# 概要

## このドキュメントの目的

ラズパイで出来る事を，あっさりと箇条書き風に残す。

または，特定の用途に限定した設定方法の具体例を記述する。

## 特徴

なお、PiはPythonの意らしく「Pythonを使って簡単にマイコンを動かす」がコンセプトらしい。

類似マイコンのArduinoと比較する事で，逆にラズパイの特徴が浮かび上がる気がする。

Arduinoは純粋なマイコンに近く，制御寄りでPLCとかに近い感じ。

以下にArduinoの特徴を示す：

・開発用には別途PCが必要。Arduino IDEを使う（Win/Mac/Linux対応）

・Arduino専用のC++に近い言語を使う必要がある。逆に、サンプルコードを読むときに迷わないのはメリット。

・シールド、と呼ばれるセンサ類が豊富に販売されている。

・GPIOは1～17MHz。(0.58～1μSec)。ラズパイはmSec単位（数kHz？情報が少ない）

・高速信号に対応しているので、PWM、サーボ運転すら可能。

・基本的にシングルタスク。

・シングルタスクである事もあって、Ether機能はラズパイに比べると弱い。もしくは高値。

・起動は一瞬。

## ラズパイのブート

ラズパイのブートは以下の様な流れになっているらしい

１．GPUが起動．（このときCPUはオフ、SDRAMは無効

２．GPUがSoC内のROMから第1段ブートローダを実行（ラズパイ製造時にSoC内のROMに書き込まれている。）※Soc；System-on-a-chip　CPUなどを１枚のチップにまとめたものの事。

３．SDカードから第２段ブートローダ /boot/bootcode.bin　を読み込む。

４．SDRAMを有効化。SDカードからGPUファームウェア(start.elf)を読み，SDRAMにロード．

GPUファームウェアにジャンプ．

５．GPUファームウェアはSDカードから，/boot/config.txtを読み，CPUの設定を行う

SDカードから，カーネルイメージ(kernel.img)を読み，SDRAMにロード．CPUのリセット信号を有効にする．

６．CPUがSDRAMに置かれたプログラムを実行する．（CPUからGPUに処理が移る）

補足

ラズパイにはBIOSが無く、boot\config.txt で様々な設定を行っている。

参考

<https://elinux.org/RPi_U-Boot>（英語）

強制終了

Ctrl＋Alt＋F2

## バックアップの取り方（DD）

Raspbianのデスクトップ版にはSD Card Copierというパッケージがあるが，CentOSなど他のものにはない。

初心者にはハードルこそ高いが，DDでのバックアップだと .img ファイルをPCで保管できるので便利。

なお，DDを使う為に別のLinux機（Virtual Boxの利用などでも可）を用意する必要がある。

ここでは，SDカードのデバイス名はsdbとし，以下はUbuntuでの操作とする。（多分Raspbianでも同じ）

前準備：

lsblk | grep “sd.” # デバイス名の確認，かつSDカードの情報を確認。（パイプ以降は必須ではない）

sudo umount /dev/sdb2 # 先にアンマウントして置く。（GUIで操作しても良い。）

ファイルサイズの最小限化：

必須ではないが，これをやらないと書き込みに時間がかかってしまうし，保管先のストレージが圧迫される。どちらにせよ同じ16GBのSDカードでも容量が若干異なる事があるので，やっておいた方が良い。

sudo e2fsck -p -f */dev/sda2* # 不整合や破断がないかどうか確認（resize2fsの為に必要）

sudo resize2fs -P */dev/sdb2* # 必要なファイルシステムの大きさを見積もる

> Estimated minimum size of the filesystem: **962363** # 単位はブロック(4kb)

# 962363 [blocks] × 4 = 3951536[kb] ÷ 1024(kb → Mb) = 3858.92[Mb]

# 3858.92 + 100M（バッファ）= 3958.92

⇒ resize2fsでファイルシステムの大きさを変える時は繰り上げた値 **3959**M を使う。

sudo resize2fs -p */dev/sdb2* **3959M** # ファイルシステムのサイズを一時的に小さくする。

# resize2fs では実数（小数点）が使えない点に注意

パーティションのリサイズ：

sudo fdisk　/dev/sdb # fdisk の対話モードへ入る。　※sdb2ではない事に注意

Command (m for help): p # 詳細情報を表示：単位はブロック[=4kb] である点に注意。

>　Device Boot Start End Blocks Size Id System

>　/dev/sdb1 8192 96042 87851 42.9M c W95 FAT32 (LBA)

>　/dev/sdb2 **98304** 60751871 60653568 28.9G 83 Linux

Command (m for help): d → 2 # パーティション番号2を削除

Command (m for help): n　→ p → 2 # primary のパーティション２を再作成（リサイズ）

※first sector は pコマンドで確認したStart の値を使う：この場合は98304

※last sector は 先ほどresize2fs で指定した値よりも大きめに：この場合 +4.2G　など

※ext4 signature は削除しない [N] （意味は分からないが20.07.13）

Command (m for help): p # サイズ変更を確認。また，最終セクタ数を控える。

>　/dev/sdb2 98304 **8896511** 8798208 4.2G 83 Linux

Command (m for help): w # 書き込み

sudo resize2fs /dev/sdb2 # ファイルシステムを再拡張。（サイズ指定が無いので最大まで拡張される）

～バックアップ 続き

DDでバックアップ

# DDで用いるサイズ数の計算

# まずはバイト数に直す。セクタサイズは通常512b なので (8896511+ 1) × 512 = 4555014144b

# bs (block size) とcount を調整しつつ設定する。めんどくさければbs=512にしてセクタサイズをそのまま使う。

#　バックアップを取る。（必要時間は約4min/1Gb） 補足：4555014144b ÷ 16777216[=16Mb] = 271.5

sudo dd if=/dev/sdb of=./raspi-backup.img bs=16M　count=272 status=progress

書き込み（復旧）

ここでは，バックアップ先のSDカードのデバイス名はsdcとする。

# 書き込む

sudo dd if=./raspi-backup.img of=/dev/sdc bs=1M　status=progress # bsの指定は必須ではない

# 普段はxzで圧縮しておき，.xzファイルを解凍しながら書き込む場合

xzcat rspi.img.xz | sudo dd of=/dev/sdc bs=1M # bsの指定が無いと恐ろしく遅くなる事がある。

# ダウンロードした .zipファイルを解凍しながら書き込む場合

unzip -p rspi.img.zip | sudo dd of=/dev/sdc bs=1M # bsの指定が無いと恐ろしく遅くなる事がある。

# パーティション，ファイルシステムを拡張（元に戻す）

# raspi-config でExpand Filesystem SDができるようであれば，そちらの方が楽

sudo fdisk　/dev/sdc # fdisk を使ってパーティションをリサイズ

Command (m for help):　p # 第２パーティションの**開始セクタを控え**ておく

Command (m for help): d → 2 ; n　→ p → 2 # 上の例を参考。最終セクタまで拡張しておく。

Command (m for help): w # 書き込み

sudo resize2fs /dev/sdc2 # ファイルシステムを再拡張。（サイズ指定が無いので最大まで拡張される）

補足

パーティションサイズの変更は partedでもできるという情報があるが，partedはバージョン変化の結果相対指定などができなくなり，むしろめんどくさい。GPartedが使える環境ならそれが一番楽。

参考）<https://qiita.com/narista/items/2812e6f596f956bc8e89>

## バックアップの取り方（アプリ）

Win32DiskImager

SDカードを差し込んでRead を押す。

問題点：メーカーごとの僅かな容量の違いに対応できない。

# 本体の仕様

## 製品のまとめ

### まとめ

|  |  |  |  |
| --- | --- | --- | --- |
|  | 特徴 | 値段 | 備考 |
| RaspberryPi Pico | 完全なるマイコン | ￥800 | WifiとBluetoothなし |
| RaspberryPi400 | キーボード型 | ￥10,000 | 2021年春発売 |
| RaspberryPi4 | PC並みに高性能 |  |  |
| RaspberryPi3 | 高性能 |  |  |
| RaspberryPi2 | 省エネ仕様 |  | 無線が無い点に注意！ |
| Raspberry Zero | 小型 |  |  |

参考：[https://en.wikipedia.org/wiki/Raspberry\_Pi#Specifications](https://en.wikipedia.org/wiki/Raspberry_Pi" \l "Specifications)

### 3 Model B+：

CPU: Broadcom BCM2837 1.2GHz 64-bit quad-core Arm v8 Cortex-A53

メモリ: 1GB

・ethernet（LAN）は100Mbps までしか対応していない。

・ラズパイのmicroSD書き速度が25Mbps らしいので，それ以上の速度のカードはあまり意味がない。

高価な高速カードを１枚買うよりは，安価でそこそこの速度のカードを２枚買った方がよっぽど有益。

（参考）[SD Card Copier](../../../../../../C:/data/info/%E3%82%BD%E3%83%95%E3%83%88%E3%82%A6%E3%82%A7%E3%82%A2%E7%B3%BB/%E3%83%9E%E3%82%A4%E3%82%B3%E3%83%B3/Raspbian.docx" \l "_SD_Card_Copier)

技適に関して：

日本で購入したElement14は技適認証マークが印刷，RSはシール。

### 4 Model Bの仕様：

CPU: Broadcom BCM2711 1.5GHz Arm v8 Cortex-A72

メモリ: 1,2,4,8GB (モデルによって異なる)

その他

・ethernet は100Mbps までしか対応していない。

・日本で購入したElement14は技適認証マークが印刷、RSはシール。

・以前のラズパイはARMv7(32bit)だったらしい。v8はArm64とも呼ばれる。

・電源はラズパイ4ならば最低でも3A，できれば4A欲しい。

### Zero

|  |  |  |  |
| --- | --- | --- | --- |
|  | Zero | W/WH | 2 W |
| 値段 | $5 | $10 | $15 |
| SoC | BCM2835  ARM1176JZF-S @ 1,0 GHz  (ARMv6Z 32-bit)  <https://gadgetversus.com/processor/broadcom-bcm2835-specs/> | | BCM2710A1  Cortex-A53 @ 1 GHz  (ARMv8-A 64/32-bit)  <https://gadgetversus.com/processor/broadcom-bcm2710a1-specs/> |
| RAM | 512MB | | |
| 特徴 | wifiなし |  |  |

### Pico

Pico W

Pico H

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pico W | Pico H | Pico WH |
| 値段 | $6 | $5 | $7 |
| wifi | 有り | なし | 有り |
| GPIOヘッダ | なし | 有り | 有り |
| JTAG | なし | 有り | 有り |

https://raspberry-pi.ksyic.com/news/page/nwp.id/108/

### 資料

Element14

ラズパイにはRS Component(OKdo)が製造しているものと，英国企業Premier Farnelが製造しているもの２種類が存在する。後者のものをElement14と呼び，少し値段が安い？。

（参考）<https://export.farnell.com/>

## ベンチマーク

UnixBench でのベンチ

git clone https://github.com/kdlucas/byte-unixbench cd byte-unixbench/UnixBench ./RunBash

Copy

もしくはi-nex （CPU-Z のLinux版）を使う手もある

sudo apt install i-nex

ベンチマークの結果：

CPU-Z でのベンチデータ

|  |  |  |
| --- | --- | --- |
| CPU | Single | Multi |
| Intel Core・i7-8700K 3.7GHz（会社のデスクトップ） | 441.5 | 3210.2 |
| 2回目 | 478.6 | 3825.4 |

UnixBenchの結果（ラズパイ）

|  |  |  |
| --- | --- | --- |
| CPU | Single | Multi |
| Broadcom BCM2837 1.2GHz quad-core ARMv8 | 275.4 | 478.6 |

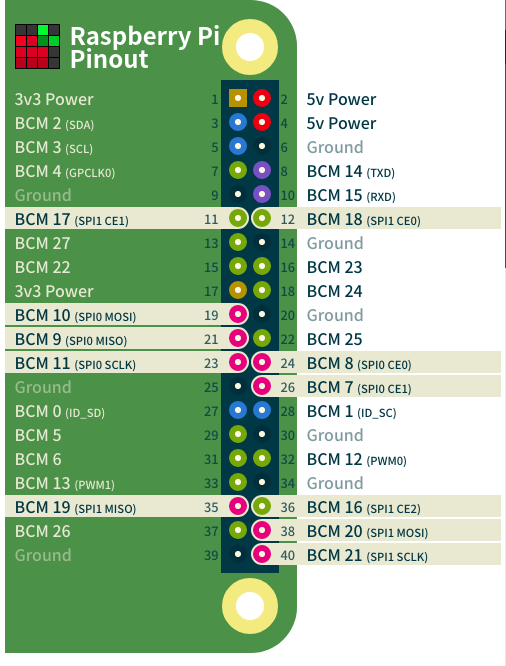
Passmark

Cinebench R15(CR15) シングル性能とマルチ性能の2種類スコアが出せる。

## GPIO

### 概要

・ラズパイのIO は3.3V。5V/24Vの入出力は通常、フォトカプラを用いる。



・ピッチ 2.54mm； 数 2列×20

・予め入出力(IN/OUT)　は割り振られているが、設定で変更する事もできる。

・1ピンあたり最大16mA。

・同時に流せる電流量合計は50mAまで。

（ここが見やすい）<https://pinout.xyz/pinout>

|  |  |
| --- | --- |
| SCL, SDA | I2C(Inter-Integrated Circuit)シリアル通信。1.8kΩのプルアップ抵抗。 |
| GPCLK0 | General Purpose Clock。ピン固定の周波数を出力。 |
| ID\_SD, ID\_SC | EEPROM(≒フラッシュメモリ) 用の端子。実際には拡張ボード（HAT）で使われる。 |
| TXD, RXD | UART(RS232C, RS422, RS485など)で使う。 |
| PCM\_DIN，PCM\_DOUT | PCM（音楽などのアナログ信号）の入出力。 |
| MOSI, MISO, SCLK  CE0, CE1 | SPI(Serial Peripheral Interface)シリアル通信。 |

### 制御モジュール

#### 概要

|  |  |
| --- | --- |
| モジュール | 詳細 |
| WiringPi | 一昔前の標準GPIOモジュール。python, C/C++ で使用可能。  最初から入っている事が多い。（raspberryPiOS-liteだと入っていない。）  Arduinoの書き方に似ているらしい  割込み非対応？  WenIOPiというリモート制御ができるとか。 |
| pigpio | 近年の標準GPIOモジュール。時間管理や割込みなどが優秀。  サービス（pigpiod）の起動が必要なのが特徴的。  関連してirrp.pyという赤外線制御モジュールがある。 |
| RPi.GPIO | pythonにのみ対応  割込み対応 |
| python-periphery | ラズパイに限らず、ワンボードコンピューター全般で使われている。  ちょっと使いにくい。 |
| gpiozero | python用のライブラリ。 |

まとめ

|  |  |  |  |
| --- | --- | --- | --- |
|  | pigpio | Rpi.GPIO | wiringPi |
| 概要 | pigpiodと言うサービスを用いる。 | python専用のモジュール | 一昔前の標準 |
| 言語 | python, C/C++ | python | bash, python, C/C++ |
| インストール方法 | apt install pigpio | pip install rpi.gpio |  |
| メリット | 時間管理が優秀 |  |  |
| デメリット |  | ハードPWMが使えない |  |
| 割込み | 対応 | 対応 |  |
| その他 |  | python-devも必要？ |  |

#### pigpio

初期設定

[ terminal ]

sudo apt install pigpio python3-pigpio# インストール(OSがlite版だと入っていない)

sudo systemctl enable pigpiod.service # pigpiod の自動起動の設定

sudo systemctl start pigpiod # pigpiod サービスを開始

pip install -U pigpio # venv を使うときは仮想環境側でこれが必要

使用方法

[ terminal ]

pigs m 17 w # 17番ピンを出力に。

pigs pud 4 d # 4番ピンをpull-downに。pull-upはu

pigs w 18 0 # 18番ピンをLOW(=0)に。HIGHは1。読み込みは r

pigs s 17 500 # 17番ピンに500mSecのパルス出力(500~2500)

pigs p 17 128 # 17番ピンを50%(128/255)のPWM出力に

echo 'w 17 0' > /dev/pigpio # こんな方法もある。２つ以上の設定を１行で可能。

（情報元）<http://abyz.me.uk/rpi/pigpio/pigs.html>

python

import pigpio

pi = pigpio.pi()

pi.set\_mode(17, pigpio.INPUT) # 17番ピンを出力に

# mode: INPUT, OUTPUT, ALT0, ALT1, ALT2, ALT3, ALT4, ALT5

pi.set\_pull\_up\_down(4, pi.PUD\_DOWM) # 4番ピンをpull-downに。pull-upはpigpio.PUD\_UP

pi.write(18, 0) # 18番ピンをLOW(=0)に

if pi.read(17) == 1 # 入力値判断

pi.hardware\_PWM(17, 50, 500000) # (gpio:ピン番号，PWMfreq:周波数，PWMduty:duty比)

# PWMfreq: 1-125M

# PWMduty: 0~1000000 つまり 500000 はduty比 0.5=50%

pi.hardware\_PWM(17, 50, 7.25) # PWMduty はfloatで与えると％指定できる

pi.set\_PWM\_frequency(17, 7.25) # PWM 周波数の変更 2021-06-23 未確認

pi.set\_PWM\_dutycycle (17, 50) # PWM duty比の変更 2021-06-23 未確認

pi.pwmSetRange(1024) # pwmRange の変更 2021-06-23 未確認

pi.pwmSetClock(375) # pwmClockの変更 2021-06-23 未確認

def pi\_cb\_func (gpio, level, tick): # コールバックメソッド

print (gpio, level, tick)

cb1 = pi.callback(4, pigpio.FALLING\_EDGE, pi\_cb\_func) # 4番立下りコールバック

cb2 = pi.callback(17, pigpio.EITHER\_EDGE) # 17番コールバック

（情報元）<http://abyz.me.uk/rpi/pigpio/python.html>

次ページに続く

～pigpio　続き

[ pigpio/pigpio.h ]

#define RISING\_EDGE 0

#define FALLING\_EDGE 1

i2c関連

（参考）http://abyz.me.uk/rpi/pigpio/pdif2.html#i2c\_open

i2c\_open

構文：int (int pi, unsigned i2c\_bus, unsigned i2c\_addr, unsigned i2c=NULL\_flags=NULL)

オープンに成功するとI2C通信のハンドルが返る

全部のピンの状態確認（例）

pi = pigpio.pi()

gpio\_pins = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]

print(" GPIO# | Mode | Value ")

print("------+-------+-------")

for pin in gpio\_pins:

mode = pi.get\_mode(pin)

value = pi.read(pin)

# モードの変換 (数字を文字列に)

if mode == pigpio.INPUT:

mode\_str = "IN"

elif mode == pigpio.OUTPUT:

mode\_str = "OUT"

else:

mode\_str = "UNKNOWN"

print(f" {pin:4} | {mode\_str:5} | {value}")

pi.stop()

トラブルシューティング

'no permission to update GPIO'

・ピンを指定する時の番号にGPIO番号(0-27)を指定しているか

#### RPi.GPIO

pip3 install RPi.GPIO # aptかpipで入れるらしい

※python用のライブラリな為，bashでは使えない

python

import Rpi.GPIO as GPIO

GPIO.setmode(GPIO.BCM) # ピン番号をBCM番号で指定する　GPIO.BOARDボードピン番号

GPIO.setup(17, GPIO.OUT) # 17番ピンを出力に　入力はGPIO.IN

GPIO.setup(4, GPIO.IN, pull\_up\_down=GPIO.PUD\_DOWN) # 4番ピンをpull-downに

GPIO.output(18, GPIO.LOW) # GPIO 18(pin12)をLOWに。

if GPIO.input(17) == GPIO.HIGH: # 入力値判断。

pwm = GPIO.PWM(4,50) # GPIO 4(pin7) を50Hz のPWMに設定

def GPIO\_cb\_func(gpio): # コールバックメソッド

print(gpio)

GPIO.add\_event\_detect(4, GPIO.RISING, bouncetime=1000) # 割り込み登録 RISING,FALLING, BOTH

GPIO.add\_event\_callback(4, pi\_cb\_func) # コールバック登録

GPIO.wait\_for\_edge(4, GPIO.FALLING, timeout=50) # 立上がり（下がり）待機

GPIO.cleanup() # 設定の初期化

補足

GPIO.add\_event\_detect() bouncetime：信号検知後，この時間[mSec]だけ次の信号を無効化する。

GPIO.wait\_for\_edge() timeout：指定した時間[mSec]だけGPIOの入力を待つ

#### WiringPi

初期設定

apt install WiringPi # 2021-05-20未確認 　gitから入れる情報が多いが？

pip install wiringpi # 2021-05-20未確認 　pythonで使う場合

bash

gpio readall # 現在のGPIO のマップが見られる。

# BCM gpioを使うときの番号　wiringpiではこの番号を使う

# wPi ラズパイ内の番号規約で付けられた番号（何に使う？）

gpio -g mode 17 out # 17番ピンを出力に。他にはin, pwm などがある。

gpio -g write 18 0 # 18番ピンをLOW(=0)に

gpio -g read 17 # 入力値判断

gpio pwm-ms # pwmの出力モードをmark:spaceにする

gpio pwmc 375 # pwmClock（分周） の変更。ラズパイ分周 19.2M/375= PWM分周 51.2kHz

gpio pwmr 1024 # pwmRange の変更。1パルスあたりのPWMクロック数。

gpio -g pwm 18 512 # 18番ピンからPWM出力。512/1024 = 50%duty

python

import wiringpi as wp

wp.wiringPiSetup()

wp.pinMode(17, wp.GPIO.OUTPUT) # 17番ピンを出力(=1)に。

# 他には PWM\_OUTPUT, INPUT　などがある

wp.digitalWrite(18, wp.GPIO.LOW) # 18番ピンをLOW(=0) に。HIGHはwp.GPIO.HIGH(=1)

if wp.digitalRead (17) == 1 # 入力値判断

wp.pinMode(17, wp.GPIO.PWM\_OUTPUT )

wp.pwmSetMode(wp.GPIO.PWM\_MODE\_MS) # pwmの出力モードをmark:spaceにする

（情報元）<https://pypi.org/project/wiringpi/>

C言語

#include <wiringPi.h>

digitalWrite(18, LOW); # 18番ピンをLOW(=0) に。HIGHはHIGH

#### その他

割込みの禁止<https://qiita.com/yagshi/items/8803400e419cf8512162>

### PWM

#### ハードウェア PWM

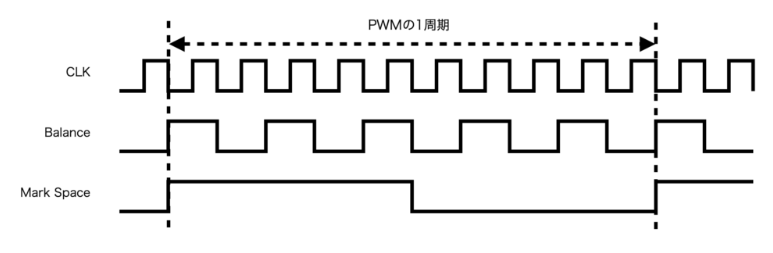
PWM Channel 0 : GPIO12、GPIO18 PWM Channel 1 : GPIO13、GPIO19

※ペアになっているPWMは同じタイマーを共有している為，同じ周期になってしまう。

BalanceモードとMark Spaceモード

balance 1周期内のON/OFFする部分のバランスを取り、より滑らかな出力特性を得る

mark:space 1周期内のON/OFFする部分を、それぞれまとめて出力。マイクロサーボなどはこちら。



#### 別途PWM制御基板を用いると、GPIOの電源不足問題を避けられる。

PCA9685　<http://akizukidenshi.com/catalog/g/gK-10350/>

### シリアル通信

手順の概要

・シリアルコンソールを無効化する

I5 Serial Port

・Bluetoothデバイスを無効化（2023-10-29これなくても動くことがあったので不要かも）

[ /boot/config.txt ]

dtoverlay=disable-bt <-- 追記

> 再起動

資料

/dev/serial0、 /dev/serial1 は/dev/ttyS0 か/dev/ttyAMA0どちらかへのシンボリックリンク

Raspberry Pi Zero, 1, 2, 3には２つのUARTがある（ラズパイ4は6つある）

名前 タイプ

UART0 PL011

UART1 miniUART

公式：

https://github.com/raspberrypi/documentation/blob/develop/documentation/asciidoc/computers/configuration/uart.adoc

参考：

https://www.ingenious.jp/articles/howto/raspberry-pi-howto/gpio-uart/

memo

こんな方法もあるらしい

buf = [ 0x01, 0xFF, 0x02, 0xFE, 0x03]

for b in buf:

a = struct.pack( "B", b )

ser.write(a)

ser.flush()

pigpioを使った方法

ser = pi.serial\_open('/dev/ttyS0', 9600)

pi.serial\_write(ser, bytearray([0xC1,0x00,0x02]))

time.sleep(0.5)

read\_bytes = pi.serial\_read(ser)

## 関連製品

### pico

#### 概要

純粋なマイコン機。OSなし、無線(wifi, Bluetooth) なし、など従来のラズパイとはコンセプトが異なる。

ラズパイでラズパイ(pico) を開発する，がコンセプト。（実際にはPCかラズパイ4以外では厳しい）

プログラムは.uf2ファイルにコンパイルして、フラッシュメモリとしてusb接続しpicoに転送する。

・.uf2ファイルはフラッシュメモリとしてのpicoディレクトリに直接ドラッグ&ドロップする。

・.uf2ファイルを書き込むと，picoは自動的にアンマウントされ，プログラムを実行する。

・プログラムを書込み後にpicoとPCを再接続したい場合は，**BOOTSELボタンを押したまま**usb接続する。

⇒ 一番簡単なのはラズパイでThonnyを使う方法

必要な物：

コードエディタ PCならVSCode、ラズパイ4ならThonnyを使うのが良いらしい。

python3.9 microPythonで書き込む場合。（C言語を使う場合は不要）

pico/stdlib.h ラズパイpico用のライブラリ。

pico\_sdk <https://github.com/raspberrypi/pico-sdk>

pico\_sdkは使用方法がいくつかある

・プロジェクト・ディレクトリに直，完全ローカルで持つ

・固定場所に格納し、なんらかの方法で $PICO\_SDK\_PATH にPATHを通す。

・git submodule として用いる 2021-06-09 未

・cmake時に自動でダウンロードさせる 2021-06-09 未

サンプルコード <https://github.com/raspberrypi/pico-examples>

（補足）結局はこのpico-examples のコードをにらめっこする事になりそう

メモ書き（2021-06-03）

※公式チュートリアル getting-started-with-pico.pdfから抜粋（かなり省略しているので注意）

/pico/pico-examples$ makedir build # コンパイル用のディレクトリを作成

/pico/pico-examples/build$ cmake .. # makeファイルなどを作成

/pico/pico-examples/build/blink$ make -j4 # 実際に.uf2ファイルを作成

補足情報：

pinの25番が基板に付いているLEDらしい。

#### 事前準備

##### ラズパイ・Linux(Debian系)

スクリプトをダウンロード→実行 chmod +x で実行権を付ける事。

wget https://raw.githubusercontent.com/raspberrypi/pico-setup/master/pico\_setup.sh

# https://github.com/raspberrypi/pico-setup/pico\_setup.sh と同じ

（注意！）あくまでもラズパイの為に作られたスクリプトなので、Linuxの場合は注意して使用する

SKIP\_OPENOCD デバッグツールのインストールをスキップする場合は=1

# Toolchain のインストール

sudo apt install cmake gcc-arm-none-eabi libnewlib-arm-none-eabi build-essential

メモ

arm-none-eabi-c++.exeが格納されているPATHを追加（手動でやる場合）

export PATH=”/c/~~/gcc-arm-none-eabi-10-2020-q4-major/bin”:${PATH}

～pico　事前準備　続き

##### Windows

必要なアプリケーション

Build Toos for Visual Studio MSYSを使用する場合は不要

開発者用コマンドプロンプト（Developer Command Prompt）と言うのがあり、これを用いるとcmakeが使える。

setx PICO\_SDK\_PATH "..\..\pico-sdk" # cmdでcmakeする時のPICO\_SDK\_PATHの指定方法

（参考資料）<https://zenn.dev/iwatake2222/scraps/725fc773fcfd7d>

MSYS Windows用の最小Unixシステムの事。VisualStudioを使用する場合は不要。

<https://www.msys2.org/>

pacman -S mingw-w64-x86\_64-cmake # 必要パッケージのインストール

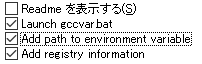
pacman -S mingw-w64-x86\_64-toolchain git make libtool pkg-config autoconf automake texinfo mingw-w64-x86\_64-libusb # 必要ならこれらもインストール

error: mingw32: signature from "\*\*\*\* (MSYS2 development key) <\*\*\*\*\*>" is unknown trust

⇒ ウイルス対策ソフトの例外設定にMSYS2のフォルダを指定する

ARM GCC compiler

方法１：

↓からDL、インストールしてPATHを通す。

<https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads>

手動でやる場合は　コントロールパネル → ユーザーアカウント → ユーザーアカウント → 環境変数の変更　で arm-none-eabi-c++.exeが格納されているPATHを追加

方法２：

同urlからポータブル(.zip) 版をダウンロードして手動でパスを通す？

方法３：MSYS内でインストール ※注意　1,2 と3は共生できない。どちらかを選ぶ。

pacman -S mingw-w64-x86\_64-arm-none-eabi-gcc

CMake (MSYSの時は不要かも？)

同じく、PATHを通す

<https://cmake.org/download/>

Windowsの設定は公式チュートリアルgetting-started-with-pico.pdfの9.2. Building on MS Windowsを参照

※Githubにこんなリポジトリがあるが？　https://github.com/ndabas/pico-setup-windows

（参考資料）<https://qiita.com/iwatake2222/items/33b4cd3a39da5a44dc02>

～pico　つづき

#### cmake

##### コンパイル方法

ファイル構成

pico\_cmake\_test

├ CMakeLists.txt # cmakeのビルド設定ファイル。cmakeコマンドの実行に必要。

├ pico\_sdk\_import.cmake # pico-sdk/externalの中からコピー

├ pico-sdk # gitでcloneしてここに配置 ※他の方法で指定してあれば不要

├ *build* # cmake で作成されたmakefileなどを格納する。

├ *my\_project* # ソースコードなどを格納

│ ├ CMakeLists.txt # プロジェクト毎にCMakeListsがあった方が管理しやすい

│ ├ main.c # 実際のプログラムソース

pico\_cmake\_test /CMakeLists.txt

cmake\_minimum\_required(VERSION 3.12) #　バージョンを指定

set(PICO\_SDK\_PATH ${CMAKE\_CURRENT\_LIST\_DIR}/pico-sdk) # PICO\_SDK\_PATH を直接指定

include(pico\_sdk\_import.cmake)

project(*my\_project\_name*) # プロジェクト名を指定

set(CMAKE\_C\_STANDARD 11) # set は環境変数の指定

set(CMAKE\_CXX\_STANDARD 17) # CXXはC++を指す

pico\_sdk\_init()

add\_subdirectory(*my\_project*)

pico\_cmake\_test/my\_project/CMakeLists.txt

set(BinName "*my\_binary\_name*") # 名前は何でも良いらしい

add\_executable(${BinName}

main.c

)

target\_link\_libraries(${BinName} pico\_stdlib)

pico\_add\_extra\_outputs(${BinName})

ビルド：

/pico\_test/my\_project/build$ cmake .. # MakeFileなどを作成

/pico\_test/my\_project/build/my\_project$ make -j4 # 実際に.uf2ファイルを作成

# 以下の様にしてもビルドできたのでメモ

/pico\_test/my\_project/$ cmake -B build # MakeFileなどを作成

/pico\_test/my\_project/build$ make -j4 # 実際に.uf2ファイルを作成

（補足）パスの末端に付いている　$　はLinux環境で一般ユーザーを指す。Windows環境では気にしなくて良い。

> 　pico\_cmake\_test /build/my\_project/my\_binary\_name.uf2 というファイルができる

～pico　cmakeつづき

##### MSYSの場合

cmake .. -G "MSYS Makefiles" # MSYSを用いる時のgenerator指定

##### リファレンス(pico特化のみ抜粋)

pico\_enable\_stdio\_usb(${BinName} 1) #

pico\_enable\_stdio\_uart(${BinName} 1) #

##### トラブルシューティング

Does not match the generator used previously: Visual Studio 15 2017

→ プロジェクト内にある余計なファイルやディレクトリを削除してもう一度やり直す。

CMake was unable to find a build program corresponding to "MSYS Makefiles"

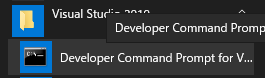
CMAKE\_MAKE\_PROGRAMが設定されていない。

～pico　つづき

#### VSCode

##### 注意事項

※先に上記の「cmake」を参考に開発用ディレクトリを作成しておく

※Windowsの場合は（開発用コマンドプロンプト）からvscodeを起動

Win → Visual Studio 20\*\* → Developer Command Prompt for...

#cmd

C:\Program Files (x86)\Microsoft Visual Studio\2019\BuildTools> code

##### 事前準備

 Extention で、以下の拡張機能のインストール

C/C++ for Visual Studio Code？　要る？？　この辺り、要再確認 2021-06-09

marus25.cortex-debug，ms-vscode.cmake-tools，ms-vscode.cpptools，CMake Tools

※Linuxの場合は terminalで以下の様にしても良い

code --install-extension ms-vscode.cpptools

設定

 → Settings → Workspace 必須ではないが、Workspace毎に設定できるので推奨

"cmake.configureOnOpen": false, //

"cmake.configureOnEdit": false, //

"cmake.generator": "NMake Makefiles", // Windowsの場合（後述）

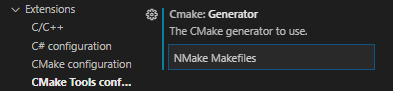
"cmake.environment": {

"PICO\_SDK\_PATH": "C:/pico-sdk/" // 当然、実際の場所に合わせる

}, // 直下ディレクトリにpico-sdkが有れば不要

"C\_Cpp.default.configurationProvider": "ms-vscode.cmake-tools",

注意点として、windows環境で有ろうがディレクトリを表す文字に / を使う。

Windowsの場合はcmakeのgeneratorを設定：

Open Folderでcmake開発用ディレクトリを開く

Extensions → CMake Tools configuration

cmake.generator というjsonがあるらしいが？

##### ビルド

下の  をクリックしてGCC for arm-none-eabi \*.\*.\* を選択　※ない場合は[Scan for kits] をクリック

コマンドパレット(Ctrl+Shift+P) → CMake:Buildまたは下の  でビルドする。

##### トラブルシューティング

could not load cache

##### 参考資料

<https://datasheets.raspberrypi.org/pico/getting-started-with-pico.pdf> 公式チュートリアル（英語）

～pico　つづき

#### コーディング（micropython）

##### Thonny(RaspberryPi)

予めWeb接続をしておく。

右下の  と言うところをクリック → MicroPython(Raspberyy Pi Pico)を選択

→ Install MicroPython firmware... と出るのでInstall　（この時、毎回Web接続が必要）

コーディングする（Lチカの一例）

from machine import Pin

import time

while True:

led = Pin(25, Pin.OUT)

led.value(1)

time.sleep(1)

led.value(0)

time.sleep(1)

→ Runでpicoに書き込む

（その他サンプルコード）<https://github.com/raspberrypi/pico-micropython-examples>

##### VSCode

pico-go を使うと良い（らしい）

公式サイトで  をクリックしrp2-pico-20210418-v1.15.uf2 をダウンロード

<https://www.raspberrypi.org/documentation/rp2040/getting-started/>

ラズパイpicoを接続し、.uf2ファイルをドラッグ＆ドロップ

Linux

/dev/ttyACM0のパーミション変更

sudo nano /lib/udev/rules.d/50-udev-default.rules

Windows(ターミナルはpower shell を想定)

python get-pip.py # <https://bootstrap.pypa.io/get-pip.py> をDL

python -m pip install micropy-cli

python -m pip install pylint

 Extention で、以下の拡張機能のインストール

pico-go 要再起動？？

pico-goの公式情報：<http://pico-go.net/docs/start/quick/>

pico-goの付属情報？<https://medium.com/all-geek-to-me/developing-for-the-raspberry-pi-pico-in-vs-code-getting-started-6dbb3da5ba97>

付属の付属<https://qiita.com/naohiro2g/items/bdaeac218f83de2d3c5b>

Pymakr

（参考：Ubuntu）<https://qiita.com/Fal318/items/8488c60649a334d16994>

（参考：Windows）<https://rikoubou.hatenablog.com/entry/2021/02/05/150631>

### Nano RP2040

Arduino Studio で開発できるRaspberry Pi Pico(RP2040)ボード。

### Compute Module

従来のRaspberry Piを使ってプロトタイプを作成し、市販製品には大量のCompute Modulesを組み込む、がコンセプト

バッテリー付きRTC

PCIe Gen 2ソケット 1つ

ビデオパフォーマンスが格段に向上

Compute Module IO Board 6,820 円

Compute Module 9



### StereoPi

カメラ搭載ボード？

## OS（ディストリビューション）

### 基本設定（共通）

・SDカードのフォーマットはFAT32。パーティションが無い状態にする。

・運営の推奨はSD Card Formatter。SDカードの速度を引き出す事ができる。

・イメージ書き込みソフト

Etcher ： 新しめのアプリ。portable版もある。一番人気。

### 主流ディストリビューションの概要

#### NOOBS

厳密にはディストリビューションではない。以下のもの含み，内１つを選択してインストールする。

但しRaspbianよりはストレージを圧迫するので，ディストリビューションが決まっている時は使わない方が良い。

Raspbian, LibreELEC\_RPi2 OSMC\_Pi2

Screenly OSE RISC OS Windows10 Iot Core Lakka\_RPi2 TLXOS

#### Raspbian

ラズパイの基本OS。初めてラズパイを使う人はこのOSから始めると良い。

サーバー、マイコンなど幅広い用途で使えるが、どちらかと言うとデスクトップ(GUI)色が強い。

#### Ubuntu MATE

デスクトップ用途(GUI)、パソコンとしての用途に適したOS。重たいのでラズパイ4でないときつい。

#### LibreELEC

動画再生専用機を作れる。OSMC と言うOSも似たような用途で使えるが、LibreELECの方が再生できるものが多いらしい。OSMCと共に、Kodiと言うOSをベースに作られている。

#### CentOS

Linux 系サーバーの代名詞CentOS もインストール可能。

⇒ Webサーバー用途でもRaspbianのCUIを使っているケースが多い。CentOSのケースは少ないらしい。

<http://ftp.yz.yamagata-u.ac.jp/pub/linux/centos-altarch/7/isos/armhfp/>

<https://wiki.centos.org/Download>

#### Win10 IoT Core

もともとラズパイ用に作られたOSではない為か、CPUの発熱が凄まじかったり、ソフトウェアが上手く動かなかったりする。どうしてもWindows環境でないといけない場合以外は使う事は無い。

この２つは失敗した：NOOBS、IoT Dashboardをそのまま使用。

参照）[httpsqiita.comfulcrumitemsaaf818696c3d14c431ff](smb://sshel93971/usi-nas/data/書類/doc/httpsqiita.comfulcrumitemsaaf818696c3d14c431ff)

ラズパイ3B ではTechnical Preview Build 17661しか動かない。このバージョンはInsiderと呼ばれるWin10の協力者メンバーにならないとダウンロードできない。19.05.21

iso イメージの中にmsi が入ってるのでインストール。C:\Program Files (x86)\Microsoft IoT\FFU\RaspberryPi2\flash.ff の存在を確認。その後Dashboard でSDに書き込む。

→19.05.22 結局失敗　イーサネットポートを認識しなかった

最新バージョンで再挑戦19.05.22

<https://qiita.com/tera1707/items/51027e2896f708959eaf>

→19.05.22 結局失敗　イーサネットポートの認識はしていたようだが、pingが通らなかった。

19.05.21情報

Iot Core を多少一般的にしたWoA (WoA Installer) というものがある。

64bitARM版+コアセット(dllのセット)を別々にダウンロードする。→どうも、ラズパイの仕様上限いっぱいで普通のWin10を動かす、というものらしい。実験するくらいなら良いかも。

netsh interface ip set address "Ethernet" static 192.168.xxx.51 255.255.255.0 192.168.xxx.1

shift ＋ 「け：」 でダブルクォーテーションが打てる

##### Windowsアプリの開発

<https://katsutoshi-otogawa.blogspot.com/2018/02/windows-10-iot-core.html>

#### その他

Pidora FedoraをRaspberry Pi向けにカスタマイズしたOS。

manjiro

DietPi

RISC ARM用に作られたデスクトップOS(GUI)

### 主流ディストリビューション詳細

#### CentOS

OSイメージ例：CentOS-Userland-7-armv7hl-RaspberryPI-Minimal-2003-sda.raw.xz

##### ログインなど

初期ユーザーはrootしかないらしい。パスワードはcentos

passwd root # rootのパスワードを変更。セキュリティの為，必ず行う。

##### キーボード設定

localectl # 現在のロケール情報を確認する

localectl set-locale LANG=ja\_JP.utf8 # ロケールを変更

localectl set-keymap jp106 # 109キーボードでも，106と指定する。

localectl set-keymap jp-OADG109A # 特に必要ないかもしれない。

localectl # 再確認

# cd /etc/X11/xorg.conf.d # このディレクトリに00-keyboard.conf というファイルが作成される

timedatectl set-timezone Asia/Tokyo # ついでにタイムゾーンを指定

date -s "01/23 4:56 2020" # 時間の指定方法

（補足情報）英字キーボード

|  |  |  |  |
| --- | --- | --- | --- |
| ： | shift +； | ｜ | shift + 」 |
| ‘ | ：（装飾キー無し） | ＿ | shift +　－（マイナス） |
| ＝ | ～（装飾キー無し） |  |  |

蛇足情報

/etc/sysconfig/keyboard # このファイルを編集するという情報もあるが？

# CentOSのviでは，Ctrl＋Cでコマンドモードへ移行するらしい

##### 最初から入っているコマンド

（抜粋：アルファベット順）

システム系： lsblk, parted, systemctl

ユーザー： cat, vi,

##### IPアドレス設定

/etc/sysconfig/network-scripts/ifcfg-eth0

nmtuiを使う

Edit a connection → Add/ネットワークプロファイルを選択 →

IPv4 CONFIGURATION：Manual

Addressess，Gatewayを設定

nmcliを使う

nmcli d # 現在のネットワークデバイスの使用状況を確認

nmcli c mod eth0 ipv4.addresses 192.168.1.110/24 # 有線LANのIP

nmcli c mod eth0 ipv4.gateway 192.168.1.1 # デフォルトゲートウェイ

nmcli c mod eth0 ipv4.dns 192.168.1.1 # DNS

nmcli c mod eth0 ipv4.method manual # IPアドレスを固定

nmcli c down eth0　&& nmcli c up eth0 # 設定の反映

確認

ip a # ipアドレスを確認

cd /etc/sysconfig/network-scripts # ifcfg-eth0　というファイルが確認できる

wifiに接続するには

nmtui → ActivateConnection

##### SSH有効

yum list | grep ‘ssh’ # ssh関連のパッケージのインストール状況の確認

vim /etc/ssh/sshd\_config # sshデーモンの設定

systemctl start sshd.service # sshデーモンを開始

systemctl status sshd.service # 状況を確認

##### パッケージ管理

リポジトリの設定

<https://qiita.com/bezeklik/items/9766003c19f9664602fe>

⇒ はっきり言って，意味不明 20.09.16

yum update # パッケージの更新

##### その他

lsblkやdf -h で確認すると，３つめのパーティションのファイルシステムが制限されている事が確認できる

rootfs-expand # ファイルシステムを拡張する

# 関連デバイス

## タッチパネル

|  |  |  |
| --- | --- | --- |
| 概要 | 写真 | 詳細 |
| 純正  7inch  ￥9400 |  | これだけでは自立しない。  少し高い。  <https://akizukidenshi.com/catalog/g/gM-09917> |
| 公式フレーム  ※純正の7inchが使える  ￥3000 |  | <https://akizukidenshi.com/catalog/g/gM-10065/> |
| ADAFRUIT  PiTFT  3.5inch  本体ケースと一体型  ＋ネジ穴式  ￥6091 |  | [https://www.sengoku.co.jp/mod/sgk\_cart/detail.php?code=EEHD-55ZD#](https://www.sengoku.co.jp/mod/sgk_cart/detail.php?code=EEHD-55ZD) |
| emerge+  VESAマウンタ  通常モニタに取り付けるタイプ  ￥1000 |  |  |

### モニタケース／スタンド

|  |  |  |
| --- | --- | --- |
| 概要 | 写真 | 詳細 |
| 公式フレーム  ※純正の7inchが使える  ￥3000 |  | <https://akizukidenshi.com/catalog/g/gM-10065/> |
| RSコンポーネンツ  ￥4,006 |  | こちらもまた，純正のが使えるらしい  <https://www.switch-science.com/catalog/2715/> |

## カメラ

第一世代 Omnivision OV5647：503万画素 5メガピクセル \3,080

第二世代 Sony IMX219PQ：808万画素 8メガピクセル 2016/4　\4,300

※通常のusbカメラ（webカメラ）の方が性能が良い事が多く，ラズパイ専用カメラを使うメリットは低いのでは？

|  |  |  |
| --- | --- | --- |
| 概要 | 写真 | 詳細 |
| RSコンポーネンツ  カメラモジュールv2.1  SONY IMX219PQ を使用  ￥2,966 |  | ◦1080 p@30 fps  ◦720 p@60 fps  ◦VGA（640 x 480 p）@90 fps  <https://akizukidenshi.com/catalog/g/gM-10518/> |
| ラズベリーパイ財団  Piカメラ  SONY IMX477Rを使用  ￥7,700 |  | 広角レンズなどをマウントできるマウントベース。  <https://akizukidenshi.com/catalog/g/gM-15318/> |
| エレコム  UCAM-C750FBBK  ¥2,850 |  | 2592p@15fps  1920p@30fps  マイク内臓（モノラル）  ※横に首を振れないらしい |
| Logicool  C920n  ¥6,964 |  | 1080p@30fps　720p@30fps  マイク内臓（ステレオ） |
| Logicool  C505  ¥3,000 |  | 720p@30fps  マイク内臓（モノラル） |
| Logicool  C270N  ¥2,040 |  | 720p@30fps  マイク内臓（モノラル） |

## 環境センサ系

|  |  |  |
| --- | --- | --- |
| 概要 | 写真 | 詳細 |
| pololu  VL53L0X  ￥1,540 |  | 赤外線レーザーを使った距離センサ  Class1のレーザー装置らしいので取り扱い注意（直視は避ける）  Gitにc++のライブラリがある。  <https://www.switch-science.com/catalog/2894/> |
| BME280  ￥1,080 |  | 温湿度・気圧センサ  I2CやSPI通信の良い練習になる  <https://akizukidenshi.com/catalog/g/gK-09421/>  （SPI) <https://tool-lab.com/pic-practice-25/> |

温湿度・気圧センサ

<https://tool-lab.com/pic-practice-25/>

## バッテリー

### カタログ

|  |  |  |
| --- | --- | --- |
| 概要 | 写真 | 詳細 |
| LiPo SHIM |  | https://www.switch-science.com/catalog/3216/ |

### 簡易UPS

情報元：<https://qiita.com/MechaTracks/items/b715f4712611e64df78e>

方法１

ADコンバーターを使う。

リチウムイオン（3.7V）

|  |  |  |  |
| --- | --- | --- | --- |
| 値段 | 型番 | 容量 | 備考 |
| ¥1,300 | DTP603048(PHR) | 860mAh | <https://www.marutsu.co.jp/pc/i/836350/> |
| ¥2,200 | DTP605068(PHR) | 2000mAh | <https://eleshop.jp/shop/g/gH4H31F/>  Data Power社 |

モバイルバッテリー(5V)

|  |  |  |  |
| --- | --- | --- | --- |
| 値段 | 型番 | 容量 | 備考 |
| ¥999 | APT100-10000M-5V-CWHBL | 10000 mAh | IO Data |
| ¥2,275 | AP20000D-DGT-5V | 20000 mAh | IO Data |
| ¥2,799 | PowerCore 10000 | 10000mAh | Anker　5V |
|  |  |  |  |

⇒ モバイルバッテリーの方が明らかに性能が良い。但し，勝手に電源が落ちたり，USB給電とバッテリー給電が切り変わらなかったりするらしい。

補足）

mAhは文字通りだが，「1時間に流せる電流」の事。

単電池は使用する材料で電圧が来まる。リチウムイオンは3.7V，鉛は2V，ニッケル水素は1.2Vなど。

## RTC（時計）

|  |  |  |
| --- | --- | --- |
| 概要 | 写真 | 詳細 |
| Raspberry Pi用RTCモジュール v1.1  ￥1,056 |  | I2C通信。  ボタン電池CR1220を使用する。  <https://www.switch-science.com/catalog/2857/> |
|  |  | <https://www.switch-science.com/catalog/1726/> |
|  |  | <https://www.switch-science.com/catalog/3392/> |

## 通信端末

|  |  |  |
| --- | --- | --- |
| 概要 | 写真 | 詳細 |
| L-02C  Docomoの専用品  ￥3,300  （中古品しかない） |  | SIMフリー化ができるらしい。 |
| WiMAXルーター |  |  |

## その他

|  |  |  |
| --- | --- | --- |
| 概要 | 写真 | 詳細 |
| sparkfun  Pan/Tilt Bracket Kit  ＄6.95 |  | 水平および垂直の2方向にカメラをパン。  <https://www.switch-science.com/catalog/3486/> |
| GPEG  KickStart  LCDディスプレイモジュール  ￥1900 |  | ４行×1８文字（列）のディスプレイ。  棒グラフも対応。  <https://jp.rs-online.com/web/p/lcd-monochrome-displays/1711685> |
| CrowPI  ￥15,000くらい？ |  | ※21.01.12現在 kick starterでのみの取り扱い  <https://www.elecrow.com/crowpi2.html> |
| GoovePi+  ￥4367 |  | Raspberry PiでGroveシステムのパーツを使えるようにする  <https://www.switch-science.com/catalog/2129/> |

ケース

|  |  |  |
| --- | --- | --- |
| 概要 | 写真 | 詳細 |
| sparkfun  Pan/Tilt Bracket Kit  ￥3,800 |  | スーファミ型のケース。  <http://www.retroflag.com/SUPERPi-CASE-J.html> |

# 内部デバイス仕様

## Bluetooth

bluetoothctl(bluez) というアプリ（サービス）を用いる事で簡単にブルートゥース接続ができる。

sudo sdptool add --channel=22 SP # これはRFCOMM（≒232C）のチャンネル（＝ポート番号）らしい

## Wifi仕様

## シリアル通信

USB→シリアル

### RS-232C

市販ケーブルなど買うのが手っ取り早い。

秋月　通販コードM-08343 \980 ※エレコムUC-SGT1も普通に認識，通信できた。20.07.06

ls -l /dev/serial/by-id/ # 有効デバイスの確認　　例) /ttyUSB0などと表示される

直接GPIOピンを使用

6番(Ground)、10番(RXD)、 8番(TXD) として使う。

初期設定の場合，baudrate は115200、パリティ無し、ストップビット1bit らしい

232Cの場合は7Vに変換，だが変換基板はあまりない。usb変換する方が一般的らしい。

先にraspi-config でシリアル通信を有効化する必要がある。その後，port名を確認。

ls -l /dev/ttyS0 # 直接ピンを使う時のポート名　ttyS0などと表示される。

ls -l /dev/ttyAMA0 # 注意！こちらはUART0用で，初期設定ではBluetoothのモジュールになっている。

python:

import serial

ser = serial.Serial( # シリアル通信の設定

port='/dev/ttyS0', # 使用するポート名の設定

baudrate= 9600,

bytesize= serial.EIGHTBITS, # SEVENBITS =7; EIGHTBITS =8

stopbits= serial.STOPBITS\_ONE,

parity= serial.PARITY\_NONE) # なぜかNONE以外指定できない 19.07.10

ser.write(data) # 送信

cuというコマンドを使っても良いらしい。

sudo apt install cu

### RS-485

ラズパイはRS-485には対応していないので，変換基板などを購入して通信する事になる。

変換基板BOB-10124

こちらの場合，変換基板とは232C通信を行う

ケーブル：USB-RS485-WE-1800-BT

## 無線ディスプレイ

厳密に言うと無線ディスプレイではないが、ラズパイでVNCを有効化してスマホやタブレットで接続するのが一番楽で確実。その際、公共wifiなどを用いず直接接続のような形にしないとVNC接続できない場合が多い。

方法１．スマホ側でテザリングを利用する　　　デメリット：スマホの通信量を使ってしまう。ラズパイで固定IPが△

方法２．ラズパイでDCHPサーバーを起動　　デメリット：少しめんどくさい。無線インターネットが使えなくなる。

方法３．ラズパイにwifiアダプタを追加 　　　デメリット：めんどくさい。（そもそも可能？）

# 開発環境

## VSCode

## Balena

RaspberryPi、Jetson、Coral Dev Boardなどに対応

<https://www.balena.io/cloud/>

10デバイスまでは無料で使える

11台目以降の最安契約は　$109/月

# ラズパイでGUI

## 概要

### X Window System

昔からLinuxで使われているGUI環境

ラズパイでGUIフレームワークを探すのなら，X Window Systemに対応している必要がある。

sudo apt install --no-install-recommends xserver-xorg

sudo apt install --no-install-recommends xinit

#### PIXEL

Pi Improved Xwindow Environment, Lightweight」の略

Raspberry Pi用にカスタマイズされた軽量GUI環境？

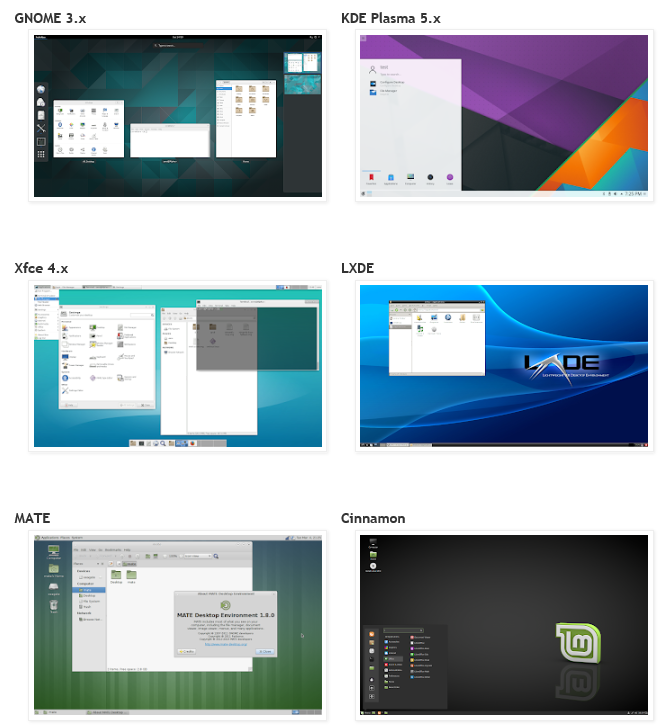
⇒ 古い記事でしか発見されないようだが？20.10.08

### GUIアプリ開発方法のまとめ

|  |  |  |  |
| --- | --- | --- | --- |
|  | 使用言語 | 概要 | 備考 |
| GTK+ | C（※１） | Linuxネイティブ。 | Linux系では圧倒的な採用率。 |
| Qt | C++（※１） | クロスプラットフォーム性が強い。 | 無料版は機能制限あり。商用版はGPL。  非推奨だが，ラズパイそのものに開発環境を入れられる。 |
| mono | C#など | .NETライブラリへのラッパーを提供してコンパイル。 | Microsoftの.NET Coreへの力の入れようで，状況は変わってくる気はする。 |
| Flutter | Dart | モバイル系に人気。 | まだまだ成長過程だと思われるが，モバイル系の人間の動向で状況は変わってくる気がする。 |

※１）但し，どちらもC++，C#，Java，Ruby，Python，Perlなどの主力言語のAPIは存在する。

参考画像



## GTK

（別ドキュメントgtk.docxからの抜粋）

総合デスクトップ環境GNOME用のツールキット等として広く利用。（例えばUbuntuはGNOME）

最初はLinuxでの画像編集ソフトGIMPを実装する為に開発されたウィジェットだった為，特にLinuxとの関係が強い。

以前はGTK+と呼ばれたが，2019年にGTKに改名された。

元々がC言語で，様々な言語版がある。

C++ 　(gtkmm) ，C#　(Gtk#)，Perl 　(gtk2-perl) ，Python　(PyGTK) ，Java　(Java-GNOME)

他にもJavaScript、Vala。非公式にFortran (gtk-fortran)、Ruby (Ruby/Gtk2) ，PHP (PHP-GTK)、Pascal、Lua、Haskell、FreeBASIC

実行環境の確認

gcc -dumpversion # gccのバージョン確認

g++ -dumpversion # g++ のバージョン確認

### 開発方法

#### Raspberry Pi上で開発：Geany

開発前準備

sudo apt install libgtk-3-dev # gtk用開発ツール

コンパイル

# コンパイルは通常このように行うらしい ⇒ gtk\_app.out というファイルが作成される

gcc *gtk\_source.c* -o *my\_app* `pkg-config --cflags --libs gtk+-3.0`

⇒ 毎回これを行うのは面倒なので，makeファイルを作成。makefile という，拡張子無しのテキストファイルを作成する。

Geanyで .cファイルを開き，メニュー：ビルド → メイク

（詳細は割愛）

（参考）<https://qiita.com/danjiro/items/efcad069d8e5e3fd6e49>

（参考；補助）<https://www.ei.tohoku.ac.jp/xkozima/lab/raspTutorial2.html>

#### Linux/Raspberry Piで開発：Glade

GladeはGUI部分のみを作成するツール。

sudo apt install glade

⇒ インストールは簡単にできた。Glade自体の使い方は別ファイル「ソフトウェア情報」にまとめた。

その他参考情報

<http://home.a00.itscom.net/hatada/cpp/gtk01.html>

[https://ja.wikibooks.org/wiki/GTK%E3%83%97%E3%83%AD%E3%82%B0%E3%83%A9%E3%83%9F%E3%83%B3%E3%82%B0](https://ja.wikibooks.org/wiki/GTKプログラミング)

#### Windowsで開発：Visual Studio

まずは普通にFormアプリケーションのプロジェクトを作成する。.NETのバージョンは4.7以降にする。

NuGetでGtkSharp.Win32（Mike Kestner）を導入。

（参考）<https://gist.github.com/seraphy/55e1428ef1b1b551d4730f4d4fe8eadc>

#### Windowsで開発：Glade

Glade公式：<https://glade.gnome.org/>

### その他

C言語のソースで以下のインクルードを追加する

#include <gtk/gtk.h>

Gtk+用のツール

資料が豊富なので学習しやすいらしい。

<http://www.peteronion.org.uk/GtkExamples/GladeTutorials.html>

使えない

<https://python-gtk-3-tutorial.readthedocs.io/en/latest/index.html>

Visual Studioでgtk　　[http://www.fecorp.co.jp/raspberry-pi%E3%81%A7windows-net](http://www.fecorp.co.jp/raspberry-piでwindows-netのアプリケーションを動かす/)

### GTKを使ったシステム

#### GNOME（グノーム）

Linux MintやFedora，OpenSUSEなどで利用。

#### MATE（マテ）

GNOMEの派生。Linux Mintとの関係が密接。

#### Cinnamon（シナモン）

GNOMEのもう一つの派生。独特な外観らしい。

## Qt（キュート）

特にマルチプラットフォーム用GUIとしての価値が高く，組み込みシステムでも使われている（らしい）。

意外と歴史は長い。用途はGUIだけではない。

Linuxにおいては総合デスクトップ環境KDE（デザインがWindowsに近い）の基盤として使われている。

Java，Ruby，Python，Perl，C#で利用できるオープンソースAPIが存在する。

非GUIも作成可能。商用ライセンス(GPL)有。

ソースはC++で記述。

sudo apt install qt-sdk

qtbase5-dev # 上とま

libgl1-mesa-dev

qt5-default

–fix-missing

・デザイン性が良い。

・QT Creator を使う事でフォームデザインが簡単に行える。

・マウス，キーボード，マルチタッチイベントなどの広範な入力に対応

sudo apt install qtbase5-dev libgl1-mesa-dev qt5-default –fix-missing

キットオプション 

ビルドディレクトリの修正

一般のビルドディレクトリ。最初は存在しないディレクトリが設定されているので修正。　必要？20.10.06

コンパイラの設定１

Manage Kits ⇒ ビルドと実行 → 〔コンパイラ〕タグ → 追加 → GCC →

C = /usr/bin/gcc C++ = /usr/bin/g++ ※但しこれは最初から設定されている。

コンパイラの設定２

Manage Kits ⇒ ビルドと実行 → 〔キット〕タグ → コンパイラ → CとC++共にGCCに。

参考

<https://www.kunimiyasoft.com/raspberry_pi_qtcreator/>

<https://qiita.com/soramimi_jp/items/b2b238d1537dbb99abea>

### KDE

Qtを利用したデスクトップ環境。Windowsにより近いインターフェイス。

## mono

基本情報

.NET Frameworkを動かすためのオープンソースソフトウェア群，またそのプロジェクト名。

2018年3月現在ではXamarinと.NET Foundationが開発，販売，サポートを行っている。

標語：Cでプログラミングするには人生は短すぎる

動作プラットフォーム：

Linux, Android, macOS，iOS，Sun Solaris，Windows

BSD（OpenBSD、FreeBSD、NetBSD）

Wii，PlayStation 3/4 # この辺りが驚き。というか意味不明。

対応言語：

C#，F#，VB.net，Java，Python，Ruby　など

ライセンス：GPL, LGPL, MIT（複合ライセンス）

公式のインストール情報

英語ではあるが，非常に分かりやすい。

[https://www.mono-project.com/download/stable/#download-lin-raspbian](https://www.mono-project.com/download/stable/" \l "download-lin-raspbian)

monoのインストール

sudo apt install mono-devel # 開発用キット。コンパイラなどを含む。

# sudo apt install mono-complete # 全てのパッケージを含む。“assembly not found” エラーが起こった時

# sudo apt install mono-dbg # デバッガ

sudo apt install fonts-ipaexfont # 日本語用のフォントをインストール。

コンパイル方法（つまりコンパイラも付属している）

csc hello.cs # Systemのみを使用したシンプルなCUIアプリならばこれで良い。

csc hello.cs -r:System.Windows.Forms.dll # dllの追加。Formを使ったGUIアプリ。

### リモートデバッグ

リモートからデバッグするためには，デバッガ専用のサーバーアプリMonoRemoteDebugger.Serverが必要

[https://github.com/techl/MonoRemoteDebugger/#usage](https://github.com/techl/MonoRemoteDebugger/" \l "usage) のUsageのところに説明がある。以下は補足。

unzip -d MonoRemoteDebugger.Server　MonoRemoteDebugger.Server.zip # -d オプションの使い方

Visual Studioで拡張機能を追加

１．VisualStudioのメニュー：ツール→拡張機能と更新プログラム

２． にして　　にMonoRemoteDebuggerと入力。

３．MonoRemoteDebugger.VSExtension.vsix がWebブラウザでダウンロードされるので，インストール

４．VisualStudioのメニューに  が追加される

Visual Studioでは普通に.NETアプリを作成（.NET Coreでは無い事に注意！）

mono someApp.exe

GitHubに手順があるらしい：<https://github.com/GtkSharp/GtkSharp/>

## Flutter

Flutterは特にAndroidやiOSで使われるケースが多い。2018年11月現在では公式には対応していないが，raspberryPiでFlutterを使おうという試みは多少みられる。

（参考）

<https://flutter.dev/docs/get-started/install/linux>（公式）

<https://medium.com/flutter/flutter-on-raspberry-pi-mostly-from-scratch-2824c5e7dcb1>（有志では一番有名）

<https://github.com/ardera/flutter-pi>

今後主流になるかも？20.10.06

チュートリアル

ここから.xz ファイルをダウンロード[https://flutter.dev/docs/get-started/install/linux#install-flutter-manually](https://flutter.dev/docs/get-started/install/linux" \l "install-flutter-manually)

mkdir sdk # 先に作業ディレクトリを作成。以後このチュートリアルはこのディレクトリを使う (~/sdk)

tar xJvf flutter\_linux\_1.22.0-stable.tar.xz 　# 解凍 ※Web説明では何故かオプションがxf

export PATH=“$PATH:/home/pi/sdk/flutter/bin/” 　# PATHを通す。 .bashrc などを使用すると良い。

# source ~/.bashrc # 一度実行しておく。（.bashrcの例）

flutter # bin/cache/dart-sdk　が作られる。

（補足）

Raspberry Pi OS lite だとGitが入っていない為，Unable to find git in your PATH.　というエラーが発生する。

ここからDartをダウンロード[https://dart.dev/get-dart#install](https://dart.dev/get-dart" \l "install)　→ downloading the SDK as a zip file.

※なぜかArm　v7のものをダウンロードするらしい 20.10.07

⇒ 解凍して sdk/flutter/bin/cache/dart-sdk　とフォルダごと置きかえる

rm -rf flutter/bin/cache/dart-sdk #

mv dart-sdk/ flutter/bin/cache/ # フォルダごと移動

export PATH=“$PATH:/home/pi/sdk/flutter/bin/cache/dart-sdk/bin” 　# 追加でPATHを通す。

flutter

参考動画：<https://www.youtube.com/watch?v=SHc3NB1LdlI&feature=youtu.be>

⇒ 2日ばかり試したが，結局うごかなかった。2020.10.12

## 亜流

### wxWidgets

各プラットフォームのネイティブツール（LinuxのGTK、Windows のWin32 API、MacOS XのCocoa)を使用する為，ネイティブの風合いなどがでる。但し他の選択肢の方が良い場合が多く，あまり選択されない。

独自のウィジェットの下にネイティブプラットフォームを用いるGTKやQtと対照的。

### .NETCore

.NETを使うもう一つの手でもある。

但しGUIアプリを作るのはめんどくさかったような。20.09.29

一部情報では「できない」とある。

但しMicrosoft自身はクロスプラットフォームへの取り組みを.NETCoreに集中させているので，今後どうなるかはだれにも分からない。

<http://www.cutt.co.jp/book/978-4-87783-466-1.html>

## Python系

### Kivy

　Python を使ったGUI開発ができる為Raspberry Pi 開発者の間で人気。

Pythonを使用⇒ 高速用途には向かないかも？

※20.01.21現在では、各GUIアプリはデザインが安っぽいので、Webアプリを作った方が良いと思う。

### Tkinter

こちらもKivyと同じくPython で動作する。

sudo apt install python3-tk # 標準で入っているらしいが？

import tkinter # python での使用　as tk などとしてもよい

機能面がやや貧弱。

Pythondeha標準でTkinterというモジュールがある（らしい）

python

from tkinter import \*

import tkinter.ttk as ttk

root = Tk()

root.geometry(“1024x768”)

## 蛇足情報（GUI）

macOS版ソフトはCocoaというフレームワークがある

FLTKも人気らしいが，それが避けられない状況でない限りは使わない？

QtとGTKで外観を似せよう，というまとめ[https://wiki.archlinux.jp/index.php/以下略](https://wiki.archlinux.jp/index.php/Qt_と_GTK_アプリケーションの外観の統合)

# 色々な使い方

## サーバーとして使う

### ルーターの設定：

ローカル環境で使う時は必要ない。

グローバル（WAN）で使う場合，アクセス要求はルータのWANに届くので，このポート毎へのアクセス要求をRaspberry Piにポートフォワーディングする設定を行う必要がある。

静的マスカレードでルーターのWANをラズパイのLAN IP に変換？

プライベート認証局の設置が必要：

<http://dreamerdream.hateblo.jp/entry/2016/01/04/000000>

### NAS

※Network Attached Storage の略

#### 必要な物：

外付けHDD （補足）HDDは電源を食う（10~18w）USBメモリ（通常1w以下）の方が良いかも

samba Raspbianには最初から入っている

#### 注意点など：

・外付けHDDのフォーマットをNTFS にするとMacから読めなくなる。

exFAT にするとMacからでも読めるようになる。

その他ファイル・サーバー・アプリ：

Openmediavault

ラズパイ用のイメージ(OMV\_4\_Raspberry\_Pi\_2\_3\_3Plus.img.xz)がある。

参考）<https://raspida.com/make-nas-rpi>

#### openmediavault

webブラウザで簡単設定

IPv6 対応

RAIDも対応（らしい）

### VPNサーバー

SoftEtherVPN

<https://nobuntu.jp/raspberrypi-vpn-server-softethervpn/>

### html

SSLサーバ証明書をインストールするとhttps 通信も可能に

Apache

言わずと知れたhtmlサーバーの代表格。

Nginx

エンジンエックス。軽量Webサーバー。近年人気。

同時アクセスに対する負荷が比較的強いが，重量処理は苦手。広く浅く。

クラウド

NextCloud

### データベース

同時に２，３ユーザー程度しか使わないのであれば問題ない。

#### SQLite

アプリケーションの一部に組み込んで動作させるタイプ

読み出しは早い。書き込みはデータ件数が増えると遅くなる。特にUPDATE が遅い。

#### mySQL

マルチスレッド。シンプルなWebサービス向き。

多くの操作をNon Blockingで操作できる。

他の人に配布する場合、GPLは適用されず商用ライセンスが必要

#### PostgreSQL

マルチプロセス。RDBMSに求められるものが過不足なく搭載されている。

Python などを利用した外部プロシージャが使える

BSDライセンス（無料）

sudo apt-get install postgresql

クライアント・マシン側にPostgreSQL ODBC Driver をインストールする必要がある

<https://www.postgresql.org/ftp/odbc/versions/msi/>

MariaDB

MariaDBは、MySQL派生として開発されている、オープンソースのRDBMS。

### Webアプリ

幾つか選択肢がある：

#### mod\_wsgi

WSGI（Webサーバーで起動するインタープリタ）をつかうとhtml から .sh やpythonモジュールが使用できるようになる。

ちなみにmod\_php ならぬmod\_python と言うのもあるが、あまりお勧めできないらしい。

#### htmlでPHP/JavaScript

JavaScriptはクライアント側で動くので，ラズパイでやる事はなにもない。

#### フレームワーク

Django　 強力なライブラリ

フル装備のアプリ作成。Python系サーバーでは最も人気。

中規模以上のシステムもカバー。

Flask　　比較的シンプル

素早くシンプルに作りたい。アプリ作成初心者向き。

必要最低限のコア機能　が売りらしい。（機能の追加は当然可能）

Bottle　　非常にシンプル。最も軽量なフレームワーク。

### ERP

※ERPはWANのサーバーでないと意味がないので、ラズパイではそのテスト運用程度にする。

grocy

### ソケット通信

## パソコンとして使う

### アプリケーション

#### 業務アプリ等

Office ：LibreOffice

$ sudo apt-get install libreoffice # 英語版？

$ sudo apt-get install libreoffice-l10-ja # 日本語版

$ sudo apt-get install libreoffice-help-ja #

メーラー ：Claws Mail, Geary, Thunderbird

ただ、Web経由でgmail 等を使っている人が多い様子。

IceDove(Thunderbird) メーラー

Webブラウザ ：Chromium (Raspbian標準), FireFox, Google Chrome,

日本語入力 ：Mozc

メディアプレーヤ：VLC, Audacious など

その他 ：画像ビューア(Viewnior)、pdfビューアー、テキストエディタ(Leafpad)

cat Linux標準コマンド。テキストファイルの表示

Mousepad 「テキストエディタ」として使っているアプリ。（元leafpad）

nano Terminal 上でテキスト編集するソフト。比較的簡単。

VIM 同じくTerminal上で動作。玄人向き。

### リモートデスクトップ

#### クライアント

rdesktop

sudo apt-get -y install rdesktop

※コマンドを打ってから少し時間がかかる -u:ユーザー名

※win10側でリモートの許可、ユーザーの登録が必要システムのプロパティ→リモートタブ

ただしwin10のリモートデスクトップは１クライアントしか認めていないので，あまり使い物にならない。

xrdp ？？？

### ノートPC化キット

Pi-Top ※但し3万円くらいするので、格安ノートを買った方が良い。



## マイコンとして使う

### 概要

配線情報：

<https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/README.md>

## 電圧変換

社内基板の例

使用機器：出力側から順番

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 型番など | 写真 | 詳細 |
| VHコネクタ  JST | B2P-VH(LF) |  | 定格電圧250VAC |
| ダイオード  東芝 | CRG03 |  | 最大逆電圧(Vrpm) 400V  コネクタ付け間違い対策で付けている？ |
| SSR（リレー）  omron | G3VM61A1 |  | In 順方向50mA  LED点灯のIFはmin5mA～Typ7.5mA～Max25mA  Out最大負荷電圧48V  最大電流500mA |
| トランジスタ  アレイ(8ch) | TBD62083A |  |  |
|  |  |  |  |

出力側

24Vのシンクロジック。

この24Vはマイコン側とは独立した別電源（当たり前ではあるが）。SSRにはこの電圧が印加される。

シーケンサ側は4.0～5.3mA流れる。X（入力）側で3.5mA以上@X0～17必要，

マイコン側

この例ではIn側（マイコン）は5Vに470Ω抵抗 ⇒ 10.6mA流している。

秋月にアイソレート基板が売っているので，大いに参考になると思う。

<https://akizukidenshi.com/catalog/g/gP-15646/>

市販品

DIO-8/4RD-IRC

RPi-GP10T

解析

TXS0108E　<https://www.switch-science.com/catalog/2309/>　※但しIN：1.2～3.6V OUT：1.65～5.5V

参考資料

論理レベル変換の話：<http://www.digikohma.com/shoho/shoho17.html>

## 旧情報

減圧すれば良い？同じようにフォトカプラを使う？

理論的にはトランジスタアレイ＋フォトカプラが無難

PC817

## プログラミング

python で入出力は管理できるらしい（詳細不明 19.06.06）

実際にプログラムで使用する時は mmap というものを使う。

→ Web上にはC言語で書かれている？例が多く、どうもポインタらしい。要するに関数的にGPIO を制御する為のメモリをマッピングするらしいが 19.05.07

## PLC

### ソフト面（ラダー）

#### OpenPLC

OpenPLCはRaspberryPiOS（旧Raspbian）上で動く。Docker上でも動くらしい。

Windows上に開発環境を作る場合

公式サイト<https://www.virtualbox.org/> からEditorのzipファイルをダウンロードしてインストール

ラズパイ／Linux上に開発環境を作る場合

git hub を使ってインストールする。

git clone https://github.com/thiagoralves/OpenPLC\_v3.git

cd OpenPLC\_v3

./install.sh linux

※但しラズパイ側は実行環境（Runtime）も必要になる。

./install.sh rpi # ラズパイの場合は最後にrpiをつける

#### CODESYS：

無料のPLCソフト。但し業務ライセンス有り。

ダウンロードにユーザーアカウント作成が必要なので，まだ試していない。20.12.24

### ハード面

#### Interface 24VアイソレートIO基板

雑誌Interface 2020年9月号で紹介されている基板。

秋月で基盤とパーツが販売されている

<https://akizukidenshi.com/catalog/g/gP-15646/>

※基板内の配膳図が公開されている。

DIO-8/4RD-IRC

フォトカプラによる絶縁では無く，

I2C接続で入出力制御信号を扱う

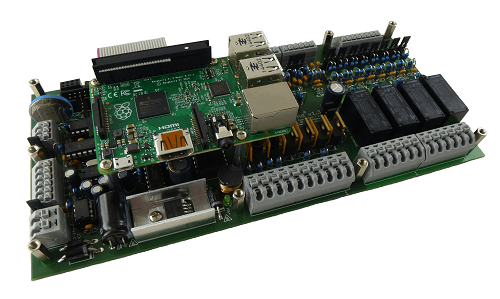
<https://www.switch-science.com/catalog/3149/>

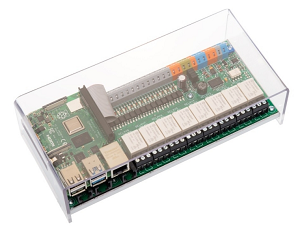
こちらも基板内の配膳図が公開されているが

I2Cなのであまり参考にはならない。

#### PiXtend

ドイツで作られている産業用拡張基板

PiXtend Sが€160



UniPi

チェコの会社

Unipi1.1は €149

#### Monarco HAT

ラズパイ自体をPLC化する。

（割と情報が古いので良く分からない）

### PLCの補助マシンとして

・PLCはIOや耐久性などは強いが、ネットワークに弱い。ネットワーク部を補う為にラズパイを用いる。

・複雑な計算をラズパイにやってもらう。

・三菱の場合：MCプロトコルによる通信でラズパイとデータのやり取りも可能。

（キーエンスの場合はUSB―RS232C をつないでシリアル通信で出来るらしい）

## ルーターとして使う

・ファイヤウォールの構築が可能

・VPN が構築できる。

自動起動

### PPP接続

インターネットプロバイダに直接接続する際、など。

以下のパッケージを利用する

sudo apt install pppoe pppoeconf # パッケージの追加

### 無線アクセスポイント

以下のパッケージを利用する

sudo apt install dnsmasq

### DHCP

以下のパッケージを利用する

sudo apt install hostapd

### DNSサーバー

Dnsmasq

sudo apt install dnsmasq

設定ファイル：/etc/dnsmasq.conf

bindを使う，という方法もある。こちらはプロ現場でも使われる分，複雑だとか。

（関連）/etc/resolv.conf （dnsサーバーのリスト）

（関連）avahi-daemon（mDNSを扱うサービスの事）

## 自動起動

ラズパイ（Linux）上でのアプリケーションの自動起動の方法はいくつかある

### /etc/rc.localやcronや自作サービス（systemd)など。

### ※しばらく別ドキュメントLinux.docxに記述する。2021-12-07

# 具体的な用途例

## オーディオレシーバー(A2DP)

⇒ 結果的にはノイズが多くて使い物にならなかった。

・Bluetooth の設定　　※XX:XX:XX:XX:XX:XX はMACアドレスを示す。

sudo bluetoothctl

>> [NEW] Controller XX:XX:XX:XX:XX:XX raspberrypi [default]

[bluetooth]# show

>> Controller XX:XX:XX:XX:XX:XX

UUID: Audio Sink が含まれていることを確認。これがないとA2DP接続できない。

⇒ ない場合はパッケージを追加。※ただし、Audio Sinkはデスクトップ版でしか使えない。

sudo apt install pulseaudio-module-bluetooth # インストール後に再起動が必要

もしくは、古いバージョン-2018-04-19-stretch などだと大丈夫らしい

・ペアリング

terminal:

[bluetooth]# pair XX:XX:XX:XX:XX:XX # ペアリング

[bluetooth]# trust XX:XX:XX:XX:XX:XX # 認証を行う

認証が通ったら、client 側の Audio 出力設定を Bluetooth（A2DP）にしておく。

・入力の確認

pactl list sources short # Audio のソース一覧を出力

　　　　　　　　　　　　 # Running がでていればOK

・出力先の確認

pactl list sinks short # Audio の出力一覧を出力

・設定ファイル

# /etc/pulse/daemon.conf （補足）; か # でコメントアウト

resample-method = trivial # ノイズ対策になるらしい

情報元：<https://qiita.com/dwarfJP/items/a4a90d5c0aa2833dac5b>

<http://belial6.hateblo.jp/entry/2017/06/30/003546>

## スマートホーム

### 概要

・SMSサービスのbot

LINE

Slack

sudo pip install slackbot

（参考）

<https://qiita.com/undo0530/items/2139a1e8b73b3eee6e00>

Discord（下に詳細情報有り）

python3 -m pip install -U discord.py

・スマートホームサービス

Apple Siri

Amazon Alexa

Googleアシスタント

Nature Remo

Switch Bot

git clone <https://github.com/OpenWonderLabs/python-host.git>

・音声認識ソフト

音声認識ソフトはJulius（ジュリアス）が良いらしい。

・デフォルトのままでも音声認識は行うが，処理に時間がかかるため，必要最小限の辞書登録をするのが無難。

（参考）https://www.pc-koubou.jp/magazine/19743

### discord

#### チュートリアル

１．discordのbot作成

discordのアプリサイトへ：<https://discordapp.com/developers/applications/>

アプリ作成　：Applications → New Application ※CientIDとPublic Keyをメモ

ボットを作る： → 

アプリの認証

（例）https://discord.com/oauth2/authorize?client\_id=\*\*\*\*\*&scope=bot&permissions=0

２．ラズパイ側の設定

sudo apt install python3-pip # python パッケージ管理（まだない場合のみ）

sudo apt install libffi-dev libnacl-dev python3-dev # 関連パッケージ

python3 -m pip install -U discord.py

# 仮想環境を作成

cd discordbot # 仮想環境を作成するディレクトリを作成して移動

python3 -m venv bot-env # 仮想環境を作成

source ./bot-env/bin/activate # 仮想環境をアクティベート

（参考）<https://discordpy.readthedocs.io/ja/latest/intro.html>

３．pythonのコードを記述

トークンを上記discordのサイトで確認　  →

ソースコードは本家サイトなどを参考にする　https://discordpy.readthedocs.io/en/latest/intro.html

注意！この時のファイル名をdiscord.py としない事。インストールしたdiscord.pyと区別できなくなり，上手く動かない。

（補足）

詳細情報：<https://realpython.com/how-to-make-a-discord-bot-python/>（英語）

４． .serviceファイルの作成

/etc/systemd/system/ に*discordbotd*.serviceファイルを作成。以下は簡単な例。

[Unit]

Description=discord bot service

[Service]

Type=simple

ExecStartPre=/bin/**bash** -c 'source /home/*pi*/src/discord/bot-env/bin/activate'

ExecStart=/home/*pi*/discord/bot-env/bin/python3 /home/*pi*/discord/discordbot.py

Restart=on-failure

#StandardOutput=file:/tmp/discordbotd.log # 必要であれば，標準出力のログ

#StandardError=append:/tmp/discordbotd\_error.log # 標準エラー出力。appendは

[Install]

WantedBy=multi-user.target

～discord　続き

#### ハマりやすいポイント

・systemdから動かすとdiscord.pyが無いと言われる（terminalからのpython3の実行は成功する）

python3をインストールする時にsudoを付ける必要がある（らしい）

sudo pip3 install discord.py

・cliend.send\_messageでエラー

原因：恐らく古い情報を参考にしてしまっている。最近のdiscord.pyではmessage.channel.send() を使う

・systemd からvenvの実行ができない。

原因：.shスクリプトの実行に/bin/shを指定してしまっている為，sourceが使えない。

対策：/bin/bashを使う。

（補足）sourceはbash特有のコマンドで，sh（実際にはdash）では使えない。

・それでもvenvから.pyの実行ができない。

原因：systemのディレクトリから実行される為 (？) 相対パスの指令方式だと上手く行かない。

対策 ：ソースコードの中では絶対パスで指定する。またはUNITファイルで作業ディレクトリを指定。

*※以下の例ではvenvは/opt/discord内に有り，実行ソースは/home/user/discordにあるものとする。*

UNIT ファイル内：

ExecStart=/opt/discord/bot-env/bin/python /home/user/discord/main.py

または

WorkingDirectory=/home/user/discord # これにより，全ての相対呼び出しが変化する点に注意

ExecStart=/opt/discord/bot-env/bin/python main.py

Shebang

#!/opt/discord/bot-env/bin/python3

※他にも，PATHを動的に通す方法がある（らしい）

UNITファイル（.service) のEnvironment でPATHを通す。

Environment=PATH=/opt/discord/bot-env/bin:$PATH # 具体例

#### 参考資料

discord.py→ver 1.0 に伴う修正点のまとめ ※2019年頃の話らしいが，Web には古い情報が多いはず。

<https://discordpy.readthedocs.io/en/rewrite/migrating.html>

トラブルシューティング

<https://qiita.com/coolwind0202/items/a4405be45734bd7f6cd5>

venvの使い方

<https://virtualenv.pypa.io/en/16.7.9/userguide.html>

### LINE

pip3 install line-bot-sdk

### 音声認識

#### Julius

マイクの設定

（参考）<https://qiita.com/yozawa/items/0695664ad89dad56e62f>

（参考）<http://www.asahi-net.or.jp/~cb9i-kn/geek/20170502mic.html>

Juliusのインストール

ダウンロード

sudo apt install osspd-alsa libasound2-dev # 必要なパッケージをインストール

# osspd-alsa はいらない？21.01.10

mkdir julius # ついでに/julius/julius-kitも作る

cd julius

wget https://github.com/julius-speech/julius/archive/v4.4.2.1.tar.gz # ソースを落とす

コンパイル＋インストール

tar xvzf v4.4.2.1.tar.gz

cd julius-4.4.2.1

./configure **--with-mictype=alsa** # ここで少し時間がかかる

make # コンパイルには更に時間がかかる

sudo make install

Julius音声認識パッケージを取得

cd ~/julius/julius-kit

wget https://osdn.net/dl/julius/dictation-kit-v4.4.zip

unzip dictation-kit-v4.4.zip

cd ~/julius/julius-kit/dicration-kit-v4.4/ # ディレクトリ移動

julius -C main.jconf -C am-gmm.jconf -demo # Juliusのデモを実行

（参考）<https://qiita.com/mininobu/items/f181b762955932cec6d6>

（参考）<https://www.pc-koubou.jp/magazine/20637>

関連設定ファイル

/etc/modprobe.d/alsa-base.conf

/proc/asound/modules ？？procはおかしいのでは？

### Text-to-Speech

#### 概要

（2020.03.03現在）

|  |  |  |
| --- | --- | --- |
| 機能 | 1 か月あたりの無料枠 | 有料使用 |
| 標準（非 WaveNet）音声 | 0〜400 万文字 | 100 万文字あたり $4.00（USD） |
| WaveNet 音声 | 0〜100 万文字 | 100 万文字あたり $16.00（USD） |

#### 新しいプロジェクトを作成

#### Google Could Platform

Google Cloud Platformへ。

https://console.cloud.google.com/

　GCPの登録方法の情報はWebに沢山あるので割愛。

　（参考）https://blog.apar.jp/web/9893/

※課金を有効にする必要あり（クレジットカードの登録）

補足：実際にお金を支払うほど使う事はほぼ無い。

GCPのコンソールでの操作。

右図の赤枠のところをクリック→新しいプロジェクト →

プロジェクト名を入力 → 作成

・上の検索Box（右図参照）にText-to-Speechと入力

もしくは　APIの概要に移動→APIとサービス → APIとサービスの有効化

※念の為、「割り当て」でリクエスト上限を指定しておくと、料金が発生しにくい。

・プロジェクト設定に移動→IAMと管理→サービスアカウント

→サービスアカウントを作成→ロール（省略可）→

プロジェクト「text-to-speech-test」のサービス アカウントが表示される

サービスアカウント

操作→鍵を管理（または、アカウントを選択→キー）

→キーを追加→新しいキーを作成→JSONを選んで作成

ダウンロードフォルダに[project\_name]-◯◯◯◯.jsonという鍵ファイルがある事を確認する。

（参考）https://ohmyenter.com/have-raspberry-pi-speak-with-google-tts/

（公式）https://cloud.google.com/docs/authentication/getting-started

#### インストール

ラズパイでの操作。

先に仮想環境を作って置くことを推奨。

[ Terminal ]

mkdir weather\_speech # 作業用ディレクトリを作成

cd weather\_speech # 作業用ディレクトリに移動

touch test.py # （任意）仮想環境を作る時、同レベルのディレクトリに.pyファイルがあると良い

python3 -m venv env # 仮想環境を作成

source env/bin/activate # 仮想環境をアクティベート

インストール

[ Terminal ]

python3 -m pip install --upgrade google-cloud-texttospeech

鍵ファイルの指定方法

以下のように環境変数で指定する

[ Terminal ]

export GOOGLE\_APPLICATION\_CREDENTIALS="JSONファイルのパス"

# 具体例

export GOOGLE\_APPLICATION\_CREDENTIALS="/home/pi/weather-speech/text-to-speech-test-\*\*\*\*.json"

ソースコードは公式のものをコピペでOK

[https://cloud.google.com/text-to-speech/docs/create-audio-text-client-libraries?hl=ja#client-libraries-usage-python](https://cloud.google.com/text-to-speech/docs/create-audio-text-client-libraries?hl=ja" \l "client-libraries-usage-python)

但し、以下の箇所だけ変更。

[ Python ]

voice = texttospeech.VoiceSelectionParams(

language\_code="en-US", ssml\_gender=texttospeech.SsmlVoiceGender.NEUTRAL

)

↓

voice = texttospeech.VoiceSelectionParams(

**language\_code="ja-JP"**, ssml\_gender=texttospeech.SsmlVoiceGender.NEUTRAL

)

公式チュートリアル：https://cloud.google.com/text-to-speech/docs/quickstart-client-libraries?hl=ja

### AquesTalk PI

### Open JTalk

[https://open-jtalk.sp.nitech.ac.jp](https://open-jtalk.sp.nitech.ac.jp/)

### Google Nestとの連携

#### [google-home-notify](https://github.com/noelportugal/google-home-notifier" \l "after-npm-install)

公式：[https://github.com/noelportugal/google-home-notifier#after-npm-install](https://github.com/noelportugal/google-home-notifier" \l "after-npm-install)

get key faild from google対策

Cloud Text-to-Speech API を有効にし、サービスアカウントキーを発行後、環境変数 GOOGLE\_APPLICATION\_CREDENTIALS を設定します。

$ export GOOGLE\_APPLICATION\_CREDENTIALS=/path/to/dir/target.json

→2022/12/23試したが、効果なし

参考２：

https://github.com/zlargon/google-tts/issues/33

<https://remoteroom.jp/diary/2020-12-31/>

#### VoiceText

HOYAが提供するText-To-Speechソフトウェア。

Node.jsのライブラリが存在する

npm install voicetext

HOYA公式：<https://cloud.voicetext.jp/webapi>

更に nori-dev-akgという人が作ったgitレポジトリがあるので利用する

参考：[https://qiita.com/nori-dev-akg/items/751e7d9bf2728afda28a#voicetext-web-api](https://qiita.com/nori-dev-akg/items/751e7d9bf2728afda28a" \l "voicetext-web-api)

## リモコン制御

### 概要

（参考）<https://qiita.com/takjg/items/e6b8af53421be54b62c9>

必要な物：

赤外線LED

赤外線受光モジュール （例）OSRB38C9AA

受光確認用LED 有っても良い，程度で必須ではない。

必要なパッケージ：

pigpio GPIO制御。入っているかどうかは dpkg -l pigpio で確認できる。

irrp.py pigpio作者による赤外線制御スクリプト [http://abyz.me.uk/rpi/pigpio/examples.html#Python\_irrp\_py](http://abyz.me.uk/rpi/pigpio/examples.html" \l "Python_irrp_py)

※補足：LIRC(Linux Infrared Remote Control) が有名だが，少し不便だとか。

sudo apt install pigpio python3-pigpio

sudo systemctl enable pigpiod.service # pigpiod の自動起動の設定

sudo systemctl start pigpiod # いますぐ pigpiod を起動

pip install -U pigpio # venv を使うときは仮想環境側でこれが必要

GPIOの設定（任意；未設定でも問題ないらしい）

echo 'm 17 w' > /dev/pigpio # 基本例 GPIO17をoutput（w）

echo 'w 17 0' > /dev/pigpio # 基本例 GPIO17の出力を low（0）に設定

# GPIO17を、out - lowに、GPIO27を、in - pull up（u）に設定。

echo 'm 17 w w 17 0 m 27 r pud 27 u' > /dev/pigpio

irrp.pyをダウンロード

curl http://abyz.me.uk/rpi/pigpio/code/irrp\_py.zip | zcat > irrp.py

このスクリプトのシェバンが　#!/usr/bin/env python　になっている点に注意

実行

python3 irrp.py -r –g 27 -f codes light:on --post 130 # 記録 --post は省略可能

python3 irrp.py -p –g 17 -f codes light:on # 実行

オプション

-r record　赤外線信号の記録

-p playback　赤外線信号の発信

--post 入力信号の受信が指定ミリ秒途絶えたら終了

--no-confirm 通常，確認のために再度信号をレコードするが，それを省く。

irrp.pyはクラス化されていない為，他の.pyファイルなどから実行する時は以下のようにする。

import subprocess

subprocess.run(["python3", "irrp.py", "-p", "-g17", "-f", "codes", "aircon:off"])

（pigpio参考資料）http://abyz.me.uk/rpi/pigpio/index.html

～リモコン制御　概要　続き

デフォルトは38kHzだが，リモコン赤外線には36kHz、40kHz、56kHzなどの周波数もある。

python3 irrp.py -p -g27 -f codes light:on --freq 36.0 # 周波数を変えて実行

補足情報

・irrp.py をモジュール化している人がいる　<https://qiita.com/Cartelet/items/1a451ec0abf5734aceae>

（個人的にはsubprocessで不自由を感じないのでメリットは少ないと考える）

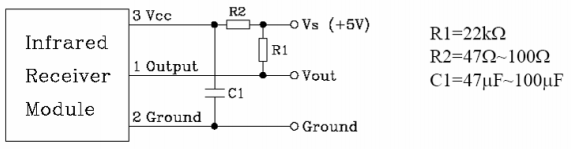
・38kHzと言う事は，パルス幅26μsec，Duty比50%である（はず）

1/38k = 0.026×10-3→26μsec この間にon/offをしているはずなのでDuty比50%と言える。

### 受信用回路

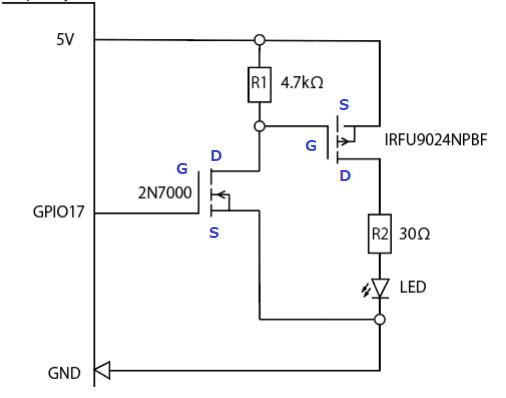
但しこの回路はノイズなどの影響で受信精度が悪くない限りは必要ない。

直接ラズパイのGPIOにつないでも問題ない。

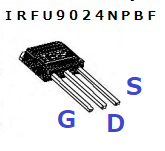
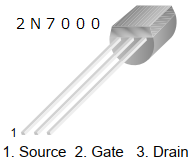


### 送信用回路

<https://vintagechips.wordpress.com/2013/10/05/赤外線LEDドライブ回路の決定版 - 電脳伝説>

補足情報

LEDは足が短く，LED内部が三角な方がカソード（－）



詳細

・2N7000(N ch)でOn/Offし，IRFU9024NPBF(P ch)でドライブする。

・FETを２つ使う理由は，コンプリメンタルにする事で信号の速度を上げている。（？）

・LEDに流す電流は500mA程度までは増強可能（らしい）。

⇒ 具体的には調整用抵抗の値を減らすか．LED（と抵抗）の数を並列して増やす。

*IRFU9024NPBはVDSが5Vの時IDは1.1Aで（データシートによる）duty比が50%だから？？？*

・この回路をトランジスタ1つで構成している例があるが，一般的なトランジスタでは十分な電流量を得られず，赤外線信号が1mくらいしか届かない（らしい）。

・「決定版」のリンク先に，オン電圧4.85Vとあるが，これは実際の測定結果だと思う。21.01.27

・2N7000のゲートとGPIO(17)の間には，負荷が（容量の大きめの）コンデンサなどの時には，ショートした時の事を考え抵抗があった方がベターだが，今回は無くても良いらしい（正直理解できていない）。

参考資料

<https://vintagechips.wordpress.com/2013/10/05/赤外線ledドライブ回路の決定版/>

## 無線アクセスポイント

### 事前準備

raspi-configでWi-fi Country をJP Japanに（必須）

直接設定ファイルを編集しても良い　/etc/wpa\_supplicant/wpa\_supplicant.conf

・wlan0（無線LAN）の状態を確認（任意）

iwconfig

・wlan0の対応モードの確認（任意）

iw list

>> Supported interface modes: # この中に AP（アクセスポイント）がある事を確認

・wlan0 に固定アドレスを割り振る（必須）

# /etc/dchpcd.conf

interface wlan0

static ip\_address=192.168.1.10/24

denyinterfaces wlan0 # この行がある解説があったりするが意図が分からない

パッケージ追加

sudo apt install dnsmasq hostapd

# systemctl status dnsmasq # 状態を表示

systemctl stop hostapd # 設定の為、一応止めておく（dnsmasqも）

### パケットの転送設定など (sysctl.conf)

必須な項目

# /etc/sysctl.conf

net.ipv4.ip\_forward=1 # IPv4 パケット転送を有効にする。

特に設定すべきもの（検討中）

net.ipv6.conf.all.disable\_ipv6 = 1 # IPv6を無効化（使ってない場合）

# なりすまし攻撃を防止する。

net.ipv4.conf.default.rp\_filter=1

net.ipv4.conf.all.rp\_filter=1

# TCP/IP SYN クッキーを無効にする。

net.ipv4.tcp\_syncookies=0

# ICMP リダイレクトを受信しない。(MITM 攻撃の防止)

net.ipv4.conf.all.accept\_redirects=0

net.ipv6.conf.all.accept\_redirects=0

# IP ソースルートパケットを受信しない。

net.ipv4.conf.all.accept\_source\_route=0

net.ipv6.conf.all.accept\_source\_route=0

# 有り得ないパケットを記録しない。

net.ipv4.conf.all.log\_martians=0

⇒ 詳細情報： <https://qiita.com/JhonnyBravo/items/d8445d21d9c79501c6f6>

・以下の２つをインストール

sudo apt-get install dnsmasq hostapd # dnsmasq はDHCPを利用する場合のみ

# systemctl status dnsmasq # 状態を表示

sudo systemctl stop hostapd # セットアップの為にサービスを止める

# 作業が終わって再開する時は

sudo systemctl start hostapd

sudo systemctl unmask hostapd # service is maskedエラーが出た場合は

sudo systemctl enable hostapd # この２行を実行してから、再度 start を行う。

・hostapd には設定ファイルのひな型がある。/usr/share/doc/hostapd/examples/hostapd.conf

・wlan0の起動がhostapd より遅くなる事があるらしい。hostapd はwlan0の後に起動するようにする。どうやって確かめて、どうやって設定？19.09.23

※wlan0 が無い場合は以下の様にして追加するらしい 19.9.23

sudo nano /etc/network/interfaces.d/wlan0

### DHCP

IPアドレスの自動割り振り。DHCPの使用は必須ではない。

・dnsmasq の設定

sudo mv /etc/dnsmasq.conf /etc/dsnmasq.conf.old # 元ファイルを取っておく cpでコピーでも可

sudo nano /etc/dnsmasq.conf # シンプルな設定ファイルを新規作成する

# /etc/dnsmasq.conf

# 以下を追加： アドレスを192.168.1.100~120 の範囲で24時間貸し出す、の意。

interface=wlan0

dhcp-range=192.168.1.100,192.168.1.120,255.255.255.0,24h

# 当然 dhcp.conf で割り振った wlan0 の固定アドレスと同じネットワークにする。

また、DHCPには isc-dhcp-server を利用する、という情報もある。

※設定ファイル名がdhcpd.conf で、dhcpcd.conf と非常に似ており、紛らわしい

# /etc/dhcp/dhcpd.conf

中略

・・・

subnet 172.16.1.0 netmask 255.255.255.0 {

range 172.16.1.100 172.16.1.200;

option domain-name-servers 8.8.8.8, 8.8.4.4;

option domain-name "raspi.localnet";

option routers 172.16.1.1;

option broadcast-address 172.16.1.255;

default-lease-time 600;

max-lease-time 7200;

}

# /etc/default/isc-dhcp-server

中略

・・・

INTERFACESv4="wlan0"

### hostapd

hostapd の設定

# /etc/hostapd/hostapd.conf　（ファイルが無い場合は新規作成する事）

interface=wlan0

driver=nl80211

ssid=YOUR\_SSID # もちろん任意のSSIDにする

hw\_mode=g # a も使えるらしいが、設定の仕方が異なるとか

channel=6 # 遅い場合などは調整が必要 最適な物をWifi Analyzerで探す

ieee80211n=1 # IEEE 802.11n を有効にする

wmm\_enabled=1

ht\_capab=[HT40][SHORT-GI-20][DSSS\_CCK-40]

macaddr\_acl=0

auth\_algs=1

ignore\_broadcast\_ssid=0

wpa=2

wpa\_key\_mgmt=WPA-PSK

wpa\_passphrase=YOUR\_PASSWORD # もちろん任意のパスワードにする　8~64文字

rsn\_pairwise=CCMP # RSN / WPAの共有鍵暗号化方式としてCCMPを使用する

# 追加情報　多くの例には見られなかったもの

country\_code=JP

disassoc\_low\_ack=1

require\_ht=0

wpa\_group\_rekey=86400

max\_num\_sta=3

wpa\_pairwise=TKIP

・hostapd起動時のconfigファイルの指定：

#/etc/default/hostapd

# 以下を追加

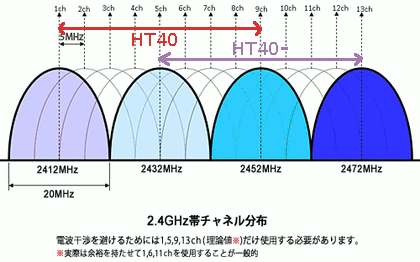
#DAEMON\_CONF=""

# ↓

DAEMON\_CONF="/etc/hostapd/hostapd.conf"

# DAEMON\_OPTS="-dd" # 未確認 ⇒ 特に設定する必要は無いらしい

hostapd.conf の情報：<https://qiita.com/JhonnyBravo/items/5df2d9b2fcb142b6a67c>



チャンネル選択用参考資料※実は3B+は5GHz(80211a) も使える。

スマホのアプリなどで適切なチャンネル番号を調べられるらしい。

### ルーティング設定

・転送設定

terminal:

sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip\_forward"

・iptables にIP4転送の情報を記述

terminal:

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE # ルーターの後ろなどに付ける場合

# -t nat -A POSTROUTING -o ppp0 -j MASQUERADE # PPPoE 直接の時は eht0 → ppp0

# この２行は必要ではないが推奨

sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT

sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT

# sudo つけても実行できない時は sudo su でスーパーユーザーになってから実行する

・iptablesの設定の自動復元

方法はいくつかあるが、 /etc/rc.local を利用する。rc.local とは、linux 系の起動時にroot権限で実行されるスクリプト。rc.localの書き込みにもroot権限が必要。まずiptableの情報をファイルに書き出す。

terminal:

sudo sh -c iptables-save > /etc/iptables.ipv4.nat

# 実行できない時は sudo su を実行してから再トライ

# sh -c "iptables-save > /etc/iptables.ipv4.nat" とダブルスラッシュが付いている情報も

rc.local にスクリプトを追加

# /etc/rc.local

iptables-restore < / etc/iptables.ipv4.nat # 一番下のexit 0の直前に追加。

exit 0

・iptables-persistent

iptables設定を保存する為のツールも存在している。IPアドレスの設定が完了した後に実行する事。

sudo apt-get install iptables-persistent

etc/iptables/rules.v4 # ipv4 版の設定；etc/iptables/rules.v6 # ipv6 版の設定

### 最後の設定

hostapd,　dnsmasq (isc-dhcp-server) の再開

# hostapdはエラーが出るので、以下のように実行

sudo systemctl unmask hostapd

sudo systemctl enable hostapd

sudo systemctl start hostapd

あとはラズパイを再起動

### 追加情報

・これはterminal で実行する？いまいちわからない。 /etc/network/interfaces に記載する？

/etc/wpa\_supplicant/wpa\_supplicant.conf

allow-hotplug wlan0

・hostapd.confにはひな型が存在する。以下はコピー方法。

cd /etc/hostapd/

sudo cp /usr/share/doc/hostapd/examples/hostapd.conf.gz ./

sudo gzip -d hostapd.conf.gz

## ファイルサーバー

### 概要

NAS（Network Attached Storage）要するにネットワークを用いたファイル共有。

### マウント

exFATでフォーマットされたドライブを使う場合：　　※exFATのドライブにはファイル所有権を付けれない。注意。

sudo apt install exfat-fuse exfat-utils # exFATのドライブを使う為の２つのパッケージ。

### 準備

sudo blkid # HDDのUUIDを調べる　　　　ls -l /dev/disk/by-uuid　これでも可能

sudo blkid /dev/sda1 # 詳細情報の表示

sudo mkdir -p /mnt/naspi # 先にマウント先のディレクトリを作成。（path名前は自由）

### 自動マウント

/etc/fstabに設定を指定する事で起動時に自動でマウントしてくれる。

[ /etc/fstab ]　※UUIDの”” は有っても無くても良いらしい

#UUID 共有path ファイルシステム マウント時のオプション

UUID="5662-CA45" /mnt/naspi exfat-fuse async,auto,dev,exec,gid=65534,rw,uid=65534,umask=007 0 0

terminal:

sudo df -h # 該当HDDがマウントされているか確認（再起動後）

/etc/fstab詳細

UUID=XXXX-XXXX /path\_to/media vfat async, auto, dev, exec, gid=65534, rw, uid=65534, umask=000 0 0

メディアのID

マウント先のパス

パーティションのフォーマット vfat/fat-fuse など

オプション

dump コマンドの対象 0:対象外 1:バックアップ対象。

fsckのチェック順序 0:no check 1:root file system

※詳しくは別ドキュメント「Linux」を参照

### sambaの設定

sudo apt install -y samba # raspberrypiOS-lite(CUI) だとデフォルトでは入っていない

補足：パッケージの設定　という画面が表示されるが「いいえ」で良い

WINS（Windows Internet Name Service）からコンピュータ名とIPアドレスをひも付けたものを提供してもらう仕組み。昔のWindowsで使われていた機能。

設定ファイルのバックアップ作成（任意）

[ terminal ]

sudo cp /etc/samba/smb.conf /etc/samba/smb.conf.backup

/etc/samba/smb.conf

[public] # 共有名

comment = NAS by Raspi #

path = /mnt/naspi/public #

read only = no # 読み取り専用かどうか；つまり書き込み禁止かどうか。

browsable = no # ブラウズリスト中に現れる利用可能な共有の一覧

force user = pi # ゲストは強制的にこのユーザーを使う事になるpublic = yes

public = yes #

guest ok = yes # ゲストユーザー(パスワードなし)での接続を許可。

unix extensions = no # Windowsからのアクセスの際はnoにする必要ある

[share] # 共有名

path = /mnt/naspi/share #

security = yes # ゲストユーザー(パスワードなし)での接続を許可。

guest ok = no # ゲストユーザー(パスワードなし)での接続を許可。

force user = pi # ゲストは強制的にこのユーザーを使う事になるpublic = yes

[share] の方を使うためには、以下の設定が必要

/etc/samba/smb.conf

sudo pdbedit -a smbuser

再起動

[ terminal]

sudo service smbd restart

# Failed to restart samba.service: Unit samba.service is masked. が出た場合

sudo service nmbd restart

sudo service smbd restart

sudo service samba-ad-dc restart

# もしくは

sudo /etc/init.d/samba restart

参考

https://qiita.com/rkonno/items/54b2e4b8770c8b0b2bcd

https://turtlechan.hatenablog.com/entry/2019/06/08/235327

注意点：

・デスクトップモードのRaspbianだと自動マウントなので注意。常時接続するならUUIDでのマウントが良い。

・HDDだと結構電源を食う（10~18w）し，うるさい。USBメモリ（通常1w以下）の方が良いかも。

・fat32のディクスにアクセスする場合はディレクトリのアクセス権なども設定する必要がある。

・厳密にはTCP445番を開放する必要があるとか？

## WordPressのテスト環境

（情報元）<https://raspida.com/wp-onthe-lamp>

## BIツール

### Metabase

概要

ラズパイで動かすには少し重たいかもしれない。

#### インストール

##### .jar版

Javaのインストール

sudo apt install default-jre # 実行環境におけるデフォルトがインストールされるらしい

sudo apt install openjdk-11k # バージョンを指定する場合

java --version # インストールが成功したかどうか確認

metabase.jarをダウンロード。

wgetで手に入れる（コンソール版RaspberryPi OSの場合）

wget https://downloads.metabase.com/v0.35.3/metabase.jar

webブラウザで手に入れる（デスクトップ版RaspberryPi OSの場合）。またはクライアント機のWebブラウザでダウンロードした後、 scpやftp、sambaなどラズパイにで転送

（公式）https://www.metabase.com/start/oss/

Metabaseのインストール

java -jar metabase.jar

# 以下の文章が表示されるとリクエスト受付状態になる

Slack is not configured, not refreshing slack user/channel cache.

ネットワーク内のPCでWebブラウザを開き、<ラズパイのIP>:3000でMetabaseは実行できる。

## 画像認識

### tensorflow-lite

#### 概要

PINTO0309という人のGithubレポジトリ（以下PINTO0309とする）を使うことで、簡単にインストール可能。

ポイント

・パフォーマンスを考えると、TensorflowよりTensorflow-liteのほうが無難。

・「aarch64 OS」のほうがパフォーマンスが良いらしい

・USBカメラの画像を取得するにはOpenCVのVideoCapturメソッドを使う。

#### インストール

PINT0309の#USAGE

・apt パッケージのインストール。最初の$が不要なだけでそのままコピペ。

・TFVER=2.10.0-rc1　以降はbash変数の設定

こんな感じ

TFVER=2.10.0-rc1

PYVER=39

ARCH=aarch64

# そのあと　sudo -H pip3 install \　以降の行を実行（そこでこれらの変数が使われる）

TFVER tensorflow-liteのバージョン。とりあえずは 2.10.0-rc1のままが無難と思われる。

PYVER pythonのバージョン

python3 -V # 確認方法

ARCH OSのアーキテクチャー

python -c 'import platform; print(platform.machine())' # 確認方法

PINTO0309のレポジトリをclone

git clone <https://github.com/PINTO0309/TensorflowLite-bin>

memo

python3 mobilenetv2ssd-sync-usbcam.py --camera\_type="raspi\_cam"

#### モデルファイルのダウンロード

例：<https://tfhub.dev/google/aiy/vision/classifier/food_V1/1>

### Torch

### YOLO

#### 概要

Torchでもtensorflowでもできる様子。

ただし、処理能力的にYOLO v3-Tinyなどでないと厳しいらしい。

参考：<https://qiita.com/daiarg/items/3da30c93b05496522acb>

## bluetoothボタン

### Ruby版

Rubyのbluebuttonというモジュールが良いらしい。

sudo apt install ruby

sudo gem install bluebutton

sudo nano ~/.config/bluebutton

[~/.config/bluebutton]

keyup=echo UP

keydown=echo DOWN

longup=echo LONG UP

longdown=echo LONG DOWN

#実行

bluebutton -d="AB Shutter3" -c ~/.config/bluebutton

# 分解記事

https://note.com/tomorrow56/n/n83cd108e65d5

・2つのボタン押し分けれないか？

・長押しが無効

### python版

terminal:

sudo bluetoothctl

[bluetooth]# scan on # 検索開始

>> Device 47:5C:2F:XX:XX:XX AB shutter3 # AB shutter3のMACアドレスをメモする

[bluetooth]# scan off # 検索終了

[bluetooth]# pair 47:5C:2F:XX:XX:XX # ペアリング

>> Failed to pair: org.bluez.Error.AuthenticationCanceled # 認証が失敗する

[bluetooth]# trust 47:5C:2F:XX:XX:XX # 認証を行う（ペアリングが成功するはず）

exit

ボリュームを一度クリックしてボリューム設定状態にする。

ペアリングが成功していれば、bluetoothボタンで音量が上下するはず。

前準備

[ terminal ]

ls /dev/input # イベントを確認

>> event0 event1 などと表示

sudo pip3 install evdev # イベントを扱うpythonパッケージ

ソースコード（サンプル）

[ main.py ]

import evdev

while True:

try:

device = evdev.InputDevice('/dev/input/event1') # 上でか

print(device)

for event in device.read\_loop():

if event.type == evdev.ecodes.EV\_KEY:

print('code:%s value:%s' % (event.code, event.value))

except:

print('Retry...')

114が大きい方のボタン。115が小さい方のボタン。

参考：https://monomonotech.jp/kurage/raspberrypi/daiso\_btshutter.html

## メディアプレイヤー

### LibreElec

#### 概要

<https://libreelec.tv/>

|  |  |  |  |
| --- | --- | --- | --- |
| バージョン | ラズパイ4 | ラズパイ3 | 備考 |
| 10.0 | 〇 | × |  |
| 9.2 | 〇 | 〇 |  |

初期ユーザー名：root　初期パスワード：libreelec （ssh接続などで必要）

#### 注意点

・ラズパイに接続したUSBキーボードなどでAmazon Primeなどのアカウント名，パスワードを入力すると正しく入力したはずなのに認証がうまく行かない事がある。その場合はLibreELEC上のソフトキーボードで入力すると良い。

・通常のDebian系Linuxとディレクトリ名が異なる。以下，説明が必要なディレクトリのみ抜粋

/ ルートディレクトリ（最下層のディレクトリ）

├ flash 他のLinux環境の /boot と同等。config.txt もここにある。

├ storage sshでログインした時の最初のディレクトリ。ホームディレクトリ。

│ ├ pictures

│ ├ music

│ …

…

#### 基本設定

・設定（左上歯車マーク：以下同様）→ LibreElec→Network→Wireless Regulatory Domain をJapan(JP) に

・interface→skinの-Fonts→Arialbasedに。その後、Regional→Language→japanese

・地域→タイムゾーン（国）をjapanに

以下，状況による

・システム→アドオン→不明なソース→はいをクリックして有効化。（.zipファイルからリポジトリをインストールする場合）

・sambaに接続出来ないとき：sshをオンにする？LibreElecのsambaを一度offしてからonし直すと良い？

#### AmazonPrime

ver 10.0の場合

アドオン → リポジトリからインストール → ビデオアドオン → Amazon VOD を選択。

Ver9.2の場合

PCなどで， <https://github.com/Sandmann79/xbmc/releases>　から .zip形式のレポジトリをダウンロード

補足：LibreElec9.2はLeia(Kodi18.9) なので，repository.sandmann79.plugins-1.0.3-leia.zipをダウンロードする。

PCなどからLibreElecにsamba経由でアクセスして先程のファイルをコピーする。

アドオン → zipファイルからインストール → 先程のファイルを選択

レポジトリからインストール → 今回のレポジトリを選択 → ビデオアドオン →AmazonVOD

#### Netflix

zip形式のレポジトリをダウンロードしてインストールする。（上のAmazon Prime ver9.2の場合を参考）

リポジトリのGitHub：<https://github.com/CastagnaIT/plugin.video.netflix>

Leia：https://github.com/castagnait/repository.castagnait/raw/master/repository.castagnait-1.0.1.zip

Matrix：<https://github.com/castagnait/repository.castagnait/raw/matrix/repository.castagnait-1.0.0.zip>

#### Disney+

設定 → システム → Add-ons で 不明なソースが有効になっている事を確認。

設定に戻って　ファイルマネージャー → ソースを追加

「無し」となっているところを選択 → https://k.slyguy.xyzと入力。

→ メディアソース名の入力に「SlyGuy」と入力してOK

設定に戻って　アドオン → zipファイルからインストール → SlyGuy→ "repository.slyguy.zip"

アドオンに戻って　リポジトリからインストール → SlyGuy Repository

メインメニューに戻って　アドオン→リポジトリからインストール→SlyGuy

→ ビデオアドオン → Disney+ → インストール

アドオン → ビデオアドオン → Disney+ → 開く → ログイン

情報もと（英語）<https://www.matthuisman.nz/2020/02/slyguy-kodi-repository.html>

#### 詳細情報

バージョンの確認方法（Linux共通）

lsb\_release

#### トラブルシューティング

##### イアフォンジャックから音が出ない

発生日時：2021年12月06日

現象

LibreElecの設定→システム→オーディオ→オーディオ出力デバイス　にイアフォンジャック（bcm2835 Headphones, など）が存在しない。HDMI経由でしか音が出せない。

対応

config.txtを修正する事で音が出せるようになる。

ssh root@192.168.1.1 # opensshでラズパイにログイン。初期パスワードは　libreelec

mount -o remount,rw /flash # /flash ディレクトリ以下のファイルを編集可能にする為にリマウント

nano /flash/config.txt # config.txtをnanoで編集する

[ /flash/config.txt ]

dtparam=audio=on # これを追加。Ctrl+X でnanoを終了できる。※保存を忘れずに！

audio\_pwm\_mode=1 # これも必要だとか（未確認）

hdmi\_drive=2 # これも必要かも？（未確認）

terminal(ssh)

reboot # ラズパイを再起動

# 以下，任意（再びsshでラズパイにログイン）

aplay –l # サウンドデバイス情報を確認

⇒ この後，LibreElecの設定→システム→オーディオ→オーディオ出力デバイス　へ移動して設定を行う。

補足

・この問題はLinuxカーネルのバージョンアップにより発生したらしい。この先，修正されるかもしれない。

・「/flash をリマウントする」の情報元：https://raspida.com/libreelec-configtxt

##### MSLエラー

発生日時：2021年12月11日

現象

Netflixの動画が全て再生できなくなる。journalctlで確認すると以下のログ：

v3d fec00000.v3d: MMU error from client CLE (4) at 0x4d41000, pte invalid

対応

アドオン → Myアドオン → Netflix → 設定　→エキスパート → インプットストリームヘルパーアドオン設定→Wadevine CDMライブラリの再インストール

情報元（頑張って読む必要がある）：https://github.com/CastagnaIT/plugin.video.netflix/issues/1278

結果：再生できないまま

DRM制御のコンテンツを再生

Widevine

##### ログの確認方法

普通にjournalctlが使えるので，以下のようにする

terminal(ssh)

journalctl | tail -n 20 # 最新20件のログを表示する

### OSMC

<https://osmc.tv/>

2021-05-08 現在、ラズパイ4には対応していない様子。

初期ユーザー名，初期パスワードは共にosmc （sshなどで使用する

#### 基本設定

IPアドレス：MyOMSC→Network（上のやつ）

#### Naughty Bulid

/etc/apt/sources.list

deb http://download.osmc.tv/dev/gmc-19/public gmc-19 main # バージョンに合わせる

wget -qO - http://download.osmc.tv/dev/gmc-19/public/pubkey.asc | sudo apt-key add -

<https://raspida.com/osmc-nightly-amazonvod?amp=1> この手順だけで良いのでは？

（参考）<https://raspida.com/osmc-primevideo>

（参考）<https://forum.kodi.tv/showthread.php?tid=343195> ???

#### AmazonPrime

#### Netflix

このshellスクリプトを実行するだけ

curl https://raw.githubusercontent.com/zjoasan/netflix-install-script/master/netflix\_prep\_install.sh | sh

2021-05-09 だめだった。 結局Netflixが見たければラズパイ3でLibreElecを使うのが良さそう。

https://raspida.com/primevideo-libreelec9

https://makingstuffwork.net/technology/install-netflix-amazon-prime-video-plex-raspberry-pi-kodi-osmc/

メインメニュー → アドオン → zipﾌｧｲﾙからｲﾝｽﾄｰﾙ

先ほど作られたaddonsディレクトリを探して、中にあるplugin.video.netflix.zipを選択

### 共通情報（OSMC, LibreElec）

#### 基本

OSMC，LibreElec共に，内部のOSにはKodiというものを使っており，基本的な動作は同じ。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Kodiの  バージョン | コード名 | Linux  カーネル | LibreELEC | OSMC | 備考 |
| 19.3 | Matrix | 5.10.x | 10.0 |  |  |
| 18.9 | Leia | 4.19.x | 9.2 |  |  |

補足

Linuxカーネル 簡単に言うとLinux OSのバージョンの事。

#### おすすめリポジトリ

##### youtube

インストールは簡単。

但し、各個人でAPIKEYを取得する必要がある。

・公式サイトへ移動し、プロジェクトを作成　<https://console.developers.google.com/>

（「場所」は「組織なし」のままでOK）

・ライブラリ → Youtube Data API v3 → 有効にする

・認証情報を作成→「ユーザーデータ」を選択して次へ→

OAuth 同意画面

アプリ名：なんでもよいらしい　ユーザーサポートメールとデベロッパーの連絡先：同じでも良いらしい

→完了

・スコープは飛ばす

・OAuthクライアントIDの種類は「テレビ」か「デスクトップ」のどちらかだと思う

・.jsonファイルをダウンロード

・KODI側での設定

.kodi/userdata/addon\_data/plugin.video.youtube/settings.xml

（参考）<https://qiita.com/god19/items/aaab2b86fa08de5ac9d5>

### LineageOS

チュートリアル

参考：公式の「How to install Google apps?」

操作方法

F2：戻る

必要ファイルのダウンロード

以下の2つをダウンロード

LineageOS

<https://konstakang.com/devices/rpi4/LineageOS18-ATV/>

> Raspberry Pi Imager、Ether、ddなどでmicroSD に書き込む

OpenGAPPS

<https://opengapps.org/?arch=arm64&api=11.0&variant=tvstock>

> USBメモリなどに配置

起動

LineageOSを書き込んだmicroSDをRaspberryPiに挿入して起動。

補足１：言語、ロケール、wifiの設定などは今は不要なので飛ばす。

補足２：周辺のBluetooth機器は電源OFFにした方が良いかも。ペアリング成功するまで先に進めない。

OpenGAPPSのインストール

advanced restart optionsを有効にする

Settings -> System -> Buttons -> Advanced reboot

再起動

Settings -> System -> Reboot > Recovery

再起動後

Install > 先程USBに保存したopen\_gapps-arm64-11.0-tvstock-xxxxxxxx.zipを選択

インストール終了したら、Back > Wipe > Factory reset > 再起動

再起動後はAndroid TVになっているはず。

次ページに続く

補足

LineageOS 19

2022年8月13日現在、LineageOS 19 Android TV(Android 12L) は存在するが、Android 12Lに対応したopen\_gappsはまだ存在しない。そのためLineageOS 18 Android TV(Android 11)を使用する。

SELinuxが有効化できない！！

https://source.android.google.cn/docs/security/selinux?hl=ja

日本語キーボード対応

Androidのキーボードアプリ（GBoardなど）を入れるのが早い

Developer Options

Settings→Device Preferences→About→Build　でBuildを7回クリック。

電源ボタンの動作

単押しでディスプレイ出力をOFF

長押しで電源メニュー表示。［ロックダウン、電源オフ、再起動］

ロックダウン：生体認証などでロックをかける。

補足情報

他にも「OmniROM Android R」というOSがラズパイで動くらしい。

エラーで再生できないアプリがあったら

セキュリティ対策の通信がうまく行っていない可能性が高い。AndroidTVのアプリはスマホ用のアプリよりセキュリティ高めなんだとか。

これ経由でスマホ用アプリの.apkファイルをダウンロード

<https://apps.evozi.com/apk-downloader/>

（有名な？apkpureには危険なファイルもあるとか）

SSH

Developer Optionsを有効化。（About→Build　でBuildを7回クリック。）

Settings -> System -> Developer optionsでUSB デバッグ、Rooted debuggingを有効化。

USB接続して秘密鍵を取得する

sudo apt -y install adb # 公式にはさらっと書いてあるが、大体入ってないと思う ^^;

adb connect 192.168.0.100

adb root

adb connect 192.168.0.100 # もう一度ログイン

adb pull /data/ssh/ssh\_host\_ed25519\_key my\_private\_key

取得した秘密鍵でログイン

chmod 600 my\_private\_key

ssh -i my\_private\_key root@192.168.0.100

adb connect 192.168.0.100

adb install helloWorld.apk

OTA；Over The Air

「ソフトウェアのアップデートを無線で行う」の意。

ようするにアップデート用のイメージらしい。

### Mirakurun

テレビ放送受信サーバー

（情報元）<https://medium.com/chinachu/c98v-raspberry-pi-4-49c9692f5c09>