ESP32 : Programmation de base

Sommaire :

I - Adresse MAC de l'ESP32

Avec **VSCode** et **PlatformIO**, si on veut relever l'**adresse MAC** de notre **ESP32**, il faut exécuter le programme suivant :

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

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  delay(1000);
  Serial.println("");
  Serial.print("=> Addresse MAC WiFi : ");
  Serial.println(WiFi.macAddress());
  Serial.println();
}

void loop() {
 // put your main code here, to run repeatedly:
delay(1000);                    // wait for 1 second
}
```

II - Connexion de l'ESP32 à un point d'accès WiFi

Avec **VSCode** et **PlatformIO**, si on veut connecter notre **ESP32** à un point d'accès WiFi ayant pour **SSID** « **AP-BTS-SNIR** », il faut exécuter le programme suivant :

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

const char* ssid = "AP-BTS-SNIR";
const char* password = "12345678";

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println("");
```

```
  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    }
  Serial.println("");
  Serial.println("WiFi connexion OK ");
  Serial.print("=> Addresse IP : ");
  Serial.print(WiFi.localIP());
  Serial.println();
  Serial.print("=> Addresse MAC : ");
  Serial.println(WiFi.macAddress());
  Serial.println();
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(1000);                    // wait for 1 second
}
```

III - L'ESP32 en client NTP

1. Présentation

NTP (**Network Time Protocol**) utilise les protocoles **UDP** *(port 123)* et **IP**, et permet de **synchroniser les horloges d'ordinateurs** *(date et heure)* d'un même réseau à partir d'une machine de référence jouant le rôle de serveur **NTP**.

Le client va émettre une requête de synchronisation de temps *(horloge* **NTP***)*. Dès réception du message, le serveur entre alors en communication avec le client afin de lui renvoyer la date et l'heure.

2. Programme d'exemple

Pour programmer un client **NTP** sur un **ESP32**, il faut que celui-ci soit connecté à Internet via un point d'accès à l'aide de son interface WiFi. Puis il suffit d'utiliser la portion de code suivante :

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

const char* ssid     = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
const char* ntpServer = "pool.ntp.org";     // Le serveur NTP
const long  gmtOffset_sec = 0;
const int   daylightOffset_sec = 3600;

void setup(){
  Serial.begin(115200);
  // Connect to Wi-Fi
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
          delay(500);
           Serial.print(".");
```

```
        }
  Serial.println("");
  Serial.println("WiFi connected.");
  // Init and get the time
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
  printLocalTime();
  //disconnect WiFi as it's no longer needed
  WiFi.disconnect(true);
  WiFi.mode(WIFI_OFF);
}

void loop(){
        delay(1000);
        printLocalTime();
}

void printLocalTime(){
  struct tm timeinfo;
  if(!getLocalTime(&timeinfo)){
        Serial.println("Failed to obtain time");
        return;
        }
  Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
  Serial.print("Day of week: ");
  Serial.println(&timeinfo, "%A");
  Serial.print("Month: ");
  Serial.println(&timeinfo, "%B");
  Serial.print("Day of Month: ");
  Serial.println(&timeinfo, "%d");
  Serial.print("Year: ");
  Serial.println(&timeinfo, "%Y");
  Serial.print("Hour: ");
  Serial.println(&timeinfo, "%H");
  Serial.print("Hour (12 hour format): ");
  Serial.println(&timeinfo, "%I");
  Serial.print("Minute: ");
  Serial.println(&timeinfo, "%M");
  Serial.print("Second: ");
  Serial.println(&timeinfo, "%S");
  Serial.println("Time variables");
  char timeHour[3];
  strftime(timeHour,3, "%H", &timeinfo);
  Serial.println(timeHour);
  char timeWeekDay[10];
  strftime(timeWeekDay,10, "%A", &timeinfo);
  Serial.println(timeWeekDay);
  Serial.println();
}
```

IV - Les timers de l’ESP32

1. Présentation

L’**ESP32** dispose de **2 Timers**. Nous allons utiliser ici la librairie **kernel_timers** disponible à l’adresse : **https://bitbucket.org/henri_bachetti/kernel_timers/src/master/**.

2. Exemple d'utilisation de la librairie avec scrutation

```
#include <kernel_timers.h>
#define TIMEOUT         HZ
struct timer_list timer;

void setup()
{
 Serial.begin(9600);
 delay(2000);
 timer_init(1);       // On choisit le Timer 1
 init_timer(&timer);        // création du  Timer 1
 add_timer(&timer);       // Ajout du  Timer 1 au système
 Serial.println("\n Armement du Timer 1");
 mod_timer(&timer, jiffies + 5*TIMEOUT);    // Armement du  Timer 1
}

void loop()
{
 if (timer.expires != 0 && timer.expires == jiffies) {
   Serial.println("Désarmement du Timer 1");
   mod_timer(&timer, 0);
 }
}
```

HZ représente la fréquence d'actualisation des timers (l'interruption) et **jiffies** représente le temps courant. Donc **jiffies + TIMEOUT = jiffies + 5 * HZ** représente le **temps courant + 5 secondes**.

La librairie aura une valeur de **HZ** valant **200**. Cela nous donnera une résolution de **5 millisecondes** pour les timers.

3. Exemple d'utilisation de la librairie avec callback

```
#include <kernel_timers.h>
#define TIMEOUT         HZ
struct timer_list timer;
volatile bool timeoutOccured=false;

void  callbackTimeout(unsigned long data)
{
timeoutOccured=true;
}

void setup()
{
 Serial.begin(9600);
 delay(2000);
 timer_init(1);       // On choisit le Timer 1
 init_timer(&timer);        // création du  Timer 1
 setup_timer(&timer, callbackTimeout, 0); // Association du callback

 add_timer(&timer);       // Ajout du  Timer 1 au système
 Serial.println("\n Armement du Timer 1");
 mod_timer(&timer, jiffies + 5*TIMEOUT);    // Armement du  Timer 1
}
```

```
void loop()
{
  if (timeoutOccured == true) {
    Serial.println("Désarmement du Timer 1");
    timeoutOccured=false;
    mod_timer(&timer, 0);
  }
}
```

V - Les interruptions avec l'ESP32

1. Présentation

Lors d'une **interruption**, une fonction d'interruption est exécutée. Une interruption du programme peut être provoquée par un événement sur une broche du microcontrôleur (interruption externe) ou par un périphérique interne au microcontrôleur : UART, TIMER, SPI ... (interruption interne).

En règle générale une interruption peut être validée ou non (à l'exception des interruptions non masquables NMI), et un bit drapeau (Flag) passe à 1 pour mémoriser la demande d'interruption.

2. Interruption interne déclenchée par le TIMER

Le programme ci-dessous permet de déclencher une interruption (exécution de la fonction **void IRAM_ATTR onTimer()**) cycliquement par le **TIMER1** (16 bits) toutes les **1s**. L'interruption est désactivée au bout de 100 cycles.

```
volatile byte state = LOW;
volatile unsigned long compteur=0;
/* create a hardware timer */
hw_timer_t * timer = NULL;

void IRAM_ATTR onTimer(){
  state = !state;
  compteur++;
}

void setup() {
  Serial.begin(115200);
  /* Use TIMER1 */
  /* 1 tick take 1/(80MHZ/80) = 1us */
  timer = timerBegin(0, 80, true);

  /* Attach onTimer function to our timer */
  timerAttachInterrupt(timer, &onTimer, true);

  /* Set alarm to call onTimer function every second 1 tick is 1us
  => 1 second is 1000000 ticks */
  /* Repeat the alarm (third parameter) */
  timerAlarmWrite(timer, 1000000, true);

  /* Start an alarm */
  timerAlarmEnable(timer);
  Serial.println("start timer");
```

```
}

void loop() {
if (state == HIGH)
        {
        Serial.printf("Compteur = %d\n", compteur);
        state = LOW;
        }
  //Detach Interrupt if compteur=100
  if (compteur==100)
    {
    timerDetachInterrupt(timer);
    Compteur=0;
    }
}
```

3. Interruption externe déclenchée par une entrée GPIO

Le programme ci-dessous permet de déclencher une interruption (exécution de la fonction **void IRAM_ATTR isr()**) lors d'un front descendant de l'entrée **GPIO 18**. L'interruption est désactivée au bout d'une minute.

```
const uint8_t PIN = 18;
uint32_t numberKeyPresses = 0;
bool pressed = false;

void IRAM_ATTR isr() {
  numberKeyPresses ++;
  pressed = true;
}

void setup() {
  Serial.begin(115200);
  pinMode(PIN, INPUT_PULLUP);
  attachInterrupt(PIN, isr, FALLING);
}

void loop() {
  if (pressed) {
      Serial.printf("Button 1 has been pressed %u times\n", numberKeyPresses);
      pressed = false;
  }

  //Detach Interrupt after 1 Minute
  static uint32_t lastMillis = 0;
  if (millis() - lastMillis > 60000) {
    lastMillis = millis();
    detachInterrupt(PIN);
    Serial.println("Interrupt Detached!");
  }
}
```

VI - Les modes de fonctionnement (Sleep Mode) de l'ESP32

1. Présentation

Les **sleep modes** sont des états qui permettent de mettre en sommeil l'**ESP 32**. On éteint alors certains périphériques et les données sont conservées dans la RAM. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html.

Il existe **5** modes d'économie d'énergies différents :

- mode actif (par défaut) ;
- mode modem ;
- mode light speed ;
- mode sommeil profond ;
- mode hibernation.

2. Le mode Actif

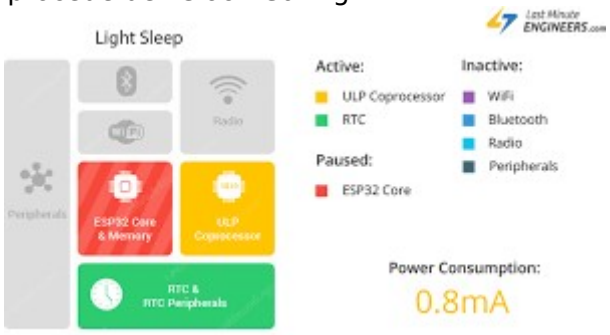
C'est le mode normal, tout est allumé.

3. Le mode Modem

Dans ce mode, le **WiFi**, le **Bluetooth** et la **radio** sont désactivés. L'**ESP32** peut entrer en mode veille du modem uniquement lorsqu'il se connecte au routeur en mode station (connexion sans fil). L'ESP 32 reçoit à intervalle régulier des message DTIM (Delivery Traffic Indication Message) du routeur pour l'informer que la communication avec celui ci est possible. Entre deux messages DTIM, les échanges de données avec le routeur sont impossibles, le WiFi et le Bluetooth sont donc désactivés durant ce laps de temps pour économiser de la batterie. Le temps de sommeil est généralement compris entre 100 et 1000 ms.

4. Le mode Light Sleep

Il comporte toutes les fonctionnalités du mode modem et il permet d'économiser encore plus de batterie en utilisant le procédé de "Clock-Gating".



Avant d'entrer en mode **light sleep**, L'**ESP 32** garde en mémoire les données et reprendra l'exécution du code là ou il s'est arrêté. L'avantage de ce mode est qu'il permet un réveil rapide tout en économisant de la batterie.

Le code suivant permet d'observer le fonctionnement du mode light sleep :

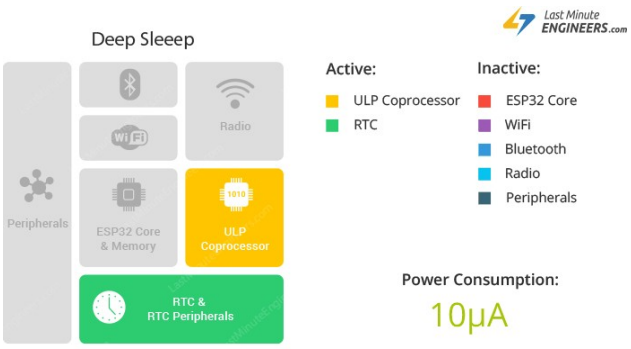
```
void setup() {
 Serial.begin(115200);
 Serial.println("setup");
}
void loop() {
 esp_sleep_enable_timer_wakeup(5000000); //5 seconds
 int ret = esp_light_sleep_start();
 Serial.print("light_sleep:");
 Serial.println(ret);
}
```

On observe que la carte ne reboote pas en sortie de sommeil et qu'elle reprend l'exécution du programme toutes les 5 secondes là où elle s'était arrêtée.

5. Le mode Deep Sleep (sommeil)

Pendant le **deep sleep** mode, la majeure partie de la RAM et tous les périphériques numériques sont mis hors tension. Les seules parties de la puce qui restent sous tension sont :

- contrôleur RTC
- périphériques RTC (y compris le processeur ULP)
- mémoires RTC (lentes et rapides).



Pour sauvegarder une variable même après avoir mis l'ESP en deep sleep mode il faut l'enregistrer dans la mémoire RTC en déclarant la variable en globale avec le type :

RTC_DATA_ATTR (par exemple :RTC_DATA_ATTR int bootCount = 0;)

Le code suivant permet d'observer le fonctionnement du mode **deep sleep** :

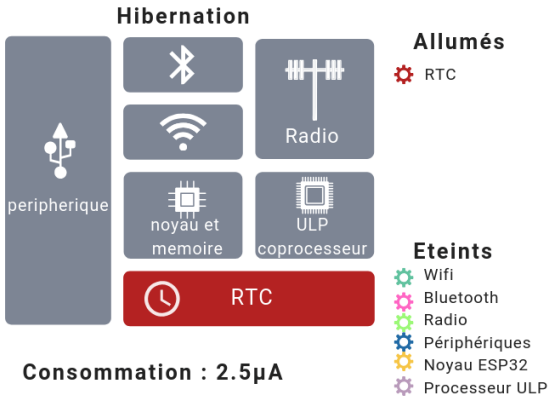
```
#define uS_TO_S_FACTOR 1000000 /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP 3 /* Time ESP32 will go to sleep (in seconds) */
RTC_DATA_ATTR int bootCount = 0;
int GREEN_LED_PIN = 25;
int YELLOW_LED_PIN = 26;
void setup(){
 pinMode(GREEN_LED_PIN,OUTPUT);
 pinMode(YELLOW_LED_PIN,OUTPUT);
 delay(500);
 if(bootCount == 0) //Run this only the first time
 {
 digitalWrite(YELLOW_LED_PIN,HIGH);
 bootCount = bootCount+1;
 }
```

```
else
 {
 digitalWrite(GREEN_LED_PIN,HIGH);
 }
 delay(3000) ;
 digitalWrite(GREEN_LED_PIN,LOW);
 digitalWrite(YELLOW_LED_PIN,LOW);
 esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
 esp_deep_sleep_start();
}
void loop(){
}
```

Contrairement au mode **light sleep**, la carte redémarre à chaque réveil et effectue donc à nouveau la fonction **setup()**.
Dans ce code, on instancie une variable bootcount qu'on stocke dans la mémoire RTC. La première fois que la fonction **setup()** est effectuée, la led jaune doit s'allumer car bountcount est égale à 0. En sortie de sommeil profond, Bountcount est égale à 1 car la variable a été incrémentée au tour d'avant. Cette fois ci, seule la led verte s'allume. La valeur de notre variable est conservée, la led jaune ne s'allume donc qu'une seule fois.

6. Le mode Hibernation

Il désactive les mêmes fonctionnalités désactivées par le mode sommeil profond. A cela s'ajoute le fait que la mémoire RTC est aussi privée d'alimentation. Il en résulte qu'il est impossible de sauvegarder des données en utilisant ce mode d'économie d'énergie. La consommation durant ce mode est censée être extrêmement faible (seulement quelques micro ampères) :



7. Comment réveiller l'ESP32 ?

Une fois mis en sommeil, il existe différentes façons de réveiller un ESP32 (par Timer ou par réveil externe). La déclaration de ces fonctions doit toujours se faire avant l'appel de la fonction provoquant le mode sleep. Cependant, vous pouvez toujours réveiller votre ESP32 en appuyant sur le bouton EN. Cela le redémarrera.

Réveil par Timer :

Cette méthode peut être utilisée quelque soit le mode d'économie d'énergie activé. La carte se met en sommeil durant un temps défini par l'utilisateur et se réveille ensuite.

```
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP)// temps en micro seconde;
```

C'est cette méthode qui a été utilisée dans les exemples ci dessus.

Réveil externe :

Il est possible de réveiller un ESP32 après lui avoir envoyé un signal digital sur une de ces pin.

```
void setup() {
 Serial.begin(115200);
 Serial.println("setup");
}
void loop() {
 esp_sleep_enable_ext0_wakeup(GPIO_NUM_27,HIGH);
 int ret = esp_light_sleep_start();
 Serial.print("light_sleep:");
 Serial.println(ret);
}
```

Ce code permet de sortir du mode light sleep dès que la pin 27 reçoit un 1 logique. Si le signal repasse à 0, la carte retombera en mode sommeil. Avec certains ajustements, il est aussi possible d'utiliser une interruption externe pour sortir du mode sommeil profond :

```
void setup() {
 Serial.begin(115200);
 Serial.println("setup");
}
void loop() {
 esp_sleep_enable_ext0_wakeup(GPIO_NUM_27,LOW);
 esp_deep_sleep_start();// la fonction ne retourne rien
}
```

On est plongé dans le mode sommeil profond tant que la valeur sur la pin 27 est à HIGH. Vous êtes maintenant en mesure de comprendre comment les différents sleep modes de l'ESP32 fonctionnent et vous pouvez désormais les utiliser.