

ディープスリープモードで省電力運用を行う

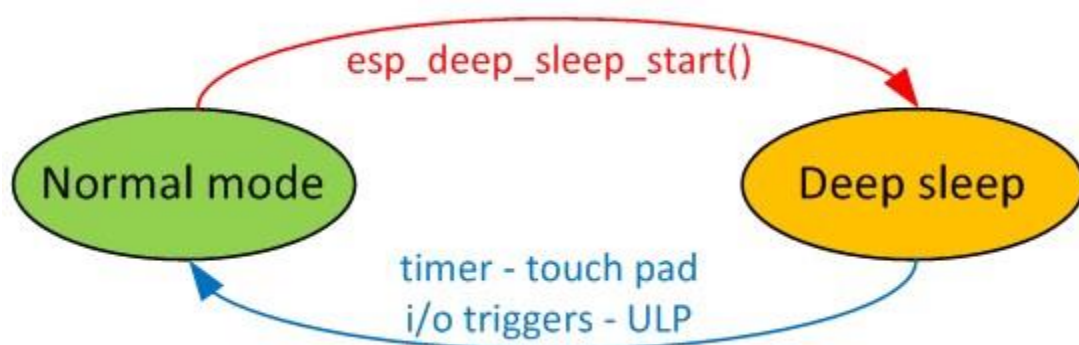
《何ができる?》 Ambient を使って IoT の仕組みが実現できるようになった。これで光センサーや環境センサー（温度・湿度・大気圧）、など様々なセンサーを接続してクラウドに刻々とデータを送ることができるようになったが、実際に観測地に設置する際に電源が確保できない場合もある。バッテリーなどで運用する際には、クラウドでの接続を行う以外は可能な限り省電力で運用し、電池などを持たせる必要がある。

通常センサーの値をアップロードする場合、頻繁に行うとしても 15 分に 1 回（1 時間に 4 回、1 日で 96 回）程度で、その際に数秒電源が供給されていれば良く、その間は「眠って」いて欲しいところだろう。

ESP32 では「ディープスリープモード」といって、API 関数呼び出しにより、アナログ入力などのセンサーデータを取得しクラウドに接続後、いったん「深い眠り (deep sleep)」に入らせることができる。

esp_deep_sleep_pd_config(設定値)
ディープスリープモードの各種設定値を定数で指定する。setup() メソッド内で一度指定する
esp_deep_sleep_enable_timer_wakeup(マイクロ秒)
次回起動(wake-up)する時間を指定する。
esp_deep_sleep_start()
ディープスリープをスタートさせる命令。これ以降、次回起動する時間まで省電力状態に移行する。

以下の様な遷移状態をもつ。今回はタイマーによるウェイクアップだが、ほかにもタッチセンサーやデジタル入力（タクトスイッチなど）をトリガー（引き金）として起動させることができる。



《サンプル①》ディープスリープのサンプルソース ※打ち込む必要はない、概念を理解できれば OK

```
1 #include "esp_sleep.h"
2
3 void setup() {
4     // put your setup code here, to run once:
5     Serial.begin(115200);
6     Serial.println("Start");
7
8     esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
9     esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_SLOW_MEM, ESP_PD_OPTION_OFF);
10    esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_FAST_MEM, ESP_PD_OPTION_OFF);
11    esp_sleep_pd_config(ESP_PD_DOMAIN_MAX, ESP_PD_OPTION_OFF);
12 }
13
14 void loop() {
15     // put your main code here, to run repeatedly:
16     delay(1000);
17     Serial.println("good night!");
18
19     esp_sleep_enable_timer_wakeup(10 * 1000 * 1000); // wakeup(restart) after 10secs
20     esp_deep_sleep_start();
21
22     Serial.println("zzz"); // ここは実行されない
23 }
```

〔1 行目〕

ディープスリープの API ライブラリの読み込み宣言を行っている。

〔8 行目から 11 行目〕

起動後 1 度しか実行されない setup() メソッド内の 8 行目から 11 行目でディープスリープモードの設定が行われている。

〔19 行目〕

繰り返し実行される loop() メソッド内の 19 行目で、次回起動する時間がセットされている。

引数はマイクロ秒のため、10 * 1000 ミリ * 1000 マイクロ=10 秒とコードを読みやすく記述している。

〔20 行目〕

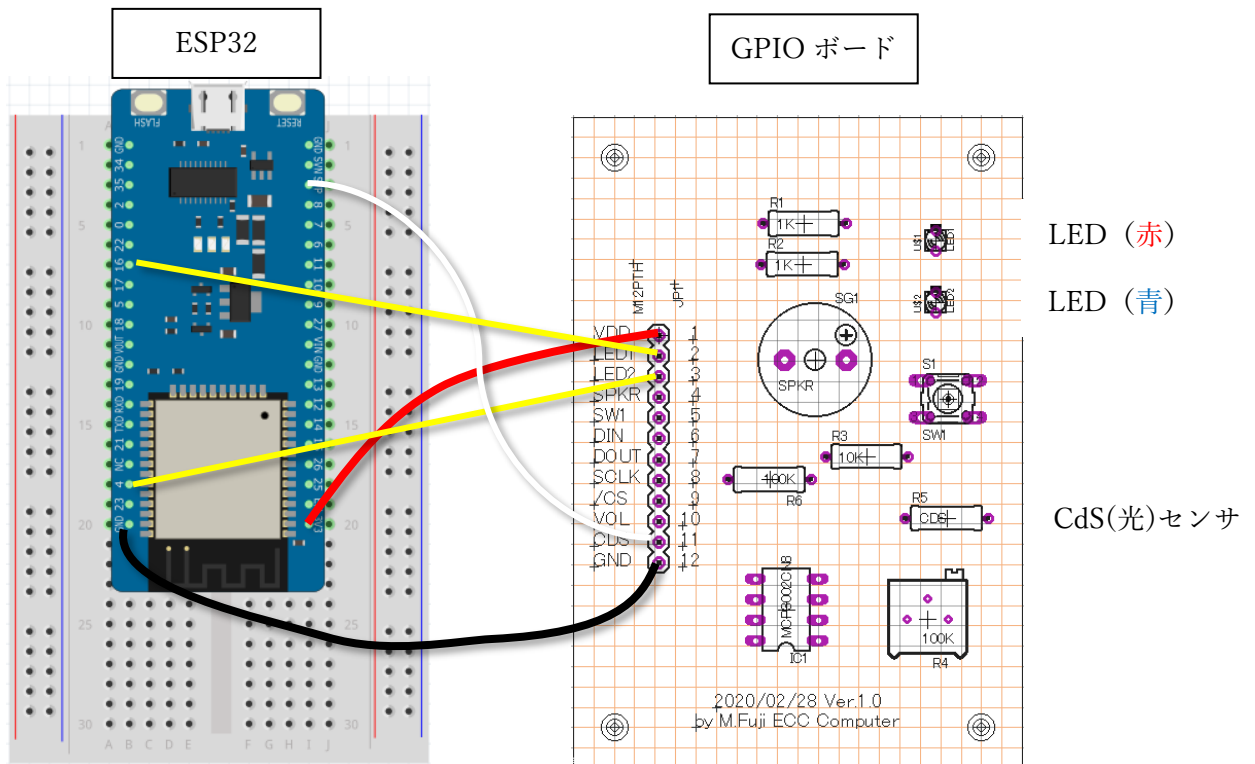
ディープスリープが開始されいったん電源オフのような状態になる。

そのため、22 行目のシリアルプリントはシリアルモニタには表示されず、10 秒後に「再起動」してプログラムがイチから実行される。

【目標】

ディープスリープモードのコード記述法と実際の振る舞いを見してみる。その後ディープスリープを用いた IoT プラットフォーム「Ambient」へのデータのエントリを行い、ブラウザで可視化する。

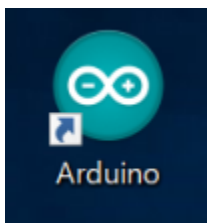
【1. ESP32 と電子工作部品との接続】



上記実体配線図を元に、5本のジャンパーコードの配線を行う。接続するピンは以下の一覧の通り。

- ① ジャンパーコード（白）ESP32 の I036 - GPIO ボード CDS (11 番ピン)
A0 ピン (I036 ピン) = 「SEN_p」と表記のあるピン (図で言うと向かって右上から 3 番目)
- ② ジャンパーコード（黄）ESP32 の I04 - GPIO ボード LED2 (3 番ピン)
- ③ ジャンパーコード（黄）ESP32 の I016 - GPIO ボード LED2 (2 番ピン)
- ④ ジャンパーコード（黒）ESP32 の GND - GPIO ボード GND (12 番ピン)
- ⑤ ジャンパーコード（赤）ESP32 の 3V3 - GPIO ボード VDD (1 番ピン)

【2. Arduino スケッチのサンプルプログラムを実行】



デスクトップのアイコンをダブルクリックして
Arduino IDE を起動する。

【課題①】プロジェクト名「kad30_cdsAmbient」

ディープスリープのサンプルと課題を行う前に、光センサーを用いて Ambient にデータをエントリする課題を行う。

Ambient サイトにアクセスして、自身のユーザーID でログインする

<https://ambidata.io/>

My チャンネルのページから自身のチャンネル ID とライトキーを控えておく。光センサーのデータをアップロードする際に必要となる。

Ambient 公開チャンネル Myチャンネル
ブログ ドキュメント お知らせ ログアウト kyoshida

Myチャンネル

ユーザーキー: [redacted]

writeKeyにコピー & ペーストする

チャンネル名	チャンネルID	リードキー	ライトキー	作成日
チャンネル7901	7901	c23ab7d3e389822f	8088d1c1d6d5349a	2018/11/20
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

channelIdにコピー & ペーストする

チャンネルを作る

AmbientData Inc. 利用

(前週に作成した課題とおなじチャンネル ID とライトキーをそのまま使う)

次に、前週の課題をコピーして、「kad30_cdsAmbient」を作る。

「kad30_cdsAmbient」には、以下の GPIO のためのグローバル変数を入れるようにする。

```
const int cdsPin = A0; //光センサー A0ピン  
const int bluePin = 4; //青色LED IO4ピン  
const int redPin = 16; //赤色LED IO16ピン
```

光センサーの値は、アナログ入力値をそのまま用いればよい。

10 秒おきに Ambient に send するプログラムにする。

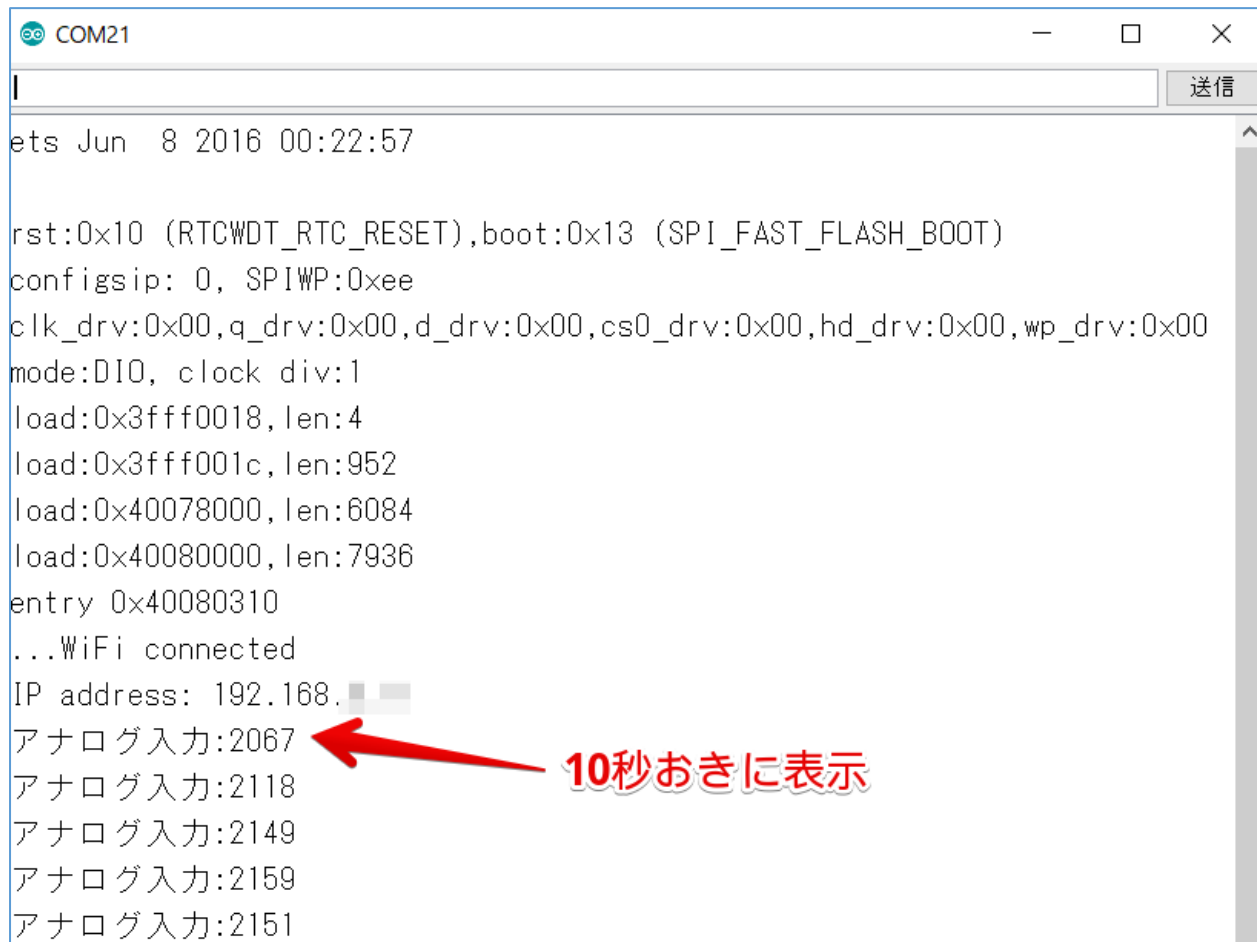
赤色 LED は、Wi-Fi の接続が始まった際に点灯する（点灯したままにする）。←setup() メソッド内

青色 LED は、Ambient に send する際に 200 ミリ秒点滅するようにする。青色 LED を点滅させる際には、自身で定義した ledFlash() 関数を用いる（前週の課題からコピーする）。

```
ledFlash(bluePin, 200); //送信時に 青色LED点滅
```

【実行結果 Arduino シリアルモニタ側】

シリアルモニタに以下の様な表示がされる。



The screenshot shows the Arduino Serial Monitor window titled 'COM21'. The text displayed is as follows:

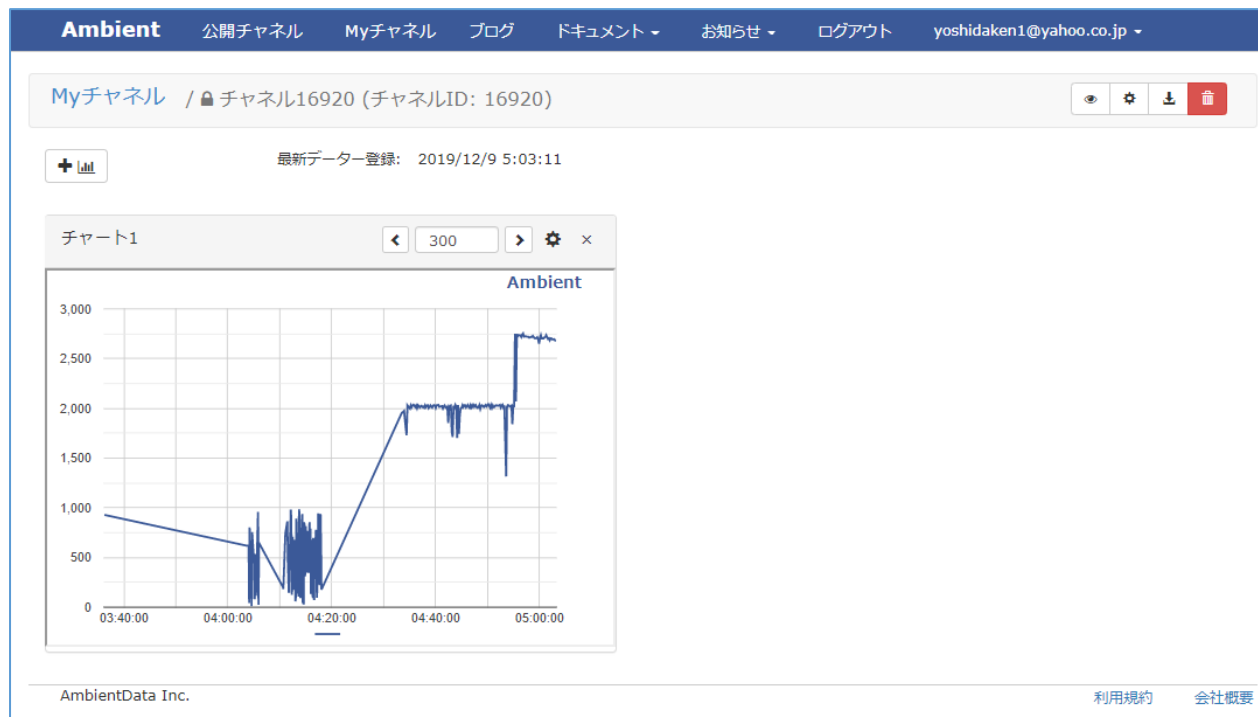
```
ets Jun  8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
config: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:952
load:0x40078000,len:6084
load:0x40080000,len:7936
entry 0x40080310
...WiFi connected
IP address: 192.168.███
アナログ入力:2067
アナログ入力:2118
アナログ入力:2149
アナログ入力:2159
アナログ入力:2151
```

A red arrow points from the text '10秒おきに表示' to the first line of sensor data, 'アナログ入力:2067'.

【実行結果 Ambient クラウド側】

ブラウザで以下の様なグラフ表示となる（0～4095 内の値を取る）。



今回はグラフの名前は「チャート1」にする。チャート2は削除しておくこと。

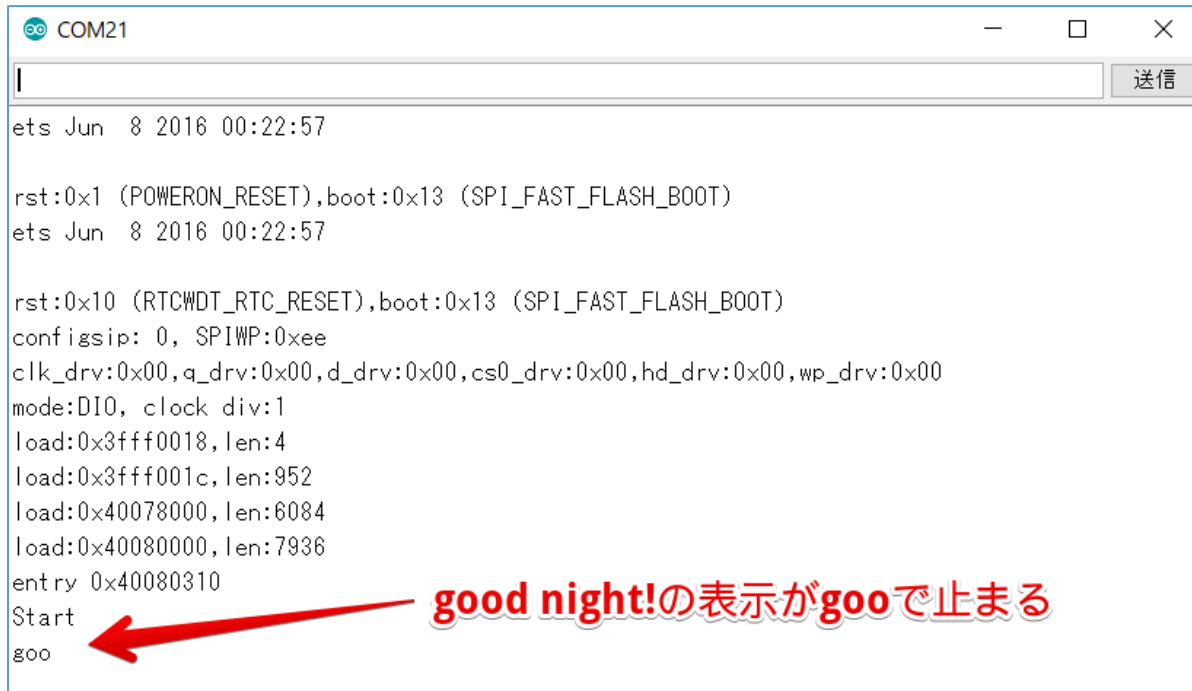
【課題②】 プロジェクト名「kad31_deepSleep」

まずはサンプル①のソースをコピー&ペーストして作成する。

```
1  #include "esp_sleep.h"
2
3  void setup() {
4      // put your setup code here, to run once:
5      Serial.begin(115200);
6      Serial.println("Start");
7
8      esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
9      esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_SLOW_MEM, ESP_PD_OPTION_OFF);
10     esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_FAST_MEM, ESP_PD_OPTION_OFF);
11     esp_sleep_pd_config(ESP_PD_DOMAIN_MAX, ESP_PD_OPTION_OFF);
12 }
13
14 void loop() {
15     // put your main code here, to run repeatedly:
16     delay(1000);
17     Serial.println("good night!");
18
19     esp_sleep_enable_timer_wakeup(10 * 1000 * 1000);
20
21     esp_deep_sleep_start();
22
23     Serial.println("zzz"); // ここは実行されない
24 }
```

【サンプル① 実行結果】

シリアルモニタを表示して確認する。一番初めは以下の様な表示になる。

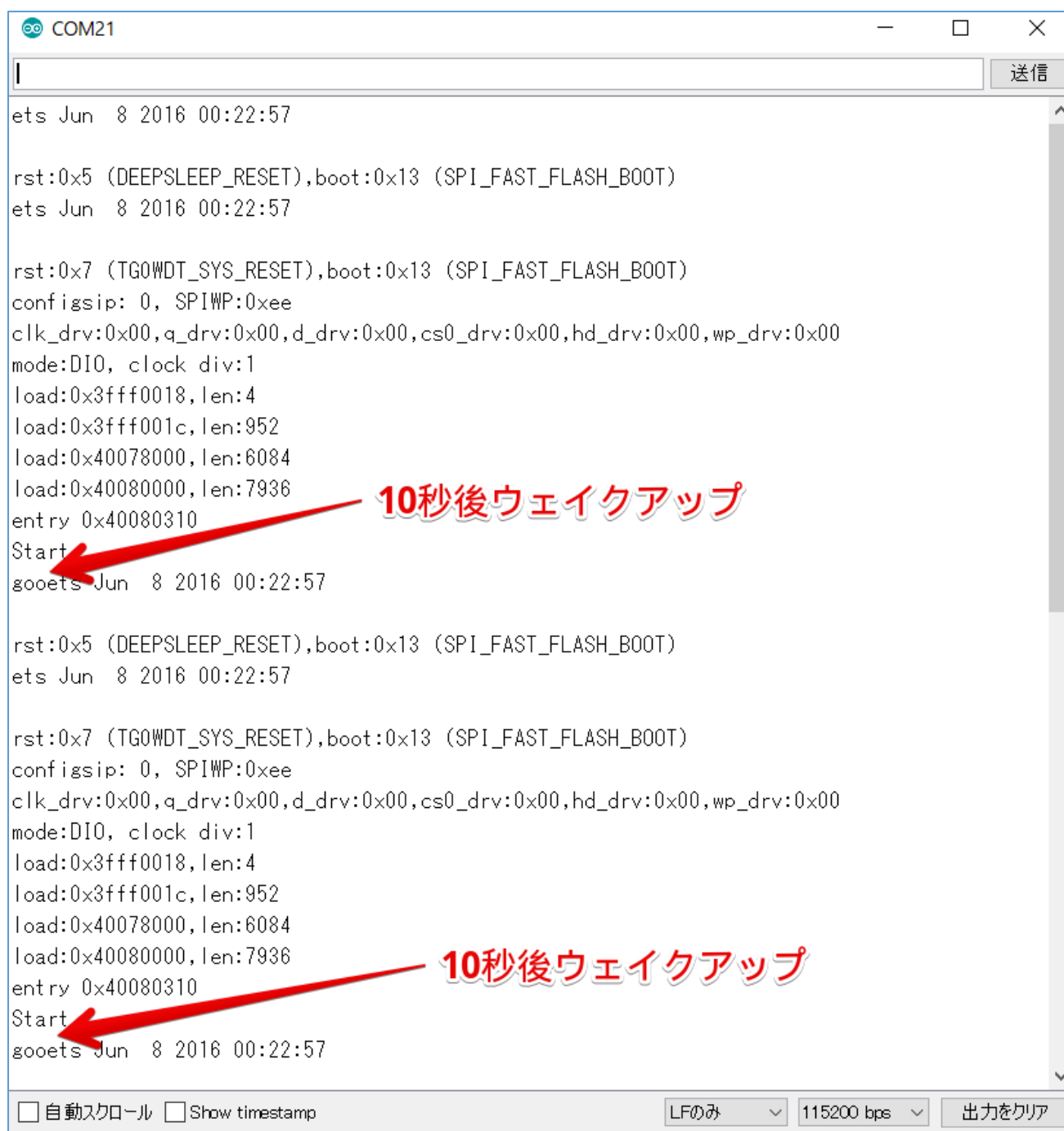


```
ets Jun 8 2016 00:22:57
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun 8 2016 00:22:57
rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
config: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:952
load:0x40078000,len:6084
load:0x40080000,len:7936
entry 0x40080310
Start
goo
```

シリアルモニタに“good night!”と表示しようとする最中に `esp_deep_sleep_start()` が実行され、ディープスリープ状態に突入する。

※ファームウェアのバージョンによって“good night!”ときちんと表示されるようになった。

10 秒待つと、以下の様な表示になる。



```
COM21
| 送信
ets Jun  8 2016 00:22:57

rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun  8 2016 00:22:57

rst:0x7 (TGWDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:952
load:0x40078000,len:6084
load:0x40080000,len:7936
entry 0x40080310
Start
gooets Jun  8 2016 00:22:57

rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun  8 2016 00:22:57

rst:0x7 (TGWDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:952
load:0x40078000,len:6084
load:0x40080000,len:7936
entry 0x40080310
Start
gooets Jun  8 2016 00:22:57

☐ 自動スクロール ☐ Show timestamp
LFのみ 115200 bps 出力をクリア
```

つまり、ディープスリープ状態はいったん「シャットダウン」のような状態になり、スリープが解除されウェイクアップする際には「再起動」するような状態でプログラムが最初から立ち上がる。

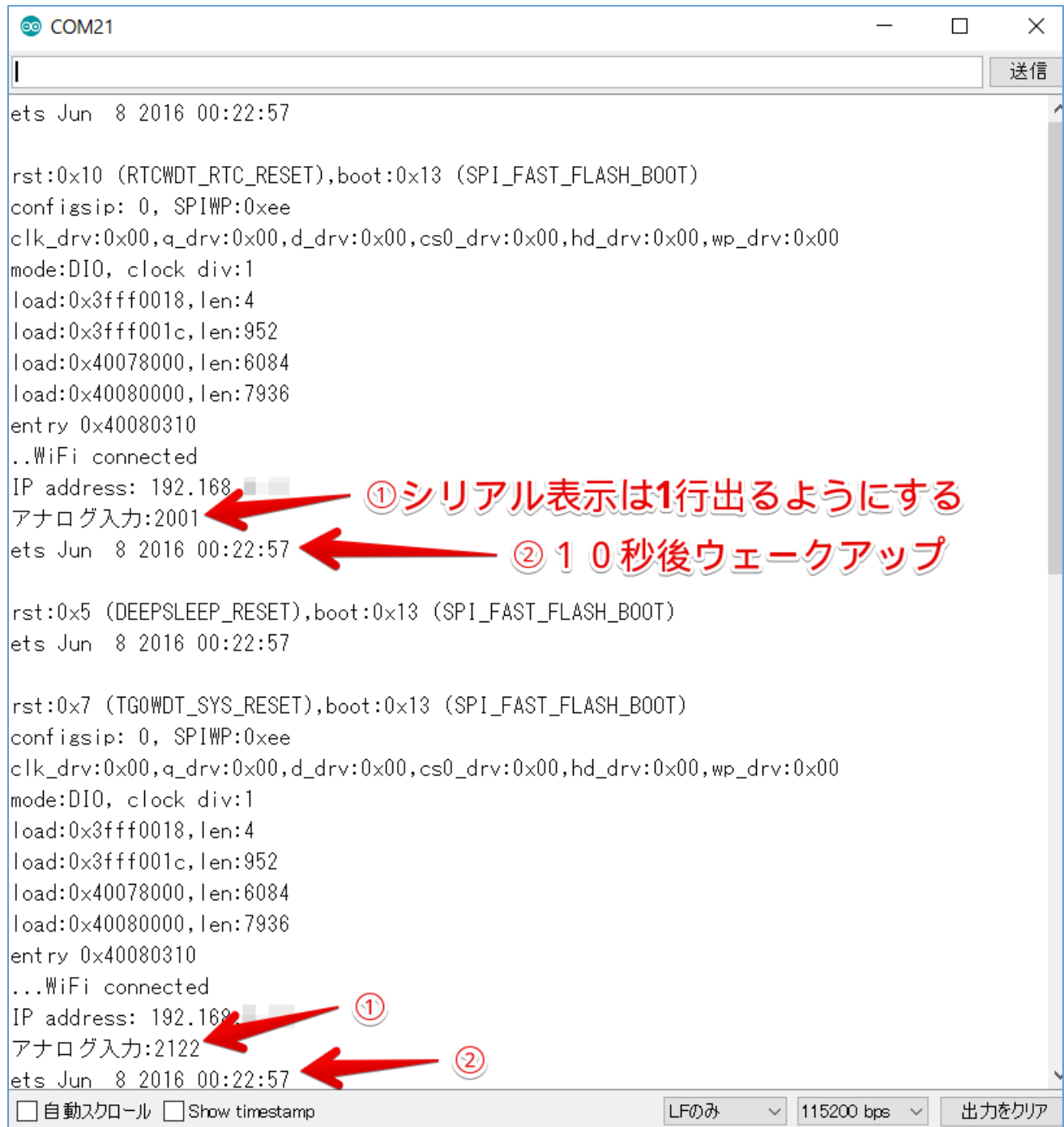
実際には、シャットダウンのような電源断にはなっておらず、ESP32 内部のタイマーを最小の電力で保持しつつ、次の起動に備えている状態となる。

【課題③】 プロジェクト名「kad32_cdsDeepSleepAmbient」

課題①とサンプル①のソースコードを組み合わせ、10秒おきにディープスリープから復帰して Ambient に光センサーのデータを send するスケッチを作成しなさい。

これにより課題①よりも省電力性の高いプログラムが出来上がる。

【実行結果 Arduino 側】



```
ets Jun 8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:952
load:0x40078000,len:6084
load:0x40080000,len:7936
entry 0x40080310
..WiFi connected
IP address: 192.168.
アナログ入力:2001
ets Jun 8 2016 00:22:57

rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun 8 2016 00:22:57

rst:0x7 (TGWDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:952
load:0x40078000,len:6084
load:0x40080000,len:7936
entry 0x40080310
...WiFi connected
IP address: 192.168.
アナログ入力:2122
ets Jun 8 2016 00:22:57
```

① シリアル表示は1行出るようにする

② 10秒後ウェークアップ

①

②

☐ 自動スクロール ☐ Show timestamp LFのみ 115200 bps 出力をクリア

光センサーのアナログ入力値が1行正常に表示されるようにすること。

【実行結果 Ambient 側】

光センサーのアナログ入力値が 0 から 4095 の間でプロットされていればよい。

