Lab 8

Programmation basse consommation pour les terminaux ESP32 avec LoRa

Ce laboratoire est un guide pour l'ESP32 fonctionnant en mode basse consommation appelé mode veille profonde (*deep_sleep*). Nous allons vous montrer comment mettre l'ESP32 en veille profonde et examiner différents modes pour le réveiller: réveil par minuterie, réveil par les broches tactiles et réveil externe.
Ce laboratoire fournit des exemples pratiques de code et explique comment créer des nœuds de Terminal LoRa basse consommation.

Table des matières

8.1 Présentation des modes de veille (*sleep modes*)

Le SoC ESP32 peut basculer entre différents modes d'alimentation :

- Mode actif
- Mode veille du modem (*WiFi/BT modem-sleep*)
- Mode veille légère (*light sleep*)
- Mode veille profonde (*deep sleep*)
- Mode veille prolongée (*hibernation*)

Power mode	Active	Modem-sleep	Light-sleep	Deep-sleep	Hibernation
Sleep pattern	Association sleep pattern			ULP sensor-monitored pattern	-
CPU	ON	ON	PAUSE	OFF	OFF
Wi-Fi/BT baseband and radio	ON	OFF	OFF	OFF	OFF
RTC memory and RTC peripherals	ON	ON	ON	ON	OFF
ULP co-processor	ON	ON	ON	ON/OFF	OFF

Tableau 8.1 Cinq modes d'alimentation différents du SoC ESP32.

La fiche technique **ESP32 Espressif** fournit également un tableau comparatif de la consommation électrique des différents modes de puissance.

Power mode	Description	Power consumption
Active (RF working)	Wi-Fi Tx packet 14 dBm ~ 19.5 dBm	Please refer to Table 10 for details.
	Wi-Fi / BT Tx packet 0 dBm	
	Wi-Fi / BT Rx and listening	
Modem-sleep	The CPU is powered on.	Max speed 240 MHz: 30 mA ~ 50 mA
		Normal speed 80 MHz: 20 mA ~ 25 mA
		Slow speed 2 MHz: 2 mA ~ 4 mA
Light-sleep	-	0.8 mA
Deep-sleep	The ULP co-processor is powered on.	150 μA
	ULP sensor-monitored pattern	100 μA @1% duty
	RTC timer + RTC memory	10 μA
Hibernation	RTC timer only	5 μA
Power off	CHIP_PU is set to low level, the chip is powered off	0.1 μA

Tableau 8.2 Consommation d'énergie pour différents modes d'alimentation de l'ESP32.

Ci-dessous un tableau pour comparer la consommation électrique en mode actif.

Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	-	240	-	mA
Transmit 802.11b, OFDM 54 Mbps, POUT = +16 dBm	-	190	-	mA
Transmit 802.11g, OFDM MCS7, POUT = +14 dBm	-	180	-	mA
Receive 802.11b/g/n	-	95 ~ 100	-	mA
Transmit BT/BLE, POUT = 0 dBm	-	130	-	mA
Receive BT/BLE	-	95 ~ 100	-	mA

Tableau 8.3 Consommation d'énergie en mode actif.

8.1.1 Pourquoi le mode veille profonde ?

Faire fonctionner votre ESP32 en mode actif avec des piles n'est pas idéal, car l'énergie des piles s'épuise très rapidement. Si vous mettez votre ESP32 en mode veille profonde, cela réduira la consommation d'énergie et vos batteries dureront plus longtemps.
Avoir votre ESP32 en mode **veille profonde** signifie réduire les activités qui consomment plus d'énergie pendant le fonctionnement, mais laisser **juste assez d'activité** pour réveiller le processeur lorsque quelque chose d'intéressant se produit.
En mode de veille profonde, ni le processeur ni les activités Wi-Fi n'ont lieu, mais le coprocesseur **Ultra Low Power** (**ULP**) est toujours alimenté.

Pendant que l'ESP32 est en mode de veille profonde, la mémoire de l'horloge en temps réel **RTC** (*Real Time Clock*) reste également allumée, afin que nous puissions écrire un programme pour le coprocesseur ULP et le stocker dans la mémoire RTC pour accéder aux périphériques, aux minuteries internes et internes et aux capteurs.
Ce mode de fonctionnement est utile si vous devez réveiller le processeur principal par un **événement externe**, une **minuterie** ou les deux, tout en maintenant une consommation d'énergie minimale.

8.1.2 Broches `RTC_GPIO`

Pendant le sommeil profond, certaines des broches ESP32 peuvent être utilisées par le coprocesseur ULP, à savoir les broches **`RTC_GPIO`** et les broches tactiles. Jetons un coup d'œil au brochage suivant pour localiser les différentes broches **`RTC_GPIO`**.
Les broches **`RTC_GPIO`** sont mises en évidence par une case rectangulaire orange. Les broches **`GPIO6`** et **`GPIO11`**, non exposées,sont connectées au **flash SPI intégré**.

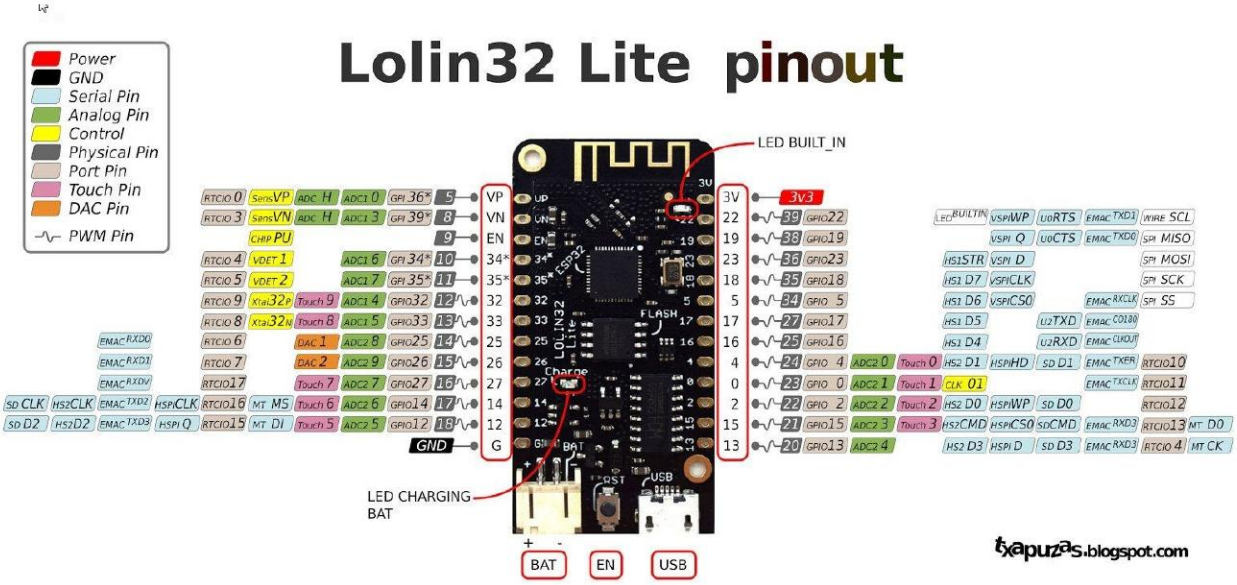


Figure 8.1 Broches **`GPIO`** ESP32 (Lolin32)

8.2 Sources du réveil (*Wake Up*)

Après avoir mis l'ESP32 en mode veille prolongée, il existe plusieurs façons de le réveiller :

1. Vous pouvez utiliser la minuterie pour réveiller votre ESP32 en utilisant des périodes de temps prédéfinies
2. Vous pouvez utiliser les broches tactiles
3. Vous pouvez utiliser deux possibilités de réveil externe : vous pouvez utiliser soit un **réveil externe**, soit plusieurs réveils externes différents
4. Vous pouvez utiliser le coprocesseur ULP - cela ne sera pas couvert dans ce guide.

8.3 Ecrire un programme pour le mode de sommeil profond

Pour écrire un programme pour mettre votre ESP32 en mode de veille profonde, puis le réactiver, vous devez garder à l'esprit que :

- Tout d'abord, vous devez **configurer les sources de réveil**. Cela signifie configurer ce qui réveillera l'ESP32. Vous pouvez utiliser une ou combiner plusieurs sources de réveil.
- Vous pouvez décider des périphériques à arrêter ou à conserver pendant le sommeil profond. Cependant, par défaut, l'ESP32 met automatiquement hors tension les périphériques qui ne sont pas nécessaires avec la source de réveil que vous définissez.
- Enfin, vous utilisez la fonction **`esp_deep_sleep_start()`** pour mettre votre ESP32 en mode sommeil profond.

8.3.1 Réveil par la minuterie (*Timer Wake Up*)

L'ESP32 peut passer en mode veille profonde, puis se réveiller à des périodes prédéfinies. Cette fonction est particulièrement utile si vous exécutez des projets qui nécessitent un horodatage ou des tâches quotidiennes (par exemple scrutation des capteurs), tout en maintenant une faible consommation d'énergie.
Le contrôleur ESP32 RTC dispose d'une minuterie intégrée que vous pouvez utiliser pour réveiller l'ESP32 après un laps de temps prédéfini.

8.3.1.1 Activer le réveil par minuterie

Activer l'ESP32 pour se réveiller après un laps de temps prédéfini est très simple. Dans l'IDE Arduino, il vous suffit de spécifier le **temps de veille en microsecondes** dans la fonction suivante:

```
esp_sleep_enable_timer_wakeup(time_in_us)
```

Exemple de code

Voyons comment cela fonctionne en utilisant un exemple de la bibliothèque. Ouvrez votre IDE Arduino, accédez à **`File`**>**`Examples`**>**`ESP32`**> **`Deep Sleep`** et ouvrez le croquis **`TimerWakeUp`**.

```
#define uS_TO_S_FACTOR 1000000  /* micro seconds to seconds */
#define TIME_TO_SLEEP  5        /* Time ESP32 will go to sleep (in seconds) */

RTC_DATA_ATTR int bootCount = 0;

void print_wakeup_reason(){
  esp_sleep_wakeup_cause_t wakeup_reason;
  wakeup_reason = esp_sleep_get_wakeup_cause();
  switch(wakeup_reason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("External signal RTC_IO");
         break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("External RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("ULP program"); break;
    default : Serial.printf("Not caused by deep sleep: %d\n",wakeup_reason);
         break;
  }
}
```

```
void setup(){
  Serial.begin(9600);
  delay(1000);
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));
  print_wakeup_reason(); // print the wakeup reason for ESP32
  // We set our ESP32 to wake up every 5 seconds
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.println("Sleep for every "+String(TIME_TO_SLEEP)+"Seconds");
 // The line below turns off all RTC peripherals in deep sleep.
 // esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
 // Serial.println("All RTC Peripherals to be powered down in sleep");
  Serial.println("Going to sleep now");
  delay(1000);
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

void loop(){//This is not going to be called
}
```

Jetons un coup d'œil sur ce code. Le premier commentaire décrit ce qui est éteint pendant le sommeil profond avec réveil par minuterie.
Dans ce mode, les processeurs, la plupart de la RAM et tous les périphériques numériques qui sont cadencés à partir d'**APB_CLK** sont mis hors tension. A noter que les minuteries sont cadencées par **APB_CLK** ; le manuel de référence technique indique que **APB_CLK** est dérivée de **CPU_CLK**.
Les seules parties du SoC qui peuvent encore être mises sous tension sont : le contrôleur **RTC**, les périphériques **RTC** et les mémoires **RTC**.
A noter que la fonction **void print_wakeup_reason)** peut être ajoutée à votre programme par

```
#include «ESP32_WakeUp.h»
```

répertoriée dans la partie Annexe de ce laboratoire.

8.3.1.2 Définir le temps de sommeil

Ces deux premières lignes de code définissent la période pendant laquelle l'ESP32 sera en veille.

```
#define uS_TO_S_FACTOR 1000000 /* Conversion from micro seconds to seconds */
#define TIME_TO_SLEEP 5 /* Time ESP32 will go to sleep (in seconds) */
```

Cet exemple utilise un facteur de conversion de microsecondes en secondes, afin que vous puissiez définir le temps de veille dans la variable **TIME_TO_SLEEP** en secondes. Dans ce cas, l'exemple mettra l'ESP32 en mode veille prolongée pendant 5 secondes.

8.3.1.3 Enregistrer les données dans les mémoires RTC

Avec l'ESP32, vous pouvez enregistrer des données dans les mémoires **RTC**. L'ESP32 dispose de 8 Ko de **SRAM** sur la partie **RTC**, appelée mémoire rapide RTC. Les données enregistrées ici ne sont pas effacées pendant le sommeil profond. Cependant, elles seront effacées lorsque vous appuyez sur le bouton de réinitialisation (le bouton étiqueté **RST** sur la carte ESP32).
Pour enregistrer des données dans la mémoire RTC, il suffit d'ajouter **RTC_DATA_ATTR** avant une définition de variable. L'exemple enregistre la variable **bootCount** sur la mémoire RTC. Cette variable comptera le nombre de fois que l'ESP32 s'est réveillé du sommeil profond.

```
RTC_DATA_ATTR int bootCount = 0;
```

```
Boot number: 9
Wakeup caused by timer
Setup ESP32 to sleep for every 5 Seconds
Configured all RTC Peripherals to be powered down in sleep
Going to sleep now
⸮⸮⸮⸮D)!11⸮⸮⸮z⸮1⸮⸮⸮⸮

Boot number: 10
Wakeup caused by timer
Setup ESP32 to sleep for every 5 Seconds
Configured all RTC Peripherals to be powered down in sleep
Going to sleep now
```

8.3.1.4 Raison du réveil

La fonction **`print_wakeup_reason()`** imprime la raison pour laquelle l'ESP32 a été réveillé du sommeil.

8.3.1.5 Le `setup()`

Dans le **`setup()`** est l'endroit où vous devez mettre la totalité de votre code. En sommeil profond, l'esquisse n'atteint jamais l'instruction **`loop()`**.
Donc, vous devez écrire tout le croquis dans setup ().
Ensuite, la variable **`bootCount`** est augmentée de un à chaque redémarrage, et ce numéro est imprimé sur le moniteur série.

```
++bootCount;
Serial.println("Boot number: " + String(bootCount));
```

Ensuite, le code appelle la fonction **`print_wakeup_reason()`**, mais vous pouvez appeler n'importe quelle fonction pour effectuer une tâche souhaitée. Par exemple, si vous souhaiterez réveiller votre ESP32 une fois par jour pour lire une valeur d'un capteur.

Ensuite, le code impose la **source de réveil** à l'aide de la fonction suivante :

```
esp_sleep_enable_timer_wakeup(time_in_us)
```

Cette fonction accepte comme argument le temps de sommeil en microsecondes comme nous l'avons vu précédemment.Dans notre cas, nous avons les éléments suivants :

```
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
```

Ensuite, une fois toutes les tâches effectuées, l'ESP32 se met en veille en appelant la fonction suivante:

```
esp_deep_sleep_start()
```

8.3.1.6 La boucle - `loop()`

La section l**`oop()`** est vide, car l'ESP32 va s'endormir avant d'atteindre cette partie du code.

8.3.2 Le réveil par broches tactiles

Vous pouvez réveiller l'ESP32 du sommeil profond à l'aide des broches tactiles.

8.3.2.1 Activer le réveil tactile

Pour activer l'ESP32 pour qu'il puisse se réveiller à l'aide d'une broche tactile est simple. Vous devez utiliser la fonction suivante :

```
esp_sleep_enable_touchpad_wakeup()
```

Exemple de code

```
#include "ESP32_WakeUp.h"  // listing in Annex section
#define Threshold 40 /* Greater the value, more the sensitivity */
RTC_DATA_ATTR int bootCount = 0;
touch_pad_t touchPin;

void print_wakeup_reason(){
  esp_sleep_wakeup_cause_t wakeup_reason;
  wakeup_reason = esp_sleep_get_wakeup_cause();
  switch(wakeup_reason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("External  RTC_IO");break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("External  RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("ULP program"); break;
    default : Serial.printf("Not caused by deep sleep: %d\n",wakeup_reason);
        break;
  }
}
```

```
// present in ESP32_WakeUp.h file
void print_wakeup_touchpad(){
  touchPin = esp_sleep_get_touchpad_wakeup_status();
  switch(touchPin)
  {
    case 0  : Serial.println("Touch detected on GPIO 4"); break;
    case 1  : Serial.println("Touch detected on GPIO 0"); break;
    case 2  : Serial.println("Touch detected on GPIO 2"); break;
    case 3  : Serial.println("Touch detected on GPIO 15"); break;
    case 4  : Serial.println("Touch detected on GPIO 13"); break;
    case 5  : Serial.println("Touch detected on GPIO 12"); break;
    case 6  : Serial.println("Touch detected on GPIO 14"); break;
    case 7  : Serial.println("Touch detected on GPIO 27"); break;
    case 8  : Serial.println("Touch detected on GPIO 33"); break;
    case 9  : Serial.println("Touch detected on GPIO 32"); break;
    default : Serial.println("Wakeup not by touchpad"); break;
  }
}

void callback(){ //placeholder callback function
}

void setup(){
  Serial.begin(9600);
  delay(1000); //Take some time to open up the Serial Monitor
  //Increment boot number and print it every reboot
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));
  //Print the wakeup reason for ESP32 and touchpad too
  print_wakeup_reason();
  print_wakeup_touchpad();
  //Setup interrupt on Touch Pad 3 (GPIO15) - the tested pad
  touchAttachInterrupt(T3, callback, Threshold);
  //Configure Touchpad as wakeup source
  esp_sleep_enable_touchpad_wakeup();
  //Go to sleep now
  Serial.println("Going to sleep now");
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

void loop(){
  //This will never be reached
}
```

```
Boot number: 1
Wakeup was not caused by deep sleep: 0
Wakeup not by touchpad
Going to sleep now
99¶J□L¶1!11□□¶jL¶9¶!¶¶

Boot number: 2
Wakeup caused by touchpad
Touch detected on GPIO 15
Going to sleep now
□□□¶□□¶1¶))))¶¶1□¶9¶¶¶

Boot number: 3
Wakeup caused by touchpad
Touch detected on GPIO 15
Going to sleep now
¶¶¶¶□D)!1q¶□¶1□¶9¶!¶¶
```

8.3.2.2 Définition du seuil

La première chose à faire est de définir une valeur de **seuil** pour les broches tactiles. Dans ce cas, nous définissons le **seuil sur 40**. Vous devrez peut-être modifier la valeur du seuil en fonction de votre projet.

```
#define Threshold 40
```

Lorsque vous touchez un GPIO tactile, la valeur lue par le capteur diminue. Ainsi, vous pouvez définir une valeur de seuil qui provoque un événement lorsque le toucher est détecté.
La valeur de seuil définie ici signifie que lorsque la valeur lue par le GPIO tactile est inférieure à 40, l'ESP32 doit se réveiller. Vous pouvez ajuster cette valeur en fonction de la sensibilité souhaitée.

8.3.2.3 Association des interruptions

Vous devez attacher des interruptions aux broches sensibles au toucher. Lorsque le toucher est détecté sur un GPIO spécifié, une fonction de rappel est exécutée. Comme exemple, regardez la ligne suivante:

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

Lorsque la valeur lue sur **`GPIO 15`** est inférieure à la valeur définie sur la variable **`Threshold`**, l'ESP32 se réveille et la fonction de rappel est exécutée.

La fonction **`callback()`** ne sera exécutée que si l'ESP32 est réveillé.

- Si l'ESP32 est en veille et que vous touchez **`T3 – GPIO 15`**, l'ESP se réveillera - la fonction **`callback()`** ne sera pas exécutée si vous **appuyez et relâchez** simplement la broche tactile;
- Si l'ESP32 est réveillé et que vous touchez **`T3 – GPIO 15`**, la fonction de rappel sera exécutée. Donc, si vous souhaitez exécuter la fonction **`callback()`** lorsque vous réveillez l'ESP32, vous devez maintenir le contact sur cette broche **pendant un moment**, jusqu'à ce que la fonction soit exécutée.

Dans ce cas, la fonction **`callback()`** est vide. I

```
void callback(){
   //placeholder callback function
}
```

```
Boot number: 5
Wakeup caused by touchpad
Touch detected on GPIO 15
Going to sleep now
ŸŸŸŸ□ŸŸJ!!!Ÿ□ŸJ~ 191ŸŸ

Boot number: 6
Wakeup caused by touchpad
Touch detected on GPIO 15
Going to sleep now
Hello from callback function
Hello from callback function
Hello from callback function
Hello from callback function
Hello from callback function
Hello from callback function
Hello from callback function
Hello from callback function
Hello from callback function
```

Si vous souhaitez réveiller l'ESP32 en utilisant différentes broches tactiles, il vous suffit d'attacher des interruptions à ces broches.
Ensuite, vous devez utiliser la fonction **`esp_sleep_enable_touchpad_wakeup()`** pour définir les broches tactiles comme source de réveil.

```
//Configure Touchpad as wakeup source
esp_sleep_enable_touchpad_wakeup()
```

8.3.3 Réveil externe

Outre la minuterie et les broches tactiles, nous pouvons également réveiller l'ESP32 du sommeil profond en basculant la valeur d'un signal sur une broche, par la pression d'un bouton. C'est ce qu'on appelle un **réveil externe**. Vous avez deux possibilités (broches) de réveil externe : **`ext0`** et **`ext1`**.

8.3.3.1 Réveil externe (`ext0`)

Cette source de réveil vous permet d'utiliser **une broche** pour réveiller l'ESP32. L'option source de réveil **`ext0`** utilise les **GPIO RTC** pour se réveiller. Ainsi, les **périphériques RTC resteront allumés** pendant le sommeil profond si cette source de réveil est demandée.
Pour utiliser cette source de réveil, vous utilisez la fonction suivante:

```
esp_sleep_enable_ext0_wakeup(GPIO_NUM_X, level)
```

Cette fonction accepte comme premier argument la broche que vous souhaitez utiliser, dans ce format **`GPIO_NUM_X`**, dans lequel **`X`** représente le numéro **GPIO** de cette broche.
Le deuxième argument, **`level`**, peut être **`1`** ou **`0`**. Cela représente l'état du GPIO qui déclenchera le réveil.

8.3.3.2 Réveil externe (`ext1`)

Cette source de réveil vous permet d'utiliser **plusieurs GPIO RTC**. Vous pouvez utiliser deux fonctions logiques différentes :

- Réveillez l'ESP32 si l'une des broches que vous avez sélectionnées est haute (**`1 – HIGH`**)
- Réveillez l'ESP32 si toutes les broches que vous avez sélectionnées sont basses (**`0 – LOW`**)

Cette source de réveil est implémentée par le contrôleur RTC. Ainsi, les **périphériques RTC** et les **mémoires** RTC peuvent être mis **hors tension** dans ce mode.

Pour utiliser cette source de réveil, vous utilisez la fonction suivante :

```
esp_sleep_enable_ext1_wakeup(bitmask, mode)
```

Cette fonction accepte deux arguments:

- Un masque de bits (**`bitmask`**) des numéros GPIO qui provoqueront le réveil
- **Mode**: la logique de réveil de l'ESP32. Ça peut être:
  - **`ESP_EXT1_WAKEUP_ALL_LOW`** : se réveiller lorsque tous les GPIO deviennent bas
  - **`ESP_EXT1_WAKEUP_ANY_HIGH`** : réveillez-vous si l'un des GPIO devient haut

Exemple complet :

```
#include "ESP32_WakeUp.h"  // listing in Annex section
#define BUTTON_PIN_BITMASK 0x200000000 // 2^33 in hex
RTC_DATA_ATTR int bootCount = 0;

void setup(){
  Serial.begin(9600);
  pinMode(33,INPUT_PULLUP);  // dafault state of the pin - HIGH
  delay(1000); //Take some time to open up the Serial Monitor
  //Increment boot number and print it every reboot
  ++bootCount;
  Serial.println(); Serial.println();
  Serial.println("Boot number: " + String(bootCount));
  print_wakeup_reason();
  esp_sleep_enable_ext0_wakeup(GPIO_NUM_33,0); //1 = High, 0 = Low
  // use the switch on the board with pin 33 to set signal to LOW
  //If you were to use ext1, you would use it like
  //esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK,ESP_EXT1_WAKEUP_ANY_LOW);
  Serial.println("Going to sleep now");
  delay(1000);
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

void loop(){
  //This is not going to be called
}
```

Cet exemple réveille l'ESP32 lorsque vous déclenchez **`GPIO 33`** sur **`LOW`**. Le code montre comment utiliser les deux méthodes: **`ext0`** et **`ext1`**. Si vous importez le code tel quel, vous utiliserez **`ext0`**.

La fonction à utiliser **ext1** est commentée. Nous allons vous montrer comment fonctionnent les deux méthodes et comment les utiliser.

```
Serial.begin(9600);
delay(1000); //Take some time to open up the Serial Monitor
```

Vous incrémentez un à la variable **bootCount** et imprimez cette variable dans le moniteur série (**Serial**).

```
++bootCount;
Serial.println("Boot number: " + String(bootCount));
```

Ensuite, vous imprimez la raison du réveil en utilisant la fonction print_wakeup_reason () définie précédemment.

```
print_wakeup_reason();
```

Après cela, vous devez activer les sources de réveil. Nous testerons séparément chacune des sources de réveil, **ext0** et **ext1**.

ext0

Dans cet exemple, l'ESP32 se réveille lorsque le **GPIO 33** est déclenché au niveau bas (**Low**) :

```
esp_sleep_enable_ext0_wakeup(GPIO_NUM_33,0); //1 = High, 0 = Low
```

Au lieu de **GPIO 33**, vous pouvez utiliser n'importe quelle autre broche **GPIO RTC**.

```
Boot number: 1
Wakeup was not caused by deep sleep: 0
Going to sleep now
ςςςς□D)!1cς□ς1□ς9ς!ςς

Boot number: 2
Wakeup caused by external signal using RTC_IO
Going to sleep now
09ς J□L0Nς!11□□ςzς1ςςς! ς

Boot number: 3
Wakeup caused by external signal using RTC_IO
Going to sleep now
□□a□ς□L0N□ς(Jƒς□N)ςa□ςς

Boot number: 4
Wakeup caused by external signal using RTC_IO
Going to sleep now
09ς J□L0Nς!11□□ςzςQ
ς
```

ext1

L'**ext1** vous permet de réveiller l'ESP à l'aide de différents boutons et d'effectuer différentes tâches en fonction du bouton sur lequel vous avez appuyé.
Au lieu d'utiliser la fonction **esp_sleep_enable_ext0_wakeup()**, vous utilisez la fonction fonction **esp_sleep_enable_ext1_wakeup()**.

Dans le code, cette fonction est commentée :

```
//If you were to use ext1, you would use it like
//esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK,ESP_EXT1_WAKEUP_ANY_HIGH);
```

Dé-commentez cette fonction pour avoir :

```
esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK,ESP_EXT1_WAKEUP_ANY_HIGH);
```

Le premier argument de la fonction est un masque de bits des GPIO que vous utiliserez comme source de réveil, et le deuxième argument définit la logique pour réveiller l'ESP32.

Dans cet exemple, nous utilisons la variable **`BUTTON_PIN_BITMASK`**, qui a été définie au début du code :

```
#define BUTTON_PIN_BITMASK 0x200000000 // 2^33 in hex
```

Ceci ne définit qu'**une seule broche** comme source de réveil, **`GPIO 33`**. Vous devez modifier le masque de bits pour configurer plus de broches GPIO comme source de réveil.

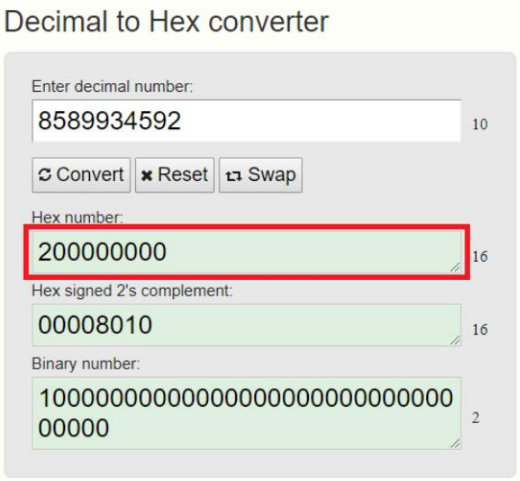
Création de GPIOs `bitmask`

Pour obtenir le masque de bits GPIO (**`bitmask`**), procédez comme suit :

1. Calculez **`2^(GPIO_NUMBER)`**. Enregistrez le résultat en décimal.
2. Allez sur **`http://rapidtables.com/convert/number/decimal-to-hex.html`** et convertissez le nombre décimal en hexadécimal;
3. Remplacez le nombre hexadécimal que vous avez obtenu dans la variable **`BUTTON_PIN_BITMASK`**.

Prenons un exemple. Dans le code de la librairie, le bouton est connecté à **`GPIO 33`**.
Pour obtenir le masque pour **`GPIO 33`** :

1. Calculez **`2^33`**. Vous devriez obtenir **8589934592,**
2. Convertissez ce nombre (**8589934592**) en **hexadécimal**. Vous pouvez accéder à ce convertisseur pour ce faire:



3. Copiez le numéro hexadécimal dans la variable **`BUTTON_PIN_BITMASK`** et vous devriez obtenir :

```
#define BUTTON_PIN_BITMASK 0x200000000 // 2^33 in hex
```

Masque pour plusieurs GPIO

Si vous souhaitez utiliser **`GPIO 34`** et **`GPIO 35`** comme source de réveil, vous devez procéder comme suit :

1. Calculez **`2^35 + 2^34`**. Vous devriez obtenir **51539607552**
2. Convertissez ce nombre en hexadécimal. Vous devriez obtenir : **`C00000000`**
3. Remplacez ce numéro dans le **`BUTTON_PIN_BITMASK`** comme suit :

```
#define BUTTON_PIN_BITMASK 0xC00000000
```

Identifier le GPIO utilisé comme source de réveil

Lorsque vous utilisez plusieurs broches pour réveiller l'ESP32, il est utile de savoir quelle broche a provoqué le réveil. Pour cela, vous pouvez utiliser la fonction suivante :

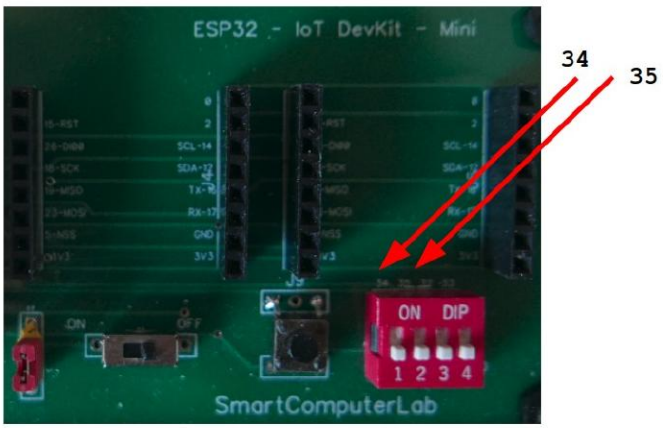
```
esp_sleep_get_ext1_wakeup_status()
```

Cette fonction renvoie un nombre de base 2, avec le nombre GPIO comme exposant: **`2^(GPIO_NUMBER)`**.
Donc, pour obtenir le GPIO en décimal, vous devez effectuer le calcul suivant :

```
GPIO = log(GPIO_NUMBER)/log(2)
```

8.3.3.2 Réveil externe - plusieurs GPIO

Maintenant, vous devriez pouvoir réveiller l'ESP32 à l'aide de différents boutons et identifier le bouton qui a provoqué le réveil. Dans cet exemple, nous utiliserons **`GPIO 34`** et **`GPIO 35`** comme source de réveil.



Le code

Vous devez apporter quelques modifications à l'exemple de code que nous avons utilisé auparavant :

- créer un masque de bits pour utiliser **`GPIO 34`** et **`GPIO 35`**. Nous vous avons montré comment faire cela auparavant;
- activer **`ext1`** comme source de réveil;
- utiliser la fonction **`esp_sleep_get_ext1_wakeup_status()`** pour obtenir le **GPIO** qui a déclenché le réveil.

L'esquisse suivante a tous ces changements mis en œuvre.

```
#include "ESP32_WakeUp.h"  // listing in Annex section
#define BUTTON_PIN_BITMASK 0xC00000000 // GPIOs 34 and 35
RTC_DATA_ATTR int bootCount = 0;

void print_GPIO_wake_up(){
long long GPIO_reason = esp_sleep_get_ext1_wakeup_status();
  Serial.print("GPIO that triggered the wake up: GPIO ");
  Serial.println((log(GPIO_reason))/log(2), 0);
}

void setup(){
  Serial.begin(9600);
  pinMode(34,INPUT_PULLUP); pinMode(35,INPUT_PULLUP);
  delay(1000); //Take some time to open up the Serial Monitor
  //Increment boot number and print it every reboot
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));
  //Print the wakeup reason for ESP32
  print_wakeup_reason();
  //Print the GPIO used to wake up
  print_GPIO_wake_up();
  //esp_deep_sleep_enable_ext0_wakeup(GPIO_NUM_33,1); //1 = High, 0 = Low
  //If you were to use ext1, you would use it like
  esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK,ESP_EXT1_WAKEUP_ANY_HIGH);
  //Go to sleep now
  Serial.println("Going to sleep now");
  delay(1000);
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

void loop(){
  //This is not going to be called }
```

8.4 Sommeil profond et modem LoRa

Dans le laboratoire 7 précédent, nous avons étudié et expérimenté avec plusieurs exemples de communication LoRa avec des nœuds terminaux et de passerelle. Afin de réduire la consommation d'énergie, le Terminal fonctionne souvent en mode basse consommation activé sur ESP32 via le mode veille profonde.

L'exemple suivant montre l'utilisation du mode de veille prolongée avec l'interruption par la minuterie dans un Terminal avec modem LoRa (SX1278) connecté via le bus SPI.

8.4.1 Un simple Terminal LoRa avec mode deep_sleep

À chaque cycle de fonctionnement avec l'envoi d'un paquet Lora, le modem est:

- connecté au bus SPI
- activé
- il envoie un paquet - ~ 40 mA (la consommation électrique dépend de la durée de la transmission)
- désactivé (veille) - 0,2 mA et
- déconnecté du bus SPI - ~ 3,3 µA

8.4.1.1 Code complet avec un simple cycle d'envoi de paquet LoRa et un sommeil profond

```
#include <LoRa.h>
#include <Wire.h>
#include "LoRa_Para.h"
#include "ESP32_WakeUp.h"
#define uS_TO_S_FACTOR 1000000  /* micro seconds to seconds */
#define TIME_TO_SLEEP  10       /* Time ESP32 will go to sleep (in seconds) */
RTC_DATA_ATTR int bootCount = 0;

char dbuff[24];

void lora_send()
{
set_LoRa();
  LoRa.beginPacket();
  LoRa.print("hello ");
  LoRa.print(bootCount);
  LoRa.endPacket();
}

void setup(){
  Serial.begin(9600);
  delay(1000); //Take some time to open up the Serial Monitor
  ++bootCount;
  Serial.println();Serial.println();
  Serial.println("Boot number: " + String(bootCount));
  print_wakeup_reason();
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.println("Setup to sleep for every "+String(TIME_TO_SLEEP)+" Seconds");
  esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
  Serial.println("Configured all RTC Peripherals to be powered down in sleep");
  delay(1000);
  lora_send();
  Serial.println("Ending LoRa");
  LoRa.end();    // disconnects SPI
  Serial.println("Going to sleep now");
  delay(300);   // wait to end the SPI bus transaction
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}


void loop(){}
```

Vous trouverez ci-dessous un fragment de la sortie affichée:

```
..
Boot number: 51
Wakeup caused by timer
Setup to sleep for every 10 Seconds
Configured all RTC Peripherals to be powered down in sleep
Starting LoRa ok!
Packet sent
Ending LoRa
Going to sleep now
Boot number: 52
Wakeup caused by timer
Setup to sleep for every 10 Seconds
Configured all RTC Peripherals to be powered down in sleep
Starting LoRa ok!
Packet sent
Ending LoRa
Going to sleep now
```

8.4.2 Terminal LoRa simple et liaison de données en mode `deep_sleep`

L'exemple suivant est l'extension de l'exemple étudié dans le laboratoire précédent. Il s'agit d'un nœud **Terminal** qui envoie les données du capteur température-humidité.
Ici, nous appliquons le mode **`deep_sleep`** avec la valeur du délai de réveil. La valeur de temporisation peut être intégrée dans les paramètres du terminal ou peut être fournie par le nœud **Gateway** (passerelle) via le paquet de retour.

8.4.2.1 Le code du Terminal avec le mode deep_sleep

```
#define TERMINAL
#define SHT21_SLEEP
#include <esp_wifi.h>
#include <SPI.h>
#include <LoRa.h>
#include "LoRa_Para.h"
#include "SHT21.h"

SHT21 SHT21;

typedef union
{
  uint8_t frame[32];    // data frame to send/receive data
  struct
  {
    uint32_t did;        // destination identifier chipID (4 lower bytes)
    uint32_t sid;        // source identifier chipID (4 lower bytes)
    float   sens[4];     // max 4 values - fields
    uint32_t  tout;      // optional timeout
    uint8_t pad[4];      // padding
  } pack;                // data packet
} DT_t;                  // DaTa type


RTC_DATA_ATTR uint32_t gwID=0, termID;  // gateway identifier stored in RTC memory
RTC_DATA_ATTR uint32_t cycle=8000, rcv_cycle, rcv_ack=0;    // main cycle in millis

void LoRa_send(uint8_t *frame) {
  LoRa_txMode();                        // set tx mode - normal mode
  LoRa.beginPacket();                   // start packet
  LoRa.write(frame,32);                  // add payload
  LoRa.endPacket(true);                 // finish packet and send it
}

void onReceive(int packetSize) {
  DT_t p;int i=0;
  if(packetSize==32)
    {
    while (LoRa.available())  { p.frame[i]= LoRa.read(); i++; }
    Serial.print("Terminal Received Packet:");
    Serial.println(p.pack.tout);
    rcv_cycle=p.pack.tout; rcv_ack=1;
    }
}

float stab[8];
```

```
void sensor_Fun()
{
  Wire.begin(12,14);
  SHT21.begin(); delay(200);
  stab[0]=(float) SHT21.getTemperature();
  stab[1]=(float) SHT21.getHumidity();
}

void setup() {
  Serial.begin(9600);
  termID=(uint32_t)ESP.getEfuseMac();
  set_LoRa();
  LoRa.onReceive(onReceive);
  LoRa.onTxDone(onTxDone);
  LoRa_rxMode();
  delay(333);
}

void loop() {
  if (runEvery(cycle))
  {
    DT_t p;
    sensor_Fun();
    p.pack.did=(uint32_t)0;
    p.pack.sid=(uint32_t)termID;
    p.pack.sens[0]=stab[0]; p.pack.sens[1]=stab[1];
    p.pack.sens[2]=0; p.pack.sens[3]=0;
    p.pack.tout=(uint32_t)millis();
    LoRa_send(p.frame); // send a packet
    Serial.println("Send Packet!");
    //while(rcv_ack==0)
    delay(2000);
    if(rcv_ack)
      {
      Serial.printf("Go to deep sleep for:%d, sec=%d\n",rcv_cycle,millis()/1000); rcv_ack=0;
      esp_sleep_enable_timer_wakeup(1000*1000*rcv_cycle);  // timeout in seconds
      esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF); //
      esp_sleep_pd_config(ESP_PD_DOMAIN_MAX, ESP_PD_OPTION_OFF);
      delay(60); LoRa.end();
      delay(300); // necessary to stop LoRa modem
      esp_deep_sleep_start();
      Serial.println("This will never be printed");
      }
  }
}
```

Figure 8.2 Les mesures de courant pour le sommeil profond et le mode actif (en mA) pour l'exécution du code ci-dessus

8.4.2.2 Le code de la Gateway

```
#define GATEWAY
#include <SPI.h>
#include <LoRa.h>
#include "LoRa_Para.h"

typedef union
{
  uint8_t frame[32];    // data frame to send/receive data
  struct
  {
    uint32_t did;        // destination identifier chipID (4 lower bytes)
    uint32_t sid;        // source identifier chipID (4 lower bytes)
    float   sens[4];     // max 4 values - fields
    uint32_t  tout;      // optional timeout
    uint8_t pad[4];      // padding
  } pack;                // data packet
} DT_t;                  // DaTa type

RTC_DATA_ATTR uint32_t gwID=0, termID;  // gateway identifier stored in RTC memory
RTC_DATA_ATTR uint32_t rcv_pack=0;

void LoRa_send(uint8_t *frame) {
  LoRa_txMode();                        // set tx mode - normal mode
  LoRa.beginPacket();                   // start packet
  LoRa.write(frame,32);                  // add payload
  LoRa.endPacket(true);                 // finish packet and send it
}

void onReceive(int packetSize) {
  DT_t p;int i=0;
  if(packetSize==32)
    {
    while (LoRa.available())  {  p.frame[i]= LoRa.read(); i++; }
    Serial.println("Gateway Received Packet ");
    Serial.printf("T:%2.2f, H:%2.2f\n",p.pack.sens[0],p.pack.sens[1]);
    rcv_pack=1;
    }
}

void setup() {
  Serial.begin(9600);
  gwID=(uint32_t)ESP.getEfuseMac();
  set_LoRa();
  LoRa.onReceive(onReceive);
  LoRa.onTxDone(onTxDone);
  LoRa_rxMode();
}

void loop() {
  if (runEvery(1000)) { // repeat every 5000 millis
    DT_t p;
    p.pack.did=(uint32_t)termID;
    p.pack.sid=(uint32_t)gwID;
    p.pack.sens[0]=23.67; p.pack.sens[1]=67.67;
    p.pack.sens[2]=0; p.pack.sens[3]=0;
    p.pack.tout=(uint32_t)(10 + random(10,40));
    if(rcv_pack)
      {
      LoRa_send(p.frame); rcv_pack=0;
      Serial.printf("Send Packet with timeout=%d, sec=%d\n",p.pack.tout,millis()/1000);
      }/
  }
}
```

Remarque

Sachant que l'état profond réinitialise toutes les données de la mémoire SRAM, nous pouvons stocker l'indicateur **gwID** et autres éléments dans une mémoire active intégrée à l'unité RTC.

```
RTC_DATA_ATTR uint32_t gwID=0;  // gateway identifier stored in RTC memory
RTC_DATA_ATTR uint32_t cycle=8000, rcv_cycle, rcv_ack=0;
```

8.6 Résumé

Dans ce laboratoire, nous vous avons montré comment utiliser le sommeil profond avec l'ESP32 et les différentes façons de le réactiver.
Vous pouvez réactiver l'ESP32 à l'aide d'une minuterie, des broches tactiles ou d'un changement d'état **GPIO**.

Résumons ce que nous avons vu à propos de chaque source de réveil:

Minuterie de réveil (*Timer Wake Up*)
Pour activer le réveil du minuteur, vous utilisez la fonction :

```
esp_sleep_enable_timer_wakeup (time_in_us);
```

Utilisez la fonction **esp_deep_sleep_start()** pour démarrer le sommeil profond.

Réveil par broches tactiles (*Touch Wake Up*)
Pour utiliser les broches tactiles comme source de réveil, vous devez d'abord attacher des interruptions aux broches tactiles à l'aide de :

```
touchAttachInterrupt (Touchpin, callback, Threshold)
```

Ensuite, vous activez les broches tactiles comme source de réveil en utilisant :

```
esp_sleep_enable_touchpad_wakeup ()
```

Enfin, vous utilisez la fonction **esp_deep_sleep_start()** pour mettre l'ESP32 en mode sommeil profond.

Réveil externe (*External Wake Up*)

- Vous ne pouvez utiliser les GPIO RTC que comme réveil externe ;
- Vous pouvez utiliser deux méthodes différentes: **ext0** et **ext1** ;
  - **ext0** vous permet de réveiller l'ESP32 en utilisant une seule broche **GPIO**;
  - **ext1** vous permet de réveiller l'ESP32 à l'aide de plusieurs broches **GPIO**.

Modem externe (LoRa)
Pour minimiser la consommation électrique du modem LoRa, nous devons mettre le modem en mode veille et déconnecter le bus SPI.

8.6.1 A faire

1. Testez les exemples ci-dessus
2. Testez le nœud de terminal LoRa avec le nœud de passerelle LoRa avec un écran OLED
3. Utilisez un multimètre pour tester la consommation de courant en mode actif a et sommeil profond; calculer le courant moyen pendant 20 secondes en mode veille profonde et 2 secondes en mode actif. Quelle est l'autonomie de votre Terminal si vous utilisez une batterie LiPo 1500mAh

8.7 Annex – include file `ESP32_WakeUp.h`

The following is the include file **`ESP32_WakeUp.h`** providing 3 functions:

```
void print_wakeup_reason()
void print_wakeup_touchpad(), and
void print_GPIO_wake_up()
```

```
// ESP32_WakeUp.h
void print_wakeup_reason(){
  esp_sleep_wakeup_cause_t wakeup_reason;
  wakeup_reason = esp_sleep_get_wakeup_cause();
  switch(wakeup_reason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("External signal RTC_IO");
       break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("External RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("ULP program"); break;
    default : Serial.printf("Not caused by deep sleep: %d\n",wakeup_reason);
       break;
  }
}

void print_wakeup_touchpad(){
  touchPin = esp_sleep_get_touchpad_wakeup_status();
  switch(touchPin)
  {
    case 0  : Serial.println("Touch detected on GPIO 4"); break;
    case 1  : Serial.println("Touch detected on GPIO 0"); break;
    case 2  : Serial.println("Touch detected on GPIO 2"); break;
    case 3  : Serial.println("Touch detected on GPIO 15"); break;
    case 4  : Serial.println("Touch detected on GPIO 13"); break;
    case 5  : Serial.println("Touch detected on GPIO 12"); break;
    case 6  : Serial.println("Touch detected on GPIO 14"); break;
    case 7  : Serial.println("Touch detected on GPIO 27"); break;
    case 8  : Serial.println("Touch detected on GPIO 33"); break;
    case 9  : Serial.println("Touch detected on GPIO 32"); break;
    default : Serial.println("Wakeup not by touchpad"); break;
  }
}

void print_GPIO_wake_up(){
long long GPIO_reason = esp_sleep_get_ext1_wakeup_status();
  Serial.print("GPIO that triggered the wake up: GPIO ");
  Serial.println((log(GPIO_reason))/log(2), 0);
}
```

Table of Contents