

Turtle Pico Kit 仕様書



目次

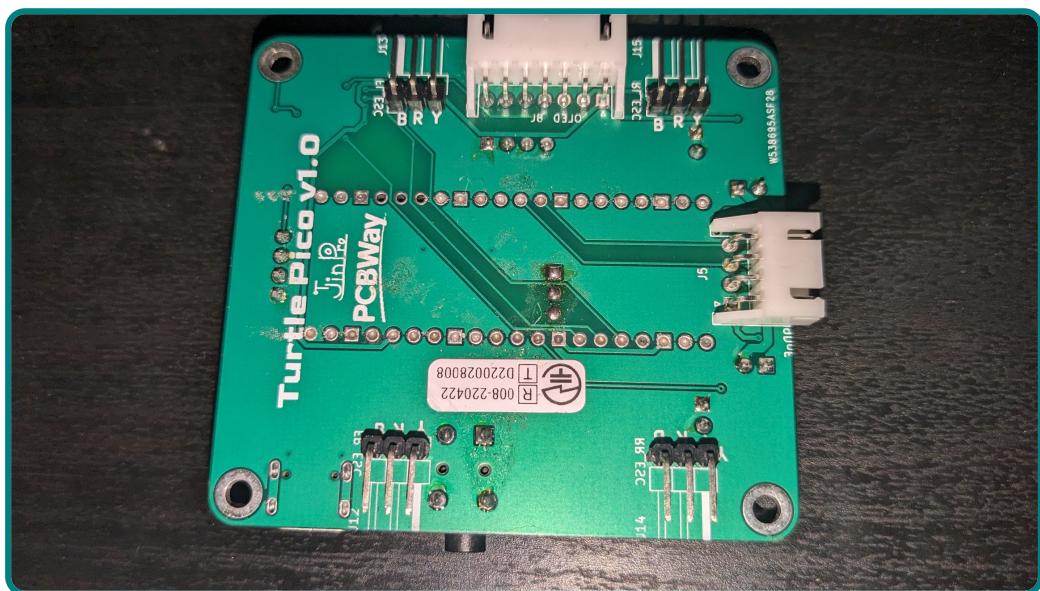
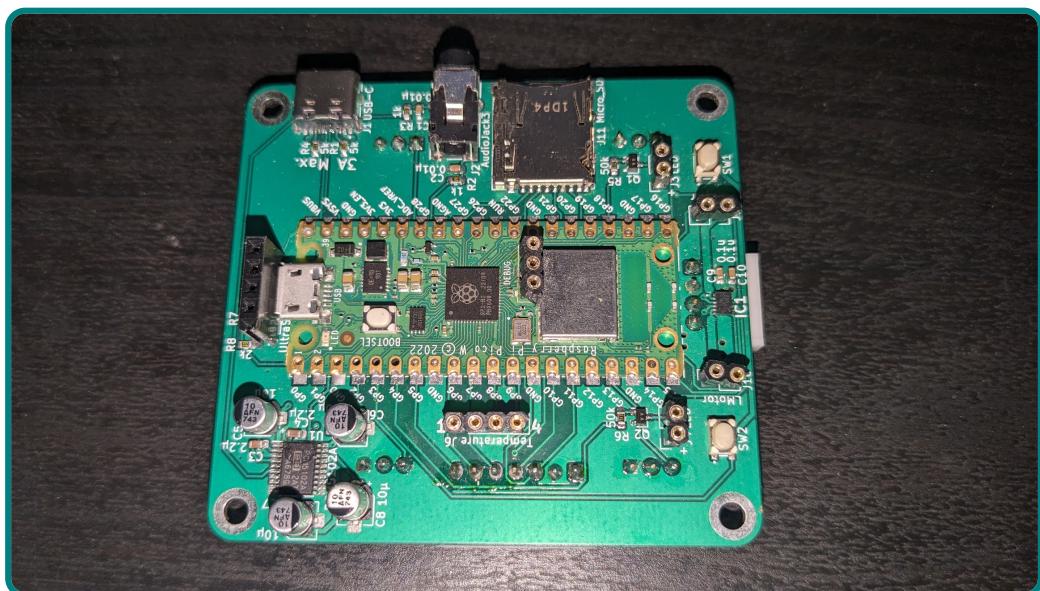
- 1. Turtle Pico Board 概要
 - 1.1. 外観
 - 1.2. インターフェース/機能
 - 1.3. ブロック図
 - 1.4. 回路図/ピンアサイン
- 2. 環境構築
 - 2.1. ファームウェアの書き込み (MicroPython)
 - 2.2. Terminal (REPL)
 - 2.3. Thonny
 - 2.3.1. Windows / Mac
 - 2.3.2. Linux
 - 2.3.3. 全OS共通
 - 2.4. VSCode (Micro Pico)
- 3. プログラム例
 - 3.1. ディレクトリ構成
 - 3.2. LED点灯

1. Turtle Pico Board 概要

本章では、**Turtle Pico Board**の概要を解説致します。

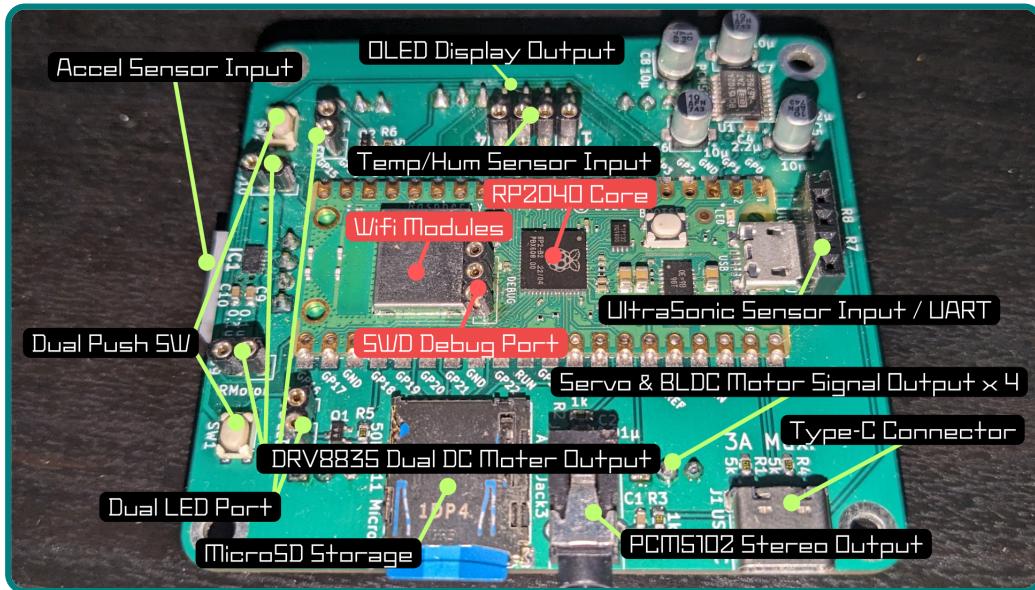
1.1. 外観

Turtle Pico Boardの外観です。上の画像が表、下の画像が裏の写真です。



1.2. インターフェース/機能

インターフェースと機能をまとめた画像を下に示します。



- **USB Type-C Connector**

他のデバイスと通信するための**USBコネクタ**です。電流の定格は、基板の配線幅の関係で**3A**とします。これ以上の電流が流れると、基板上のコネクタ付近が熱を持ち、配線が焼ききれる可能性がありますので、この値は守ってご使用いただくようお願い致します。

- **MicroSD Card Slot (SPI)**

Micro SDカードを接続するためのスロットです。**SPI**通信で接続します。

- **PCM 5102 Stereo Output (I2S)**

3.5mmステレオジャックによる**音声出力ポート**です。PCM5102による出力を直結しています。PCM5102とは**I2S**通信で接続します。

そのままの出力では音が小さいため、スピーカー等大きな出力を必要とする場合はアンプを接続してください。イヤホン等で聞く分に関しては、その必要はありません。

- **OLED Display Port (SPI)**

透明有機ELの制御基板(SSD1309)に接続するためのポート（裏面）です。**SPI**通信で接続します。

- **Servo & BLDC Signal Output**

サーボモーターやブラシレスDCモーターに制御信号を送信するためのポートです。

- **DRV8835 Dual DCMotor Output**

DRV8835モータードライバーに接続する**DCモーター**用のポートが2つついています。DCモーターに流せる定格電流値は、1ポート最大で1.5A、2ポート合計で3Aです。

- **UltraSonic Sensor Input / UART**

超音波センサーの入出力ポートです。超音波センサーを使用しない場合、中央2つのピンを使用して**UART**通信のポートとしても使用いただけます。

- **Temp/Hum Sensor Input (I2C)**

温湿度センサーの入力ポートです。**I2C**通信で接続します。

- **Accel Sensor Input (I2C)**

Accel Sensorとなっていますが実際には**9軸センサー**の入力ポート（裏面）です。**I2C**通信で接続します。

- **Dual Push SW**

Raspi Pico W直結の**プッシュスイッチ**が2つついています。

- **Dual LED Port**

トランジスターのコレクターに接続されている、オープンコレクター出力の**LEDポート**です。出力することができる電流は1ポート最大15mAほどで、PWMで明るさを制御することができます。

- **SWD Debug Port**

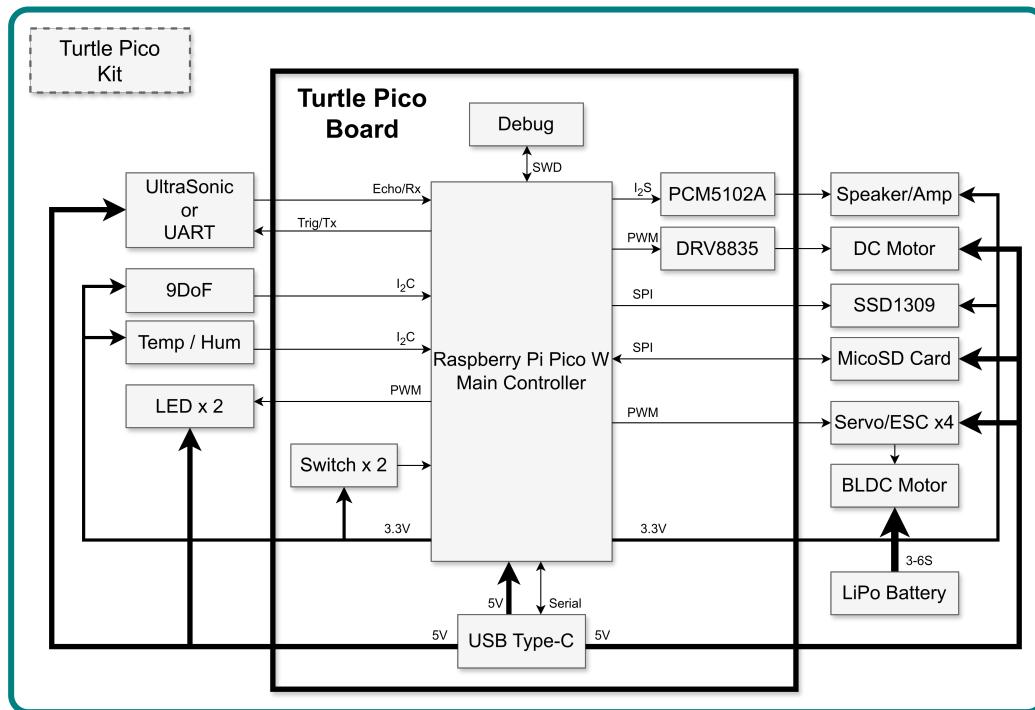
Pico SDKやRTOSなどで使用する場合に**書き込み/デバッグ**に使用するポートです。**Pico Probe**などを使用して**OpenOCD**などでデバッグをします。

- **WiFi Module**

Raspberry Pi Pico W内蔵の**WiFiモジュール**です。バンドは2.5GHzのみに対応しています。このモジュールを使用してWebに接続します。

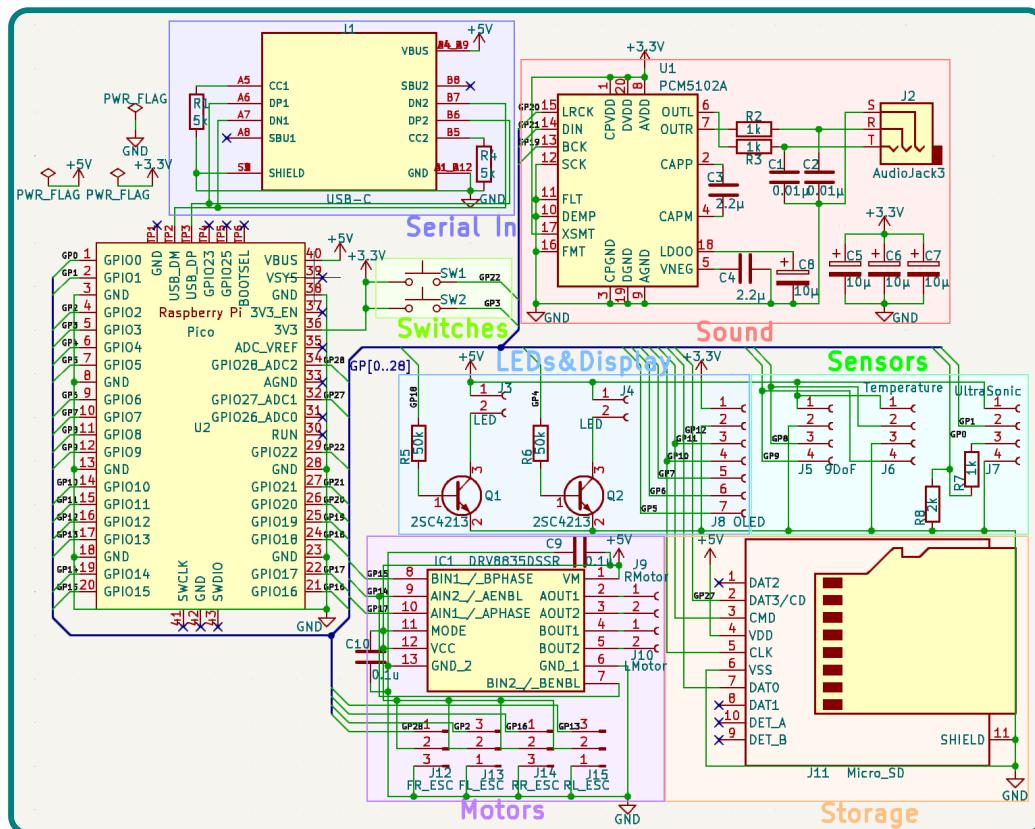
1.3. ブロック図

Turtle Pico Boardのブロック図を以下に示します。



1.4. 回路図/ピンアサイン

Turtle Pico Boardの回路図及びRaspberry Pi Pico Wの対応するピンアサインを以下に示します。



GPIO	Funciton
0	Uart - Rx / Ultrasonic - Echo
1	Uart - Tx / Ultrasonic - Trig
2	ESC / Servo - Signal
3	Push SW (Left)
4	LED (Left)
5	OLED(SSD1306) - RST
6	OLED(SSD1306) - DC
7	OLED(SSD1306) - CS
8	I2C - SDA
9	I2C - SCL
10	SPI - SCK
11	SPI - MOSI
12	SPI - MISO
13	ESC / Servo - Signal
14	DRV8835 - Enable
15	DRV8835 - Bin
16	ESC / Servo - Signal
17	DRV8835 - Ain
18	LED (Right)
19	I2S - BCLK (SCK)
20	I2S - LRCLK (WS)
21	I2S - SDATA
22	Push SW (Right)
26	None
27	SD Card - CS
28	ESC / Servo - Signal

2. 環境構築

Turtle Pico(RaspberryPi Pico)の環境構築には様々な方法があります。ここでは、代表的な以下の3つについて、簡単に紹介します。

1. Terminal (REPL)
2. Thonny
3. VSCode (Micro Pico)

言語はいずれも**MicroPython**で、C言語での環境はここでは取り上げません。また、MicroPythonとはいいつつも、Pythonの環境をわざわざ構築する必要はなく、上記3つの方法で手軽に環境を構築することができます。

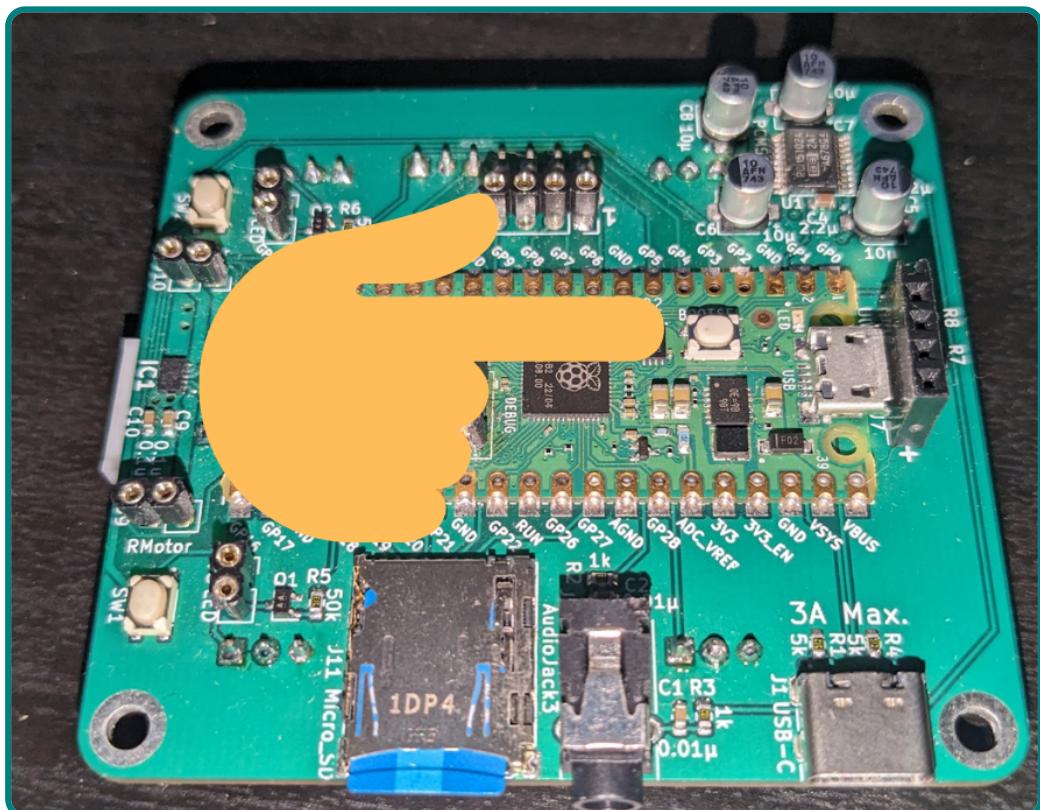
※厳密にはThonnyはPythonのIDEでPythonの環境が必要みたいですが、導入するときに自動で設定してくれているのか、なにか特別に導入する必要はありません。

2.1. ファームウェアの書き込み (MicroPython)

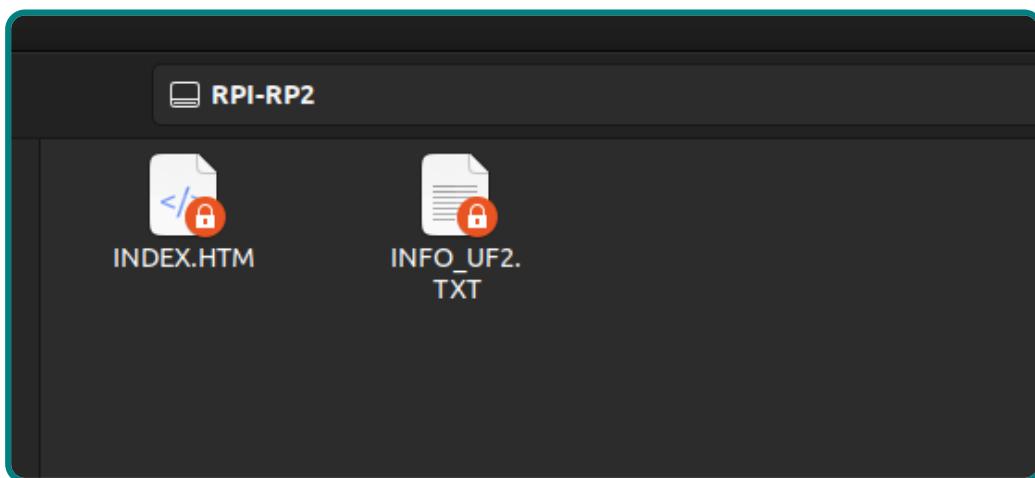
ファームウェアとは、Raspberry Pi Picoにて**MicroPython**を実行するのに必要なソフトウェアのことです、拡張子は.**uf2**です。

ファームウェアに書き込むにはまず、Raspberry Pi PicoをストレージモードでPCと接続する必要があります。

ストレージモードにするには、下の画像の位置にある**BOOTSEL**ボタンを押しながら、USB TypeCケーブルを使用してPCとTurtle Picoを接続するだけです。



ストレージモードで接続すると、その名の通りストレージとしてアクセスできます。RPI-RP2という名前でマウントされるので、そのストレージを開きます。



ストレージを開くと、上の画像のようになっており、INFO_UF2.TXTファイルと、INDEX.HTMファイルの2つがあるのがわかります。このうち、INDEX.HTMファイルを開き、Raspberrypi公式HPにアクセスしてください。

次に、一覧のメニューからMicroPythonをクリックし、Drag-and-Drop MicroPythonの中の Download the correct MicroPython UF2 file for your board: からRaspberry Pi Pico Wを選択し、ファームウェア (.uf2ファイル) をダウンロードします。

Download the correct MicroPython UF2 file for your board:

- Raspberry Pi Pico
- Raspberry Pi Pico W with Wi-Fi and Bluetooth LE support

Documentation introducing working with Wi-Fi and Bluetooth on Raspberry Pi Pico W can be found in the [Connecting to the Internet with Raspberry Pi Pico W book](#). Full details of software stack can be found in the [BTStack Github repository](#).

NOTE

あとはダウンロードしたuf2ファームウェアを、ドラッグアンドドロップでRPI-RP2にコピーするだけで、ファームウェアが書き換わります。コピーが終わると、自動的にシリアルデバイスとして認識するようになります。

※このとき自動的に認識しなければ、再度ケーブルを挿し直してください。

補足ですが、Linuxユーザーはシリアル通信するときには注意が必要です。というのも、Raspberry Pi Picoをシリアル通信で接続したときに、設定をしていないと**権限**の問題が発生してしまいます。

毎回`chmod`などを使用して権限を設定しなおすのも面倒かと思いますので、

```
$ sudo adduser {username} dialout
```

のように`adduser`コマンドを使用し、`dialout`グループをユーザーに追加する方法を推奨します。

2.2. Terminal (REPL)

Windowsであれば**TeraTerm**、MacやLinux各種であれば、コマンドラインから`cu`や`minicom`コマンドを使用して**2.1 ファームウェアの書き込み済みのTurtle Picoにシリアル通信**で接続するだけで、対話形式でプログラムの実行が可能です。

ボーレートはどの設定でも基本繋がるようです（おそらく、USB-CDCデバイスの仕様）。

TeraTermの設定について詳しくはここで説明しませんが、シリアルポートでCOMデバイスを選択して接続すれば利用できます。

`cu`コマンドを使用する場合は、下図のように`-l`でデバイスを選択する必要があります。下記はLinuxでの例です。

```
$ cu -l /dev/ttyACM0
```

```
> cu -l /dev/ttyACM0
Connected.

>>> print('Hello World')
Hello World
>>> 
```

Pythonの構文等の説明につきましては、ここでは割愛させていただきます。

2.3. Thonny

Thonnyを使用する方法は簡単です。導入しないといけないソフトウェアは1つだけで、**Thonny**というPythonのIDEのみです。

- **Thonny**

OSは**Windows**、**Mac**、**Linux**に対応しています。

2.3.1. Windows / Mac

公式サイトよりインストーラーをダウンロードし、インストールしてください。

2.3.2. Linux

下記のように**wget**コマンドを使用してシェルスクリプトを読み込ませるだけで、簡単にインストールできます。

```
$ wget -O - https://thonny.org/installer-for-linux
```

起動する時は、コマンドラインに**thonny**と入力するだけでアプリが使用できるようになります。

2.3.3. 全OS共通

pipコマンドで、各OS上のターミナルで下記のようにコマンドを入力し、インストールすることも可能です。

この方法では、**pip**のインストールが事前に必要ですが、それについてはここでは割愛させていただきます。

```
$ pip install thonny
```

```

import time
from machine import Pin

# ピンを設定
trig = Pin(1, Pin.OUT)
echo = Pin(0, Pin.IN)
red = Pin(4, Pin.OUT)
blue = Pin(18, Pin.OUT)

while True:
    trig.low()
    time.sleep_us(2)
    trig.high()
    time.sleep_us(10)
    trig.low()
    signaloff, signalon = 0, 0
    while echo.value() == 0:
        signaloff = time.ticks_us()
    while echo.value() == 1:

```

シェル

```

distance: 17.1043 cm
distance: 17.0471 cm
distance: 17.02995 cm
distance: 16.99565 cm
distance: 17.0814 cm
distance: 17.1157 cm
distance: 17.13285 cm
distance: 17.09855 cm

```

MicroPython (Raspberry Pi Pico) • Board in FS mode @ /dev/ttyACM0 三

画像はlinuxで実行したものになります。

Thonnyでの詳しい操作方法や導入方法等は、お手数ですが公式Wikiをご覧ください。

2.4. VSCode (Micro Pico)

```

#!/usr/bin/python
# ピンを設定
trig = Pin(1, Pin.OUT)
echo = Pin(0, Pin.IN)
red = Pin(4, Pin.OUT)
blue = Pin(18, Pin.OUT)

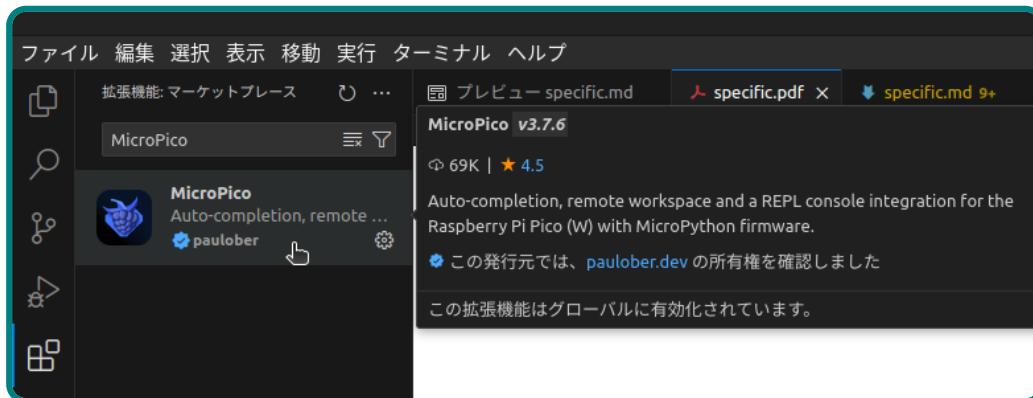
while True:
    trig.low()
    time.sleep_us(2)
    trig.high()
    time.sleep_us(10)
    trig.low()
    signaloff, signalon = 0, 0
    while echo.value() == 0:
        signaloff = time.ticks_us()
    while echo.value() == 1:

```

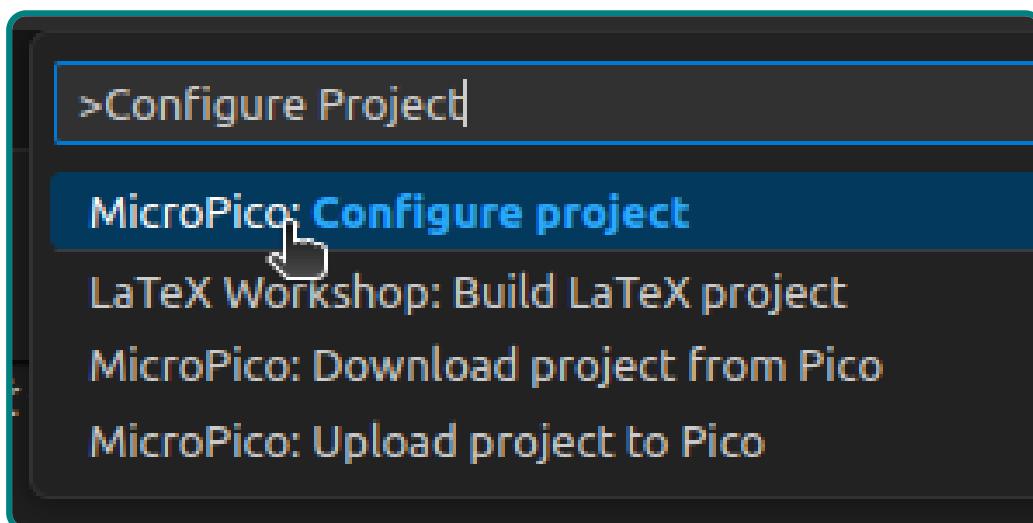
VSCodeというよく知られたテキストエディタを使用する方法もあります。導入しないといけないものが多いですが、こちらは非常に操作がしやすく、デザインもよくて分かりやすく、更にエディタ機能が充実しており、話題のAIなども拡張機能で導入できるのでおすすめです。

- **VSCode本体**
- VSCode拡張機能 - **MicroPico**
- VSCode拡張機能 - **Python**
- VSCode拡張機能 - **IntelliCode**
- VSCode拡張機能 - **Pylance**

VSCode本体については、[公式HP](#)より各OSのインストーラをダウンロードして、インストールするだけです。**VSCode拡張機能**については、VSCode内の拡張機能のメニューから、それぞれの名称で検索をかけて導入するだけです。



MicroPicoを起動するには、フォルダを開いた状態で、**コマンドパレット** (ショートカット: **Ctrl + Shift + P**で開く) を開き、**MicroPico: Configure Project**を選択する必要があります。これによって設定されたMicroPicoフォルダをVSCodeで開くと、以降毎回自動でMicroPicoを起動してくれます。



MicroPicoのそれ以外の操作につきましては、ここでは割愛させていただきます。
お手数ですが、詳細は[公式リポジトリ](#)をご覧ください。

3. プログラム例

ここからは、各部品を制御する**テストプログラム**についての紹介と、各制御方法などをご紹介させていただきます。

ここで紹介するソースコードは、[Turtle Pico公式リポジトリ](#)に掲載しており、[MITライセンス](#)で提供しています。ご自由にダウンロード/改変してお使いください。

※ここで紹介するのは、あくまで**テスト用**のものなので、そこまで高度な制御を提供するものではありませんので、その点だけ予めご了承ください。

3.1. ディレクトリ構成

[Turtle Pico公式リポジトリ](#)に格納しているプログラム例のディレクトリ構成は下記のようになっております。

```
IoT-IdeaKit/kit/src/
├─ 9Axis/
│   └─ lib/
├─ BLDCMotor/
│   └─ lib/
├─ DCMotor/
│   └─ lib/
├─ I2S_Sound/
│   └─ lib/
├─ Ironman/
│   └─ lib/
├─ MusicPlayer/
│   └─ lib/
├─ SDCard/
│   └─ lib/
├─ Sensors_Web/
│   └─ lib/
├─ Servo/
│   └─ lib/
├─ SmartHomeo/
│   └─ lib/
├─ Temperature/
│   └─ lib/
├─ Ultrasonic/
│   └─ lib/
├─ flexibleLED/
│   └─ lib/
├─ flexibleLED_Web/
│   └─ lib/
└─ oled/
    └─ lib/
```

各フォルダには、**lib** フォルダ、及び**main.py**と**boot.py**ファイル、**.micropico**ファイルの4つで成立しています。各ファイルの役割は、下記のとおりです。

- **lib**

必要なライブラリのファイルを格納しています。使用する時は、

```
from {fileName} import {className}
```

として使用します。

- **.micropico**

VSCode拡張機能のmicropicoを、該当のフォルダ内で有効にするファイルです。特に意味はありません。

- **main.py**

MicroPythonのメインとなるファイルです。**boot.py**の次に実行され、このファイルの処理が終了するとPicoは**休止状態**となってしまいますので、繰り返し実行する必要がある場合は、**while**ループなどを使用して処理を続行させます。

- **boot.py**

最初に実行されるファイルです。ボード上の使用する各パーツのセットアップなどを記述すると良いですが、**main.py**の冒頭に各フォルダごとにそれらを記述すればいいので、特に使用することはありません。

全ライブラリフォルダには、下記に示す**TurtlePico.py**というTurtlePicoボードを動かす上で必要な定義ファイルを配置しています。このクラスから、ピン情報を参照することができます。

```
# Turtle Pico Pin Define File

class TurtlePico:

    LED_R:        int = 18
    LED_L:        int = 4

    SW_R:         int = 22
    SW_L:         int = 3

    MOTOR_EN:     int = 14
    MOTOR_R:      int = 17
    MOTOR_L:      int = 15

    SPI_ID:       int = 1
    SPI_SCK:      int = 10
    SPI_MOSI:     int = 11
    SPI_MISO:     int = 12

    OLED_CS:      int = 7
    OLED_DC:      int = 6
    OLED_RST:     int = 5

    SD_CS:        int = 27

    I2C_ID:       int = 0
    I2C_SCL:      int = 9
    I2C_SDA:      int = 8

    I2S_ID:       int = 0
    I2S_BCLK:     int = 19
    I2S_LRCLK:    int = 20
    I2S_SDATA:    int = 21

    ESC_SERVO_FR:int = 28
    ESC_SERVO_RR:int = 16
    ESC_SERVO_FL:int = 2
    ESC_SERVO_RL:int = 13

    TRIG_TX:      int = 1
    ECHO_RX:      int = 0
```

ピン情報の詳細は、[1.4. 回路図/ピンアサイン](#)を参照してください。

バージョンによってピンアサインが異なるので、ダウンロードするバージョンを間違えないように注意してください。

3.2. LED点灯

LEDを点灯させるには、マイコンの出力では電流が足りません。そこでTurtle Picoでは、マイコンからの出力をトランジスターのベース部分につなぎ、出力のピンソケットをVCCとコレクターの間に接続することで、**オープンコレクタ接続**としています。

これにより、最大で15mAほどまでの電流が出力可能になります。この電流の値は、接続するLEDのVFがVccを超えない限り一定となります。

それではプログラムを解説していきます。まず、ライブラリの読み込み部分です。

```
import machine
from machine import PWM
import time
from lib.TurtlePico import TurtlePico
```

`import`はライブラリを読み込むための命令で、`machine`ライブラリは主にRaspberry Pi Picoの**ハードウェア**に関連するもの（タイマーやGPIOなど）で、2行目ではその中でも**PWM**クラスを読み込んでいます。

3行目の`time`ライブラリは時間管理をするためのもので、中でも遅延用の`sleep`関数などはよく使用します。

最後の行は、[3.1 ディレクトリ構成](#)で説明した**TurtlePico**ライブラリを読み込んでいます。

ピンは以下のようにして設定します。

```
blue = PWM(machine.Pin(TurtlePico.LED_R, machine.Pin.OUT))
red = PWM(machine.Pin(TurtlePico.LED_L, machine.Pin.OUT))
```

`PWM`はGPIOピンからHighとLowの出力を繰り返す矩形波を出力し、そのうちHigh時間の割合（**デューティ比**）を変化させることにより、平均的な電圧を上げたり下げたりする機能で、マイコンの基本的な機能の一つです。ボード右側のLEDを`blue`、左側のLEDを`red`として定義しています。

High(3.3V)とLow(0V)のみしか出力することが出来ないRaspberry Pi Picoでも、`PWM`を使用すれば0～3.3V間で連続的に電圧を変化させることができるため、本プログラムではこれを使用して**明るさ**を調節します。

`PWM`の該当するピンはトランジスターのベースにつながっている**抵抗器**に接続されているため、この電圧を変更すると**オームの法則**に従ってベースに流れる電流も変化し、オープンコレクタ出力の電流も同様に変化します。

```
blue.freq(1000)
red.freq(1000)
```

ここでは、各PWM出力の波形の周波数を調節しています。PWMの電圧を変化させるのに必要なのはデューティ比だけですが、周波数がおそすぎるとこの変化が目に見えてしまい、ただ点滅しているだけにしか見えないので、`freq`関数の引数に周波数を与え、周波数を目に見えない速度に設定する必要があります。

```
while True:
    if i > 65536:
        status = 1
    elif i < 0:
        status = 0
    red.duty_u16(i)
    blue.duty_u16(65536-i)
    print(i)
    if status == 0:
        i += 30
    else:
        i -= 30

    time.sleep(0.001)
```

`aaa` デューティ比は`duty_u16`関数によって、0~65535の間で設定する必要があり、これが0~100%に対応しています。このプログラムでは、`i`という変数を0~65536までの間で30ずつ増減させます。最初に0から65536まで向かう時は`status`という変数を0にして`i`が**増加中**だということを示し、65536から0に向かう時は`status`を1にして`i`が**減少中**だということを示します。

`status`がどのような値であるかで、`i`に30を加えるか/減じるかを`if`分岐によって処理しています。またそうして変化するデューティ比を、`red`ではそのまま、`blue`では65536から引くことによって出力し、`bule`と`red`で明るさを**反転**させています。

最後に、`print`でデューティ比をデバッグように出力し、また各ループごとに`sleep`関数で1ミリ秒の遅延を挟んでいます。