

概要 3

必要な物品リスト 3

PC を WiFi に接続??

準備 3

ステップ 0 (物品準備、PC を WiFi に接続)3

ステップ 1 (ターミナル接続 6

ステップ 2 (WiFi 設定)8

Hello Real World (Lチカを実行する) 10

配線 10

プログラムを書く 12

実行する 3

Raspberry Pi について 14

Raspberry Pi Zero のピン配列 14

JavaScript の基礎 16

GPIO を試す 17

GPIO を理解する 17

GPIO 出力 17

回路について 18

コードを読む 18

GPIO 入力 19

コードを読む 19

GPIO 入力(ポーリング)19

コードを読む 20

I2C デバイスを試す 21

I2C を理解する 21

SHT30 編 21

I2C センサー(SHT30)が認識されていることを確認する 23

実行する 23

コードを読む 23

ADT7410 編 3

I2C センサーが認識されていることを確認する 24

実行する 25

IoTを試す 25

遠隔LEDコントロール 25

IoT3

WebSoeketとRelayServer3

配線する 26

CHIRIMEN デバイス側にコードを入れ、実行する 27

PC側のコードを準備し、実行する 28

コードを読む 29

Raspberry Pi Zero 側コード 29

PC側コード 29

自分専用チャンネルで制御 29

他のいろいろなデバイスを試してみる 30

常駐プログラム化する 30

予備知識 32

CHIRIMEN ブラウザー版との差異 3

CHIRIMEN 環境の任意のディレクトリへのセットアップ 32

CHIRIMEN Raspberry Pi Zero W チュートリアル

概要

CHIRIMEN Raspberry Pi Zero 版 を用いた IoT 実習資料です。

[pizeronodejs.md](#) の内容をもとに、Web Serial RPiZero Terminal を使うことで操作を簡単化し、更にプログラム作法を ECMA Script Module にあわせています。

準備

ステップ 0 (物品準備、PC を WiFi に接続)

必要な物品リスト

以下を用意します

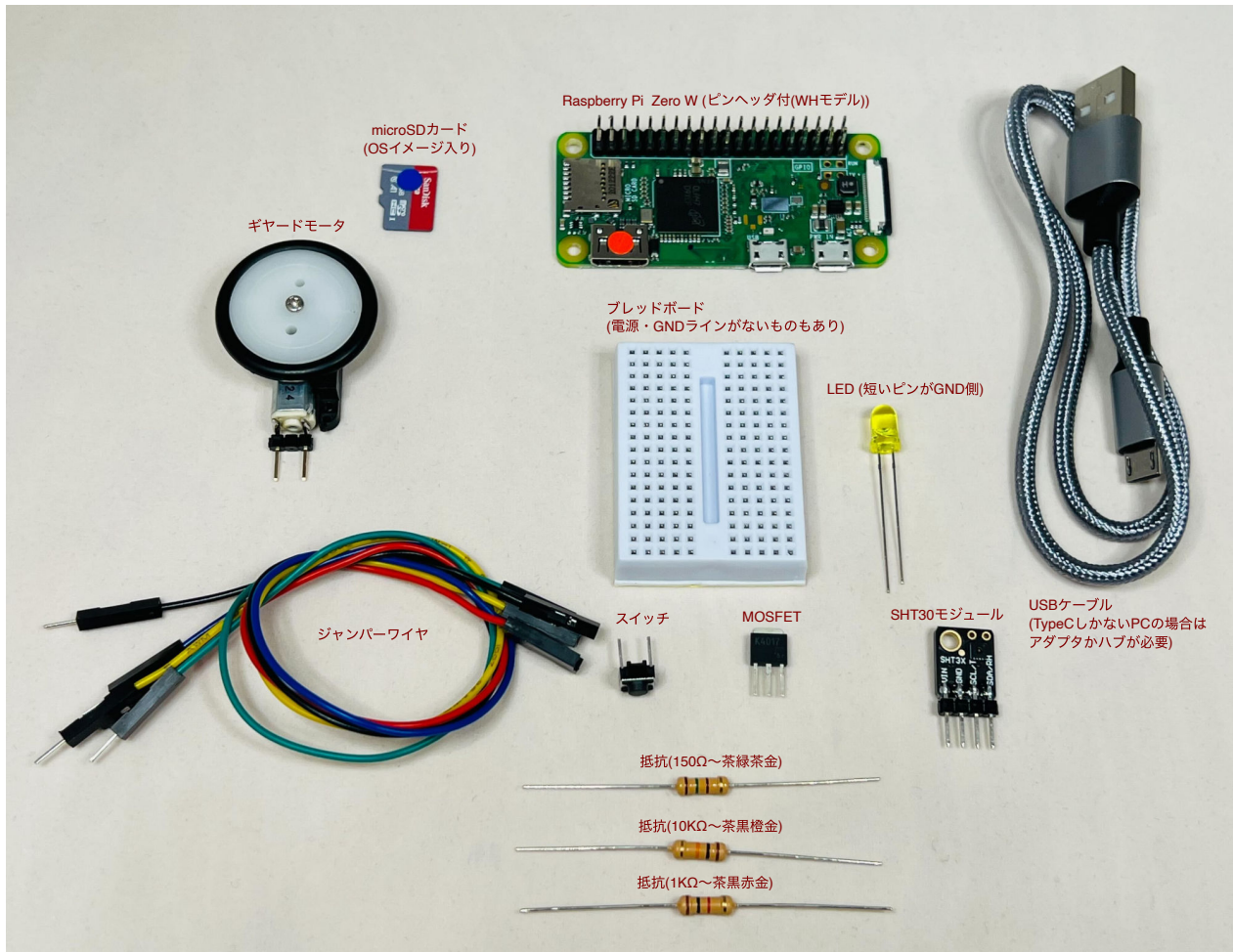
- Raspberry Pi Zero W 、 (Pi Zero **2** W も使用できます(2022/10/12 update))
 - Pi Zero: ケイエスワイ^{*1}, 秋月電子^{*2}, スイッチサイエンス^{*3}, マルツ^{*4}
 - Pi Zero **2** W: ケイエスワイ^{*5}
- Raspberry Pi OS Lite を USB Serial で使用可能にしたイメージ^{*6} を書き込んだ microSD カード
 - 講習会では書き込み済み microSD が配布されると思いますが、自分で作る場合は[こちら](#)や[こちら](#)(書き込むべきイメージは Lite 版の方です。(これを書き込みます^{*7}))
- ブラウザの載ったパソコン
 - Windows 10 PC
 - ブラウザは標準の Edge もしくは Chrome を使います。
 - Macintosh
 - ブラウザは Chrome が必要です。
 - いずれも USB と WiFi が使える必要があります。
 - *Note: Linux PC の Chrome では次の設定で利用可能になるとの報告をいただいています*
 - Ubuntu Studio: `sudo chmod a+rw /dev/ttyACM0`

-
1. <https://rasberry-pi.ksyic.com/main/index/pdp.id/799/pdp.open/799>
 2. <https://akizukidenshi.com/catalog/g/gM-12961/>
 3. <https://www.switch-science.com/catalog/3646/>
 4. <https://www.marutsu.co.jp/pc/i/1320453/>
 5. <https://rasberry-pi.ksyic.com/main/index/pdp.id/851/pdp.open/851>
 6. <https://github.com/chirimen-oh/chirimen-lite/releases>
 7. <https://github.com/chirimen-oh/chirimen-lite/releases>

- Ubuntu Desktop 20.04 LTS: `sudo gpasswd -a "$(whoami)" dialout`
- USBケーブル (USB A - MicroB)
- Lチカ用パーツ
 - ブレッドボード
 - LED
 - 1K Ω 抵抗
 - ジャンパーワイヤ オス-メス 2本
- GPIO入力実験用追加パーツ
 - タクトスイッチ
- モーター制御用追加パーツ
 - 10K Ω 抵抗
 - MOSFET
 - ちびギヤモーター
- 温度センシング実験用追加パーツ
 - ADT7410 モジュール^{*8} もしくは SHT30 モジュール^{*9}
 - ジャンパーワイヤ オス-メス 2本

8. <https://akizukidenshi.com/catalog/g/gM-06675/>

9. <https://www.amazon.co.jp/dp/B083NHJSL9/>



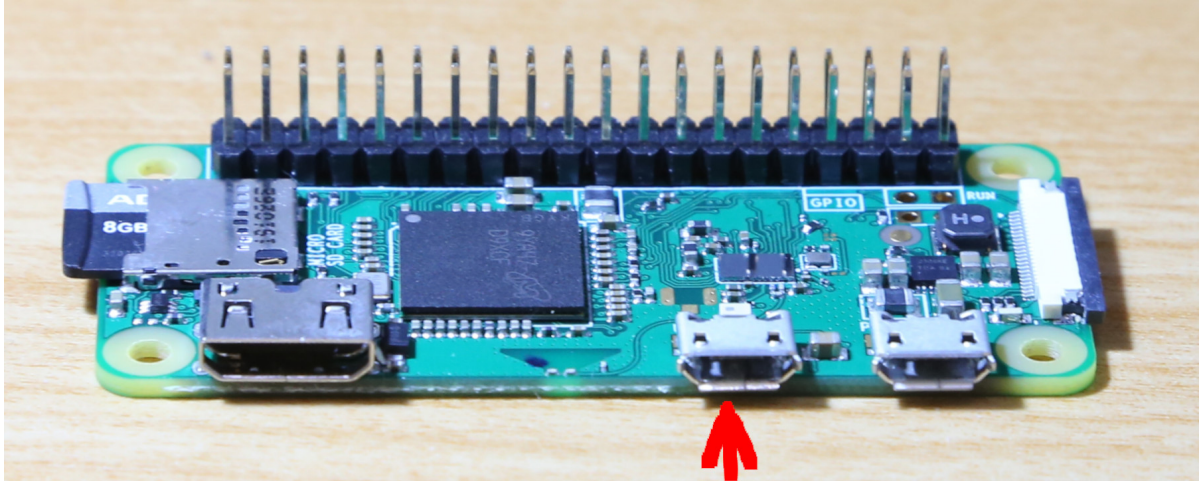
PiZero 自体はディスプレイやキーボードを接続する必要はありません。



PCをWiFiに接続

- 会場(もしくは開発場所)で提供されている WiFi アクセスポイントにまずはPCを接続してください。

ステップ1 (ターミナル接続*¹⁰)

- Raspberry Pi OS LiteをUSB Serialで使用可能にしたイメージ*¹¹を書き込んだmicroSDカードをRaspberry Pi Zeroに差し込みます。
- PCのUSBとRaspberry Pi ZeroのUSB OTGポートをUSBケーブルでつなぎます
 - PiZero側はつなぐポート要注意 ここに繋がります



- PCからのUSB給電でRaspberry Pi Zeroが起動します。
 - PCでRaspberry Pi Zeroが認識されたことを確認します (Windows10のデバイスマネージャ*¹²の例)
 - 給電後USBデバイスとして出現するまでにしばらく(数十秒)かかります
 - Windowsの場合、ポートの番号(COMnのnの部分)は環境ごとに異なります
- ▼  ポート (COM と LPT)
-  USB シリアル デバイス (COM5)
- こちらの**Web Serial RPiZero Terminal**ページにPCのブラウザでアクセス*¹³ (以降、このウィンドをターミナルウィンドと呼びます)
 - [Connect and Login PiZero] ボタンを押す

10. <https://chirimen.org/PiZeroWebSerialConsole/PiZeroWebSerialConsole.html>

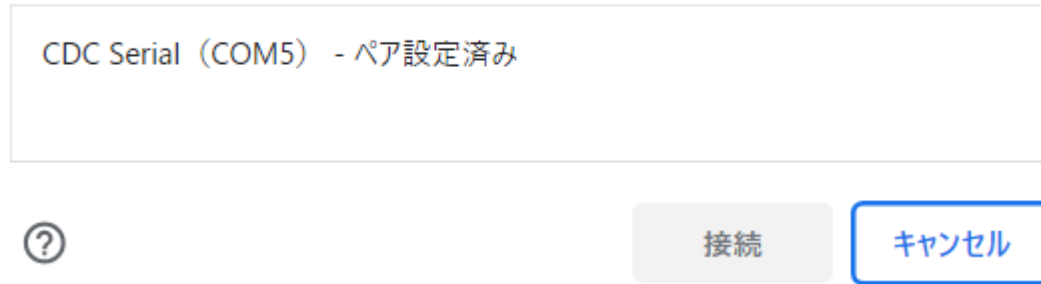
11. <https://github.com/kou029w/chirimen-os/releases/>

12. <https://askpc.panasonic.co.jp/beginner/guide/ten07/7013.html>

13. <https://chirimen.org/PiZeroWebSerialConsole/PiZeroWebSerialConsole.html>

- 接続ダイアログが出現

chirimen.org がシリアルポートへの接続を要求しています



- 上で認識したデバイス（ポート番号）を接続する
- コンソール(左側の黒い画面の最下部)に以下のコマンドプロンプトが表示されればステップ1完了です。引き続きステップ2に進んでください
 - `pi@raspberrypi:~$`

Note:

- CHIRIMEN Raspberry Pi Zero 版では Raspberry Pi OS Lite^{*14}(Linux)をコマンドラインインターフェース(CLI)・シェル(bash)で操作します。
 - ただしこの講習で使うコマンドはごくわずかです。
 - **node** コマンド(後述)
 - CTRL+c^{*15}(CTRL キーとcを同時に押す:実行中のコマンドを終了させる))
 - その他のほとんどの操作（コマンド）は、ターミナルウィンドやそこから起動される別画面のGUIがコマンド操作を代行しています。図1.1のGUIを操作するとコンソールにコマンドが入力されるのがわかると思います。
- CLIとは^{*16}
- シェルとコマンドプロンプト^{*17}
- もしもあなたがlinuxのシェルコンソール画面に慣れている場合は、ターミナルウィンドのコンソールにその他のシェル(bash)コマンドをタイプして使用することもできます。
 - たとえば `ls -al` とタイプするとおコンソール画面にディレクトリ内のファイルのリストが表示されます。
- ターミナルウィンドの概要 (図1.1)

14. <https://www.raspberrypi.com/software/operating-systems/>

15. https://atmarkit.itmedia.co.jp/ait/articles/1708/04/news015_2.html

16. <https://atmarkit.itmedia.co.jp/ait/articles/1602/19/news025.html>

17. <https://atmarkit.itmedia.co.jp/ait/articles/1603/02/news016.html>

ステップ2 (WiFi設定)

- ターミナルウィンドの `[wifi panel]` ボタンを押す
- ウィンドが開き、WiFiアクセスポイントがスキャンされます。ステルスでないものはリストアップされているので、以降の作業の参考にしてください。
- Raspberry Pi Zero Wは2.4GHz帯のWiFiにのみ対応しています。

wifi Scan

wifi Info

```

WiFi Scan Result:
SSID : Buffalo-
address : 8A:
channel : 3
frequency : 2.422 GHz (Channel 3)
quality : Quality=33/70 Signal level=-77 dBm
spec : IEEE 802.11i/WPA2 Version 1,TKIP,CCMP TKIPPSK,TKIP,CCMP TKIPPSK

SSID : IODATA-8
address : 34:
channel : 1
frequency : 2.412 GHz (Channel 1)
quality : Quality=70/70 Signal level=-39 dBm
spec : IEEE 802.11i/WPA2 Version 1,CCMP,CCMPSPK

SSID : IO-Guest
address : 36:
channel : 1
frequency : 2.412 GHz (Channel 1)
quality : Quality=70/70 Signal level=-34 dBm
spec : IEEE 802.11i/WPA2 Version 1,CCMP,CCMPSPK

SSID : Buffalo-
address : 88:
channel : 3
frequency : 2.422 GHz (Channel 3)
quality : Quality=34/70 Signal level=-76 dBm
spec : IEEE 802.11i/WPA2 Version 1,CCMP,CCMPSPK

SSID : Buffalo-
address : 8A:
channel : 3
frequency : 2.422 GHz (Channel 3)
quality : Quality=33/70 Signal level=-77 dBm
spec : IEEE 802.11i/WPA2 Version 1,TKIP,CCMP TKIPPSK,TKIP,CCMP TKIPPSK

SSID : elecom-0
address : 04:
channel : 4

```

SSID: , PASS PHRASE:

SET WiFi

Reboot

- ウィンド下部に、会場(もしくは開発場所)で提供されているWiFiアクセス情報を入力する (いずれも大文字小文字の区別があるので注意してください。)

- SSID 欄
- PASS PHRASE 欄
- `[SET WiFi]` ボタンを押す
- `[Reboot]` ボタンを押す
- これでRaspberry Pi Zeroが再起動をはじめます

- WiFi ウィンドを閉じ、ターミナルウィンドに戻る
- ターミナルウィンドの `[Close Connection]` ボタンを押す
- 30 秒ほど待つ (Raspberry Pi Zero が再起動します)
- `[Connect and Login PiZero]` ボタンを押して接続する
 - 接続ダイアログが出現⇒接続するとこれまで同様コマンドプロンプトが出現
- `[wifi panel]` ボタンを再び押す
- `[wifi Info]` ボタンを押す
 - 表示された情報をチェックします
 - wlan0: inet xxx.xxx.xxx.xxx (xxx は数字) のように IP アドレスが設定されていれば接続成功しています。

```

wlan0: inet 192.168.0.32 netmask 255.255.255.0 broadcast 192.168.0.255
wlan0 IEEE 802.11 ESSID:"IO-XXXXXXXXXX"
Mode:Managed Frequency:2.412 GHz Access Point: 36XXXXXXXXXX
Bit Rate=24 Mb/s Tx-Power=31 dBm
Retry short limit:7 RTS thr:off Fragment thr:off
Power Management:on
Link Quality=70/70 Signal level=-40 dBm
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
Tx excessive retries:0 Invalid misc:0 Missed beacon:0

Confirmed connection to chirimen.org.

Raspberry Pi's IP Address: 192.168.0.32
IPアドレス
  
```

- もしもあなたが ssh や scp (WinSCP, teraterm 等) などのツールに慣れている場合、上記のアドレスで ssh 接続できます
 - PORT: 22
 - ID: `pi`
 - PASSWORD: `raspberrypi`
- 確認できたら WiFi Setting ウィンドを閉じてください。
- 以上ですべての初期設定完了です！

Hello Real World (Lチ力を実行する)

配線

PiZero とパーツを使って下の図の通りに配線します。

- LEDの極性に注意！^{*18}
 - [LEDの説明](#)
- [ブレッドボードの使い方](#)
- [抵抗値の読み方](#)
- その他、配線の基礎知識^{*19}

注意

- 間違ったピンに差し込むと場合によってはPiZeroが停止したり故障することもあります。(たとえば3.3V端子とGND端子を接続してしまうなど。)
- そのため、慣れるまでは一旦PiZeroをシャットダウン、USBケーブルも外し電源OFFにしてから配線すると安全です
 - シャットダウンコマンド：`sudo shutdown -h now`

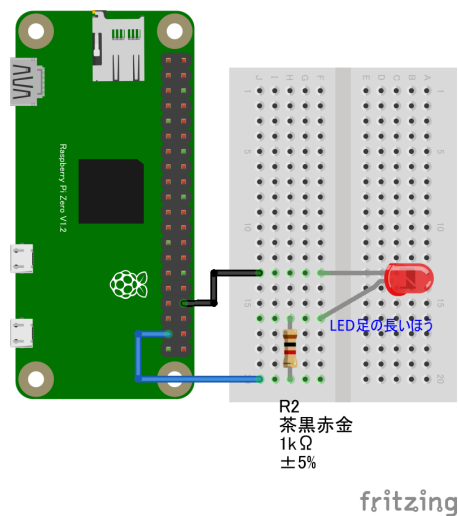


図2: PiZero 配線図

- 配線に使うケーブルの色に厳密な決まりはありませんが、一般的にGNDは黒(や黒っぽい色)、電源(VCC, +3.3V, +5V)には赤(や赤っぽい色)が用いられます。配線間違いを防ぐためにもなるべく合わせましょう。

18. <https://tutorial.chirimen.org/raspi/hellorealworld#section-1>

19. <https://tutorial.chirimen.org/reference#section-1>

- 抵抗やLEDの足(リード線)は手で簡単に曲げられます。ブレッドボードに差し込めるように適当に成型してください。
- 上図のPiZeroは上から見たものです

プログラムを書く

Raspberry Pi に接続した LED を点滅させるプログラムを書きます。

- ターミナルウィンドでRaspberry Pi Zeroに接続します。(ステップ1が完了した状態)
- myAppディレクトリに移動します。
 - コンソールの右側のファイルマネージャでmyApp⇒移動を選ぶ
 - このディレクトリが開発環境が設定されているディレクトリです。
- [Create New Text] ボタンを押す
- 入力欄に `hello.js` と入力
- [create] ボタンを押す
- JS Editorウィンドが出現

以下のプログラムをJS Editorに書き写します ～ コピペ（下記プログラム部分を選択してCTRL+c、JS Editorウィンド上でCTRL+v）

```
import {requestGPIOAccess} from "../node_modules/node-web-gpio/dist/index.js"; // We
bGPIO を使えるようにするためのライブラリをインポート
const sleep = msec => new Promise(resolve => setTimeout(resolve, msec)); // sleep 関
数を定義

async function blink() {
  const gpioAccess = await requestGPIOAccess(); // GPIO を操作する
  const port = gpioAccess.ports.get(26); // 26 番ポートを操作する

  await port.export("out"); // ポートを出力モードに設定

  // 無限ループ
  for (;;) {
    // 1秒間隔で LED が点滅します
    await port.write(1); // LEDを点灯
    await sleep(1000); // 1000 ms (1秒) 待機
    await port.write(0); // LEDを消灯
    await sleep(1000); // 1000 ms (1秒) 待機
  }
}

blink();
```

- 書き終えたら保存します。([Save] ボタン もしくはCTRL+s)
- ターミナルウィンドの右側(ファイルマネージャ)に `hello.js` が出現していることを確認します
- エディタウィンドを閉じます

実行する

- ターミナルウィンドのコンソール部(ウィンド左側)のプロンプト(画面一番下)が以下になっていることを確認します
 - `pi@raspberrypi:~/myApp$`
- コンソール部をクリックして、入力可能状態にしてから、以下の文字を入力します。
- `node hello.js` ENTERキー
 - **node** はJavaScriptのコードを実行するインタプリタ^{*20}
 - node コマンドについて^{*21}
- LED が点滅すれば完成です 🎉
- プログラムを止めるには、コンソール部で `CTRL+c` を押します。

20. <https://ja.wikipedia.org/wiki/%E3%82%A4%E3%83%B3%E3%82%BF%E3%83%97%E3%83%A4%E3%82%BF>

21. <https://atmarkit.itmedia.co.jp/ait/articles/1102/28/news105.html>

Raspberry Pi について

- 教育・学習用として設計されたシングルボードコンピュータ
 - シングルボードコンピュータ～PC・スマホとの違い^{*22}
- Linuxが動作するシングルボードコンピュータとして、安価でとても高いシェアを持ち世界中で容易に入手できる
- 今回使用する Raspberry Pi ZeroW は、その中でも特に安価(2000 円以下^{*23})で小型・低消費電力の機種、HDMI 出力はあるもののブラウザを動かすだけの処理能力がありませんが、IoT のエッジデバイス（センサーやアクチュエータが載ったデバイスでディスプレイはあるとしても簡易のもの）には適しています。
 - フルセットのブラウザが内蔵されたデバイスを作りたい場合は CHIRIMEN Raspberry Pi 版 が使用できます。
 - インターネットを経由して PC やスマホのブラウザから遠隔操作するシステムは この Pi ZeroW 版でつくれます。 [IoT の章](#) まで進めましょう。

Raspberry Pi Zero のピン配列

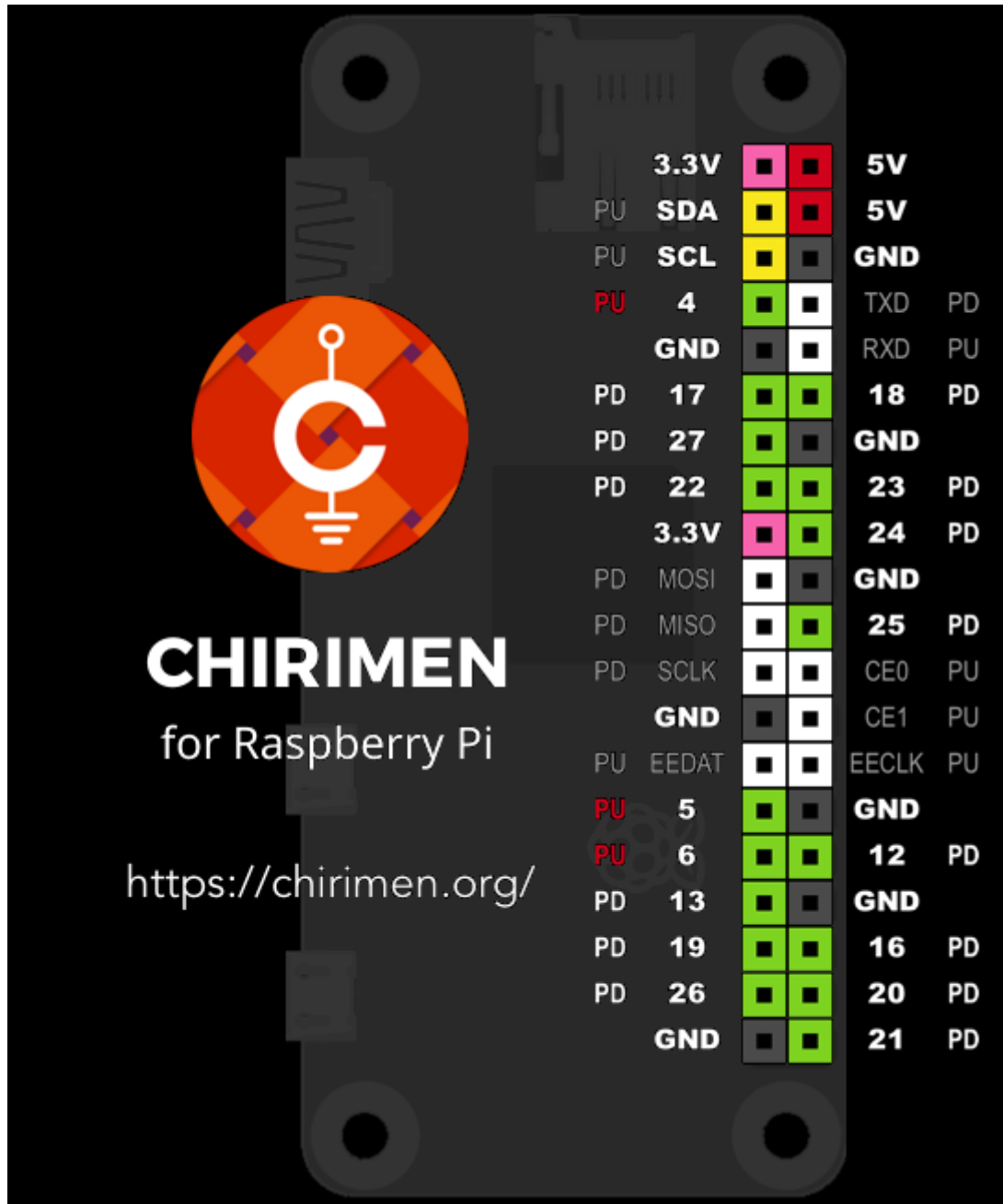
GPIO, 電源, GND, I2C 信号線などのピン配列を記載します。

- 白い文字で書かれたピンだけが使えます
- GND、3.3V、5V はそれぞれ電源とグランドです
- 数字 + PD||PU と書かれているピンは GPIO 端子(詳細は次章)
 - PD: プルダウン, PU: プルアップ

22. <https://elchika.com/dic/%E3%82%B7%E3%83%B3%E3%82%B0%E3%83%AB%E3%83%9C%E3%83%BC%E3%83%89%E3%82%B3%E3%83%B3%E3%83%94%E3%83%A5%E3%83%BC%E3%82%BF/>

23. <https://www.switch-science.com/catalog/3200/>

- SCL, SDAはI2Cインターフェースのピンです(詳細は次章)



CHIRIMEN について

- [こちらを参照ください](#)

CHIRIMEN Raspberry Pi Zero 版について

- PiZero 上では Web Browser を動かさない。
- Node.js という JavaScript インタープリターだけが動く
 - ブラウザの機能のうち一部だけが PiZero 上で使える
 - プログラミング言語 ～ JavaScript
 - 画面表示や GUI に関わらない API
 - 通信プロトコル
 - 使えないのはブラウザを使った画面表示や GUI
- ブラウザを使った画面や GUI は
 - ネットを介してスマホや PC からコントロール ⇒ これが代表的に IoT と呼ばれるデザインパターン
 - このとき、Raspberry Pi Zero は、IoT エッジデバイスとして動作

JavaScript の基礎

- [こちらを参照してください](#)
- [非同期処理 async/await](#) を多用します。

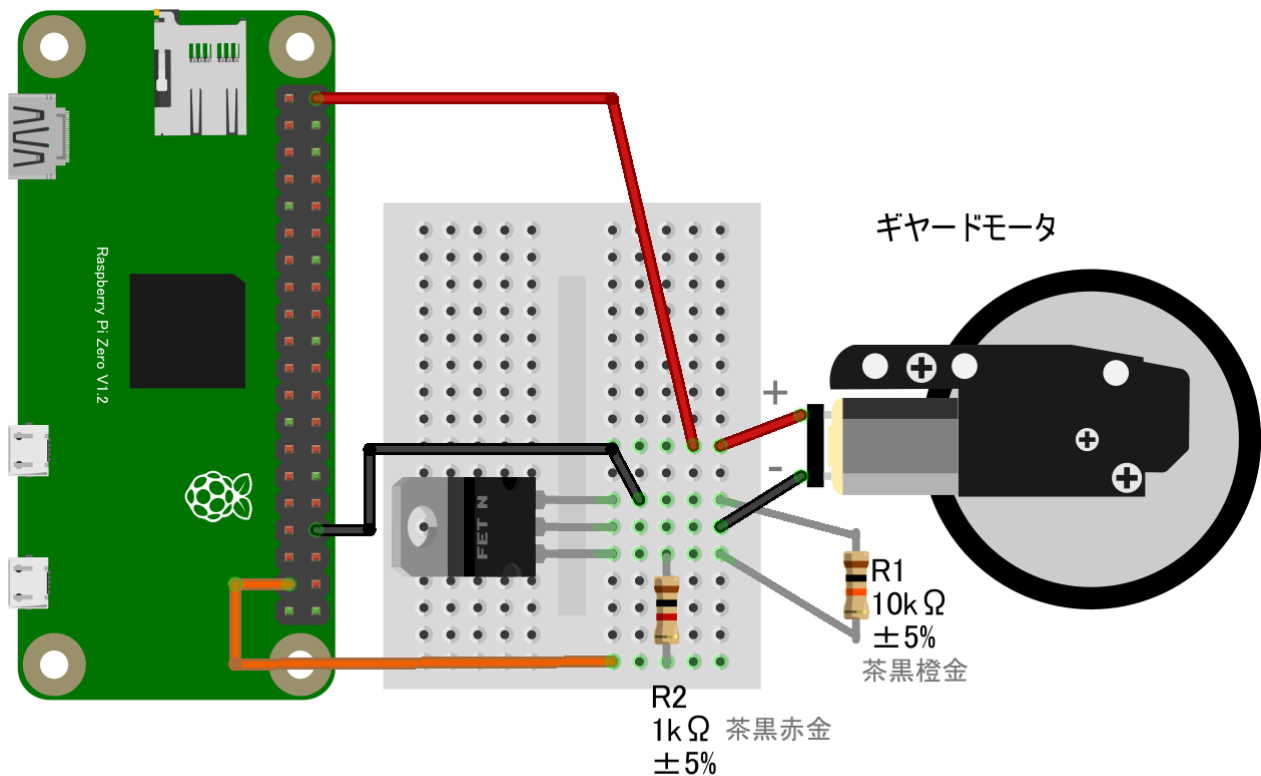
GPIOを試す

GPIOを理解する

- GPIOとは?

GPIO出力

GPIOの出力はLチカで実験済みですね。そこで今回はモーターを動かしてみよう。MOSFETを使った回路図は以下のようになります。



fritzing

コードはLチカと全く同じです。

回路について

- [MOSFETを使った大電力制御](#)

コードを読む

- 前提：CHIRIMEN Rasoberryu Pi ZeroはNode.jsをプログラム実行環境（インタープリタ）として使っています。
 - [Node.jsについて](#)
- ターミナルウィンドの右側のファイルマネージャでhello.js⇒表示 を選び、ソースコードを読んでみましょう
- WebGPIO ライブラリを読み込み（[JavaScript Module](#)仕様に従って）

```
import {requestGPIOAccess}
from "../node_modules/node-web-gpio/dist/index.js";
```

 - [JavaScript module](#) に基づいてWebGPIO ライブラリを読み込みます。これでWeb GPIO APIが使えるようになりました。
- [GPIO ポートの初期化处理](#)
- [GPIOPortの出力処理](#)

GPIO 入力

GPIO 端子の入力が変化したら関数を実行という機能によって GPIO の入力を使います。

- ターミナルウィンドの [CHIRIMEN Panel] ボタンを押す
- 出現した CHIRIMEN Panel の [Get Examples] ボタンを押す
- ID : **gpio-onchange** を探します
- 回路図リンクを押すと回路図が出てきますので、回路を組みます。
- [JS GET] ボタンを押すと、開発ディレクトリ(~/myApp)に、サンプルコードが保存されます。
 - **main-gpio-onchange.js** というファイル名で保存されます。
 - ターミナルウィンドの右側のファイルマネージャで main-gpio-onchange.js ⇒ 編集 を選びます。
 - ソースコードを見てみましょう
 - 今回は編集不要ですが、サンプルをベースに応用プログラムを作るときには編集しましょう。
- 実行する
 - ターミナルウィンドのコンソールのプロンプトが `pi@raspberrypi:~/myApp$` となっていることを確認
 - ターミナルウィンドのコンソールに、`node main-gpio-onchange.js` [ENTER] と入力して実行。
 - タクトスイッチを押してみます。
 - タクトスイッチが押されるたびにコンソール画面に **0**(押された状態)、**1**(離した状態)が交互に表示されます。
 - Note: GPIO ポート 5 は、Pull-Up(開放状態で High レベル)です。そのため離した状態で 1 が出力されます。スイッチを押すとポートが GND と接続され、Low レベルになり、0 が出力されます。
- 終了は CTRL+c

コードを読む

- ターミナルウィンドの右側のファイルマネージャで main-gpio-onchange.js ⇒ 表示 を選び、ソースコードを読みましょう
- WebGPIO ライブラリを読み込み

```
import {requestGPIOAccess} from "../node_modules/node-web-gpio/dist/index.js";
```

 - JavaScript module に基づいて WebGPIO ライブラリを読み込みます。これで Web GPIO API が使えるようになりました。
- GPIO ポートの初期化処理を行う
- onchange による入力処理

GPIO 入力(ポーリング)

入力ではイベントの他にポーリングというテクニックが広く使われます。(次章の I2C デバイスからの入力では専らポーリング)

- ターミナルウィンドの [CHIRIMEN Panel] ボタンを押す
- 出現した CHIRIMEN Panel の [Get Examples] ボタンを押す
- ID : **gpio-polling** を探します

- 回路は前章と同じなのでそのままにしておきます。
- **[JS GET]** ボタンを押すと、開発ディレクトリ(`~/myApp`)に、サンプルコードが保存されます。
 - **main-gpio-polling.js** というファイル名で保存されます。
 - ターミナルウィンドの右側のファイルマネージャで `main-gpio-polling.js` ⇒ **編集** を選びソースコードを見てみましょう
- 実行する
 - プロンプトが `pi@raspberrypi:~/myApp$` となっていることを確認
 - コンソールに、 `node main-gpio-polling.js` [ENTER] と入力して実行。
 - 0.3秒おきにポート5の値がコンソールに表示されていきます。
 - タクトスイッチを押してみます。
 - タクトスイッチが押されると、**0**に変化します。
- 終了は CTRL+c

コードを読む

- ターミナルウィンドの右側のファイルマネージャで `main-gpio-polling.js` ⇒ **表示** を選び、ソースコードを読みましょう
- WebGPIO ライブラリを読み込み

```
import {requestGPIOAccess} from './node_modules/node-web-gpio/dist/index.js';
```

 - JavaScript module に基づいて WebGPIO ライブラリを読み込みます。これで Web GPIO API が使えるようになりました。
- GPIO ポートの初期化処理を行う
- 単純入力+ポーリングによる入力処理

I2C デバイスを試す

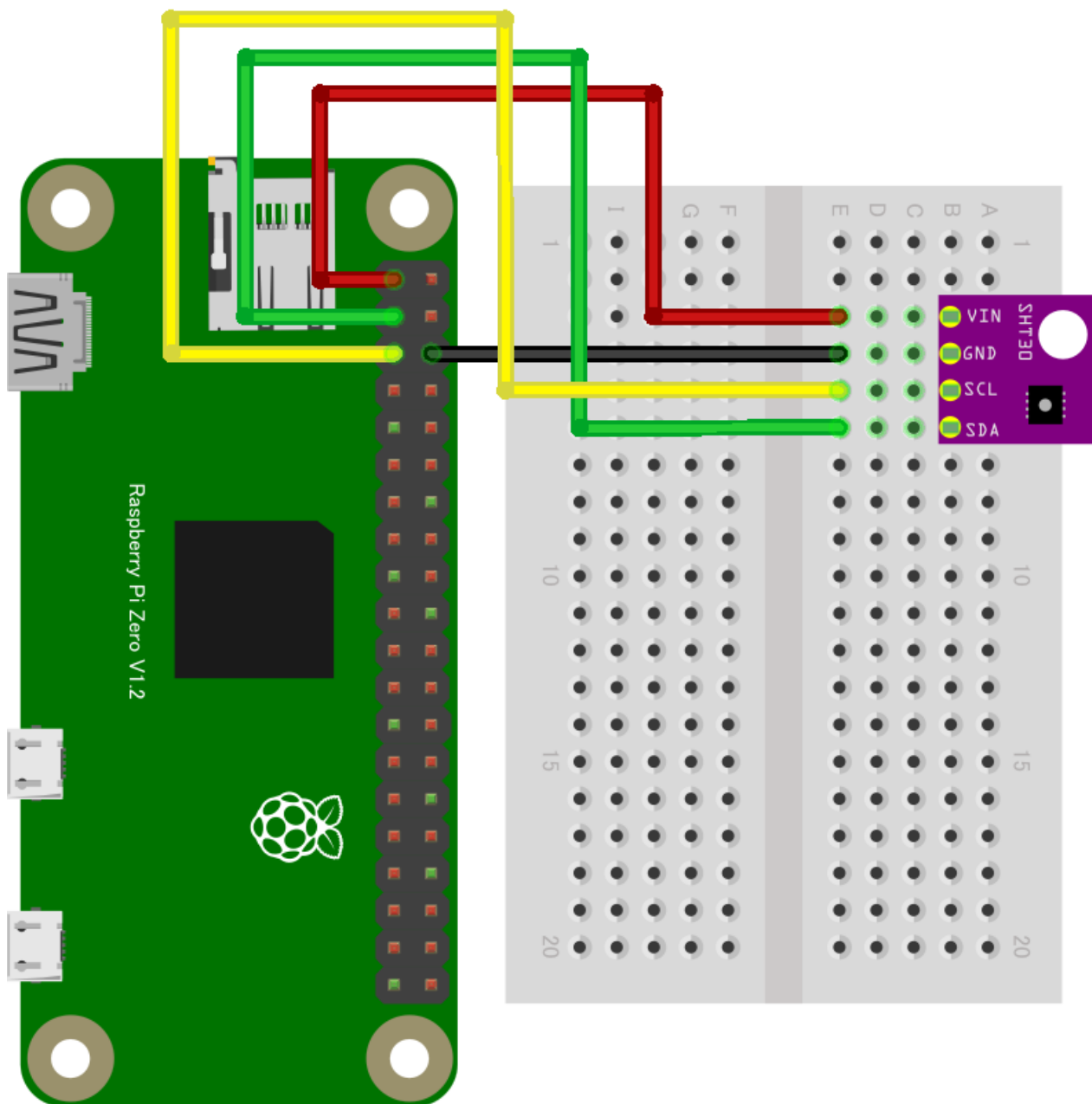
I2C を理解する

- I2C とは？

SHT30 編

SHT30 は温度に加えて湿度も測定できる I2C 接続の多機能センサーです。SHT31 もほぼ同等に使えます。(SHT31 のほうが精度が高い)

- SHT30/SHT31 について
- ターミナルウィンドの `[CHIRIMEN Panel]` ボタンを押す
- 出現した CHIRIMEN Panel の `[Get Examples]` ボタンを押す
- ID : sht30 を探します
- 回路図リンクを押すと回路図が出てきますので、回路を組みます。なお、接続は下の図のようになります。



fritzing

- [JS GET] ボタンを押すと、開発ディレクトリ(`~/myApp`)に、サンプルコードが保存されます。
 - **main-sht30.js** というファイル名で保存されます。
 - Note: ターミナルウィンドの右側のファイルマネージャで `main-sht30.js` ⇒ **編集** を選ぶと、エディタで編集できます。
 - ソースコードを見てみましょう
 - 今は編集不要ですが、サンプルをベースに応用プログラムを作るときには編集しましょう。

I2Cセンサー(SHT30)が認識されていることを確認する

- CHIRIMEN Panelの `[i2c detect]` ボタンを押すと、SHT30のI2Cアドレス^{*24} `0x44`が表示されていればうまく接続されています。

- `ic2 detect`とは

```

      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- 44 -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --

```

実行する

- ターミナルウィンドのコンソールのプロンプトが `pi@raspberrypi:~/myApp$` となっていることを確認
- ターミナルウィンドのコンソールに、`node main-sht30.js` [ENTER] と入力して実行。
- 温度と湿度が1秒ごとにコンソールに表示されます。
- 終了は CTRL+c

コードを読む

- ターミナルウィンドの右側のファイルマネージャで `main-sht30.js⇒表示` を選び、ソースコードを読んでみましょう
- WebI2CライブラリとSHT30デバイスドライバを読み込み

```

import {requestI2CAccess} from "./node_modules/node-web-i2c/index.js"; import SHT30
from "@chirimen/sht30";

```

- JavaScript module に基づいてWebI2Cライブラリを読み込みます。
- I2C 温湿度センサー (SHT30, SHT31)の初期化と使用

24. https://strawberry-linux.com/pub/Sensirion_Humidity_SHT3x_DIS_Datasheet_V3_J.pdf

ADT7410 編

温度センサー ADT7410 を使います。もし、SHT30 を使用する場合は、「IoT を試す」の章まで読み飛ばしてください。

- ターミナルウィンドの [CHIRIMEN Panel] ボタンを押す
- 出現した CHIRIMEN Panel の [Get Examples] ボタンを押す
- ID : adt7410 を探します(上から5個目ぐらい)
- 回路図リンクを押すと回路図が出てきますので、回路を組みます。なお、接続は下の図のようになります。
 - センサーの極性に注意！*25

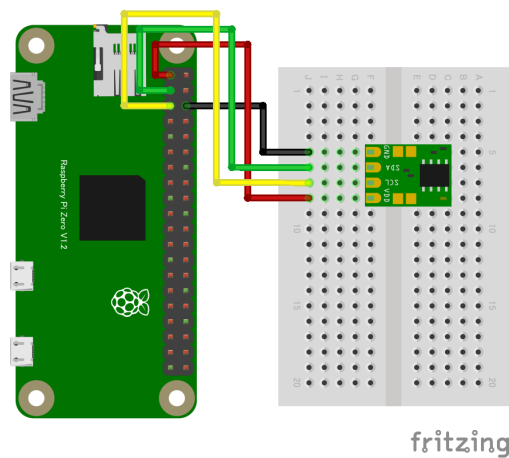


図 3: PiZero 温度センサー図

- [JS GET] ボタンを押すと、開発ディレクトリ(~/myApp)に、サンプルコードが保存されます。
 - **main-adt7410.js** というファイル名で保存されます。
 - Note: ターミナルウィンドの右側のファイルマネージャで main-adt7410.js ⇒ 編集 を選ぶと、エディタで編集できます。
 - ソースコードを見てみましょう
 - 今は編集不要ですが、サンプルをベースに応用プログラムを作るときには編集しましょう。

I2Cセンサーが認識されていることを確認する

- CHIRIMEN Panel の [i2c detect] ボタンを押すと、ADT7410 の I2C アドレス*26 0x48 が表示されていればうまく接続されています。
 - i2c detect とは*27

25. <https://tutorial.chirimen.org/raspi/hellorealworld#section-2>

26. https://akizukidenshi.com/download/ds/akizuki/AE-ADT7410_aw.pdf

27. <https://tutorial.chirimen.org/ty51822r3/i2cdetect>


```

4 5 6 7 8 9 a b c d e f
00:      -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- 48 -- -- --
50: -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- --

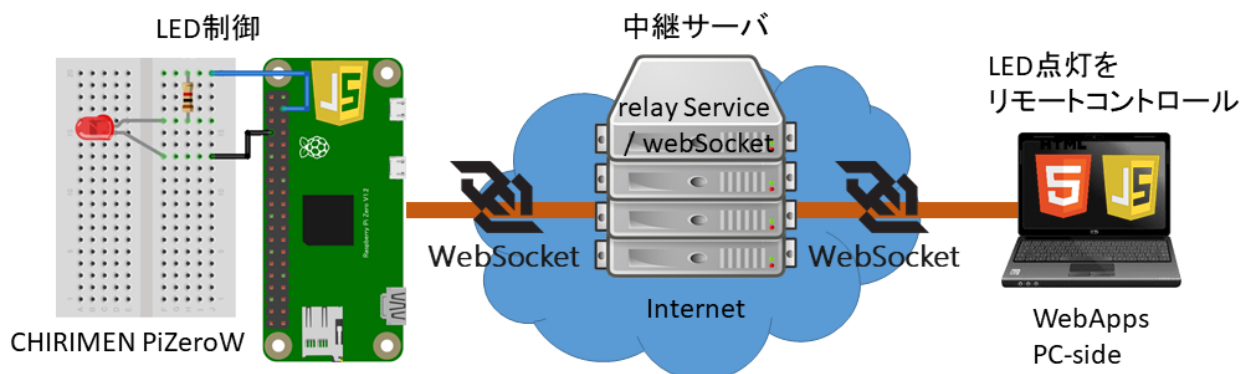
```

実行する

- ターミナルウィンドのコンソールのプロンプトが `pi@raspberrypi:~/myApp$` となっていることを確認
- ターミナルウィンドのコンソールに、`node main-adt7410.js [ENTER]` と入力して実行。
- 温度が1秒ごとにコンソールに表示されます。
- 終了は CTRL+c

IoTを試す

遠隔LEDコントロール



IoTは、制御されるデバイス（上図ではCHIRIMEN PiZeroW）と、利用者端末（上図ではWebApp PC-side）に加えて、これらの間でデータを中継するサーバ（クラウド）が必要になります。今回は Web 標準技術である WebSocket プロトコルを中継するサーバを用いてLEDを備えたCHIRIMENデバイスとスマホやPCのWebAppを繋いだIoTシステムを作ります。

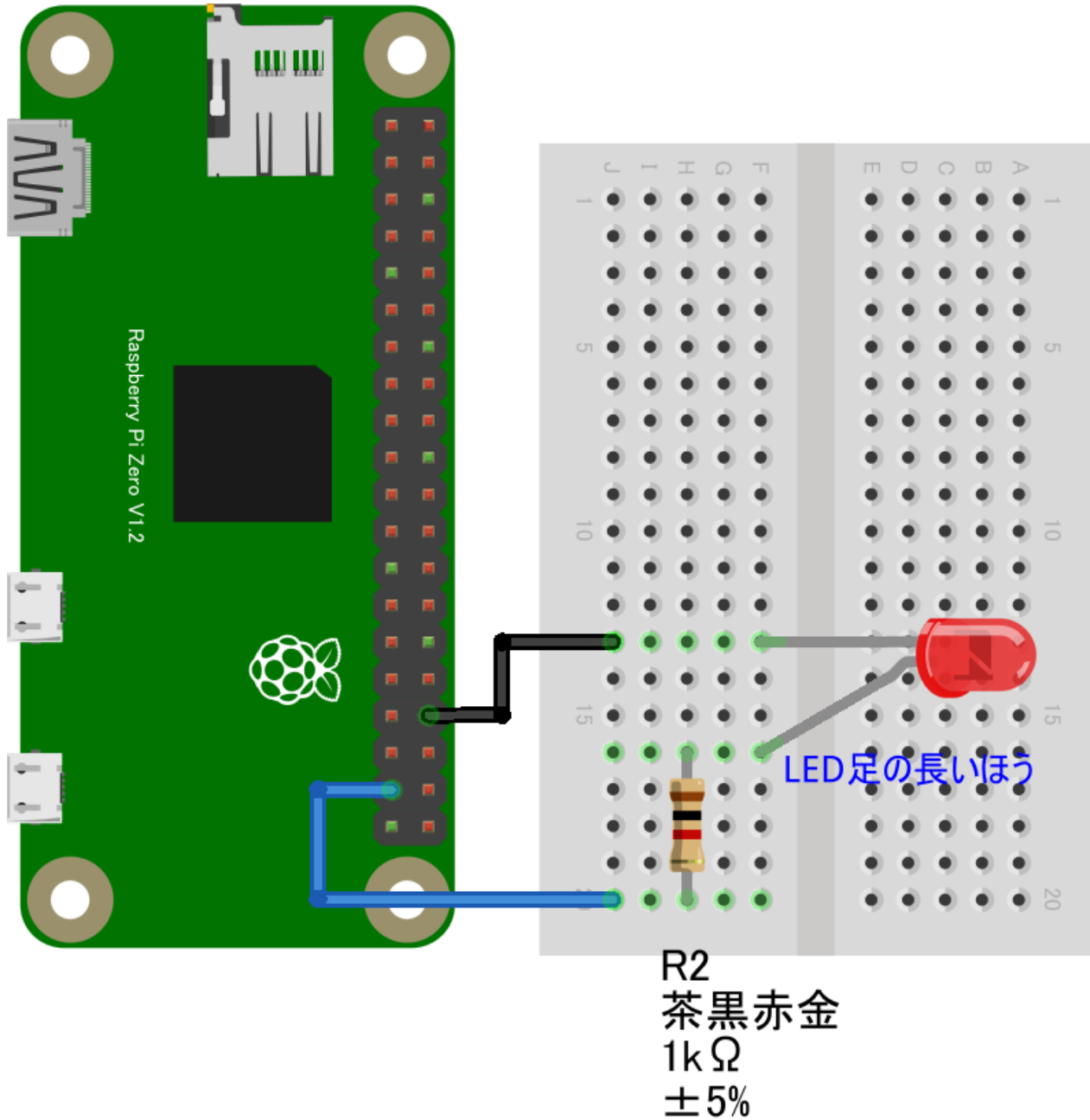
Note: モーター制御の回路を組めば、そのまま遠隔モーターコントロールができます

IoT

WebSoeketとRelayServer

配線する

配線は最初のLチカそのままです。



fritzing

CHIRIMENデバイス側にコードを入れ、実行する

- ターミナルウィンドの [CHIRIMEN Panel] ボタンを押す
- 出現したCHIRIMEN Panelの [Get Examples] ボタンを押す
- ID : **remote_gpio_led**の行を探します (もう一度この行の情報を使います)
- [JS GET] ボタンを押すと、開発ディレクトリ(~/myApp)に、サンプルコードが保存されます。
 - **main-remote_gpio_led.js**というファイル名で保存されます。
 - ターミナルウィンドの右側のファイルマネージャでmain-remote_gpio_led.js⇒編集 を選びソースコードを見てみましょう
- 実行する

- ターミナルウィンドのコンソールのプロンプトが `pi@raspberrypi:~/myApp$` となっていることを確認
- ターミナルウィンドのコンソールに、 `node main-remote_gpio_led.js` [ENTER] と入力して実行。

```
pi@raspberrypi:~/myApp$ node main-remote_gpio_led.js
object https://chirimen.org
wss://cloud.achex.ca/chirimenSocket null https://chirimen.org
achexModule:channelOpened
achex web socketリレーサービスに接続しました
```

- なお、実験が終わったら終了は CTRL+c です。

PC側のコードを準備し、実行する

- CHIRIMEN Panelに戻り、ID : **remote_gpio_led**の行にある、**CSB EDIT** リンクをクリックする。
- CodeSandbox というオンラインのWebApp 開発環境のウィンドが開き、PC 側のコードが表示されています。編集もできます。
 - 詳しい解説 : CodeSandbox ガイド^{*28}
- また、右側（もしくは右下）のフレームにはLEDを遠隔コントロールするためのwebAppが既に実行されています。
- webAppを使ってLEDが制御できることを確かめてみましょう。



図4: Code Sandbox Image

28. <https://csb-jp.github.io/>

コードを読む

Raspberry Pi Zero側コード

- ターミナルウィンドの右側のファイルマネージャでmain-remote_gpio_led.js⇒表示 を選び、ソースコードを読みましょう
- これまで通りWebGPIOライブラリの読み込み
- relayServer.jsライブラリの読み込み
 - Node.jsではrelayServerライブラリに加えて、websocketライブラリの読み込みが必要です。

```
import nodeWebSocketLib from "websocket";
import {RelayServer} from "../RelayServer.js";
```

- relayServer.jsを使って、PCからの操作指示を受信
 - 初期化
 - 受信処理(コールバック関数の設定)
- 受信した内容をもとにGPIO出力を操作してLEDを点灯・消灯

PC側コード

- CodeSandboxで開いているPC.jsを見てみましょう
- JavaScript Module仕様に基づいてrelayServer.jsを読み込み

```
import {RelayServer} from "https://chirimen.org/remote-connection/js/beta/RelayServer.js";
```

- relayServer.jsを使い、UIを通してユーザからの操作指示を送信
 - 初期化
 - 送信処理～(UI(ボタン)に設置したコールバック関数をもとに送信

自分専用チャンネルで制御

サンプルのコードは共通のチャンネルを使って制御しています。この状態では複数の人が同時に実習していると混信します。(他の人のPCでON/OFFを指示しても、自分のLEDがON/OFFする。同じチャンネルを使っているため。)

これはこれで使い道はあるのですが、自分のLEDは自分だけで制御したい場合は専用のチャンネルを使って制御しましょう。チャンネルの指定はPiZero側のコードと、PC側のコード両方を同時に同じ内容で設定する必要があり、以下の部分になります。

```
channel = await relay.subscribe("chirimenLED");
```

この `chirimenLED` という文字列(チャンネル名)を他の人と被らない別のチャンネル名に書き換えます(`chirimenLED5` など)

他のいろいろなデバイスを試してみる

- ターミナルウィンドの `[CHIRIMEN Panel]` ボタン⇒CHIRIMEN Panelの` `[Get Examples]` ボタンで出現するリストのデバイスがすぐ試せます。
- このリストの直リンクは[こちら\(サンプル一覧\)](#)です。CHIRIMEN RPiZeroをPCにつないでいないときはこちらを眺めてください。

また、こちらには、Web GPIO や Web I2C によって扱うことのできる外部デバイスの写真や様々なCHIRIMEN環境のサンプルコードの一覧があります*²⁹。こちらも参考になるかもしれません。(CHIRIMENは Raspberry Pi ZeroW以外に、Raspberry Pi 3,4や、micro:bit等でも使用できます)

常駐プログラム化する

ターミナルウィンドから node コマンドで実行指示しなくても、電源投入後 自動的に指定したコードを起動する設定(常駐プログラム化)ができます。このチュートリアルでは、forever^{*30}を使用する設定を専用GUIを用いて行ってみましょう。

- ターミナルウィンドの `[CHIRIMEN Panel]` ボタン⇒CHIRIMEN Panelの `[Resident App Conf.]` ボタンを押します。
 - 専用画面のUIが使用可能状態になるまで数秒かかります。
- 開発ディレクトリ `~/myApp` 内にある javascript コードがリストアップされます。
- 各行の `Now Running` 列は常駐状態、`App Name` はコードのファイル名、`Select` は選択用チェックボックスです。
 - `Now Running` 欄には現在常駐プログラム化しているコードに、`RUNNING` が表示されています。(常駐プログラムがなければ全部の行が空白になります)
 - `Select` 欄のチェックボックスをチェックすると、そのコードが常駐プログラム化します。(常駐プログラムは一個だけ指定できます)
 - 設定が反映され、常駐状態が確認できるようになるまで、20秒ぐらいかかります
 - 常駐状態の再確認は `[Resident App Conf.]` ボタンで可能
 - 設定できたらシャットダウンしてPCとのUSB接続も外します
 - シャットダウンコマンド: `sudo shutdown -h now`
 - その後PiZeroをモバイルバッテリーなどにつないで独立して稼働させます。
 - PiZeroの緑色LEDの点滅が収まると、概ね常駐プログラムが起動
 - その後PCからリモートコントロールしてみましょう
 - PCに接続しなおして、一番上の `STOP ALL APPS` のチェックボックスをチェックすると、常駐プログラムを解除できます。

29. <https://tutorial.chirimen.org/raspi/partslist>

30. <https://www.npmjs.com/package/forever>

- Note: 常駐化のツールとしては、他にも *systemd service unit*, *openrc*, *cron*, *pm2*, *forever* 等があります。Web でそれぞれの特徴を調べて用途に合ったものを選択して設定しても良いでしょう。

予備知識

CHIRIMEN for Raspberry Pi を利用するに際して、知っておくと良い予備知識やツールの使い方が学べるドキュメントです。

- GitHub ハンズオン^{*31}
 - GitHub の基本的な使い方の分かるハンズオン資料です。
- CodeSandbox ガイド^{*32}
 - ブラウザ上で開発する CodeSandbox の使い方を確認しましょう。
- JavaScript 初学者向け資料集
 - JavaScript 1 Day 講習資料、JavaScript 本格入門書、チートシートなどはこちら

その他、電子工作など一般的な知識は 予備知識・資料集 や、共通資料集を参照してください。

31. <https://github.com/webiotmakers/github-handson>

32. <https://csb-jp.github.io/>

CHIRIMEN ブラウザー版との差異

CHIRIMEN ブラウザー版	Node.js
ライブラリ、ドライバーはhtmlで読み込む	jsの中で直接読み込む
<code><script src="polyfill.js"></code> <code></script ></code>	<code>import {requestGPIOAccess} from</code> <code>"/node_modules/node-web-gpio/dist/index.js";</code> <code>import {requestI2CAccess} from</code> <code>"/node_modules/node-web-i2c/index.js";</code>
<code><script src=".../adt7410.js"></code> <code></script ></code>	<code>import ADT7410 from "@chirimen/adt7410";</code>
—	Sleep関数を宣言する
—	<code>const sleep = msec => new Promise(resolve =></code> <code>setTimeout(resolve, msec));</code>

CHIRIMEN 環境の任意のディレクトリへのセットアップ

以下のコマンド手順で~/myAppディレクトリ以外にも設定できます。

- `mkdir` [自分用の作業ディレクトリ] (`mkdir`^{*33} コマンドとは)
- `cd` [自分用の作業ディレクトリ] (`cd`^{*34} コマンドとは)
- `wget https://tutorial.chirimen.org/pizero/package.json` (`wget`^{*35} コマンドとは)
- `wget https://chirimen.org/remote-connection/js/beta/RelayServer.js` (RelayServer.js^{*36}を使う場合)
- `npm install` (`npm`^{*37} とは)

33. <https://atmarkit.itmedia.co.jp/ait/articles/1606/07/news015.html>

34. <https://atmarkit.itmedia.co.jp/ait/articles/1712/14/news021.html>

35. <https://atmarkit.itmedia.co.jp/ait/articles/1606/20/news024.html>

36. <https://chirimen.org/remote-connection/>

37. <https://atmarkit.itmedia.co.jp/ait/articles/1606/17/news030.html>