ボード・コンピュータ・シリーズ

MICRO CONTROLLER マイコン

定番Wi-Fi/Bluetoothマイコンの
ハードウェアを拡張して外付けパーツを制御する

# カメラ/センサ/測定器

# ESP&M5Stack 電子工作プログラム集

Interface編集部 編



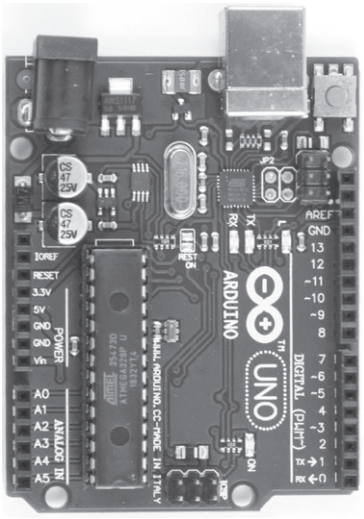

CQ出版社

第1章

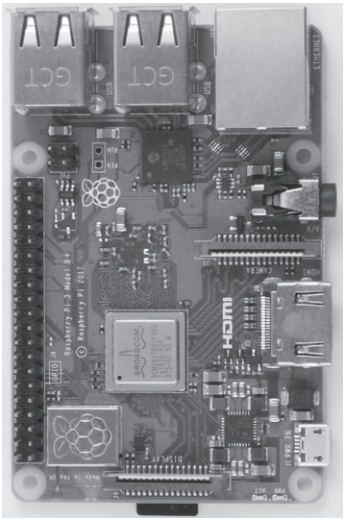
700円からWi-Fi付きで本格的

# IoTマイコン ESP32の世界

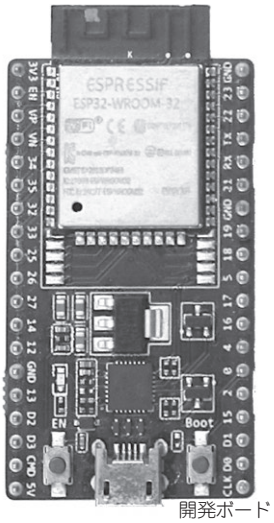
宮田 賢一




（a）定番マイコン「Arduino UNO」




（b）定番コンピュータ「ラズベリー・パイ（Raspberry Pi）」




開発ボード

（c）第3の定番マイコン「ESP32」

写真1　ちょうどいい第3の定番マイコン・ボードESP32を研究する

## 機能も価格もちょうど良い 第3の定番「ESP32」

小型で汎用的なマイコン&コンピュータ・ボードの代表といえば，Arduinoやラズベリー・パイです（**写真1**）．

これらの魅力は，

- 小型である
- I²CやSPIなど標準的なマイコン機能が使える
- オープンソースでソフトウェアが提供されている
- 入手しやすい価格帯である

ということです．この特徴から，一般の人にも浸透しました．

### ● 現在進行形で育ち中

この分野に，中国の上海に拠点を置くEspressif Systems（Shanghai）社が2014年にESP8266，2016年にESP32を開発しました（**写真2**）．特にESP32については，

- 700円程度で入手可能
- 価格の割には高性能なCPU（240MHz，デュアルコア）を搭載
- Wi-FiとBluetoothが同時に使える（しかも日本の技適認証を取得済み）
- 非常に多くのソフトウェア・ライブラリがSoCやボード・メーカによって公開されている

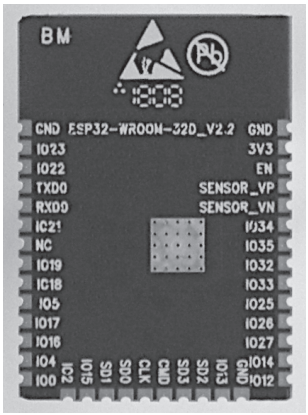
といった特徴を持っていたことから，今やマイコン&コンピュータ・ボード界の第3の柱として，多くのユーザが開発に取り組んでいます．

その流れに乗って，ESP32搭載のバラエティに富んだマイコン・ボードが開発・販売されており，有志の開発者によってさまざまな開発環境の整備も現在進行形で進んでいます．

### ● ESP32が入った液晶付きキットM5Stackも登場

M5StackはESP32をシステムの中心に[illegible]，[illegible]


（a）表面




（b）裏面

写真2　小型でWi-Fi付きのESP32モジュールが700円程度で入手できる


写真3　ESP32が入ったカラー液晶付きキット
M5Stackは便利で持ち運びやすい

カラー・ディスプレイ，スピーカ，バッテリ，microSDカード・スロットを1つのパッケージに収めています．拡張モジュールを積み重ねられるようになっていて，好きな機能を追加することもできます．2018年に登場すると注目を集めました（**写真3**）．

本書は，国内で入手できるESP32搭載のマイコン・ボード，ソフトウェア・ライブラリ，プログラミング言語，開発環境に関する情報をできるだけ収集し，まとめています．

## 特徴

### ● その1：Wi-FiやBluetoothといった無線標準装備

ESP32モジュール内にWi-FiとBluetoothの両方のハードウェアを備えています．これらを備えた手頃な価格のマイコンは意外と少なく，ESP32が開発者の心を掴んだ最大の理由とも言えます．

センサ・データをインターネットのクラウド・サーバに送信したり，Bluetoothスピーカにしてスマホの音楽を再生したり，赤外線で家電をコントロールしたりといったIoTやスマート・ホーム向けアプリケーションの開発を，デバイスを後付けすることなくできます．

その他，ESP32の特徴をあげてみます．

### ● その2：今どきのマイコン・ボードとしてふつうのことはできる

ESP32は$I^2C$やSPI，PWM，UARTなど，標準的なインターフェースを一通り備えています．それらを使うためのライブラリも公開されているので，好きなデバイスを接続してすぐに使えます．

ESP32はArduinoに対応しています．多数公開されているArduinoライブラリをそのまま使って開発ができます．

### ● その3：IoTも意識している①…低消費電力モード対応

ESP32は低消費電力モードの設定が可能です．さらに低消費電力モード時に動作する低クロック周波数のコプロセッサULP（Ultra Low Power）も内蔵していて，メイン・コアが停止していてもタイマや外部割り込みで自ら起動して専用メモリ内のプログラムを実行できます．長期間にわたって電池交換なしでセンサ・データを取得するようなアプリケーションへの応用が可能です．

### ● その4：IoTも意識している②…リアルタイムOS対応

ESP32の内部では本格的なリアルタイムOSであるFreeRTOSが動作しており，マルチタスク・プログラムの開発もライブラリ利用により比較的簡単に行えます．例えば電光掲示板にお知らせをずっと流しながら，ESP32をウェブ・サーバにしてお知らせ情報をインターネットからアップロードするという，シンプルなマイコンでは難しいプログラミングが可能です．

皆さんもESP32を使って，オリジナリティあふれる作品を作ってみませんか？

## ESP32は「使える」マイコン

Wi-FiもBluetoothも使えて700円程度で入手可能なESP32は，バラエティ豊かな作例や，数々のディープな解析記事を見かけるようになりました．そのためマイコンが好きな方であればESP32の名前は聞い

ことがあるかもしれません．

ESP32が「使える」マイコンであることを紹介します．

### ● 本格的なマイコン開発もOK

次のような場合にESP32はお勧めです．

- マイコン開発の経験はあるけど，話題性のあるマイコンは技術者として押さえておきたい．
- 動作を正しく理解するために公式のリファレンス・ボードが欲しい．
- 標準的なデバッグ手法はほしい

ESP32はとかく安さが強調されがちですが，実は本格的に「使える」マイコンです．

CoreMark値700[注1]を達成するデュアルコア・プロセッサの上で，FreeRTOSが動作します．FreeRTOSはAmazon傘下で開発が行われているリアルタイムOSで，STマイクロエレクトロニクスやNXPセミコンダクターズのマイコン・ボードでも公式にサポートされています．つまり複数のベンダのマイコン・ボード間でソフトウェアの流用がしやすいということです．

ライブラリという面では，ESP32用のライブラリはほぼ全てオープンソースとして公開されていることも重要です．それらは，ただ公開しているだけではなく高頻度でソースコードが更新されており，最新技術へのキャッチアップが常に行われています．ただし，その分下位互換性を犠牲にしているところもあり，プロフェッショナルの視点で見ると少し注意が必要かもしれません．

その他にもたくさん便利な機能を備えているのですが，その詳細については次章以降を参照してください．機能の充実を理解していただけると思います．

**▶リファレンス・ボードESP-WROVER-KITについて**

Espressif社はESP32の公式リファレンス・ボードを用意しています．ESP32の全ピンが引き出されているだけではなく，USBシリアル変換IC経由でESP32のJTAGにアクセスすることができるため，デバッグ用のハードウェアを別途用意しなくても，使い慣れたOpenOCD + GNUデバッガ(GDB)[注2]によるデバッグが可能です．

### ● マイコンの入門やプロトタイピングもOK

次のような場合にESP32はお勧めです．

- とにかく作ってみたいIoTシステムがある
- マイコンを使った開発を入門したい
- スマートフォンと連携したりインターネットに接続できるシステムにしたい

ESP32はWi-FiとBluetooth通信機能を両方内蔵しているので，外付け部品なしでスマホ連携やインターネット接続ができます．その開発のためにC，C++，C#，MicroPython，JavaScript，Lua，mrubyなどのメジャーなプログラミング言語の開発環境を使うことができます．またHTTP(S)，MQTT，CoAPなどのIoT向きプロトコルのライブラリが標準で(プログラミング言語により対応レベルの差はあるが)使えます．さらに，M5StackやObnizといった，マイコンが初めての方にも使いやすい市販のESP32搭載ハードウェアを使えば，ハードウェアの知識がそれほどないときのマイコン入門やプロトタイピングにも便利です．下記に代表的なボードを幾つかあげます．

**▶プロトタイピングお勧めボード**

M5Stackはカラー液晶ディスプレイ，スピーカ，センサ，ボタン，microSDカード・スロットなどがオールインワンになったものです．機能拡張用のモジュールやサンプル・プログラムが豊富で，すぐにアイデアを実現できます．開発言語はMicroPythonなどで，ブラウザ上のオンライン・エディタでプログラミングが可能です．またブロック・エディタにも対応しており，ハードウェアを扱うための細かい作法を知らなくてもマイコン・プログラミングができます．

Obnizはインターネット経由で接続したデバイスを制御できる，インターネット・リモコンのようなマイコン・ボードです．Obniz専用のクラウドに常時接続しており，PCやスマホなどからObnizが提供しているAPIを呼び出すだけで，GPIOによるデバイス制御が可能です．APIとしてはクライアント・サイドのJavaScriptやPython用のSDKの他，REST APIもサポートしているため，ほぼ任意の開発言語や実行環境で利用可能です．

### ● ハードが続々登場しているのが注目の理由

これまでに紹介したマイコン・ボードの他にも，ブレッドボードでの試作に適したもの，AIカメラとして使えるもの，ESP32以外を用いた既存の回路にWi-Fi/Bluetoothを追加するのに特化したものなど，いろいろなタイプのESP32ボードが市販されています．

またクラウド・ファンディング・サイトを見てみると，ESP32を使った製品が次々と現れ，開発資金の獲得に成功しています．ESP32の世界を体験していただき，新たなアイデアの実現に役立てていただければ幸いです．

注1：筆者による測定結果．
注2：Linuxで一般的に使われるデバッグ・ソフトウェア．

みやた・けんいち

第1章

すぐに使える回路図とプログラム

# IoTセンサをつなげるハードウェア&ソフトウェア

小池 誠

ここでは，実際に農業などのアウトドア用途にも使えそうな厳選9種類のIoTセンサの使い方を紹介します．マイコンには，ラズベリー・パイやArduinoとはひと味違う，Wi-Fi付きで低価格な新定番IoT向けモジュールESP32を使います．センサをつなげることでどんな可能性があるかも探っていきます．

## ● 実験の構成

実験の構成を**図0-1**に示します．

ソフトウェアの開発環境は，**表0-1**に示す2種類を試してみました．

「Arduino core for ESP32 WiFi chip」は，Arduino IDEを使って開発できるようになるため，Arduino経験者やC++開発経験者にはとっつきやすい環境だと思います．また，USB接続するだけでArduino IDEからプログラムの書き込みができる点もポイントです．MicroPythonは，Pythonを使ったモダンなコーディングができる他，REPLを使って取りあえず動かしてみるといった用途に向いています．Pythonプログラムの実行＆書き込みなどは，Adafruit Micro Python Tool（ampy）[注1]を使用すると便利です．

なお，誌面の都合上，MicroPythonのプログラムのみ掲載し，Arduinoのプログラムはダウンロード・データで用意しました．

## ● 紹介する筆者厳選センサ

今回動かしてみたセンサの一覧を**表0-2**に示します．ESP32の動作電圧は2.3 ～ 3.6V（推奨3.3V）のため，使用センサもそれに合わせて選定する必要があります．特に，開発環境がArduinoと似ているからといって，Arduino用センサ・モジュールを選ぶと5Vが要求されている場合があるので注意が必要です．

注1：https://github.com/adafruit/ampy

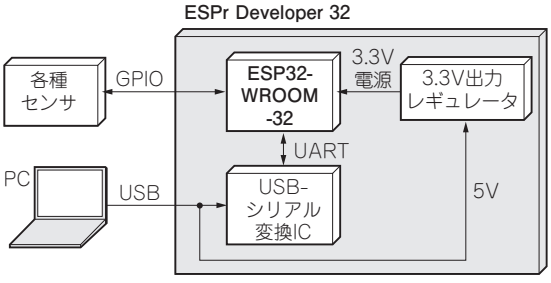



**図0-1　本稿のESP32 IoTセンシング実験の基本構成**

**表0-2　農業などのアウトドア用途にも使えそうで入手しやすいIoTセンサ厳選10種**

| No. | 種　類 | 型　番 | メーカ |
|---|---|---|---|
| 1 | 温湿度・気圧センサ | BME280 | Boshc Sensortec |
| 2 | $CO_2$センサ | CCS811 | ams |
| 3 | 土壌湿度センサ | SEN0114 | DFROBOT |
| 4 | 距離センサ | VL53L0X | ST マイクロエレクトロニクス |
| 5 | 人感センサ | EKMC1601111 | パナソニック |
| 6 | 地磁気センサ | HMC5883L | Honeywell |
| 7 | 加速度センサ | ADXL345 | アナログ・デバイセズ |
| 8 | UVセンサ | VEML6070 | Vishay |
| 9 | 圧力センサ | FSR406 | Interlink Electronics |

**表0-1　IoTセンシング実験のベースに使うESP32開発環境**

| 環境名 | 開発言語 | 使用バージョン | URL |
|---|---|---|---|
| Arduino core for ESP32 WiFi chip | C，C++ | ver.2.0.5 | https://github.com/espressif/arduino-esp32 |
| MicroPython（Firmware for ESP32 boards） | Python | esp32-20220618-v1.19.1.bin | https://micropython.org/download |

本書の記事のプログラムは，以下のページからダウンロードできます．
https://interface.cqpub.co.jp/2023esp/

# 1　温湿度・気圧センサ

プログラム名
micropython\bme280.py

BME280（Bosch Sensortec）は，気温／湿度／気圧センサが1つになったMEMSセンサです（**写真1-1**）．センサ自体は2.5mm×2.5mmと米粒程の大きさですが，SPIとI²Cの2つのインターフェースへの対応や，消費電流を抑えるための動作モード切り替えを搭載するなど，とても高機能なセンサです．しかし，85℃を超える温度は計測できないことや，他の高精度なセンサと比べると若干精度が劣るのが欠点です．気温，湿度，気圧の測定レンジと精度，分解能は**表1-1**の通りです．また，主なセンサ仕様は**表1-2**の通りです．

## ● 応用例

気温，湿度，気圧を一気に取得できるこのセンサは，部屋の環境モニタとしてぴったりです．ESP32でインターネットにつなげて，データをクラウドに常時アップすることで，いつでもどこでも部屋の状況を確認できます．例えば夏場などペットを部屋に残して外出するときも安心ですね．

## ● 回路

回路を**図1-1**に示します．今回はこのセンサを搭載した「BME280使用　温湿度・気圧センサモジュールキット」（AE-BME280，秋月電子通商）を使用します．


写真1-1　BME280センサ・モジュール

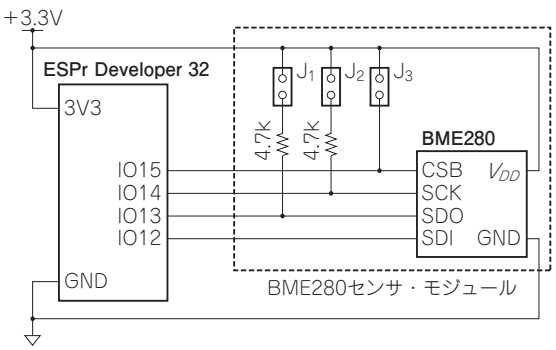



図1-1　ESP32モジュール×温度センサBME280モジュール実験の回路

BME280センサ・モジュールは，SPIとI²Cのどちらでも接続することができます．今回は，SPIを使って接続する方法を紹介します．SPIを使う場合ジャンパJ1～J3のはんだ付けは不要です．なお，I²Cを使用する場合は，J3のみはんだ付けしてください．

## ● プログラム

ノーマル・モードで動作させるプログラムを**リスト1-1**（次頁）に示します．

センサの制御フローを**図1-2**に示します．

### ▶ SPI初期化

初めにSPI通信の初期化を行います．ESP32には

表1-1　温度センサBME280の測定レンジと測定精度と分解能

| | 測定レンジ | 測定精度 | 分解能 |
|---|---|---|---|
| 気　温 | −40～+85℃ | ±1℃ | 0.001℃ |
| 湿　度 | 0～100% | ±3% | 0.008% |
| 気　圧 | 300～1100hPa | ±1hPa | 0.18Pa |

表1-2　温度センサBME280の主な仕様

| 項　目 | 値など |
|---|---|
| 動作電圧 | 1.71～3.6V |
| 動作電流 | 3.6μA（気温，湿度，気圧測定時） |
| | 0.1μA（スリープ時） |
| インターフェース | SPI（3線式・4線式対応，最大10MHz） |
| | I²C（最大3.4MHz） |
| サイズ | 10mm×16mm×2mm（モジュール） |
| | 2.5mm×2.5mm×0.93mm（センサ単体） |

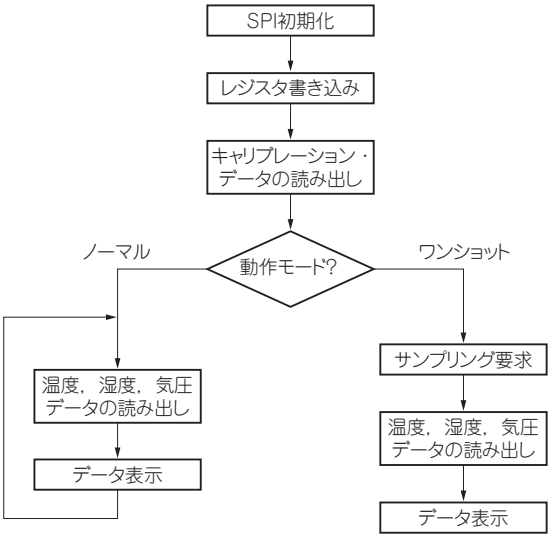



図1-2　BME280の制御フロー
動作モードによって使い方が異なることに注意

**リスト1-1　気温，湿度，気圧を表示するプログラム**（MicroPython）

```
#-*- coding:utf-8 -*-
from machine import Pin
from machine import SPI
import struct
import time

#-- Pin Assignment --
PIN_SCLK = 14
PIN_CS   = 15
PIN_MISO = 12
PIN_MOSI = 13


class BME280:
  def __init__(self):
    #キャリブレーション値
    self._dig_T = 0
    self._dig_P = 0
    self._dig_H = 0
    self._dig_H4 = 0
    self._dig_H5 = 0
    self._dig_H6 = 0
    self._t_fine = 0

    #SPI通信の初期化 --- ①
    self._cs = Pin(PIN_CS, Pin.OUT)
    self._spi = SPI(1, baudrate=10000000,
                                polarity=0, phase=0,
                    sck=Pin(PIN_SCLK), mosi=Pin
                      (PIN_MOSI), miso=Pin(PIN_MISO))

    #レジスタに設定値を書き込む --- ②
    self._spi_write(0xF2, 0x01)
    self._spi_write(0xF4, 0x27)
    self._spi_write(0xF5, 0xa0)
    self.readTrim()

  def _spi_write(self, addr, data):
    addr &= 0x7F
    self._cs.value(0)
    self._spi.write(addr.to_bytes(1, 'big'))
    self._spi.write(data.to_bytes(1, 'big'))
    self._cs.value(1)

  def _spi_read(self, addr, size):
    addr |= 0x80
    data = bytearray(size)
    self._cs.value(0)
    self._spi.write(addr.to_bytes(1, 'big'))
    self._spi.write_readinto(data, data)
    self._cs.value(1)
    return data

  def readTrim(self):
    """ キャリブレーション・データの読み出し ---③
    """
    data = self._spi_read(0x88, 24)
    data += self._spi_read(0xA1,  1)
    data += self._spi_read(0xE1,  7)

    self._dig_T = struct.unpack('<Hhh', data[0:6])
    self._dig_P = struct.unpack
                          ('<Hhhhhhhhh', data[6:24])
    self._dig_H = struct.unpack('<Bhb', data[24:28])

    t0,t1,t2,t3 = struct.unpack('BBBb', data[28:32])
    self._dig_H4 = (t0 << 4) | (t1 & 0x0F)
    self._dig_h5 = (t2 << 4) | ((t1 >> 4) & 0x0F)
    self._dig_h6 = t3


  def readTemperature(self):
    """ 気温データを取得する(℃)
    """
    data = self._spi_read(0xFA, 3)
    data = struct.unpack('BBB', data)

    raw = (data[0] << 12) | (data[1] << 4) | (data[2]
                                              >> 4)
    var1 = (((raw >> 3) - (self._dig_T[0] << 1)) *
                                  self._dig_T[1]) >> 11
    var2 = ((((raw >> 4) - self._dig_T[0]) *
                ((raw >> 4) - self._dig_T[0]) >> 12)
                              * self._dig_T[2]) >> 14

    self._t_fine = var1 + var2
    return ((self._t_fine * 5 + 128) >> 8) / 100.0

  def readPressure(self):
    """ 気圧データを取得する(hPa)
    """
    data = self._spi_read(0xF7, 3)
    data = struct.unpack('BBB', data)

    raw = (data[0] << 12) | (data[1] << 4) | (data[2]
                                                >> 4)
    var1 = ((var1 >> 2) * (var1 >> 2)) >> 11) *
                                        self._dig_P[5]
    var2 = var2 + ((var1 * self._dig_P[4]) << 1)
    var2 = (var2 >> 2) + (self._dig_P[3] << 16)
    var1 = (((self._dig_P[2] * (((var1 >> 2) *
        (var1 >> 2)) >> 13)) >> 3) + ((self._dig_P[1]
                                  * var1) >> 1)) >> 18
    var1 = ((32768 + var1) * self._dig_P[0]) >> 15
    if var1 == 0:
      return 0
    pres = ((1048576 - raw) - (var2 >> 12)) * 3125
    if pres < 0x80000000:
      pres = int((pres << 1) / var1)
    else:
      pres = int((pres / var1) * 2)
    var1 = (self._dig_P[8] * (((pres >> 3) *
                              (pres >> 3)) >> 13)) >> 12
    var2 = ((pres >> 2) * self._dig_P[7]) >> 13
    pres = pres + ((var1 + var2 + self._dig_P[6])
                                                >> 4)
    return pres / 100.0

  def readHumidity(self):
    """ 湿度データの取得(%)
    """
    data = self._spi_read(0xFD, 2)
    data = struct.unpack('BB', data)

    raw = (data[0] << 8) | data[1]
    v_x1 = self._t_fine - 76800
    v_x1 = (((((raw << 14) - (self._dig_H4 << 20)
           - (self._dig_H5 * v_x1)) + 16384) >> 15) *
           (((((((v_x1 * self._dig_H5) >> 10) *
           (((v_x1 * self._dig_H[2]) >> 11) + 32768))
                                    >> 10) + 2097152) *
           self._dig_H[1] + 8192) >> 14))
    v_x1 = v_x1 - (((((v_x1 >> 15) * (v_x1 >> 15))
                          >> 7) * self._dig_H[0]) >> 4)
    v_x1 = 0 if v_x1 < 0 else v_x1
    v_x1 = 419430300 if v_x1 > 419430400 else v_x1
    return (v_x1 >> 12) / 1024.0


sensor = BME280()

while True:
  temp = sensor.readTemperature()
  pres = sensor.readPressure()
  rh   = sensor.readHumidity()

  print("TEMP(deg):%f"%(temp))
  print("RH   ( %% ):%f"%(rh))
  print("PRES(hPa):%f"%(pres))
  time.sleep(1)
```

第3章

マイコンとカメラがセットになった

# 2000円ESP32カメラ「TTGO T-Camera」を使う

岩貞 智

## ESP32+カメラの世界

Wi-FiとBluetoothが利用できて700円から購入できる大人気マイコンESP32ですが，多くの場合，プロトタイプ開発や自作する用途によって，センサなどといった周辺部品と組み合わせる必要があります．特に画像センサ（カメラ・モジュール）は，周辺部品の中でも，最近のAIでの画像認識ブームもあってか人気です．

ESP32などといったマイコンでカメラを作るのは，ラズベリー・パイなどとは異なり，CSI（Camera Serial Interface）やUSBも装備されておらず，簡単にはいかないと思います．そのような状況ですが，ESP32とカメラ・モジュールとを組み合わせたボードが幾つか発売されています．これらを使用することで，簡単にESP32ベースのIoTカメラを開発できます．ここではESP32＋カメラ・モジュールの可能性を探っていきたいと思います．

## ESP32カメラの特徴

ESP32＋カメラ・モジュールの特徴は，本来のESP32の特徴である，

・安価　・省電力　・小型　・ネット接続可

にプラスして，画像を取得し表示できるようになることです．

### ● 無線によるインターネット網への接続が容易

Wi-FiとBluetooth通信機能を搭載するESP32を利用すれば，カメラ・モジュールが取得した画像データをすぐに外部のネットワークに送信できます．送信された画像を開発者やユーザがすぐに閲覧，確認できます．本章で紹介するESP32ボードは，標準でカメラ・モジュールを搭載していることから，公式のリポジトリでの実装例も充実しており，すぐにアイデアを試すことができます．

### ● 定番コンピュータ：ラズベリー・パイ＋カメラ・モジュールと比較してコストに優れる

ラズベリー・パイは一見，本体価格が安く設定されているように見えますが，動作させるまでに必要な周辺機器が意外と多く，トータルで見ると高くつきます．例えば今回のようにカメラ・モジュールを利用したアプリケーションを作成したいと考えて，定番のPiCameraを利用しようと思うと，それだけで4000円以上の追加費用がかかります．

マイコンとカメラ・モジュールがセットになり，2000円前後で買えてしまうESP32ボードは，PiCamera 1つ分で2つ買えてしまうくらい安価です．

### ● 小型

ESP32自体も十分小型ですが，ESP32と組み合わされるカメラ・モジュールも小さいため，Wi-Fi，Bluetoothを利用できつつ，とてもコンパクトに収まります．中にはフリスクのケース（58×32mm）に収まるボードもあります．

### ● 既にデバイスに組み込まれているので配線不要

カメラ・モジュールとマイコンとを接続する際には，複数の配線が必要です．ブレッドボードなどで配線するとゴチャッとなりやすく，初心者にはハードルが高いです．ですが，後述するESP32ボードは，既にカメラ・モジュールがマイコンに接続されているため，開封後，すぐに利用できます．

## マイコンで作れる利点

マイコンで作るネットワーク・カメラから広がる世界を妄想してみます．

### ● 複数設置できる

1万円あれば5台購入できますから，自宅の周辺のあらゆる場所に設置できます．

表1　現時点で入手可能なESP32搭載カメラ・モジュール

| 名称 | SoC | 搭載RAM*1 | イメージセンサ*2 | 値段［円］ | その他 |
|---|---|---|---|---|---|
| M5Camera | ESP32 | 4Mバイト PSRAM | OV2640 | 2,035 | UART（CP2104 USB TTL），USB-Cケーブル付き．EOLとなっており，Timer Camera Xなどが後継製品となる |
| TTGO T-Camera | ESP32 | 8Mバイト PSRAM | OV2640 | 3,899 | LCD（128 × 64），PIR（AS312），UART（CP2104 USB TTL），チャージ・ソケット（IP5306 I²C） |
| ESP32-CAM | ESP32 | 4Mバイト PSRAM | OV2640 | 1,699 | 技適未取得 |
| Unit CAM | ESP32 | − | OV2640 | 1,815 | − |
| PoE CAM | ESP32 | 8Mバイト PSRAM | OV2640 | 6,545 | − |
| Timer Camera X | ESP32 | 8Mバイト PSRAM | OV3660 | 2,959 | − |
| Timer Camera F | ESP32 | 8Mバイト PSRAM | OV3660 | 3,267 | − |
| ESP-EYE | ESP32 | 8Mバイト PSRAM | OV2640 | 2,980 | − |

＊1：疑似SRAMを使用
＊2：全てオムニビジョン製

### ● 壊れても惜しくない

屋外に設置して植物の日々の成長記録を付けられます．濡れたり落下させて壊れても，さほど痛くないです．

### ● コンパクトで邪魔にならない

消しゴムほどの大きさですから，冷蔵庫の中や下駄箱の中，ポストの中にも設置できます．紹介するTTGO T-Cameraで約68×28mm，M5Cameraで約48×24mmです．

### ● 手軽に見守り

外出先からペットや玄関の様子を確認できます．

### ● 動体検知

来客や不審者を簡単に検知できます．これまでは人感センサと言えば，赤外線センサを使うことが多かったのですが，風や熱で誤動作するなどの課題がありました．組み合わせて使うと誤動作を減らせそうです．


写真1　ESP32マイコンとカメラ・モジュールを搭載するM5Camera
M5Stack社が販売している．ESP32と，2Mピクセル・イメージセンサOV2640が標準装備されている

## 向いていないところ

### ● 連続撮影

動画像の取得とネットワークへの配信を同時に行うことは，CPUへの負荷が高いため，性能がそこまで高くないESP32には向きません．2秒撮影→5秒休み→2秒撮影などとインターバル時間を設けて，定期的に画像を取得するような使い方や，特定のイベント発生時のみ動作するような使い方をする必要があります．

### ● 本格的な画像処理

取得した画像に対して，凝った画像処理を施すには，若干，パワー不足を否めません．また，ウェブ・サーバとして動作させる際にも，複数のクライアントからのリクエストをさばけるわけでもありません．性能を引き出すにはリソースを適切に利用したり，工夫したりが求められます．

## ESP32カメラ・ボードの現状

現在，ESP32とカメラ・モジュールを合わせて搭載しているボードとしては，入手性を加味すると3つあります（**表1**）．ただし，ESP32-CAMは「技適未取得」のようですので，実質，選択肢は2つとなります．いずれもPSRAM（疑似SRAM）搭載でRAM容量を強化しています．イメージセンサにはOV2640（オムニビジョン）を使っています．

### ● M5Camera

M5Camera（**写真1**）は，M5Stack社が販売しているESP32と，2Mピクセル・イメージセンサOV2640が標準搭載されています．カメラ・レンズを画像に

第3章

スマート・スピーカの裏方としても使われている

# サーバ機能付き赤外線学習リモコンの製作

崎田 達郎


写真1　本章で製作するサーバ機能付きESP32赤外線学習リモコン「IRServer」

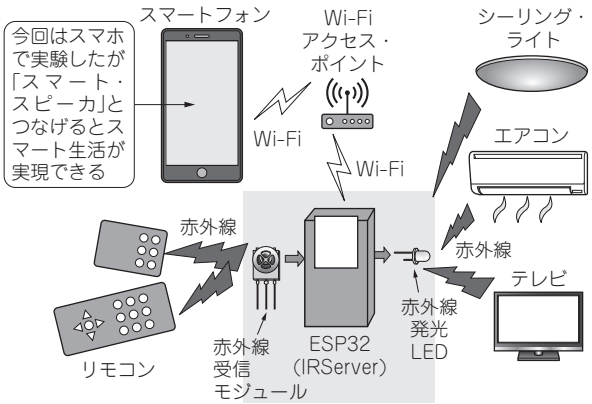



図1　スマート・スピーカの裏側で威力を発揮するサーバ機能付き赤外線学習リモコンの構成

## 作るもの…サーバ機能付きESP32赤外線学習リモコン「IRServer」

### ● スマート・スピーカ時代のマスト・アイテム…サーバ機能付き赤外線学習リモコン

最近，「スマート・スピーカ」に接続される「スマート・リモコン」がいろいろと販売されています．

これらのリモコンはAmazon Echo，Google Homeなどから操作でき，機種によってはスマートフォンから直接操作できるものもあります．これによって家にあるいろいろな家電機器を，機器ごとのリモコンを使わずに統合的に操作できるようになります．

この「スマート・リモコン」を実現するときに必要になるのがサーバ機能付きの赤外線学習リモコンです．ここでは，その「サーバ機能付き赤外線学習リモコン」を自作します（**写真1**）．**図1**にハードウェア構成を示します．

マイコンは安価でWi-Fi/Bluetoothなどの通信機能が一体化されたモジュールESP32-WROOM-32（以降，ESP32と呼ぶ）を使用します．ESP32は搭載するWi-Fi機能などのライブラリが整備されており，それらを使用することで容易にアプリケーションを構築できます．

### ● 機能

製作する学習リモコン（以降，「IRServer」と呼ぶ）の主な機能は次の3つです．

**▶機能1…いろいろな機器のリモコンを学習（記憶）できる**

一般的によく使われている「NEC」，「家電製品協会」フォーマットを対象にします．その他の形式については対応しません．

使用している赤外線受信モジュールが受信できる赤外線信号であれば，受信データ解析処理部分に処理を追加することで対応可能と思われますが，フォーマット（フレーム長やフレーム数）が大きく異なるリモコンへの対応は大きな改造が必要となります．

**▶機能2…ネットワーク経由（HTTP，MQTT）で操作できる**

HTTP（GET）をサポートすることでスマートフォンやPCからウェブ・ブラウザで操作できます．また，Raspberry PiなどのLinuxマシン上のアプリケーションからも操作できます．

**▶機能3…操作画面をカスタマイズできる**

ここでは，操作画面の説明を簡単にするためにHTML + CSS + JavaScriptで作成した画面を使用します．応用例では「任意の画像（JPEG/BMPなど）」上のクリッカブル・マップ[注1]を使ったオ[illegible]ル



本書の記事のプログラムは，以下のページからダウンロードできます．
https://interface.cqpub.co.jp/2023esp/

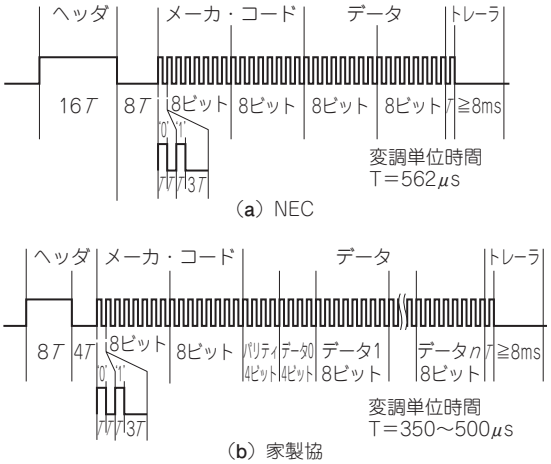



**図2　赤外線リモコン信号のフォーマット**

モコン画面を作れるようにしています．

## 赤外線リモコンの仕組み

### ● 信号フォーマット

国内で使用されるリモコンは主にNECフォーマット，家電製品協会（家製協，AEHA）フォーマット，SONYフォーマットが使われています．今回のリモコンではNECフォーマットと家電製品協会フォーマットだけに対応します．

2つのフォーマットを**図2**に示します．

NECフォーマットは，ヘッダ，メーカ・コード（16ビット），データ（16ビット），トレーラで構成され，全体のデータの長さは固定です．

家製協フォーマットでは，ヘッダ，メーカ・コード（16ビット），可変長のデータ，トレーラで構成され，全体の長さはメーカや機器により決められています．

両フォーマットではヘッダ部の赤外線信号のON時間，OFF時間の長さが異なりますのでヘッダ部のON時間，OFF時間を計測することでどちらのフォーマットであるか判別できます．

注1：クリッカブル・マップ（イメージ・マップ）はブラウザ上に表示した画像上の複数の領域（四角形，円形，多角形）をクリック可能にする機能です．
IRServerでは任意の画像イメージの任意の四角形領域を操作ボタンとして定義するのに使用します．

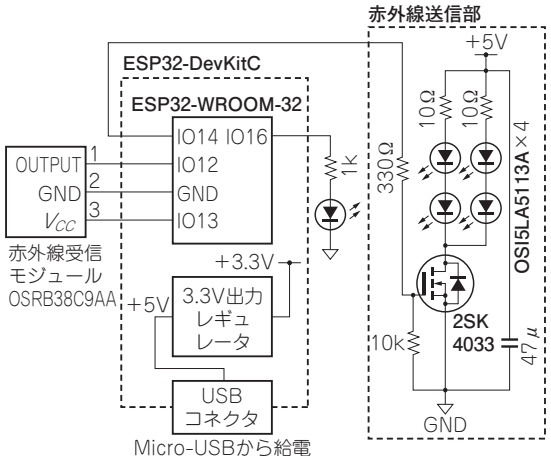



**図3　赤外線学習リモコンの回路**

**表1　赤外線学習リモコンに使用した部品**

| 部品名 | 型　式 | 入手先 | 個数 | 参考価格[円] |
|---|---|---|---|---|
| ESP32ボード注 | ESP32-DevKitC | 秋月電子通商 | 1 | 1,480 |
| 赤外線受信モジュール | OSRB38C9AA | | 1 | 100（2個入り） |
| 赤外LED | OSI5LA5113A | | 4 | 100（10個入り） |
| FET | 2SK4033（60V/4A） | | 1 | 120（5個入り） |
| ユニバーサル基板 | − | | 1 | 200 |
| 基板用リード・フレーム | PD2.54-1.6-7-SN | | 8 | 25（12本入り） |

注：ESP32ボードはその他のESP32ボードでも可．その他に付加回路にある抵抗器とコンデンサが必要

### ● 信号保存形式

学習リモコンとするには，学習元のリモコン信号を読み取り，保存して，必要なときにその信号を再生（赤外線の発光）することで実現します．元のリモコン信号をどの程度正確に再生できるかが操作対象機器（リモコン信号を受け取る側）の操作性（リモコン・ボタンの効き具合）に関わることになります．

この信号の正確性には，赤外線の波長，搬送周波数，変調単位時間，データ値，フレーム長などの項目が影響します．ハードウェアで決定される赤外線の波長以外の項目の全てを元のリモコンから読み取り保存することになります．

他の学習リモコン・アプリケーションでは，搬送信号周波数や個々の信号ビットのON時間，OFF時間を保存して信号フォーマットに依存せず再現性を高める方法を採用しているものもあります．IRServerでは，保存データの簡略化のために，搬送周波数[38kHz，デューティ（duty）比1/3]と変調単位時間（$T$）は，個々のリモコンの固有値を保存せず，データ値だけを保存して，他の項目はフォーマットごとの固定値を使用します．これにより，保存すべき情報量は少なくなりますが，信号の規格合致性に厳格な操作対象機器の場合，再生したリモコン信号を受け付けない可能性があります．

## ハードウェア

**表1**に主な部品を，**図3**に回路を，［　　］に

第3章

アナログ値を読み込んで補正し小型液晶ディスプレイに表示する

# 温度データのセンシング

下島 健彦

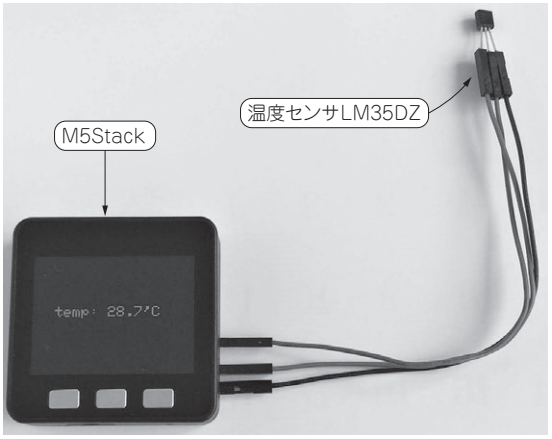



写真1　M5Stackとアナログ温度センサで温度を測る

本章ではアナログ温度センサを使って温度を測ります(**写真1**).

温度，湿度は住居やオフィスにおいて基本的な環境データです．また，農業や製造業でも作業員だけでなく農作物の生育や製造物の精度などに大きな影響を与える指標です．温度，湿度は継続的に測定して記録することで，1日や季節ごとの寒暖の差や平年との差が分かるため，記録し，比較できたら便利です．

## 使用するデバイス…温度センサ

### ● 最初は温度センサで試す

センサには値がアナログ値として読めるものと，ディジタル数値データとして読めるものがあります．

例えば温度センサの場合，LM35DZやLM61BIZ(ともにテキサス・インスツルメンツ)といったアナログ出力タイプは，電源を供給することで周囲の温度に比例した電圧が出力されます．

この電圧をA-Dコンバータでディジタル値に換算すれば温度が得られます.例えばArduinoの場合，この出力を`analogRead()`関数で読み取り，比例計算することで温度が得られます．

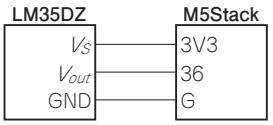



図1　M5Stackと温度センサとの接続

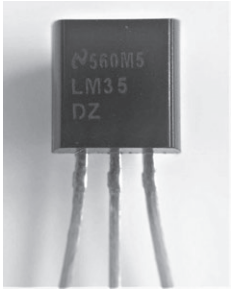



写真2　使用する温度センサLM35DZ…温度に応じて出力電圧が変化する

ディジタル・センサはI²CやSPIといった方式でマイコンとセンサが通信することで，測定したデータが数字として得られます．

ディジタル・センサはアナログ値をディジタルに変換するA-D変換器や，I²CやSPIといった通信回路が内蔵されているのに対し，アナログ・センサの方は構造が比較的単純で，安価なものが多いです．

### ● 今回使用する温度センサ…LM35DZ

温度センサにはサーミスタ，測温抵抗体，熱電対，IC温度センサなどの種類があります．周囲の温度によって抵抗値などの特性が変化することを利用して温度を測ります．

今回利用するLM35DZはIC温度センサの1つで，次のような特性を持っています．

- −55 ～ +150℃の温度を測定できる
- 温度係数はリニアで1℃当たり10mVの電圧が出力される
- +25℃において0.5℃の精度を保証
- 低自己発熱で，静止空気で0.08℃の発熱

M5Stackとは**図1**に示すように接続します．外形は**写真2**のようにトランジスタのようで，3本の足があります．電源とグラウンドを接続すると温度に応じた電圧が出力されます．





本書の記事のプログラムは，以下のページからダウンロードできます．
https://interface.cqpub.co.jp/2023esp/

## 開発環境

### ● 種類

M5StackのプログラムM開発環境としてはArduino IDEとUIFlowが提供されています．

M5StackはESP32を搭載しているので，Espressif Systems社が提供するESP-IDFというESP32用のプログラム開発環境も使えます．

Arduino IDE，ESP-IDF，UIFlowの特徴は次のようになります．

**▶開発環境①…Arduino IDE**

Arduino IDEという統合開発環境を使って開発できます．言語はC++でArduinoのシンプルなAPIとライブラリが使えます．プログラムのサンプルも豊富ですし，インターネット上にもたくさんの作例が公開されています．

**▶開発環境②…ESP-IDF**

Espressif Systems社が提供する開発環境です．言語はC/C++です．ESP-IDF独自の豊富なAPIが提供されており，ESP32のフル機能が使えます．

**▶開発環境③…UIFlow**

グーグルが提供するBlocklyというビジュアル・プログラミング言語をベースにしたプログラム開発環境です．BlocklyとMicroPythonでプログラミングできます．

本稿ではArduino IDEとMicroPythonを使います．

## その①…Arduinoプログラム

### ● アナログ値の読み出しにはESP32内蔵のA-Dコンバータを使う

Arduinoでアナログ温度センサの値を読むには`analogRead()`関数を使います．

M5Stackに搭載されているESP32には，A-Dコンバータが内蔵されています．`analogRead()`はI/Oピンに加えられた電圧をA-Dコンバータで数値に変換して読み取ります．

ESP32のA-Dコンバータの分解能はデフォルトで12ビットで，11dBの減衰器が設定されています．これにより，0～3.6Vの入力に対して0～4095の値が返されます．

### ● A-Dコンバータは補正が必要

ESP32のA-Dコンバータは直線性に問題があることが知られています．

0～3.3Vまで電圧を変えながらテスタとA-Dコンバータの値を比較すると，次の**図2**のような特性になりました．電圧が0～0.2Vくらいまでは`analogRead()`の値が緩やかに上昇し，0.2～2.5Vくらいまでは電圧に比例した値が得られます．

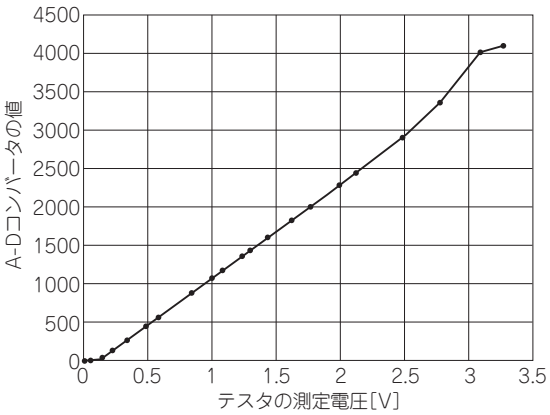



**図2　テスタの測定電圧とESP32のA-Dコンバータの出力値の関係**

そこで，次のような補正関数を使って`analogRead()`で得られる値を補正しました．

```
v = (float)analogRead(PIN) / 4095.0
    * 3.6 + 0.1132;
```

一般的に，A-Dコンバータはノイズの影響を受けます．ノイズの影響を小さくするには0.1μF程度のコンデンサを入れる，複数回測って平均を取るなどの方法があります．ここでは複数回測って平均を取る方法を採用しました．

### ● ステップ1…センサの値の読み出し

この補正関数を使ってLM35DZの値を読んでみましょう．LM35DZの出力は1℃当たり10mV(0.01V)なので，`analogRead()`の値を補正したものを0.01で割ると温度が求められます．

LM35DZの値を読み，M5StackのシリアルとLCDに表示するArduinoプログラムを**リスト1**に示します．

### ● ステップ2…LCDに値を表示する

**リスト1**の6行目の`#define`文をコメントにしたままビルドすると，温度データがLCDに標準フォントで表示されます．

M5StackのLCDやボタンなどをアクセスする関数は以下のAPIのページに書かれています．

```
https://github.com/m5stack/M5Stack/
blob/master/src/M5Stack.h#L19
```

標準フォントで表示するには次の3つの関数を使います．

**▶関数1…`M5.Lcd.setTextSize(uint8_t size);`**

テキスト・サイズを指定します．指定できる値はAPIページには書かれていませんが，ソースコードを見ると1～7の値です．