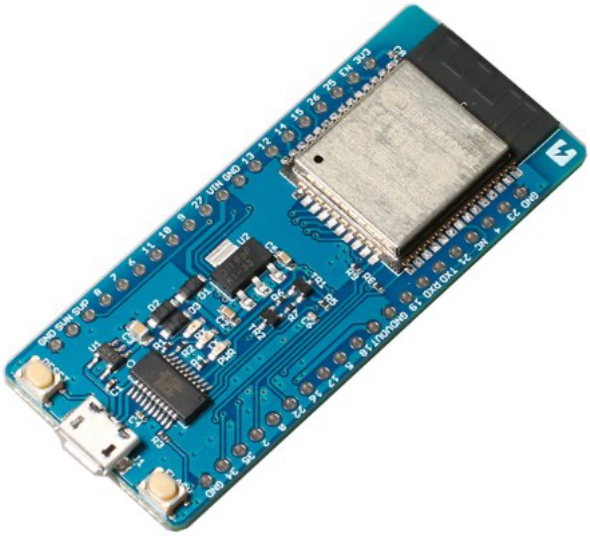
ESP-WROOM-32に関するTIPS



ESP-WROOM-32は、Xtensa Dual-Core 32-bit LX6 マイクロプロセッサを搭載する、上海のESPRESSIF社が開発した無線モジュールです。WiFiとBluetooth v4.2に対応しています。 開発ボードはスイッチサイエンスでも販売されている ESPr Developer 32の他にも、ESPRESSIF社からESP32-DevKitCなどが発売さています。

本ページでは、無線モジュール ESP-WROOM-32の各種機能をArduinoとして使うにあたってのサンプルスケッチや注意点などを解説します。ESP32と呼ぶときはコントローラーそのもの、無線モジュール全体を指すときはESP-WROOM-32と記述しています。

ESP-WROOM-32は本記事執筆時点(2017/4~5月)でも頻繁にライブラリにアップデートがあります。記事中の内容は最新の状態とは異なる場合があるため注意してください。特に、GitHubのコードの特定行への参照は古いものの可能性が高くなっています。

本ページで解説しているピン設定はスイッチサイエンスの ESPr® Developer 32を想定しています。他社製開発ボードでは異なる場合があるため注意してください

詳しくはボートの別種から適宜読み替えてください。ESPr Developer 32は標準の開発ボード(varaiants/esp32/)とピン配列はおなじになっています。

資料について

- ESP32-WROOM-32 Datasheet(PDF) 一番基本的なデータシートです。
- ESP32 Technical Reference Manual(PDF) ブロック図、レジスタなどの解説が含まれるテクニカルリファレンスマニュアルです。
- ESP32 Hardware Design Guidelines(PDF) ESP-WROOM-32モジュールやDevkitCのハードウェアデザインガイドラインです。

ESP-WROOM-32のピンアサイン

以下はESP32-WROOM-32 Datasheet(PDF)より抜粋。

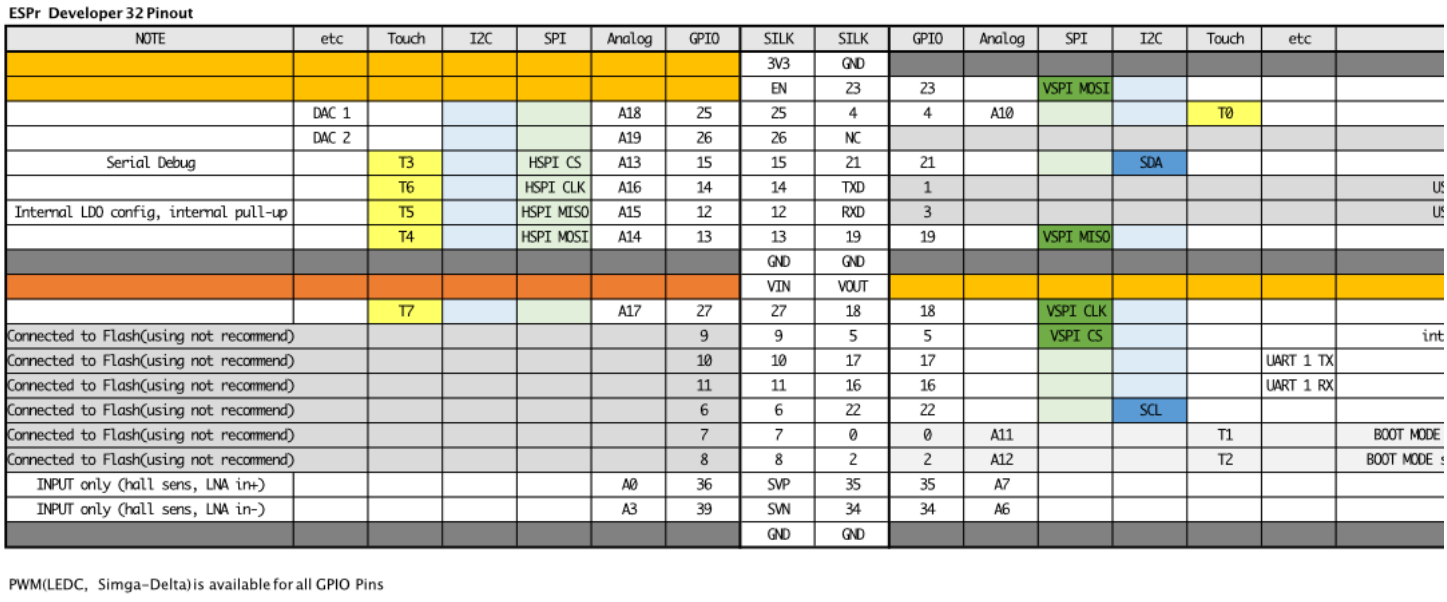
Name	No.	Type	Function
GND	1	P	Ground
3V3	2	P	Power supply
EN	3	I	Module-enable signal. Active high.
SENSOR_VP	4	I	GPIO36, ADC1_CH0, RTC_GPIO0
SENSOR_VN	5	I	GPIO39, ADC1_CH3, RTC_GPIO3
IO34	6	I	GPIO34, ADC1_CH6, RTC_GPIO4

IO35	7	I	GPIO35, ADC1_CH7, RTC_GPIO5
IO32	8	I/O	GPIO32, XTAL_32K_P (32.768kHz crystal oscillator input), ADC1_CH4, TOUCH9, RTC_GPIO9
IO33	9	I/O	GPIO33, XTAL_32K_N (32.768kHz crystal oscillator output), ADC1_CH5, TOUCH8, RTC_GPIO8
IO25	10	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
IO26	11	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
IO27	12	I/O	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV
IO14	13	I/O	GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2
IO12	14	I/O	GPIO12, ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI, HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3
GND	15	P	Ground
IO13	16	I/O	GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER
SHD/SD2*	17	I/O	GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD
SWP/SD3*	18	I/O	GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD
SCS/CMD*	19	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD, U1RTS
SCK/CLK*	20	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK, U1CTS
SDO/SD0*	21	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0, U2RTS
SDI/SD1*	22	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1, U2CTS
IO15	23	I/O	GPIO15, ADC2_CH3, TOUCH3, MTDO, HSPICS0, RTC_GPIO13, HS2_CMD, SD_CMD, EMAC_RXD3
IO2	24	I/O	GPIO2, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0
IO0	25	I/O	GPIO0, ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK
IO4	26	I/O	GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER
IO16	27	I/O	GPIO16, HS1_DATA4, U2RXD, EMAC_CLK_OUT
IO17	28	I/O	GPIO17, HS1_DATA5, U2TXD, EMAC_CLK_OUT_180
IO5	29	I/O	GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK
IO18	30	I/O	GPIO18, VSPICLK, HS1_DATA7
IO19	31	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
NC	32	-	-
IO21	33	I/O	GPIO21, VSPIHD, EMAC_TX_EN
RXD0	34	I/O	GPIO3, U0RXD, CLK_OUT2
TXD0	35	I/O	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
IO22	36	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
IO23	37	I/O	GPIO23, VSPID, HS1_STROBE
GND	38	P	Ground
GND	39	P	Ground

*がついているピン(GPIO6~11)はモジュール内部でSPIフラッシュメモリに接続されているため、他の用途での利用は推奨されません。また、後述するピンマトリクス機能においても、GPIO6~11を別の機能に割り当てることは推奨されません。

ESPr Developer 32のピンアサインについて

詳細な機能を省いた簡易版について画像を作成しました。



pdf版

IO0/IO2ピンのNOTEが入れ替わっていたので修正しました(2021/06/09)

GPIOマトリクス機能について

各種ペリフェラルを説明する前に、本機能について少し説明します。Arduinoからは離れた話なので、Arduinoとして少し使うぶんには気にしなくとも問題ありません。詳細については、説明しきれない部分が多いため、以下の資料を御覧ください。

ESP32 Pinoutの`Note.12`にある通り、内蔵ペリフェラル用の信号は設定によって好きなピンに再割り当てすることが可能です。

回路ブロック図に関しては、ESP32 Technical Reference Manualの`4. IO_MUX and GPIO Matrix`が詳しいです。

ESP32には「(物理的な)ピン」「IO MUX(マルチプレクサ)」「GPIO マトリックス」という3つの回路があります。ピンに対してはIO MUXが接続されています。一つのピンは内蔵機能またはGPIOとして割り当てる事ができます。これをIO MUXにて行います。

IO MUXにてGPIOに割り当てられたピンは、GPIO マトリックスにて機能の割り当てを行うことができます。GPIOマトリックスは、入力と出力を文字通り自由に組み替えら得れる回路になっており、ピンへの入力を内蔵ペリフェラルピンに割り当てたり、逆にペリフェラルピンを出力ピンに接続したりできます。

このGPIOマトリックスにて操作できるものは、基本的に内蔵ペリフェラルほぼ全てです。よって、他のデバイスとのコミュニケーションにはおおよそ好きなピンを利用することができます。

ただし、以下の制約があります。

- GPIO 34~39 ピンは出力には使えない
- GPIO 0,2 ピンは回路的にブートモードの設定に利用されることがあり、実際ほぼこの通り使われていると考えます
- GPIO 1,3 ピンは回路的にUART通信(U0RXD、U0TXD)に利用されることがあり、実際ほぼこの通り使われていると考えます
- GPIO 12 は内蔵LDOの電圧設定に利用されることがある
- GPIO 15 はU0TXDピンのデバッグログの設定に利用されることがある
- Ethernet, SDIO, SPI, JTAG, UARTなどの高速信号は、パフォーマンス低下を回避するためにGPIOマトリクスを介さないで接続可能なものがあり、その場合は出力ピンが固定される
- GPIO 6~11 はESP-WROOM-32モジュール内部では既にFLASHメモリと接続されている

特に、GPIO 0,1,2,3に関しては電圧を固定してしまうと以降ArduinoIDEなどからのスケッチの書き込みができなくなるため、注意が必要です。更には、ESP-WROOM-32モジュールには内蔵FLASHメモリのために幾つかの回路が配線済みであるため、SDIO Slaveの機能やParallel QSPIといった機能は使えません。

SPI通信

内部の仕様について

ESP-WROOM-32に搭載されているコントローラーにはSPIモジュールが3つ内蔵されています。それぞれ、

- SPI
- HSPI
- VSPI

の名称がつけられています。このうち、SPIに関しては上記ピンアサインの項目の通りFSPIとして内蔵FLASHとの接続に使われています。よって、普段周辺機器を接続する際にはHSPIもしくはVSPIの回路を利用することになります。この2つに関しては、SlaveとMasterどちらも利用可能です。

一方で、ArduinoのSPIライブラリは、ライブラリ中に定義されるように、VSPI回路を利用します。更に、独自にSPIクラスをインスタンス化すると、こちらにあるようにHSPIが利用されます。例えば、`SPIclass SPI2;`とやると、`SPI2`インスタンスはHSPI回路を利用し、標準の`SPI`インスタンスとは別の回路で動きます。

各SPIモジュールのピンアサインは以下の通りです。ただし、GPIOマトリックスの項目で説明した通り、比較的自由にリマッピング可能です。

- HSPI - SCK,MISO,MOSI,SS = 14,12,13,15
- VSPI - SCK,MISO,MOSI,SS = 18,19,23,5

ピンを指定するときは、`SPI.begin(SCK, MISO, MOSI, SS);`というふうに引数を指定します。

サンプルスケッチ

以上の仕様と、GPIOマトリックス機能を合わせると、ものによってはブレッドボードに上手く挿すだけでジャンパワイヤなしに通信が可能です。

以下のサンプルスケッチは、液晶モジュールとの通信をジャンパワイヤなしに行うというものです。

ESPr® Developer 32 向けSPI通信サンプルスケッチ (Github)

I²C通信

内部の仕様について

SPI通信と同じく、内蔵回路は2つ(I2C0とI2C1)あります。基本的なピンアサインは以下のとおりです。

- I2C0 - SDA,SCL = 21,22
- I2C1 - 未定義

未定義、というのも、I²C通信の回路のSDA、SCLピンはGPIOマトリクスのみに接続されているため、IO MUXに直結(=高速通信のために物理ピンに直結)していません。GPIOマトリクス経由で自由に再マッピングできます。もしI2C1回路を使う場合は、このあたりを参考に、`Towire Wire2(1);`などと宣言すると良さそうです。

ピンを指定するときは`Wire.begin(SDA, SCK);`もしくは`Wire.begin(SDA, SCL, FREQ);`というふうに引数を指定します。`FREQ`は`uint32_t`型の動作周波数で、標準では100kHzです。

サンプルスケッチ

SPI通信のときと同じように、ものによってはブレッドボードに上手く挿すだけでジャンパワイヤなしに通信が可能です。

以下のサンプルスケッチは、温度センサとの通信をジャンパワイヤなしに行うというものです。

ESPr® Developer 32 向けI2C通信サンプルスケッチ (Github)

GPIO

基本的に一般的なArduinoと同じです。3.3V動作であり5Vトレラントでない点、および他のマトリクス機能との衝突を起こす可能性がある点に注意すると良いです。

サンプルスケッチ

ADC(アナログ入力)

ADCに使えるピンを以下の表に示します。

定数	GPIOピン
A0	36
A3	39
A4	32
A5	33
A6	34
A7	35
A10	4
A11	0
A12	2
A13	15
A14	13
A15	12
A16	14
A17	27
A18	25

A19	26

※A1,A2は定義されていません。本来ならGPIO37、38ピンで利用できますが、後述するLow-Noise Amplifier回路を利用するためにすでに部品が配線されています。 ※GPIO0、2に関しては書き込みモードの制御、GPIO15,12も他の機能で使われることがあります。

内部の仕様について

ESP32には逐次比較型(SAR)ADCモジュールが2つ内蔵されています。分解能は9〜12bitで、デフォルトは12bitです。(参照)

SAR ADC1 には、GPIO32~39に相当するピンと、後述するホールセンサが接続されています。

SAR ADC2 には、それ以外のピンと、deep sleep機能用の入力などが接続されています。

また、ADCの入力には減衰器を設定することができます。実のところ、ESP32のADCモジュールは0〜1 Vの間でAD変換を行っています。このモジュールに例えば11db(約1/3.6倍)の減衰器を設定することで、0〜3.6 Vの間のAD変換ができるようになります。

デフォルトでは11dBの減衰が設定され、12bitでの最大値は3.6 Vを示しています。 また、現在AD変換の結果は少し誤差があるとのことです(公式スレッド参照)。 他にも議論が見受けられ(例1,例2)、状況によって値が変わる可能性がある点に注意したほうが良いでしょう。

このADCの補正に関しては色々議論が進んでいるようです。参考情報として、電源電圧3.31 VでA0ピン(GPIO36)に1.667 Vを入力した時のADCの値は、1950程度でした。値が4095の時3.50 Vになる計算です。この値も何度かのアップデートで変わっている可能性があります。

setupのブロックで `analogSetAttenuation()` 関数を呼ぶことで減衰率の設定ができます。例えば減衰無し(入力電圧範囲 0〜1 V)であれば、 `analogSetAttenuation(ADC_0db);` と設定します。詳しくはこのあたりを参照ください。

また、ADC2回路を利用するピンは、今のところWiFi機能と並行して利用できず、常に値が4095になります(参照)。将来的に修正するようですが、注意してください。

サンプルスケッチ

DAC(アナログ出力)

いわゆるArduinoUNOなどで実行する `analogWrite()` ではなく、ArduinoDUEのDACピンのようにアナログ電圧が直接ピンから出てくる機能です。

内部の仕様について

ESP32は2chの8bit DACを内蔵しています。出力ピンはGPIO25,26ピンの位置に固定されています。

定数	GPIOピン
DAC1	25
DAC2	26

`dacWrite(pin, value)` と引数を渡します。pinはDAC1,DAC2を指定します。valueは0から255までの値です。

サンプルスケッチ

PWM等

特にLEDC機能を利用する場合、いわゆるArduinoUNOの `analogWrite()` と同様の振る舞いをします。つまり、LED調光などを想定した一般的なPWMということです。SigmaDelta変調に関しての詳細な説明は省きます。

内部の仕様について

SigmaDelta変調に関しては、0~7の8ch、1220~312500 Hz の範囲で動作可能の模様です。IO MUX経由で物理ピンにアタッチして使います。この際、特に推奨されるピンはありません。IO MUX内部では `gpio_sdX_out` という呼称で扱われています。

LEDC機能は、0~15の16chです。

余談ですが、現在はtone関数は利用できません。代替機能として、 `ledcWriteTone(ch, 440);` と関数を呼ぶと、chにアタッチされたピンに440Hzで矩形波が出力されます。

更には、`ledcWriteNote(ch, NOTE_A, 4);`と関数を呼ぶと、chにアタッチされたピンにラの音を第4オクターブの高さで鳴らすことができます。つまり440Hzの矩形波が出力されます。 音階は`hardware\espressif\esp32\cores\esp32\esp32-hal-ledc.h`において

```
typedef enum {
    NOTE_C, NOTE_Cs, NOTE_D, NOTE_Eb, NOTE_E, NOTE_F, NOTE_Fs, NOTE_G, NOTE_Gs, NOTE_A, NOTE_Bb, NOTE_B, NOTE_MAX
} note_t;
```

と定義されています。

サンプルスケッチ

LED C機能を使ったサンプルスケッチがexamplesにあります。LEDCSoftwareFadeという名前です。

SigmaDelta変調を使ったサンプルスケッチもexamplesにあります。SigmaDeltaという名前です。

ホールセンサ

ESP32にはホールセンサ(磁気センサー)が内蔵されています。自由度は1です。

内部の仕様について

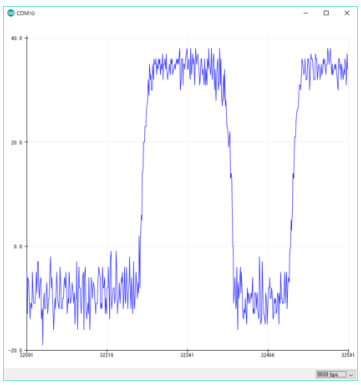
ホール効果によって発生する電圧を先のADC回路にて読み取っています。生じる電圧は小さいので、内蔵のLow-Noise Amplifierを利用することが回路的には可能ですが、Arduinoライブラリではサポート外のようです。(参照)

ホールセンサの出力は、内蔵LNAに接続されている、つまりGPIO36,37(SENSOR_VP,SENSOR_VN)につながっているため、他の回路とは基本的に排他利用となっています。

サンプルスケッチ

サンプルスケッチはHallsSensorという名前で用意されています。

磁石を近づけたり遠ざけたりした時の実行例



温度センサー

内部の仕様について

サンプルスケッチ

タッチセンサ

タッチセンサとして使える入力を以下に示します。

定数	GPIOピン
T0	4
T1	0

T2	2
T3	15
T4	13
T5	12
T6	14
T7	27
T8	33
T9	32

例によってGPIO0,2はブート設定で使われることがあったり、GPIO15はデバッグログの設定ピンだったり、GPIO32,33はクリスタル用(ESPr 32だと基板裏面に配置)であったりするため、利用するピンには気をつけなければいけません。

`touchRead(T0)` とすることで値が帰ってきます。この値は、ピンに触っているときとそうでないときで大きく異なるので、これを利用してタッチ判定ができます。

内部の仕様について

ピンに対して少し信号を出力した直後にピンの状態を読み取ることで、容量性の負荷が接続されているかどうかを見ています。現在ADCがWiFiと共存できないという仕様がありますが、タッチセンサとはせず使うことができます。

サンプルスケッチ

いくつか公式でサンプルスケッチが用意されている ので、書き込むことで動作確認ができます。

ULP コプロセッサとdeep sleep

ESP32はデュアルコアですが、これらとは別に低消費電力用にもう一つプロセッサを載せています。これをULP コプロセッサなどと呼びます。スイッチサイエンスのESPr Developerの基板裏面に存在するクリスタル用のパッドはこの機能向けに設けられているため、関連事項として説明します。

ただ、Arduino IDEから利用する分にはほぼ気にする必要がありません。まず、現在のところこのコプロセッサのプログラミングはアセンブラで行う必要があります。

詳しくはesp-idfのプログラミングガイドやesp-idfのサンプルプログラム をご覧ください。

内部の仕様について

ULPコプロセッサは、RTCと呼ばれる一連の内部一つの機能として呼称されます。他には、PMU(電源)、RTCリカバリメモリが存在します。

このRTCメモリは2種類存在し、それぞれFASTメモリとSLOWメモリと呼ばれます。どちらも8KBのSRAMです。(ちなみに本体スペックの520KBのうち8KBはこのFASTメモリのことを指します)

FASTメモリははデータストレージなどに利用でき、Deep-sleepからのRTC boot時などに利用できます。SLOWメモリはDeep-sleep中にコプロセッサのほうから利用されます。

また、ESP32は以下のような動作モードを持っています。

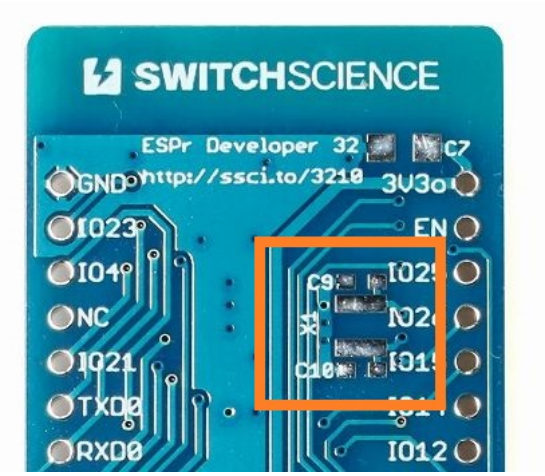
- active mode
  - 無線機能ON。チップは受信送信待機が可能。
- modem-sleep mode
  - CPUは動作し、かつクロックは設定可能。WiFiやBluetoothベースバンドは無効。
- light-sleep mode
  - CPUは動作を一時停止。RTCメモリとRTC周辺機器つまりULPコプロセッサは動作。全Wake-upイベントが有効。(MAC, Host, RTCタイマ, 外部割り込み)
- deep-sleep mode
  - RTCメモリとRTC周辺機器のみ動作。WiFiやBluetoothの接続情報はRTCメモリに保存。ULPコプロセッサは動作可能。
- hibernation mode
  - 内部8MHz発振子とULPコプロセッサも動作停止。RTCリカバリメモリの電源OFF。低速なRTCタイマーと一部のRTC GPIOのみ有効。RTCタイマーもしくはRTC GPIOはHibernation Modeからの復帰に利用可能。

RTC用のシステムクロックは5種類のソースから選ぶことができます。

- 外部低速(32 kHz)水晶発振子
- 外部推奨発信器の4分周
- 内蔵RC発振器(150kHz Typ. アジャスタブル)
- 内蔵8MHz発振器
- 内蔵31.25kHz(内蔵8MHzの256分周)

ここで、チップが通常動作していて、高速なCPUアクセスが必要な場合、アプリケーションは外部の高速なクリスタルの4分周もしくは内部8MHzを動作に利用可能です。 一方で、チップがlow-powerモードで動作している場合、アプリケーションは外部低速水晶発振器(32kHz)か内部RCクロックか内部31.25kHzを選択可能です。

背面(以下の画像)部のクリスタルパッドは、この外部用32KHzを接続する用に設計されています。 詳しくは、 ハードウェアデザインガイドラインの3.1.4.2をご覧ください。



サンプルスケッチ

esp-idfのサンプルプログラム をご覧ください。

Attachments (9)

Last modified on Jun 9, 2021 4:00:22 PM