Le programme

Etude Décembre 2020 - Thierry Picquart -

- pour contacter Thierry Picquart

La programmation est faite avec l'**IDE Arduino** sur un PC raccordé en USB sur l'**ESP32**. J'ai donc découvert l'**ESP32**, l'**IDE Arduino** et le langage C dans le cadre de ce projet. Dans cette page, je vous présente les codes fonctionnels par thème. Ces différents codes seront ensuite concaténés et organisés dans le programme final. Certains thèmes sont simples à traiter car bien documentés, d'autres ont été complexes du fait de manque d'informations ou de codes non fonctionnels.

La température

Une résistance de **4,7kOhms** entre le Plus et la broche data branchée sur **GPIO 4**

Lecture de la température 18DB20

```
#include <OneWire.h>
#include <DallasTemperature.h>

const int oneWireBus = 4;     //GPIO 4

OneWire oneWire(oneWireBus);
DallasTemperature sensors(&oneWire);

void setup(){
sensors.begin();
}
void loop () {
sensors.requestTemperatures();
  float temperatureC = sensors.getTempCByIndex(0);
}
```

L'humidité

Le **DHT11** est branché au + et au moins. La patte data sur la broche **GPIO 14** Une résistance de pullup de **10kOhms** entre data et +

Lecture de l'humidité DHT11

```
#include <DHT.h>

#define DHTPIN 14     // Branchement sur GPIO 14
```

```
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE);

void setup () {
dht.begin();
}
void loop() {
float h = dht.readHumidity();   //humidité
float t = dht.readTemperature();//température
}
```

La balance

Le module **HX711** est connecté au + et au -, ainsi qu'aux broches 5 et 2 de l'**ESP32**. Il y a 2 paramètres à positionner: la **Calibration** et la **Tare** Chaque montage avec 4 pesons à besoin d'être calibré. Vous trouverez la méthodologie simple sur le net. Le principe étant de noter le poids à vide, de peser un poids connu, de diviser pour trouver le coefficient multiplicateur “**scale**”. La tare n'est pas obligatoire dans mon projet car c'est la variation de poids qui m'intéresse, mais vous pouvez la renseigner.

Lecture du poids HX711

```
#include "HX711.h"
const int broche_DT = 5;
const int broche_SCK = 2;
HX711 balance;
void setup(){
  balance.begin(broche_DT, broche_SCK);
  balance.set_scale(**11.7**); //calibration: le paramètre dépend de
votre cellule de charge.
  balance.tare(); //ajustement du zéro
}
void loop(){
balance.get_units(10);  //lecture du poids en grammes
}
```

Le mode Veille

Le mode veille est déclenché par esp_deep_sleep_start L'ESP est réveillé par son RTC et redémarre **Attention**, la loop n'est jamais exécutée, il faudra donc organiser notre programme final en conséquence.

Mise en veille

```
#define uS_TO_S_FACTOR 1000000  //* Conversion factor de micro seconds
en secondes
#define TIME_TO_SLEEP  5        //* Temps de veille de l'ESP32  (en
secondes)
void setup(){
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
esp_deep_sleep_start();  // démarrage de la veille
}
void loop(){
//  La loop n'est jamais exécuté, il faut mettre le code dans le setup
}
```

ESP-NOW

Les données seront transmise à l'**ESP32 maître** par le protocole **ESP-NOW**.

Il faut connaitre l'adresse MAC du maître

ESP-NOW récupération adresse MAC du maître

```
#include <WiFi.h>
void setup(){
Serial.begin(115200);
  Serial.print("ESP Board MAC Address:  ");
  Serial.println(WiFi.macAddress());
}
```

Principes du code. Dans cet exemple la structure des données n'étant pas encore définis définitivement, le code sera adapté.

ESP-NOW coté Esclave

```
#include <esp_now.h>
#include <WiFi.h>

uint8_t broadcastAddress[] = {0x30, 0xAE, 0xA4, 0xDF, 0x5A, 0x54}; //
Adresse MAC du maître

typedef struct struct_message {  //structure de la table des données à
envoyer (définition)
  char a[32];
  int b;
  float c;
  String d;
  bool e;
} struct_message;
```

```
struct_message myData; //création de la table


// callback Résultat de l_envoi
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
{
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}

void setup(){
  // Init Serial Monitor
  Serial.begin(115200);
    delay(2000);

  WiFi.mode(WIFI_STA); // Mode Wifi Station

  if (esp_now_init() != ESP_OK) {                  // initialisation
ESP-NOW
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  esp_now_register_send_cb(OnDataSent);            //redirection quand
réception

  // Register peer
  esp_now_peer_info_t peerInfo;
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
}

}

void loop (){
//préparation des données
       esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)
&myData, sizeof(myData));// envoi des données
}
```

Code Ruche V01

Code ruche version 1:

1. mesure des capteurs
2. envoi des données au maître
3. paramétrage en dur

Ce code est opérationnel mais nécessite un téléversement avec les paramètres différents par ruche. (décembre 2020)

Menu

```
#include <esp_now.h>
#include <WiFi.h>
#include <HX711.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <DHT.h>

// REPLACE WITH THE RECEIVER S MAC Address
uint8_t broadcastAddress[] = {0x30, 0xAE, 0xA4, 0xDF, 0x5A, 0x54};

const int broche_DT = 5;
const int broche_SCK = 2;
// GPIO where the DS18B20 is connected to
const int oneWireBus = 4;
#define DHTPIN 14     // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11   // DHT 11

#define uS_TO_S_FACTOR 1000000  /* Conversion factor for micro seconds
to seconds */
#define TIME_TO_SLEEP  900        /* Time ESP32 will go to sleep (in
seconds) */

// Structure example to send data
// Must match the receiver structure
typedef struct struct_message {
  char a[32];
  int ID;
  float Poids;
  float Temp1;
  float Temp2;
  float Humidite;
} struct_message;

// Create a struct_message called myData
struct_message myData;

HX711 balance;
```

```
// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
{
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}

// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);
// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

DHT dht(DHTPIN, DHTTYPE);

void setup() {

  // Init Serial Monitor
  Serial.begin(115200);
  delay(2000);
  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);
  Serial.println ("Slave");
  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);

  // Register peer
  esp_now_peer_info_t peerInfo;
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
   Serial.println("Initialisation de la balance...");
  balance.begin(broche_DT, broche_SCK);
  while (!balance.is_ready())
  {
    ;
```

```
  }
  balance.set_scale(11.7); //calibration: le paramètre dépend de votre
cellule de charge.
  balance.tare(); //ajustement du zéro
  Serial.println("La balance est prete!");

  // Start the DS18B20 sensor
  sensors.begin();
  dht.begin();
  delay(1000);
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);

       delay(2000);
        sensors.requestTemperatures();
        // Set values to send

        strcpy(myData.a, "Ruche 1");
        myData.ID = 1;
        myData.Poids=balance.get_units(10);
        myData.Temp1 = sensors.getTempCByIndex(0);
        myData.Temp2 = dht.readTemperature();
        myData.Humidite = dht.readHumidity();

      Serial.println(myData.Humidite);

        // Send message via ESP-NOW
        esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)
&myData, sizeof(myData));

        if (result == ESP_OK) {
          Serial.println("Sent with success");
        }
        else {
          Serial.println("Error sending the data");
        }


//Serial.println("Configured all RTC Peripherals to be powered down in
sleep");
esp_deep_sleep_start();
}

void loop() {

}
```

Paramétrage

Le paramétrage de l'ESP32 se fera par bluetooth en mode ligne. Il sera possible de paramétrer:

Menu

```
>--------------------
Bonjour HappyCulteur
>--------------------
>Commandes:
>nom XXXXXX sans espaces et 16 max
>id XX
>mac  XX XX XX XX XX XX
>calibrage
>tare
>data lecture de toutes les valeurs
>eeprom tableau eeprom
>--------------------
```

Bluetooth

Démarrage du bluetooth en mode liaison série avec le nom que vous souhaitez vous devrez appairer ce nom dans le menu bluetooth de votre smartphone et télécharger une application bluetooth serial Attention, cela ne fonctionne pas avec les iPhone et iPad. il faut donc aller au rucher avec un smartphone ou une tablette android.

Bluetooth

```
#include "BluetoothSerial.h"
BluetoothSerial SerialBT;    //initialisation du bluetooh en serial
SerialBT.begin("CONFIGRUCHE"); //Bluetooth device name   // mettre un
nom


SerialBT.write   // pour envoyer des caractères
SerialBT.read    // pour lire des caractères
```

Organisation du code

A ce stade, le code devient compliqué et long ce qui est peu ergonomique au niveau de l'IDE Arduino J'ai donc décidé d'utiliser les onglets afin de saucissonner et d' organiser le code

Du coup je vais ajouter:

- Un onglet de documentation du code en mode commentaire
- Un onglet pour les variables
- Un onglet pour le gros morceau Commander
- Un onglet pour les releases notes

J'ai mis dans l'onglet variables.h toutes les variables du projet (18DB20, DHT11, la balance, l'Eeprom, etc.) et je fais un #include “variables.h” dans l'onglet principal qui contiendra finalement que le void setup() et le void loop() vide (à cause du sleep.

Commander

Pour faciliter la composition des menus en mode texte pour avoir un paramétrage simple et efficace, j'ai choisi “**Commander**”. Lien vers le Wiki Commander Au niveau du code, on a le code principale de notre projet dans lequel on configure Commander sur une liaison série ou bluetooth série. Les menus et les commandes se font dans l'onglet Commander. Il faudra donc adapter cet onglet pour notre besoin

Commander

```
#include <Commander.h> // la librairie
Commander cmd;          // initialisation
void setup(){
initialiseCommander(); // dans l_onglet commander: initialisation de
Commander en mode serial ou BTserial
}
void loop(){
cmd.update();
}
```

Utilisation de l'EEPROM

Dans cette fenêtre vous trouverez les codes nécessaires et qui fonctionnent pour un ESP32. Il y a beaucoup d'informations sur internet qui ne fonctionnent pas pour un ESP32.

- Write et read pour écrire et lire un octet.
- put et get pour écrire et lire des variables typées.

- update n'est pas implémenté pour l'ESP32 (write et put suffisent).

Nombre d'octets selon le type de variable (sizeof(variable) permet de le donner)

- CHAR: 1 octet
- BYTE: 1 octet
- INT: 2 octets ou 4 octets
- DOUBLE: 4 ou 8 octets
- FLOAT: 4 octets
- LONG: 4 octets
- SHORT: 2 octets

utilisation EEPROM

```
#include <EEPROM.h>                        // librairie
#define EEPROM_SIZE 512                    // taille de l_eeprom
souhaitée

char read_eeprom[21];
EEPROM.get(adNomRuche,read_eeprom);        //lecture d_une chaine de 21
caractères à partir de l_adresse adNomRuche depuis l_EEPROM



EEPROM.writeString(adNomRuche, myStr);   //ecriture d_une chaine de
caractère à l_adresse adNomRuche dans l_EEPROM
EEPROM.commit();                           // appliquer l_écriture

byte read_eeprom;
EEPROM.get(adIDRuche,read_eeprom);         //lecture d_un octet à
l_adresse adID depuisl_EEPROM

EEPROM.put(adIDRuche, myStr.toInt());    //ecriture d_un octet à
l_adresse adID dans l_EEPROM
EEPROM.commit();                           // appliquer l_écriture
```

Dans les codes suivants les procédure sont appelées par Commander, Elles utilisent quelques variables dans variables.h

lecture tableau de l'EEPROM

```
bool eepromHandler(Commander &Cmdr){                              //
affichage eeprom
byte read_eeprom;
 adresse = 0 ;
 ligne =0;
 compt = 0;
 compt1 = 19 ;
```

```
 Cmdr.println("    ");
   Cmdr.print("      ");
   // Preparation du comptage des colonnes de 0 a 9
  for ( compt =0; compt <= 9 ; compt++)
  {
  Cmdr.print(" ");
   Cmdr.print(compt);
    Cmdr.print("  ");
  }
   // Preparation du comptage des colonnes de 10 a 20
  for ( compt = 10 ; compt <= 19 ; compt++)
  {
  Cmdr.print(" ");
   Cmdr.print(compt);
    Cmdr.print(" ");
  }
   // Position du numero 0
  Cmdr.println("");
  Cmdr.print("  ");
   Cmdr.print(adresse);
    Cmdr.print("   ");



  for ( adresse =0; adresse <= 511 ; adresse++)
  {
    // commence à lire le premier octet (adresse 0) de la mémoire
EEPROM
    // conversion au format int pour affichage valeur numérique


  EEPROM.get(adresse,read_eeprom);
if (read_eeprom < 10) {
    Cmdr.print("  ");
    Cmdr.print(read_eeprom);
  }
else if (read_eeprom < 100) {
    Cmdr.print(" ");
    Cmdr.print(read_eeprom);
}
else {
  Cmdr.print(read_eeprom);
}
    Cmdr.print(" ");
    delay (5);
    if (( adresse == compt1)&(adresse <= 99))
      {
        Cmdr.println("");
        Cmdr.print(" ");
        Cmdr.print(adresse);
```

```
        Cmdr.print("   ");
        compt1 =compt1+20 ;
      }
    // affichage des adresses au-dessus de 99
    else if (adresse == compt1)
        {
        Cmdr.println("");
        Cmdr.print(" ");
        Cmdr.print(adresse);
        Cmdr.print("  ");
        compt1 =compt1+20 ;
        }

  }
 Cmdr.println("    ");

  return 0;
}
```

Ce code permet de vérifier que ce que l'on met en EEPROM est bien dedans et sous le bon format souhaité. Le résultat:

Lecture tableau de l'EEPROM

```
       0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
16  17  18  19
  0   108  97  98 101 108 108 101 114 117  99 104 101   0 106 100 107
1  14   0   0
 19     0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 39   255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 59   255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 79   255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 99   255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 119  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 139  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 159  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 179  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 199  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 219  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

```
255 255 255 255
 239  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 259  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 279  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 299  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 319  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 339  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 359  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 379  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 399  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 419  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 439  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 459  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 479  255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255
 499  255 255 255 255 255 255 255 255 255 255 255 255
```

Ecriture du nom de la ruche

```
bool nomHandler(Commander &Cmdr){
//Changement du nom

String myStr = "";
char read_eeprom[21];
//byte read_eeprom;
  Cmdr.println(">-------------------- ");
  Cmdr.println(">  Changement nom ");
  EEPROM.get(adNomRuche,read_eeprom);
  Cmdr.print(">  ancien nom : ");
  Cmdr.println(read_eeprom);
 // Cmdr.print("  Longueur : ");
//  Cmdr.println(read_eeprom.length());
  Cmdr.println(">-------------------- ");

  cmd.getString(myStr);
  Cmdr.print(">nom reçu : ");
  Cmdr.println(myStr);
```

```
     //On claque en EEPROM
     EEPROM.writeString(adNomRuche, myStr);
     EEPROM.commit();
     Cmdr.println(">Enregistré ");
     Cmdr.println(">-------------------- ");
     Cmdr.println("");
  return 0;
}
```

Ecriture de l'ID de la ruche

```
bool IdHandler(Commander &Cmdr){                //Changement ID
String myStr = "";
byte read_eeprom;
  Cmdr.println(">-------------------- ");
  Cmdr.println(">  Changement ID ");
  EEPROM.get(adIDRuche,read_eeprom);
  Cmdr.print(">  ancien ID : ");
  Cmdr.println(read_eeprom);
  Cmdr.println(">-------------------- ");
  ;
  cmd.getString(myStr);
  Cmdr.print(">Id reçu: ");
  Cmdr.println(myStr);


     //On claque en EEPROM
    EEPROM.put(adIDRuche, myStr.toInt());
    EEPROM.commit();
      Cmdr.println(">Enregistré ");

  Cmdr.println(">-------------------- ");
  Cmdr.println("");
  return 0;
}
```

Pour la MacAdress, c'est plus compliqué. on la rentre en Hexadécimal du genre FF et il faut convertir en octet pour rentrer dans l'EEPROM

Conversion Caractères HEX en octet

```
String myStr="";
cmd.getString(myStr);
cmd.println(myStr);
char paramChar[myStr.length()+1];
myStr.toCharArray(paramChar,myStr.length()+1);
int val = (int)strtol(paramChar,NULL,16);
```

la programmation se fera comme cela: mac 30 40 50 AF EF FF (valeurs en hexa)

Ecriture de l'adresse Mac du maitre dans l'EEPROM

```
String myStr="";
cmd.getString(myStr);                // MAC1
char paramChar[myStr.length()+1];
myStr.toCharArray(paramChar,myStr.length()+1);
int val = (int)strtol(paramChar,NULL,16);
EEPROM.put(adMac, val);
EEPROM.commit();
Cmdr.print(val);
 cmd.getString(myStr);                  //Next MAC2
 paramChar[myStr.length()+1];
 myStr.toCharArray(paramChar,myStr.length()+1);
 val = (int)strtol(paramChar,NULL,16);
 EEPROM.put(adMac+1, val);
 Cmdr.print(" ");
 Cmdr.print(val);
 cmd.getString(myStr);                  //Next MAC3
 paramChar[myStr.length()+1];
 myStr.toCharArray(paramChar,myStr.length()+1);
 val = (int)strtol(paramChar,NULL,16);
 EEPROM.put(adMac+2, val);
 Cmdr.print(" ");
 Cmdr.print(val);
 cmd.getString(myStr);                  //Next MAC4
 paramChar[myStr.length()+1];
 myStr.toCharArray(paramChar,myStr.length()+1);
 val = (int)strtol(paramChar,NULL,16);
 EEPROM.put(adMac+3, val);
 Cmdr.print(" ");
 Cmdr.print(val);
  cmd.getString(myStr);                  //Next MAC5
 paramChar[myStr.length()+1];
 myStr.toCharArray(paramChar,myStr.length()+1);
 val = (int)strtol(paramChar,NULL,16);
 EEPROM.put(adMac+4, val);
 Cmdr.print(" ");
 Cmdr.print(val);
 cmd.getString(myStr);                  //Next MAC6
 paramChar[myStr.length()+1];
 myStr.toCharArray(paramChar,myStr.length()+1);
 val = (int)strtol(paramChar,NULL,16);
 EEPROM.put(adMac+5, val);
 Cmdr.print(" ");
 Cmdr.println(val);
Cmdr.println(">-------------------- ");
EEPROM.commit();
Cmdr.println("");
return 0;
```

```
}
```

Passage en paramétrage

Le module **ESP32** est en veille quasi permanente. Il faut donc convenir du passage en paramétrage. J'ai choisi de démarrer le paramétrage avec un fil sur le GPIO15 (T3)de l'ESP32 grâce à la fonctionnalité “Touch Pad”. Sur simple touché du fil, l'ESP32 sort de son sommeil et passe en mode programmation par Bluetooth serial.

Voici le principe de la gestion du mode veille, du mode paramétrage et du mode mesures:

Passage en paramétrage

```
//Define touch sensitivity. Greater the value, more the sensitivity.
#define Threshold 40
#define uS_TO_S_FACTOR 1000000  /* Conversion factor for micro seconds
to seconds */
#define TIME_TO_SLEEP  10        /* Time ESP32 will go to sleep (in
seconds) */

touch_pad_t touchPin;

void callback(){
  //placeholder callback function: on arrive ici quand on est pas en
mode sleep
  Serial.println ("-------------------------------------callback");
}

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

 //débranchement
 touch_pad_t pin;
 touchPin = esp_sleep_get_touchpad_wakeup_status();
 switch(touchPin)
  {
    case 3  : configuration(); break;
    default : mesures(); break;
  }
}

void configuration(){
  Serial.println("-------------Configuration");
      //Go to sleep now

         //Setup interrupt on Touch Pad 3 (GPIO15)
touchAttachInterrupt(T3, callback, Threshold);
```

```
  //Configure Touchpad as wakeup source
esp_sleep_enable_touchpad_wakeup();
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);

  Serial.println ("Configuration: go to sleep");
  esp_deep_sleep_start();}
void mesures(){
  Serial.println("-------------Mesures");
    //Setup interrupt on Touch Pad 3 (GPIO15)
touchAttachInterrupt(T3, callback, Threshold);
  //Configure Touchpad as wakeup source
esp_sleep_enable_touchpad_wakeup();
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
//Go to sleep now
  Serial.println ("Mesures: go to sleep");
  esp_deep_sleep_start();

  }
void loop(){
  Serial.println("-->loop");
  delay (5000);
  Serial.println("<--loop");
}
```

E la fin des mesures, on doit repasser en mode sleep. En ce qui concerne le paramétrage, il faut rajouter une ligne pour repasser en mode sleep. Pour sécuriser, il convient de tester un temps maximum de paramétrage si oublie d'actionner le menu.

enfin, il faut merger l'ensemble des programmes précédent en un seul pour avoir un code final dans la ruche avec toutes les fonctionnalités embarquées.

Le code final V02 (en cours)

Code Final RUCHEV02

```
à ce stade (décembre 2020) le code nest pas encore terminé
```

L'alimentation (en cours)

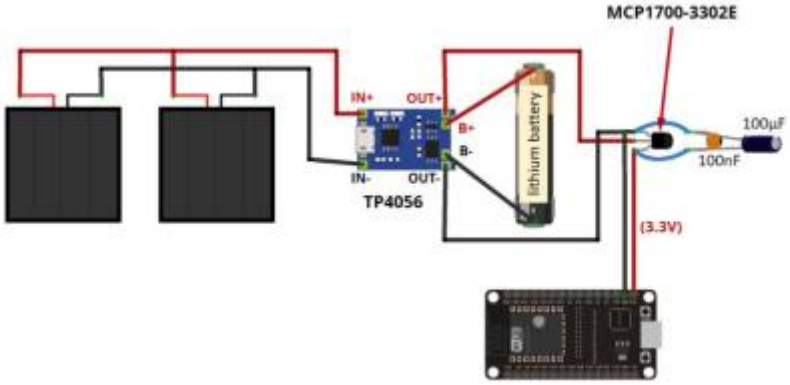
Le sujet est très compliqué. L'alimentation doit être entre 2,3V à 3,6V Les piles pertinentes sont en 3,7V. Sans doute faudra-t-il un renfort de panneaux solaires. Les régulateurs 3,3V existants demande une tension d'entrée minimum de 4,5V. etc...

Il est possible de vérifier la charge de la batterie et de l'envoyer en tant que données afin d'anticiper

le changement de la pile.

Je vais porter l'ensemble du code et du montage sur ce kit avec une pile 18650 intégrée. Dans un premier temps des tests de durée de la pile sont nécessaires.

Après de longs échanges et de longues recherches, je vais m'arrêter sur ce montage:



Le schéma électronique (en cours)

Le circuit imprimé (en cours)

Le montage (en cours)

Le prototype (en cours)

Les tests (en cours)

Le coût pour une ruche(en cours)

Liste des composants:

- un **ESP32** (**5**€)
- un **HX711** et **4 pesons** (**15**€)
- un **DS18DB20** (**2**€)
- un **DHT11** (**2**€)
- une pile **18650** (**2**€)
- un **boitier** de pile (**1**€)
- Un **boitier étanche** pour le montage ()
- Quelques **résistances** et circuit ()
- fil, vis, etc. ()
- **Total**:

Cette page a été consultée : Aujourd'hui: 7 Hier: 0 Jusqu'à maintenant: 830

From:
http://chanterie37.fr/fablab37110/ - **Castel'Lab le Fablab MJC de Château-Renault**

Permanent link:
http://chanterie37.fr/fablab37110/doku.php?id=start:projets:thierrylpr&rev=1612020869

Last update: **2023/01/27 16:08**