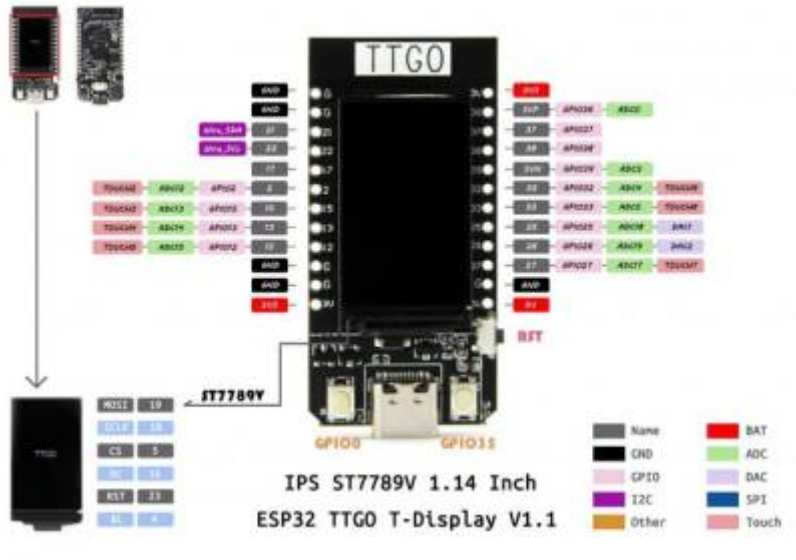
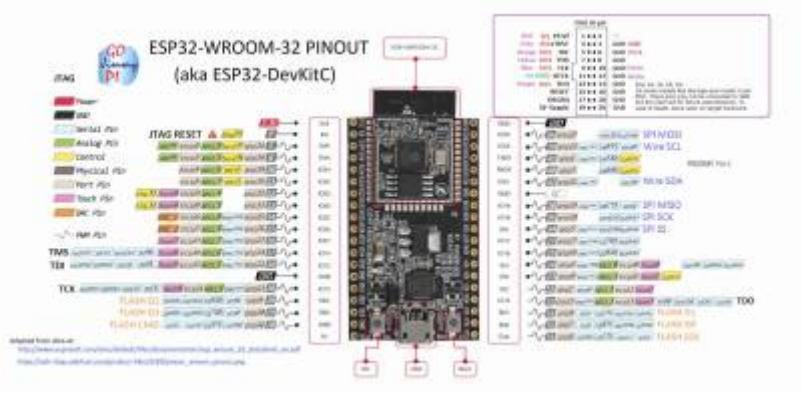
ESP 32 Alimentation Autonome

Brochages de quelques ESP32

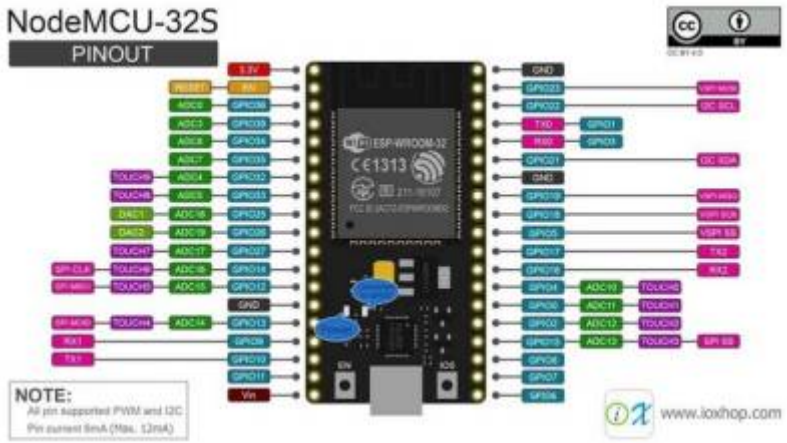
ESP32 et le TTGO T-Display



ESP32 DEVKITC



NodeMCU ESP32



Tension entrées et sorties

Carte de développement ESP32 Wemos D1 R32 :

- Compatible avec le brochage Arduino Uno
- Tension de fonctionnement :
  - 3,3 V via broche Vin
  - 5V via USB
  - 9-24V via fiche alimentation
- WIFI et Bluetooth
- 22 broches d'entrée / sortie numériques
- Interfaces I2C, SPI, UART, DAC (x2), ADC (x6)
- Connexion micro USB
- NB : La broche A0 est sensible au téléversement.

Module ESP32 NodeMCU

- Tension de fonctionnement :
  - 3,3 V via broche Vin
  - 5V via USB
- WIFI et Bluetooth
- 10 broches d'entrée / sortie numériques
- Interfaces I2C, SPI, UART, DAC, ADC
- Connexion micro USB

Tension maximum et recommendée

5.1 Absolute Maximum Ratings

Table 7: Absolute Maximum Ratings

Parameter	Symbol	Min	Max	Unit
Input low voltage	V_{IL}	-0.3	0.25×V_{IO}	V
Input high voltage	V_{IH}	0.75×V_{IO}	3.3	V
Input leakage current	I_{IL}	-	50	nA
Output low voltage	V_{OL}	-	0.1×V_{IO}	V
Output high voltage	V_{OH}	0.8×V_{IO}	-	V
Input pin capacitance	C_{pad}	-	2	pF
VDDIO	V_{IO}	1.8	3.3	V
Maximum drive capability	I_{MAX}	-	12	mA
Storage temperature range	T_{STR}	-40	150	°C

5.2 Recommended Operating Conditions

Table 8: Recommended Operating Conditions

Parameter	Symbol	Min	Typ	Max	Unit
Battery regulator supply voltage	V_{BAT}	2.8	3.3	3.6	V
I/O supply voltage	V_{IO}	1.8	3.3	3.6	V
Operating temperature range	T_{OPR}	-40	-	125	°C
CMOS low level input voltage	V_{IL}	0	-	0.3 x V_{IO}	V
CMOS high level input voltage	V_{IH}	0.7 x V_{IO}	-	V_{IO}	V
CMOS threshold voltage	V_{TH}	-	0.5 x V_{IO}	-	V

Consommation et puissance fournie

Power mode	Comment	Power consumption
Active mode (RF working)	Wi-Fi Tx packet 13 dBm ~ 21 dBm	160 ~ 260 mA
	Wi-Fi / BT Tx packet 0 dBm	120 mA
	Wi-Fi / BT Rx and listening	80 ~ 90 mA
	Association sleep pattern (by light-sleep)	0.9 mA@DTIM3, 1.2 mA@DTIM1
Modem-sleep mode	The CPU is powered on.	Max speed: 20 mA
		Normal: 5 ~ 10 mA
		Slow speed: 3 mA
Light-sleep mode	-	0.8 mA
Deep-sleep mode	The ULP-coprocessor is powered on.	0.5 mA
	ULP sensor-monitored pattern	25 μA @1% duty
	RTC timer + RTC memories	20 μA
Hibernate mode	RTC timer only	2.5 μA

Comme on peut le voir la consommation électrique est très réduite, en alimentant le circuit avec un pack de 2 piles AA 2500 mAh par exemple, on a environ un peu moins de 7h d'autonomie en émission continue à puissance maximale, 60 jours en veille avec les modules radio actifs, et environ 80 ans en hibernation ! Bien évidmment, à faible consommation électrique l'autonomie sera surtout définie par l'auto-décharge des accumulateurs.

Comment alimenter un ESP32

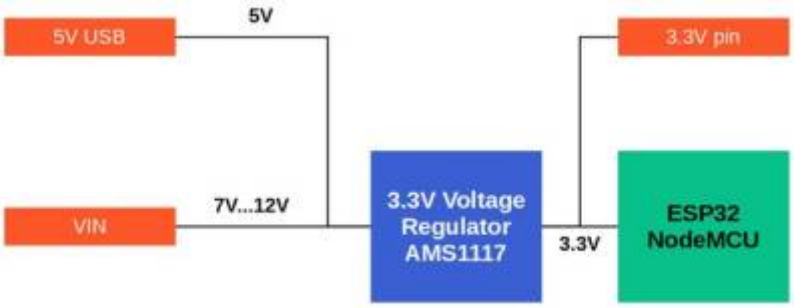
Pour alimenter votre kit de développement ESP32, vous avez trois options:

1. -Via le port USB.
2. -Utilisation d'une tension non régulée entre 5V et 12V, connectée aux broches 5V et GND. Cette tension est régulée à bord.
3. -Utilisation d'une tension régulée de 3,3 V, connectée aux broches 3,3 V et GND. Soyez très prudent avec cela: ne dépassez pas la limite de 3,3V, ou votre module ESP32 sera endommagé.

Attention : soyez très, très prudent de n'utiliser qu'une seule de ces options à la fois. Par exemple, n'alimentez pas votre kit de développement ESP32 via la broche 5V en utilisant une entrée 10V alors

qu'en même temps vous avez le module connecté à votre ordinateur via USB. Cela endommagera sûrement votre module, et peut-être même votre ordinateur. Différents niveaux de tension de la carte microcontrôleur ESP32 Avant de pouvoir analyser différentes batteries en combinaison avec le microcontrôleur ESP32, nous devons comprendre qu'il existe différents niveaux de tension sur la carte ESP32 NodeMCU. L'image suivante montre un schéma simplifié des niveaux de tension et des composants importants.

Différents niveaux de tension de la carte microcontrôleur ESP32



Sur l'image, vous voyez que la connexion USB 5V et la broche VIN sont connectées à un régulateur de tension 3,3V, qui transforme la tension d'entrée entre 5V et 12V en une tension de sortie

Mode USB

La première possibilité et aussi la plus simple pour une alimentation électrique est le câble USB 5V. Mais comme l'ESP32 fonctionne à 3,3 V, il existe un régulateur de tension intégré pour transformer le 5 V de la connexion USB au 3,3 V souhaité. La broche 3,3 V du PCB NodeMCU est également alimentée par cette connexion.

Mode alimentation externe

La deuxième possibilité consiste à utiliser la broche VIN du NodeMCU comme entrée pour l'alimentation. Le régulateur de tension AMS1117 a une tension d'entrée maximale de 15V, mais dans ce cas, le régulateur produit beaucoup de chaleur car le régulateur n'a pas de dissipateur thermique ou de ventilateur de refroidissement pour la dissipation thermique. Par conséquent, une tension comprise entre 7V et 12V est recommandée lorsque l'ESP32 est alimenté par la broche VIN.

Mode Piles/Batteries

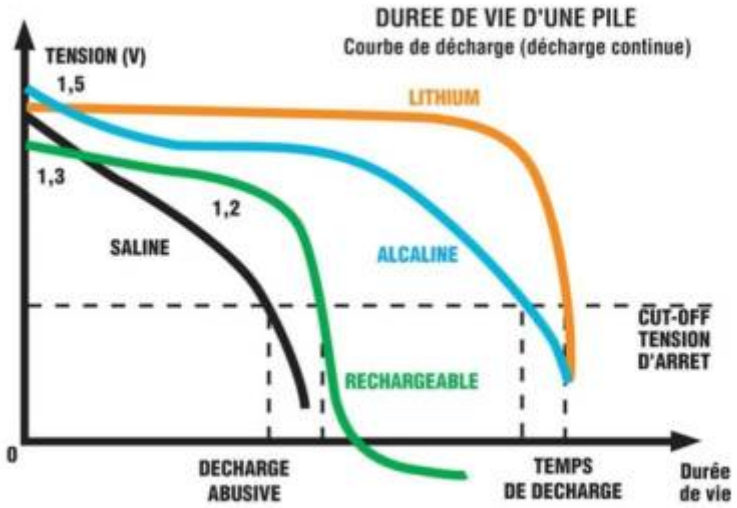
Pile alcaline AA pour ESP32

Critères des piles alcalines AA	spécifications
Tension de décharge minimale	1V
Tension de travail	1,5 V
Tension de charge maximale	1,65 V
Nombre de recharges	500
Densité d'énergie	80 Wh / kg

Les piles alcalines AA ont une tension nominale de 1,5 V et si vous en connectez deux en série, vous obtenez une tension nominale de 3 V. Vous pouvez connecter deux piles alcalines AA directement à la broche 3,3 V du NodeMCU, mais le courant fourni par les piles alcalines AA n'est que de 50 mA par pile. Connecté en série, vous obtenez toujours un courant global de 50 mA.

Malheureusement, l'ESP32 NodeMCU peut tirer jusqu'à 300 mA lors du démarrage. Lorsque l'ESP32 démarre, il tire tellement de courant des piles alcalines AA que la tension chute entièrement à zéro, réinitialisant / écrasant l'ESP32.

En résumé, je ne peux pas alimenter l'ESP32 NodeMCU avec 2 piles AA .



Batterie LiFePO4 pour ESP32

Plusieurs options pour alimenter l'ESP32 avec différents types de batteries. Ceci est particulièrement intéressant lorsque vous souhaitez construire un projet indépendant d'une alimentation électrique normale comme une station météo extérieure.

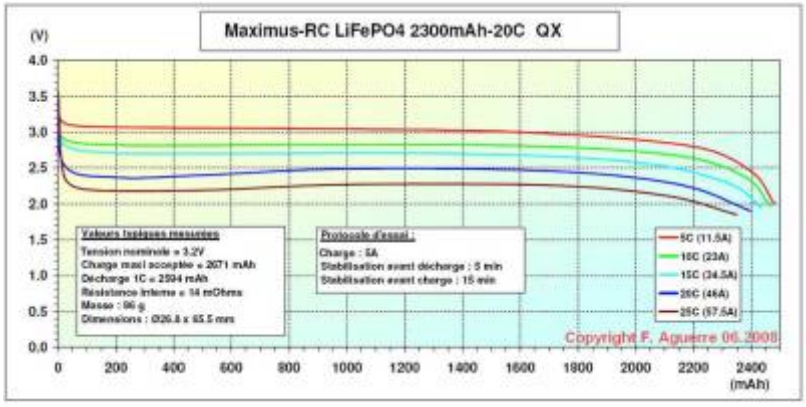
Si la tension maximale de la batterie est supérieure à la tension maximale de l'ESP32 (3,6V), vous devez utiliser un régulateur de tension pour réduire la tension à 3,3V. La sortie du régulateur de tension est alors connectée à la broche 3,3 V de la carte ESP32.

Ma recommandation pour une alimentation par batterie est la batterie LiFePO4, car vous n'avez pas besoin de régulateur de tension supplémentaire entre l'ESP32 et la batterie et elles sont rechargeables. Les batteries LiFePO4 ont également une capacité allant jusqu'à 6000 mAh, similaire aux batteries LiPo et Li-ion, ce qui confère à votre projet une longue durée de vie en combinaison avec un mode d'alimentation qui réduit la consommation d'énergie au minimum.

Critères de la batterie LiFePO4	**spécifications**
Tension de décharge minimale	2,5V
Tension de travail	3,0 V à 3,2 V
Tension de charge maximale	3,65V
Nombre de recharges	5000
Densité d'énergie	90 Wh / kg... 160 Wh / kg

La batterie lithium fer phosphate (batterie LiFePO4) a une tension nominale de 3,2 V et une tension maximale de 3,65 V. Le principal avantage d'une batterie LiFePO4 est la courbe de décharge très plate de sorte que la tension chute très lentement pendant le processus de décharge. Étant donné que la tension maximale de la batterie au lithium fer phosphate n'est avec 3,65 V que légèrement supérieure à la tension de fonctionnement maximale de l'ESP32 avec 3,6 V, vous pouvez connecter ce type de batterie directement avec la broche 3,3 V du microcontrôleur.

En résumé **une batterie LiFePO4 convient très bien à l'ESP32** et surtout lorsque votre objectif principal est d'alimenter votre circuit pendant un temps maximum. Si tel est le cas, je recommande d'alimenter l'ESP32 avec un LiFePO4 sur la broche 3,3V. L'inconvénient est qu'il est très compliqué de charger la batterie pendant son utilisation. Actuellement, je n'ai pas de solution à ce problème. La solution la plus simple serait d'avoir deux batteries LiFePO4 que vous pouvez changer rapidement et un chargeur de batterie externe.



Batterie LiPo et batterie Li-ion pour ESP32

Critères des batteries LiPo et Li-ion	spécifications
Tension de décharge minimale	2,7 V à 3,0 V
Tension de travail	3,7 V
Tension de charge maximale	4,2V
Nombre de recharges	5000
Densité d'énergie	100 Wh / kg... 265 Wh / kg

La tension maximale des batteries LiPo et Li-ion est d'environ 4,2 V et trop élevée pour se connecter directement à la broche 3,3 V. Par conséquent, vous avez besoin d'un régulateur à faible perte de charge ou LDO qui réduit la tension de la batterie à 3,3 V. Le MCP1725T-3302E / MC LDO s'adapte parfaitement à l'ESP32 en combinaison avec une batterie LiPo ou Li-ion. Dans le dernier chapitre de cet article, vous trouverez une explication détaillée sur l'utilisation du régulateur LDO en combinaison avec une batterie et l'ESP32.

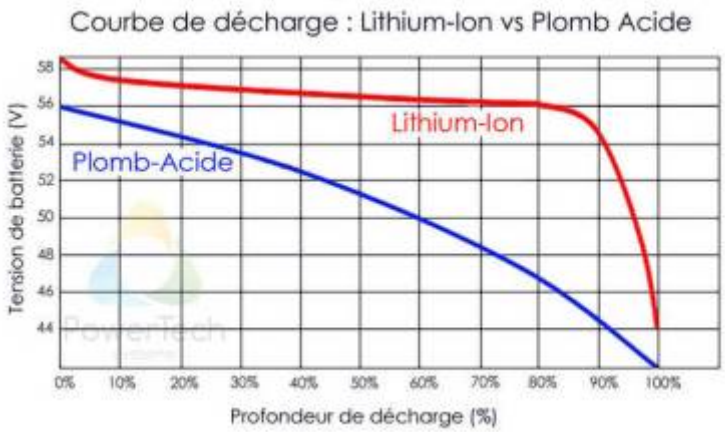
Les batteries LiPo et Li-ion associées à un régulateur de tension à faible chute de tension conviennent parfaitement pour alimenter votre ESP32. Surtout si vous souhaitez charger la batterie pendant que votre circuit est en marche,

Il existe des cartes EPS32 spéciales avec un connecteur JST où vous connectez la batterie LiPo directement à votre carte ESP32. Si vous souhaitez charger la batterie, il vous suffit de brancher le câble micro USB sur l'EPS32. La connexion USB alimente non seulement l'EPS32 mais charge également la batterie LiPo. Les cartes suivantes ont le connecteur JST ainsi qu'un chargeur LiPo à bord:

- Adafruit HUZZAH32
- Sparkfun ESP32 Thing Plus

- FireBeetle ESP32

Batteries rechargeables Li-ion 18650

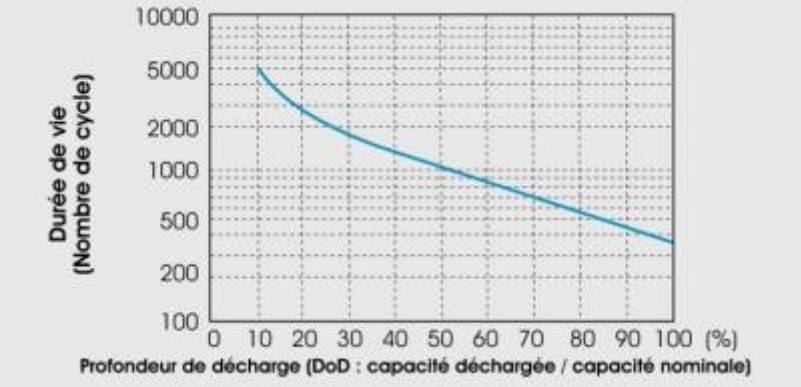


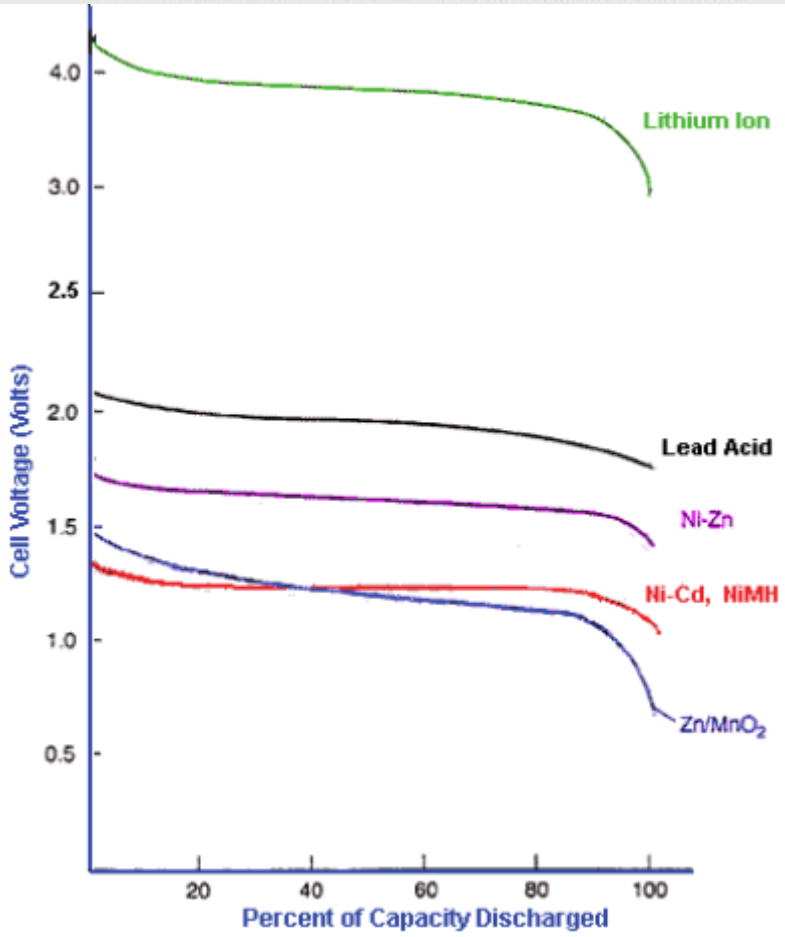
Piles AAA NiMH pour ESP32

Critères des piles AAA NiMH	**spécifications**
Tension de décharge minimale	0,8V
Tension de travail	1,2V à 1,25 V
Tension de charge maximale	1,4V
Nombre de recharges	1000
Densité d'énergie	60 Wh / kg… 120 Wh / kg

Si vous souhaitez acheter des piles AAA, assurez-vous d'acheter des piles NiMH, car elles sont rechargeables et ont la capacité la plus élevée et une tension nominale de 1,2 V… 1,25 V par pile. La combinaison avec quatre piles AAA NiMH résulte en une tension de fonctionnement de 4,8 V… 5 V qui est supérieure à la tension de fonctionnement maximale de l'ESP32 avec 3,6 V. Tout comme les batteries LiPo et Li-ion, vous pouvez utiliser quatre batteries AAA NiMH en combinaison avec un régulateur LDO qui réduit la tension d'entrée à 3,3V. Avec la tension réduite, vous pouvez connecter l'alimentation à la broche 3,3V de l'ESP32.

Par rapport aux batteries LiPo et Li-ion, les batteries NiMH nécessitent la même connexion au microcontrôleur mais ont une densité d'énergie plus faible et **il n'est pas recommande d'utiliser les batteries NiMH .**





Pile bloc alcaline 9V pour ESP32

Critères de la pile alcaline 9 V	spécifications
Tension de décharge minimale	6V

Critères de la pile alcaline 9 V	spécifications
Tension de travail	9 V
Tension de charge maximale	9,9V
Nombre de recharges	500
Densité d'énergie	80 Wh/kg

Avec une batterie bloc 9V, vous pouvez utiliser la broche VIN du NodeMCU, qui est connectée en interne avec le régulateur de tension 3.3V AMS1117. Par conséquent, vous n'avez besoin d'aucun composant externe. Mais comme l'ESP32 n'a besoin que de 3,3 V, vous êtes surpuissant en ce qui concerne la tension. Étant donné qu'une batterie bloc alcaline 9 V n'est rien d'autre que 6 piles alcalines AA connectées en série, vous avez la même courbe de décharge par paliers, ce qui entraîne une courte durée de vie de votre système alimenté par batterie.

Il n'est pas recommandé d'utiliser une pile alcaline 9V

Régulateur de tension à faible chute pour ESP32

Le MCP1725T-3302E / MC s'adapte parfaitement à l'ESP32 en combinaison avec des batteries ayant une tension maximale supérieure à 3,6V. Les tableaux suivants présentent les principes fondamentaux de la fiche technique LDO et expliquent pourquoi ces principes fondamentaux correspondent parfaitement à l'ESP32.

MCP1725T-3302E / MC	Valeurs	Explications
Tension de sortie	3,3 V	Le régulateur a besoin d'une tension de sortie égale à la tension de fonctionnement de l'ESP32 qui est de 3,3V.
Courant de sortie	500 mA	Il est également important que le LDO ait un courant de sortie de 500mA car l'ESP32 a besoin d'environ 450mA pendant la communication WiFi et dans la fiche technique de l'ESP32, un courant de sortie de 500mA est recommandé.
Tension d'entrée maximale	6V	Avec une tension d'entrée maximale de 6V, nous sommes en mesure de combiner le LDO avec les batteries 3,7V LiPo et Li-ion ainsi que les batteries 5V AAA NiMH.
Tension d'entrée minimale	2,3 V	La tension d'entrée minimale doit correspondre à la tension de fonctionnement minimale de l'ESP32 qui est de 2,3 V.

Régulateur 3,3 V S7V8F3 500 mA à 1 A Régulateur élévateur/abaisseur permettant de délivrer une tension de 3,3 Vcc à partir d'une tension de 2,7 à 11,8 Vcc. Un connecteur droit ou coudé est à souder soi-même en fonction de l'utilisation. La tension de sortie est indépendante de la tension d'entrée.

Remarque: en utilisation, le module peut devenir très chaud.

S7V8F3	valeurs
Alimentation:	2,7 à 11,8 Vcc
Tension de sortie:	3,3 Vcc
Courant de sortie:	500 mA à 1 A en fonction de la tension d'entrée
Dimensions:	17 x 12 x 3 mm
Référence fabricant:	2122

Régulateur 3,3 V S7V8F3 - 1

ou

Régulateur 3,3 V S7V8F3 -2

ESP8266 et ESP32 sur batterie

ESP32 sur batterie

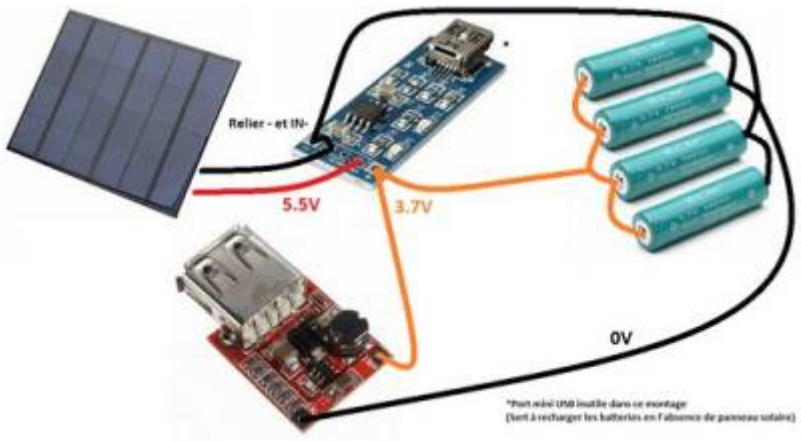
Alimentation solaire

Chargement de la batterie LiPo à partir d'un panneau solaire 1W 5V

chargement de la batterie Lipo à partir d 'un panneau solaire

U=RI | Comment réaliser un chargeur solaire?

Comment réaliser un chargeur solaire?



Mode veille sur ESP32

ESP32 Deep Sleep

ESP32 Deep Sleep

Doc sur les Sleep Modes ESP32 sur site EXPRESSIF EN

Le sommeil profond

Le sommeil profond a un fonctionnement différent de celui de l'ESP8266.

L'ESP32 est capable de faire la distinction entre plusieurs sources de réveil :

- un réveil par une GPIO (ext0)
- un réveil par plusieurs GPIOs (ext1)
- un réveil par le touchpad
- un réveil par RTC

Il est possible d'activer l'une ou l'autre ou plusieurs.

Réveil par une GPIO (0, 2, 4, 12 à 15, 25 à 27, 32 à 39) :

1.ino

```
esp_sleep_enable_ext0_wakeup(gpio, state);
```

Dans ce mode, les GPIOs peuvent bénéficier de résistances internes de pull-up ou pull-down :

2.ino

```
#include <driver/rtc_io.h>
rtc_gpio_pullup_en(gpio);
rtc_gpio_pulldown_en(gpio);
```

Réveil par plusieurs GPIOs (32 à 39) :

3.ino

```
esp_sleep_enable_ext1_wakeup(gpios, state);
```

Dans ce mode, les GPIOs devront être équipées de pull-up ou pull-down matérielles externes.

Réveil par le touchpad :

4.ino

```
touchAttachInterrupt(pad, callback, threshold);
esp_sleep_enable_touchpad_wakeup();
```

Réveil par la RTC :

4.ino

```
esp_sleep_enable_timer_wakeup(µs);
```

Activation du mode deep-sleep :

5.ino

```
esp_deep_sleep_start();
```

Lors du réveil la fonction esp_sleep_get_wakeup_cause() sera appelée pour connaître la cause du réveil :

6.ino

```
switch (esp_sleep_get_wakeup_cause()) {
    case ESP_SLEEP_WAKEUP_EXT0:
      Serial.println("Wakeup by EXT0");
      break;
    case ESP_SLEEP_WAKEUP_EXT1:
      Serial.println("Wakeup by EXT1");
      break;
    case ESP_SLEEP_WAKEUP_TIMER:
      Serial.println("Wakeup by RTC");
      break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD:
      Serial.println("Wakeup by TouchPad");
      break;
}
```

Si le réveil par plusieurs GPIOs a été activé, il est possible de déterminer quelle GPIO a provoqué le réveil :

7.ino

```
uint64_t wakeupBit = esp_sleep_get_ext1_wakeup_status();
if (wakeupBit & GPIO_SEL_33) {
  // GPIO 33 woke up
}
else if (wakeupBit & GPIO_SEL_34) {
  // GPIO 34
}
```

Et enfin, si le réveil par le touchpad a été activé, il est possible de déterminer quelle touche a provoqué le réveil :

8.ino

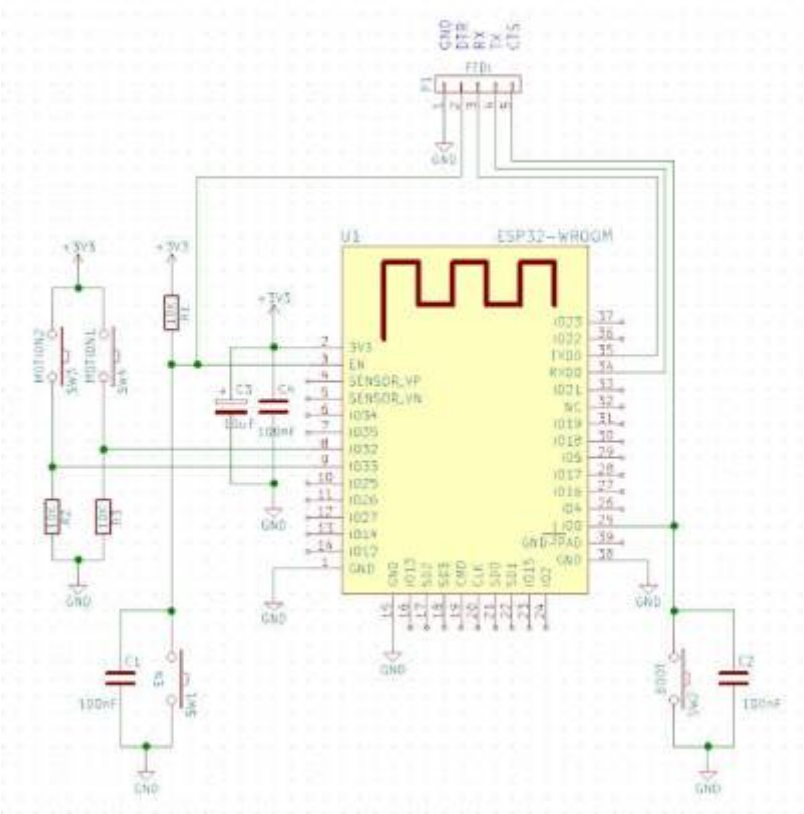
```
touch_pad_t pin = esp_sleep_get_touchpad_wakeup_status();
switch(touchPin) {
  case 0  : Serial.println("Touch detected on GPIO 4"); break;
```

```
  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;
}
```

Nous allons partir du même exemple de client mail que pour l'ESP8266 :

- un capteur de température (réveil cyclique)
- un ou deux capteurs de passage PIR (réveil par GPIO)

Voici un petit schéma :



Comme vous pouvez le constater, par rapport au même schéma utilisant un ESP8266, celui-ci est beaucoup plus simple.

Les boutons poussoirs simulent les PIRs. Si de vrais capteurs PIR sont utilisés les résistances R2 et R3 sont inutiles.

Bien sûr, si l'on utilise une ESP-WROOM-32, seuls les boutons poussoirs doivent être câblés.

Si un seul bouton poussoir est utilisé sur GPIO33 : Dans ce cas, connecter un bouton poussoir entre 3.3V et GPIO33, sans résistance de pull-down.

Si deux boutons poussoirs sont utilisés sur les GPIO32 et GPIO33 : Dans ce cas, connecter deux boutons poussoirs entre 3.3V et GPIO32 et GPIO33, avec 2 résistances de pull-down.

Réveil par un ou deux PIR

Le problème est simplifié par rapport à l'ESP8266. Il est possible de spécifier sur quelle GPIO le processeur peut être réveillé. Dans l'exemple, GPIO_32 et GPIO_33 sont utilisées.

Possibilités Contrairement à ce qui se passait sur l'ESP8266 le front montant ou descendant de la source de réveil peut être choisi. Il n'y aura pas de modification hardware à apporter pour inverser le signal.

On pourra également activer des résistances internes de pull-up ou pull-down en cas de besoin.

Également il n'y aura pas besoin de maintenir le signal sur la ou les GPIOs si les impulsions de réveil sont courtes.

Réveil périodique

Ici également, il n'y a pas de modification hardware à réaliser pour prendre en compte le réveil par la RTC.

Le sketch suivant est prévu pour fonctionner dans deux modes :

Un seul bouton poussoir sur la GPIO33 : Dans ce cas, connecter un bouton poussoir entre 3.3V et GPIO33, sans résistance de pull-down. Commenter la ligne suivante :

9.ino

```
//#define EXT1_WAKEUP
```

Deux boutons poussoirs sur les GPIO32 et GPIO33 : Dans ce cas, connecter deux boutons poussoirs entre 3.3V et GPIO32 et GPIO33, avec 2 résistances de pull-down. Décommenter la ligne suivante :

10.ino

```
#define EXT1_WAKEUP
```

Le sketch :

11.ino

```
#include <WiFi.h>
#include <rom/rtc.h>
```

```
#include <driver/rtc_io.h>
#include <OneWire.h>
#include <DallasTemperature.h>

#define SMTP_PORT     587
#define ONE_WIRE_PIN  5
#define SLEEP_TIME    (30*60)

#define EXT1_WAKEUP

#ifdef EXT1_WAKEUP
#define BUTTON_PIN_BITMASK 0x300000000
#endif

const char* ssid = "Livebox-XXXX";
const char* password = "XXXXXXXXXXXXXXXXXXXXXXXXXX";
char server[] = "smtp.xxxxxx.xx";
// Change to your base64, ASCII encoded user
const char userID[] = "XxXxXxXxXxXx";
// change to your base64, ASCII encoded password
const char userPWD[] = "YyYyYyYyYyYyYy";
// sender
const char sender[] = "sender@xxxxxx.xx";
// recipent
const char recipient[] = "xxxxx.xxxxxxxx@gmail.com";

WiFiClient client;
OneWire oneWire(ONE_WIRE_PIN  );
DallasTemperature DS18B20(&oneWire);

void setup()
{
  byte ret;
  uint64_t wakeup_pin_mask;

  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)   {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi Connected");
  Serial.print("IPess: ");
  Serial.println(WiFi.localIP());
  switch (esp_sleep_get_wakeup_cause()) {
    case ESP_SLEEP_WAKEUP_EXT0:
      ret = sendEmail("**** Motion Detected ****");
      break;
```

```
    case ESP_SLEEP_WAKEUP_EXT1:
#ifdef EXT1_WAKEUP
      wakeup_pin_mask = esp_sleep_get_ext1_wakeup_status();
      if (wakeup_pin_mask & GPIO_SEL_32) {
        ret = sendEmail("**** Motion1 Detected ****");
      }
      if (wakeup_pin_mask & GPIO_SEL_33) {
        ret = sendEmail("**** Motion2 Detected ****");
      }
      else {
      }
      break;
#endif
    case ESP_SLEEP_WAKEUP_TIMER:
      char temp[6];
      char s[32];
      dtostrf(getTemperature(), 5, 2, temp);
      sprintf(s, "**** temperture is %s ****", temp);
      ret = sendEmail(s);
      break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD:
      ret = sendEmail("**** TOUCH ****");
      break;
    case ESP_SLEEP_WAKEUP_ULP:
      ret = sendEmail("**** ULP ****");
      break;
    default:
      ret = sendEmail("**** Just Started ****");
      break;
  }
  Serial.print("Going into deep sleep for ");
  Serial.print(SLEEP_TIME);
  Serial.println(" seconds");
  delay(50);
  esp_sleep_enable_timer_wakeup(1000000L * SLEEP_TIME);
#ifdef EXT1_WAKEUP
  esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK,
ESP_EXT1_WAKEUP_ANY_HIGH);
  // internal pull-ups not available !!!
#else
  esp_sleep_enable_ext0_wakeup(GPIO_NUM_33, 1);
  rtc_gpio_pulldown_en(GPIO_NUM_33);
#endif
  esp_deep_sleep_start();
}

void loop()
{
}

byte sendEmail(const char *data)
```

```
{
  byte thisByte = 0;
  byte respCode;

  if (client.connect(server, SMTP_PORT) == 1) {
    Serial.println(F("connected"));
  } else {
    Serial.println(F("connection failed"));
    return 0;
  }
  if (!recv()) return 0;

  Serial.println(F("Sending HELLO"));
  client.println("EHLO www.example.com");
  if (!recv()) return 0;
  Serial.println(F("Sending auth login"));
  client.println("auth login");
  if (!recv()) return 0;
  Serial.println(F("Sending User"));
  client.println(userID);
  if (!recv()) return 0;
  Serial.println(F("Sending Password"));
  client.println(userPWD);
  if (!recv()) return 0;
  Serial.print(F("Sending From ")); Serial.println(sender);
  client.print(F("MAIL From: ")); client.println(sender);
  if (!recv()) return 0;
  Serial.print(F("Sending To ")); Serial.println(recipient);
  client.print(F("RCPT To: ")); client.println(recipient);
  if (!recv()) return 0;
  Serial.println(F("Sending DATA"));
  client.println(F("DATA"));
  if (!recv()) return 0;
  Serial.println(F("Sending email"));
  client.print(F("To: ")); client.println(recipient);
  client.print(F("From: ")); client.println(sender);
  client.println(F("Subject: My first Email from ESP32\r\n"));
  client.print(F("From ESP32 N° "));
  uint64_t chipID = ESP.getEfuseMac();
  client.println((uint16_t)(chipID >> 32), HEX);
  Serial.println(data);
  client.println(data);
  client.println(F("."));
  if (!recv()) return 0;
  Serial.println(F("Sending QUIT"));
  client.println(F("QUIT"));
  if (!recv()) return 0;
  client.stop();
  Serial.println(F("disconnected"));
```

```
  return 1;
}

byte recv()
{
  byte respCode;
  byte thisByte;
  int loopCount = 0;
  while (!client.available()) {
    delay(1);
    loopCount++;
    if (loopCount > 10000) {
      client.stop();
      Serial.println(F("\r\nTimeout"));
      return 0;
    }
  }
  respCode = client.peek();
  while (client.available()) {
    thisByte = client.read();
    Serial.write(thisByte);
  }

  if (respCode >= '4') {
    //  efail();
    return 0;
  }
  return 1;
}

float getTemperature() {
  float tempC;
  do {
    DS18B20.requestTemperatures();
    tempC = DS18B20.getTempCByIndex(0);
    delay(100);
  } while (tempC == 85.0 || tempC == (-127.0));
  return tempC;
}
```

Le code est simplifié par rapport à celui de l'ESP8266.

Il vous faudra bien sûr remplacer certaines valeurs (ssid, password, etc.) comme dans l'exemple de l'ESP8266.

La directive suivante permet d'attendre un réveil sur les deux GPIOs 32 et 33 :

11.ino

```
#define EXT1_WAKEUP
```

Si la directive est commentée, seule la GPIO 33 est surveillée :

12.ino

```
// #define EXT1_WAKEUP
```

L'IDE ARDUINO

Il vous faudra bien entendu installer le support ESP32 :

https://github.com/espressif/arduino-esp32/blob/master/docs/arduino-ide/boards_manager.md

Il vous faut aussi installer deux librairies :

https://github.com/PaulStoffregen/OneWire.git

https://github.com/milesburton/Arduino-Temperature-Control-Library.git

De préférence installez la dernière version.

Documentation en ligne sleep modes pour ESP32 en EN

ICI EN

Ci-dessous traduction Google .... en FR

Modes de veille

L'ESP32 est capable de modes d'économie d'énergie de veille légère et de veille prolongée. En mode veille légère, les périphériques numériques, la plupart de la RAM et les processeurs sont synchronisés par horloge et la tension d'alimentation est réduite. À la sortie du sommeil léger, les périphériques et les processeurs reprennent leur fonctionnement, leur état interne est préservé.

En mode veille profonde, les processeurs, la plupart de la RAM et tous les périphériques numériques qui sont cadencés à partir d'APB_CLK sont mis hors tension. Les seules parties de la puce qui peuvent encore être mises sous tension sont: le contrôleur RTC, les périphériques RTC (y compris le coprocesseur ULP) et les mémoires RTC (lentes et rapides).

Le réveil à partir des modes de sommeil profond et léger peut être effectué à l'aide de plusieurs sources. Ces sources peuvent être combinées, dans ce cas, la puce se réveillera lorsque l'une des sources est déclenchée. Les sources de réveil peuvent être activées à l'aide des esp_sleep_enable_X_wakeupAPI

et peuvent être désactivées à l'aide de l' esp_sleep_disable_wakeup_source()API. La section suivante décrit ces API en détail. Les sources de réveil peuvent être configurées à tout moment avant d'entrer en mode veille légère ou profonde.

De plus, l'application peut forcer des modes de mise hors tension spécifiques pour les périphériques RTC et les mémoires RTC à l'aide de l'esp_sleep_pd_config()API. Une fois les sources de réveil configurées, l'application peut entrer en mode veille à l'aide des API

esp_light_sleep_start()

ou esp_deep_sleep_start()

À ce stade, le matériel sera configuré en fonction des sources de réveil demandées et le contrôleur RTC mettra hors tension ou hors tension les CPU et les périphériques numériques.

WiFi / BT et modes veille

En modes veille profonde et veille légère, les périphériques sans fil sont mis hors tension. Avant d' entrer dans un sommeil profond ou modes de sommeil léger, les applications doivent désactiver le WiFi et BT en utilisant des appels appropriés (esp_bluedroid_disable(),esp_bt_controller_disable(), esp_wifi_stop()).

Les connexions WiFi et BT ne seront pas maintenues en veille profonde ou en veille légère, même si ces fonctions ne sont pas appelées.

Si la connexion WiFi doit être maintenue, activez la mise en veille du modem WiFi et activez la fonction de veille automatique légère (voir API de gestion de l'alimentation). Cela permettra au système de se réveiller automatiquement lorsque le pilote WiFi l'exige, maintenant ainsi la connexion au point d'accès.

Sources de réveil

Minuteur

Le contrôleur RTC a une minuterie intégrée qui peut être utilisée pour réveiller la puce après un laps de temps prédéfini. L'heure est spécifiée avec une précision de l'ordre de la microseconde, mais la résolution réelle dépend de la source d'horloge sélectionnée pour RTC SLOW_CLK. Voir le chapitre «Réinitialisation et horloge» du Manuel de référence technique ESP32 pour plus de détails sur les options d'horloge RTC.

Ce mode de réveil ne nécessite pas la mise sous tension des périphériques RTC ou des mémoires RTC pendant le sommeil.

esp_sleep_enable_timer_wakeup() La fonction peut être utilisée pour activer le réveil du sommeil profond à l'aide d'une minuterie.

Pavé tactile

Le module RTC IO contient une logique pour déclencher le réveil lorsqu'une interruption du capteur tactile se produit. Vous devez configurer l'interruption du pavé tactile avant que la puce ne commence le sommeil profond.

Les révisions 0 et 1 de l'ESP32 ne prennent en charge ce mode de réveil que lorsque les périphériques RTC ne sont pas forcés d'être mis sous tension (c'est-à-dire que ESP_PD_DOMAIN_RTC_PERIPH doit être défini sur ESP_PD_OPTION_AUTO).

esp_sleep_enable_touchpad_wakeup() peut être utilisée pour activer cette source de réveil.

Réveil externe (ext0)

Le module RTC IO contient une logique pour déclencher le réveil lorsque l'un des GPIO RTC est défini sur un niveau logique prédéfini. RTC IO fait partie du domaine d'alimentation des périphériques RTC, de sorte que les périphériques RTC resteront sous tension pendant la veille prolongée si cette source de réveil est demandée.

Etant donné que le module RTC IO est activé dans ce mode, des résistances de rappel ou de réduction internes peuvent également être utilisées. Ils doivent être configurés par l'application à l'aide des fonctions rtc_gpio_pullup_en() et rtc_gpio_pulldown_en(), avant d'appeler esp_sleep_start().

Dans les révisions 0 et 1 de l'ESP32, cette source de réveil est incompatible avec les sources de réveil ULP et tactile.

esp_sleep_enable_ext0_wakeup() peut être utilisée pour activer cette source de réveil.

Attention

Après le réveil, le pad IO utilisé pour le réveil sera configuré comme RTC IO. Avant d'utiliser ce pad comme GPIO numérique, reconfigurez-le en utilisant la rtc_gpio_deinit(gpio_num)fonction.

Réveil externe (ext1)

Le contrôleur RTC contient une logique pour déclencher le réveil à l'aide de plusieurs GPIO RTC. L'une des deux fonctions logiques peut être utilisée pour déclencher le réveil:

- se réveiller si l'une des broches sélectionnées est haute (ESP_EXT1_WAKEUP_ANY_HIGH)
- se réveiller si toutes les broches sélectionnées sont faibles (ESP_EXT1_WAKEUP_ALL_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. Cependant, si les périphériques RTC sont mis hors tension, les résistances de rappel internes et de réduction seront désactivées. Pour utiliser des résistances pullup ou pulldown internes, demandez au domaine d'alimentation des périphériques RTC de rester allumé pendant le sommeil et configurez les résistances pullup / pulldown à l'aide des rtc_gpio_fonctions, avant d'entrer en veille:

- esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_ON);
- gpio_pullup_dis(gpio_num);
- gpio_pulldown_en(gpio_num);

Attention

Après le réveil, le (s) pad (s) IO utilisé (s) pour le réveil sera configuré comme RTC IO. Avant d'utiliser ces pads comme GPIO numériques, reconfigurez-les à l'aide de la rtc_gpio_deinit(gpio_num)fonction.

esp_sleep_enable_ext1_wakeup() peut être utilisée pour activer cette source de réveil.

Réveil du coprocesseur ULP

Le coprocesseur ULP peut fonctionner pendant que la puce est en mode veille et peut être utilisé pour interroger des capteurs, surveiller les valeurs de l'ADC ou des capteurs tactiles et réveiller la puce lorsqu'un événement spécifique est détecté. Le coprocesseur ULP fait partie du domaine d'alimentation des périphériques RTC et exécute le programme stocké dans la mémoire lente RTC. La mémoire lente RTC sera allumée pendant le sommeil si ce mode de réveil est demandé. Les périphériques RTC seront automatiquement mis sous tension avant que le coprocesseur ULP ne commence à exécuter le programme; une fois le programme arrêté, les périphériques RTC sont automatiquement mis hors tension.

Les révisions 0 et 1 de l'ESP32 ne prennent en charge ce mode de réveil que lorsque les périphériques RTC ne sont pas forcés d'être mis sous tension (c'est-à-dire que ESP_PD_DOMAIN_RTC_PERIPH doit être défini sur ESP_PD_OPTION_AUTO).

esp_sleep_enable_ulp_wakeup() peut être utilisée pour activer cette source de réveil.

Réveil GPIO (veille légère uniquement)

En plus des sources de réveil EXT0 et EXT1 décrites ci-dessus, une autre méthode de réveil à partir d'entrées externes est disponible en mode veille légère. Avec cette source de réveil, chaque broche peut être configurée individuellement pour déclencher le réveil à un niveau haut ou bas à l'aide de la gpio_wakeup_enable()fonction. Contrairement aux sources de réveil EXT0 et EXT1, qui ne peuvent être utilisées qu'avec les E / S RTC, cette source de réveil peut être utilisée avec n'importe quelle E / S (RTC ou numérique).

esp_sleep_enable_gpio_wakeup() peut être utilisée pour activer cette source de réveil.

Réveil UART (sommeil léger uniquement)

Lorsque ESP32 reçoit une entrée UART de périphériques externes, il est souvent nécessaire de réveiller la puce lorsque les données d'entrée sont disponibles. Le périphérique UART contient une fonction qui permet de réveiller la puce du sommeil léger lorsqu'un certain nombre de fronts positifs sur la broche RX sont visibles. Ce nombre de fronts positifs peut être défini à l'aide de la uart_set_wakeup_threshold()fonction. Notez que le caractère qui déclenche le réveil (et tous les caractères avant) ne seront pas reçus par l'UART après le réveil. Cela signifie que le périphérique externe doit généralement envoyer un caractère supplémentaire à l'ESP32 pour déclencher le réveil, avant d'envoyer les données.

esp_sleep_enable_uart_wakeup() peut être utilisée pour activer cette source de réveil.

Mise hors tension des périphériques et des mémoires RTC

Par défaut, les fonctions esp_deep_sleep_start()et met esp_light_sleep_start()hors tension tous les domaines d'alimentation RTC qui ne sont pas nécessaires aux sources de réveil activées. Pour remplacer ce comportement, une esp_sleep_pd_config()fonction est fournie.

Remarque: dans la révision 0 de l'ESP32, la mémoire rapide RTC sera toujours maintenue activée en veille profonde, afin que le stub de veille profonde puisse fonctionner après la réinitialisation. Cela peut être remplacé si l'application n'a pas besoin d'un comportement de réinitialisation propre après un sommeil profond.

Si certaines variables du programme sont placées dans la mémoire lente RTC (par exemple, en utilisant l' RTC_DATA_ATTRattribut), la mémoire lente RTC sera maintenue sous tension par défaut. Cela peut être annulé en utilisant la esp_sleep_pd_config()fonction, si vous le souhaitez.

Entrer dans le sommeil léger

esp_light_sleep_start()La fonction peut être utilisée pour entrer en veille légère une fois que les sources de réveil sont configurées. Il est également possible de passer en veille légère sans qu'aucune source de réveil ne soit configurée.Dans ce cas, la puce sera indéfiniment en mode veille légère, jusqu'à ce que la réinitialisation externe soit appliquée.

Entrer dans le sommeil profond

esp_deep_sleep_start()La fonction peut être utilisée pour entrer en veille profonde une fois que les sources de réveil sont configurées. Il est également possible de passer en veille profonde sans qu'aucune source de réveil ne soit configurée.Dans ce cas, la puce sera en mode veille prolongée indéfiniment, jusqu'à ce que la réinitialisation externe soit appliquée.

Configuration des E / S

Certains ESP32 IOs ont des pullups ou des pulldowns internes, qui sont activés par défaut. Si un circuit externe pilote cette broche en mode de veille profonde, la consommation de courant peut augmenter en raison du courant circulant à travers ces pullups et pulldowns. Pour isoler une broche, évitant une consommation de courant supplémentaire, appelez la rtc_gpio_isolate()fonction.

Par exemple, sur le module ESP32-WROVER, GPIO12 est extrait en externe. GPIO12 a également un pulldown interne dans la puce ESP32. Cela signifie qu'en sommeil profond, un certain courant circulera à travers ces résistances externes et internes, augmentant le courant de sommeil profond au-dessus de la valeur minimale possible. Ajoutez le code suivant avant esp_deep_sleep_start()de supprimer ce courant supplémentaire:

13.ino

```
rtc_gpio_isolate(GPIO_NUM_12);
```

Gestion de la sortie UART

Avant d'entrer en mode veille, esp_deep_sleep_start()voir le contenu des FIFO UART.

Lorsque vous entrez en mode veille légère en utilisant esp_light_sleep_start(), les FIFO UART ne seront pas vidées. Au lieu de cela, la sortie UART sera suspendue, et les caractères restants dans le FIFO seront envoyés après le réveil du sommeil léger.

Vérification de la cause du réveil du sommeil

esp_sleep_get_wakeup_cause() La fonction peut être utilisée pour vérifier quelle source de réveil a déclenché le réveil depuis le mode veille. Pour le pavé tactile et les sources de réveil ext1, il est possible d'identifier la broche ou le pavé tactile qui a provoqué le réveil à l'aide des fonctions esp_sleep_get_touchpad_wakeup_status()et esp_sleep_get_ext1_wakeup_status().

Désactiver la source de réveil du sommeil

La source de réveil précédemment configurée peut être désactivée ultérieurement à l'aide de l' esp_sleep_disable_wakeup_source()API. Cette fonction désactive le déclenchement pour la source de réveil donnée. De plus, il peut désactiver tous les déclencheurs si l'argument est ESP_SLEEP_WAKEUP_ALL.

Exemple d'application

La mise en œuvre des fonctionnalités de base du sommeil profond est illustrée dans l' exemple de protocoles / sntp , où le module ESP est réveillé périodiquement pour récupérer l'heure du serveur NTP.

Un exemple plus complet dans system / deep_sleep illustre l'utilisation de divers déclencheurs de réveil en veille profonde et de la programmation du coprocesseur ULP.

Référence API

En tête de fichier

- esp_system / include / esp_sleep.h

Les fonctions esp_err_t esp_sleep_disable_wakeup_source (source esp_sleep_source_t)

Désactivez la source de réveil.

Cette fonction permet de désactiver le déclencheur de réveil pour la source définie comme paramètre de la fonction. Voir docs / sleep-modes.rst pour plus de détails.

Remarque

Cette fonction ne modifie pas la configuration de réveil dans RTC. Il sera exécuté dans la fonction esp_sleep_start.

Retour

- ESP_OK en cas de succès
- ESP_ERR_INVALID_STATE si le déclencheur n'était pas actif

Paramètres

- source: - numéro de source à désactiver de type esp_sleep_source_t

esp_err_t esp_sleep_enable_ulp_wakeup (void)

Activez le réveil par le coprocesseur ULP.

Remarque

Dans les révisions 0 et 1 de l'ESP32, la source de réveil ULP ne peut pas être utilisée lorsque le domaine d'alimentation RTC_PERIPH est forcé d'être mis sous tension (ESP_PD_OPTION_ON) ou lorsque la source de réveil ext0 est utilisée.

Retour

- ESP_OK en cas de succès
- ESP_ERR_NOT_SUPPORTED si le courant supplémentaire au toucher (CONFIG_ESP32_RTC_EXT_CRYST_ADDIT_CURRENT) est activé.
- ESP_ERR_INVALID_STATE si le coprocesseur ULP n'est pas activé ou si le réveil déclenche un conflit

esp_err_t esp_sleep_enable_timer_wakeup (uint64_t time_in_us)

Activez le réveil par minuterie.

Retour

- ESP_OK en cas de succès
- ESP_ERR_INVALID_ARG si la valeur est hors limites (à déterminer)

Paramètres

- time_in_us: temps avant le réveil, en microsecondes

esp_err_t esp_sleep_enable_touchpad_wakeup (void)

Activez le réveil par capteur tactile.

Remarque

- Dans les révisions 0 et 1 de l'ESP32, la source de réveil tactile ne peut pas être utilisée lorsque le domaine d'alimentation RTC_PERIPH est forcé d'être mis sous tension (ESP_PD_OPTION_ON) ou lorsque la source de réveil ext0 est utilisée.

Remarque

- Le mode FSM du bouton tactile doit être configuré comme mode de déclenchement de la minuterie.

Retour

- ESP_OK en cas de succès
- ESP_ERR_NOT_SUPPORTED si le courant supplémentaire au toucher (CONFIG_ESP32_RTC_EXT_CRYST_ADDIT_CURRENT) est activé.
- ESP_ERR_INVALID_STATE si le réveil déclenche un conflit

touch_pad_t esp_sleep_get_touchpad_wakeup_status (void)

Obtenez le pavé tactile qui a provoqué le réveil.

Si le réveil a été provoqué par une autre source, cette fonction renverra TOUCH_PAD_MAX;

Retour

pavé tactile qui a provoqué le réveil

bool esp_sleep_is_valid_wakeup_gpio (gpio_num_t gpio_num)

Renvoie true si un numéro GPIO est valide pour une utilisation comme source de réveil.

Remarque

Pour les SoC avec capacité RTC IO, il peut s'agir de n'importe quelle broche d'entrée RTC IO valide.

Retour

Vrai si ce numéro GPIO sera accepté comme source de réveil du sommeil.

Paramètres

- gpio_num: Numéro du GPIO à tester pour la capacité de la source de réveil

esp_err_t esp_sleep_enable_ext0_wakeup (gpio_num_t gpio_num , niveau int)

Activez le réveil à l'aide d'une épingle.

Cette fonction utilise la fonction de réveil externe du périphérique RTC_IO. Cela ne fonctionnera que si les périphériques RTC restent allumés pendant le sommeil.

Cette fonction peut surveiller n'importe quelle broche qui est un IO RTC. Une fois que la broche passe dans l'état donné par l'argument de niveau, la puce sera réveillée.

Remarque

Cette fonction ne modifie pas la configuration des broches. La broche est configurée dans esp_sleep_start, juste avant d'entrer en mode veille.

Remarque

Dans les révisions 0 et 1 de l'ESP32, la source de réveil ext0 ne peut pas être utilisée avec des sources de réveil tactiles ou ULP.

Retour

- ESP_OK en cas de succès
- ESP_ERR_INVALID_ARG si le GPIO sélectionné n'est pas un GPIO RTC ou si le mode n'est pas valide
- ESP_ERR_INVALID_STATE si le réveil déclenche un conflit

Paramètres

- gpio_num: Numéro GPIO utilisé comme source de réveil. Seuls les GPIO dotés de la fonctionnalité RTC peuvent être utilisés: 0,2,4,12-15,25-27,32-39.
- level: niveau d'entrée qui déclenchera le réveil (0 = bas, 1 = haut)

esp_err_t esp_sleep_enable_ext1_wakeup (uint64_t masque , esp_sleep_ext1_wakeup_mode_t Mode)

Activez le réveil en utilisant plusieurs broches.

Cette fonction utilise la fonction de réveil externe du contrôleur RTC. Cela fonctionnera même si les périphériques RTC sont arrêtés pendant le sommeil.

Cette fonction peut surveiller n'importe quel nombre de broches qui se trouvent dans les E / S RTC. Une fois que l'une des broches sélectionnées entre dans l'état donné par l'argument mode, la puce sera réveillée.

Remarque

Cette fonction ne modifie pas la configuration des broches. Les broches sont configurées dans esp_sleep_start, juste avant d'entrer en mode veille.

Remarque

Les pullups et pulldowns internes ne fonctionnent pas lorsque les périphériques RTC sont arrêtés. Dans ce cas, des résistances externes doivent être ajoutées. Alternativement, les périphériques RTC (et les pullups / pulldowns) peuvent être maintenus activés à l'aide de la fonction

esp_sleep_pd_config.

Retour

- ESP_OK en cas de succès
- ESP_ERR_INVALID_ARG si l'un des GPIO sélectionnés n'est pas un GPIO RTC ou si le mode n'est pas valide

Paramètres

- mask: masque de bits des nombres GPIO qui provoqueront le réveil. Seuls les GPIO dotés de la fonctionnalité RTC peuvent être utilisés dans ce bitmap: 0,2,4,12-15,25-27,32-39.
- mode: sélectionnez la fonction logique utilisée pour déterminer la condition de réveil:
  - ESP_EXT1_WAKEUP_ALL_LOW: réveil lorsque tous les GPIO sélectionnés sont faibles
  - ESP_EXT1_WAKEUP_ANY_HIGH: se réveiller lorsque l'un des GPIO sélectionnés est élevé

esp_err_t esp_sleep_enable_gpio_wakeup (void)

Activez le réveil à partir du sommeil léger à l'aide de GPIO.

Chaque GPIO prend en charge la fonction de réveil, qui peut être déclenchée à un niveau bas ou élevé. Contrairement aux sources de réveil EXT0 et EXT1, cette méthode peut être utilisée à la fois pour tous les E / S: E / S RTC et E / S numériques. Cependant, il ne peut être utilisé que pour se réveiller après un sommeil léger.

Pour activer le réveil, appelez d'abord gpio_wakeup_enable, en spécifiant le numéro gpio et le niveau de réveil, pour chaque GPIO utilisé pour le réveil. Appelez ensuite cette fonction pour activer la fonction de réveil.

Remarque

Dans les révisions 0 et 1 de l'ESP32, la source de réveil GPIO ne peut pas être utilisée avec des sources de réveil tactiles ou ULP.

Retour

- ESP_OK en cas de succès
- ESP_ERR_INVALID_STATE si le réveil déclenche un conflit

esp_err_t esp_sleep_enable_uart_wakeup (int uart_num)

Activez le réveil du sommeil léger en utilisant UART.

Utilisez la fonction uart_set_wakeup_threshold pour configurer le seuil de réveil UART. Le réveil du sommeil léger prend un certain temps, donc tous les caractères envoyés à l'UART ne peuvent pas être reçus par l'application.

Remarque

ESP32 ne prend pas en charge le réveil depuis UART2.

Retour

- ESP_OK en cas de succès
- ESP_ERR_INVALID_ARG si le réveil à partir d'un UART donné n'est pas pris en charge

Paramètres

- uart_num: Port UART à partir duquel se réveiller

esp_err_t esp_sleep_enable_wifi_wakeup (void)

Activez le réveil par WiFi MAC.

Retour

- ESP_OK en cas de succès

uint64_t esp_sleep_get_ext1_wakeup_status (void)

Obtenez le masque de bits des GPIO qui ont provoqué le réveil (ext1) Si le réveil a été causé par une autre source, cette fonction renverra 0.

Retour masque de bits, si GPIOn a provoqué le réveil, BIT (n) sera défini

esp_err_t esp_sleep_pd_config (esp_sleep_pd_domain_t domaine , esp_sleep_pd_option_t choix)

Définissez le mode de mise hors tension pour un domaine d'alimentation RTC en mode veille.

S'il n'est pas défini à l'aide de cette API, tous les domaines d'alimentation sont définis par défaut sur ESP_PD_OPTION_AUTO.

Retour

- ESP_OK en cas de succès
- ESP_ERR_INVALID_ARG si l'un des arguments est hors limites

Paramètres

- domain: domaine d'alimentation à configurer
- option: option de mise hors tension (ESP_PD_OPTION_OFF, ESP_PD_OPTION_ON ou ESP_PD_OPTION_AUTO)

void esp_deep_sleep_start (nul)

Entrez dans le sommeil profond avec les options de réveil configurées. Cette fonction ne revient pas.

esp_err_t esp_light_sleep_start (void)

Entrez en veille légère avec les options de réveil configurées.

Retour

- ESP_OK en cas de succès (renvoyé après le réveil)
- ESP_ERR_INVALID_STATE si WiFi ou BT n'est pas arrêté

void esp_deep_sleep (uint64_t time_in_us)

Entrez en mode sommeil profond.

L'appareil se réveillera automatiquement après le temps de sommeil profond. Au réveil, l'appareil appelle le stub de veille de sommeil profond, puis procède au chargement de l'application.

L'appel à cette fonction équivaut à un appel à esp_deep_sleep_enable_timer_wakeup suivi d'un appel à esp_deep_sleep_start.

esp_deep_sleep n'arrête pas correctement les connexions WiFi, BT et de protocole de niveau supérieur. Assurez-vous que les fonctions de pile WiFi et BT appropriées sont appelées pour fermer toutes les connexions et désinitialiser les périphériques. Ceux-ci inclus:

- esp_bluedroid_disable
- esp_bt_controller_disable
- esp_wifi_stop

Cette fonction ne revient pas.

Paramètres

- time_in_us: temps de sommeil profond, unité: microseconde

esp_sleep_wakeup_cause_t esp_sleep_get_wakeup_cause (void)

Obtenez la source de réveil qui a provoqué le réveil du sommeil.

Retour

cause du réveil du dernier sommeil (sommeil profond ou sommeil léger)

void esp_wake_deep_sleep (nul)

Stub par défaut à exécuter au réveil après un sommeil profond.

Permet d'exécuter du code immédiatement au réveil, avant le démarrage du chargeur de démarrage du logiciel ou de l'application ESP-IDF. Cette fonction est faiblement liée, vous pouvez donc implémenter votre propre version pour exécuter le code immédiatement lorsque la puce sort du mode veille.

Voir docs / deep-sleep-stub.rst pour plus de détails.

void esp_set_deep_sleep_wake_stub (esp_deep_sleep_wake_stub_fn_t nouveau_stub)

Installez un nouveau stub au moment de l'exécution pour l'exécuter au réveil après un sommeil profond.

Si vous implémentez esp_wake_deep_sleep (), il n'est pas nécessaire d'appeler cette fonction.

Cependant, il est possible d'appeler cette fonction pour remplacer un autre stub de sommeil profond. Toute fonction utilisée comme stub de sommeil profond doit être marquée RTC_IRAM_ATTR, et doit obéir aux mêmes règles que celles données pour esp_wake_deep_sleep ().

esp_deep_sleep_wake_stub_fn_t esp_get_deep_sleep_wake_stub (void)

Obtenez le réveil actuel du stub de sommeil profond.

Retour Renvoie l'actuel réveil du stub de veille profonde, ou NULL si aucun stub n'est installé.

void esp_default_wake_deep_sleep (nul)

Le stub esp_wake_deep_sleep () par défaut fourni par esp-idf.

Voir docs / deep-sleep-stub.rst pour plus de détails.

void esp_deep_sleep_disable_rom_logging (nul)

Désactivez la journalisation à partir du code ROM après une veille prolongée.

Utilisation de LSB de RTC_STORE4.

Void esp_sleep_gpio_status_init (nul)

Désactivez toutes les broches GPIO en état de veille.

Void esp_sleep_gpio_status_switch_configure (bool enable)

Configurez le basculement de l'état des broches GPIO entre l'état de veille et l'état de veille.

Paramètres

- enable: décider de changer d'état ou non

Définitions de type

typedef esp_sleep_source_t esp_sleep_wakeup_cause_t

typedef void (* esp_deep_sleep_wake_stub_fn_t) (void)

Type de fonction pour que le stub s'exécute au réveil.

Énumérations

énumération esp_sleep_ext1_wakeup_mode_t

Fonction logique utilisée pour le mode de réveil EXT1.

Valeurs:

ESP_EXT1_WAKEUP_ALL_LOW = 0

- Réveillez la puce lorsque tous les GPIO sélectionnés deviennent bas.

ESP_EXT1_WAKEUP_ANY_HIGH = 1

- Réveillez la puce lorsque l'un des GPIO sélectionnés devient élevé.

énumération esp_sleep_pd_domain_t

Domaines d'alimentation pouvant être mis hors tension en mode veille.

Valeurs:

ESP_PD_DOMAIN_RTC_PERIPH

- RTC IO, capteurs et coprocesseur ULP.

ESP_PD_DOMAIN_RTC_SLOW_MEM

- Mémoire lente RTC.

ESP_PD_DOMAIN_RTC_FAST_MEM

- Mémoire rapide RTC.

ESP_PD_DOMAIN_XTAL

- Oscillateur XTAL.

ESP_PD_DOMAIN_MAX

- Nombre de domaines.

énumération esp_sleep_pd_option_t

Options de mise hors tension.

Valeurs: ESP_PD_OPTION_OFF

- Mettez le domaine d'alimentation hors tension en mode veille.

ESP_PD_OPTION_ON

- Gardez le domaine d'alimentation activé en mode veille.

ESP_PD_OPTION_AUTO

- Gardez le domaine d'alimentation activé en mode veille, s'il est requis par l'une des options de réveil. Sinon, mettez-le hors tension.

énumération esp_sleep_source_t

Cause de réveil du sommeil.

Valeurs: ESP_SLEEP_WAKEUP_UNDEFINED

- En cas de sommeil profond, la réinitialisation n'a pas été provoquée par la sortie du sommeil profond.

ESP_SLEEP_WAKEUP_ALL

- Pas une cause de réveil, utilisée pour désactiver toutes les sources de réveil avec esp_sleep_disable_wakeup_source.

ESP_SLEEP_WAKEUP_EXT0

- Réveil provoqué par un signal externe utilisant RTC_IO.

ESP_SLEEP_WAKEUP_EXT1

- Réveil provoqué par un signal externe utilisant RTC_CNTL.

ESP_SLEEP_WAKEUP_TIMER

- Réveil causé par la minuterie.

ESP_SLEEP_WAKEUP_TOUCHPAD

- Réveil causé par le pavé tactile.

ESP_SLEEP_WAKEUP_ULP

- Réveil provoqué par le programme ULP.

ESP_SLEEP_WAKEUP_GPIO

- Réveil causé par GPIO (veille légère uniquement)

ESP_SLEEP_WAKEUP_UART

- Réveil causé par UART (sommeil léger uniquement)

ESP_SLEEP_WAKEUP_WIFI

- Réveil causé par WIFI (sommeil léger uniquement)

ESP_SLEEP_WAKEUP_COCPU

- Réveil provoqué par COCPU int.

ESP_SLEEP_WAKEUP_COCPU_TRAP_TRIG

- Réveil causé par un crash du COCPU.

ESP_SLEEP_WAKEUP_BT

- Réveil causé par BT (sommeil léger uniquement)

autre exemple « sommeil profond » Deep Sleep

49.ino

```
/*
Simple Deep Sleep with Timer Wake Up
=====================================
ESP32 offers a deep sleep mode for effective power
saving as power is an important factor for IoT
applications. In this mode CPUs, most of the RAM,
and all the digital peripherals which are clocked
from APB_CLK are powered off. The only parts of
the chip which can still be powered on are:
RTC controller, RTC peripherals ,and RTC memories

This code displays the most basic deep sleep with
a timer to wake it up and how to store data in
RTC memory to use it over reboots

This code is under Public Domain License.

Author:
Pranav Cherukupalli <cherukupallip@gmail.com>
*/

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

RTC_DATA_ATTR int bootCount = 0;

/*
Method to print the reason by which ESP32
has been awaken from sleep
*/
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("Wakeup caused by
external signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by
external signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by
timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by
touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP
program"); break;
```

```
    default : Serial.printf("Wakeup was not caused by deep sleep:
%d\n",wakeup_reason); break;
  }
}

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

  /*
  First we configure the wake up source
  We set our ESP32 to wake up every 5 seconds
  */
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.println("Setup ESP32 to sleep for every " +
String(TIME_TO_SLEEP) +
  " Seconds");

  /*
  Next we decide what all peripherals to shut down/keep on
  By default, ESP32 will automatically power down the peripherals
  not needed by the wakeup source, but if you want to be a poweruser
  this is for you. Read in detail at the API docs
http://esp-idf.readthedocs.io/en/latest/api-reference/system/deep_sleep
.html
  Left the line commented as an example of how to configure
peripherals.
  The line below turns off all RTC peripherals in deep sleep.
  */
  //esp_deep_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH,
ESP_PD_OPTION_OFF);
  //Serial.println("Configured all RTC Peripherals to be powered down
in sleep");

  /*
  Now that we have setup a wake cause and if needed setup the
  peripherals state in deep sleep, we can now start going to
  deep sleep.
  In the case that no wake up sources were provoidd but deep
  sleep was started, it will sleep forever unless hardware
  reset occurs.
  */
```

```
  Serial.println("Going to sleep now");
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

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

50.ino

```
/*

Simple Deep Sleep with Timer Wake Up

=====================================

ESP32 offers a deep sleep mode for effective power

saving as power is an important factor for IoT

applications. In this mode CPUs, most of the RAM,

and all the digital peripherals which are clocked

from APB_CLK are powered off. The only parts of

the chip which can still be powered on are:

RTC controller, RTC peripherals ,and RTC memories


This code displays the most basic deep sleep with

a timer to wake it up and how to store data in

RTC memory to use it over reboots


This code is under Public Domain License.


Author:

Pranav Cherukupalli <cherukupallip@gmail.com>
```

```
*/



#define uS_TO_S_FACTOR 1000000ULL /* Conversion factor for micro
seconds to seconds */

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



RTC_DATA_ATTR int bootCount = 0;



/*

Method to print the reason by which ESP32

has been awaken from sleep

*/

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("Wakeup caused by external
signal using RTC_IO"); break;

case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external
signal using RTC_CNTL"); break;

case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer");
break;

case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by
touchpad"); break;
```

```
case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP
program"); break;

default : Serial.printf("Wakeup was not caused by deep sleep:
%d\n",wakeup_reason); break;

}

}



void setup(){

Serial.begin(115200);

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();



/*

First we configure the wake up source

We set our ESP32 to wake up every 5 seconds

*/

esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);

Serial.println("Setup ESP32 to sleep for every " +
String(TIME_TO_SLEEP) +

" Seconds");
```

```
/*

Next we decide what all peripherals to shut down/keep on

By default, ESP32 will automatically power down the peripherals

not needed by the wakeup source, but if you want to be a poweruser

this is for you. Read in detail at the API docs

http://esp-idf.readthedocs.io/en/latest/api-reference/system/deep_sleep
.html

Left the line commented as an example of how to configure peripherals.

The line below turns off all RTC peripherals in deep sleep.

*/

//esp_deep_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH,
ESP_PD_OPTION_OFF);

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



/*

Now that we have setup a wake cause and if needed setup the

peripherals state in deep sleep, we can now start going to

deep sleep.

In the case that no wake up sources were provided but deep

sleep was started, it will sleep forever unless hardware

reset occurs.

*/

Serial.println("Going to sleep now");

Serial.flush();

esp_deep_sleep_start();
```

```
Serial.println("This will never be printed");

}



void loop(){

//This is not going to be called

}
```