ESP32 マイコン用 MicroPython プログラム

Wi-Fi 搭載 ESP32 マイコンは，IoT システムの試作や実証実験，小規模な IoT システムの構築などに活用されている，IoT レディ・マイコンです．本書内でも IoT Sensor Core のマイコン部に使用しています．

ESP32 マイコンは，MicroPython 用のデバイスとしても注目されているので，使い方について紹介します．

写真 1　Raspberry Pi と ESP マイコン開発ボードの接続例
Raspberry Pi の USB 端子へ ESP32 マイコン開発ボードを接続し，MicroPython のファームウェアを書き込む

ESP32 マイコン用 MicroPython のインストール方法

Espressif Systems 製 ESP32-WROOM-32 モジュールのファームウェアを書き換えることで MicroPython を動かすことが出来ます．ESP32 マイコン用ファームウェアは，ビルドしたものが公式サイト（http://micropython.org/）より配布されています．筆者が準備したスクリプトを図 1 のように，「git clone https://bokunimo.net/git/iot⏎」でインストールし，download.sh を実行することで，Raspberry Pi の USB 端子（ttyUSB0）に接続した ESP32 マイコンへ書き込むことも出来ます．

```
pi@raspberrypi: $ git clone https://bokunimo.net/git/iot ⏎
                                        ～～省略～～          （未ダウンロード時のみ）
pi@raspberrypi: $ cd ~/iot/micropython/esp32/ ⏎
pi@raspberrypi:~/iot/micropython/esp32 $ ./download.sh ⏎
ESP32 へ MicroPython を書き込みます (usage: ./download.sh port)
                                        ～～省略～～
シリアル・ポート /dev/ttyUSB0 を使用します.
esptool.py をダウンロードします
                                        ～～省略～～
esp32-20190529-v1.11.bin をダウンロードします
                                        ～～省略～～
ESP32 を初期化します.
esptool.py v2.8-dev
Serial port /dev/ttyUSB0
Connecting.......
Chip is ESP32D0WDQ6 (revision 0)   （ESP32 マイコンからの応答）
                                        ～～省略～～
ESP32 へ書き込みます
                                        ～～省略～～
Wrote 1146864 bytes (717504 compressed) at 0x00001000 in 19.7 seconds (effective 465.7
kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
ESP32 への書き込みが完了しました.
ライセンス esptool_LICENSE.txt と micropython_LICENSE.txt を確認してから使用してください.
Done
pi@raspberrypi:~/iot/micropython/esp32 $
```

図 1　MicroPython のファームウェアを書き込む様子
GitHub から筆者が作成した iot フォルダを Raspberry Pi へダウンロードし，iot フォルダ→micropython→esp32 に収録した download.sh を実行した

ただし，MicroPython 用のファームウェアは Espressif Systems 製では無いので，ファームウェアのビルド方法によっては国内の電波法に基づいた工事設計認証に含まれない場合があります．無線に関わるソフトウェアは Espressif Systems のものを使用しているので，技術的な問題は無いと思いますが，筆者は念のため「コラム：ESP32-WROOM-32 モジュールのアンテナの取り外し方」に記載の方法で実験を行いました．

Thonny Python IDE でプログラムを開く

Thonny Python IDE は，Raspberry Pi に予めインストールされている Python 用のソフトウェア統合開発環境です．Thonny Python IDE を起動し，git clone でインストールしたサンプル・プログラム（iot フォルダ→micropython→esp32）を，Thonny Python IDE の画面上の左から 2 番目のアイコンで開いてください．

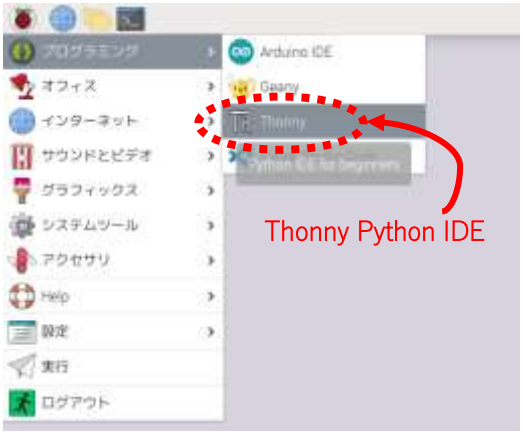

図 2 Tonny Python IDE を開く
Raspberry Pi に予めインストールされている Python 用ソフトウェア統合開発環境 Thonny Python IDE を起動する

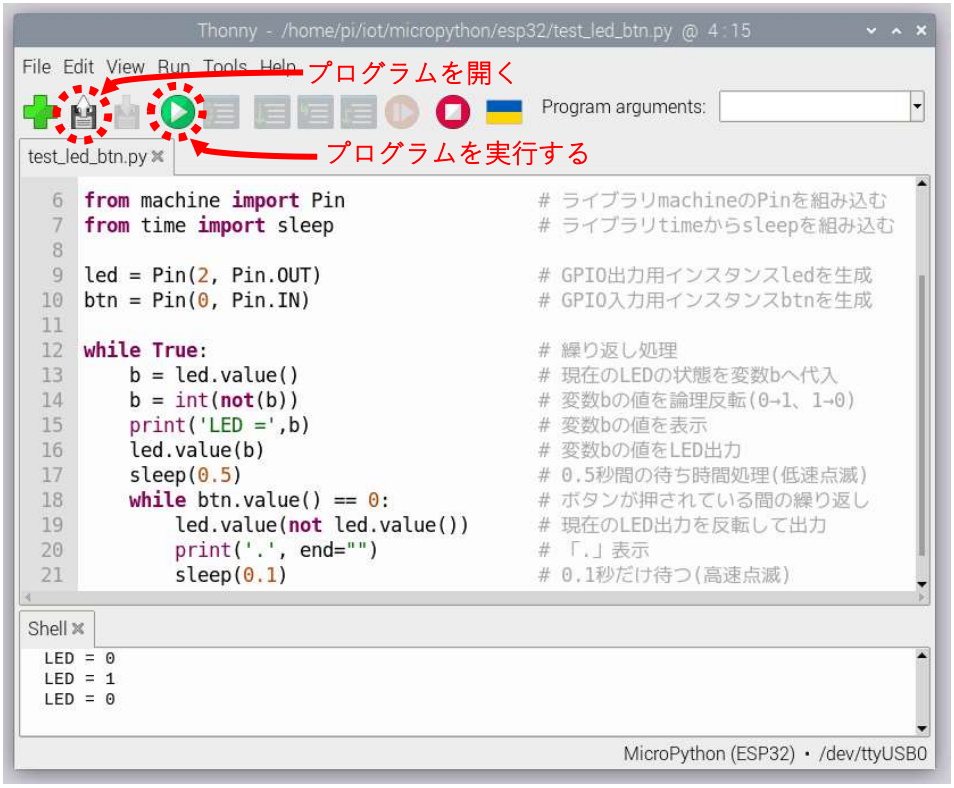

図 3 Tonny Python IDE でプログラムを開く
画面左上から 2 番目のプログラムを開くアイコンから iot フォルダ→micropython→esp32 に収録したサンプル・プログラムを開く

Thonny Python IDE からプログラムを実行する

プログラムを ESP32 上で実行できるようにするには，画面右下のコーター部をクリックし，マイコンとシリアル・ポートを選択します．選択後，図 3 の右向き三角のアイコン「プログラムを実行する」をクリックすると，表示中のプログラムが ESP32 に転送され，自動的に実行されます．

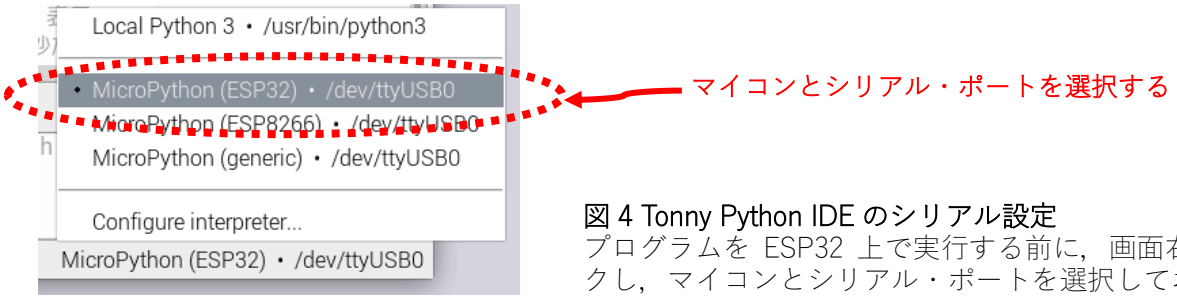

図 4 Tonny Python IDE のシリアル設定
プログラムを ESP32 上で実行する前に，画面右下をクリックし，マイコンとシリアル・ポートを選択しておく

ターミナルと REPL モードでプログラムを転送する方法もある

他にも REPL モードを使って，ESP32 上でプログラムを実行する方法もあります．Raspberry Pi の場合は LXTerminal 上で動作する cu コマンドを，Windows の場合は TeraTerm などのシリアル通信ターミナルを使い，シリアル通信速度 115200bps で ESP32 マイコンに接続してください．

REPL モードでプログラムを転送するには，図 5 のように，キーボードから[Ctrl]＋[E]を入力した後に，プログラムをクリップ・ボードからペーストします．ここでは，動作確認用のサンプル・プログラム test_led_btn.py（iot/micropython/esp32）をテキスト・エディタで開き，クリップ・ボードへコピーしてからシリアル通信ターミナルにペーストしてください．

プログラムを実行するには，転送後，キーボードから[Ctrl]＋[D]を入力します．

ESP32 マイコン用 MicroPython で L チカ test_led_btn.py

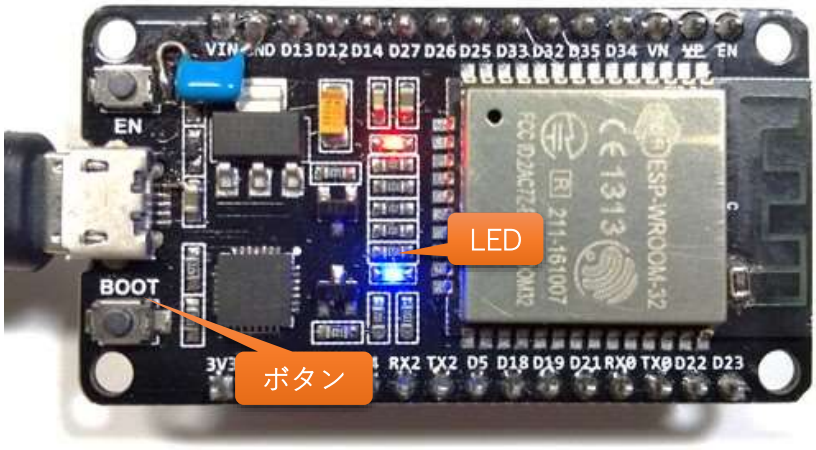

写真 2　ESP32 マイコン用 MicroPython で L チカ
LED を点滅させる L チカのサンプル・プログラム test_led_btn.py を DOIT 製 ESP32 開発ボード Dev Kit V1 上で実行したときの様子

test_led_btn.py は，「LED=1」と「LED=2」を 0.5 秒毎に繰り返し表示しつつ，ESP32 マイコンの GPIO ポート 2 の出力の High レベルと Low レベルを変化させます．GPIO ポート 2 に LED が接続されている開発ボードでは，LED が点滅します．また， BOOT ボタンが GPIO ポート 0 に接続されている ESP32 マイコン開発ボードでは，BOOT ボタン押下中，シリアル・ターミナルに「.」が表示され，LED が高速点滅します．

以下に，図 5 内のプログラム test_led_btn.py の GPIO 制御方法について説明します．NUCLEO-F767ZI でも同じように使用することが出来ます（ただし，STM32 マイコンでは GPIO ポート名を文字列で入力する）．

① ライブラリ machine から GPIO 制御用のモジュール Pin を組み込みます．

② GPIO ポート 2 用の変数（オブジェクト）led を生成します．引数はポート番号と，出力設定です．

③ ポート番号 0 の GPIO 入力用の変数 btn を生成します．第 3 引数に，プルアップ（Pin.PULL_UP），プルダウン（Pin.PULL_DOWN），なし（Pin.None）を指定することが出来ます．

④ LED を 1 秒間隔で点滅させるための繰り返し処理です．0.5 秒ごとにデジタル出力値を反転します．

⑤ 変数 led の GPIO ポート 2 に対し，変数 b の値が 1 のときに High レベルを，0 のときに Low を出力します．0.5 秒ごとに b を反転し，繰り返し実行することで，LED の点滅制御を行います．

⑥ GPIO ポート 0 の入力ボタン状態を取得し，Low レベルを示す 0 の時に処理⑦を実行します．

⑦ 0.1 秒ごとの LED の点滅制御部です．value 命令を 2 回，使用し，1 行の中で GPIO 出力の反転処理を行います．括弧内の引数の無い value 命令は，現在の GPIO 出力状態の値を得ます．取得値を not で論理を反転し，もう一方の value 命令で出力することで，GPIO の出力状態を反転させることが出来ます．

```
pi@raspberrypi:~ $ sudo apt-get install cu          ←(cu 未インストール時)
pi@raspberrypi:~ $ cd ~/iot/micropython/esp32
pi@raspberrypi:~/iot/micropython/esp32 $ stty -F /dev/ttyUSB0 sane 115200
pi@raspberrypi:~/iot/micropython/esp32 $ cu -s 115200 -l /dev/ttyUSB0
Connected.
(キーボードから[Enter]を入力)
>>>
(キーボードから[Ctrl]+[E]を入力)
paste mode; Ctrl-C to cancel, Ctrl-D to finish
(サンプル・プログラム test_led_btn.py をペースト)
=== # coding: utf-8
=== # ESP32 マイコンの動作確認 L チカ＋ボタン操作
=== # 0.5 秒おきに LED の点灯と消灯を反転し，ボタンが押されている間は高速点滅する
=== # Copyright (c) 2019 Wataru KUNINO
===
=== from machine import Pin                ←①   # ライブラリ machine の Pin を組み込む
=== from time import sleep                       # ライブラリ time から sleep を組み込む
===
=== led = Pin(2, Pin.OUT)                  ←②   # GPIO 出力用インスタンス led を生成
=== btn = Pin(0, Pin.IN , Pin.PULL_UP)     ←③   # GPIO 入力用インスタンス btn を生成
===
=== while True:                            ←④   # 繰り返し処理
===     b = led.value()                          # 現在の LED の状態を変数 b へ代入
===     b = int(not(b))                          # 変数 b の値を論理反転(0→1, 1→0)
===     print('LED =',b)                         # 変数 b の値を表示
===     led.value(b)                       ←⑤   # 変数 b の値を LED 出力
===     sleep(0.5)                               # 0.5 秒間の待ち時間処理(低速点滅)
===     while btn.value() == 0:            ←⑥   # ボタンが押されている間の繰り返し
===         led.value(not led.value())           # 現在の LED 出力を反転して出力
===         print('.', end="")             ←⑦   # 「.」表示
===         sleep(0.1)                           # 0.1 秒だけ待つ(高速点滅)
(キーボードから[Ctrl]+[D]を入力)
LED = 1
LED = 0
LED = 1
LED = 0
(ESP32 マイコン開発ボード上の BOOT ボタン押下)
..................................... LED = 1
LED = 0
(キーボードからチルダ[~]とピリオド[.]を入力)
Disconnected.
pi@raspberrypi:~/iot/micropython/esp32 $
```

図 5 ESP32 マイコン用 LED とボタン制御用サンプル・プログラム test_led_btn.py の実行例
ESP32 マイコンを USB へ接続し，シリアル通信ターミナル cu で MicroPython へプログラムを転送し，実行した

ESP32 用 MicroPython でインターネット HTTP 通信実験 test_htget_usocket.py

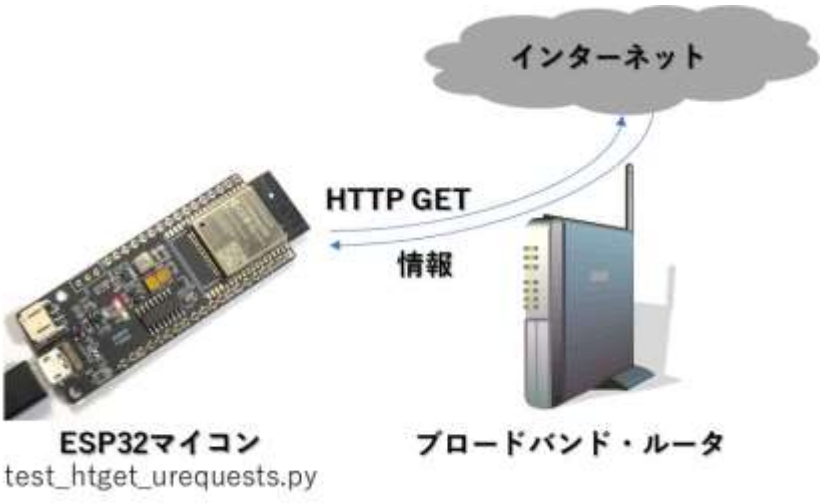

図 6 ESP32 マイコンからインターネットへ HTTP 通信実験
Wi-Fi 搭載 ESP32 マイコンを使用し，HTTP GET でインターネット上の情報を取得する実験を行う

ESP32 マイコン用の MicroPython は，HTTP 通信を行う際に便利なライブラリ μRequests に対応しているので，IoT 機器のプログラミングが簡単に行えます．

そこで，NUCLEO-F767ZI 用サンプル・プログラム tcp_htget_u.py を ESP32 用に移植した実験用サンプル・プログラム test_htget_usocket.py と，ライブラリ μRequests を使った ESP32 用 test_htget_urequests.py を作成しました（どちらも iot/micropython/esp32 に収録）．比較してみると，μRequests を使用することで，行数が半分程度になり，プログラム固有の処理が主体となっていることが分かるでしょう．同じ機能であれば，行

数が少ない方が，特徴部分の開発に専念することが出来，またソフトウェアの信頼性を高めることも出来ます．

ライブラリ μRequests を使ったリスト１の実験用サンプル・プログラム test_htget_urequests.py の主な処理について，以下に説明します．

① Ethernet 通信や Wi-Fi 通信に必要なライブラリ network を組み込みます．

② HTTP 通信用のライブラリ μRequests （urequests）を組み込みます．

③ 本機を Wi-Fi 接続するアクセスポイントの SSID とパスワードを各変数に代入します．お手持ちの Wi-Fi アクセスポイント本体などに書かれた SSID とパスワードに書き換えてから，プログラムを ESP32 マイコンに転送してください．

④ Wi-Fi 接続用の変数（オブジェクト）を生成します．

⑤ 本機の Wi-Fi を起動する active 命令です．

⑥ Wi-Fi アクセスポイントを検索し，SSID の一覧を表示します．本コマンドが無くても，接続できます．

⑦ 命令 connect を使って，処理③で設定した SSID とパスワードの Wi-Fi アクセスポイントへの接続指示を行います．

⑧ 関数 isconnected を使って，Wi-Fi 接続状態を確認し，接続が完了するまで「.」を繰り返し表示します．

⑨ HTTP GET リクエストを送信し，応答を変数 res へ代入します．

⑩ ライブラリ μRequests に含まれる json 命令を使って，応答結果を辞書型変数へ代入します．

```
import network                                  # ネットワーク通信ライブラリ
import urequests                                # HTTP 通信ライブラリ
from sys import exit                            # ライブラリ sys から exit を組み込む
from machine import Pin                         # ライブラリ machine の Pin を組み込む
from time import sleep                          # ライブラリ time から sleep を組み込む

wifi_ssid = '<AP_name>'                         # Wi-Fi アクセスポイントの SSID を記入
wifi_pass = '<password>'                        # パスワードを記入

url = 'http://bokunimo.net/iot/cq/test.json'# アクセス先の URL

led = Pin(2, Pin.OUT)                           # GPIO 出力用インスタンス led を生成
led.value(1)                                    # LED を点灯
sta_if = network.WLAN(network.STA_IF)           # Wi-Fi 接続用インスタンスの生成
sta_if.active(True)                             # Wi-Fi の起動
sta_if.scan()                                   # Scan for available access points
sta_if.connect(wifi_ssid, wifi_pass)            # Connect to an AP

while not sta_if.isconnected():                 # Check for successful connection
    print('.', end='')
    led.value(not led.value())
    sleep(0.5)                                  # 0.5 秒間の待ち時間処理(低速点滅)

try:                                            # 例外処理の監視を開始
    res = urequests.get(url)                    # HTTP リクエストを送信し，受信する
except Exception as e:                          # 例外処理発生時
    print(e)                                    # エラー内容を表示
    sta_if.disconnect()                         # Wi-Fi の切断
    sta_if.active(False)                        # Wi-Fi の停止
    exit()

res_dict = res.json()                           # 受信データを変数 res_dict へ代入
print('-----------------------------------') # ----------------------------
print('title :', res_dict.get('title'))         # 項目'title'の内容を取得・表示
print('descr :', res_dict.get('descr'))         # 項目'descr'の内容を取得・表示
print('state :', res_dict.get('state'))         # 項目'state'の内容を取得・表示
print('url   :', res_dict.get('url'))           # 項目'url'内容を取得・表示
print('date  :', res_dict.get('date'))          # 項目'date'内容を取得・表示

sta_if.disconnect()                             # Wi-Fi の切断
sta_if.active(False)                            # Wi-Fi の停止
led.value(0)                                    # LED を消灯
```

リスト１　ESP32 マイコンからインターネット上の情報を取得する実験用サンプル・プログラム test_htget_urequests.py

ESP32 マイコン用 MicroPython で使用可能な ureqests ライブラリにより，簡潔になった

ESP32 用 MicroPython でクラウド・サービス Ambient 対応 IoT 温度センサの実験 test_htpost_ambient.py

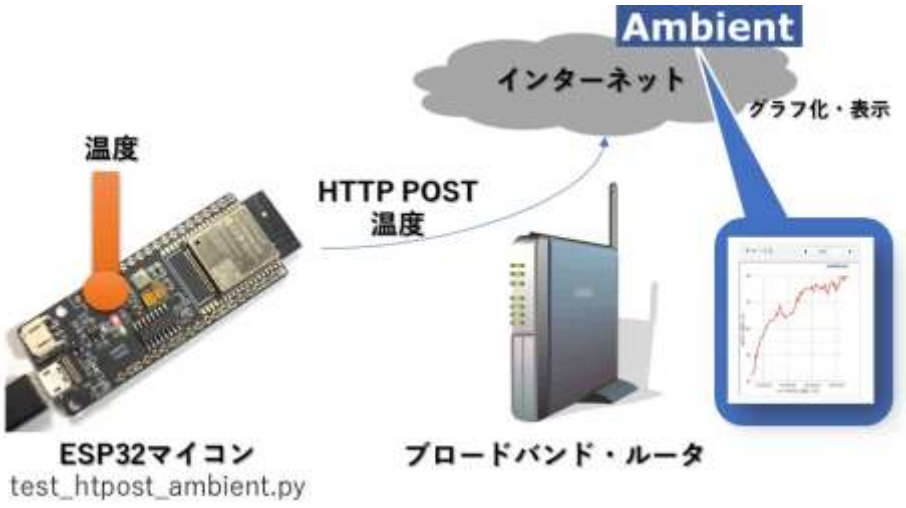

図 7 ESP32 用 MicroPython でクラウド・サービス Ambient 対応 IoT 温度センサの実験
Wi-Fi 搭載 ESP32 マイコンで測定した温度値をクラウドへ HTTP POST 送信する実験を行う

ESP32 用 MicroPython であれば，クラウド・サービスとの連携も簡単です．リスト 2 の Ambient への実験用サンプル・プログラム test_htpost_ambient.py では，主に下記の①～③の処理部で，HTTP POST 通信を行います．

① Ambient 用チャネル ID を文字列変数 ambient_chid に代入します．予め Ambient のウェブサイト（https://ambidata.io/）で取得したチャネル ID を，ここに入力してください．

② Ambient のウェブサイトで取得したライトキーを，文字列変数 ambient_wkey に代入します．

③ HTTP POST で送信を行うためのアクセス先，ヘッダ，コンテンツ情報を変数 url，head，body へ代入します．

④ ESP32 マイコン内蔵の温度センサから温度値を取得します．マイコンの内部発熱などにより室温よりも高い値が得られるので，予め補正値を変数 temp_offset へ代入しておき，減算します．

⑤ ライブラリ μRequests に含まれる post コマンドを使って HTTP POST による送信を実行します．

ESP32 用 IoT センサの仕上げ，ケチケチ運転技術ディープ・スリープ

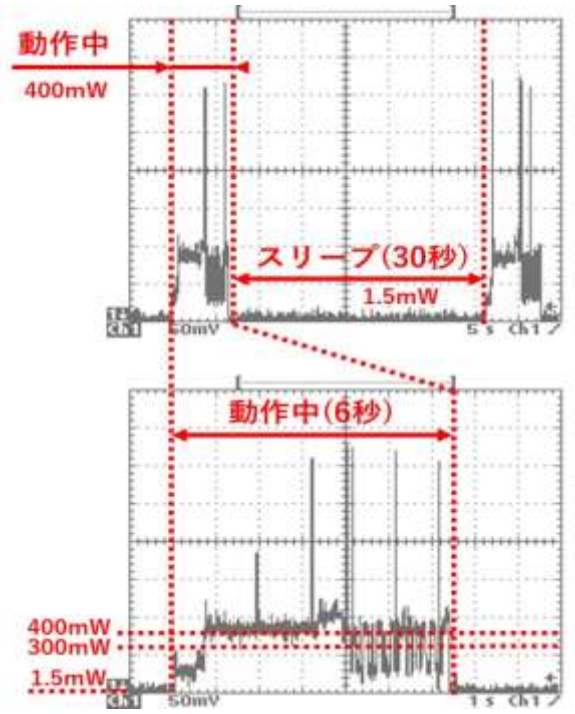

図 8 ケチケチ運転術で動作する ESP32 マイコンの消費電力測定結果例
ディープ・スリープを使用したケチケチ運転術による ESP32 マイコンの消費電力の一例．
安定化電源 5V を村田製作所製 DCDC コンバータ LXDC55 へ入力し，出力 3.3V を ESP32-WROOM-32 へ入力した場合の平均消費電力は約 70mW だった

乾電池などによる長期間動作が可能な IoT センサの実現性について，確認してみましょう．ESP32 マイコン用 MicroPython はディープ・スリープに対応しており，待機と動作を繰り返すことで平均の消費電力を下げることが出来ます．ディープ・スリープを利用するには，リスト 2 の実験用サンプル・プログラム test_htpost_ambient.py の末尾行（⑥）の deepsleep 命令を有効にし，ファイル名を main.py に変更後，ESP32 マイコンへ書き込みます．書き込み方法は「appendix： ESP32 マイコンへのプログラム転送方法」を参照してください．

図 8 は，村田製作所製 DCDC コンバータ LXDC55 を電源回路に用いたときの消費電力の測定結果例です．平均消費電力はディープ・スリープ時間を 30 秒にしたときが約 70mW，より長い 23 分にしたときが約 5mW でした．消費電力 5mW は，単 3 アルカリ乾電池 3 本で約 2.5 か月の動作期間に相当します．本実験により，MicroPython を使った実用的なワイヤレス IoT センサの実現性について確認することが出来ました．

```
wifi_ssid = '<AP_name>'                    # Wi-Fi アクセスポイントの SSID を記入
wifi_pass = '<password>'                   # パスワードを記入
ambient_chid='0000'                        # Ambient で取得したチャネル ID を入力
ambient_wkey='0123456789abcdef'            # ここにはライトキーを入力
amdient_tag='d1'                           # データ番号 d1～d8 のいずれかを入力
temp_offset = 30.0                         # CPU 温度上昇値(要調整)

import network                             # ネットワーク通信ライブラリ
import urequests                           # HTTP 通信ライブラリ
from sys import exit                       # ライブラリ sys から exit を組み込む
from machine import Pin, deepsleep         # GPIO 用 Pin とディープスリープを組込
from time import sleep                     # ライブラリ time から sleep を組み込む
from esp32 import raw_temperature

url = 'http://ambidata.io/api/v2/channels/'+ambient_chid+'/data' # アクセス先
head = {'Content-Type':'application/json'}  # ヘッダを変数 head_dict へ
body = {'writeKey':ambient_wkey, amdient_tag:0.0}  # 内容を変数 body へ

led = Pin(2, Pin.OUT)                      # GPIO 出力用インスタンス led を生成
led.value(1)                               # LED を点灯
sta_if = network.WLAN(network.STA_IF)      # Wi-Fi 接続用インスタンスの生成
sta_if.active(True)                        # Wi-Fi の起動
sta_if.scan()                              # Scan for available access points
sta_if.connect(wifi_ssid, wifi_pass)       # Connect to an AP

while not sta_if.isconnected():            # Check for successful connection
    print('.', end='')
    led.value(not led.value())
    sleep(0.5)                             # 0.5 秒間の待ち時間処理(低速点滅)

# 温度を取得する
temp = (raw_temperature() - 32) * 5 / 9    # 温度を取得
temp = round(temp - temp_offset, 1)        # 温度補正と小数第二位以下の丸め処理
print('Temperature =',temp)                # 温度を表示する
body[amdient_tag] = temp                   # 辞書型変数 body 内に埋め込む

# Ambient へ送信する
try:                                       # 例外処理の監視を開始
    res = urequests.post(url, json=body, headers=head)  # HTTP リクエストを送信
    print('HTTP Status Code =', res.status_code)
except Exception as e:                     # 例外処理発生時
    print(e)                               # エラー内容を表示
    sta_if.disconnect()                    # Wi-Fi の切断
    sta_if.active(False)                   # Wi-Fi の停止
    exit()

sta_if.disconnect()                        # Wi-Fi の切断
sta_if.active(False)                       # Wi-Fi の停止
led.value(0)                               # LED を消灯
# deepsleep(30000)                         # 30 秒間スリープ
```

リスト 2 ESP32 マイコンから Ambient へ温度値を送信する実験用 IoT 温度センサ test_htpost_ambient.py
ESP32 マイコン内蔵の温度センサから得られた温度値を，HTTP POST で送信する実験用 MicroPython プログラム

appendix： ESP32 マイコンへのファイル転送方法

ESP32 にプログラムを転送する方法には，前述の Thonny Python IDE を使う方法や，WebSocket 通信用の WebREPL モードを使用する方法，米 Adafruit 社が開発した ampy を使用する方法などがあります．ESP32 マイコン開発ボード上の USB シリアル変換 IC には，BBC micro:bit や STM32 マイコン F767ZI のように，ESP32 マイコン内のファイルシステムを USB ストレージとして利用する機能が搭載されていないので，MicroPython の REPL モードを使ってプログラムを転送します．

WebREPL モードと，ampy の使い方について，以下で説明します．

（プログラム転送方法① WebREPL モード）

ESP32 マイコン用 MicroPython には，（通常の）シリアル通信用の REPL モードに加え，WebSocket 通信用の WebREPL モードがあります．WebREPL モードを有効にするには，通常の REPL モードから図 9 のように，WebREPL 用のパスワードを設定し，Wi-Fi アクセスポイントへ接続してください．接続に成功すると ESP32 マイコンの IP アドレスが表示されます．

同じ LAN 内の Raspberry Pi や PC のインターネット・ブラウザで「https://micropython.org/webrepl/」にアクセスすると，図 0-10 のような WebREPL クライアント・ソフトが起動します．

ESP マイコンの IP アドレスを入力し，[Connect]ボタンで接続し，画面右側の「Send a file」からプログラムを転送することが出来ます．通常の REPL モードと同じ操作も可能です．例えば，「import os」と「os.listdir()」を入力することで，転送したファイルを確認することも出来ます．

以上の通り，LAN 内からリモートで通常の REPL モードと同様の処理が行えるので，MicroPython の開発ツールとしても使用することが出来ます．とくに，ソフトウェアがブラウザ上で動作するので，Raspberry Pi や PC，スマートフォンなど様々な機器から利用できる点は，開発時だけでなく小規模なシステムでの運用時にも役立つでしょう．ただし，Wi-Fi 接続が必要となるので，電波法に基づいた認証または技適を受けた ESP32 マイコン用 MicroPython ファームウェアを使用する必要があります．

```
（通常のシリアル通信用 REPL モードから WebREPL モード設定する）
MicroPython v1.11 on 2019-05-29; ESP32 module with ESP32
Type "help()" for more information.
>>> import webrepl_setup
WebREPL daemon auto-start status: disabled

Would you like to (E)nable or (D)isable it running on boot?
(Empty line to quit)
> E
To enable WebREPL, you must set password for it
New password (4-9 chars): cqpub
Confirm password: cqpub
Changes will be activated after reboot
Would you like to reboot now? (y/n) y

Started webrepl in normal mode  ←(WebREPL が有効になった)
MicroPython v1.11 on 2019-05-29; ESP32 module with ESP32
Type "help()" for more information.
>>> import network
>>> sta = network.WLAN(network.STA_IF); sta.active(True)  ←(Wi-Fi の有効化)

>>> sta.connect(<SSID>, <PASS>)  ←(Wi-Fi アクセスポイントへ接続)
I (44048) network: CONNECTED
I (45768) event: sta ip: 192.168.0.9, mask: 255.255.255.0, gw: 192.168.0.1  ←(本機の IP アドレス)
I (45768) network: GOT_IP
```

図 9 MicroPython の WebREPL サーバを設定し，起動したときのようす
WebREPL を有効化し，パスワードを設定し，再起動後に Wi-Fi を起動すると ESP32 マイコン上で WebREPL サーバが起動する

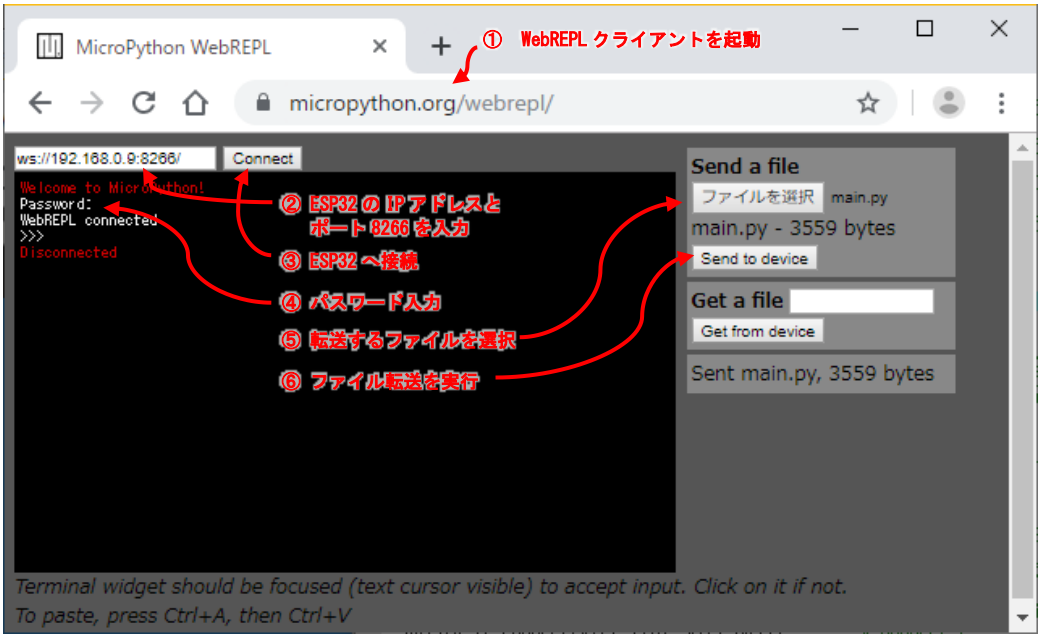

図 0-10 ESP32 マイコン用 WebREPL クライアント・ソフトの画面例
インターネットブラウザで「https://micropython.org/webrepl/」にアクセスすると，WebREPL へアクセス可能なクライアント・ソフトが起動する

（プログラム転送方法② Adafruit 製 ampy）

Adafruit 製 ampy は，Python で書かれた MicroPython 用ファイル転送ツールです．シリアル接続された ESP32 マイコンなどに対応しています．図 11 に Raspberry Pi へのインストール方法，ファイル main.py の転送方法，確認方法を示します．

コマンド「ampy」と「-p」に続く「/dev/ttyUSB0」はシリアル・ポートです．複数の USB シリアル機器を接続している場合は「ttyUSB1」や「ttyUSB2」などになる場合があります．また，Windows（Cygwin 等）の場合は「ttyS2」や「ttyS3」などになり，数字は COM ポート番号から 1 少ない数になります．

シリアル・ポートの後には ampy コマンドを入力します．「ls」はファイル表示命令，「put」はファイル転送命令です．「ampy --help」でコマンド一覧が表示されます．

```
（Adafruit 製 ampy のインストール）
pi@raspberrypi:~ $ sudo pip install adafruit-ampy
Successfully installed adafruit-ampy-1.0.7 python-dotenv-0.10.3 typing-3.7.4.1

（ファイル main.py を ESP32 へ転送する）
pi@raspberrypi:~ $ ampy -p /dev/ttyUSB0 ls ←（ファイル確認）
/boot.py
pi@raspberrypi:~ $ ampy -p /dev/ttyUSB0 put main.py ←（ファイル main.py の転送）
pi@raspberrypi:~ $ ampy -p /dev/ttyUSB0 ls ←（ファイル確認）
/boot.py
/main.py

（転送した main.py を実行する）
pi@raspberrypi:~ $ cu -s 115200 -l /dev/ttyUSB0 ←（REPL モードでアクセス）
Connected.

（キーボードから[Ctrl]+[D]を入力）
MPY: soft reboot
I (1676239) phy: phy_version: 4007, 9c6b43b, Jan 11 2019, 16:45:07, 0, 2
......Temperature = 33.3 ←（温度測定結果）
HTTP Status Code = 200 ←（HTTP 応答:200＝送信成功）
（起動時のログ表示）
......Temperature = 33.3
HTTP Status Code = 200
（キーボードからチルダ[~]とピリオド[.]を入力）
Disconnected.
pi@raspberrypi:~ $
```

図 11 Adafruit 製 ampy を Raspberry Pi へインストールし，ファイル main.py を ESP32 マイコンへ転送する
コマンド「ampy」とオプション「-p」に続けてシリアル・ポートを指定し，ampy コマンド put を使ってファイル転送を行う

コラム：ESP32-WROOM-32 モジュールのアンテナの取り外し方

Espressif Systems 製 ESP32-WROOM-32 モジュールは国内の電波法に基づいた工事設計認証を受けていますが，MicroPython 用のファームウェアは Espressif Systems 製ではありません．無線に関わる部分は Espressif Systems が作成したものを使っているので，技術的な問題は無いと思いますが，ファームウェアのビルド方法によっては認証に含まれない可能性が考えられます．

そこで，本書では無線設備の適用を回避するために，ESP32-WROOM-32 モジュールからアンテナを取り外し，終端抵抗器を取り付けてから，ESP32 マイコン用 MicroPython を試しました．図 12 のように，アンテナ・パターンを剥がし，50Ωまたは 51Ωのチップ抵抗を半田付け実装することで，アンテナへの給電を行わないようにすることが出来ます．

アンテナが無い状態であっても，同じシールド・ボックス※内に無線 LAN アクセスポイントを近づければ，ESP マイコン回路内の微弱な電流を検知することが出来，通信の実験も可能です．

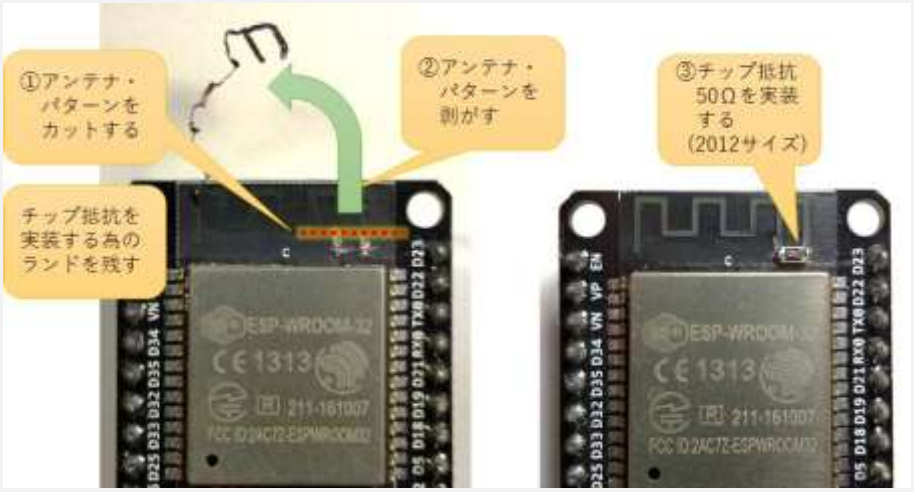

図 12 無線設備の適用を回避するために，アンテナを取り外した
アンテナ・パターンの根元をカットしてから剥がし，チップ抵抗（50Ωまたは 51Ω・2012 サイズ）を実装した

※無線設備には該当しないものの，実質的には微弱無線機相当の扱いと見做される場合も考えられます．送受信を行う場合はシールド・ボックス内で実験を行ってください．

著者について

国野　亘（くにの・わたる）　ボクにもわかる電子工作（https://bokunimo.net/）

関西生まれ．言葉の異なる関東や欧米などさまざまな地域で暮らすも，近年は住みよい関西圏に生息し続けている哺乳類・サル目・ヒト属・関西人．おもにホビー向けのワイヤレス応用システムの研究開発を行い，その成果を書籍やウェブサイトで公開している．

著書・ウェブサイト

2004 年 11 月　ボクにもわかる地上デジタル（https://bokunimo.net/tdtv/）

2009 年　5 月　地デジ TV 用プリアンプの実験（CQ 出版株式会社）

2014 年　5 月　ZigBee/Wi-Fi/Bluetooth 無線用 Arduino プログラム全集（CQ 出版株式会社）

2014 年 12 月　ボクにもわかる衛星デジタル放送の受信方法（https://bokunimo.net/bstv/）

2016 年　3 月　1 行リターンですぐ動く! BASIC I/O コンピュータ IchigoJam 入門（CQ 出版株式会社）

2017 年　2 月　Wi-Fi/Bluetooth/ZigBee 無線用 ラズベリー・パイプログラム全集（CQ 出版株式会社）

2018 年 10 月　ボクにもわかる電子工作（https://bokunimo.net/）

2019 年　2 月　超特急 Web 接続！ESP マイコン・プログラム全集（CQ 出版株式会社）

2021 年　2 月　ボクにもわかる IchigoJam BASIC で作る IoT システム（https://bokunimo.net/ichigojam/iot/）

Python について

Python は Python Software Foundation の著作物です．同団体の PSF ライセンスにしたがって使用することが出来ます（https://docs.python.org/ja/3/license.html）.

参考文献

- MicroPython documentation
  - Damien P. George 他（https://micropython-docs.readthedocs.io/）
- ESP32
  - ESPRESSIF SYSTEMS (https://www.espressif.com/en/products/socs/esp32)

履歴

2019 年　9 月　1 日	GitHub リリース（https://git.bokunimo.com/iot/）
2021 年 11 月 13 日	PDF リリース Ver. 1.0
2023 年　4 月　12 日	PDF 改定（Thonny Python IDE 対応）Ver. 1.1

権利情報