=Device|System
- DataCollection=CommonInverterData|CummulationInverterData quand Scope=Device

Ils sont passés dans l'URL ; on utilisera donc une requête HTTP de type **GET**. Et dans tous les cas, le serveur interne de la carte Pilot retourne des données textuelles formatées en **JSON**<sup>21</sup>.

Par exemple, avec [http://192.168.1.13/solar\\_api/v1/GetInverterRealtimeData.cgi?Scope=System](http://192.168.1.13/solar_api/v1/GetInverterRealtimeData.cgi?Scope=System) on obtient<sup>22</sup> :

```
{
  "Body" : {
    "Data" : {
      "DAY_ENERGY" : {
        "Unit" : "Wh",
        "Values" : {
          "1" : null
        }
      },
      "PAC" : {
        "Unit" : "W",
        "Values" : {
          "1" : 108.48963165283203
        }
      },
      "TOTAL_ENERGY" : {
        "Unit" : "Wh",
        "Values" : {
          "1" : 718826.934444444449
        }
      },
      "YEAR_ENERGY" : {
        "Unit" : "Wh",
        "Values" : {
          "1" : null
        }
      }
    }
  },
  "Head" : {
    "RequestArguments" : {
      "Scope" : "System"
    },
    "Status" : {
      "Code" : 0,
      "Reason" : "",
      "UserMessage" : ""
    },
    "Timestamp" : "2022-10-12T16:18:15+00:00"
  }
}
```

On notera que la puissance de la **Production** AC est ici renommée en Watt :

```
{
  "Body" : {
    "Data" : {
      "DAY_ENERGY" : {
        "Unit" : "Wh",
        "Values" : {
```

<sup>20</sup> Common Gateway Interface.

<sup>21</sup> Java Script Object Notation : [https://fr.wikipedia.org/wiki/JavaScript\\_Object\\_Notation](https://fr.wikipedia.org/wiki/JavaScript_Object_Notation) par exemple.

<sup>22</sup> On rappelle que 192.168.1.13 est l'adresse IP attribuée à la carte Wifi de l'onduleur sur le réseau local.

```

        "1" : null
    },
},
"PAC" : {
    "Values" : {
        "1" : 108.48963165283203
    }
},
...
},
...
}
}

```

Elle peut être obtenue en décodant la chaîne retournée avec une instruction de la forme :

```
prod = double(JSON.parse(http.getString())["Body"]["Data"]["PAC"]["Values"]["1"]);
```

De la même manière, **Erreur ! Référence de lien hypertexte non valide.** conduit à :

```
{
    "Body" : {
        "Data" : {
            "0" : {
                "Current_AC_Phase_1" : 1.8320000000000001,
                "Current_AC_Sum" : 1.8320000000000001,
                "Details" : {
                    "Manufacturer" : "Fronius",
                    "Model" : "Smart Meter TS 100A-1",
                    "Serial" : "3683975661"
                },
                "Enable" : 1,
                "EnergyReactive_VArAC_Sum_Consumed" : 260805.0,
                "EnergyReactive_VArAC_Sum_Produced" : 58702.0,
                "EnergyReal_WAC_Minus_Absolute" : 3056.0,
                "EnergyReal_WAC_Plus_Absolute" : 698158.0,
                "EnergyReal_WAC_Sum_Consumed" : 698158.0,
                "EnergyReal_WAC_Sum_Produced" : 3056.0,
                "Frequency_Phase_Average" : 49.89999999999999,
                "Meter_Location_Current" : 0.0,
                "PowerApparent_S_Phase_1" : 398.8000000000001,
                "PowerApparent_S_Sum" : 398.8000000000001,
                "PowerFactor_Phase_1" : -0.997,
                "PowerFactor_Sum" : -0.997,
                "PowerReactive_Q_Phase_1" : 25.600000000000001,
                "PowerReactive_Q_Sum" : 25.600000000000001,
                "PowerReal_P_Phase_1" : 398.0,
                "PowerReal_P_Sum" : 398.0,
                "TimeStamp" : 1665592701,
                "Visible" : 1,
                "Voltage_AC_Phase_1" : 230.3000000000001
            }
        }
    },
    "Head" : {
        "RequestArguments" : {
            "Scope" : "System"
        },
        "Status" : {
            "Code" : 0,
            "Reason" : "",
            "UserMessage" : ""
        },
        "Timestamp" : "2022-10-12T16:38:21+00:00"
    }
}
```

Elle indique en particulier la puissance réelle en Watt du flot mesuré par le Smart Meter :

```
{
    "Body" : {
        "Data" : {
            "0" : {
                ...
                "PowerReal_P_Phase_1" : 398.0,

```

```

        }
    },
},
...
}

```

Si cette puissance est > 0, c'est la valeur du **Soutirage**, sinon c'est celle de l'**Injection**.

On peut donc obtenir la **Consommation** avec une instruction de la forme :

```
cons = prod + double(JSON.parse(http.getString())["Body"]["Data"][0]["PowerReal_P_Phase_1"]);
```

Pour mettre en œuvre ce décodage, la bibliothèque **Arduino\_JSON**<sup>23</sup> sera utilisée.

Le sketch suivant illustre tout cela en établissant une connexion Wifi entre l'ESP32 et la carte Pilot de l'onduleur et on notera que la validité de cette connexion est testée en permanence. Dans ce cas, des requêtes HTTP GET sont envoyées et les réponses JSON sont décodées comme indiqué précédemment.

```
// TestJSON.ino

/*
Wifi access to Solar API.

References :
- https://github.com/arduino-libraries/Arduino\_JSON
- https://randomnerdtutorials.com/esp32-http-get-post-arduino/
- https://github.com/pfeerick/elapsedMillis
*/

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Fonts/FreeSerif9pt7b.h>

#include <WiFiClient.h>
#include <HTTPClient.h>
#include <Arduino_JSON.h>

#include <elapsedMillis.h>

// Screen constants
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define SCREEN_ADDRESS 0x3C

// Wifi access point constants
const char *SSID      = "Livebox-486c";
const char *PWD       = "FAAAD971F372AA5CCE13D25969E";

// Web service host address
const char *HOST = "192.168.1.13";

// Period (ms) to obtain and display JSON data
#define JSON_PERIOD 5000 // 5s

// Declaration for an SSD1306 display connected to I2C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire1, RST_OLED);

// Timer for JSON
elapsedMillis tmrJSON;

// SOLAR.API parameters
double prod, cons;

void connectToWiFi(const char * ssid, const char * pwd)
{
    WiFi.begin(ssid, pwd);
    while (WiFi.status() != WL_CONNECTED) delay(500);
```

---

<sup>23</sup> [https://github.com/arduino-libraries/Arduino\\_JSON](https://github.com/arduino-libraries/Arduino_JSON)

```

}

void setup()
{
    // Init OLED
    Wire1.begin(SDA_OLED, SCL_OLED);
    if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) for(;;);
    display.setTextColor(WHITE);
    display.setFont(&FreeSerif9pt7b);

    // Connect to the Wifi access point
    connectToWiFi(SSID, PWD);
}

void loop()
{
    if (tmrJSON > JSON_PERIOD)
    {
        // Reset timer
        tmrJSON = 0;

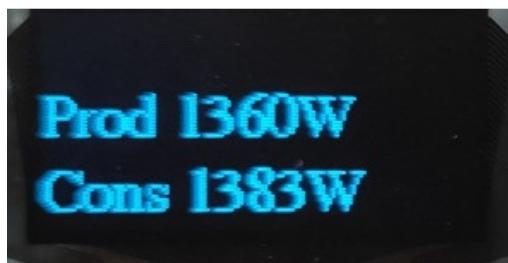
        if(WiFi.status() == WL_CONNECTED)
        {
            // Local variables
            HTTPClient http;

            // Request Production
            http.begin("http://" + String(HOST) + "/solar_api/v1/GetInverterRealtimeData.cgi?Scope=System");
            if(http.GET() == HTTP_CODE_OK)
                prod = double(JSON.parse(http.getString())["Body"]["Data"]["PAC"]["Values"]["1"]);
            http.end();

            // Request Consommation
            http.begin("http://" + String(HOST) + "/solar_api/v1/GetMeterRealtimeData.cgi");
            if(http.GET() == HTTP_CODE_OK)
                cons = prod + double(JSON.parse(http.getString())["Body"]["Data"]["0"]["PowerReal_P_Phase_1"]);
            http.end();
        }
        else connectToWiFi(SSID, PWD); // WiFi reconnect

        // OLED display
        display.clearDisplay();
        display.setCursor(0, 35);
        display.print("Prod ");
        display.print(prod, 0); // No decimal
        display.println("W");
        display.print("Cons ");
        display.print(cons, 0); // No decimal
        display.println("W");
        display.display();
    }
}

```



#### 4.2.7 Date et heure

Les fonctionnalités Wifi de l'ESP32 seront également mises à profit pour obtenir la date et l'heure courante auprès d'un serveur NTP<sup>24</sup>, évitant ainsi l'ajout d'un module RTC<sup>25</sup>.

---

<sup>24</sup> Network Time Protocol : [https://fr.wikipedia.org/wiki/Network\\_Time\\_Protocol](https://fr.wikipedia.org/wiki/Network_Time_Protocol) par exemple.

<sup>25</sup> Real Time Clock.

Conformément à l'excellent tutoriel de Rui Santos<sup>26</sup>, on utilise ici la bibliothèque POSIX **time**<sup>27</sup> de l'ESP32 dont la fonction **configTime**<sup>28</sup> permet de déclarer un tel serveur NTP, qui sera uniquement utilisé pour les initialiser. Elles sont ensuite automatiquement mises à jour et on peut les obtenir grâce à la fonction **getLocalTime** qui les retourne dans une structure de type **tm**.

Pour les afficher avec mise en forme, **strftime** convient bien.

```
// TestDateTime.ino

/*
Display the date and the time using an NTP server at setup.

References :
- https://randomnerdtutorials.com/esp32-date-time-ntp-client-server-arduino/
- https://sourceware.org/newlib/libc.html#Timefns
- https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/esp32-hal-time.c#L47
- https://github.com/pfeerick/elapsedMillis
*/

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#include <WiFi.h>
#include <time.h>

#include <elapsedMillis.h>

// Screen constants
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define SCREEN_ADDRESS 0x3C

// Wifi access point constants
const char *ssid      = "Livebox-486c";
const char *password  = "FAAD971F372AA5CCE13D25969E";

// NTP Server constants
const char* ntpServer = "pool.ntp.org"; // Server address
const long gmtOffset_sec = 3600; // GMT+1 zone
const int  daylightOffset_sec = 3600; // Summer time

// Period (ms) to display the time
#define DISPLAY_TIME_PERIOD 30000 // 30s

// Declaration for an SSD1306 display connected to I2C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire1, RST_OLED);

// Time info structure
struct tm timeInfo;

// Timer to display time
elapsedMillis tmrDisplayTime;

void setup()
{
    // Init OLED
    Wire1.begin(SDA_OLED, SCL_OLED);
    if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) for(;;);
    display.setTextColor(WHITE);

    // Connect to the Wifi access point
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) delay(500);

    // Init the time
    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
    while (!getLocalTime(&timeInfo)) delay(100);
}
```

---

<sup>26</sup> <https://randomnerdtutorials.com/esp32-date-time-ntp-client-server-arduino/>

<sup>27</sup> <https://sourceware.org/newlib/libc.html#Timefns>

<sup>28</sup> <https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/esp32-hal-time.c#L47>

```

// Disconnect Wifi as it's no longer needed
WiFi.disconnect(true);
WiFi.mode(WIFI_OFF);

// Init the timer to immediately display time
tmrDisplayTime = DISPLAY_TIME_PERIOD;
}

void loop()
{
  if (tmrDisplayTime > DISPLAY_TIME_PERIOD)
  {
    // Reset timer
    tmrDisplayTime = 0;

    // Get the time
    getLocalTime(&timeInfo);

    // Display the time DD/MM/YY HHmm
    display.clearDisplay();
    display.setCursor(0, 3);
    char buff[15];
    strftime(buff, 15, "%d/%m/%y %H%M", &timeInfo);
    display.print(buff);
    display.display();
  }
}

```



#### 4.2.8 Carte SD

C'est du classique<sup>29</sup> et il suffit ici d'utiliser la bibliothèque **SD**, pour des modules SPI uniquement.

On essaie ici de simuler la création (s'il existe, il est d'abord détruit), puis le remplissage d'un fichier de texte Excel délimité, **22-09-2022.csv**, avec 3 colonnes, une ligne d'en-têtes (« Heure », « Production (W) » et « Consommation (W) »), et des lignes de données correspondant à un enregistrement toutes les 5 minutes.

```

// TestSD.ino

/*
File creation and data appening on a SD card.

References :
- https://RandomNerdTutorials.com/esp32-microsd-card-arduino/
- https://github.com/espressif/arduino-esp32/tree/master/libraries/SD
*/

#include "SD.h"

void setup()
{
  Serial.begin(115200);
  if(!SD.begin(5))
  {
    Serial.println("Card Mount Failed");
    return;
  }

  char buf[41];
  int j = 22;

```

---

<sup>29</sup> <https://randomnerdtutorials.com/esp32-microsd-card-arduino/> par exemple.

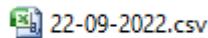
```

int m = 9;
int a = 2022;
sprintf(buf, "/%02d-%02d-%d.csv", j, m, a);
File file = SD.open(buf, FILE_WRITE);
if(!file)
{
    Serial.println("Failed to open file for writing");
    return;
}

sprintf(buf, "%s;%s;%s\n", "Heure", "Production(W)", "Consommation(W)");
if(!file.print(buf)) Serial.println("Append failed");
for (int h=0; h<24; h++)
{
    for (int m=0; m<60; m = m+5)
    {
        float p = random(10000);
        float c = random(10000);
        sprintf(buf, "%02d:%02d;%5.0f;%5.0f\n", h, m, p, c);
        if(!file.print(buf)) Serial.println("Append failed");
    }
}
file.close();
}

void loop(){}

```



	A	B	C
1	Heure	Production(W)	Consommation(W)
2	00:00	6978	3383
3	00:05	6300	3989
4	00:10	6143	7413
5	00:15	8592	5952
6	00:20	5733	2712
7	00:25	1442	8440
8	00:30	2364	3927
9	00:35	8450	6991
10	00:40	8807	1797
11	00:45	5222	8118
12	00:50	6130	2844
13	00:55	6051	6460
14	01:00	8141	5972
15	01:05	3929	4027
16	01:10	8597	9591
17	01:15	5933	1659
18	01:20	3701	4307
19	01:25	8366	2109
20	01:30	4653	437
21	01:35	1151	8383
22	01:40	9130	2306
23	01:45	3549	6265
24	01:50	2635	4500
25	01:55	2465	8145

...

## 4.2.9 Buffer circulaire

Pour éviter de perdre des données lorsque la carte SD n'est pas présente (retirée par exemple pour vider son contenu sur un PC), celles-ci seront d'abord systématiquement enregistrées dans un buffer circulaire de type FIFO et le contenu de ce dernier sera vidé à intervalles réguliers sur la carte SD, lorsque c'est possible.

La gestion de ce buffer (déclaration, méthodes Push, Pop et Get) sera implémentée dans une classe template, **RingBuffer<T, N>**, pour laquelle le type des données stockées et la taille du buffer doivent être précisés.

```

// RingBuffer.hpp
#pragma once

```

```

#include <Arduino.h>

//*********************************************************************
Class RingBuffer<T, N>
//*********************************************************************

    Circular Buffer (FIFO) data structure, with N elements of type T
*/



template <typename T, int N> class RingBuffer
{
public:
    // Constructor
    RingBuffer()
    {
        pStart = buffer;      // &buffer[0];
        pEnd = buffer + N-1; // &buffer[N-1];
        pHead = pStart;
        pTail = pStart;
        size = 0;
    }

    // Number of elements
    int Size() {return size;}

    // Operations
    void Push(T *t) // Before test if Size() < N
    {
        if (size == N) return;
        *pTail++ = *t;
        if (pTail > pEnd) pTail = pStart;
        size++;
    }

    void Pop(T *t) // Before test if Size() > 0
    {
        if (size == 0) return;
        *t = *pHead++;
        if (pHead > pEnd) pHead = pStart;
        size--;
    }

    void Get(T *t, int n) // Before test if 0 <= n <= Size()
    {
        if (n < 0 || n > size) return;
        T *p = pHead + n;
        if (p > pEnd) p -= N;
        *t = *p;
    }

private:
    T buffer[N];
    T *pHead, *pTail, *pStart, *pEnd;
    int size;
};

```

Dans le test ci-dessous, le buffer, de taille 2, est rempli avec des structures de type :

```

struct S
{
    String a;
    int b;
};

```

On le déclare donc dans l'instruction :

```

RingBuffer<S,2> rb;

// TestRingBuffer.ino

/*
    Test the RingBuffer class.
*/

```

```
#include <RingBuffer.hpp>
```

```

struct S
{
    String a;
    int b;
};

RingBuffer<S,2> rb;

void printRingBuffer(String title)
{
    Serial.println();
    Serial.println(title);
    S s;
    for (int i = 0; i < rb.Size(); i++)
    {
        rb.Get(&s, i);
        Serial.print("a="); Serial.print(s.a); Serial.print(" b="); Serial.println(s.b);
    }
}

void setup()
{
    Serial.begin(115200);
    S s = {"Hello", 0};

    rb.Push(&s);
    s.b = 1;
    rb.Push(&s);
    printRingBuffer("Ring buffer après 2 Push :");

    rb.Pop(&s);
    printRingBuffer("Ring buffer après 1 Pop :");

    s.b = 2;
    rb.Push(&s);
    printRingBuffer("Ring buffer après 1 nouveau Push :");

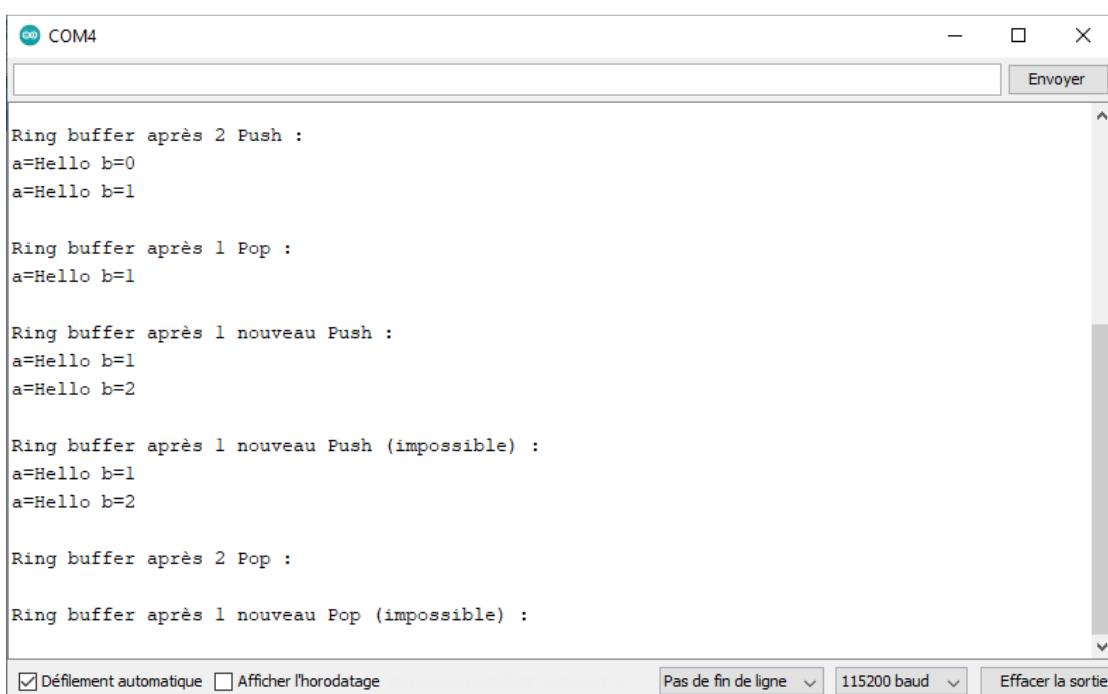
    rb.Push(&s);
    printRingBuffer("Ring buffer après 1 nouveau Push (impossible) :");

    rb.Pop(&s);
    rb.Pop(&s);
    printRingBuffer("Ring buffer après 2 Pop :");

    rb.Pop(&s);
    printRingBuffer("Ring buffer après 1 nouveau Pop (impossible) :");
}

void loop() {}

```



## 4.2.10 Boutons poussoir

Ils vont permettre de changer l'affichage des données sur l'écran OLED de la carte Heltec et de commander un reset : on va ici considérer 4 boutons poussoirs connectés respectivement aux broches n°13, 12, 14 et 27 de cette carte.

Pour résoudre ce problème, on utilise une bibliothèque **Mbutton**<sup>30</sup> réalisée pour un autre projet qui détecte des pressions successives, courtes ou longues, sur des boutons poussoirs, avec un traitement par interruptions. On notera qu'ici, une fonction de rappel (callback) static doit être déclarée et implémentée pour chaque bouton dans la bibliothèque ; mais, comme elles le sont toutes sur un même modèle, ce n'est pas bien difficile de le faire. Par exemple :

```
static void IRAM_ATTR buttonInterrupt12();

void IRAM_ATTR MButton::buttonInterrupt12()
{
    if (_okButton && !_toProcess)
    {
        _num = 12;
        _okButton = false;
        _timer.once_ms(DELAY_DEBOUNCE, timerInterrupt);
    }
}
```

Ceci étant fait, l'utilisation de cette bibliothèque est très simple :

```
// TestButtons.ino

/*
Test 4 push buttons connected to the pins number 13, 12, 14 and 27.
Can be long or short pressed.

Reference :
- https://github.com/jlemaire06/Esp32-async-multi-button-library
*/

#include <MButton.h>

MButton mButton;

void setup()
{
    Serial.begin(115200);
    while(!Serial);
    mButton.begin(4, 13, 12, 14, 27);
}

void loop()
{
    if (mButton.toProcess())
    {
        int num = mButton.getNum();
        int act = mButton.getAction();
        switch (num)
        {
            case 13:
                Serial.print("Button 13 ");
                break;
            case 12:
                Serial.print("Button 12 ");
                break;
            case 14:
                Serial.print("Button 14 ");
                break;
            case 27:
                Serial.print("Buttton 27 ");
                break;
        }
    }
}
```

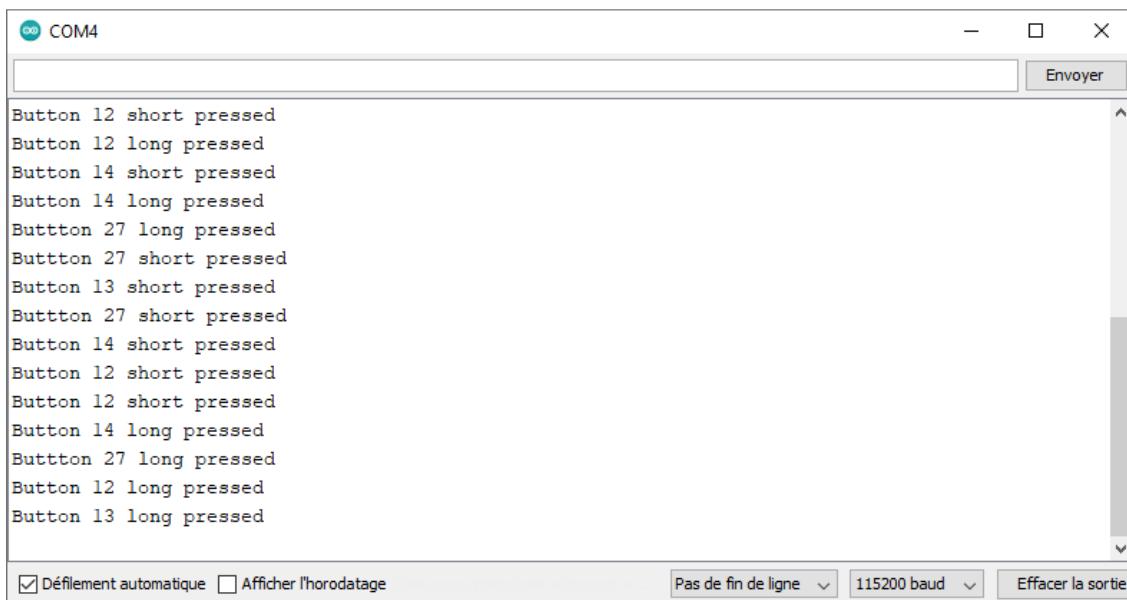
---

<sup>30</sup> <https://github.com/jlemaire06/Esp32-async-multi-button-library>

```

switch (act)
{
    case 1:
        Serial.println("short pressed");
        break;
    case 2:
        Serial.println("long pressed");
        break;
}
mButton.processed();
}
delay(500);
}

```



## 4.3 Sketch Solar Monitor

Il implémente les fonctionnalités suivantes :

- initialisation de la date et de l'heure courante dans le setup, celles-ci étant fournies par un serveur NTP accédé en Wifi ;
- calcul périodique des puissances (W) de la Production PV et de la Consommation, en accédant en Wifi à l'API Fronius Solar API ;
- estimation périodique de la tension de batterie, mesurée sur la broche 37 de la carte ;
- affichage périodique de toutes ces valeurs sur l'écran OLED de la carte ;
- calcul des puissances moyennes sur des intervalles de 5mn et enregistrement de celles-ci dans un buffer circulaire ;
- vidage périodique du buffer sur la carte SD, avec création d'un fichier de texte délimité pour chaque jour : JJ-MM-YYYY.csv ;
- clignotement de la led intégrée pour indiquer que la carte SD ne doit pas être enlevée ou insérée dans le lecteur ;
- traitement des pressions sur les 4 boutons poussoirs pour réaliser les actions suivantes :
  - reset de la carte (broche 13) ;
  - changement de l'affichage OLED, pour visualiser les énergies journalières (kWh) produites et consommées, estimées à partir des fichiers de la carte SD<sup>31</sup> ;

---

<sup>31</sup> Ces énergies journalières (kWh) sont estimées en divisant par 12000 la somme de toutes les puissances : on suppose donc qu'on dispose bien d'un enregistrement toutes les 5mn, soit 288 enregistrements au total.

- recul d'un jour ;
- avance d'un jour.

```
// SolarMonitor.ino

//*********************************************************************
Solar monitoring of a Fronius Primo GEN24 Plus inverter using an Heltec Wifi Kit 32 v2.1 card.

Functionnalities :
- initialization of the current date and time in the setup, these being supplied by an NTP server accessed via Wifi ;
- periodic calculation of the powers (W) of PV Production and Consumption, by accessing the Fronius Solar API via WiFi ;
- periodic estimation of the battery voltage (V), measured on pin 37 of the card ;
- periodic display of all these values on the board OLED screen ;
- calculation of the average powers over 5 min intervals and recording in a circular buffer ;
- periodic emptying of the buffer on the SD card, with creation of a delimited text file (.csv) for each day ;
- led blinking to indicate that the SD card must not be removed or inserted in its reader ;
- processing of pressures on the 4 push buttons to carry out the following actions :
  o card reset ;
  o change to the OLED display, to view the daily energy (kWh) produced and consumed, estimated from the SD card files ;
  o going backward one day ;
  o going forward one day.

References :
- https://riton-duino.blogspot.com/2021/06/esp32-heltec-wifi-kit-32.html
- https://github.com/adafruit/Adafruit\_SSD1306
- https://github.com/adafruit/Adafruit\_GFX-Library
- https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/
- https://github.com/arduino-libraries/Arduino\_JSON
- https://randomnerdtutorials.com/esp32-http-get-post-arduino/
- https://github.com/pfeerick/elapsedMillis
- https://randomnerdtutorials.com/esp32-date-time-ntp-client-server-arduino/
- https://sourceware.org/newlib/libc.html#Timefns
- https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/esp32-hal-time.c#L47
- https://RandomNerdTutorials.com/esp32-microsd-card-arduino/
- https://github.com/espressif/arduino-esp32/tree/master/libraries/SD
- https://github.com/jlemaire06/Esp32-async-multi-button-library
- https://github.com/pfeerick/elapsedMillis

***** */

//*********************************************************************
Libraries and structures
***** */

// OLED
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Fonts/FreeSerif9pt7b.h>

// Wifi
#include <WiFiClient.h>
#include <HTTPClient.h>
#include <Arduino_JSON.h>

// Time
#include <time.h>

// Ring buffer
#include <RingBuffer.hpp>

struct Record
{
    uint8_t day;           // Day
    uint8_t month;         // Month
    uint16_t year;         // Year
    uint8_t hour;          // Hour
    uint8_t minute;        // Minute
    uint16_t prod;         // Production(W)
    uint16_t cons;         // Consumption(W)
};

// SD
```

```

#include <SD.h>

// Timers
#include <elapsedMillis.h>

// Display modes
enum DisplayMode {POWER, ENERGY};

// Buttons
#include <MButton.h>

//*********************************************************************
// Constants
//********************************************************************/

// Screen
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define SCREEN_ADDRESS 0x3C

// SD card
#define CS_SD 5

// Wifi access point
const char *SSID = "Livebox-486c";
const char *PWD = "FAAD971F372AA5CCE13D25969E";

// Web service host address of the Fronius inverter
const char *HOST = "192.168.1.13";

// Battery
#define VBAT_PIN 37      // GPIO connected to VBAT in Heltec Wifi kit 32 V2.1
#define KV 1.7E-3         // Coefficient to convert in Volts the analog reads

// NTP Server
const char* NTP_SERVER = "pool.ntp.org"; // Server address
const long GMT_OFFSET = 3600;             // GMT+1 zone
const int DAY_LIGHT_OFFSET = 3600;        // Summer time

// Save frequency in 1 hour
#define SAVE_FREQUENCY 12                  // => 5mn period

// Update periods (ms)
#define RECORD_PERIOD 2000                // 2s
#define SAVE_PERIOD 300000                // 5mn
#define DISPLAY_PERIOD 1000               // 1s
#define LED_PERIOD 200                   // 200ms

// Led blinking max number (need to be odd to switch off the led at last)
#define MAX_LED_BLINK 11

// Button pins
#define RESET_PIN 12
#define MODE_PIN 27
#define BACKWARD_PIN 14
#define FORWARD_PIN 13

//*********************************************************************
// Global variables
//********************************************************************/

// SSD1306 display connected to I2C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire1, RST_OLED);

// Timers
elapsedMillis tmrRecord;
elapsedMillis tmrSave;
elapsedMillis tmrDisplay;
elapsedMillis tmrLed;

// Parameters
struct tm timeInfo;           // Current time
double batt;                  // battery voltage(V)
double prod;                  // Production power(W)
double cons;                  // Consumption power(W)

// Records

```

```

Record rec;           // Current record
double sumProd, sumCons; // Sums of production and consumption values
int nbData;           // Number of data in that sums
RingBuffer<Record, SAVE_FREQUENCY> rb; // => Can store 1 hour of records

// Led state
uint8_t ledState;      // LOW/HIGH
uint8_t ledBlink;       // Led blinking count

// Display
DisplayMode dispMode;

// Buttons
MButton mButton;

// Energy
uint8_t dayE;          // Day
uint8_t monthE;         // Month
uint16_t yearE;         // Year
double prodE;           // Production(kWh)
double consE;           // Consumption(kWh)
uint16_t iDay;          // Index of the day before

/********************* Functions *********************/
void connectToWiFi(const char * ssid, const char * pwd)
{
    // Wifi connection
    WiFi.begin(ssid, pwd);
    while (WiFi.status() != WL_CONNECTED) delay(500);
}

bool EnergyEval(const uint8_t _dayE, const uint8_t _monthE, const uint16_t _yearE, double *_prodE,
                double *_consE)
{
    // Energy evaluation using a file DD-MM-YYYY.csv on the SD card
    *_prodE = 0;
    *_consE = 0;
    if (!SD.begin(CS_SD)) return false;
    char buf[20];
    sprintf(buf, "%02d-%02d-%d.csv\0", _dayE, _monthE, _yearE);
    File file = SD.open(buf, FILE_READ);
    if (!file)
    {
        SD.end();
        return false;
    }
    file.readStringUntil('\n');
    file.readStringUntil('\n');
    file.readStringUntil('\n');
    String str;
    uint8_t i1, i2;
    while (file.available())
    {
        str = file.readStringUntil('\n');
        i1 = str.indexOf(';');
        i2 = str.indexOf(';', i1+1);
        *_prodE += (str.substring(i1+1, i2)).toInt();
        *_consE += (str.substring(i2+1)).toInt();
    }
    file.close();
    SD.end();
    *_prodE /= SAVE_FREQUENCY*1000;
    *_consE /= SAVE_FREQUENCY*1000;
    return true;
}

void changeDate(uint8_t *_day, uint8_t *_month, uint16_t *_year, long int _sec)
{
    // Change a date according an offset in seconds
    struct tm t;
    t.tm_sec = 0;
    t.tm_min = 0;
    t.tm_hour = 0;
    t.tm_mday = *_day;
    t.tm_mon = *_month - 1;
}

```

```

t.tm_year = *_year - 1900;
t.tm_wday = 0;
t.tm_yday = 0;
t.tm_isdst = 0;
time_t tt = mktime(&t) + _sec;
localtime_r(&tt, &t);
*_day = t.tm_mday;
*_month = t.tm_mon + 1;
*_year = t.tm_year + 1900;
}

void setup()
{
    // Init OLED
Wire1.begin(SDA_OLED, SCL_OLED);
if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) for(;;);
display.setTextColor(WHITE);

// Connect to the Wifi access point
connectToWiFi(SSID, PWD);

// Init time
configTime(GMT_OFFSET, DAY_LIGHT_OFFSET, NTP_SERVER);
while (!getLocalTime(&timeInfo)) delay(100);

// Init record
rec.day = timeInfo.tm_mday;
rec.month = timeInfo.tm_mon + 1;    // tm_mon = month, between 0 (January) and 11 (December).
rec.year = timeInfo.tm_year + 1900; // tm_year = year since 1900
rec.hour = timeInfo.tm_hour;
rec.minute = timeInfo.tm_min - timeInfo.tm_min%(60/SAVE_FREQUENCY);
sumProd = 0;
sumCons = 0;
nbData = 0;

// Init timer to immediately update
tmrRecord = RECORD_PERIOD;

// Initialize digital pin LED_BUILTIN as an output
pinMode(LED_BUILTIN, OUTPUT);

// Initialize the Led
pinMode(LED_BUILTIN, OUTPUT);
digitalWrite(LED_BUILTIN, LOW);
ledState = LOW;
ledBlink = MAX_LED_BLINK;

// Initialize display mode
dispMode = POWER;

// Buttons
mButton.begin(4, RESET_PIN, MODE_PIN, BACKWARD_PIN, FORWARD_PIN);
}

void loop()
{
    if (tmrRecord > RECORD_PERIOD)
    {
        // Reset timer
        tmrRecord = 0;

        // Get battery voltage
        batt = KV*analogRead(VBAT_PIN);

        // Record flag
        bool okRecord = true;

        // Get time
        okRecord = getLocalTime(&timeInfo);

        // Get Production and Consumption
        if(WiFi.status() == WL_CONNECTED)
        {
            // Local variables
            HTTPClient http;

            // Request Production
            http.begin("http://" + String(HOST) +
"/solar_api/v1/GetInverterRealtimeData.cgi?Scope=System");

```

```

    if(http.GET() == HTTP_CODE_OK) prod =
double(JSON.parse(http.getString())["Body"]["Data"]["PAC"]["Values"][1]);
    else okRecord = false;
http.end();

    // Request Consumption
http.begin("http://" + String(HOST) + "/solar_api/v1/GetMeterRealtimeData.cgi");
if(http.GET() == HTTP_CODE_OK) cons = prod +
double(JSON.parse(http.getString())["Body"]["Data"][0]["PowerReal_P_Phase_1"]);
    else okRecord = false;
http.end();
}
else connectToWiFi(SSID, PWD); // WiFi reconnect

    // Recording
if (okRecord)
{
    sumProd += prod;
    sumCons += cons;
    nbData++;
if ((timeInfo.tm_min%(60/SAVE_FREQUENCY) == 0) && (timeInfo.tm_min != rec.minute))
{
    // Means
    if (nbData > 0)
    {
        rec.prod = round(sumProd/nbData);
        rec.cons = round(sumCons/nbData);
    }
    else
    {
        rec.prod = 0;
        rec.cons = 0;
    }

    // Save record in the Ring Buffer
rb.Push(&rec);

    // Reinit record
rec.day = timeInfo.tm_mday;
rec.month = timeInfo.tm_mon + 1;
rec.year = timeInfo.tm_year + 1900;
rec.hour = timeInfo.tm_hour;
rec.minute = timeInfo.tm_min;
sumProd = 0;
sumCons = 0;
nbData = 0;
}
}
}

if (tmrSave > SAVE_PERIOD)
{
    // Reset timer
tmrSave = 0;

    // Turn the LED on
digitalWrite(LED_BUILTIN, HIGH);
ledState = HIGH;
ledBlink = 0;
tmrLed = 0;

    // Save to SD card
if (SD.begin(CS_SD) && (rb.Size() > 0)) // SD card mounted and any record(s) to save
{
    // Save to SD
    bool okSave = true;
    while (okSave && (rb.Size() > 0))
    {
        // Update okSave
        okSave = false;

        // Read the record to save in the ring buffer
        Record r;
        rb.Get(&r, 0);

        // File name
        char buf[41];
        sprintf(buf, "%02d-%02d-%d.csv", r.day, r.month, r.year);

```

```

// Saving
File file;
if (SD.exists(buf))
{
    file = SD.open(buf, FILE_APPEND);
    if (file) // The file exists
    {
        sprintf(buf, "%02d:%02d;%d\n", r.hour,r.minute, r.prod, r.cons);
        file.print(buf);
        file.close();
        okSave = true;
    }
}
else // The file doesn't exist, then create it
{
    file = SD.open(buf, FILE_WRITE);
    if (file)
    {
        sprintf(buf, "%02d/%02d/%d;\n;;\n", r.day, r.month, r.year);
        file.print(buf);
        sprintf(buf, "%s;%s;%s\n", "Heure", "Production(W)", "Consommation(W)");
        file.print(buf);
        sprintf(buf, "%02d:%02d;%d;%d\n", r.hour,r.minute, r.prod, r.cons);
        file.print(buf);
        file.close();
        okSave = true;
    }
}

// Pop the ring buffer if okSave
if (okSave) rb.Pop(&r);
}

// To enable ejection
SD.end();
}

if (tmrDisplay > DISPLAY_PERIOD)
{
    // Reset timer
    tmrDisplay = 0;

    // OLED display
    display.clearDisplay();
    display.setFont();
    display.drawRect(93,0, 32, 13, 1);
    display.setCursor(97, 3);
    display.print(batt, 1);
    display.println('V');
    char buf[15];
    switch (dispMode)
    {
        case POWER:
            strftime(buf,15, "%d/%m/%y %H:%M", &timeInfo);
            display.setCursor(0, 3);
            display.print(buf);
            display.setFont(&FreeSerif9pt7b);
            display.setCursor(0, 35);
            display.print("Prod ");
            display.setCursor(40, 35);
            display.print(prod, 0); // No decimal
            display.println(" W");
            display.setCursor(0, 55);
            display.print("Cons ");
            display.setCursor(40, 55);
            display.print(cons, 0); // No decimal
            display.println(" W");
            break;
        case ENERGY:
            sprintf(buf, "%02d/%02d/%d", dayE, monthE, yearE);
            display.setCursor(0, 3);
            display.print(buf);
            display.setFont(&FreeSerif9pt7b);
            display.setCursor(0, 35);
            display.print("Prod ");
            display.setCursor(40, 35);
            display.print(prodeE, 1); // 1 decimal
            display.println(" kWh");
    }
}

```

```

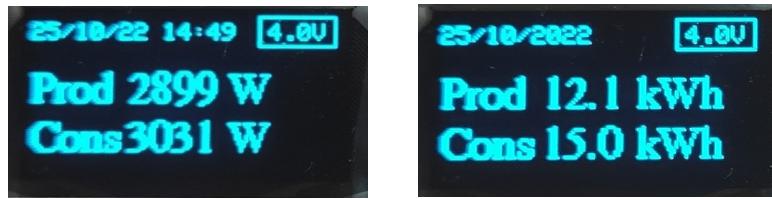
        display.setCursor(0, 55);
        display.print("Cons ");
        display.setCursor(40, 55);
        display.print(consE, 1); // 1 decimal
        display.println(" kWh");
        break;
    }
    display.display();
}

if ((ledBlink < MAX_LED_BLINK) && (tmrLed > LED_PERIOD))
{
    // Reset timer and update the led blinking count
    tmrLed = 0;
    ledBlink++;

    // Switch on/off the LED
    if (ledState == HIGH)
    {
        digitalWrite(LED_BUILTIN, LOW);
        ledState = LOW;
    }
    else
    {
        digitalWrite(LED_BUILTIN, HIGH);
        ledState = HIGH;
    }
}

if (mButton.toProcess())
{
    switch (mButton.getNum())
    {
        case RESET_PIN: // Reset
            ESP.restart();
            break;
        case MODE_PIN: // Display mode
            switch (dispMode)
            {
                case POWER:
                    dispMode = ENERGY;
                    dayE = timeInfo.tm_mday;
                    monthE = timeInfo.tm_mon + 1;
                    yearE = timeInfo.tm_year + 1900;
                    EnergyEval(dayE, monthE, yearE, &prodE, &consE);
                    iDay = 0;
                    break;
                case ENERGY:
                    dispMode = POWER;
                    break;
                }
            break;
        case BACKWARD_PIN: // Going back a day
            if (dispMode == ENERGY)
            {
                changeDate(&dayE, &monthE, &yearE, -86400);
                EnergyEval(dayE, monthE, yearE, &prodE, &consE);
                iDay++;
            }
            break;
        case FORWARD_PIN: // Advance one day
            if ((dispMode == ENERGY) && (iDay > 0))
            {
                changeDate(&dayE, &monthE, &yearE, +86400);
                EnergyEval(dayE, monthE, yearE, &prodE, &consE);
                iDay--;
            }
            break;
    }
    mButton.processed();
}
}

```



11-10-2022.csv  
 12-10-2022.csv  
 13-10-2022.csv  
 14-10-2022.csv  
 15-10-2022.csv  
 16-10-2022.csv  
 17-10-2022.csv  
 18-10-2022.csv  
 19-10-2022.csv  
 20-10-2022.csv  
 21-10-2022.csv  
 22-10-2022.csv  
 23-10-2022.csv  
 24-10-2022.csv  
 25-10-2022.csv

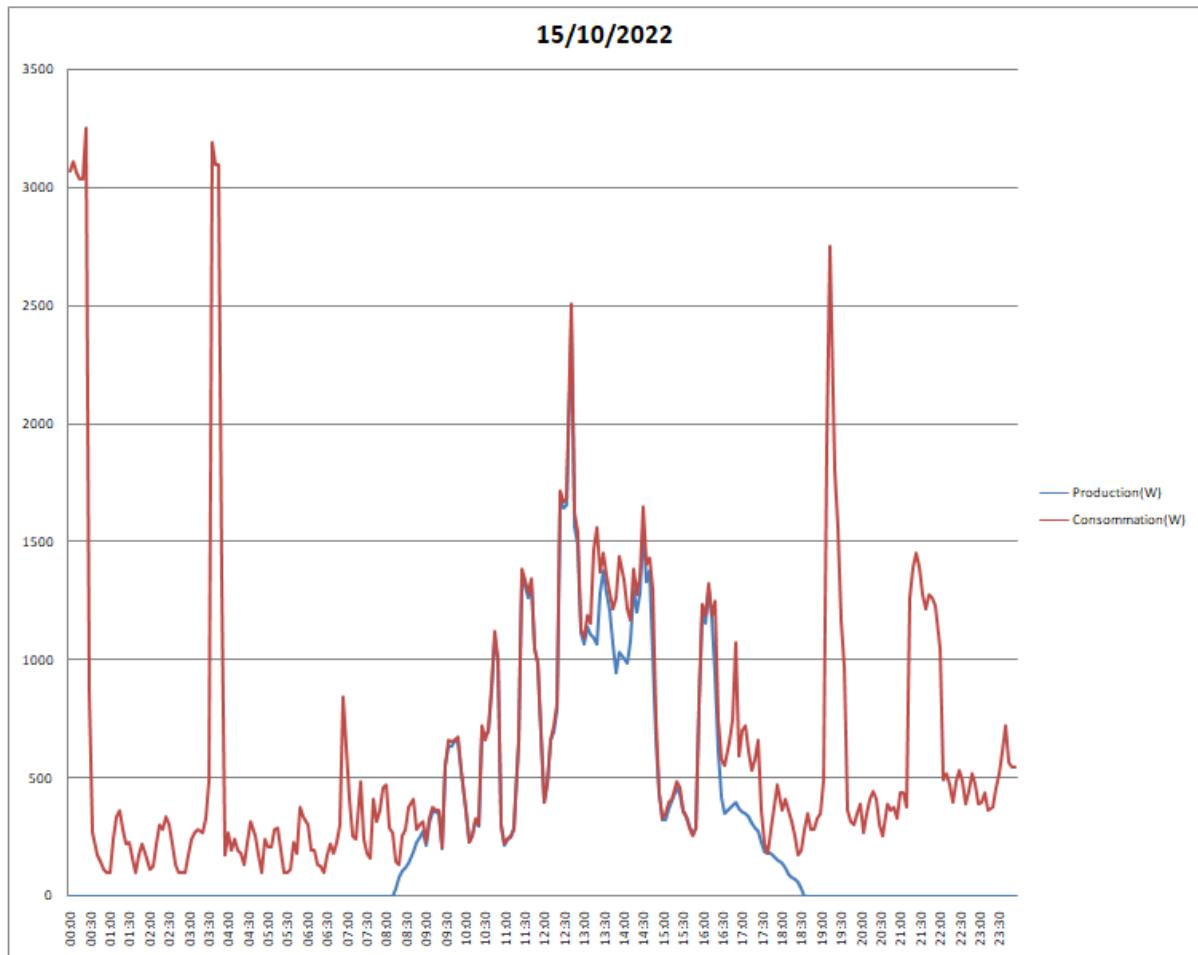
	A	B	C
1	15/10/2022		
2			
3	Heure	Production(W)	Consommation(W)
4	00:00	0	3072
5	00:05	0	3108
6	00:10	0	3060
7	00:15	0	3036
8	00:20	0	3036
9	00:25	0	3252
10	00:30	0	864
11	00:35	0	264
12	00:40	0	168
13	00:45	0	144
14	00:50	0	108
15	00:55	0	96
16	01:00	0	96
17	01:05	0	240
18	01:10	0	336
19	01:15	0	360
20	01:20	0	276
21	01:25	0	216
22	01:30	0	228
23	01:35	0	156
24	01:40	0	96
25	01:45	0	168

...

280	23:00	0	384
281	23:05	0	396
282	23:10	0	432
283	23:15	0	360
284	23:20	0	372
285	23:25	0	456
286	23:30	0	516
287	23:35	0	612
288	23:40	0	720
289	23:45	0	564
290	23:50	0	540
291	23:55	0	540

On peut stocker de nombreux fichiers de ce type sur la carte SD<sup>32</sup>.

On peut aussi les ouvrir dans Excel et tracer des graphiques, comme pour les données issues de SOLAR.WEB :



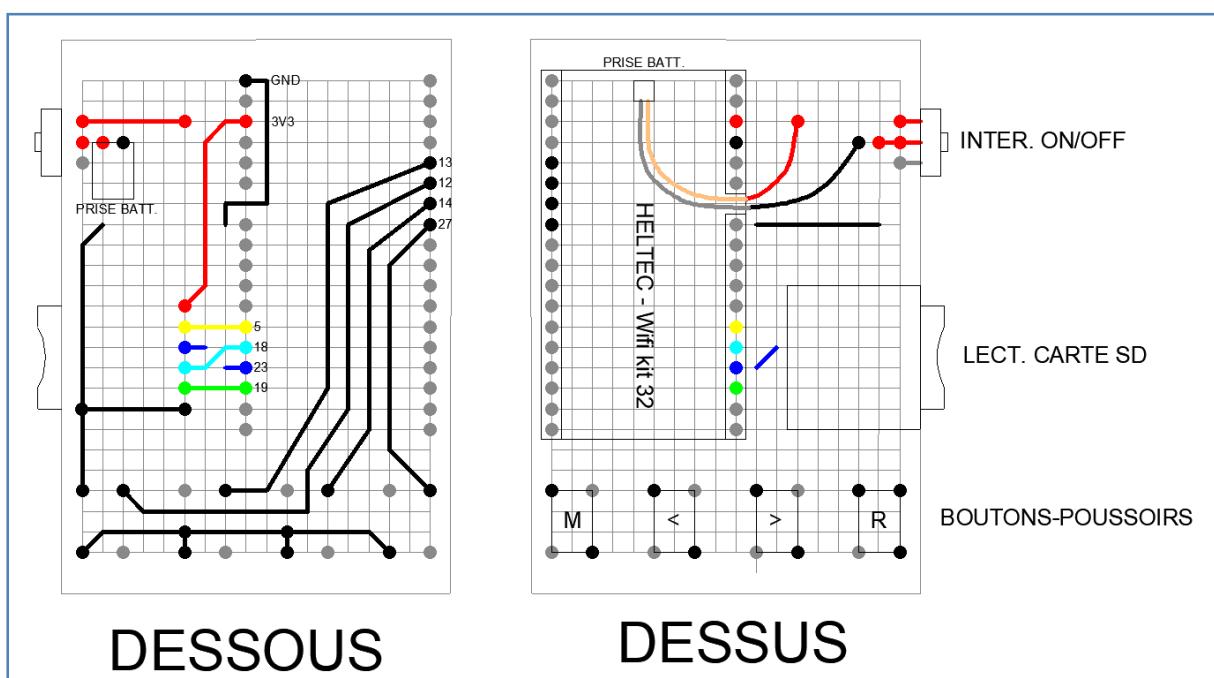
<sup>32</sup> Plus de 3000 sur une carte SD de 32GB.

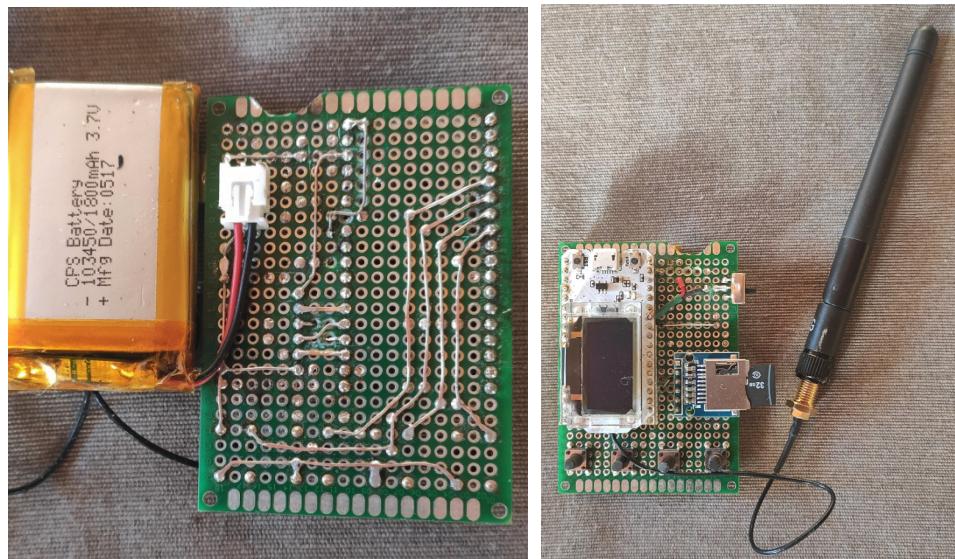
## 5 Implémentation



### 5.1 Carte

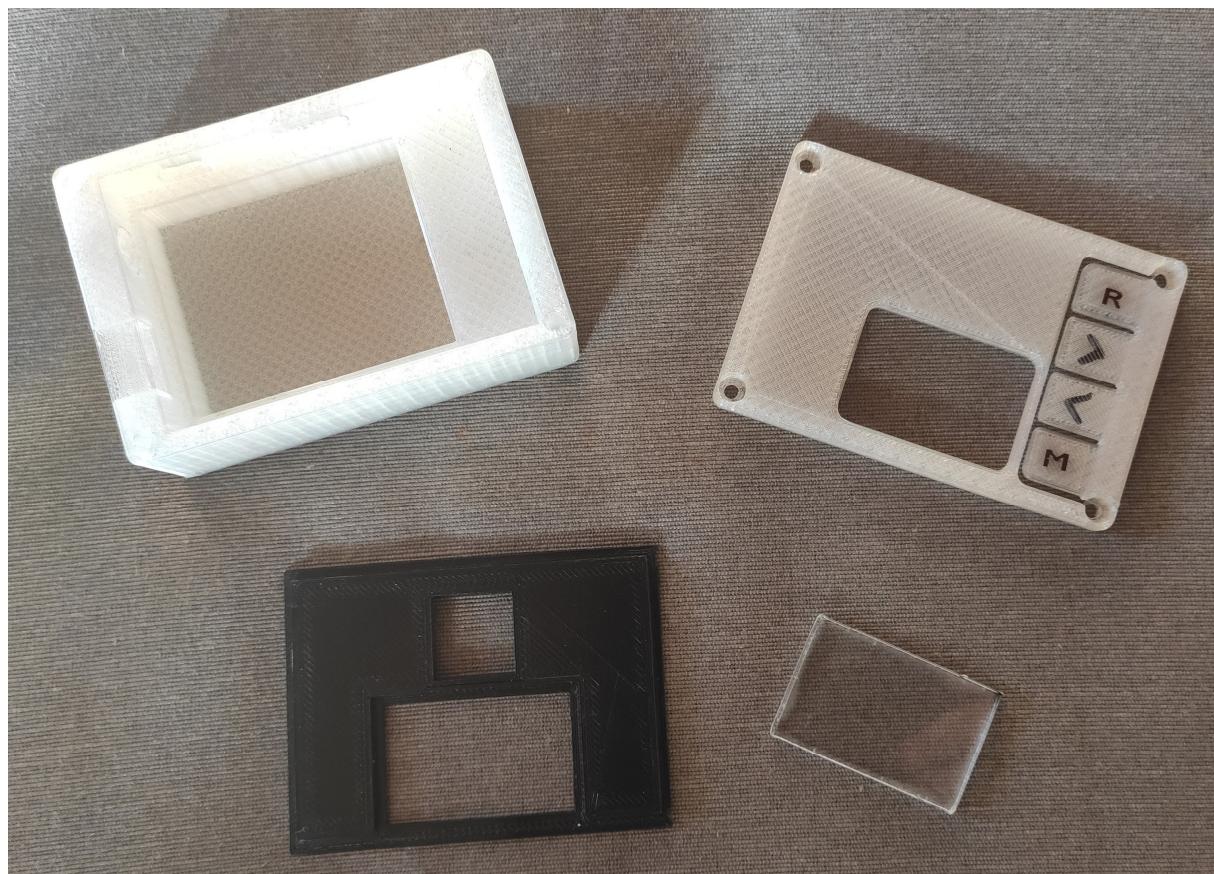
Elle a été réalisée en utilisant une plaque d'essais double face à pastilles rondes étamées au pas de 2,54mm<sup>33</sup>.





## 5.2 Boitier

Il a été réalisé par Pierre NUNEZ, avec une imprimante 3D :



## 6 Conclusion

Ce petit superviseur solaire « fait le job » : il affiche en temps réel sur un écran OLED la puissance électrique AC, produite par l'onduleur de chaîne ainsi que celle de la consommation domestique, il enregistre périodiquement (5mn) la moyenne de ces puissances sur une carte SD en créant un fichier journalier lisible dans Excel et il peut également afficher les énergies journalières produites et consommées, en exploitant ces fichiers.

Mais surtout, il a permis de tester un certain nombre de techniques utiles, en vue de la réalisation d'un routeur statique fonctionnant en mode « 0-injection » : accès Wifi à SOLAR API exposée ici par un onduleur Fronius de type GEN24, initialisation de la date et de l'heure courante via un serveur NTP également accédé en Wifi, usage d'une batterie comme alimentation de secours notamment.

Ici le choix d'un microcontrôleur à base d'ESP32 doté d'un écran OLED s'est révélé pertinent à tous points de vue : facilité de programmation avec l'IDE Arduino, rapidité, capacités mémoire, fonctionnalités wifi. Plusieurs cartes de développement possèdent ces caractéristiques ; le Wifi kit 32 d'Heltec en est une, avec un bon rapport qualité/prix.

Evidemment, comme tout prototype, ce superviseur solaire aurait besoin d'être testé d'avantage sur la durée. Et il pourrait être bien sûr amélioré : écran plus grand pour une meilleure lisibilité et permettre l'affichage d'autres paramètres, prise en compte des données manquantes dans le calcul des énergies, mode économie d'énergie pour augmenter l'autonomie du fonctionnement sur batterie, etc.

Avec tous les outils fournis, les nombreux tutoriaux et bibliothèques existant pour l'ESP32, ce ne devrait pas être bien difficile !