**LẬP TRÌNH ESP32**

Bài 8: Lập trình ESP32 Sleep mode chế độ ngủ tiết kiệm năng lượng

POSTED ON 03/07/2021 BY KHUÊ NGUYỄN

03
Th7

ESP32 và Platform IO

**Khuê Nguyễn Creator**

Bài 8: Lập trình ESP32 Sleep Mode tiết kiệm năng lượng

Năng lượng luôn là một vấn đề quan trọng trong các thiết bị IoT, vì vậy chúng ta cần học cách sử dụng ESP32 Sleep mode giúp đưa ESP32 vào chế độ ngủ và đánh thức phù hợp.

Trong bài này, hãy cùng tìm hiểu các vấn đề về năng lượng của ESP32 và cách lập trình nó nhé

Bài 9 trong Serie Học ESP32 từ A tới Z

Mục Lục



1. Sleep Mode là gì?
2. ESP32 Sleep Mode
 - 2.1. Sơ đồ khối của ESP32 và cách ESP32 tiêu thụ năng lượng
 - 2.2. Các chế độ tiết kiệm năng lượng của ESP32 Sleep Mode
 - 2.2.1. ESP32 Active Mode
 - 2.2.2. ESP32 Modem Sleep
 - 2.2.3. ESP32 Light Sleep
 - 2.2.4. ESP32 Deep Sleep
 - 2.2.5. ESP32 Hibernation mode
3. Các cơ chế đánh thức (Wake Up) ESP32 khỏi chế độ Deep Sleep
4. Lập trình ESP32 Timer Wake Up
 - 4.1. Code và giải thích code
 - 4.2. Nạp và kết quả
5. Lập trình ESP32 Touch Wake Up
 - 5.1. Sơ đồ nguyên lý
 - 5.2. Code và giải thích code
6. Lập trình ESP32 External Wake Up
 - 6.1. Sơ đồ nguyên lý
 - 6.2. Code và giải thích code
7. Kết
 - 7.1. Related posts:

Sleep Mode là gì?

Trên thực tế, tất cả các dòng vi điều khiển đều có chế độ ngủ (sleep mode) thế nhưng mỗi dòng sẽ có hiệu suất khác nhau. Với những ứng dụng sử dụng điện trực tiếp từ điện lưới quốc gia, chúng ta không cần quan tâm tới vấn đề năng lượng.

Thế nhưng những ứng dụng đó thì luôn phải đặt một chỗ, không thể di chuyển. Mà các thiết bị IoT hiện nay số lượng sản phẩm gắn trên người hay động vật... rất nhiều, thế nên phải sử dụng pin cho các ứng dụng đó.

Vậy nên Sleep Mode ra đời để đưa MCU về chế độ tiết kiệm năng lượng, bằng cách tắt bớt các phần chưa cần hoạt động đi. Và đặt những sự kiện có thể đánh thức (Wake Up) MCU dậy

ESP32 Sleep Mode

Trong ESP32 khi vào chế độ Sleep mode, các phần không cần thiết sẽ ngừng hoạt động. Với từng chế độ ngủ khác nhau, chúng ta sẽ tắt lần lượt các khối khác nhau. Ngoài ra ESP32 còn có một bộ xử lý tiết kiệm năng lượng **ULP Processor** (Ultra-Low Power) độc lập với Core chính, và có thể truy cập được một số ngoại vi nhất định. Giúp khả năng xử lý của ESP32 trong khi ngủ tốt hơn.

Sơ đồ khối của ESP32 và cách ESP32 tiêu thụ năng lượng

Hãy cùng xem cấu trúc của ESP32



Sơ đồ khối ESP32

Đây là các nguồn tiêu thụ năng lượng của ESP32, thực chất khi vào Sleep mode, ESP32 sẽ turn off một số phần, dẫn tới năng lượng tiêu thụ giảm. Các phần đó bao gồm:

- Bluetooth + Wifi + Radio: Đây là phần tiêu thụ điện năng lớn nhất trên ESP32
- ESP32 Core+Memory: Lõi chính và bộ nhớ
- ULP Coprocessor: Bộ xử lý tiết kiệm năng lượng
- RTC: Khối thời gian thực
- Peripherals: Các ngoại vi

Các chế độ tiết kiệm năng lượng của ESP32 Sleep Mode

Trong ESP32 có 5 chế độ tiết kiệm năng lượng đó là:

- Active mode
- Modem Sleep mode
- Light Sleep mode
- Deep Sleep mode
- Hibernation mode

Với 5 chế độ này các ngoại vi sẽ được bật/tắt theo bảng sau:

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

5 Chế độ tiết kiệm năng lượng ESP32

Để tra cứu năng lượng tiêu thụ của các khối, chúng ta có thể tham khảo bảng sau:

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 μ A
	ULP sensor-monitored pattern	100 μ A @1% duty
	RTC timer + RTC memory	10 μ A
Hibernation	RTC timer only	5 μ A
Power off	CHIP_PU is set to low level, the chip is powered off	0.1 μ A

*Bảng tra cứu năng lượng tiêu thụ ESP32***Table 10: RF Power-Consumption Specifications**

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

Bảng tra cứu năng lượng tiêu thụ của Wifi và BT/BLE ESP32

Chi tiết các bạn xem thêm tại: [ESP32 Espressif datasheet](#)

Để hiểu chi tiết hơn về các chế độ tiết kiệm năng lượng của ESP32 chúng ta sẽ đi vào từng chế độ

ESP32 Active Mode

Trong chế độ này, tất cả các khối chức năng của chip đều được bật, và đây là chế độ bình thường khi chạy các chương trình

Vì **Active Mode** bật mọi chức năng (đặc biệt là module WiFi, Bluetooth và nhân xử lý), nên chip yêu cầu dòng điện hơn **240mA** để hoạt động. Ngoài ra, nếu bạn sử dụng cả hai chức năng WiFi và Bluetooth cùng lúc thì năng lượng tiêu thụ sẽ lớn hơn rất nhiều (lớn nhất là **790mA**).

Lưu ý: Vì thế nên khi sử dụng các KIT esp32 chúng ta cần cấp nguồn ngoài cho KIT nếu không wifi hoặc BT/BLE sẽ không hoạt động nếu phải điều khiển thêm nhiều các phần tử khác trong mạch.

ESP32 Modem Sleep

Trong **Modem Sleep**, mọi thứ đều hoạt động, chỉ có WiFi, Bluetooth và Radio bị tắt. Để duy trì sự hoạt động của WiFi và BT/BLE, chúng phải được đánh thức định kì theo một thời gian mà lập trình viên mong muốn.

Chế độ này chỉ hoạt động trong chế độ máy trạm (Wifi Station), không được sử dụng trong chế độ điểm truy cập (Access Point). Khi ở chế độ này ESP32 vẫn kết nối với bộ định tuyến thông qua cơ chế **DTIM Beacon**.

Để tiết kiệm năng lượng, ESP32 vô hiệu hóa module Wi-Fi giữa hai khoảng thời gian DTIM Beacon và tự động thức dậy trước khi đến Beacon tiếp theo. Thời gian wake up có thể từ 100 – 1000ms.

Với chế độ này ESP32 tiêu thụ từ: 3mA – 20mA



ESP32 Light Sleep

Với chế độ này ESP32 tắt các thành phần phát sóng như wifi, BT/BLE như chế độ modem sleep, cùng với tắt nguồn xung đồng hồ (Clock) của CPU và RAM và ngoại vi. Nhưng RTC và ULP – coprocessor vẫn hoạt động bình thường.

Kỹ thuật này được gọi là clock-gating.

Clock gating là một kỹ thuật để giảm tiêu thụ năng lượng. Nó vô hiệu hóa các phần của mạch điện bằng cách tắt các xung clock, để các flip-flop trong chúng không phải chuyển trạng thái. Vì khi có chuyển đổi trạng thái thì năng lượng bị tiêu thụ, ngược lại, mức tiêu thụ năng lượng bằng 0.

Trong chế độ này ESP32 tiêu thụ khoảng 0.8mA



ESP32 Deep Sleep

Ở chế độ **Deep Sleep**, CPU, RAM và tất cả các ngoại vi đều bị tắt. Các bộ phận duy nhất của chip vẫn được cấp nguồn là: bộ RTC, ngoại vi RTC (bao gồm bộ ULP) và bộ nhớ RTC.

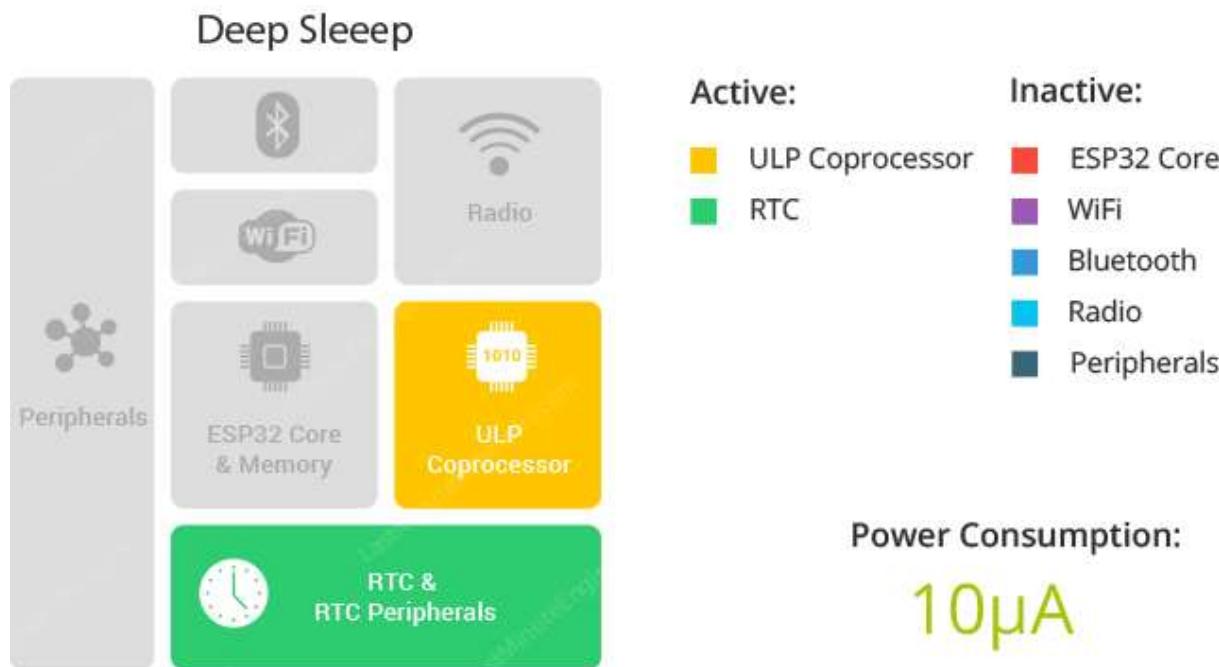
CPU chính bị tắt nguồn còn bộ ULP thực hiện các phép đo cảm biến và đánh thức hệ thống chính dựa trên dữ liệu đo được.

Cùng với CPU, bộ nhớ chính của chip cũng bị tắt. Vì vậy, mọi thứ được lưu trữ trong bộ nhớ đó bị xóa sạch và không thể truy cập được.

Tuy nhiên, bộ nhớ RTC vẫn được bật. Vì vậy, nội dung của nó được bảo quản trong **Deep Sleep** và có thể được lấy ra sau khi chip được đánh thức. Đó là lý do mà chip lưu trữ dữ liệu kết nối Wi-Fi và Bluetooth trong bộ nhớ RTC trước đó.

Khi Wake up khỏi Deep Sleep, ESP32 sẽ hoạt động lại từ đầu, tương tự như việc reset vậy.

Trong chế độ này ESP32 tiêu thụ từ **10µA** đến **0.15mA**



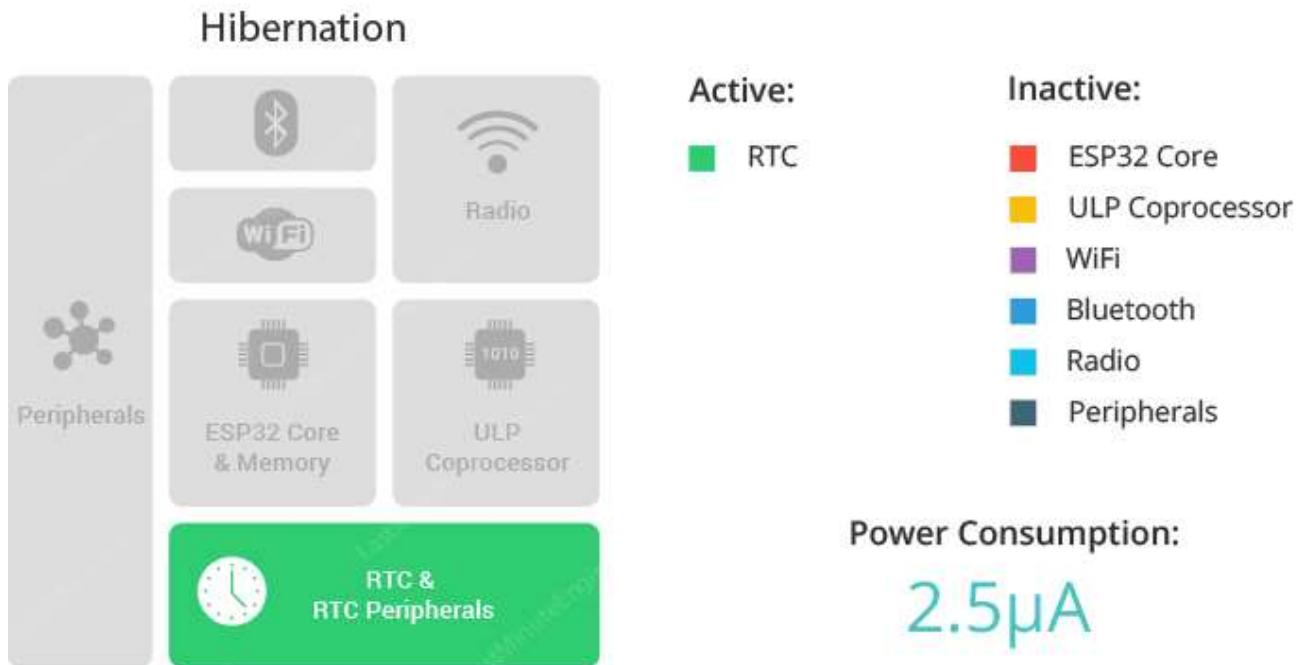
ESP32 Hibernation mode

Trong chế độ **Hibernation Mode**, chip vô hiệu hóa bộ tạo dao động 8MHz bên trong và bộ ULP. Bộ nhớ phục hồi RTC cũng bị tắt nguồn, có nghĩa là không có cách nào chúng ta có thể bảo quản dữ liệu trong chế độ ngủ đông.

Mọi thứ khác đều bị tắt ngoại trừ bộ đếm thời gian RTC slow clock và một số GPIO RTC đang hoạt động. Chúng có trách nhiệm đánh thức chip ra khỏi **Hibernation Mode**.

Vậy nên hãy lưu ý khi sử dụng chế độ này nhé.

Trong chế độ **Hibernation Mode**, chip chỉ tiêu thụ khoảng $2.5\mu\text{A}$.



Các cơ chế đánh thức (Wake Up) ESP32 khỏi chế độ Deep Sleep

Trong bài này, chúng ta sẽ đề cập tới chế độ Deep Sleep và cách đánh thức nó. Để có thể đánh thức được ESP32 từ trạng thái ngủ về hoạt động bình thường, chúng ta có các nguồn sau

- **Timer:** Định thời gian cho ESP32 thức dậy theo chu kì nhất định
- **Touch pad:** Sử dụng các chân cảm ứng-
- **External wake up:** Sử dụng 2 nguồn bên ngoài để đánh thức ESP32 đó là (**ext0 & ext1**)
- **ULP Processor:** Sử dụng bộ xử lý tiết kiệm năng lượng để đánh thức (Sẽ không được đề cập trong bài này)

Tùy vào nhu cầu cụ thể, chúng ta sẽ lựa chọn các phương án đánh thức cho phù hợp với dự án của mình.

Lập trình ESP32 Timer Wake Up

Với chế độ đánh thức bằng timer chúng ta sẽ sử dụng hàm

`esp_sleep_enable_timer_wakeup(time_in_us)` . Khi sử dụng hàm này, mỗi khi esp32 rơi vào trạng thái ngủ, chúng sẽ được tự động đánh thức sau khoảng thời gian `time_in_us`

Để vào chế độ ngủ chúng ta sử dụng câu lệnh `esp_deep_sleep_start();`

Code và giải thích code

Copy code vào platform IO

Full Code

```

02 #include <Arduino.h>
03
04 #define uS_TO_S_FACTOR 1000000 // biến chuyển từ micro giây
05 #define TIME_TO_SLEEP 5 //Thời gian thức dậy
06
07 RTC_DATA_ATTR int bootCount = 0; // biến này lưu tại bộ nhớ I
08 int bootCountInRam = 0; // biến này lưu tại RAM
09
10 //in ra nguồn đánh thức
11 void print_wakeup_reason(){
12     esp_sleep_wakeup_cause_t wakeup_reason;
13     wakeup_reason = esp_sleep_get_wakeup_cause();
14     switch(wakeup_reason)
15     {
16         case ESP_SLEEP_WAKEUP_EXT0 :
17             Serial.println("Wakeup caused by external signal using
18             break;
19         case ESP_SLEEP_WAKEUP_EXT1 :
20             Serial.println("Wakeup caused by external signal using
21             break;
22         case ESP_SLEEP_WAKEUP_TIMER :
23             Serial.println("Wakeup caused by timer");
24             break;
25         case ESP_SLEEP_WAKEUP_TOUCHPAD :
26             Serial.println("Wakeup caused by touchpad");
27             break;
28         case ESP_SLEEP_WAKEUP_ULP :
29             Serial.println("Wakeup caused by ULP program");
30             break;
31         default :

```

```

32         Serial.printf("Wakeup was not caused by deep sleep: %d"
33         break;
34     }
35 }
36
37 void setup(){
38     Serial.begin(115200);
39     delay(1000); //chờ 1 khoảng nhỏ cho serial hoạt động
40
41     //mỗi lần thức dậy sẽ tăng biến này 1 lần và in ra
42     ++bootCount;
43     ++bootCountInRam;
44     Serial.println("Boot number: " + String(bootCount));
45     Serial.println("Boot in ram number: " + String(bootCountInRam));
46
47     //in ra nguồn đánh thức esp32
48     print_wakeup_reason();
49
50     //gọi hàm thức dậy mỗi 5s
51     esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
52     Serial.println("Setup ESP32 to sleep for every " + String(
53         " Seconds"));
54
55     Serial.println("Going to sleep now");
56     delay(1000);
57     Serial.flush();
58     // bắt đầu vào chế độ ngủ
59     esp_deep_sleep_start();
60 }
61
62 void loop(){
63     //This is not going to be called
64 }
```

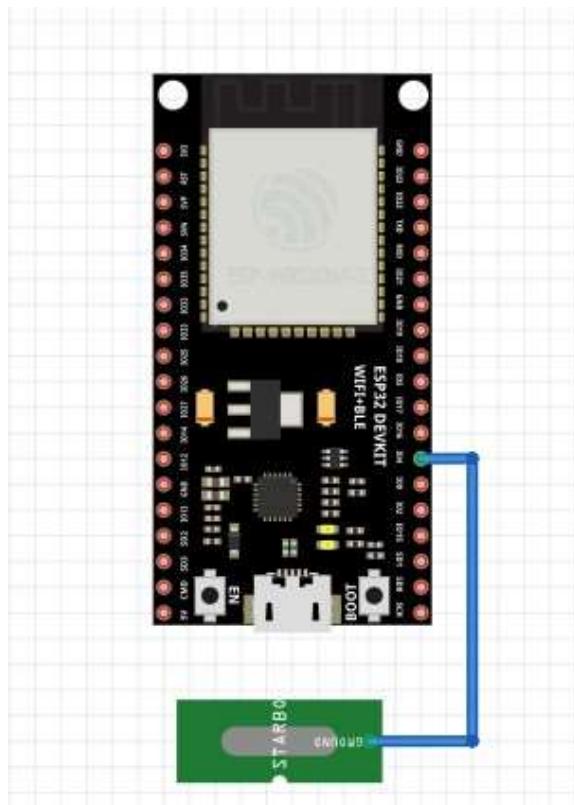
Nạp và kết quả

Cứ sau 5s chúng ta sẽ thấy ESP32 thức dậy 1 lần, biến `bootCount` được lưu trong bộ nhớ RTC thế nên không bị reset lại giá trị

Biến `bootCountInRam` sẽ bị xóa giá trị liên tục khi ESP32 thức dậy.

Lập trình ESP32 Touch Wake Up

Sơ đồ nguyên lý



esp32 touch wake up

Code và giải thích code

Full code

Full Code

```

02 #include <Arduino.h>
03
04 #define Threshold 40 //biến giới hạn (độ nhạy) cho touch pin
05
06 // khai báo 1 biến kiểu touch_pad_t
07 touch_pad_t touchPin;
08
09 RTC_DATA_ATTR int bootCount = 0; // biến này lưu tại bộ nhớ I
10
11 //in ra nguồn đánh thức
12 void print_wakeup_reason(){
13     esp_sleep_wakeup_cause_t wakeup_reason;
14     wakeup_reason = esp_sleep_get_wakeup_cause();
15     switch(wakeup_reason)
16     {
17         case ESP_SLEEP_WAKEUP_EXT0 :

```

```

18     Serial.println("Wakeup caused by external signal using
19     break;
20     case ESP_SLEEP_WAKEUP_EXT1 :
21         Serial.println("Wakeup caused by external signal using
22         break;
23     case ESP_SLEEP_WAKEUP_TIMER :
24         Serial.println("Wakeup caused by timer");
25         break;
26     case ESP_SLEEP_WAKEUP_TOUCHPAD :
27         Serial.println("Wakeup caused by touchpad");
28         break;
29     case ESP_SLEEP_WAKEUP_ULP :
30         Serial.println("Wakeup caused by ULP program");
31         break;
32     default :
33         Serial.printf("Wakeup was not caused by deep sleep: %d
34         break;
35     }
36 }
37
38 //in ra chân wake up
39 void print_wakeup_touchpad(){
40     touchPin = esp_sleep_get_touchpad_wakeup_status();
41
42     switch(touchPin)
43     {
44         case 0 : Serial.println("Touch detected on GPIO 4"); break;
45         case 1 : Serial.println("Touch detected on GPIO 0"); break;
46         case 2 : Serial.println("Touch detected on GPIO 2"); break;
47         case 3 : Serial.println("Touch detected on GPIO 15"); break;
48         case 4 : Serial.println("Touch detected on GPIO 13"); break;
49         case 5 : Serial.println("Touch detected on GPIO 12"); break;
50         case 6 : Serial.println("Touch detected on GPIO 14"); break;
51         case 7 : Serial.println("Touch detected on GPIO 27"); break;
52         case 8 : Serial.println("Touch detected on GPIO 33"); break;
53         case 9 : Serial.println("Touch detected on GPIO 32"); break;
54     default : Serial.println("Wakeup not by touchpad"); break;
55     }
56 }
57
58 void callback(){
59     //Hàm được gọi mỗi khi có touch pin dc kích hoạt
60 }
61
62 void setup(){
63     Serial.begin(115200);
64     delay(1000); //chờ 1 khoảng nhỏ cho serial hoạt động
65
66     //mỗi lần thức dậy sẽ tăng biến này 1 lần và in ra
67     ++bootCount;

```

```

68 Serial.println("Boot number: " + String(bootCount));
69
70 //in ra nguồn đánh thức esp32
71 print_wakeup_reason();
72
73 //in ra chân touch đánh thức ESP32
74 print_wakeup_touchpad();
75
76 //Cài đặt ngắt cho chân Touch Pad 3 (GPIO15)
77 touchAttachInterrupt(T3, callback, Threshold);
78
79 //Cấu hình chế độ thức dây là touchpad
80 esp_sleep_enable_touchpad_wakeup();
81
82 Serial.println("Going to sleep now");
83 delay(1000);
84 Serial.flush();
85 // bắt đầu vào chế độ ngủ
86 esp_deep_sleep_start();
87 }
88
89 void loop(){
90 //This is not going to be called
91 }
```

Chúng ta sẽ sử dụng hàm `esp_sleep_enable_touchpad_wakeup();` để cấu hình thức dậy cho ESP32 bằng Touch.

Sau khi cấu hình xong esp32 sẽ được đưa về trạng thái ngủ bằng hàm

`esp_deep_sleep_start();`

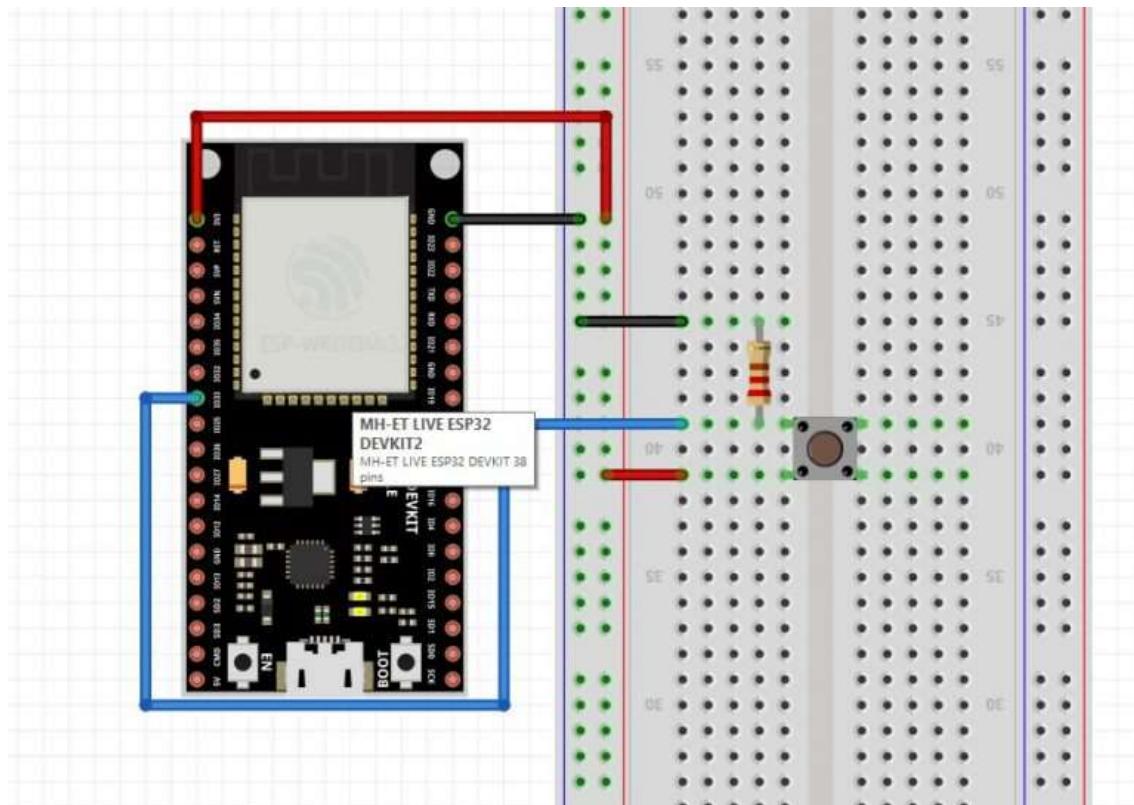
Mỗi khi chạm vào chân Touch, ESP32 sẽ wake up và in ra kiểu wake up và chân wake up.

Lập trình ESP32 External Wake Up

Các chân External Wake up phải là các chân của bộ RTC (Bởi vì chỉ bộ này hoạt động trong chế độ Deep Sleep). Chúng ta cần treo trả bên ngoài cho nút nhấn, bởi vì khi vào Sleep tất cả các trả treo bằng lệnh PinMode hay các lệnh liên quan tới IO đều không hoạt động.

Nếu không treo trở cho chân IO, chân đó sẽ có mức điện áp là Floating (trôi nổi) dẫn tới kết quả sẽ bị sai.

Sơ đồ nguyên lý



esp32 ext wake up

Code và giải thích code

Full Code

```

02 #include <Arduino.h>
03
04 #define BUTTON_PIN_BITMASK 0x200000000 // 2^33 in hex
05
06 RTC_DATA_ATTR int bootCount = 0; // biến này lưu tại bộ nhớ I
07
08 // in ra nguồn đánh thức
09 void print_wakeup_reason(){
10     esp_sleep_wakeup_cause_t wakeup_reason;
11     wakeup_reason = esp_sleep_get_wakeup_cause();
12     switch(wakeup_reason)
13     {

```

```

14   case ESP_SLEEP_WAKEUP_EXT0 :
15     Serial.println("Wakeup caused by external signal using
16       break;
17   case ESP_SLEEP_WAKEUP_EXT1 :
18     Serial.println("Wakeup caused by external signal using
19       break;
20   case ESP_SLEEP_WAKEUP_TIMER :
21     Serial.println("Wakeup caused by timer");
22     break;
23   case ESP_SLEEP_WAKEUP_TOUCHPAD :
24     Serial.println("Wakeup caused by touchpad");
25     break;
26   case ESP_SLEEP_WAKEUP_ULP :
27     Serial.println("Wakeup caused by ULP program");
28     break;
29   default :
30     Serial.printf("Wakeup was not caused by deep sleep: %d"
31     break;
32   }
33 }
34
35 void setup(){
36   Serial.begin(115200);
37   delay(1000); //chờ 1 khoảng nhỏ cho serial hoạt động
38
39   //mỗi lần thức dậy sẽ tăng biến này 1 lần và in ra
40   ++bootCount;
41   Serial.println("Boot number: " + String(bootCount));
42
43   //in ra nguồn đánh thức esp32
44   print_wakeup_reason();
45
46   //cấu hình chế độ thức dậy là ext0
47   //chế độ này chỉ bật được khi bộ RTC còn hoạt động
48   esp_sleep_enable_ext0_wakeup(GPIO_NUM_33,1); //1 = High, 0
49
50
51   //cấu hình chế độ thức dậy là ext1
52   //chế độ này có thể bật ngay cả khi RTC không còn hoạt động
53   //esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK,ESP_EXT1_
54
55   Serial.println("Going to sleep now");
56   delay(1000);
57   Serial.flush();
58   // bắt đầu vào chế độ ngủ
59   esp_deep_sleep_start();
60 }
61
62 void loop(){
63   //This is not going to be called

```

Với thức dậy bằng Ext0 chúng ta sử dụng hàm

`esp_sleep_enable_ext0_wakeup(GPIO_NUM_33,1);` với 2 tham số `GPIO_NUM_33` là chân GPIO sử dụng trong ngắn và 1 là mức sảy ra ngắn (khi IO đó có điện áp là Vcc).

Với thức dậy bằng Ext1 chúng ta sử dụng hàm

`esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK,ESP_EXT1_WAKEUP_ANY_HIGH);` trong đó:

- `BUTTON_PIN_BITMASK` được định nghĩa bằng lệnh `#define BUTTON_PIN_BITMASK 0x2000000000` (bit thứ 32 là 1 khi chuyển đổi qua BIN)
- `ESP_EXT1_WAKEUP_ANY_HIGH` là kiểu thức dậy khi bất kì chân nào được có giá trị là 1.

Kết

Năm vũng được các chế độ Sleep Mode của ESP32 thì mới có thể làm được các ứng dụng sử dụng Pin, accquy... Đó là tiêu chuẩn bắt buộc cho các thiết bị di động. Thực ra ESP32 Sleep Mode không khó, bạn hoàn toàn có thể làm chủ được.

Nếu thấy bài này có ích hãy chia sẻ cho cộng đồng nhé. Đừng quên ra nhập [Hội Anh Em Nghiên Lập trình](#) để kết nối với những người bạn cùng đam mê.

3.5/5 - (4 bình chọn)

Related Posts:

1. [Bài 1: Lập trình ESP32 Webserver chế độ Wifi Station bật tắt Led](#)
2. [Bài 7: Lập trình ESP32 Touch Pin bật tắt led với một cái chạm tay](#)
3. [Bài 6: Lập trình ESP32 Timer Millis và ngắn Timer](#)
4. [Bài 5: Lập trình ESP32 ngắn ngoài EXTI](#)

5. Hướng dẫn cài đặt Platform IO lập trình ESP32

6. Tổng quan về sơ đồ chân ESP32 và ngoại vi



KHUÊ NGUYỄN

Chỉ là người đam mê điện tử và lập trình. Làm được gì thì viết cho anh em xem thôi. :D

Trả lời

Email của bạn sẽ không được hiển thị công khai. Các trường bắt buộc được đánh dấu *

Bình luận *

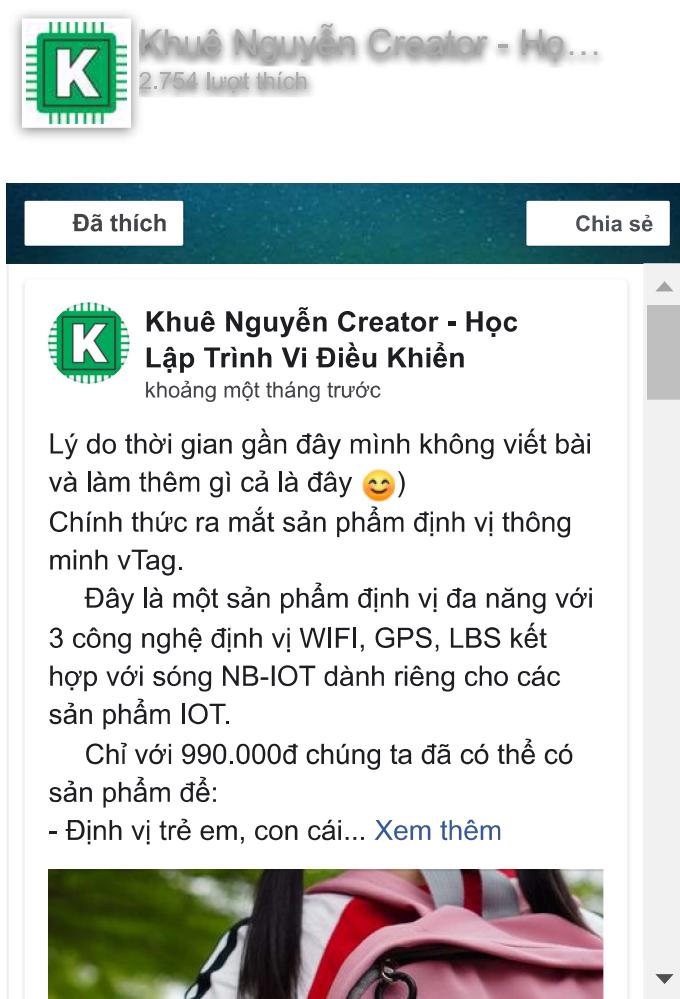
Tên *

Email *

Trang web

PHẢN HỒI

Fanpage



Khuê Nguyễn Creator - Họ...
2.754 lượt thích

Đã thích Chia sẻ

Khuê Nguyễn Creator - Học Lập Trình Vi Điều Khiển
khoảng một tháng trước

Lý do thời gian gần đây mình không viết bài và làm thêm gì cả là đây 😊)
Chính thức ra mắt sản phẩm định vị thông minh vTag.

Đây là một sản phẩm định vị đa năng với 3 công nghệ định vị WIFI, GPS, LBS kết hợp với sóng NB-IOT dành riêng cho các sản phẩm IOT.

Chỉ với 990.000đ chúng ta đã có thể có sản phẩm đẽ:

- Định vị trẻ em, con cái... [Xem thêm](#)



Bài viết khác



Lập trình 8051 - AT89S52

Khuê Nguyễn Creator



Bài 1: Tổng quan về 8051 và chip AT89S51 - 52

Tổng quan về 8051

8051 là một dòng chip nhập môn cho lập trình viên nhúng, chúng được sử...

[ĐỌC THÊM](#)

Lập trình STM32 và CubeMX



Khuê Nguyễn Creator



Lập trình STM32 HID Host giao tiếp với chuột và bàn phím

Lập trình STM32 USB HID Host giao tiếp với chuột và bàn phím máy tính

Trong bài này chúng ta sẽ cùng học STM32 HID Host, biến STM32 giống như...

[ĐỌC THÊM](#)



Lộ trình học lập trình nhúng từ A tới Z

Lập trình nhúng là một ngành có cơ hội nhưng cũng đòi hỏi nhiều kiến...

3 COMMENTS

[ĐỌC THÊM](#)

Lập trình STM32 và CubeMX

Lập trình STM32F407 SDIO đọc dữ liệu thẻ nhớ

Lập trình STM32 SDIO đọc ghi dữ liệu vào thẻ nhớ SD card

Trong bài này chúng ta cùng học cách lập trình STM32 SDIO, một chuẩn giao...

[ĐỌC THÊM](#)

Lập trình STM32 và CubeMX



Khuê Nguyễn Creator



Lập trình STM32F407 DAC chuyển đổi số sang tương tự

Lập trình STM32 DAC tạo sóng hình Sin trên KIT STM32F407 Discovery

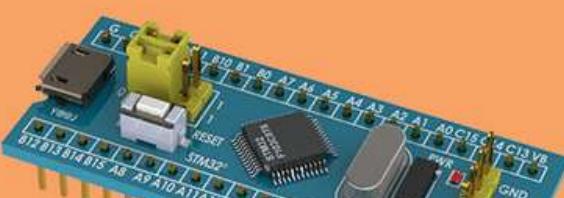
Trong bài này chúng ta sẽ cùng nhau tìm hiểu STM32 DAC với KIT STM32F407VE...

[ĐỌC THÊM](#)

Lập trình STM32 và CubeMX



Khuê Nguyễn Creator



Sử dụng hàm printf để in Log khi Debug trên STM32

Hướng dẫn sử dụng printf với STM32 Uart để in Log trên Keil C

Trong bài này chúng ta sẽ học cách retarget hàm printf của thư viện stdio...

3 COMMENTS

[ĐỌC THÊM](#)

ESP32 và Platform IO

Khuê Nguyễn Creator

Bài 9 WIFI: Lập trình ESP32 OTA nạp firmware trên Internet

Lập trình ESP32 FOTA nạp firmware qua mạng Internet với OTA Drive

Trong bài này chúng ta sẽ học cách sử dụng ESP32 FOTA (Firmware Over The...

4 COMMENTS

[ĐỌC THÊM](#)

Lập trình Nuvoton



Khuê Nguyễn Creator



Cài đặt SDC Complier và Code:Blocks IDE

Hướng dẫn cài đặt SDCC và Code::Blocks lập trình Nuvoton

Ở bài này chúng ta sẽ cài đặt các công cụ cần thiết cho việc...

ĐỌC THÊM



Blog này làm ra để lưu trữ tất cả những kiến thức, những câu chuyện của mình. Đôi khi là những ý tưởng nhất thời, đôi khi là các dự án tự mình làm. Chia sẻ cho người khác cũng là niềm vui của mình, kiến thức mỗi người là khác nhau, không hẳn quá cao siêu nhưng sẽ có lúc hữu dụng.

Liên Kết

Nhóm: Nghiện Lập Trình

Fanpage: Khuê Nguyên Creator

My Shop

Thông Tin

Tác Giả

Chính Sách Bảo Mật



Copyright 2022 © Khuê Nguyễn