

RaspberryPiと電子モジュールで作るMathematicaの実行環境

https://s3-ap-northeast-1.amazonaws.com/kobayashikorio/math_slides/slides.html

Ver.3.6

kobayashikorio@gmail.com

2018/6/1

RaspberryPiは、Mathematicaを外部のソフトウェアやハードウェア・モジュールと接続して、その機能を拡大するための、プラットフォームと考えることができる。

プラットフォームとしてのRaspberryPi、特にRaspberryPi Zeroは、その小型であること、安価であることなどの理由によって、Mathematicaと外部モジュールの組み合わせから生まれる、様々なアイデアの実現に、大いに役立つ。

一方、RaspberryPiをプラットフォームとした、システム開発をするために、Mathematicaを選ぶと以下のような利点がある。

- 開発環境が備わっている

Mathematicaの関数と看做することができるように構成できる。

- プロジェクトドキュメントを容易に作成できる

MathematicaのNotebookは実行環境であると同時に、プログラムの実行結果や、プログラムや考え方を示したテキスト、サポートのための図などを統合して、プロジェクト全体を表すドキュメントとして、構成することができる。

- 多様な関数が準備されている

StringJoin等の文字列と文字操作関数、Associate等のデータベース操作関数、Shift等のbit処理関数、などがこれにあたる。

- 各種の実行環境を実現できる。

外部モジュールを接続したRaspberryPiを構成し、これを動作させるにあたって、Mathematicaは以下のような実行環境を提供できる。

(1) 外部モジュールとの接続にあたって、Mathematicaのノートブック上で、外部モジュールを操作し、試験する作業台としての環境

- (2) きあがったプログラムを、Mathematicaカーネルだけで実行し、動作を確認する環境
- (3) プログラムをスクリプトとして、電源投入だけで自動的に実行させる、スタンドアローン環境
- (4) クラスタシステムをRaspberryPiで構成して、各ノードに計算コードを分配し、計算結果を回収する並列計算環境

■ この資料の目的

ここに取りまとめたのは、MathematicaとRaspberryPiの組合せによってアイデアを実現する過程で、筆者が調査あるいは開発した、ノウハウや、ソースコード、回路図、などの資料である。筆者は、我が国の多くの人が自らのアイデアを実現する時に、この資料が役立つことを願うものである。

コラム ノートを取るということ

自分を振り返って、プログラミングとは何かを考えてみると、それは何かの衝動、例えば、難しい問題に面と向かって、自分自身の能力を確かめてみたい、だとか、これは一体何を言わんとしているのだろうかとか、あの時は結局よく解らなくて知ったふりをしていただけなのだけれど、実際はどうなのだろうか、とかいう、流れの中、突き出た岩にひっかかっている枯れ草がどうにも、気になって仕方がない、というような、何かに引っ掛かって、すっきりしない気持ちを、解消したい、解消した時のあの、高揚感とも言える気持ちを味わいたい、というような想いが、背後にあるように思われる。

だが、そんな想いは、瞬間的なものであるもので、放っておくとあっという間に、消散してしまう。想いは少し変質してしまってもその新鮮さは失われるかも知れないが、それを固定化しなければ、ならないという客観的な視点もまた同時に存在する。

思いつき、アイデアとも呼ばれるこの想いを固定化するのが、ノートに書き付けることであろう。そしてMathematicaは、ノートに書き付けるべきテキスト、モノの写真、各種の図を、アイデアを実現する過程としてのプログラムと同時進行のかたちとして、NoteBookに残せる。プログラミングの意義を問いかけた、であろうS.Wolframが、その答えとしてNoteBookという環境を準備したことを、私は深く尊敬する。

■ 注

- * \$ echo 等の表記は、ターミナルから、shell commandを入力することを示す
- * 16進数を0x00～0xffと表す
- * 各章毎に個別にサンプルプログラムを実行できるようになっているので、一部のルーチンは、各サンプルプログラムで、重複している場合がある

* ケーブルの色分け

一般的に、赤色のケーブルはプラス、黒色はマイナスを示すのであるが、この資料内のモジュールに使うケーブルには、色別に以下の役割を与える。

赤: 5V

黄: 3.3V

黒: GND

青: I2C-SCL

緑: I2C-SDA

■ 改訂履歴

- 3.0: Collaboration with Arduino を追加 (2018/4/1)
- 3.5: Projectを追加 (2018/6/1)
- 3.6: Contents/Parallel Mathematica computingとコラム欄を追加 (2018/7/10)

A Mathematica environment composed of RaspberryPi and hardware modules

目次

はじめに

コラム ノートを取るということ

RaspberryPiのソフトウェアの設定

1. RaspberryPi Zero/Zero W のヘッドレス設定
2. Mathematicaの実行環境
3. Node.jsとのコラボレーション
4. RaspberryPi のクラスターを使ったMathematicaの並列計算
5. NFS によるネットワークブーツアップ
6. Cloudストレージの設定

コラム Mathematica on RaspberryPi Zero

RaspberryPiの外部インターフェースの仕組み

1. RaspberryPi Zeroのコネクタ類
2. GPIOピンの配置
3. I2C
4. SPI
5. GPIOの操作
6. PWM

コラム RaspberryPi Zeroというもの

Arduinoとのコラボレーション

1. ArduinoとIDEの設定
2. ATtiny85のピン操作
3. I2Cを通じたRaspberryPiとATtiny85の結合
4. Arduinoのオブジェクト指向プログラミング
5. ATtiny85のマスターとスレーブ
6. 開発ユーティリティ

コラム パーツとしてのArduino

センサー

1. Berry IMU モジュール
2. Dovewill (Adafruit) 10 DOF モジュール
3. GY-511 LSM303DLHC 方位モジュール
4. GY 521 MU6050 モジュール
5. カメラモジュール
6. 距離／近接モジュール
7. A/D 変換器

コラム センシング・モジュールはどうあるべきか

各種ディスプレイ

1. SSD1306 有機ディスプレイ
2. SSD1306のためのドットイメージ生成
3. ST7032 for LCD
4. HDMI接続のディスプレイ

コラム ディスプレイ・モジュールの扱い

アクチュエーター類

1. サーボ・モーター
2. ステッパー・モーター
3. DCモーター
4. DCモーター・ドライバー
5. スピーカー

コラム アクチュエータと外部パーツ

インジケータとスイッチ

1. インジケータ
2. スイッチ

3. タイマー
 4. カウンター
 5. DA変換器と音楽デバイス
 6. I/Oの拡張と電圧レベルシフター
- コラム スイッチとインジケーター

電源

1. 電源モジュール
 2. リチウム電池と充電
- コラム 電源の重要性は重要

各種パーツ

1. 配線材料
 2. パーツ等の購入先
- コラム 組み立てという人間的なふるまい

プロジェクトのマネジメント

RaspberryPiのソフトウェアの設定

RaspberryPi Zeroをディスプレイなしにセットアップする

RaspberryPiは、さまざまな手段で、ハードウェアモジュール、あるいは、他のRaspberryPiやネットワークシステムと結合できる。この章は、それらを実現するための、RaspberryPi本体のセットアップ、そして、RaspberryPiと外部機器を接続するための、ドライバーや関連ルーチンのセットアップについて説明する。

1. RaspberryPi Zero/Zero Wのヘッドレス設定

PCとRaspberryPiをUSB接続するだけで、プログラムできるようにする準備

■ 開発の方針

RaspberryPiには外部ディスプレイを接続するためのHDMI端子が備わっているし、USBを通じてキーボードとマウスを接続できるので、独立したコンピュータとして利用できる。しかし、一般的なPCとしてみた時には、CPU速度とメモリー量が限られているという制約があるのは否めない。そこで、RaspberryPiの小型であることや軽量であること、また安価であることを活かす方法を考える。

小型であることを追求するためには、RaspberryPiに本体より大きいようなマウスやキーボードを接続するのでは、するのではなく、逆に、できるだけ何も接続しないようにすることを考える。

RaspberryPi Zeroは、一本のUSBケーブルを、電源の供給とネットワークとの結合に共用できる。ソフトウェアの面から見れば、USB-Ethernet-Gadgetを利用し、PCとRaspberryPi Zeroを接続することになる。

実際には、RaspberryPi Zero上にVNCサーバーを立ち上げ、これにPCからVNC接続を行なうと、RaspberryPi ZeroとPCのみで、プログラムの開発や実行したり、各種外部モジュールをテストできる。

さらに、RaspberryPi Zero Wであれば、バッテリーを接続するだけで、PCとWiFiを通じて通信することができる。

Headless configure とは、RaspberryPi本体にはUSBケーブル以外何も接続せずに、RaspberryPiをセットアップする手順である。

ここでは、Linux raspberrypi 4.4.34を対象に、Macを用いて、Headless configureを行なう。確認した最新バージョンは、2017-09-07-raspbian-stretchである。

■ 準備するもの

- RaspberryPi ZeroもしくはRaspberryPi Zero W
- micro-SD 8GB
- usbケーブル

■ RaspberryPi Zero/Wのセットアップ手順

■ The Unarchiverユーティリティの準備

The UnarchiverユーティリティがMacにセットされていない場合には、最初にこのユーティリティをダウンロードする

1. "http://wakaba.c3.cx/s/apps/unarchiver.html"から、"The Unarchiver"をダウンロードする

■ イメージファイルのダウンロードと解凍

1. "https://www.raspberrypi.org/downloads/raspbian/Rasbian"からRASPBIAN STRETCH WITH DESKTOPをダウンロード
2. 明示的に"The Unarchiver"でunzipすると、20xx-xx-xx-raspbian-stretch.imgファイルがDesktopに置かれる

■ イメージファイルをSDに書き込む

SDカードリーダーをMacにセットした後、ターミナルからコマンドによって、SDにファイルをコピーする。

1. SDを探す（SDは"targetdisk"という名前であるとするが、おそらくは"dev/disk2"というような名前である）
2. このSDをアンマウントする->
3. 以下のコマンドにより、ディスクイメージ(おそらくは"xx-stretch.img"という名前である)を、SDにコピーする

```
mac$ > diskutil list
mac$ > diskutil unmountDisk /dev/"targetdisk"
mac$ > sudo dd bs = 1 m if = /Users/"your user id"/Desktop/"image.img" of = /dev/"targetdisk"
```

*SD書き込みには、40分程度の時間を要するので、途中経過をctrl-tでモニタすることもできる。

■ bootディスクを編集する

1. bootディスクの/config.txt をエディットする

SDのbootディスクを開き、nanoエディタで /config.txt を開き、ファイルの最後に、"dtoverlay=dwc2"という行を追加する。

```
mac$ > cd /Volumes/boot
mac$ > sudo nano ./config.txt
```

2. 同じディスクの ./cmdline.txt をエディットする

./cmdline.txt ファイルを開き、modules-load=dwc2,g_etherの1文をrootwaitの後に挿入する。改行せずスペース区切りで続けて記述すること。

```
mac$ > sudo nano ./cmdline.txt
```

3. 結果をみてる

```
cat ./cmdline.txt
```

```
dwc_otg.lpm_enable = 0 console = serial0, 115200
console = tty1 root = PARTUUID = 020 c3677 - 02 rootfstype = ext4
elevator = deadline fsck.repair = yes rootwait modules - load = dwc2, g_ether
```

```
quiet init = /usr/lib/raspi - config/init_resize.sh splash plymouth.ignore - serial -
consoles
```

4. /Volumes/bootディレクトリにsshを起動するためのダミーファイルを作る

```
mac$ > touch ssh
```

5. SDを取り外す

```
mac$ > cd ~
mac$ > diskutil unmountDisk /dev/" targetdisk"
```

■ RaspberryPiとMacを繋ぐ

1. SDをRaspberryPi Zeroにセットする
2. RaspberryPi Zeroの2つのマイクロUSBコネクタのうち、中央の側のコネクタを用いて、USBケーブルでMacと繋ぐ
3. LEDの点滅が開始された後、点灯された状態になるまで待つ

■ Macのセットアップ

1. RealVNCを準備

<https://www.realvnc.com/en/connect/download/vnc/> からRealVNC MacOSをダウンロードしインストールする。

2. Macにraspberrypiの持つネットワークを結びつける

- (1) インターネット共有->切
- (2) 共有する接続経路->Wi-Fi
- (3) ポート->RNDIS/Ethernet Gadget
- (4) インターネット共有->入

* Macが動作中に再度インターネット接続を試みても失敗する場合、Macを再起動してみる。

■ SSHでRaspberryPiを繋げてセットアップする

1. sshでraspberrypiにログインする

```
mac$>ssh pi@raspberrypi.local
pi$>password : raspberry
```

* 次のメッセージが出た場合にはyesを選択して継続する
The authenticity of host xxx can't be established. Are you sure...? yes

* 次のメッセージが出た場合には、"known_hosts"ファイルを削除する
@ WARNING: POSSIBLE DNS SPOOFING DETECTED! @

```
$ rm ~/.ssh/known_hosts
```

2. raspberrypiをセットアップするためにraspi-configを起動する

```
pi$>sudo raspi-config
```

- (1) Hostname -> 任意の名前を設定する
- (2) Localisation Options (tabでokを選択する)
 - Change Locale -> ja_JP.UTF-8 UTF-8
 - Change Timezone -> Asia -> Tokyo
- (3) Interfacing Options -> SSH,VNC,I2C -> Yes
- (4) Advanced Options
 - A1 Expand Filesystem -> ok ルートサイズを拡大
 - A5 Resolution -> 1024x768
- (5) Finish & reboot

■ Mac上のVNCviewerからRaspberryPiを設定する

1. Mac上のVNCviewerを起動し、raspberrypi上のVNCサーバーを探し、usernameとpasswordを入力して、VNCウィンドウを開く

```
VNC server -> raspberrypi.local::5900
```

```
Username-> pi
Password-> raspberry
```

* 最初にMac上のVNCviewerは、以下のメッセージを出力するので、実行継続を選択する
VNC Server not recognized -> continue

2. VNCviewerに現れるRaspberryPiの仮想画面を変更する場合

方法1. sudo raspi-config -> advanced で設定 (7.2-7-A5で設定した方法)

方法2. /boot/config.txt をeditし、以下をuncommentする。rebootすると拡大画面となる

```
#framebuffer_width=1024
#framebuffer_width=768
```

* Real VNCサーバーにリモート接続すると、それまでRaspberryPiに接続されていたローカルなマウスとキーボードは、使えなくなるが、Real VNCのview-onlyオプションにより、この状態を回避することもできる

3. RaspberryPiデスクトップの設定メニューで各種設定を行なう
 - (1) タイムゾーンを設定する
 - (2) i2Cでバイスをenableにする
4. Pi Zero WのWiFi設定
 - (1) ローカライゼーション -> 無線LANの国設定
 - (2) デスクトップのUSBメニューをクリック -> WiFiメニュー -> password設定

■ Reference

1. RaspberryPi usb gadget <https://learn.adafruit.com/turning-your-raspberry-pi-zero-into-a-usb-gadget/ethernet-gadget>
2. REAL VNC <https://www.realvnc.com/en/>

2. Mathematicaの実行環境

RaspberryPiに接続された外部モジュールを動作させるためには、幾つかの方法がある。Mathematicaに準備されている関数を用いる

方法の例としては以下があげられる

■ Mathematicaの関数を通じた制御

Mathematicaに準備されている関数を用いる—GPIOを制御する関数の例

```
DeviceConfigure["GPIO", n->dir]
```

■ Shellを通じてのコマンドの実行—独立したプロセスとして実行する

```
RunProcess["echo 'hello'"];
```

外部コマンドの実行と結果の取得が一つのプロセスとして実行される。

■ 外部コマンドのパス

自動実行スクリプトにおいて、外部コマンドが自動実行されるためには、コマンドのパスが正しく記述されている必要がある。COMMANDのパスを知るためには、以下を実行して調べ、結果をスクリプトに反映させる。

```
$ which COMMAND
```

■ パスの設定例

デスクトップファイル"file.txt"/home/pi/Desktop/file.txt"

ユーザーアプリケーション i2cset "/usr/sbin/i2cset -y 1 0x3c 0x00"

ユーザーアプリケーション gpio "/usr/bin/gpio -g mode 12 in"

■ シェルを通じた制御

Shellを通じて、コマンドを実行する方法

1. 独立したプロセスとして実行する

```
RunProcess["echo 'hello'"];
```

外部コマンドの実行と結果の取得が一つのプロセスとして実行される。

2. システムシェルを起動しておき、シェルと通信を行なう

```
process = StartProcess[$SystemShell];  
WriteLine[process, "cat test"];  
ReadLine[process]
```

システムシェルを起動して、シェルへのコマンド書き込みと、結果の読み込みを分離する方法である。実行時間はより短くなる。

バッファに蓄積されたデータをReadStringで読み込む手法については、具体例として後で説明される。

■ 制御のための関数類

外部モジュールを接続したRaspberryPiを制御し、動作させるためには、各種の関数が必要となるが、共通に使われる関数を準備しておく、プログラム開発が効率化できる。

■ 2の補数表現の変換

8bit*2 のレジスタの2の補数表現の変換ルーチンである

```
(* base 16 string to integer *)  
If[# > 32767, -65536 + #, #] &[FromDigits["8000", 16]]
```

```
-32768
```

```
(* integer(-32768~0~32767) to hex string *)
```

```
IntegerString[If[# < 0, 65536 + #, #], 16, 4] &[-32768]
```

```
"8000"
```

■ バイトー整数変換

センサーからの入力信号をMathematicaの入力に変換する関数。2個の16ビットレジスタからの値を、符号なし整数と符号付き整数に変換する関数が必要である。レジスタからの読み込み関数をrGet[r1, r2]としたとき、以下に定義される。

```
(* Get register value *)
rGet[reg_] := StringTrim[RunProcess[{"i2cget", "-y", "1", chipAdr, reg},
"StandardOutput"]];
rPut[reg_, value_] := StringTrim[RunProcess[{"i2cset", "-y", "1", chipAdr, reg,
value}, "StandardOutput"]];
(* Get unSigned and Short Integer *)
usGet[reg1_, reg2_] := (
  r = Flatten[{rGet[reg1], rGet[reg2]}];
  r1 = Map[FromDigits[#, 16] &, StringReplace[r, "0x" -> ""]];
  BitShiftLeft[r1[[1]], 8] + r1[[2]]
);
(* Get Signed Short Integer *)
ssGet[reg1_, reg2_] := (
  r = Flatten[{rGet[reg1], rGet[reg2]}];
  r1 = Map[FromDigits[#, 16] &, StringReplace[r, "0x" -> ""]];
  r2 = BitShiftLeft[r1[[1]], 8] + r1[[2]];
  If[r2 > 32767, -32768 + (r2 - 32768), r2]
);
```

■ カルマンフィルター

センサーからの出力には必ずノイズ分が含まれている。ノイズを減少させるためには、第一に、モジュールの動作パラメータを調整するべきであるが、それだけでは不十分な場合には、出力データに統計的処理を施すことが考えられる。

```
Kalman estimator is composed of
  Observation equation = Real value + Noise
  Conditional equation = Previous condition (value) + Noise
```

```
kalman[name_] := Module[{nVobsEq, nVcondEq, kGain, wPrevious, xPrevious, xForecast,
xEstimated},

  initPrev[name[xp_, wp_]] ^= (
    xPrevious = xp(* previous observed value *);
    wPrevious = wp(* previous variance of estimated values *);
  );
  setVariance[name[ceq_, oeq_]] ^= (
    nVcondEq = ceq(* noise variance of condition equation *);
    nVobsEq = oeq(* noise variance of observation equation *);
  );
  estimated[name[xObs_]] ^= (
    xForecast = xPrevious(* local level model *);
    wPrevious = wPrevious + nVcondEq;
    kGain = wPrevious/(wPrevious + nVobsEq);
```

```

wPrevious = (1 - kGain)*wPrevious;
xPrevious = xForecast + kGain*(xObs - xForecast);
Return[xPrevious]
);
];

```

```

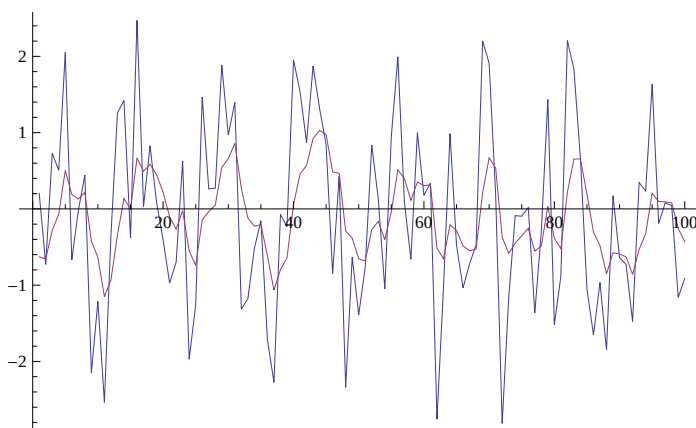
kalman[gyro];
initPrev[gyro[0, 0.01]];
setVariance[gyro[0.1, 1]]

```

```

u = Table[Sin[ $\frac{2 \pi i}{13.0}$ ] + RandomReal[NormalDistribution[0, 1]], {i, 100}];
v = Table[estimated[gyro[u[[i]]]], {i, 100}];
ListLinePlot[{u, v}]

```



■ オブジェクト指向プログラミング(Object Oriented Program : OOP)

外部モジュールが、複数の要素から成り立っている時、ある場合には、それぞれの要素が、内部状態によって独立な挙動できるように、モジュールを構成する必要がある。このとき、これらの制御ルーチンをOOPで構成した場合にどのような結果が得られるかは、検討に値する。例えば、

Control sub-systems -> Switches -> 4-ch touch switch TTP224の動作検証プログラムに、その例をみることができる。

■ Mathematica の自動実行とシャットダウン

RaspberryPiに様々な外部モジュールを結合したシステムは、スタンド・アローン・システムとして動作させたい場合があるだろう。この場合、RaspberryPiの起動後に、Mathematicaと応用プログラムを自動的に実行を開始し、また終了する必要がある。

スタンドアローンプログラムは、フロント・エンドを必要としない場合と、フロント・エンドを結果の表示などに用いるためにこれを必要とする場合とに分けられる。

■ 自動実行 1 - カーネルのみで実行する場合

カーネルがスクリプトを読んで、プログラムが実行される場合には、幾つかの自動起動方法が考えられる。ここでは、cronを用いた方法を説明する。

● Mathematica scriptを準備

1. Mathematicaのノートブックからテキスト形式のスクリプトを作成するために、ノートブックを実行していない状態で、Mathematicaのファイルメニューから、text形式で、デスクトップにそのまま保存する。
2. Text Editorでこのファイルを開き、以下の行をファイルの最初におく（Mathematica 11.0の場合）。

```
#!/opt/Wolfram/WolframEngine/11.0/Executables/math -script
```

3. Terminalを開き、デスクトップ・ディレクトリーに移動する。

```
$ cd ./Desktop
```

4. デスクトップにあるスクリプトに、実行可能フラグを以下のように設定する。スクリプトのファイル名をMATHSCRIPT.txtとした場合には、以下になるであろう。

```
$ chmod +x MATHSCRIPT.txt
```

5. 自動実行するかどうかを確かめるために、Terminalから以下のように実行してみる

```
$ ./MATHSCRIPT.txt
```

- cronを自動起動させる

1. \$ sudo nano /etc/rc.local として、/etc/rc.localをeditしてexit 0行の前に、以下の行を挿入する

```
/etc/init.d/cron start
```

2. \$ crontab -e によってcron tableをeditして、以下の行を付け加える

```
@reboot /home/pi/Desktop/MATHSCRIPT.txt >>/tmp/log 2>>/tmp/logerr
```

3. \$ crontab -l によって変更を確認する

*自動起動に問題が生じた場合、/tmp/logおよび/tmp/logerrを見ることで、その手掛かりが得られる。

■ 自動実行 2 - フロント・エンドを必要とする場合

ノートブックファイルを実行するためには、FrontEnd、すなわちデスクトップ環境が必要である。そこで、自動実行させるために、デスクトップ環境が整った後に、Mathematicaコマンドを発行しなければならない。例えばノートブックファイルtest.nb が、/home/piにある時、これを自動実行させるために、以下のような設定を行なう。

1. ノートブックから、test.nb 全体のセルを初期化セルに変更する

2. ノートブックから、初期化セル実行に係るオプションを設定する（セキュリティ強度が低下することに注意）

```
Format ->
Options for Global Preferences ->
Global Preferences ->
Notebook Options ->
Evaluation Options ->
InitializationCellWarning -> False
...
InitializationCellEvaluation -> True
```

3. ターミナルから、自動実行のセットアップの前に、test.nb が自動的に実行されるか確かめてみる


```
home/pi $ mathematica test.nb
```

4. ターミナルから、RaspberryPiのPIXEL Desktop環境でautostartさせるために、

```
/home/pi /. config/lxsession/LXDE // pi/autostartファイルに以下の行を挿入する
@/opt/Wolfram/WolframEngine/11.0/Executables/mathematica /home/pi/test.nb
```

5. PIXEL Desktop環境は、画面に変更のない場合はスリープするように設定されている。自動的にスリープしないようにする場合には、
ターミナルから、4.と同じ、

```
/home/pi /. config/lxsession/LXDE // pi/autostartファイルに行を追加して、以下のように変更
する
@xset s 0 0
@xset s noblank
@xset s noexpose
@xset dpms 0 0 0
@/opt/Wolfram/WolframEngine/11.0/Executables/mathematica /home/pi/test.nb
```

■ 外部コマンドのパス

自動実行スクリプトにおいて、外部コマンドが自動実行されるためには、コマンドのパスが正しく記述されている必要がある。例えば、COMMANDのパスを知るためには、以下を実行して調べ、結果をスクリプトに反映させる。

```
$ which COMMAND
```

パスの設定例

デスクトップファイル	"file.txt"/"/home/pi/Desktop/file.txt"
ユーザーアプリケーション i2cset	"/usr/sbin/i2cset -y 1 0x3c 0x00"
ユーザーアプリケーション gpio	"/usr/bin/gpio -g mode 12 in"

■ 自動終了

プログラムを自動実行させた場合には、これをどのように終了させるかについて、準備しておく必要がある。以下は、利用者にスイッチを押してもらうことを期待して、システムを終了、シャットダウンさせる例である。

- プルアップされたピンをGNDレベルに落とすことで、スイッチの押下を検出する方法

結線

Interrupt switch connection (BCM 07 pin is initially pull-upped)
Pin7 - tactswitch - GND

```
process = StartProcess[$SystemShell];
WriteLine[process, "echo '7' > /sys/class/gpio/export"];
WriteLine[process, "cat /sys/class/gpio/gpio7/value"];
If[ReadLine[process] == "0",
  $Epilog := Run["sudo shutdown -t 2"];
  Exit[]
];
```

■ Reference

1. https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf
2. <https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf>

3. <https://qiita.com/MoriKen/items/0c80ef75749977767b43>
4. <https://logics-of-blue.com/kalman-filter-concept/>
5. <https://logics-of-blue.com/kalman-filter-mle/>
6. <https://logics-of-blue.com/wp-content/uploads/2017/04/kf-concept-R.txt>
7. RaspberryPiのプログラムを自動実行する方法 -> <http://hendigi.karaage.xyz/2016/11/auto-boot/>
8. Cronを自動実行させる方法 -> <https://rahulmahale.wordpress.com/2014/09/03/solved-running-cron-job-at-reboot-on-raspberry-pi-in-debianwheezy-and-raspbian/>
9. Notebook Initialization -> <https://mathematica.stackexchange.com/questions/111378/notebook-initialization>
10. InitializationCellWarning -> <http://reference.wolfram.com/language/ref/InitializationCellWarning.html>
11. RaspberryPi3スリープ設定解除 -> <https://qiita.com/BearcubThaw/items/1a7a821ac9fc15659ebe>

3. Node.jsとのコラボレーション

Node.js によるGPIO 制御

■ rpio

rpio(node-rpio)はNode.jsを通じて外部モジュールを制御するルーチンである

Node.js環境を作っておき、外部モジュールの制御をNode.js環境のjsプログラムで実行し、制御変数をMathematicaのHTTP関数を用いて、Node.jsに引き渡す方法

■ Node.js をインストールする

```
pi@raspberrypi : ~ $ sudo apt install - y nodejs npm
pi@raspberrypi : ~ $ sudo npm cache clean
pi@raspberrypi : ~ $ sudo npm install n - g
pi@raspberrypi : ~ $ sudo n stable
pi@raspberrypi : ~ $ sudo npm update npm - g
pi@raspberrypi : ~ $ node - v
v4 .8 .2
pi@raspberrypi : ~ $ npm - v
5.7 .1
```

■ rpioをインストールする

```
raspberrypi : ~ $ sudo npm install rpio
```

■ Node.jsのスク립トでGPIOを制御してLEDを点滅させる

```
var rpio = require (' rpio');
var LED_PIN _ 1 = 13; // GPIO27
rpio.open (LED_PIN _ 1, rpio.OUTPUT, rpio.LOW);

for (var i = 0; i < 100; i++ ) {
  rpio.write (LED_PIN _ 1, rpio.HIGH);
  rpio.msleep (100);
  rpio.write (LED_PIN _ 1, rpio.LOW);
  rpio.msleep (100);
}
```

■ MathematicaとNode.jsとのcooperation

Node.jsでLEDの点滅を実行し、点滅周期をMathematicaから変更する。

1. webサーバーを立ち上げ、レスポンスを待ち受けている状態にしておく

```
//
var http = require (' http');
var url = require (' url');
var rpio = require (' rpio');

var Led = 13; // GPIO2
rpio.open (Led, rpio.OUTPUT, rpio.LOW);
var period = 100;
var isOn = false;
var blink;
//
timer = {
  start : function (period) {
    blink = setInterval ( function () {
      if (isOn) {
        rpio.open (Led, rpio.OUTPUT, rpio.LOW);
        isOn = false;
      }
      else {
        rpio.open (Led, rpio.OUTPUT, rpio.HIGH);
        isOn = true;
      }
    }, period);
    return blink;
  },

  change : function (newPeriod) {
    clearInterval (blink);

    blink = setInterval ( function () {
      if (isOn) {
        rpio.open (Led, rpio.OUTPUT, rpio.LOW);
        isOn = false;
      }
      else {
        rpio.open (Led, rpio.OUTPUT, rpio.HIGH);
        isOn = true;
      }
    }, newPeriod);
    return blink
  }
};
//
var blink = timer.start (period);

var server = http.createServer ();
server.on (' request', function (req, res) {
```

```

var obj = url.parse (req.url, true).query;

    period = Number (obj.val); // get value of val parameter
    console.log (period); // val value
    var blink = timer.change (period);

    res.writeHead (200, {' Content - Type' : ' text/plain'});
    res.write (' period changed');
    res.end ();
});
server.listen (3000);

```

2. サーバーを動かす（上記プログラムファイル名をprogramとする）

```
$ node program.js
```

サーバーはCtrl-cで停止する。

3. MathematicaのHTTP関数からリクエストを送信し、サーバーからのレスポンスを受け取る。valの数値を変えると、LEDの周期が変わる。

```

req = HTTPRequest["localhost:3000"]
URLRead[req]
URLExecute[req, {"val" -> 1000, "opt" -> "good"}]

```

■ rpio2

Events

rising - Triggering an event when GPIO signal rising

falling - Triggering an event when GPIO signal falling

change - Triggering an event when GPIO signal rising or falling

■ reference

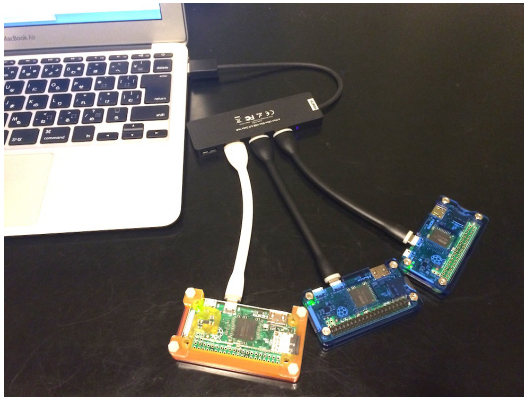
1. rpio2 -> <https://www.npmjs.com/package/rpio2>

4. RaspberryPi のクラスターを使った $\textit{Mathematica}$ の並列計算

一台のMathematica計算コントローラーに、複数のMathematica計算ノードを結合して、インタラクティブなMathematica計算クラスター環境を形成する

■ 特徴

- (1) OOPにより全体を構成する
- (2) 各計算サーバーはインスタンスであり、これにメッセージを配信して計算を実行する
- (3) インスタンスの要件はアソシエーションにより与える
- (4) **nc(netcat)**を用いたソケットよりコントローラーとノードとを結合する
- (5) 計算クラスターを構成するのは、各RaspberryPi Zeroノードである



3台のRaspberryPiを用いて計算クラスターを構成した例

■ 計算ノードの準備

- RaspberryPi Zeroを複数台用意し、これらをUSB Data Hubに繋ぐ。
- RaspberryPiに一意的なnode-nameを付ける (raspberrypi1, raspberrypi2 など)。
Macのhost-nameを調べる

```
uname -n
```

- RaspberryPiの各計算ノードにサーバープログラムをnanoエディタなどを用いてインストールする。プログラムファイル名はinitであり、内容は以下の通り(ソケット番号は各計算ノード毎に異なる。またchmod +x initしておくこと)。

```
$ cat init
While[True,
Run["nc -l 8000>input"];
temp=ReleaseHold[<<input];
temp >>output;
Run["nc kobayashi-no-MacBook-Air-2.local 8002<output"]
]
```

各RaspberryPi計算ノードのinitファイル中のポート番号は、互いにバッティングしないように付ける。
outputポートのノード名は、コントローラーのhost-nameであり、この場合"kobayashi-no-MacBook-Air-2.local"である。
sshで接続した、各RaspberryPi計算ノードにおいて、Mathematicaを立ち上げる。

```
$ wolfram <init &
```

設定をチェックするためにMacから以下を実行してみる(計算サーバーがraspberrypi.localの場合)。

```
$ nc -l 8002 >output | nc raspberrypi.local 8000 <<EOF
> 10!
> EOF
$ cat output
3628800
```

あるいは、名前付きパイプを作っておいて

```
$ mkfifo npipe
$ nc -l 8002 > npipe | nc raspberrypi.local 8000 << EOF | cat npipe
```

*LinkCreateはMathematica V10 <-> Mathematica RaspberryPiの間ではうまく働かない
*以下の方法は動作する

```
$ nc -l 2222
socket=SocketConnect["kobayashi-no-MacBook-Air-2.local:2222"];
str=ReadString[socket]
```

■ プログラムの実行

ディレクトリをセット

```
SetDirectory[NotebookDirectory[]];
```

ソケットの通信プロセスをスタティックに記述する

```
com1 = "nc -l 8002 >output1 lnc raspberrypi.local 8000 <input1";
com2 = "nc -l 9002 >output2 lnc raspberrypi1.local 9000 <input2";
com3 = "nc -l 9502 >output3 lnc raspberrypi2.local 9500 <input3";
```

オブジェクトの要件を連想により記述する

```
obj = {
  Association["name" -> node1, "comm" -> com1, "in" -> "input1", "out" -> "output1", "p"
-> {2000, 3500}],
  Association["name" -> node2, "comm" -> com2, "in" -> "input2", "out" -> "output2", "p"
-> {3501, 4000}],
  Association["name" -> node3, "comm" -> com3, "in" -> "input3", "out" -> "output3", "p"
-> {4000, 4500}];
```

計算サーパークラスの定義。この例では、メルセンヌ素数を計算する

```
new[nam_] := Module[{ps, pe},
  mersenneQ[n_] := PrimeQ[2^n - 1];
  setv[nam[{s_, e_}]] ^= {ps, pe} = {s, e};
  calc[nam] ^= Select[Range[ps, pe], mersenneQ]
];
```

インスタンスを生成する（連想を使う時には、純関数に連想キーの名前を結合できる）

```
Map[new[# name] &, obj];
```

objectを計算サーバーに書き込む（インスタンスのイメージがSaveでダンプされる）

```
Map[Save[# in, # name] &, obj];
Map[Run[# comm] &, obj];
```

各計算サーバーにおかれたインスタンスに対して、メッセージを送る（setv関数を実行する）

```
Map[Put[Hold@setv[# name[# p]], # in] &, obj];
Map[Run[# comm] &, obj];
```

関数（calc関数）を各計算サーバーで実行する
calc関数は実行時間がかかるのでプロセス化する

```
Map[Put[Hold@calc[# name], # in] &, obj];
proc = Map[StartProcess[{$SystemShell, "-c", # comm}] &, obj]
```

```
{ProcessObject[0], ProcessObject[1], ProcessObject[2]}
```

プロセスが終了するのを待ち合わせる（クラスターの各ノードの計算が終了したかどうか、以下のコマンドにより何回か確認する。必要な場合には、待ち合わせをプログラム化すべきである）

```
Map[ProcessStatus[#] &, proc]
```

```
{"Finished", "Finished", "Finished"}
```

結果を読み込む

```
Map[FilePrint[# out] &, obj];
```

```
{2203, 2281, 3217}
{}
{4253, 4423}
```

この後も、インスタンスを生成することなく、計算範囲を変更して（連想キーp）、計算サーバーによる並列計算を何度も実行できる。

メルセンヌ素数は、...,1279,2203,2281,3217,4253,4423,9689,...である。

5. NFS によるネットワークブーツアップ

RaspberryPiをSDカードなしでブーツさせる手順である。root file systemは、RaspberryPiに接続されたUSB diskあるいはNFSから得ることができる

■ in preparation

■ Reference

1. boot without SD

<https://jyn.jp/raspberrypi-usb-only-boot/>

2. network boot

<https://qiita.com/khayate/items/c69ab816fe0cdf54ff64>

f64

3. Raspberry PiをNFSルートファイルシステムで使う

<http://www.nspl.co.jp/content/raspberry-piをnfsルートファイルシステムで使う>

6. Cloudストレージの設定

RaspberryPiにCloud storageを接続する手順

■ in preparation

<https://www.sbprojects.net/projects/raspberrypi/webdav.php> ->

<https://dbpro.xyz/2749> -> Raspberry Pi オンラインストレージを利用する

コラム Mathematica on RaspberryPi Zero

PCのためのMathematicaは、大変に高価なソフトウェアである。とても、単なる趣味で購入するのは憚られる。なので、多くのMathematicaユーザは、学校単位で導入されたパッケージを使うか、企業の開発部門でしか導入されないであろうパッケージを使うか、しないと、Mathematicaに触れる機会さえないであろう。

Mathematicaの開発元であるWolframも、このままでは、ユーザー数が先細りになることを懸念して、ユーザーの拡大を狙って、RaspberryPiに標準で、これを搭載することにしたのだと思われる。しかしながら、インターネット上に流れる、RaspberryPiに関する記事は、よく見るけれども、RaspberryPi上のMathematicaに言及した記事は、ずっと少ない割合であるように思われる。

計算速度の点からは、PCに比べて、とうてい及ばないRaspberryPiでは、Mathematicaは使いものになるまい、という意見もあるだろうし、GPUに圧倒的な差があるから、Mathematicaの得意なグラフィックスを活用するのは、相対的に小さな画面しか扱えないRaspberryPiには無理であろうという声もあるに違いない。

しかし、見方を変えれば、そもそも高速な動作を期待するわけではないRaspberryPiの、さらにその上に乗るMathematicaに高速な動作を求めるのは、木に縁りて魚を求めるが如く、あるいはARMv6にXeon E5を望むが如く、誰もがそれを知っているのである。では、RaspberryPi上のMathematicaに何を求めるべきだろう。計算速度でもなく、描画速度でもないもの。

それが、アルゴリズムの追求と、アイデアの確認のためのワークベンチであると。どちらも、計算速度や描画速度はそれほど重要とは目されない。確かに、RaspberryPi上のMathematicaは、お世辞にも速いとは言えない。しかし、アルゴリズムの追求も、アイデアの確認も、一番の要は、自分自身との対話であるから、速度よりも、正確に少しずつ積み上げていく思考の手助けとなる、Mathematicaの有効性は、確かなものとなることができる。

RaspberryPiの外部インターフェースの仕組み

RaspberryPi Zeroのハードウェア・セットアップ、ピン配置、ハードウェアインターフェース等

RaspberryPi Zero/Zero Wと外部機器を接続するためには、GPIOピンのRead Writeを用いる方法、特定のピンにアサインされているI2C機能を使う方法、PWMによる方法等が考えられる。複雑な機能を持つモジュールの制御には、I2CやSPIが便利であるし、簡単なLEDの点滅やスイッチ入力等にはGPIOピンを直接制御する方法が適当である。またサーボモーター等の制御にはPWMが必須となる。

1. RaspberryPi Zeroのコネクタ類

外部とのコネクタ接続

■ ボードに準備されているコネクタ類

RaspberryPi ZeroはBroadcom BCM2835チップセットを用いており、ボード上には、以下のようなコネクタが準備されている（Ref.4）。

■ USB コネクタ

RaspberryPi Zeroには、USBコネクタが2個用意されている。チップが装着されている面から見て、中央がUSB (J1)コネクタ、端側がPWR IN (J10)コネクタである。入力電圧として5Vが与えられた場合、内部のPAM2306 DC/DCコンバータで3.3Vと1.8Vを作り出している。

■ HDMI コネクタ

J6コネクタにはHDMIが接続される。

■ カメラコネクタ

J12コネクタにはカメラが接続される。

■ DNI コネクタ

J8コネクタはGPIOとその他の入出力が接続されている。

RaspberryPi data sheetにあるように、**RaspberryPiのGPIOは3.3Vで駆動され、その入力の最大値は4.1V**であるので、5Vが供給されている外部回路と直接的に接続をすることはできない。この場合には、level shifterを介在させなければならない。

■ Run コネクタ

J2コネクタは、RUN-pinと隣接するGNDに接続されている。RUN-pinは3.3Vにプルアップされている。RUNをGNDに接続後リリースすると、BCM2837はブートされる。すなわち、J2コネクタにタクトスイッチを取付けると、シャットダウン後の電源が接続されている状態で、リブートすることができる。J2はGPIO2と4に隣接している。

■ TV Out コネクタ

J11コネクタは、コンポジットビデオ出力(PALもしくはNTSC)が接続されている。接続されたデバイスが検出されるとRaspberryPiは適切な設定を出力しようとする。

■ Reference

1. RaspberryPiの概要~Wikipedia -> https://ja.wikipedia.org/wiki/Raspberry_Pi
2. RaspberryPi CM data sheet -> https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM_1p0.pdf
3. RaspberryPi Zero schematic -> <https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md>
4. on the composite video -> <https://chipprogrammers.wordpress.com/raspberry-pi/>

2. GPIOピンの配置

外部接続ピンの詳細

■ ピンの配置と割り当て

RaspberryPi Zeroの外部接続ピンは、ボード上のICが見えるように、かつコネクタを左側にして置いて、以下の図のように構成されている。GPIOは3.3Vで駆動され、その入力の最大値は4.1Vである。外部モジュールの電源電圧が5Vである場合、モジュールとGPIOピンを直接的に接続するべきではない。

	Initial state	WiringPI Name	BCM Name			BCM Name	WiringPI Name	GPIO-	Initial state
			3.3V	1	2	5V			
1		SDA	BCM02	3	4	5V			
1		SCL	BCM03	5	6	Gnd			
1		GPIO_GCLK	BCM04	7	8	BCM14	TXD0		
			Gnd	9	10	BCM15	RXD0		
0		GPIO_GEN0	BCM17	11	12	BCM18	GEN1(PWM0)		0
0		GPIO_GEN2	BCM27	13	14	Gnd			
0		GPIO_GEN3	BCM22	15	16	BCM23	GPIO_GEN4		0
			3.3V	17	18	BCM24	GPIO_GEN5		0
0		SPI_MOSI	BCM10	19	20	Gnd			
0		SPI_MOSO	BCM09	21	22	BCM25	GPIO_GEN6		0
0		SPI_CLK	BCM11	23	24	BCM08	SPI_CE0_N		1
			Gnd	25	26	BCM07	SPI_CE1_N		1
			ID_SD	27	28	ID_SC			
1		GPIO_GEN21	BCM05	29	30	Gnd			
1		GPIO_GEN22	BCM06	31	32	BCM12	GPIO_GEN26		0
0		GPIO_GEN23	BCM13	33	34	Gnd			
0		GPIO_GEN24	BCM19	35	36	BCM16	GPIO_GEN27		0
0		GPIO_GEN25	BCM26	37	38	BCM20	GPIO_GEN28		0
			Gnd	39	40	BCM21	GPIO_GEN29		0

pin

pin

GPIO pin end

■ ピンの名称

GPIOピンはBCM(Broadcom)ピンと呼ばれることがある。GPIOピンの初期状態(Initial State)の1は3.3Vにプルアップされている状態、0はプルダウンされている状態を示す。BCMはRaspberryPiシリーズがBroadcom社の BCM2835/6を用いていることを示している。

Initial stateが1にプルアップされているピンと、対向しているピンがGNDあるいは0にプルダウンされている場合には、タクトスイッチを取付けることが容易である。

■ ピン電流

GPIOピンのSink電流は17mA(VDD_IO=2.7V, VO=0.4V)、Source電流は16mA(VDD_IO=2.7V, VO=2.3V)である。

■ ケーブルの色分け

赤: 5V
黄: 3.3V
黒: GND
緑: I2C-SDA
青: I2C-SCL

■ Reference

1. RaspberryPi pinout -> <https://pinout.xyz>
2. RaspberryPi pinnumber -> https://www.sunfounder.com/learn/Super_Kit_V2_for_RaspberryPi/raspberry-pi-pin-number-introduction.html

3. I2C

制御コマンドとプログラム

RaspberryPiは様々な外部モジュールを、多様な方式で接続できる。I2C(Inter-Integrated Circuit)はその接続方式の一つであるが、I2C機能を持つ多様なモジュールが展開されている。モジュールには複数のI2C機能を持つチップが複合化されているものがある。この場合はチップに与えられたI2Cスレーブアドレスによって、チップを区別する。さらに、Arduinoはソフトウェア的にI2Cスレーブとなることができるので、RaspberryPiとArduinoをI2Cにより統合することができる。

■ I2Cによるデバイスの接続

I2Cは、2本の双方向信号線、シリアルデータ (SDA) とシリアルクロック (SCL)で構成される。供給される電圧は3.3Vもしくは5Vである。

ただし、RaspberryPiのGPIOは3.3Vで駆動され、その入力の上限度は4.1Vであるので、5Vが供給されている外部回路との間とI2C接続をすることはできない。この場合には、I2C level shifterを介在させなければならない。

■ I2Cコマンド

RaspberryPiにはI2Cコマンドが組み込まれている。

```
$ sudo i2cdetect -y 1          // 接続されているデバイスのアドレスをスキャンする
$ sudo i2cget -y 1 0x1e 0x20    // アドレス0x1eを持つデバイスの0x20レジスタを読み込む
$ sudo i2cget -y 1 0x1e 0x20 w  // アドレス0x1eを持つデバイスの0x20レジスタと0x21レジスタから1wordを読み込む
$ sudo i2cset -y 1 0x1e 0x20 0x00 // アドレス0x1eを持つデバイスの0x20レジスタに0x00を書き込む
```

```
$ sudo i2cset -y 1 0x1e 0x20 0x00 0x01 0x02 i // 一群のデータを書き込む
```

例えば、OzzMaker BerryIMU モジュールは三つの独立したI2Cチップを内蔵しており、三つのI2Cアドレスが確認できる。

```
$ sudo i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00 : -- -- -- -- -- -- -- -- -- -- -- -- -- --
10 : -- -- -- -- -- -- -- -- -- -- -- -- -- 1e --
20 : -- -- -- -- -- -- -- -- -- -- -- -- -- --
30 : -- -- -- -- -- -- -- -- -- -- -- -- -- --
40 : -- -- -- -- -- -- -- -- -- -- -- -- -- --
50 : -- -- -- -- -- -- -- -- -- -- -- -- -- --
60 : -- -- -- -- -- -- -- -- -- -- 6a -- -- -- --
70 : -- -- -- -- -- -- -- -- 77
```

■ I2C プログラムの例

i2Cに接続されたデバイスからの出力の読み取りには、RunProcessを用いる方法、StartProcessとWriteLine-ReadLineを用いる方法、および、StartProcessとWriteLine-ReadStringを用いる方法がある。読み取り速度は後者の方が高いのであるが、後者は読み取りに失敗する場合がある。読み取りに失敗して、関数が待ち続ける状態に陥ることを防ぐためには、TimeConstraintを使う必要がある。

■ using “RunProcess”

```
(* RunProces style is simple but slow*)
Timing[Table[
  RunProcess[{"i2cget","-y","1","0x1e","0x02"},"StandardOutput"],
  {100}];]
{20.01`,Null}
```

■ using “StartProcess”, “ReadLine”, or “ReadString”

```
process=StartProcess[$SystemShell];

(* read line from process work well, and time-constraint may be nessessary for some devices *)
Timing[Table[WriteLine[process,"i2cget -y 1 0x1e 0x02"];
  ReadLine[process,TimeConstraint->0.1],{100}];]
{1.38`,Null}

(* read string from the process doesn' work, and must be aborted *)
WriteLine[process,"i2cget -y 1 0x1e 0x02"];
ReadString[process]
$Aborted

(* read string work with end of buffer, but waiting time is necessary but fast *)
Timing[Table[WriteLine[process,"i2cget -y 1 0x1e 0x02"];
  Pause[0.03];a=ReadString[process,EndOfBuffer];
  If[a=="",Print[error]],{100}];]
{0.5`,Null}
```

■ reading and writing a byte to I2C connected sensor register

```
process=StartProcess[$SystemShell];
chipAdr="0x1e";rGet[r_]:= (WriteLine[process,StringRiffle[{"i2cget -y 1",chipAdr,r}]]);
ReadLine[process]);
rPut[r_,d_]:=WriteLine[process,StringRiffle[{"i2cset -y 1",chipAdr,r,d}]]];
rPut["0x02","0x00"];
rGet["0x02"]
```

■ ReadLineの利用

StartProcessでプロセスを立ち上げた後で、レジスタを指定するコマンドをチップに連続的に送出した後

で、ReadLineを使って、一括してデータを読み込むことができる。ただし、この場合には、チップ内のレジスタの読み込みと書き込み時間を考慮して、Pauseを使う必要がある場合がある。この場合Pauseの時間は、ヒューリスティックに決定しなければならない。

■ ReadStringの利用

ある場合にはReadLineに代えて、ReadStringを使うことで、読み込み速度を上げることができる。このために、Sow,Reepを使って非同期的にレジスタの指定とレジスタの読み込みを行なう。しかし、ReadStringにEndOfBufferオプションを付けてより読み込むとしても、モジュールチップ内のバッファへの書き込みタイミングと同期している訳ではない。その結果、読み込みデータのリークが生じる。これを防ぐためにPauseを使わなければならないが、Pauseの設定値はリークが0であることを保証できないことに注意する必要がある。

■ I2Cの制限

I2Cインターフェースを持つICのアドレスは一般的には固定されているので、同じ供給元から供給されるI2Cモジュールを複数、同時に使うことはできないのであるが、Adafruit TCA9548A I2C Multiplexerはこの問題を回避できる。

■ Reference

- | | |
|--|---|
| 1. I2C commands | http://www.hogetan.net/note/memo/pi_i2ccmd.html |
| 2. LSM9DS0 data sheet | http://ozzmaker.com/wp-content/uploads/2014/12/LSM9DS0.pdf |
| 3. How to enable i2c | http://ozzmaker.com/i2c/ |
| 4. LTC4316 data sheet | http://www.linear-tech.co.jp/product/LTC4316 |
| 5. i2xdetect(8) - Linux man page | https://linux.die.net/man/8/i2cdetect |
| 6. SPI and I2C tutorial and-i2c-tutorial | https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial |
| 7. i2c-0の利用 | http://www.denshi.club/pc/raspi/raspberry-pi-i2c.html |

■ I2Cバスの拡大

RaspberryPiにはI2Cバスとして、初期設定済みのi2c-1の他にi2c-0が準備されている。ただし、i2c-0は、Hardware Added On Top (HAT)アッドオンボードをRaspberryPiボードに接続した時に、HATのEEPROMを読みとって、適切な設定を行なうために用意されているものである。設計ノートには、HATのEEPROMの読み取り以外に使用してはならないと記述されている。

```
$ sudo nano /boot/config.txt // "dtparam=i2c_vc=on" の行を付け加え、rebootする。
```

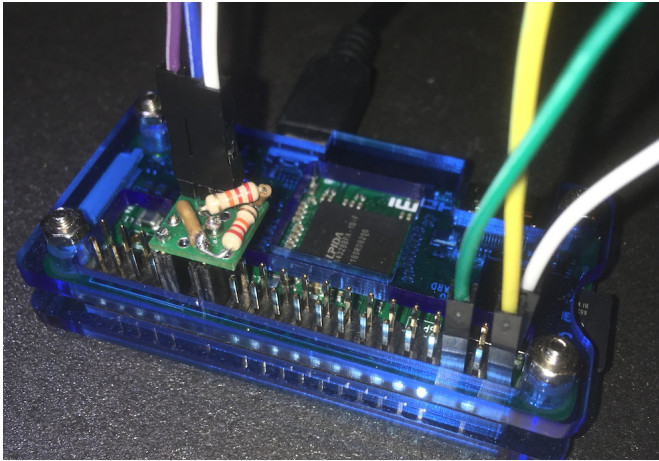
I2Cバスにi2c-0が付け加えられた。

```
$ sudo i2cdetect -l
i2c - 1 i2c      bcm2835 I2C adapter      I2C adapter
i2c - 0 i2c      bcm2835 I2C adapter      I2C adapter
```

■ I2C - 0 アダプター

これによってpin27, 28 にi2c - 0 がアサインされる。ただし、SDA、SCLピンは3.3 Vへプルアップする必要がある。pin27, 28 にプルアップ抵抗を接続するためには、アダプターが必要である。以下の試作アダプターの結線は以下の通りである。

```
pin27 -> 22 k ohm -> adptrPin1 -> 3.3 V
pin28 -> 22 k ohm -> adptrPin1 -> 3.3 V
pin27 -> adptrPin2 -> SDA
pin28 -> adptrPin3 -> SCL
```



試作アダプターはプルアップ抵抗を搭載している

アダプターを介して、i2c-0バスに、0x63というアドレスを持つ、i2cデバイス（例として DRV8830モータードライバー）を接続して、それを検出した結果である。

```
$ sudo i2cdetect -y 0
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00 : -- -- -- -- -- -- -- -- -- -- -- -- --
10 : -- -- -- -- -- -- -- -- -- -- -- -- --
20 : -- -- -- -- -- -- -- -- -- -- -- -- --
30 : -- -- -- -- -- -- -- -- -- -- -- -- --
40 : -- -- -- -- -- -- -- -- -- -- -- -- --
50 : -- -- -- -- -- -- -- -- -- -- -- -- --
60 : -- -- -- 63 -- -- -- -- -- -- -- -- --
70 : -- -- -- -- -- -- -- -- -- -- -- -- --
```

4. SPI

■ in preparation

spinel ユーティリティは、シェルからSPIモジュールを制御することができる。

■ Reference

1. SPI command utility <http://ipsolutionscorp.com/raspberry-pi-spi-utility/>
2. LTC1694-1 i2c accelerator <http://cds.linear.com/docs/en/datasheet/16941fa.pdf>

5. GPIOの操作

RaspberryPiのGPIO(general purpose input/ouput)ピンを直接操作することもできる。以下の方法を実行するためのコマンド類はRaspberryPiにインストールされている。

■ 仮想ファイルへの読み書きによるピン操作

(ユーザー権限の実行が許可されている)

```
//gpioの仮想ファイルを作成するexportにピン18を操作するために、gpio18ファイル作成を依頼する
$ echo 18 > /sys/class/gpio/export
```

```
//gpio18ファイルの値を得る
$ cat /sys/class/gpio/gpio18/value

//gpio18ピンを出力用に設定する
$ echo out > /sys/class/gpio/gpio18/direction
//gpio18ピンの出力を"1"に設定する (3.3Vにする)
$ echo 1 > /sys/class/gpio/gpio18/value
//gpio18ピンの出力を"0"に設定する (0Vにする)
$ echo 0 > /sys/class/gpio/gpio18/value
//gpio18ピンの設定を解除する
$ echo 18 > /sys/class/gpio/unexport
```

■ WiringPi (Gordons Projects) によるピン操作

```
$ gpio -v           //バージョン表示
$ gpio readall      //アクセス可能なピンの状態を表示

$ gpio -g mode 14 out //ピンを出力 (out) に設定して
$ gpio -g write 14 0  //0レベルを出力

$ gpio -g mode 14 out //ピンを出力 (out) に設定して
$ gpio -g write 14 1  //1レベル (High=3.3V) を出力

$ gpio -g mode 12 in  //ピンを入力 (in) に設定して
$ gpio -g read 12     //ピンのレベルを読み込む
```

WiringPiでは、WiringPiで定義されているGPIOピン番号が使われる。BCMピン番号を使う場合には、-gオプションをつける。

■ 仮想ファイルへのOpenWrite書き込みによるピン操作

```
vf = OpenWrite["/sys/class/gpio/gpio4/value"];
Do[Write[vf, 1]; Write[vf, 0], {1000}];
```

バイナリ書き込みにすると若干速くなる。

```
bvf = OpenWrite["/sys/class/gpio/gpio4/value", BinaryFormat -> True];
Do[ BinaryWrite[bvf, 49];
[bvf, 48], {1000}];
```

■ Mathematicaに準備されている関数によるピン操作

この方法は利用が簡単であるが、実行開始まで時間がかかることがある。あるいは、RaspberryPi Zeroでは実行できない。

```
DeviceConfigure["GPIO", n -> dir]
```

■ Node.js によるピン操作

エディタでプログラムを作成する。内容は以下の通り。

```
var rpio = require('rpio');

var LED_PIN_1 = 15; // GPIO22

rpio.open(LED_PIN_1, rpio.OUTPUT, rpio.LOW);
```

```
for (var i = 0; i < 10; i++) {
  rpio.write(LED_PIN_1, rpio.HIGH);
  rpio.msleep(500);
  rpio.write(LED_PIN_1, rpio.LOW);
  rpio.msleep(500);
}
```

プログラムファイル名が、sample2.js であるとする、以下により実行される。

```
$ node program2.js
```

■ 参考URL

- | | |
|-------------------------------------|---|
| 1. Wiring Pi library | http://wiringpi.com/the-gpio-utility/ |
| 2. Gordon projects | https://projects.drogon.net/raspberry-pi/wiringpi/ |
| 3. Gordon projects GPIO utility | https://projects.drogon.net/raspberry-pi/wiringpi/the-gpio-utility/ |
| 4. Control PWM frequency | https://raspberrypi.stackexchange.com/questions/4906/control-hardware-pwm-frequency |
| 5. Using GPIO with Wolfram language | http://community.wolfram.com/groups/-/m/t/157473 |

6. PWM

■ PWMのセットアップ

PWM (Pulse Wide Modulation) は、特定の周波数のパルス列のパルスがオンである時間を変化させて、情報を伝える方法である。デジタルサーボモーターは、このオンとオフ（マークとスペースとも呼ぶ）の比率（デューティ比）によって、モータ軸の回転角度が指定できる。LEDをPWM方式で点灯すれば、デューティ比を変化させて、見かけ上明るさを変えることができる。BCM2385/6 は、2つのPWMチャンネルを持っており、RaspberryPi Zeroはその両方を利用できる。

■ PWMピン

デフォルト状態で、BCM18ピンが、PWM出力として使える。このピンはWiringPiでの指定では1である。
このピンをPWMモードに切り替える。

```
$ gpio mode 1 pwm
```

■ PWMのモード

PWMにはBALANCEモードとMARK_SETモードの2つがあり、PWMモードに切り替えた時はBALANCEモードになっているので、MARK:SETモードに変更する。

```
$ gpio pwm_ms
```

（gpio mode 1 pwmコマンドを実行すると、Balanceモードに戻ってしまうので、その後に実行する。）

■ 周波数の設定

PWMの周波数は、CLOCK値とRANGE値で決まる。システムの周波数は19.2MHzであるので、PWM周波数 = 19.2MHz / CLOCK値 / RANGE値となる。

```
$ gpio pwmc CLOCK CLOCKの範囲は 2 - 4095
$ gpio pwmr RANGE RANGEの範囲は 1 - 4096
```

以上から周波数の最大値と最小値は
 $19.2\text{MHz} / 2 / 1 = 9.1\text{MHz}$
 $19.2\text{MHz} / 4095 / 4096 = 1.1446\text{Hz}$
 となる。

■ デューティ比の設定

デューティ比dは、 $d = \text{VALUE} / \text{RANGE}$ で決定される。VALUEは以下のように設定する。

`$ gpio pwm 1 VALUE` //VALUEの値はRANGEより小さい値とする

■ 例

最も低い周波数のクロックを出力してみる

```
$ gpio mode 1 pwm //GPIO18 (BCM) ピンをPWM出力に設定
$ gpio pwm-ms //mark-spaceモード
$ gpio pwmc 4095 //CLOCKを4095 (2 ~4095)
$ gpio pwmr 4096 //RANGEを4096 (~4096)
$ gpio pwm 1 2048 //valueを2048としてデューティ比2048/4096= 0.5に設定
```

出力の周波数は、1.1446Hzとなるので、テスター等で変化を見ることができる。

■ 2-channels PWM

■ 2チャンネルのPWMを利用する方法

/boot/config.txtに以下を追記してBCM18とBCM19を出力チャンネルに設定する (Ref.8)
`dtoverlay=pwm-2chan,pin=18,func=2,pin2=19,func2=2`

その結果overlayが実行されてpwm_bcm2835がロードされる

```
$ lsmod | grep pwm
pwm_bcm2835      3254 0
```

2つのchannelが生成された

```
$ cat /sys/class/pwm/pwmchip0/npwm
2
```

rootになる

```
$ sudo su -
```

当該ディレクトリーに移動して

```
root@raspberrypi3:~# cd /sys/class/pwm/pwmchip0
```

チャンネル0を設定する (周期0.8sec, mark0.4sec)

```
root@raspberrypi3:/sys/class/pwm/pwmchip0# echo 0 > export
root@raspberrypi3:/sys/class/pwm/pwmchip0# echo 800000000 > pwm0/period
root@raspberrypi3:/sys/class/pwm/pwmchip0# echo 400000000 > pwm0/duty_cycle
root@raspberrypi3:/sys/class/pwm/pwmchip0# echo 1 > pwm0/enable
```

チャンネル1を設定する (周期0.4sec, mark0.2sec)

```
root@raspberrypi3:/sys/class/pwm/pwmchip0# echo 1 > export
root@raspberrypi3:/sys/class/pwm/pwmchip0# echo 400000000 > pwm1/period
root@raspberrypi3:/sys/class/pwm/pwmchip0# echo 200000000 > pwm1/duty_cycle
root@raspberrypi3:/sys/class/pwm/pwmchip0# echo 1 > pwm1/enable
```

実行中にチャンネル0のduty_cycleを変更できる

```
root@raspberrypi3:/sys/class/pwm/pwmchip0# echo 100000000 > pwm0/duty_cycle
```

■ 参考URL

1. <http://kutajin.blog101.fc2.com/blog-entry-279.html>
2. <http://raspberrypi.stackexchange.com/questions/4906/control-hardware-pwm-frequency/9725#9725>
3. <https://raspberrypi.stackexchange.com/questions/4906/control-hardware-pwm-frequency>
4. <http://www.jumpnowtek.com/rpi/Using-the-Raspberry-Pi-Hardware-PWM-timers.html>
5. <https://learn.adafruit.com/adding-basic-audio-output-to-raspberry-pi-zero/pi-zero-pwm-audio>
6. <https://dbpro.xyz/3711>
7. https://manual.atmark-techno.com/armadillo-4x0/armadillo-400_series_software_manual_ja-2.0.0/ch09.html
8. <https://github.com/raspberrypi/firmware/blob/master/boot/overlays/README>

コラム RaspberryPi Zeroというもの

RaspberryPiはワンボードコンピュータのひとつとして出発したので、ディスプレイとキーボードとマウス、そしてイーサネットと繋げることを前提に設計されている。ところが、RaspberryPi Zero/Wだけは、ホストとして想定されるUSB-Aではなく、USB On-The-Goを想定したUSB Micro-Bが採用されている。そして、イーサネットアダプタは存在しない。

おそらく、RaspberryPi Zeroは、コストの低減を最大の目標として、その結果としての教育現場へのさらなる浸透を図って設計されたものであろう。そして、その結果は、他のRaspberryPiの1/5~1/6の\$5という戦略的な価格となって、表れているのである。

だが、VNC(Virtual Network Computing)をRaspberryPiに適用した時、RaspberryPi Zero/W 回りには新たな様相が表れてくる。それは、アイデアを実現するためのプラットフォームとしてのRaspberryPi Zeroである。アイデアというのは、確立されたものではないので、それを実現する過程で、様々な予期せぬ事態が発生するし、そして予期せぬ事態とは、思わぬ出費であり、物理的な取りまとめの失敗であり、制御機器の焼損やダメージである。

このとき、RaspberryPi Zeroの、使い捨てにできる価格、どこにでも押し込めるサイズ、そしてVNCによる仮想プログラミング環境が、強力にその意義を現すのである。使い捨てにできる程の価格の優位さを利用した、クラスタリング、どこにでも押し込めるサイズの優位さを利用した、他モジュールとの抱き合わせ等が、アイデアの展開に、どれほど助けになるかは、設計意図とは異なるかも知れないが、称讃されるべきものであろう。

Arduinoとのコラボレーション

Atmel ATtiny85 と RaspberryPi ZeroをI2Cで統合する (Ver.3.0から)

Arduinoとは、Atmel社のAVRチップ、それをういたボード、C++ likeな開発言語、および開発環境など、全体の総称である。Arduino(AVRチップとArduino言語)は、割り込みなどを用いた、リアルタイム制御を実行できる。一方、RaspberryPi上のMathematicaは、複雑なシステムの構築が可能であるものの、リアルタイム処理は不得手である。そこで、両者を統合すれば、新たな展開が期待できる。ここでは、Arduino/ATtiny85を対象にした各種システムについて説明する。ハードウェアとしては、ATtiny85を搭載したDigisparkを主に用いる。

1. ArduinoとIDEの設定

■ MacへのArduino IDEの導入

ArduinoはAVRのプロセッサとその開発システム環境を言うのであるが、この統合された開発システム環境 (IDE: Integrated Development Environment)そのものもArduinoと呼ぶ場合がある。以下はMacを例としたArduino IDEの設定と実行手順である。

1. Arduino IDEをダウンロードし、展開した後に開発環境を立ち上げる
2. 環境設定 -> 追加のボードマネージャにURL：
http://digistump.com/package_digistump_index.jsonを設定する
3. tool -> board -> Digispark(Default-16.5mHz)として、ターゲットボードをATtiny85に設定する
4. tool->シリアルポート -> /dev/cu.Bluetooth-Incoming-portに設定する
5. IDEのプログラム開発ウィンドウを用いて、プログラムを開発する
6. 画面上のボタンをクリックして、開発したプログラムをコンパイルしダウンロードする
7. Arduinoプロセッサへのダウンロード待ち受け状態に移行する
8. MacのUSBにUSBハブを取付け、その先にArduinoプロセッサを接続する
9. プログラムのダウンロードが完了するのを待つ
10. Arduinoがブーツアップし、ダウンロードされたプログラムが走り始める

ここで、MacにUSBハブを取付けたのは、Arduinoの書き込み操作のタイミングを調整するためである。USBハブが必要かどうかは、環境によるので、試行錯誤する必要があるかも知れない。また、USBハブの有無で、ダウンロードの始まりのタイミングが影響される場合もある。また、プログラムをダウンロードするまえに、電源を接続すると、プリロードされたプログラム（ボード上のLEDが点滅）が開始される

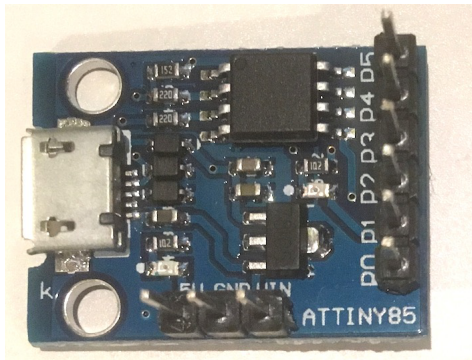
■ Reference

1. Download the Arduino IDE -> <https://www.arduino.cc/en/Main/Software>
2. digistumpのボード情報設定URL -> http://digistump.com/package_digistump_index.json

■ Digistump Digisparkの準備

Attiny25/45/85はAtmel社（2016年にAtmelはMicrochipに買収された）のマイクロチップである。このチップを使って、Digistump LLCは、Digisparkを開発した。Digisparkはハードウェアがクリエイティブ・コモン・ライセンス(CC-BY-SA 3.0)によりオープンになっている。

Digistump LLCが販売するDigispark Proは、Attiny87チップを用いている。また、Attiny85チップを用いたDigispark Kickstarter Attiny85の互換品も入手することができる。



Digispark Kickstarter Attiny85互換品

■ 参考URL

1. Atmel 8-bit AVR Microcontroller with 2/4/8K Bytes In-System Programmable Flash -> http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet-Summary.pdf
2. Digistump -> <http://digistump.com>
3. Digispark Pro Schematics -> https://s3.amazonaws.com/digistump-resources/files/d312a4a8_ProFinal.pdf
4. Digispark Kickstarter Attiny85の互換品 -> https://www.amazon.co.jp/gp/product/B01N0C5PTJ/ref=oh_aui_detailpage_o02_s00?ie=UTF8&psc=1
5. AVR® 8-Bit Microcontroller -> <https://www.pollin.de/productdownloads/D101130D.PDF>

■ IDEを用いたATtiny85へのプログラム書き込み

以下の手順で、Arduino IDEから、プログラムをATtiny85にダウンロードする。

1. Digispark/ATtiny85へのUSB microケーブルを用いた電源への接続

Digispark/ATtiny85は予めブートローダーとLED blinkが書き込まれているので、USB microケーブルで電源供給しただけで、ATtiny85がブートアップし、基盤上のLEDがブリンクする。

2. Arduino IDE上で、必要なプログラムを開発する

3. プログラムが完成したら、ATtiny85には、リセットスイッチがないので、一旦USBから切り離す

4. IATtinyのプログラム書き込み

IDEからコンパイルとダウンロードを指示した後、IDEがATtinyの接続待ちになるので、改めてUSBに接続すると、ダウンロードが始まる。

*書き込みエラーの起きる場合には、上記手順を繰り返す必要があるかもしれない。

■ ATtiny85によるLED点滅プログラムの例

■ LED blink with delay[] source

Digisparkの基盤に搭載されているLEDを点滅させる。delay関数を使う。pinMode(pin,OUTPUT)関数はATtinyでは、pin=0,1,2,3,4で動作するが、pin=5は、“weak drive strength as compared to regular I/O pins”と定義されていて、LEDを点灯させるのに十分な電流は流せない。

```
//
const int ledPin = 1;
void setup() {
  // initialize the digital pin as an output.
  pinMode(ledPin , OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(ledPin , HIGH);
  delay(1000);          // wait for a second
  digitalWrite(ledPin , LOW);
  delay(1000);          // wait for a second
```

```
}
```

■ LED blink with millis[] source

LEDを点滅させるのに、ピンへの割り込みを妨げない、millis関数を使う。

```
//
const int ledPin = 1;
int ledState = 0;
unsigned long previousMillis = 0;
int long interval = 2000;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    ledState=ledState^1;
    digitalWrite(ledPin,(ledState) ? LOW : HIGH);
  }
}
```

■ RGB color LED pulse control

RGBの三色LEDの明るさを個別に制御する

```
#include "TinyWire.h"
#include <util/delay.h> // accurate delay routine

//
const int ledRed = 1;
const int ledGreen = 3;
const int ledBlue = 4;

void setup() {
  // initialize the digital pin as an output.
  pinMode(ledRed , OUTPUT);
  pinMode(ledGreen , OUTPUT);
  pinMode(ledBlue , OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(ledRed , HIGH);
  _delay_us(100);
  digitalWrite(ledRed , LOW);
  _delay_us(1000);

  digitalWrite(ledGreen , HIGH);
  _delay_us(80);
  digitalWrite(ledGreen , LOW);
  _delay_us(1000);

  digitalWrite(ledBlue , HIGH);
  _delay_us(80);
```

```
digitalWrite(ledBlue , LOW);
_delay_us(1000);

}
```

■ RGB color LED control with Three PWM

RGB LEDの個別の明るさ制御をPWMで行なう。

```
void setup()
{
  pinMode (0, OUTPUT); // pin 5 // OC0A
  pinMode (1, OUTPUT); // pin 6 // OC0B
  pinMode (4, OUTPUT); // pin 3 // OC1B

  // Timer 0, A side
  TCCR0A = _BV (WGM00) | _BV (WGM01) | _BV (COM0A1); // fast PWM, clear OC0A on compare
  TCCR0B = _BV (CS00); // fast PWM, top at 0xFF, no prescaler
  OCR0A = 127; // duty cycle (50%)

  // Timer 0, B side
  TCCR0A |= _BV (COM0B1); // clear OC0B on compare
  OCR0B = 63; // duty cycle (25%)

  // Timer 1
  TCCR1 = _BV (CS10); // no prescaler
  GTCCR = _BV (COM1B1) | _BV (PWM1B); // clear OC1B on compare
  OCR1B = 31; // duty cycle (25%)
  OCR1C = 127; // frequency
} // end of setup

void loop() { }
```

<http://forum.arduino.cc/index.php?topic=134754.0>

■ LED on by the tact switch

タクトスイッチのチャタリングを防ぐためにタイマーを使う方法

```
//
#define led_pin 1
#define button_pin 3

byte own_address = 9;
byte slave_address = 10;

unsigned long previousMillis = 0;
int long interval = 100;

void setup() {
  // config led_pin as Output for driving an LED
  pinMode(led_pin, OUTPUT);
  // config error_led_pin as Output for driving an LED
  pinMode( button_pin, INPUT);
  unsigned long currentMillis = millis();
}
```


上図のUSB-microのV+と5Vを接続するSchottky diodeの方向は実際とは、逆であると考えられる。Digispark/Digispark-Compatible ATtiny85 USB Mini Development Board 回路図 (Reference others 1.) において、Schottky diodeの方向は、USB 5Vからボードの5Vに電流が供給されるようになっており、ボードの観察からもショットキーバリアダイオード SD103AWが、上図と逆の方向に接続されていることがわかる。

ATtiny85そのものは3.3Vの電源電圧でも動作するので、USBを利用しないならば、RaspberryPiから3.3Vの電源電圧を供給してもよい。

■ Port B ピンの役割

ATtinyのPort B (PB0~5)ピンは一つのピンに、複数の役割が割り当てられている。特定の入出力ルーチンには、予め決まったピンが設定されている。

ピンに配備された役割の概略

PB0 (MOSI/DI/SDA/AIN0/OC0A/OC1A/AREF/PCINT0)	-> Pin0 ->SDA
PB1 (MISO/DO/AIN1/OC0B/OC1A/PCINT1)	-> Pin1-> on-board LDE
PB2 (SCK/USCK/SCL/ADC1/T0/INT0/PCINT2)	-> Pin2 -> SCL
PB3 (PCINT3/XTAL1/CLKI/OC1B/ADC3)	-> Pin3 -> USB
PB4 (PCINT4/XTAL2/CLKO/OC1B/ADC2)	-> Pin4 -> USB
PB5 (PCINT5/RESET/ADC0/dW)	-> Pin5

各ピンの役割の詳細

PB5: Pin5

PCINT5: Pin Change Interrupt, Source 5

RESET: Reset Pin

ADC0: ADC Input Channel 0

dW: debugWIRE I/O

PB4: Pin4

PCINT4: Pin Change Interrupt 0, Source 4

XTAL2:

CLKO: External Clock

OC1B:

ADC2: ADC Input Channel 2

PB3: Pin3

PCINT3: Pin Change Interrupt 0, Source 3

XTAL1:

CLKI: External Clock Input

OC1B:

ADC3: ADC Input Channel 3

PB2: Pin2

SCK: Serial Clock Input

USCK:

SCL:

ADC1: ADC Input Channel 1

T0: Timer/Counter0 Clock Source.

INT0: External Interrupt 0 Input

PCINT2: Pin Change Interrupt 0, Source 2

PB1: Pin1

MISO: SPI Master Data Input

DO: SPI Slave Data Output

AIN1: Analog Comparator, Negative Input

OC0B: Timer/Counter0 Compare Match B Output
 INT0: External Interrupt 0 Input
 OC1A:
 PCINT1: Pin Change Interrupt 0, Source 1

PB0: Pin0
 MOSI:: SPI Master Data Output
 DI: SPI Slave Data Input
 SDA:
 AIN0: Analog Comparator, Positive Input
 OC0A: Timer/Counter0 Compare Match A output
 OC1A:
 AREF:
 PCINT0: Pin Change Interrupt 0, Source 0

■ DDRB、PORTB、PINBの操作

Pin0~4まではピンに接続されたLEDが点滅するが、PB5(pin 5)はoutputとしては働かない。(PCINT5/RESET/ADC0/dW) であるため

```
// ATtiny85
// PBx drive
// A simple blinky program for ATtiny85
// Connect red LED at pin0 (PB0)
//
#include <avr/io.h>
#include <util/delay.h> // accurate delay routine

int main (void)
{
    // set PB0 to output
    DDRB = 0b00000001; //
    while (1) {
        PORTB = 0b00000001; // pin0 to HIGH
        _delay_ms(500);
        PORTB = 0b00000000; // pin0 to LOW
        _delay_ms(500);
    }

    return 1;
}
```

ATtiny25/45/85 [DATASHEET], 52p 10.2 Ports as General Digital I/O

■ _delay_ms() and _delay_us()

<util/delay.h>を必要とし、引数はconstである必要がある。

■ delay() and tws_delay() with I2C

* NOTE: You must not use delay() or I2C communications will fail, use tws_delay() instead (or preferably some smarter timing system)

3. I2Cを通じたRaspberryPiとATtiny85の結合

Arduinoには、何種類かのI2Cライブラリが開発されており、錯綜している。どのライブラリを利用するかは、開発が継続されているか、情報へのアクセスが容易であるかどうかを考慮して、決定すべきである。ここでは、TinyWireS/M、TinyWireの二種類のライブラリを適宜用いている。

■ ATtiny85のI2Cの設定

RaspberryPiから、各種のデバイスを制御中のATtinyからデータを受け取る、あるいはATtinyをRaspberryPiから制御するために、ATtinyをI2Cのマスターあるいは、スレーブにする必要がある。この時、I2CのためにPB0(SDA: Pin0)とPB2(SCL:Pin2)が占有される。

なお、Digispark/ATtiny85の書き込みはUSBで接続して行なうので、Digispark/ATtiny85に対するプログラムのダウンロードには、5Vの電源電圧の供給が必要である。一方、ATtiny85そのものは3.3Vの電源電圧でも動作するので、5Vではなく、3.3Vの電源電圧を供給すれば、電源電圧が5Vの時に必要とされる、RaspberryPiとの間のレベルシフターは、介在させる必要がない。

■ I2Cのための TinyWireSルーチン

■ includeファイルの設定

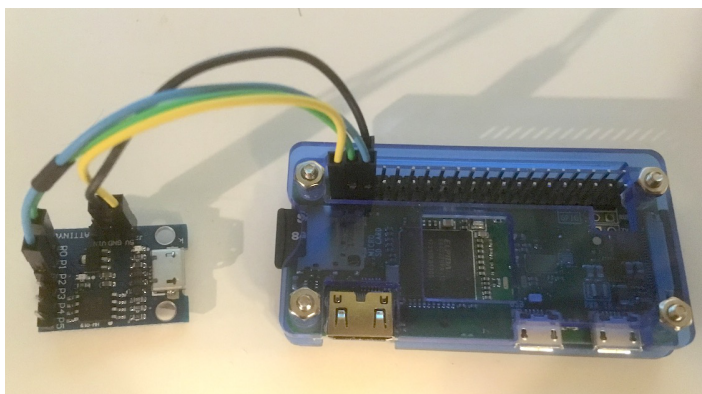
TinyWireS.hは<https://github.com/rambo/TinyWire>からzip形式でダウンロードして、スケッチ->ライブラリをインクルード、から設定した後、コンパイル、ダウンロードする。

■ 例: LEDI2CからLEDの点滅を制御する

RaspberryPiから、I2Cを通じて、ATtinyの動作を制御する。I2Cのアドレス0x05のレジスタ0x00に、点滅サイクルデータを、任意のタイミングで送って、ATtiny基盤上のLED点滅サイクルを制御する。

以下は、RaspberryPi ZeroのターミナルからコマンドをATtiny85に送り、LED点滅を制御する例である。

```
$ i2cget -y 1 0x5a 0x00 0xde
$ i2cset -y 1 0x5a 0x00 0xff
$ i2cset -y 1 0x5a 0x00 0x50
```



RaspberryPi Zeroから3.3V-GNDとI2CラインをDigiSpark/ATtiny85に繋げる

■ 結線

Digispark/ATtiny85: 5V -> RaspberryPi Zero 3.3V (黄)
 Digispark/ATtiny85: GND -> RaspberryPi Zero GND (黒)
 Digispark/ATtiny85: PB0 -> RaspberryPi Zero SDA (緑)
 Digispark/ATtiny85: PB2 -> RaspberryPi Zero SCL (青)

■ LED blinker with I2C source

```
#define I2C_SLAVE_ADDRESS 0x5A // the 7-bit address
#define LED_PIN 1 // DigiSpark LED pin1
```

```

#include "TinyWireS.h" // I2C routines

#ifndef TWI_RX_BUFFER_SIZE
#define TWI_RX_BUFFER_SIZE ( 16 )
#endif
//
int ledState = 0;
unsigned long previousMillis = 0;

volatile uint8_t i2c_regs[] =
{ 0xDE, // initial value of registers
  0xAD,
  0xBE,
  0xEF,
};

volatile byte reg_position;
const byte reg_size = sizeof(i2c_regs);

void requestEvent()
{ reg_position %= reg_size;
  TinyWireS.send(i2c_regs[reg_position]);
  reg_position++;
}

void receiveEvent(uint8_t howMany)
{
  // Sanity check
  if ((howMany < 1) || (howMany > TWI_RX_BUFFER_SIZE)) return;

  reg_position = TinyWireS.receive();
  howMany--;

  while(howMany--)
  {
    reg_position %= reg_size;
    i2c_regs[reg_position] = TinyWireS.receive();
    reg_position++;
  }
}

void setup()
{ pinMode(LED_PIN,OUTPUT);
  mt08Blink(LED_PIN, 2); //LED blink 2-times following bootup

  TinyWireS.begin(I2C_SLAVE_ADDRESS);
  TinyWireS.onReceive(receiveEvent);
  TinyWireS.onRequest(requestEvent);
}

void loop()
{ TinyWireS_stop_check();
  int long interval= i2c_regs[0]; // get data of register0 changed through I2C
  unsigned long currentMillis = millis(); // get time in milli second

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    ledState=ledState^1; // flip-flop led status using XOR
    digitalWrite(LED_PIN,(ledState ? LOW : HIGH); // C style IF
  }
}

void mt08Blink(byte led, byte times)
{
  times *= 2;
  while(times > 0) {
    digitalWrite(led,(times & 0x01) ? LOW : HIGH);
    delay (200);
    times--;
  }
}

```

4. Arduinoのオブジェクト指向プログラミング

Arduinoについても、オブジェクト指向プログラミングを適用することができる。このとき、以下の3種類のファイルが必要である。3種類のファイルはArduino IDEで自動的に管理される。

x.hファイル：クラスの関数と変数を定義する

x.cppファイル：クラスのプログラムを定義する

x.inoファイル：インスタンスをコンストラクトしてから、インスタンスを操作する

■ LED control with I2C

ATtiny85に3色LEDとI2Cの両方を接続するためには、ATtiny85のPB0~5のピンを全て使わなければならない。しかし、PWM方式では使用するピンが制限されるために、前提条件を満たすことができない。そこで、直接ピンをon/offすることで各色の明るさを制御する。

以下はLED Redの光量を増加させる手順である。

Led Redの初期値を読み込む -> 初期値は0a (=10) に設定されている
\$ i2cget -y 1 0x5a 0x00
0x0a

Led Redの明るさを設定する -> 明るさを1a (=26) に設定する
\$ i2cset -y 1 0x5a 0x00 0x1a

*各レジスタと3色の関係は以下のとおり

register[0] -> LED Red、register[2] -> LED Blue、register[4] -> LED Green

*TinyWire.hは、i2cget のwオプションをサポートしていない。

■ 結線

ATtiny85: 5V -> RpZ 3.3V
ATtiny85: GND -> RpZ GND
ATtiny85: PB0 -> RpZ SDA
ATtiny85: PB2 -> RpZ SCL
ATtiny85: GND -> RpZ GND
ATtiny85: PB1 -> LED Red
ATtiny85: PB3 -> LED Green
ATtiny85: PB4 -> LED Blue

■ LEDtest.h

```
// .h file
// definition of function names for LEDtest
//
#ifndef LedTest_h
#define LedTest_h

class LedTest
{ public:
  LedTest( int pin ); //constructor
  void set( int valInt );

private:
  int ledPin;
  int onInterval;
  int offInterval;
};
#endif
```

■ LEDtest.cpp

```
// .cpp file
// definition of Class for LEDtest
//
#include "Arduino.h"
#include "LedTest.h"

LedTest::LedTest( int pin )
{ ledPin = pin;
  onInterval = 10;
  pinMode(ledPin, OUTPUT);
}

void LedTest::set(int valInt)
{ onInterval= valInt;
  offInterval = 255- onInterval; // LED on interval must be 0 to 255
  digitalWrite(ledPin, HIGH);
  delayMicroseconds( onInterval );
  digitalWrite(ledPin, LOW);
  delayMicroseconds( offInterval );
}
```

■ LEDtest.ino Program 1; ATtiny85 stand-alone

```
// .ino file
#include "LedTest.h"

LedTest ledRed = LedTest(1);
LedTest ledBlue = LedTest(4);
LedTest ledGreen = LedTest(3);

void setup() { }

void loop()
{ ledRed.set(10);
  ledBlue.set(10);
  ledGreen.set(10);
}
```

■ LEDtest.ino Program 2; with I2C control

```
// .ino file
//
#include "LedTest.h"
#include "TinyWire.h" // I2C routines
#define I2C_SLAVE_ADDRESS 0x5A // the 7-bit address

#ifndef TWI_RX_BUFFER_SIZE
#define TWI_RX_BUFFER_SIZE ( 16 )
#endif
//

volatile uint8_t i2c_regs[] =
{ 10, 0, 10, 0, 10, 0 };

volatile byte reg_position;
const byte reg_size = sizeof(i2c_regs);
```

```

LedTest ledRed = LedTest(1); // construct instances
LedTest ledBlue = LedTest(4);
LedTest ledGreen = LedTest(3);

void requestEvent()
{ reg_position %= reg_size;
  TinyWire.send(i2c_regs[reg_position]);
  reg_position++;
}

void receiveEvent(int howMany)
{ // Sanity check
  if ((howMany < 1) || (howMany > TWI_RX_BUFFER_SIZE)) return;

  reg_position = TinyWire.receive();
  howMany--;

  while(howMany--)
  { reg_position %= reg_size;
    i2c_regs[reg_position] = TinyWire.receive();
    reg_position++;
  }
}

void setup()
{ TinyWire.begin(I2C_SLAVE_ADDRESS);
  TinyWire.onReceive(receiveEvent);
  TinyWire.onRequest(requestEvent);
}

void loop()
{ int intRed= i2c_regs[0]; // get data of registers changed through I2C
  ledRed.set( intRed );
  int intBlue= i2c_regs[2];
  ledBlue.set( intBlue );
  int intGreen= i2c_regs[4];
  ledGreen.set( intGreen );
}

```

5. ATtiny85のマスターとスレーブ

TinyWireライブラリーには、ATtiny85のI2Cが、スレーブになる／マスターになる／スレーブだが必要に応じてマスターになる、の三つの機能が含まれている。

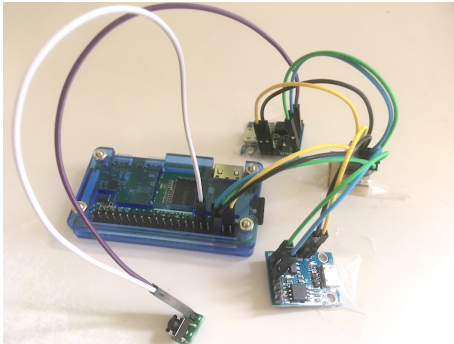
■ ATtinyのクラスターとMaster/Slaveの設定

ATtinyにI2Cのマスター、スレーブの両方の機能を持たせ、タクトスイッチを接続したものを、ATtiny-A、スレーブ機能だけを与えたものをATtiny-Bとする。両者をI2Cで接続し、電源をUSBからではなく、RaspberryPi Zeroから供給する状態にする。この状態でATtiny-Aのタクトスイッチを押すと、ATtinyは一時的にマスターとなって、ATtiny-BのLEDの点灯と消灯をトグルできる。

ATtiny-AのLEDもI2C経由でトグルできるので、I2Cのアドレスがユニークであれば、二つ以上のATtiny-Aを連結させることができる。

■ 結線

(黄)RaspberryPi Zero 3.3V -> ATtiny-A: 3.3V -> ATtiny-B: 3.3V
 (黒)RaspberryPi Zero GND -> ATtiny-A: GND -> ATtiny-B: GND
 (緑)RaspberryPi Zero SDA -> ATtiny-A: PB0 -> ATtiny-B: PB0
 (紫)ATtiny-A: PB3 -> tact switch -> (白)GND
 (青)RaspberryPi Zero SCL -> ATtiny-A: PB2 -> ATtiny-B: PB2



手前から反時計回りに、タクトスイッチ、ATtiny-B、ジャンパーボード、ATtiny-A、RaspberryPi Zeroである

*ATtiny85のPB3にタクトスイッチを取付けたとき、電源をUSBから取っている状態では、スイッチは働かないことに注意

■ ATtiny-A: Simultaneous Master/Slave program

source ->

https://github.com/lucullusTheOnly/TinyWire/blob/master/examples/simultaneous_master_slave/simultaneous_master_slave.ino

\$ i2cset -y 1 0x0a 0x61 は、RaspberryPiから、I2Cを通じて'a'を送信することであり、ATtiny-Aのon-boardのLED on/offをトグルできる。

IntegerString[ToCharacterCode["a"],16] -> 61 である。

```
//
#include <TinyWire.h>

#define led_pin 1
#define button_pin 3
#define error_led_pin 4

byte own_address = 9;
byte slave_address = 10;

void setup() {
  // config led_pin as Output for driving an LED
  pinMode(led_pin, OUTPUT);
  // config error_led_pin as Output for driving an LED
  pinMode(error_led_pin, OUTPUT);
  // config button_pin als INPUT for a connected button (normally open; connects to GND)
  pinMode( button_pin, INPUT_PULLUP);

  // config TinyWire library for I2C slave functionality
  TinyWire.begin( own_address );
  // set slave receive callback
  TinyWire.onReceive( onI2CReceive );
}
```



```

void loop() {
  // check, if the button was pressed
  if(digitalRead(button_pin) == 0) {
    // begin a master transmission with the specified slave address
    // the library temporarily takes the functionality of a master during sending
    TinyWire.beginTransmission( slave_address );
    // fill the send buffer
    TinyWire.send( 'b' );
    // execute the master sending and check for an error
    // returns 0 if there was no error (otherwise you can find the different error
    code definitions in TinyWire.h)
    if(TinyWire.endTransmission()!=0) {
      // turn on the error LED, if there was an error
      digitalWrite(error_led_pin, HIGH);
    }
    delay(300);
    // reset error LED after a short delay
    digitalWrite(error_led_pin, LOW);
  }
}

void onI2CReceive(int howMany){
  // loops, until all received bytes are read
  while(TinyWire.available()>0){
    // toggles the led everytime, when an 'a' is received
    if(TinyWire.read()==' a' ) digitalWrite(led_pin, !digitalRead(led_pin));
  }
}

```

■ ATtiny-B: Slave-receiver program

source ->

https://github.com/lucullusTheOnly/TinyWire/blob/master/examples/slave_receiver/slave_receiver.ino
 RaspberryPiから、\$ i2cset -y 1 0x0a 0x62 と出力することは、I2Cを通じてデバイス'0x0a'に、'b'を送信することであり、on-boardのLED of/off をトグルできる。
 IntegerString[ToCharCode["b"],16] -> 62 である。

```

//
# include < TinyWire.h >
# define led_pin 1

byte own_address = 10;
void setup () {
  pinMode (led_pin, OUTPUT);
  // config TinyWire library for I2C slave functionality
  TinyWire.begin ( own_address );
  // sets callback for the event of a slave receive
  TinyWire.onReceive ( onI2CReceive );
}

void loop () {
}

/*
I2C Slave Receive Callback :

```

```
Note that this function is called from an interrupt routine and shouldn't take long to
execute
*/
void onI2CReceive (int howMany) {
    // loops, until all received bytes are read
    while (TinyWire.available () > 0) {
        // toggles the led everytime, when an 'b' is received
        if (TinyWire.read () == 'b') digitalWrite (led_pin, ! digitalRead (led_pin));
    }
}
```

6. 開発ユーティリティ

■ Digikeyboardをモニターとして用いる方法

Digispark/ATtiny85が、USBキーボードをエミュレートする、Digisparkに準備されているスケッチ・サンプルである。PC側では、キーボードはUSを選び、英数入力モードにしておく必要がある。デスクトップでアクティブなウィンドウに、キーボードでタイプするように、DigiKeyboard.println関数に設定された文字列あるいは整数を、USBで接続されたPCに、一文字ずつ送り込む(Arduino IDEのウィンドウがアクティブであれば、直接的に書き込まれていく)。

Digikeyboardには、USB接続のために、PB3(Pin3)とPB4(Pin2)が使われるので、この二つのピンを同時に他の用途に使うことはできない。

Arduio IDEのシリアルモニタは、Digispark/ATtiny85では利用できないので、その代替りのモニターとして用いることができる。なお、PCとDigispark/ATtiny85の間にUSBハブがあると、プログラムが働かない場合があるので、試行錯誤が必要となるかもしれない。

```
#include "DigiKeyboard.h"

void setup() {
    // don't need to set anything up to use DigiKeyboard
}

void loop() {
    // this is generally not necessary but with some older systems it seems to
    // prevent missing the first character after a delay:
    DigiKeyboard.sendKeyStroke(0);

    // Type out this string letter by letter on the computer (assumes US-style
    // keyboard)
    DigiKeyboard.println("Hello Digispark!"); // or int variable

    // It's better to use DigiKeyboard.delay() over the regular Arduino delay()
    // if doing keyboard stuff because it keeps talking to the computer to make
    // sure the computer knows the keyboard is alive and connected
    DigiKeyboard.delay(5000);
}
```

■ 参考URL

datasheet

1. AVRWiki -> <http://avrwiki.osdn.jp/cgi-bin/wiki.cgi?page=AVRWiki>
2. Digistump Wiki -> <http://digistump.com/wiki/start>

3. ATtiny25/V / ATtiny45/V / ATtiny85/V Summary ->
http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet-Summary.pdf
4. ATtiny25/V / ATtiny45/V / ATtiny85/V -> <https://www.marcelpost.com/wiki/images/3/31/ATtiny85.pdf>
5. ATtiny85 ADC -> https://www.marcelpost.com/wiki/index.php/ATtiny85_ADC
6. Arduinoで遊ぶページ ->
https://garretlab.web.fc2.com/arduino/inside/hardware/arduino/avr/cores/arduino/wiring_digital.c/digitalWrite.html
7. AVR Libc 日本語 -> <http://cega.jp/avr-libc-jp/index.html>
8. GETTING STARTED WITH ATTINY85 -> <http://electronut.in/getting-started-with-attiny85-avr-programming/>
9. DigiStump schematic -> <https://s3.amazonaws.com/digispark/DigisparkSchematicFinal.pdf>

I2C

1. Composite Master and Slave I2C library for Atmels ATTiny microcontrollers ->
<https://github.com/lucullusTheOnly/TinyWire>
2. TinyWire.h -> <https://github.com/rambo/TinyWire>
3. Arduino I2C master library -> <https://github.com/rambo/I2C>

Node-Red, Johnny-Five

1. Raspberry PiからArduinoをJohnny-FiveとFirmataで操作する ->
<http://masato.github.io/2015/03/22/raspberrypi-arduino-johnny-five-firmata/>
2. gort -> <http://gort.io>

others

1. Learn To Use ATtiny85 USB Mini Development Board -> <https://www.electroschematics.com/12102/learn-to-use-attiny85-usb-mini-development-board/>
2. Digispark(ATtiny85) Arduino開発環境で、Lチカ -> <https://qiita.com/mt08/items/df4ed8c659b205d1fa1e>
3. Digispark(ATtiny85) をI2CのSlaveで使用(TinyWireS)して、RaspberryPi(Master)と通信してみる ->
<https://qiita.com/mt08/items/46d73c4e1957c1eb55f5>
4. Lチカコード作成・書き込み・実演 ->
http://digistump.com/wiki/digispark/tutorials/connecting#your_first_upload
6. 文字定数をフラッシュ・メモリに配置するように書き換えてコンパイルする -> <http://yoshi-s.cocolog-nifty.com/cpu/avr/index.html>
7. Arduino日本語リファレンス -> <http://www.musashinodenpa.com/arduino/ref/index.php>
8. Arduinoリファレンス -> https://garretlab.web.fc2.com/arduino_reference/index.html
9. Digisparkを試してみました -> <http://nuneno.cocolog-nifty.com/blog/2014/12/digispark-d14d.html>
10. 外部割り込み -> <http://d.hatena.ne.jp/hijouguchi/20100623/1277270332>
11. AVR avr/io.h インクルードファイルは一体何をしているのか -> <http://morokyuuu.way-nifty.com/blog/2012/12/avr-avrtoh-8475.html>
12. Four PWM Outputs from the ATtiny85 -> <http://www.technoblogy.com/show?LE0>

OOP

1. Arduinoのオブジェクト指向プログラミング ->
<http://monakaice88.hatenablog.com/entry/20130404/1365016051>

コラム パーツとしてのArduino

Arduino、ことに、ここで取り上げたATtiny85は、チップの機能として規模は小さいものの、圧倒的に小型で、低消費電力で、低価格である。これ程のレベルになると、コンピュータとして扱うよりは、システム全体のパーツとしてこれを捉える、つまり、如何にシステム全体のコストと開発時間を、最適にするかを問うための、あるいは問われるパーツとして、考えるべきだ。

つまり、要求される機能を満たすために、その個数で要求をカバーできるならば、出来るだけ、プログラミングを最小にして、結果を得ようとする態度を取ることである。

であれば、プログラムはできるだけ再利用すべきだし、チップはクラスタ化することを考えるべきだし、各チップに分担させる機能はできるだけ単純にするべきだという、方針が得られる。さらには、複雑な計算はRaspberryPiのMathematicaで実行するために、RaspberryPiとArduinoの連携は必須となる。

であれば、要求を満たすための、ハードウェアとは、できるだけ開発過程が単純であること、開発後のメンテナンスが簡単であること、さらには、電源モジュールとウォッチドッグタイマーと組み合わせて、自動で電源をオンオフできたり、冗長性を持たせることができるようにする、などが考えられるだろう。

センサー

Sensing sub-systems

I2Cを通じて外部環境データとしての、気温、気圧、加速度、磁力、角速度、距離、等を取得できる、様々な外部モジュールが市販されている。I2Cバスにモジュールを並列に接続することで、これらを単独であるいは複数同時に扱うことができる。

1. Berry IMU モジュール

加速度計と磁力計、ジャイロ、気温/気圧計の組合せ

■ MPU6050 motion tracking sensor

GY521 MPU6050 モジュールの説明を参照のこと

■ LSM9DS0 magnetic field sensor

```
process = StartProcess[$SystemShell];
(*set interrupt service to GPIO Pin7*)
chipAdr = "0x1e"; rSet[r_] := WriteLine[process, StringRiffle[{"i2cset -y 1", chipAdr,
"0x00", r}]];
(* CTRL_REG5 _XM, 24h set magnetsensor data rate*)
rSet[{"0x24", "0x00"}];
(* CTRL_REG7 _XM, 26h set magnetsensor to continuous conversion*)
rPut["0x26", "0x00"];
```

```
ac4 = usGet["0x09", "0x08"]
```

```
4517
```

```
d = Table[usGet["0x09", "0x08"], {10}];
ListLinePlot[d]
```

■ 結線図

VDD 3.3V

■ 参考URL

- | | |
|-----------------------|--|
| 1. LSM9DS0 data sheet | http://ozzmaker.com/wp-content/uploads/2014/12/LSM9DS0.pdf |
| 2. MPU6050 | https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/ |
| 3. ST gyroscope | http://www.st.com/ja/mems-and-sensors/gyroscopes.html?querycriteri- |
| a=productId=SC1288 | |

2. Dovewill (Adafruit) 10 DOF モジュール

加速度計と磁力計、ジャイロ、気温/気圧計の組合せ

■ 設定

AdaFruit 10 DOFモジュールとその互換品であるDovewill 10 DOFモジュールでは、ジャイロスコープに異なるICが使われている。DovewillのジャイロはA3G4250Dである。

■ 結線

10 DOFはLDO電圧レギュレータとして、MIC5225-QT33が用いられているので、VDDとして 5Vを接続する。

■ アドレス

```
$ sudo i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- 19 -- -- -- -- 1e --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- 69 -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- 77
```

```
LSM303DLHC Accelometer, chipSlaveAdrss: 0x19
LSM303DLHC Magnetmeter, chipSlaveAdrss: 0x1e
A3G4250D Gyroscopes, chipAdrss: 0x69
BMP180 Temperature/Barobetric pressure, chipAdrss: 0x77
```

■ LSM303DLHC 加速度計

```
chipAdr = "0x19";
rGet[reg_] :=
  StringTrim[RunProcess[{"i2cget", "-y", "1", chipAdr, reg}, "StandardOutput"]];
{rGet["0x20"], rGet["0x21"], rGet["0x22"]}
{0x07, 0x00, 0x00}
```

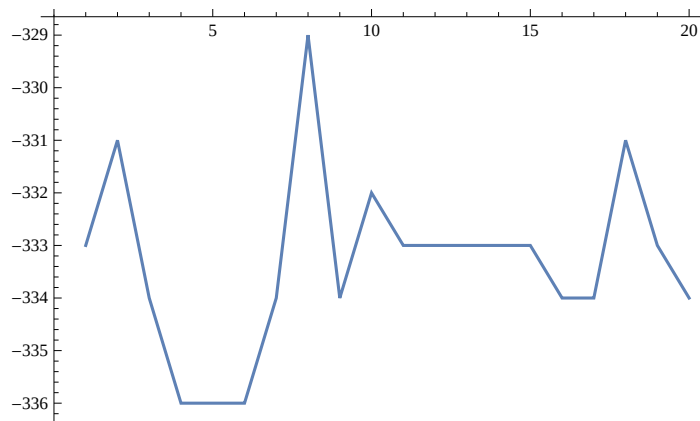
■ LSM303DLHC 磁場センサー

All axes data must read for the next data series, namely OUT_X_H_M (0x03) to OUT_Y_L_M (0x08) 6-registers should be read seirally.

```
chipAdr = "0x1e";
process = StartProcess[$SystemShell];
rGet[r_] := (WriteLine[process, StringRiffle[{"i2cget -y 1", chipAdr, r}]];
  ReadLine[process]);
rPut[r_, d_] := WriteLine[process, StringRiffle[{"i2cset -y 1", chipAdr, r, d}]];
(* Get Signed Short Integer *)
ssGet[reg1_, reg2_] := (
  r = Flatten[{rGet[reg1], rGet[reg2]}];
  r1 = Map[FromDigits[#, 16] &, StringReplace[r, "0x" -> ""]];
  r2 = BitShiftLeft[r1[[1]], 8] + r1[[2]];
  If[r2 > 32767, -32768 + (r2 - 32768), r2]
);

(* data rate 30Hz *)
rPut["0x00", "0x14"];
(* continuous conversion *)rPut["0x02", "0x00"];
{ssGet["0x03", "0x04"], ssGet["0x05", "0x06"], ssGet["0x07", "0x08"]}
{-222, -81, -647}
```

```
d = Table[{ssGet["0x03", "0x04"], ssGet["0x05", "0x06"], ssGet["0x07", "0x08"]}, {20}];
ListLinePlot[d[[All, 1]]]
```



■ Reference

1. LSM303DLHC data sheet

<http://www.st.com/content/ccc/resource/technical/document/datasheet/56/ec/ac/de/28/21/4d/48/DM00027543.pdf/files/DM00027543.pdf/jcr:content/translations/en.DM00027543.pdf>

2. L3GD20 data sheet

<http://www.st.com/content/ccc/resource/technical/document/datasheet/43/37/e3/06/b0/bf/48/bd/DM00036465.pdf/files/DM00036465.pdf/jcr:content/translations/en.DM00036465.pdf>

■ A3G4250D ジャイロスコープ

```
chipAdr = "0x69";
rGet[reg_] := StringTrim[RunProcess[{"i2cget", "-y", "1", chipAdr, reg},
"StandardOutput"]];
rGet["0x0f"](* datasheet shows Who_am_I register 0x0f value is 0xd3 *)
```

```
"0xd3"
```

■ Reference

1. A3G4250D data sheet

<http://www.st.com/content/ccc/resource/technical/document/datasheet/5c/f1/a4/70/1b/fa/40/d2/DM00047823.pdf/files/DM00047823.pdf/jcr:content/translations/en.DM00047823.pdf>

■ BMP180 温度・気圧センサー

```
chipAdr = "0x77";
ac1 = ssGet["0xaa", "0xab"];
ac2 = ssGet["0xac", "0xad"];
ac3 = ssGet["0xae", "0xaf"];
ac4 = usGet["0xb0", "0xb1"];
ac5 = usGet["0xb2", "0xb3"];
ac6 = usGet["0xb4", "0xb5"];
b1 = ssGet["0xb6", "0xb7"];
b2 = ssGet["0xb8", "0xb9"];
mb = ssGet["0xba", "0xbb"];
mc = ssGet["0xbc", "0xbd"];
md = ssGet["0xbe", "0xbf"];
```

```
rPut["0xf4", "0x2e"]; Pause[0.05];
ut = usGet["0xf6", "0xf7"];
```

```
x1 = (ut - ac6)*ac5/2.^15;
x2 = mc*2.^11/(x1 + md);
b5 = x1 + x2;
t = (b5 + 8.)/2.^4/10.
```

26.1774

```
rPut["0xf4", "0x34"]; Pause[0.05];
up = usGet["0xf6", "0xf7"];
b6 = b5 - 4000.;
x1 = (b2*(b6*b6/2^12))/2^11;
x2 = ac2*b6/2^11;
x3 = x1 + x2;
b3 = ((ac1*4 + x3) + 2)/4.;
x1 = ac3*b6/2^13;
x2 = (b1*(b6*b6/2^12))/2^16;
x3 = ((x1 + x2) + 2)/2^2;
b4 = ac4*(x3 + 32768)/2^15;
b7 = (up - b3)*50000;
If[b7 < FromDigits["80000000", 16], p = b7*2/b4, p = b7/b4*2];
x1 = p/2^8*p/2^8;
x1 = x1*3038/2^16;
x2 = (-7357*p)/2^16;
p = p + (x1 + x2 + 3791)/2^4;
p = p*0.01
```

1009.89

■ Reference

1. BMP180 data sheet <https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf>
2. BMP280 data sheet https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001-19.pdf

3. GY-511 LSM303DLHC 方位モジュール

加速度計と磁力計の組合せ

- 設定
- アドレス

```
$ sudo i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  19  --  --  --  --  1e  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```


LSM303DLHC Accelometer, chipSlaveAdrss: 0x19
 LSM303DLHC Magnetmeter, chipSlaveAdrss: 0x1e

■ LSM303DLHC Accelometer

Where as the Magnetic field sensor registers require to be read all sequentially, Accelometer each OUTPUT register, can be read independantly.

Also, Accelometer OUTPUT registers have faster response than Magneto-meters'.

```
chipAdr = "0x19";
process = StartProcess[SystemShell];
rGet[r_] := (WriteLine[process, StringRiffle[{"i2cget -y 1", chipAdr, r}]];
ReadLine[process]);
rPut[r_, d_] := WriteLine[process, StringRiffle[{"i2cset -y 1", chipAdr, r, d}]];
(* Get Signed Short Integer *)
ssGet[{reg1_, reg2_}] := (
  r = Flatten[{rGet[reg1], rGet[reg2]}];
  r1 = Map[FromDigits[#, 16] &, StringReplace[r, "0x" -> ""];
  r2 = BitShiftLeft[r1[[1]], 8] + r1[[2]];
  If[r2 > 32767, -32768 + (r2 - 32768), r2]
);
areg = Association[cntlReg1 -> "0x20", mrReg -> "0x02",
  outXA -> {"0x29", "0x28"}, outYA -> {"0x2b", "0x2a"}, outZA -> {"0x2d", "0x2c"}];
```

■ CNTL_REG1_A

CNTL_REG1_A

7	6	5	4	3	2	1	0
ODR3	ODR2	ODR1	ODR0	LPen	Zen	Yen	Xen

ODR[3:0] 0100→50Hz

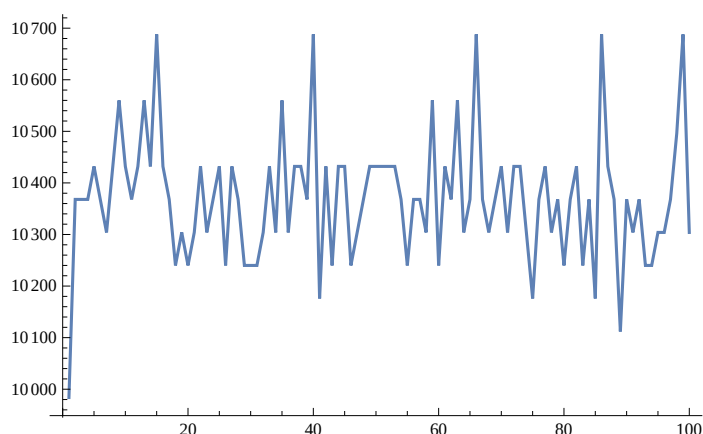
```
Column[{"CNTL_REG1_A",
  Grid[{{7, 6, 5, 4, 3, 2, 1, 0}, {"ODR3", "ODR2", "ODR1", "ODR0",
    "LPen", "Zen", "Yen", "Xen"}}, Frame -> All, ItemSize -> All],
  "ODR[3:0] 0100→50Hz"]]
```

■ sample code

```
(* set output data rate to 50Hz *)
rPut[areg@cntlReg1, "0x47"];
{ssGet[areg@outXA], ssGet[areg@outYA], ssGet[areg@outZA]}
{-5632, 8704, 10240}

Timing[d = Table[{ssGet[areg@outXA], ssGet[areg@outYA], ssGet[areg@outZA]}, {100}];]
{9.53, Null}

ListLinePlot[d[[All, 3]]]
```



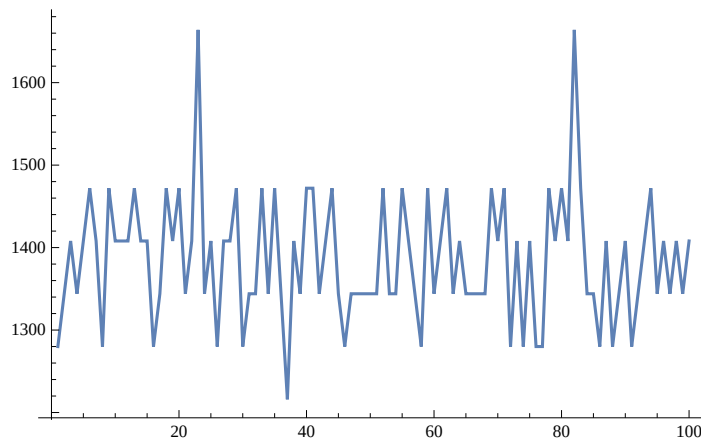
■ Asynchronous read/write trial

```

rPush[r_] := WriteLine[process, StringRiffle[{"i2cget -y 1", chipAdr, r}]];
ssCon[d_] := (
  r = Flatten[d];
  r1 = Map[FromDigits[#, 16] &, StringReplace[r, "0x" -> ""]];
  r2 = BitShiftLeft[r1[[1]], 8] + r1[[2]];
  If[r2 > 32767, -32768 + (r2 - 32768), r2]
);
n = 200;
Timing[
  (* clean-up buffer *) ReadString[process, EndOfBuffer];
  d = Flatten[Table[Map[rPush, Flatten[{areg@outXA, areg@outYA, areg@outZA}]]];
  Pause[0.08];
  StringSplit[ReadString[process, EndOfBuffer]], {i, n}];
  If[Length[d] != 6 n, Print["Error"]]
]{3.39, Null}

d = Partition[Partition[d, 2], 6];
ListLinePlot[Map[ssCon, d][All, 1]]

```



■ LSM303DLHC Magnetic field sensor

To refresh OUTPUT registers, All OUTPUT registers must be read, however TEMP data can be omitted when reading registers, noise part of OUTPUT becomes notably large. You should read data sequentially, TEMP, XOUT, YOUT, ZOUT with this order.

```

chipAdr = "0x1e";
process = StartProcess[SystemShell];
rGet[r_] := (WriteLine[process, StringRiffle[{"i2cget -y 1", chipAdr, r}]];
  ReadLine[process]);
rPut[r_, d_] := WriteLine[process, StringRiffle[{"i2cset -y 1", chipAdr, r, d}]];
(* Get Signed Short Integer *)
ssGet[{reg1_, reg2_}] := (
  r = Flatten[{rGet[reg1], rGet[reg2]}];
  r1 = Map[FromDigits[#, 16] &, StringReplace[r, "0x" -> ""]];
  r2 = BitShiftLeft[r1[[1]], 8] + r1[[2]];
  If[r2 > 32767, -32768 + (r2 - 32768), r2]
);
mreg = Association[
  craReg -> "0x00", mrReg -> "0x02",
  outXM -> {"0x03", "0x04"}, outYM -> {"0x07", "0x08"}, outZM -> {"0x05", "0x06"}, tmp
-> {"0x31", "0x32"}
];

```

■ CRA_REG_M

7	6	5	4	3	2	1	0
TEMP_EN	0	0	D02	D01	D00	0	0

Clk_Sel (0:internal oscillator,1:PLL with X axis gyro,2:PLL with Y gyro,3:PLL with Z gyro)

```
Grid[{{7, 6, 5, 4, 3, 2, 1, 0}, {"TEMP_EN", 0, 0, "D02", "D01", "D00", 0, 0}},
Frame -> All, ItemSize -> All]
```

7	6	5	4	3	2	1	0
TEMP_EN	0	0	D02	D01	D00	0	0

■ sample code

```
(* TEMP enable and data rate 30Hz *)
rPut[mreg@craReg, "0x94"];
(* continuous conversion *)rPut[mreg@mrReg, "0x00"];
```

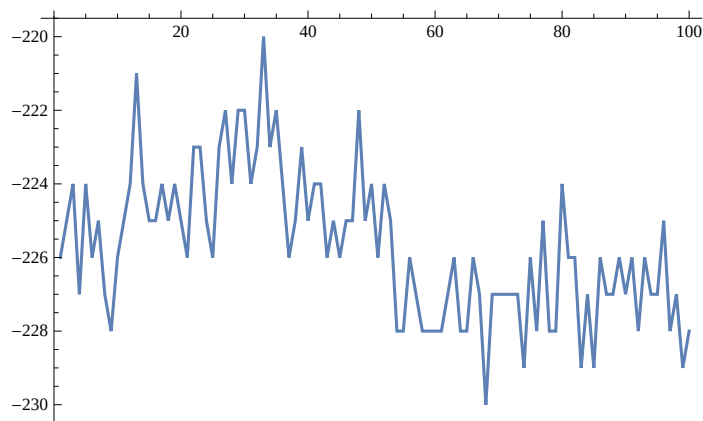
```
{ssGet[mreg@tmp]/16., ssGet[mreg@outXM], ssGet[mreg@outYM], ssGet[mreg@outZM]}
```

```
{40., -50, 283, -249}
```

```
Timing[d = Table[{ssGet[mreg@tmp]/16., ssGet[mreg@outXM], ssGet[mreg@outYM], ssGet[m-
reg@outZM]}, {100}];]
```

```
{12.16, Null}
```

```
ListLinePlot[d[[All, 4]]]
```



■ Asynchronous read/write trial

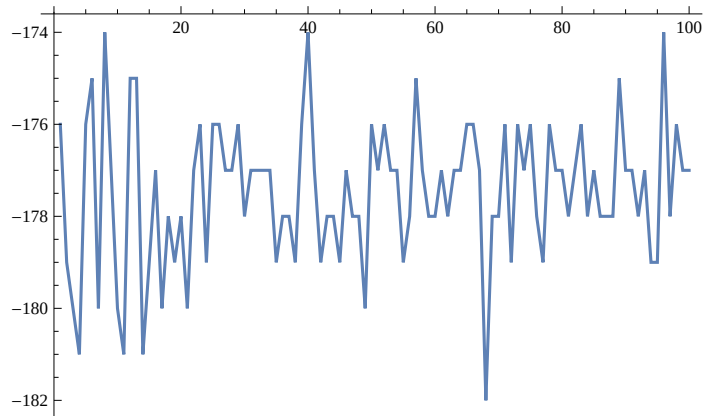
```
rPush[r_] := WriteLine[process, StringRiffle[{"i2cget -y 1", chipAdr, r}]];
ssCon[d_] := (
r = Flatten[d];
r1 = Map[FromDigits[#, 16] &, StringReplace[r, "0x" -> ""]];
r2 = BitShiftLeft[r1[[1]], 8] + r1[[2]];
If[r2 > 32767, -32768 + (r2 - 32768), r2]
);
```

```

ReadString[process,EndOfBuffer];
Timing[d=Table[
  {Map[rPush,mreg@tmp],Map[rPush,mreg@outXM],Map[rPush,mreg@outYM],
  Map[rPush,mreg@outZM]};
  Pause[0.17];Partition[StringSplit[ReadString[process,EndOfBuffer]],2],{i,100}];]
{2.41`,Null}

d=Partition[Flatten[d,1],4];
ListLinePlot[Map[ssCon,d[[All,4]]]]

```



■ Reference

1. LSM303DLHC data sheet <http://www.st.com/content/ccc/resource/technical/document/datasheet/56/ec/ac/de/28/21/4d/48/D-M00027543.pdf/files/DM00027543.pdf/jcr:content/translations/en.DM00027543.pdf>
2. LSM303DLH application note https://www.pololu.com/file/download/LSM303DLH-compass-app-note.pdf?file_id=0J434

4. GY 521 MU6050 モジュール

加速度計とジャイロの組合せ。

- 設定
- アドレス

```

$ sudo i2cdetect -y 1
 0 1 2 3 4 5 6 7 8 9 a b c d e f
00:  -----
10:  -----
20:  -----
30:  -----
40:  -----
50:  -----
60:  ----- 68 -----
70:  -----

```

■ 結線図

GY 521はLDO電圧レギュレータとして、XC6504A331MR-G 4A2Dが用いられているので、VDDとして5Vを接続する。

■ MPU6050の制御

```
chipAdr="0x68";
process=StartProcess[$SystemShell];

(* shell interface to i2c *)
rGet[r_] := (WriteLine[process,StringRiffle[{"i2cget -y 1",chipAdr,r}]]];
ReadLine[process]);
rGetW[r_] := (WriteLine[process,StringRiffle[{"i2cget -y 1",chipAdr,r,"w"}]]];
ReadLine[process]);
rPut[r_,d_] := WriteLine[process,StringRiffle[{"i2cset -y 1",chipAdr,r,d}]]];

(* Get Signed Short Integer from a word *)
ssGet[r0_] := (
  r=rGetW[r0];
  r1=StringTake[r,{5,6}]<>StringTake[r,{3,4}];
  r2=FromDigits[r1,16];
  If[##>32767,-32768+(##-32768),##]&[r2]);

(* target registers, suffix-0 is the first address of hi-lo bytes address *)
reg=Association[whoAmI->"0x75",LPF->"0x1a",gyroConf->"0x1b",accelConf->"0x1c",
  powerManage1->"0x6b",powerManage2->"0x6c",
  accelX0->"0x3b",accelY0->"0x3d",accelZ0->"0x3f",tmp0->"0x41",
  gyroX0->"0x43",gyroY0->"0x45",gyroZ0->"0x47"];

(* WHO_AM_I will return 0x68 *)
rGet[reg@whoAmI]=="0x68"
```

True

```
Grid[{{7, 6, 5, 4, 3, 2, 1, 0}, {"devise_reset", "SLEEP", "CYCLE", "-",
  "Temp_disable", "CLK_SEL", SpanFromLeft}}, Frame -> All, ItemSize -> All]
```

7	6	5	4	3	2	1	0
devise_reset	SLEEP	CYCLE	-	Temp_disable	CLK_SEL		

Clk_Sel (0: internal oscillator, 1: PLL with X axis gyro, 2: PLL with Y gyro, 3: PLL with Z gyro)

＊モジュールを初期位置から90度程度傾けた位置で、動作していたジャイロがストップすることがある。

```
(* select clock to PLL with X gyro ; power manage 1 register is 0x6b *)
rPut[reg@powerManage1,"0x01"];
rGet["0x6b"]
"0x01"

(* BandWidth to 5Hz (Fs=1kHz) *)
rPut[reg@LPF,"0x06"];

(* get temperature of die *)
ssGet[reg@tmp0]/340.+36.53
27.703529411764706`

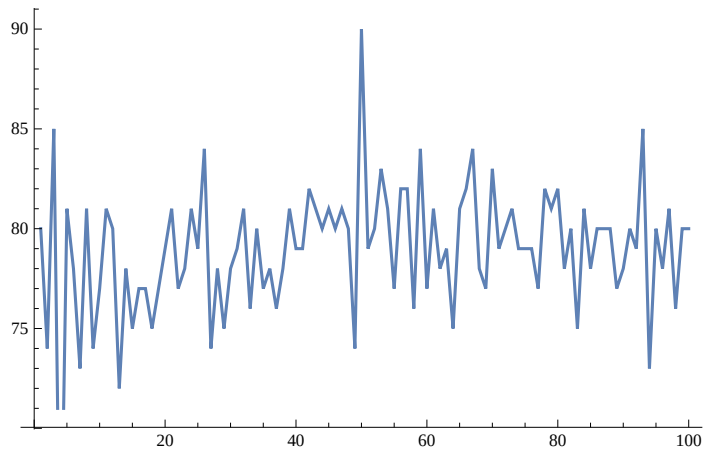
{ssGet[reg@accelX0],ssGet[reg@accelY0],ssGet[reg@accelZ0]}

{-714,-1358,21746}

{ssGet[reg@gyroX0],ssGet[reg@gyroY0],ssGet[reg@gyroZ0]}

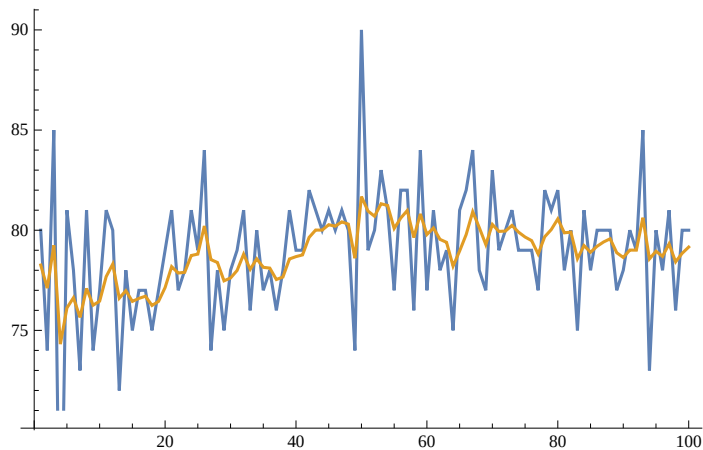
{82,-371,8}
```

```
d=Table[ssGet[reg@gyroX0],{100}];
ListLinePlot[d]
```



ジャイロからの出力には必ずノイズ分が含まれている。ノイズを減少させるために、カルマンフィルターを適用してみる。

```
(* construct kalman object named "gyro" from kalman class *)
kalman[gyro];
(* gyro object initialize and set values *)
initPrev[gyro[0,0.01]];
setVariance[gyro[0.1,1]]
ed=Table[estimated[gyro[d[[i]]]],{i,100}];
ListLinePlot[{d,ed}]
```



青色で示したプロットが元データ、橙色のプロットがカルマンフィルタを施した結果である。

■ KalmanEstimator

Kalman equation is composed of

Observation equation = Real value+ Noise

Conditional equation = Previous condition(value) + Noise

```

kalman[name_] := Module[{nVobsEq, nVcondEq, kGain, wPrevious, xPrevious, xForecast, xEstimated},

  initPrev[name[xp_, wp_]]^:= (
    xPrevious = xp (* previous observed value *);
    wPrevious = wp (* previous variance of estimated values *);
  );

  setVariance[name[ceq_, oeq_]]^:= (
    nVcondEq = ceq (* noise variance of condition equation *);
    nVobsEq = oeq (* noise variance of observation equation *);
  );

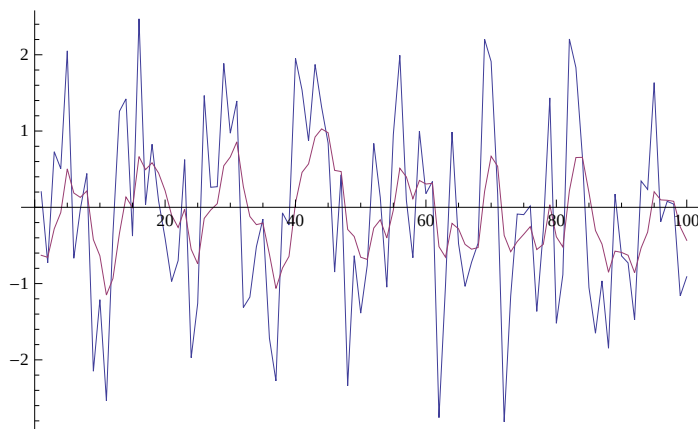
  estimated[name[xObs_]]^:= (
    xForecast = xPrevious (* local level model *);
    wPrevious = wPrevious + nVcondEq;
    kGain = wPrevious / (wPrevious + nVobsEq);
    wPrevious = (1 - kGain) * wPrevious;
    xPrevious = xForecast + kGain * (xObs - xForecast);
    Return[xPrevious]
  );

];

kalman[gyro];
initPrev[gyro[0, 0.01]];
setVariance[gyro[0.1, 1]]

(* test of Kalman filter *)
u = Table[Sin[ $\frac{2 \pi i}{13.0}$ ] + RandomReal[NormalDistribution[0, 1]], {i, 100}];
v = Table[estimated[gyro[u[[i]]]], {i, 100}];
ListLinePlot[{u, v}]

```



■ ジャイロセンサのオフセット・キャンセル出力

ジャイロセンサの各チャンネルの出力には、オフセットが含まれている。一般的には、センサを適切な姿勢で固定した時の出力の平均値を求めるなどして、オフセット量は推定できる。さらに、推定したオフセットを、特定のレジスタに書き込むことで、書き込んだオフセット値を、出力からキャンセルすることができる。オフセット値を書き込むレジスタは公開されていないが、以下のレジスタをこれらの目的のために使うことができる。ジャイロセンサのオフセットのキャンセルに使えるのは、XG_OFFS_USR、YG_OFFS_USR、ZG_OFFS_USR である。以下は各レジスタのアドレスとビットパターンである。

XG_OFFS_USR	0x00	// [7] PWR_MODE, [6 : 1] XG_OFFS_USR, [0] OTP_BNK_VLD
YG_OFFS_USR	0x01	// [7] PWR_MODE, [6 : 1] YG_OFFS_USR, [0] OTP_BNK_VLD
ZG_OFFS_USR	0x02	// [7] PWR_MODE, [6 : 1] ZG_OFFS_USR, [0] OTP_BNK_VLD
X_FINE_GAIN	0x03	// [7 : 0] X_FINE_GAIN

```

Y_FINE _GAIN    0 x04 // [7 : 0] Y_FINE _GAIN
Z_FINE _GAIN    0 x05 // [7 : 0] Z_FINE _GAIN
XA_OFFS _H      0 x06 // [15 : 0] XA_OFFS
XA_OFFS _L _TC  0 x07
YA_OFFS _H      0 x08 // [15 : 0] YA_OFFS
YA_OFFS _L _TC  0 x09
ZA_OFFS _H      0 x0A // [15 : 0] ZA_OFFS
ZA_OFFS _L _TC  0 x0B
XG_OFFS _USRH   0 x13 // [15 : 0] XG_OFFS _USR
XG_OFFS _USRL   0 x14
YG_OFFS _USRH   0 x15 // [15 : 0] YG_OFFS _USR
YG_OFFS _USRL   0 x16
ZG_OFFS _USRH   0 x17 // [15 : 0] ZG_OFFS _USR
ZG_OFFS _USRL   0 x18

```

GYRO_CONFIG register(1B)がデフォルト状態にある、ジャイロYからの出力である。

```
{rGet["0x45"], rGet["0x46"], ssGet["0x45"]}
```

```
{0xfe, 0xb5, -327}
```

ジャイロYのオフセット分を2bit分シフトした値を、YG_OFFS_USRLレジスタに書き込む。

```
"0x"<>ToString[FromDigits[IntegerDigits[BitShiftRight[327,2],16]]]
```

```
"0x51"
```

```
rPut["0x15","0x00"];rPut["0x16","0x51"];
```

ジャイロYの出力からオフセット分がキャンセルされる。

```
{rGet["0x45"], rGet["0x46"], ssGet["0x45"]}
```

```
{0xff, 0xfa, -2}
```

■ 参考URL

1. MPU-6050 data sheet https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf
2. MPU-6050 register map <https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf>
3. hardware offset registers note <http://www.digikey.jp/Web%20Export/Supplier%20Content/invensense-1428/pdf/invensense-mpu-hardware-offset.pdf>
4. カルマンフィルタの導出 <https://qiita.com/MoriKen/items/0c80ef75749977767b43>
4. カルマンフィルタの考え方 <https://logics-of-blue.com/kalman-filter-concept/>
6. カルマンフィルタと最尤法 <https://logics-of-blue.com/kalman-filter-mle/>
7. カルマンフィルタのプログラム <https://logics-of-blue.com/wp-content/uploads/2017/04/kf-concept-R.txt>
8. MPU6050-Pi-Demo <https://github.com/richardghirst/PiBits/blob/master/MPU6050-Pi-Demo/MPU6050.cpp>
9. XC6504A331MR-G 4A2D <https://www.torexsemi.com/file/x6504/XC6504.pdf>

5. カメラモジュール

6. 距離／近接モジュール

■ US-015 超音波距離センサー

トリガー(TRIG)を10 μ 秒以上HIGHにすると、距離の測定を始める

距離=(ECHOがHIGHの時間 X 340)/2

5V 待機時消費電流：<2 mA 有効な角度：<15° 測定可能距離：2cm~400cm 分解能：0.3cm

■ 結線

VCC -> 5V, 2.2mA

Trig -> minimum 10 microSec High

Echo ->

GND -> GND

■ UltraSonic sensor test source

```
// Arduino
# include "DigiKeyboard.h"
# define echoPin 0 // PB0
# define trigPin 2 // PB2

unsigned int duration;
void setup () {
  pinMode ( echoPin, INPUT );
  pinMode ( trigPin, OUTPUT );
}

void loop () {
  DigiKeyboard.sendKeyStroke (0);

  digitalWrite (trigPin, LOW);
  delayMicroseconds (2);
  digitalWrite ( trigPin, HIGH );
  delayMicroseconds ( 50 );
  digitalWrite ( trigPin, LOW );
  delayMicroseconds ( 50 );

  duration = pulseIn ( echoPin, HIGH, 100000 ); // obserb pulse duration
  // distance = duration * 0.017 -> 400~9000 duration -> 6.8 cm~ 153 cm

  DigiKeyboard.println ("duration");
  DigiKeyboard.delay (200);

  DigiKeyboard.println (duration);
  DigiKeyboard.delay (200);

  delay (1000);
}
```

■ UltraSonic sensor with I2C source

```
//Arduino
#include "TinyWireS.h" // I2C routines
```

```

#define echoPin 3 // PB3
#define trigPin 4 // PB4
#define LED_PIN 1 // PB1
                // SDA PB0
                // SCL PB2
#define I2C_SLAVE_ADDRESS 0x5A // the 7-bit address
#ifndef TWI_RX_BUFFER_SIZE
#define TWI_RX_BUFFER_SIZE ( 16 )
#endif
//

unsigned int duration;
int interval = 100;
int ledState = 0;
unsigned long previousMillis = 0;

volatile uint8_t i2c_regs[] =
{  0xDE,  // initial value of registers
    0xAD,
    0xBE,
    0xEF,
};

volatile byte reg_position;
const byte reg_size = sizeof(i2c_regs);

void requestEvent()
{  reg_position %= reg_size;
   TinyWireS.send(i2c_regs[reg_position]);
   reg_position++;
}

void receiveEvent(uint8_t howMany)
{  // Sanity check
   if ((howMany < 1) || (howMany > TWI_RX_BUFFER_SIZE)) return;

   reg_position = TinyWireS.receive();
   howMany--;

   while(howMany--)
   {
       reg_position %= reg_size;
       i2c_regs[reg_position] = TinyWireS.receive();
       reg_position++;
   }
}

void setup()
{  pinMode( echoPin, INPUT );
   pinMode( trigPin, OUTPUT );
   pinMode( LED_PIN, OUTPUT );
   mt08Blink(LED_PIN, 2);           //LED blink 2-times following bootup

```

```

    TinyWireS.begin(I2C_SLAVE_ADDRESS);
    TinyWireS.onReceive(receiveEvent);
    TinyWireS.onRequest(requestEvent);
}

void loop()
{
    TinyWireS_stop_check();
    //
    unsigned long currentMillis = millis(); // get time in milli second
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite( trigPin, HIGH );
    delayMicroseconds( 50 );
    duration = pulseIn( echoPin, HIGH, 100000 ); // obserb pulse duration
    if (currentMillis - previousMillis >= interval) //wait loop for detector analog delay
    {
        previousMillis = currentMillis;
        //
        i2c_regs[0]=duration & 0xff;; // set data of register0 changed through I2C
        i2c_regs[1]= (duration>>8) & 0xff;
    }
}

void mt08Blink(byte led, byte times)
{
    times *= 2;
    while(times > 0) {
        digitalWrite(led,(times & 0x01) ? LOW : HIGH);
        tws_delay (200);
        times--;
    }
}

```

■ 参考URL

1. https://www.amazon.co.jp/サインスマート（SainSmart）-101-60-142-HC-SR04-超音波距離センサーモジュール-Arduino/dp/B004U8TOE6/ref=sr_1_1?ie=UTF8&qid=1509675562&sr=8-1&keywords=距離センサー
2. Manual -> <http://akizukidenshi.com/download/ds/sainsmar/US-015Manul.pdf>
3. Arduinoでパーツやセンサを使ってみよう～超音波モジュール編 -> <http://deviceplus.jp/hobby/entry016/>

■ RCWL-0516 マイクロ波近接センサー

■ 結線

V3 3.3V regulated output. Max 100mA (?)

GND Ground

OUT Trigger: high (3.3V) if motion detected. 0V normally.

VIN 4 - 28V supply voltage

CDS LDR 10-20k RL, U_LDR > 0.7V = On

repeat trigger time: the default (unpopulated) time is 2s.

■ 参考URL

1. Schematic diagram -> <https://github.com/jdesbonnet/RCWL-0516>
2. <https://plaza.rakuten.co.jp/v62000/diary/201801170000/>

■ APDS-9960 ジェスチャーセンサー

■ 参考URL

1. https://www.amazon.co.jp/GAOHOU-GAOHOUGY-9960LLC-APDS-9960-RGBおよびジェスチャーセンサーモジュールI2CブレイクアウトArduino-GL用/dp/B074CW7NJ6/ref=sr_1_2?s=diy&ie=UTF8&qid=1508933629&sr=1-2&keywords=ジェスチャーセンサ
2. <http://pdf1.alldatasheet.com/datasheet-pdf/view/918047/AVAGO/APDS-9960.html>
3. <http://akizukidenshi.com/download/ds/akizuki/ae-apds9960.pdf>

■ IR 近接センサー

■ Reference

1. https://www.amazon.co.jp/KKHMF-LM393-IR赤外線障害物回避センサモジュール-Arduino用/dp/B010UZQ8J6/ref=pd_sbs_60_2?_encoding=UTF8&psc=1&refRID=5A19F34QKXWZAKJK0BY4

■ UM2039 距離計

赤外線ダイオードを使った距離計

■ Reference

1. UM2039 User Manual -> <http://strawberry-linux.com/pub/en.DM00279088.pdf>
2. VL53L0X TOFレーザー測距センサモジュール -> <http://strawberry-linux.com/catalog/items?code=15310>
3. VL53L0X ranging module cover window guidelines -> <http://strawberry-linux.com/pub/en.DM00326504.pdf>

7. A/D 変換器

■ ADS1115(ADS1113, ADS1114,)

■ 結線図

VDD: 2~5.5V

ADDR -> GND -> 0x48, -> VDD -> 0x49

■ Reference

1. ADS111x i2c 16bit ADC -> <http://www.ti.com/lit/ds/symlink/ads1114.pdf>
2. 購入先 -> https://www.amazon.co.jp/ADS1115-4チャンネル-16ビット-Arduino-Rpiと互換/dp/B06XR84YVC/ref=sr_1_1?s=electronics&ie=UTF8&qid=1508943861&sr=1-1&keywords=ADS1115

コラム センシング・モジュールはどうあるべきか

ここで取り上げたATiny85にもA/Dコンバータが内蔵されているように、様々なセンシング・モジュールの小型化と低価格化が進んだ現在、モジュールをクラスタ化し、制御プログラムもこれに一体化してパッケージとした方が、トータルな開発時間とコストが削減できる可能性を排除すべきではないだろう。

つまり、たとえパッケージに、使う予定のないセンシング・モジュールが組み込まれていたとしても、コストとスペースに問題がなければ、あらかじめ準備されたパッケージをなるべく使うという考え方である。

であれば、例えば、ATiny85にセンシング・モジュールの初期化プログラムその他を組み込んだ上で、これをマスターであるRaspberryPiに接続して、Mathematicaでデータを利用するという方式が考えられよう。

各種ディスプレイ

Display sub-systems

RaspberryPi上で計算された結果は、何らかの形でこれを出力する必要がある。OLED(organic electroluminescence:有機EL)は小型であるものの、文字を表示するのに適しているし、より大型の表示装置が必要な場合には、RaspberryPiのHDMIコネクタにLCDを接続して、これに出力することが可能である。

1. SSD1306 有機ディスプレイ

■ デバイスのサイズ

SSD1306を使ったOLEDディスプレイには、**1.3 inch**と**0.96 inch**サイズの二種類が販売されている。小さいサイズのディスプレイに、高さ8ドットの文字を表示するのは、視認性の問題が起きるかも知れない。このような場合には、大きなフォント、例えば、高さ16ドットのフォントを用意してから、これを上下に二分するなどの方法が考えられる。

■ SSD1306 128*64 dots OLEDの設定

サンプル表示プログラムには、連想で構成されたドット・イメージ・テーブルを必要とする。イメージは、別のプログラム（Fontの節を参照）によって準備される。

■ Sample code part-1: dot image table

以下のテーブルは、各文字が、8 dot分の高さを持つ固有のdot幅のイメージで、表されることを示している。例えば、スペースは、高さ8 dot、幅3 dotの”0”のイメージで表され、“ “→ {“0x00”, “0x00”, “0x00”}と定義される。

各文字が横方向につらな行となつた時に、上の行と下の行の中の文字がつながって見えることのないように、ほとんどの文字が実際には7 dot分しか使わず、1 dot分は隙間となるように調整されている。ただし、特定の文字については、8 dot分が全て使われる場合がある。例えば、“→ {“0x00”, “0x8”, “0x77”, “0x41”, “0x00”, “0x00”}などである。これらの文字は、頻繁に出現することはないという前提のもとに、8 dot分全てを使って、視認しやすさを優先しているのである。

テーブルは、別プログラムによって準備される。

```
fontTable = Association[
  " " -> {"0x00", "0x00", "0x00"}, "!" -> {"0x00", "0x7c", "0x00", "0x00"}, "\" ->
{"0x00", "0x60", "0x60", "0x00", "0x00", "0x00"}, "#" -> {"0x00", "0x3c", "0x68", "0x7c",
"0x28", "0x00", "0x00"}, "$" -> {"0x14", "0x2c", "0x7e", "0x2c", "0x00", "0x00"}, "%" ->
{"0x00", "0x60", "0x50", "0x78", "0x2c", "0x10", "0xc", "0x00", "0x00", "0x00"}, "&" ->
{"0x00", "0x18", "0x64", "0x54", "0xc", "0x1c", "0x00", "0x00"}, "'" -> {"0x00", "0x60",
"0x00", "0x00", "0x00"}, "(" -> {"0x00", "0x3e", "0x40", "0x00", "0x00"}, ")" -> {"0x42",
"0x3c", "0x00", "0x00"}, "*" -> {"0x00", "0x00", "0x70", "0x00", "0x00", "0x00"}, "+" ->
{"0x00", "0x10", "0x10", "0x3c", "0x10", "0x10", "0x00", "0x00", "0x00"}, "," -> {"0x00",
"0x6", "0x00", "0x00"}, "-" -> {"0x8", "0x8", "0x00", "0x00"}, "." -> {"0x00", "0x4",
"0x00", "0x00"}, "/" -> {"0xe", "0x70", "0x00"}, "0" -> {"0x38", "0x44", "0x44", "0x38",
"0x00"}, "1" -> {"0x00", "0x44", "0x7c", "0x4", "0x00"}, "2" -> {"0x4", "0x4c", "0x54",
"0x64", "0x00", "0x00"}, "3" -> {"0x44", "0x54", "0x54", "0x6c", "0x00", "0x00"}, "4" ->
{"0x8", "0x38", "0x48", "0x7c", "0x8", "0x00"}, "5" -> {"0x00", "0x64", "0x64", "0x1c",
```

```

"0x00", "0x00"}, "6" -> {"0x38", "0x64", "0x64", "0x58", "0x00", "0x00"}, "7" -> {"0x00",
"0x44", "0x58", "0x60", "0x00", "0x00"}, "8" -> {"0x6c", "0x54", "0x6c", "0x00", "0x00",
"0x00"}, "9" -> {"0x64", "0x54", "0x54", "0x38", "0x00", "0x00"}, ":" -> {"0x00", "0x24",
"0x00"}, ";" -> {"0x00", "0x26", "0x00", "0x00"}, "<" -> {"0x00", "0x10", "0x18", "0x18",
"0x24", "0x4", "0x00", "0x00"}, "=" -> {"0x00", "0x18", "0x18", "0x18", "0x18", "0x00",
"0x00"}, ">" -> {"0x00", "0x24", "0x28", "0x18", "0x10", "0x10", "0x00", "0x00"}, "?" ->
{"0x40", "0x5c", "0x60", "0x00", "0x00"}, "@" -> {"0x00", "0x38", "0x54", "0xaa", "0xaa",
"0x78", "0x70", "0x00", "0x00", "0x00"}, "A" -> {"0x4", "0x38", "0x68", "0x18", "0x4",
"0x00"}, "B" -> {"0x7c", "0x54", "0x54", "0x6c", "0x00"}, "C" -> {"0x38", "0x44", "0x44",
"0x44", "0x00", "0x00"}, "D" -> {"0x7c", "0x44", "0x44", "0x38", "0x00"}, "E" -> {"0x7c",
"0x54", "0x54", "0x44", "0x00"}, "F" -> {"0x7c", "0x50", "0x50", "0x00", "0x00"}, "G" ->
{"0x38", "0x44", "0x44", "0x54", "0x5c", "0x00"}, "H" -> {"0x7c", "0x10", "0x10", "0x7c",
"0x00"}, "I" -> {"0x7c", "0x00", "0x00"}, "J" -> {"0x00", "0x2", "0x7e", "0x00"}, "K" ->
{"0x7c", "0x10", "0x28", "0x44", "0x00", "0x00"}, "L" -> {"0x7c", "0x4", "0x4", "0x00",
"0x00"}, "M" -> {"0x7c", "0x70", "0x18", "0x60", "0x7c", "0x00"}, "N" -> {"0x7c", "0x30",
"0x8", "0x7c", "0x00"}, "O" -> {"0x38", "0x44", "0x44", "0x44", "0x38", "0x00"}, "P" ->
{"0x7c", "0x50", "0x60", "0x00"}, "Q" -> {"0x38", "0x44", "0x44", "0x46", "0x38",
"0x00"}, "R" -> {"0x7c", "0x50", "0x6c", "0x00", "0x00"}, "S" -> {"0x64", "0x54", "0x54",
"0x4c", "0x00"}, "T" -> {"0x40", "0x40", "0x7c", "0x40", "0x40", "0x00"}, "U" -> {"0x7c",
"0x4", "0x4", "0x7c", "0x00"}, "V" -> {"0x60", "0x18", "0xc", "0x30", "0x40", "0x00"},
"W" -> {"0x60", "0x1c", "0x3c", "0x70", "0xc", "0x3c", "0x40", "0x00"}, "X" -> {"0x4",
"0x68", "0x30", "0x4c", "0x00", "0x00"}, "Y" -> {"0x40", "0x20", "0x1c", "0x60", "0x00",
"0x00"}, "Z" -> {"0x44", "0x5c", "0x64", "0x44", "0x00", "0x00"}, "[" -> {"0x00", "0x7f",
"0x41", "0x00", "0x00", "0x00"}, "\\" -> {"0x70", "0xe", "0x00"}, "]" -> {"0x41", "0x7f",
"0x00", "0x00"}, "^" -> {"0x00", "0x20", "0x40", "0x40", "0x20", "0x00", "0x00"}, "_" ->
{"0x00", "0x2", "0x2", "0x2", "0x2"}, "`" -> {"0x00", "0x40", "0x00", "0x00", "0x00"},
"a" -> {"0x2c", "0x34", "0x3c", "0x00", "0x00"}, "b" -> {"0x7c", "0x24", "0x24", "0x18",
"0x00"}, "c" -> {"0x18", "0x24", "0x24", "0x00", "0x00"}, "d" -> {"0x18", "0x24", "0x24",
"0x7c", "0x00"}, "e" -> {"0x18", "0x34", "0x34", "0x10", "0x00"}, "f" -> {"0x20", "0x7c",
"0x60", "0x00"}, "g" -> {"0x18", "0x26", "0x26", "0x3e", "0x00", "0x00"}, "h" -> {"0x7c",
"0x20", "0x3c", "0x00"}, "i" -> {"0x7c", "0x00"}, "j" -> {"0x2", "0x7e", "0x00", "0x00"},
"k" -> {"0x7c", "0x18", "0x24", "0x00", "0x00"}, "l" -> {"0x7c", "0x00", "0x00"}, "m" ->
{"0x3c", "0x20", "0x3c", "0x20", "0x3c", "0x00", "0x00"}, "n" -> {"0x3c", "0x20", "0x3c",
"0x00"}, "o" -> {"0x18", "0x24", "0x24", "0x18", "0x00"}, "p" -> {"0x3e", "0x24", "0x24",
"0x18", "0x00"}, "q" -> {"0x18", "0x24", "0x24", "0x3e", "0x00"}, "r" -> {"0x3c", "0x20",
"0x20", "0x00"}, "s" -> {"0x00", "0x34", "0x2c", "0x2c", "0x00"}, "t" -> {"0x00", "0x7c",
"0x24", "0x00"}, "u" -> {"0x3c", "0x4", "0x3c", "0x00", "0x00"}, "v" -> {"0x30", "0xc",
"0x1c", "0x20", "0x00"}, "w" -> {"0x30", "0xc", "0x30", "0x18", "0x3c", "0x00", "0x00"},
"x" -> {"0x24", "0x18", "0x38", "0x4", "0x00"}, "y" -> {"0x20", "0x1e", "0x18", "0x20",
"0x00"}, "z" -> {"0x2c", "0x34", "0x24", "0x00", "0x00", "0x00"}, "{" -> {"0x00", "0x8",
"0x77", "0x41", "0x00", "0x00"}, "|" -> {"0x00", "0x7f", "0x00", "0x00", "0x00"}, "}" ->
{"0x00", "0x41", "0x77", "0x8", "0x00", "0x00"}];

```

■ Sample code part-2:SSD1306 driver

```

chipAdr="0x3c";

cmdStr:={"i2cset -y 1",chipAdr,"0x00"};(* command string is indexed by 0x00 *)
dataStr:={"i2cset -y 1",chipAdr,"0x40"};(* data string is indexed by 0x40 *)

rPut[r_]:=WriteLine[process,StringRiffle[Join[cmdStr,{r}]]];(* register setup process *)

setCursor[r_]:= (
  rPut[StringJoin["0xb",ToString[r]]];(* setup cursor *)
  rPut["0x10"];rPut["0x02"] (* 10: set higher nibble is 0, 02: set lower nibble to 2*));

clrLine[r_]:= (
  blnkStr=Join[dataStr,ConstantArray["0x00",32],{"i"}];(* output buffer length is 32 *)
  setCursor[r];
  Table[WriteLine[process,StringRiffle[blnkStr]],{4}];(* one line is composed of 32*4 *)

clrScreen:=Table[clrLine[i],{i,7,0,-1}];(* clear 7 lines *)

dispLine[r_,str_]:= (
  clrLine[r];(* display a line on r-line *)
  setCursor[r];
  cstr=Map[fontTable,Characters[str]]];(* convert string to bitmap *)
  dcom=Map[StringRiffle[Join[dataStr,#{ "i" }]]&,cstr];
  Map[WriteLine[process,#{&,dcom}]);

(* start display *)
process=StartProcess[$SystemShell];

rPut["0x8d"];(*8d:charge pump on*)
rPut["0x14"];(*14:enable charge pump*)
rPut["0xaf"];(*af:display on in normal mode*)
rPut["0xa1"];(*a1:set segment remap to reverse*)
clrScreen;

dispLine[7,"SSD1306 is a single-chip CMOS"];
dispLine[6,"OLED/PLED driver with"];
dispLine[5,"controller for organic/polymer"];

```

■ I2Cインターフェース

VCC = 3.3V/5Vであり、バックライトを点灯するために用いられる。
動作後は、VCCに電圧を印加しなくても、チップは動作は継続するので、チップの内部状態をリセットできない。

■ 結線

Raspberry Pi Pin

- 1 : 3.3 V
- 3 : SDA
- 5 : SCL
- 6 : GND

■ SPIインターフェース

spi library

■ Reference

- | | |
|----------------------------------|---|
| 1. Font converter for SSD 1306 | http://oleddisplay.squix.ch/#/home |
| 2. SSD1306 data sheet | https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf |
| 3. OEL display module data sheet | shop.adafruit.com/datasheets/UG-2832HSWEG04.pdf |
| 4. SPI command line utility | http://ipsolutionscorp.com/raspberry-pi-spi-utility/ |

2. SSD1306のためのドットイメージ生成

Dot-image-table generator for SSD1306

SSD1306のメモリーレイアウトに合わせて、高さ8ドットの領域にビットマップ文字が適切に配置されるように、ドット・イメージ・テーブルを作成する。出来上がったイメージが、SSD1306のドライバーによって、表示装置のメモリーに送られ、文字として認識できるようになる。

あるサイズ（ポイント）の、ビットマップ・フォントは、特定された書体の、各文字をビットがつくるイメージで定義したものであり、文字同士の相対的の大きさは調整されているが、文字同士の間隔や、位置関係は、これとは別に調整しなければならない。

ここでは、ビットマップ・フォントデータを入手した後に、各文字の相対的位置関係を表すデータを別個に入手して、これらの二つのデータから、ドット・イメージ・テーブルを作り出す手順を述べる。

■ Adafruit GFX Font converter for a 7bit high font set to SSD1306 display

SSD1306ディスプレイのための、任意のポイントのビットマップ・フォントを作ってくれる、ページである。

Thanks to "http://oleddisplay.squix.ch/#/home" to convert bit-map font for SSD1306

■ Prepare font bit-map data of 7 bit height

ダウンロードしたビットマップデータをa0に格納し、Adafruit-GFX-Libraryから文字位置の設定ファイルをダウンロードしてb0に格納する。

```
(* character " should be changed to escaped style \" *)
a0 = "0x00, //' '
    0x49,0x24, //'!'
    0xF0, //'\"'
    0x31,0xE5,0x1E,0x50, // '#'
    0x27,0xA7,0xF2,0x00, // '$'
    0xE2,0xC6,0x8D,0x14, // '%'
    0x61,0x0A,0xA6,0x78, // '&'
    0xC0, // ''
    0x6A,0xA0, // '('
    0x95,0x60, // ')'
    0x21,0x08,0x00, // '*'
    0x01,0x3E,0x42,0x00, // '+'
    0x50, // ','
    0xC0, // '-'
    0x40, // '.'
    0x49,0x49,0x00, // '/'
    0x69,0x99,0x60, // '0'
    0x61,0x08,0x47,0x00, // '1'
    0x70,0x88,0x8F,0x00, // '2'
    0xF1,0x61,0xF0, // '3'
    0x32,0x95,0xF1,0x00, // '4'
    0x66,0x11,0x70, // '5'
    0x7E,0x99,0x60, // '6'
    0x71,0x22,0x40, // '7'
    0xF5,0x5E, // '8'
    0xE9,0x71,0xE0, // '9'
    0x41, // ':'
```

```

0x41,0x40, // ';'
0x00,0xB8,0xC1,0x80, // '<'
0x79,0xE0, // '='
0x06,0x0E,0xC8,0x00, // '>'
0xE5,0x24, // '?'
0x31,0x3B,0xF3,0xB9,0x03,0x00, // '@'
0x23,0x14,0xE8,0x80, // 'A'
0xF9,0xE9,0xF0, // 'B'
0x74,0x21,0x07,0x00, // 'C'
0xE9,0x99,0xE0, // 'D'
0xF8,0xE8,0xF0, // 'E'
0xE8,0xE8,0x80, // 'F'
0x7C,0x27,0x17,0x80, // 'G'
0x99,0xF9,0x90, // 'H'
0xF8, // 'I'
0x24,0x92,0xC0, // 'J'
0x95,0x31,0x49,0x00, // 'K'
0x88,0x88,0xE0, // 'L'
0xDE,0xFB,0x58,0x80, // 'M'
0x9D,0xDB,0x90, // 'N'
0x74,0x63,0x17,0x00, // 'O'
0xF7,0x48, // 'P'
0x74,0x63,0x17,0x08, // 'Q'
0xEA,0xCA,0xA0, // 'R'
0xF8,0x61,0xF0, // 'S'
0xF9,0x08,0x42,0x00, // 'T'
0x99,0x99,0xF0, // 'U'
0x8C,0x94,0xC2,0x00, // 'V'
0x93,0x69,0xD3,0x66,0xC0, // 'W'
0x53,0x08,0xA9,0x00, // 'X'
0x92,0x88,0x42,0x00, // 'Y'
0xF1,0x10,0x8F,0x00, // 'Z'
0xEA,0xAC, // '['
0x92,0x24,0x80, // '\'
0xD5,0x5C, // ']'
0x31,0x20, // '^'
0x78, // '_'
0x08, // '`'
0xEE,0xF0, // 'a'
0x8E,0x99,0xE0, // 'b'
0x68,0x86, // 'c'
0x17,0x99,0x70, // 'd'
0x6F,0x86, // 'e'
0x6E,0x44,0x40, // 'f'
0x79,0x97,0x70, // 'g'
0x9E,0xDA, // 'h'
0xF8, // 'i'
0x55,0x70, // 'j'
0x8A,0xCC,0xA0, // 'k'
0xF8, // 'l'
0xFD,0x6B,0x50, // 'm'
0xF6,0xD0, // 'n'
0x69,0x96, // 'o'

```

```

0xE9,0x9E,0x80, // 'p'
0x79,0x97,0x10, // 'q'
0x1E,0x48, // 'r'
0x74,0x37, // 's'
0x46,0x44,0x60, // 't'
0x16,0xDE, // 'u'
0x9A,0x66, // 'v'
0xAA,0xE5,0x92, // 'w'
0xA6,0x69, // 'x'
0x96,0x64,0x40, // 'y'
0xEA,0x70, // 'z'
0x32,0x24,0x22,0x30, // '{'
0xFE, // '|'
0x61,0x08,0x22,0x11,0x80 // '}'
";

```

Adafruitのフォントライブラリーから、文字位置と文字間隔を設定するファイルをダウンロードし、b0に格納する。

```

b0 = "{0,1,1,3,0,0}, //"
{1,3,5,4,0,-5}, //'!'
{3,2,2,5,1,-5}, //'\"'
{ 4, 6, 5, 7, 0, -5 }, // '#'
{ 8, 4, 7, 6, 0, -5 }, // '$'
{ 12, 6, 5, 9, 1, -5 }, // '%'
{ 16, 6, 5, 7, 1, -5 }, // '&'
{ 20, 1, 2, 4, 1, -5 }, // '''
{ 21, 2, 7, 4, 1, -5 }, // '('
{ 23, 2, 7, 4, 0, -5 }, // ')'
{ 25, 5, 4, 6, 0, -5 }, // '*'
{ 28, 5, 5, 8, 1, -5 }, // '+'
{ 32, 2, 2, 4, 0, -1 }, // ','
{ 33, 3, 1, 4, 0, -2 }, // '-'
{ 34, 2, 1, 3, 0, -1 }, // '.'
{ 35, 3, 6, 3, 0, -5 }, // '/'
{ 38, 4, 5, 7, 1, -5 }, // '0'
{ 41, 5, 5, 6, 0, -5 }, // '1'
{ 45, 5, 5, 6, 0, -5 }, // '2'
{ 49, 4, 5, 6, 0, -5 }, // '3'
{ 52, 5, 5, 6, 0, -5 }, // '4'
{ 56, 4, 5, 6, 0, -5 }, // '5'
{ 59, 4, 5, 7, 1, -5 }, // '6'
{ 62, 4, 5, 5, 0, -5 }, // '7'
{ 65, 3, 5, 6, 1, -5 }, // '8'
{ 67, 4, 5, 7, 1, -5 }, // '9'
{ 70, 2, 4, 3, 0, -4 }, // ':'
{ 71, 2, 5, 4, 0, -4 }, // ';'
{ 73, 5, 5, 7, 1, -5 }, // '<'
{ 77, 6, 2, 7, 0, -3 }, // '='
{ 79, 5, 5, 7, 1, -5 }, // '>'
{ 83, 3, 5, 5, 0, -5 }, // '?'
{ 85, 6, 7, 9, 1, -6 }, // '@'
{ 91, 5, 5, 6, 0, -5 }, // 'A'

```

```

{ 95, 4, 5, 7, 1, -5 }, // 'B'
{ 98, 5, 5, 7, 1, -5 }, // 'C'
{ 102, 4, 5, 7, 1, -5 }, // 'D'
{ 105, 4, 5, 6, 1, -5 }, // 'E'
{ 108, 4, 5, 6, 1, -5 }, // 'F'
{ 111, 5, 5, 8, 1, -5 }, // 'G'
{ 115, 4, 5, 7, 1, -5 }, // 'H'
{ 118, 1, 5, 4, 1, -5 }, // 'I'
{ 119, 3, 6, 4, -1, -5 }, // 'J'
{ 122, 5, 5, 6, 1, -5 }, // 'K'
{ 126, 4, 5, 6, 1, -5 }, // 'L'
{ 129, 5, 5, 8, 1, -5 }, // 'M'
{ 133, 4, 5, 7, 1, -5 }, // 'N'
{ 136, 5, 5, 8, 1, -5 }, // 'O'
{ 140, 3, 5, 6, 1, -5 }, // 'P'
{ 142, 5, 6, 8, 1, -5 }, // 'Q'
{ 146, 4, 5, 6, 1, -5 }, // 'R'
{ 149, 4, 5, 7, 1, -5 }, // 'S'
{ 152, 5, 5, 6, 0, -5 }, // 'T'
{ 156, 4, 5, 7, 1, -5 }, // 'U'
{ 159, 5, 5, 6, 0, -5 }, // 'V'
{ 163, 7, 5, 8, 0, -5 }, // 'W'
{ 168, 5, 5, 6, 0, -5 }, // 'X'
{ 172, 5, 5, 6, 0, -5 }, // 'Y'
{ 176, 5, 5, 7, 1, -5 }, // 'Z'
{ 180, 2, 7, 5, 1, -5 }, // '['
{ 182, 3, 6, 3, 0, -5 }, // '\'
{ 185, 2, 7, 4, 0, -5 }, // ']'
{ 187, 6, 2, 7, 0, -5 }, // '^'
{ 189, 5, 1, 5, -1, 1 }, // '_'
{ 190, 3, 2, 5, 0, -6 }, // ``
{ 191, 3, 4, 6, 1, -4 }, // 'a'
{ 193, 4, 5, 7, 1, -5 }, // 'b'
{ 196, 4, 4, 6, 1, -4 }, // 'c'
{ 198, 4, 5, 7, 1, -5 }, // 'd'
{ 201, 4, 4, 6, 1, -4 }, // 'e'
{ 203, 4, 5, 4, 0, -5 }, // 'f'
{ 206, 4, 5, 7, 1, -4 }, // 'g'
{ 209, 3, 5, 6, 1, -5 }, // 'h'
{ 211, 1, 5, 4, 1, -5 }, // 'i'
{ 212, 2, 6, 4, 0, -5 }, // 'j'
{ 214, 4, 5, 6, 1, -5 }, // 'k'
{ 217, 1, 5, 4, 1, -5 }, // 'l'
{ 218, 5, 4, 8, 1, -4 }, // 'm'
{ 221, 3, 4, 6, 1, -4 }, // 'n'
{ 223, 4, 4, 7, 1, -4 }, // 'o'
{ 225, 4, 5, 7, 1, -4 }, // 'p'
{ 228, 4, 5, 7, 1, -4 }, // 'q'
{ 231, 3, 5, 5, 1, -5 }, // 'r'
{ 233, 4, 4, 6, 0, -4 }, // 's'
{ 235, 4, 5, 4, 0, -5 }, // 't'
{ 238, 3, 5, 6, 1, -5 }, // 'u'
{ 240, 4, 4, 5, 0, -4 }, // 'v'

```

```

{ 242, 6, 4, 7, 0, -4 }, // 'w'
{ 245, 4, 4, 5, 0, -4 }, // 'x'
{ 247, 4, 5, 5, 0, -4 }, // 'y'
{ 250, 3, 4, 6, 1, -4 }, // 'z'
{ 252, 4, 7, 6, 0, -5 }, // '{'
{ 256, 1, 8, 4, 1, -5 }, // '|'
{ 257, 5, 7, 6, 0, -5 } // '}'
";

```

■ Converting to memory image

前項で作成したa0,b0データから、ディスプレイにダウンロードするメモリーイメージを作成する。

```

(* preparing font bitmap *)
a1=StringReplace[a0,Whitespace->""];
a2=StringReplace[a1,"/'"->""];
a3=StringReplace[a2,"/'~_~'"->""];
a4=StringReplace[a3,"'"->""];
a5=StringReplace[a4,"0x"->""];
a=StringSplit[a5,""];

(* preparing font position parameter *)
b1=StringReplace[b0,Whitespace->""];
b2=StringReplace[b1,"/'"->""];
b3=StringReplace[b2,"/'~_~'"->""];
b4=StringReplace[b3,"'"->""];
b=ToExpression[b4];

(* get bitmap hex data series for each character *)
begin:=b[[i,1]]+1;
end:=b[[i+1,1]]/;i<94;
end:=Length[a]/;i==94;
c=Table[Take[a,{begin,end}],{i,Length[b]}];

(* combine to long binary before partitioning *)
binDigit=Map[Flatten,Table[Map[IntegerDigits[FromDigits[#,16],2,8]&,c[[i]]],{i,Length[c]}]];

(* getting font style parameters i=1:"space" to 94:"" *)
width=b[[i,2]];
height=Min[7,b[[i,3]]];
xAdvance=b[[i,4]];
xOffset=b[[i,5]];
yOffset=Min[b[[i,6]],0];

(* prepare pads *)
pad=ConstantArray[0,width];
bytePad=ConstantArray[0,8];

(* set padding number *)
upPadding=6+yOffset;
downPadding=Max[8-upPadding-height,0];

(* raw font rectangle *)
rd=Take[Partition[binDigit[[i]],width],height];

(* padding rectangle *)
rt=Join[Table[pad,{upPadding}],rd,Table[pad,{downPadding}]];
rz=Join[Table[bytePad,{xOffset}],Transpose[rt](*,Table[bytePad,{xAdvance-width}]*)];
(*rrz=Map[Reverse,rz];*)

(* prepare form for downloading to SSD1306 *)
hex=IntegerString[Map[FromDigits[#,2]&,rz],16]/."0"->"00";
strSeries=StringReplace[ToString[Map[StringJoin["0x",#]&,hex]],{"","->","{"->","}"->"}"];
Rule[FromCharacterCode[31+i],strSeries]

"."->"0x00 0x4"

i=15;

{FromCharacterCode[31+i],MatrixForm[Transpose[rz]/. 1->x]}

```

アクチュエーター類

モーターとドライバーモジュール、スピーカー

RaspberryPi上で計算された結果を機械的な出力として表すこともできる。サーボモーター、ステッピングモーター、DCモーターなどが代表的な装置である。音声出力もこの類である。なお、モーター類は、本体と出力軸に結合するギア等が、容易に入手できるという観点から選択するのが妥当である。

1. サーボ・モーター

■ SG90 Micro servo

■ Specification

- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10 μ s
- Temperature range: 0 °C – 55 °C

■ Wiring

pwm= orange

vcc=red

groung=brown

SG90 sevo-motor setting

SG90 specification is defined the pulse width as follows where 1-cycle length is 20 ms because 50 Hz pulse series is used.

position 0 deg: 1.45 ms

position 90 deg: 2.4 ms

position -90 deg: 0.5 ms

To set pin output to 50 Hz for SG90, where system clock is 19.2 MHz, should set COUNTER is 192, and REESOLUTION is 2000 as, $19.2 \times 10^6 / 192 / 2000 \rightarrow 50$,

hereafter the duty cycle is set by the equation,

$$\text{DutyCycle} = N / \text{RESOLUTION}.$$

So, command parameter N of “gpio pwm 1 N” becomes as followings.

0 deg: $1.45/20 \rightarrow 145/2000 \rightarrow$ “gpio pwm 1 145”

90 deg: $2.4/20 \rightarrow 240/2000 \rightarrow$ “gpio pwm 1 240”

-90 deg: $0.5/20 \rightarrow 50/2000 \rightarrow$ “gpio pwm 1 50”

```
process=StartProcess[$SystemShell];
```

```
(* setup SG90 servo motor *)
```

```
Writeline[process,"gpio mode 1 pwm"];
```

```
Writeline[process,"gpio pwm-ms"];
```

```
Writeline[process,"gpio pwmc 192"]; (* set couther *)
```

```
Writeline[process,"gpio pwmr 2000"]; (* set resolution *)
```

```
Writeline[process,"gpio pwm 1 60"]; (* set to home position *)
```

```
clock:=Do[WriteLine[process,"gpio pwm 1 "<>ToString[n]];Pause[0.05],{n,250,60,-2}];
```

```
unlock:=Do[WriteLine[process,"gpio pwm 1 "<>ToString[n]];Pause[0.05],{n,60,250,2}];
```



```
While[True,
  unlock;
  Pause[1.0];
  clock];
```

■ Reference

1. LSM9DS0 data sheet <http://ozzmaker.com/wp-content/uploads/2014/12/LSM9DS0.pdf>

■ PCA9685 16Channel 12bit PWM モジュール

■ Reference

1. PCA9685 data sheet <https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf>
2. KKHMF PCA9685 16チャンネル 12-ビット PWM Servo モーター ドライバー IIC モジュール -> https://www.amazon.co.jp/KKHMF-PCA9685-16チャンネル-12-ビット-Arduinoに対応/dp/B078YRJ8D7/ref=sr_1_3?ie=UTF8&qid=1522582672&sr=8-3&keywords=PCA9685

2. ステッパー・モーター

■ 28BYJ-48 Stepper motor

■ Specification

OUTPUT:0.02W-0.65W

WEIGHT:g

NOMINAL:5V

FREQUENCY: 100Hz

REDUCTION RATIO: 64

STEP ANGLE: 5.625deg

■ Wiring

MotorController→Raspberrypi

Vcc Red → pin04 5V

GND Blue → pin06 GND

IN1 Brown → pin15 GPIO22

IN2 Orange → pin13 GPIO27

IN3 Yellow → pin11 GPIO17

IN4 Green → pin07 GIO04

■ Half-Step Switching Sequence

Half-Step Switching Sequence → CW direction (1-2Phase)

Wire# cloor	1	2	3	4	5	6	7	8
4 Orange	-	-						-
3 Yellow		-	-	-				
2 Pink				-	-	-		
1 Blue						-	-	-

```
Column[
  {Text[Style["Half-Step Switching Sequence → CW direction (1-2Phase)", Bold]],
   Grid[{{"Wire# cloor", 1, 2, 3, 4, 5, 6, 7, 8},
         {" 4 Orange", "-", "-", "", "", "", "", "", ""},
         {" 3 Yellow", "", "-", "-", "-", "", "", "", ""},
         {" 2 Pink", "", "", "-", "-", "-", "", "", ""},
         {" 1 Blue", "", "", "", "-", "-", "-", "-", ""}}
    ], Frame → All, ItemSize → {{Full, {2.5}}, Full}, Alignment → {{Left}, Baseline}}}]
```

■ Reference

<http://robocraft.ru/files/datasheet/28BYJ-48.pdf>

■ sample code: using DeviceConfigure/DeviceWrite

GPIOの制御については、Mathematicaの関数を用いることができる。ただし、実行開始までに不規則な設定時間を要することがある。

```
(* For the first time the GPIO configured driver requires system reboot *)
DeviceConfigure["GPIO", {4→"Output", 17→"Output", 27→"Output", 22→"Output"}];

cw:= (DeviceWrite["GPIO", {4→0, 22→1}]);
DeviceWrite["GPIO", {22→0, 27→1}];
DeviceWrite["GPIO", {27→0, 17→1}];
DeviceWrite["GPIO", {17→0, 4→1}];
ccw:= (DeviceWrite["GPIO", {22→0, 4→1}]);
DeviceWrite["GPIO", {4→0, 17→1}];
DeviceWrite["GPIO", {17→0, 27→1}];
DeviceWrite["GPIO", {27→0, 22→1}];
stop:= (DeviceWrite["GPIO", {4→0, 17→0, 27→0, 22→0}]);
```

```
Timing[
  Do[ccw, {i, 64*8}];
  stop; Pause[0.1];
  Do[cw, {i, 64*8}];
  stop]
```

```
{37.26, Null}
```

■ sample code: using GPIO utility with OOP

RaspberryPiに最初から実装されている、GPIO utilityを用いると起動までに時間を要しない。

```
(* OOP style gpio definition *)
gpio[name_[n_]] := Module[{process, gpn = n},

  (* each object has own shell process *)
  set[name_] ^= (process = StartProcess[$SystemShell];
```

```

WriteLine[process, StringRiffle[{"gpio -g mode", gpn, "out"}]]);

(* pause[] is for preventing writeline missing *)
pulse[name] ^= (WriteLine[process, StringRiffle[{"gpio -g write", gpn, "1"}]];
Pause[0.01];
WriteLine[process, StringRiffle[{"gpio -g write", gpn, "0"}]];
Pause[0.01];);

(* construct instances for each pin, and set to output mode *)
{gpio[p4[4]], gpio[p17[17]], gpio[p22[22]], gpio[p27[27]]};
{set[p4], set[p17], set[p22], set[p27]};
cw := {pulse[p4], pulse[p22], pulse[p27], pulse[p17]};
ccw := {pulse[p17], pulse[p27], pulse[p22], pulse[p4]};
Do[cw, {i, 64*8}]; Do[ccw, {i, 64*8}];

```

3. DCモーター

DCブラシモーターは、ブラシ部分に発生する火花によって、ノイズが発生し、条件によっては、RaspberryPiの誤動作を招く場合があるので、適切なノイズ対策を施す必要がある。(Reference 5)

■ RC-300-FT/14270

■ Specification

OUTPUT:0.02W-0.65W

WEIGHT:g

NOMINAL:3V RANGE(V)

NO-LOAD:SPEED:5940r/min,CURRENT:0.031A

■ Reference

1. スペックシート <http://www.standardmotor.net/product/r300-ft-2/?lang=ja#top>

■ RF-500TB-12560

■ Specification

OUTPUT:0.01W-2.0W

WEIGHT:45g

NOMINAL:V RANGE(3-12V)

NO-LOAD:SPEED:18000r/min,CURRENT:A

■ Reference

<https://product.mabuchi-motor.co.jp/detail.html?id=77>

■ RF-500TB-14415

■ Specification

OUTPUT:0.01W-2.0W

WEIGHT:45g

NOMINAL:12V RANGE(4 - 14V)

NO-LOAD:SPEED:5500r/min,CURRENT:0.025A

■ Reference

<https://product.mabuchi-motor.co.jp/detail.html?id=77>

■ RF-500TB-18280

■ Specification

OUTPUT:0.29W

WEIGHT:45g

NOMINAL:3V RANGE()

NO-LOAD:SPEED:2700r/min,CURRENT:0.055A

■ Reference

1. Specification sheet <https://www.jameco.com/Jameco/Products/ProdDS/2158442.pdf>

■ TAMIYA Double Geabox No.168

モーターの出力軸と車輪との間は、一般的にはギアを介して接続するのであるが、車輪とギアがセットになったものを利用するのが、便利である。

■ Specification

Motor: FA-130 *2

WEIGHT:17g

NOMINAL:1.5~3V

TORK:2.55mN・m

NO-LOAD:RPM:12300, CURRENT:200~230mA

OUTPUT SHAFT:2mm

GEAR BOX

GEAR RATIO: 12.7, 38.2, 114.7, 344.2

■ Reference

1. ダブルギアボックス <http://www.tamiya.com/japan/products/70168/index.html>
https://www.banggood.com/ja/6V-210RPM-Encoder-Motor-DC-Gear-Motor-with-Mounting-Bracket-and-Wheel-p-1044064.html?cur_warehouse=CN

■ Encoder Motor

■ Specification

Rated Voltage	DC 6V
No-load Speed	210RPM 0.13A
Max Efficiency	2.0kg.cm/170rpm/2.0W/0.60A
Max Power	5.2kg.cm/110rpm/3.1W/1.10A
Stall Torque	10kg.cm 3.2A
Retarder Reduction Ratio	1 :34
Hall Resolution	Hall x Ratio 34.02 = 341.2PPR

■ Reference

1. ALEXNLD <https://alexnld.com/product/6v-210rpm-encoder-motor-dc-gear-motor-with-mounting-bracket-and-wheel/>

4. DCモーター・ドライバー

■ Pololu md30a (MAX14870/MAX14872)

DCブラシモーターへ給電と正逆制御ができる。

4.5V~36V

Max 2.5A

Pololu MAX14870 Single Brushed DC Motor Driver Carrier

VIN	Reverse-protected power supply input; supply this pin with 4.5 V to 36 V.
GND	Ground connection points for the power supply and control signals.
VM	This pin gives access to the motor power supply after the reverse-voltage protection.
M1	H-bridge output 1.
M2	H-bridge output 2.

PWM	Speed control input; logic high causes the motor to drive.(Max :50 kHz)
DIR	Direction control input
FAULT	Open-drain, active-low fault output during an over-current or over-temperature condition.
EN	Active-low enable input; drive high to tri-state the driver outputs.(Default :LOW)

■ Sample code

RF-500TB-18280モーターを制御する

■ 結線

Motor -> Max14870-brd -> RaspberryPi

```
-----
Motor-Red   -> MAX14870-M2
Motor-Black -> MAX14870-M1
              MAX14870-VIN -> RsPi-5V
              MAX14870-GND -> RsPi-GND
              MAX14870-PWM -> RsPi-BCM18
              MAX14870-DIR -> RsPi-BCM17
```

GPIO pin17によりモーター回転方向を指定する

```
process=StartProcess[$SystemShell];
Writeline[process,"echo 17 > /sys/class/gpio/export"];
Writeline[process,"echo out > /sys/class/gpio/gpio17/direction"];
```

軸からみてCCW方向に設定

```
Writeline[process,"echo 1 > /sys/class/gpio/gpio17/value"];
```

軸からみてCW方向に設定

```
Writeline[process,"echo 0 > /sys/class/gpio/gpio17/value"];
```

/sys/class/pwmはrootとして操作する

```
Writeline[process,"sudo su -"];
Writeline[process,"cd /sys/class/pwm/pwmchip0"];
Writeline[process,"echo 0 > export"];
```

モーターの回転数を、周波数100HzでMPW制御を行なう。周波数が200Hzを超えるとモーターが不安定になる

```
Writeline[process,"echo 1000000 > pwm0/period"];(*setup 100 Hz*)
Writeline[process,"echo 2000000 > pwm0/duty_cycle"];
```

0.1秒間だけモーターを回転させる

```
Writeline[process,"echo 1 > pwm0/enable"];Pause[0.1];
Writeline[process,"echo 0 > pwm0/enable"];
```

■ Reference

1. MAX14870 data sheet https://datasheets.maximintegrated.com/jp/ds/MAX14870-MAX14872_jp.pdf
2. MAX14870 driver carrier <https://www.pololu.com/product/2961>
3. Pololu md30a <https://www.pololu.com/file/0J1337/max14870-single-brushed-dc-motor-driver-carrier-dimensions.pdf>

■ DRV8830

I2Cを通じて、DCブラシモーターへのPWM制御された電圧供給と正逆転制御ができる。

動作電源電圧範囲が2.75V～6.8Vであるので、LiPo電池を直接接続し、これを電源として、モーターを制御することができる。

■ デバイスアドレス設定の留意点

DRV8830デバイスのアドレスの下3ビットは、A0ピンとA1ピンの、VCCもしくはGNDへの接続、ある

いか、どちらにも接続されていない (open) かによって決定される。アドレスの上位4ビットは、1100すなわち0xc0に固定されている。従って、7bitの長さを持つアドレスは、1100000から1101000の範囲、すなわち0x60から0x68までのアドレスを設定できる。例えば、A0ピンとA1ピンの接続の仕方によって、デバイスアドレスが以下のように設定される (データシートのTable 5は、誤解を招く記述となっている)。ただし、アドレス1101000すなわち0x68だけは、アドレスの下4ビットと上位4ビットが1ビット分だけ重なっていて、このルールには沿っていない。

A0->Open A1->GND デバイスアドレス -> 0x61
A0->GND A1->Open デバイスアドレス -> 0x63

■ ISENSEピンの接続

電流制限機能を設定しない場合、ISENSEピンはGNDに接続しなければならない。

■ Control Register (registre 0)

register-0 は電圧の設定と正逆転の設定を指示する。

Grid[[{7, 6, 5, 4, 3, 2, 1, 0}, {"VSET", SpanFromLeft, SpanFromLeft, SpanFromLeft, SpanFromLeft, SpanFromLeft, "IN2", "IN1"}], Frame → All, ItemSize → All]

7	6	5	4	3	2	1	0
"VSET"						"IN2"	"IN1"

VSET : 0x06 - 0x3f

IN1 IN2 condition

0 0 stanby
0 1 reverse
1 0 forward
1 1 break

ex.

0x31: 0.48V forward

0x32: 0.48V reverse

0xfd: 5.06V forward

0xfe: 5.06V reverse

■ Fault Register (registre 1) とUVLOフラッグ

0 x01レジスタを読みこんだ時、0 x40の返り値は、UVLO (undervoltage lockout) フラッグがセットされたことを意味する。UVLOフラッグは、VCC < Vuvlo (2.47 V) の場合にセットされ、VCC > Vuvlo (2.575 V) のときにクリアされる。電源に余裕がない場合に、電圧が瞬間的に低下するためである。この問題を回避するためには、典型的にはVCCとGNDの間に10μF程度のコンデンサを付加する。

■ 結線

Motor -> DRV8830-brd -> RaspberryPi

DRV8830-GND -> RsPi-GND

DRV8830-VCC -> RsPi-5V (Bypass capacitor 10uF to GND)

DRV8830-SDA -> RsPi-SDA

DRV8830-SCL -> RsPi-SCL

Motor-Red -> DRV8830-OUT1

Motor-Black -> DRV8830-OUT2

DRV8830-ISENSE -> GND

DRV8830-A0

DRV8830-A1

■ Sample code

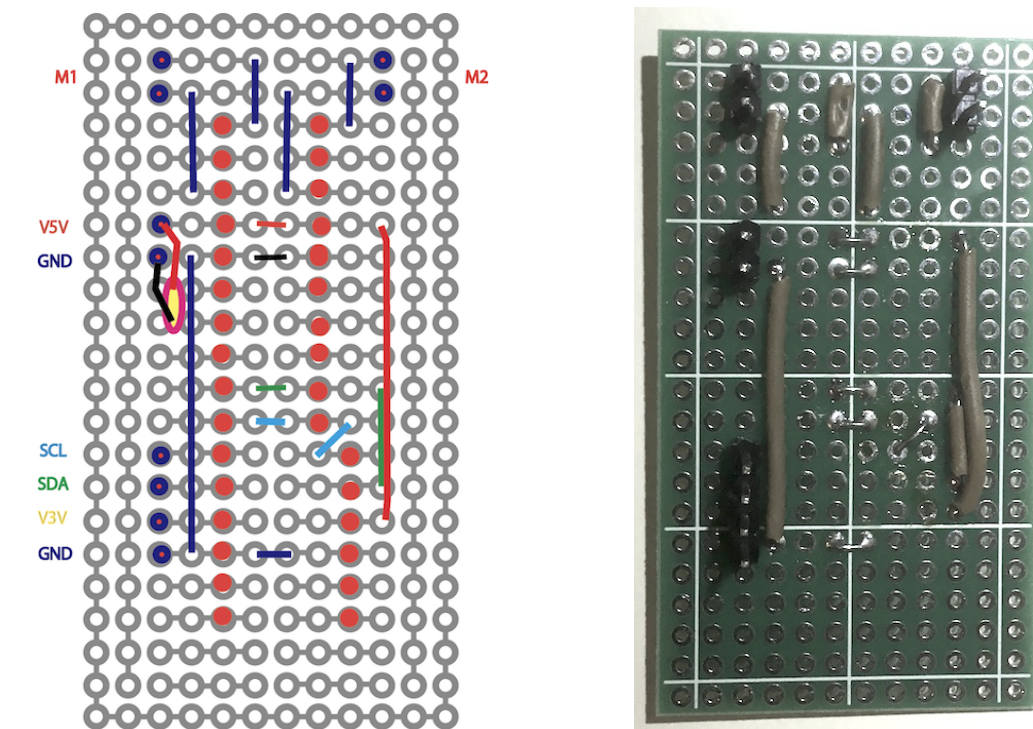
ブラシモーターを0.1秒回転させ、続いて、0.1秒間逆回転させる例

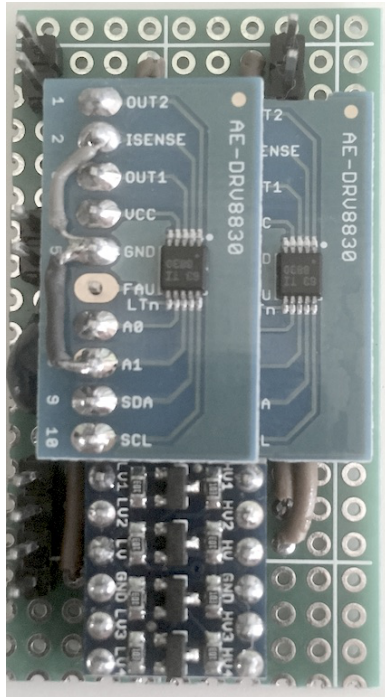
```
chipAdr = "0x61";
process = StartProcess[$SystemShell];
rGet := (WriteLine[process, StringRiffle[{"i2cget -y 1", chipAdr, "0x01"}]];
ReadLine[process]);
rPut[d_] := WriteLine[process, StringRiffle[{"i2cset -y 1", chipAdr, "0x00", d}]];

rPut["0x5e"]; Pause[0.1]; rPut["0x00"];
rPut["0x5d"]; Pause[0.1]; rPut["0x00"];
```

■ Sample configuration

2つのDCブラシモーターを動作させるために、DRV8830ボードを2チャンネル分用意し、これをブレッドボードタイプの、両面スルーホール基盤にセットした例を示す。下図の左は、配線とピンの配置図、中は配線後の写真、右がその完成図である





ブレッドボードタイプの両面スルーホール基盤の配線図
I2Cレベルシフター
c: 10microF

配線後

2チャンネルのDRV8830と

■ Reference

1. DRV8830 モジュール スペックシート
2. DRV8830 モジュール説明書
DRV8830.pdf
3. DRV8830 data sheet
4. DRV8830 data sheet
5. マブチモーターのノイズ対策
cles/1602/19/news010.html

<http://akizukidenshi.com/catalog/g/gK-06273/>

<http://akizukidenshi.com/download/ds/akizuki/AE->

<https://strawberry-linux.com/pub/drv8830j.pdf>

<http://www.tij.co.jp/jp/lit/ds/slvsab2g/slvsab2g.pdf>

<http://monoist.atmarkit.co.jp/mn/arti->

5. スピーカー

■ Dynamic speaker

■ Direct Dynamic speaker wiring

speaker + -> 10 micro C -> 100 ohm R -> PWM pin

speaker - -> GND

■ Power Amp

PAM8403: 2*3W power amp

■ 圧電 speaker

圧電素子によるスピーカー。high impedanceなので直接pinに接続できる。

■ Reference

1. 音の出力 -> http://maicommon.ciao.jp/ss/Arduino_g/sound/index.htm
2. PAM8403 data sheet -> <https://www.diodes.com/assets/Datasheets/PAM8403.pdf>
3. SPT08 -> <http://akizukidenshi.com/download/ds/spl/SPL-SPT08.pdf>

コラム アクチュエータと外部パーツ

自分で作り上げたものが動き出すのは、心踊る体験であるので、アクチュエータは特別の位置を占めるかもしれない。問題となるのは、例えば、モーターと車輪を接続する時、モーター軸と車輪をどのように機械的に接続するかという、手段の実現である。

多くの場合、モーターは単独で供給されるのであるが、これに市販される車輪、ギアボックス、リンク機構などに、何らの修正なしに、機械的に結合できることは殆どない。

そこで、モーターとギアボックスと車輪がセットになったものを利用する、あるいはレゴシステムを用いて、機械的部分は全てレゴが供給するパーツで構築するなどが採用されるのである。しかし、当然のことながら、結果として、ギアの選択や、車輪の選択などに極めて強い制約がもたらされるのである。

これを回避するには、外部に特注する、さらには、機械加工のための工作機械の導入などが考えられるのであるが、コンピュータ回りの他のパーツと比較すると、いかにもバランスを欠く。まだ、コストの問題は残るが、3Dプリンタでパーツを作成する準備を開始すべき時期とも考えられる。

インジケータースイッチ

インジケータースイッチ、I/O expander、DAC、などの構成

RaspberryPi上に単純な指示を与える場合には、スイッチ類が必要となり、その結果を簡単にモニターするための、何らかのインジケータが必要となる場面も多い。その他、タイマーやカウンターが使われる場合もあるだろう。

1. インジケータ

■ LED モジュール

GNDに対向したGPIOピンに、ピンヘッダーを差し込んで利用することを意図した、LEDインジケータ・モジュールである。GPIOピンの対として、以下を想定している。

ピンのペア

13(BCM27):14(GND)

29(BCM05):30(GND)

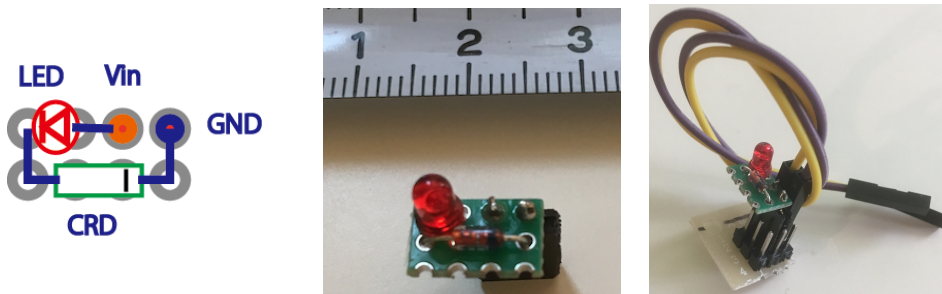
33(BCM13):34(GND)

■ LEDの回路

GPIO -> CRD(E-153) -> LED -> GND

ここでは、LEDに流れる電流を制限するために定電流ダイオードCRD(12~18 mA)を用いたが、抵抗を用いる方が一般的である。

■ 部品配置とモジュールの外観



該当するGPIOピンをHigh (3.3 V) にすると約5mAの電流が流れてLEDが点灯する。
pin header boardに接続するとピンのレベルチェックに使用する。

■ フルカラーLED モジュール

コモンカソードタイプの3色LEDの明るさをPWMで制御し、LEDの色合い変化させることを意図した、LEDインジケータ・モジュールである。ここでは、定電流ダイオードを用いた例をあげる。

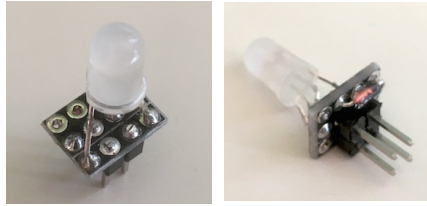
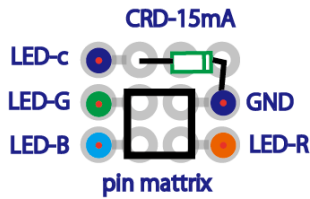
■ LEDの回路

LED Red -> CRD(E-153) -> GND

LED Green -^

LED Blue -^

■ 部品配置とモジュールの外観



■ 振動モーター

定格電圧および電流：DC 3 V, 0.1 A DC 1.5 V, 0.05 A

サイズ：10 x 2.7 mm (D * T) ; ケーブル長：9.5 cm

重量 6 g

購入先

1. https://www.amazon.co.jp/gp/product/B00PZYMCT8/ref=oh_aui_detailpage_o02_s00?ie=UTF8&psc=1

2. スイッチ

■ PIMORONI TOUCH PHAT

I2C接続方式を持つ、静電方式のタッチスイッチである。制御チップとして、CAP1166が用いられている。

Pin接続

- 1: 3.3V Power
- 2: 5V Power
- 2: SDA
- 3: SCL
- 6: GND

6個のキーを、“TOUCH PHAT”のロゴが読める状態にして、左からBack, A, B, C, D, Enterキーと呼ぶことにすれば、レジスタ0x03の内容は、各キーがタッチされると、内容は、0x00からそれぞれのキーについて、0x01,0x02,0x04,0x08,0x10,0x20に書き換えられる。

タッチスイッチの基盤の背面にLEDが設置されており、レジスタ0x74のビットを1とすることで、点灯する。各キーの背面LEDを単独で点灯する場合には、それぞれ、0x20,0x10,0x08,0x04,0x02,0x01をレジスタ0x74に書き込む。

試験的に使用したところ、6個あるうち、4個のLEDが焼損した。

```
chipAdr = "0x2c";
rGet[reg_] := StringTrim[RunProcess[{"i2cget", "-y", "1", chipAdr, reg},
"StandardOutput"]];
rPut[reg_, value_] := StringTrim[RunProcess[{"i2cset", "-y", "1", chipAdr, reg,
value}, "StandardOutput"]];
```

```
map = Association["0x01" -> "0x20", "0x02" -> "0x10", "0x04" -> "0x08", "0x08" ->
"0x04", "0x10" -> "0x02", "0x20" -> "0x01"];
```

```
While[True,
rPut["0x00", "0x00"];
While[(rcode = rGet["0x03"]) == "0x00"];
rPut["0x74", map[rcode]];
rPut["0x74", "0x00"];
]
```

■ Reference

1. pin assignment -> https://pinout.xyz/pinout/touch_phat
2. CAP1166 -> <http://ww1.microchip.com/downloads/en/DeviceDoc/CAP1166.pdf>

■ TTP224 4チャンネルタッチスイッチ

タッチパッド4個の静電スイッチモジュールである。ジャンパーの全てがオープン（デフォルト状態）で、タッチすると出力がHighになり、離すとLowになる。

供給電圧：2.5 - 5.5 V

Jumper selection

TOG, OD, AHLB

```
open, open, open    -> activeHigh, Direct, CMOS output
open, open, short   -> activeLow, Direct, CMOS output
open, short, open   -> activeHigh, Direct, openDrain
open, short, short  -> activeLow, Direct, openDrain
short, open, open   -> powerOnLow, Toggle, CMOS output
short, open, short  -> powerOnHigh, Toggle, CMOS output
short, short, open  -> activeHigh, Toggle, powerOnHighZ
short, short, short -> activeLow, Toggle, powerOnHighZ
```

Jumper selection

LDMR (LPMB) : short/open -> LowPower/FastMode

SM : short/open -> SingleKeyMode/MultiKeyMode

BCM-19,16,26,20に、静電スイッチモジュールTTP224からの4出力を接続し、タッチを検出する。

■ OOPを適用したサンプルプログラム

連想により、4つの静電パッドの名称、a、b、c、dと、それぞれが接続されるBCMピンの対応を定義する

```
gpioPin = Association[a -> 19, b -> 16, c -> 26, d -> 20];
```

ピンのクラスを定義する。パッドへのタッチは、タスクにより0.1秒毎に検出する。
メソッドset[]は、gpioライブラリーへのアクセス手続きと、タスクの生成を行なう。
メソッドstar[]は、定義したタスクを開始する。

```
gpio[nam_] := Module[{process, comstr, schTask},
set[nam[pin_]] ^= (
  process = StartProcess[$SystemShell];
  comstr = "gpio -g read " <> ToString[pin];
  schTask = CreateScheduledTask[WriteLine[process, comstr];
  status[nam] = ReadLine[process], 0.1];
);
start[nam] ^= StartScheduledTask[schTask];
]
```

クラスからインスタンスを生成し、各インスタンスを初期化して、タスクの実行を開始する。

```
Map[gpio, Keys[gpioPin]];
Map[set, KeyValueMap[#1[#2] &, gpioPin]];
```

```
Map[start, Keys[gpioPin]];
```

それぞれの静電パッドをタッチすることにより、ステータスが変化するのが確認できる。

```
{Dynamic[status[a] == "1"], Dynamic[status[b]], Dynamic[status[c]], Dynamic[status[d]]}
```

■ Reference

- [illegible]

■ TTP223 タッチスイッチ

タッチパッド1個の静電スイッチモジュールである。ジャンパーA, Bの両方がオープン（デフォルト状態）で、タッチすると出力がHighになり、離すとLowになる。

供給電圧：2.5 - 5.5 V

Jumper selection

A : open/short -> Active High/Low

B : open/short -> Direct/Toggl

■ Referenceと購入先

- [illegible]

■ タクトスイッチの検出

プルアップされた、あるいはプルダウンされたピンは、電流制限のための抵抗等を必要としないで、これをスイッチとして用いることができる。

初期条件としてプルアップされたGPIOピンがあれば、このピンとGroundとの間にタクトスイッチを置けば、タクトスイッチを押すことによって、該当のGPIOピンを1->0の変化として捉えることができる。あるいは、プルダウンされているGPIOピンと、プルアップされたピンの間にスイッチを置くことで、スイッチの押された状態を0->1の変化として捉えることができる。

また、タクトスイッチをできるだけ簡易に設置することを考えると、対向するピンが、上記の機能を持っている組合せを利用すると、配線が必要でない等の利点がある。

■ 方式 1

例えばGPIO pin7 (BCM07)は、初期状態ではプルアップされているのでその状態を読み込むと、その値は“1”である。一方、このピンに対向するピンはGroundである。そこで、この対向するピンの間にタクトスイッチを接続する。スイッチが押されていない状態で、BCM07ピンのピンの状態を読み込むと、“1”が値として得られるが、スイッチを押した状態では“0”が得られる。

```
(* set interrupt service to GPIO Pin7 *)
```

```
process = StartProcess[$SystemShell];
WriteLine[process, "echo '7' > /sys/class/gpio/export"];
```

```
WriteLine[process, "cat /sys/class/gpio/gpio7/value"];
ReadLine[process]
```

■ 方式2

また、別のピンを用いることもできる。GPIO pin12 (BCM12)は、初期状態ではプルダウンされているのでその状態を読み込むと、その値は”0”である。一方、このピンに対向するピンである、GPIO pin06 (BCM06)は、初期状態では”1”にプルアップされている。そこで、この対向するピンの間にタクトスイッチを接続しておいて、BCM12ピンをINPUTモードとしてから、このピンの状態を読み込むと、”0”が値として得られるが、スイッチを押した状態では”1”が得られる。

```
(* start-up shell process *)
process = StartProcess[$SystemShell];
```

```
(* BCM12 (GPIO_GEN26) pin set to input *)
WriteLine[process, "gpio -g mode 12 in"];
```

```
(* read BCM12 pi status *)
WriteLine[process, "gpio -g read 12"];
ReadLine[process]
```

3. タイマー

■ DS1371 ウォッチドッグタイマー

ハードウェアの動作経過時間の監視や、一定時間後のリセットなどに用いることができる。

i2C address -> 0x68とすると、

check time of day counter

```
$ sudo i2cget - y 1 0 x68 0 x00 w // returns seconds from startup
$ sudo i2cset - y 1 0 x68 0 x00 0 x00 // reset counter byte 0 to 0
$ sudo i2cset - y 1 0 x68 0 x01 0 x00 // reset counter byte 1 to 0
```

Initial conditions are

```
$ sudo i2cget - y 1 0 x68 0 x08 // 0 x80 -> Oscillator was stopped
$ sudo i2cget - y 1 0 x68 0 x07 // 0 x06 -> Interrupt Contorl set to 0, and RS2 and
RS1 set to 1 output to 1 Hz
```

SQW 1Hz output

```
$ sudo i2cset - y 1 0 x68 0 x07 0 x00 // start SQW output to 1 Hz
```

■ 結線

pin1 -> 2X1,X2 -> crystal 32.768kHz

pin3 -> WDS -> restart watchdog counter with signal L to H

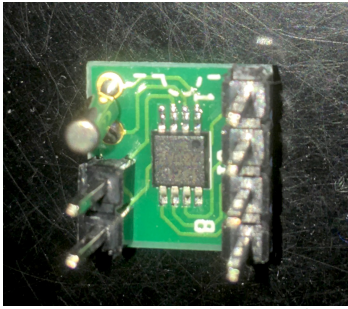
pin4 -> GND

pin5 -> SDA

pin6 -> SCL

pin7 -> SQW/-Int -> requires external pull-up -> sinc current:3mA -> programmable output/

alarm interrupt
pin8 -> VCC -> 3.3V -> 2kΩ -> pin7



DS1371をMSOP変換基盤にハンダ付けした後、
接続用のピンを立てたもの。左上の円筒は水晶発振子

■ Referenceと購入先

1. i2c Watchdog clock data sheet
2. 水晶発振子

<https://datasheets.maximintegrated.com/en/ds/DS1371.pdf>
<http://akizukidenshi.com/catalog/g/gP-04005/>

4. カウンター

■ PCF8593 カウンター

■ Reference

1. PCF8593 data sheet

http://download.siliconexpert.com/pdf-s/2010/10/10/1/27/12/561/phi_/manual/pcf8593_3.pdf

5. DA変換器と音楽デバイス

■ Shareport - AirPlay emulator

Audio機能を使う。

```
$ sudo apt - get update
$ aplay - l
$ Alsamixer
$ speaker - test
```

Shareportのインストール。

```
$ git clone https : // github.com/hendrikw82/shairport.git cd shairport make
$ ./shairport.pl - a AirPi
```

■ Reference

1. AirPlay化して音楽ストリーミング
<https://www.lifehacker.jp/2013/03/130306raspberrypi-airplay.html>
2. VolumioでAirPlay
http://deviceplus.jp/hobby/raspberrypi_entry_029/

■ TI PCM5102A DAC

■ Reference

1. サンハヤト (Sunhayato) ハイレゾリューションオーディオDACモジュール MM - 5102, TI PCM5102A
2. PCM5102Aデータシート, <http://www.tij.co.jp/jp/lit/ds/slas859c/slas859c.pdf>

3. D - class power amp, PAM8403

■ PT08-Z185

圧電スピーカー

Max Input Voltage: 30V p-p

■ Reference

- | | |
|------------------|---|
| 1. SabreBerry+ | http://nw-electric.way-nifty.com/blog/sabreberryplusj.html |
| 2. Volumio用ドライバー | http://nw-electric.way-nifty.com/blog/da/index.html |
| 3. 圧電スピーカー | http://akizukidenshi.com/catalog/g/gP-04118/ |

6. I/Oの拡張と電圧レベルシフター

■ I2C 電圧レベルシフター

I2Cを5Vの電圧で動作する外部モジュールの制御に用いる場合、RaspberryPiのI2Cピンの出力レベルである3.3Vバスとの間の、電圧レベルを変換するモジュールが必要である。IPCA9306は、2chのenable端子付きlevel shifter ICであり、これを利用したモジュールがある。また、2N7002トランジスタをダイレクトに接続した、level shifterモジュールも存在する。簡易的には、抵抗でレベルシフターを代用することもできる。

■ IPCA9306の結線

```
pin1 -> Vref1(low voltage) -> 3.3V
pin2 -> SCL-raspberryPi
pin3 -> SDA-raspberryPi
pin4 -> enable pin
pin5 -> GND
pin6 -> SDA-5V-circuit
pin7 -> SCL-5V-circuit
pin8 -> Vref2(high voltage) -> 5V
```

■ 2N7002(D702)の結線

```
pin1 -> LV1 -> (raspberryPi SDA)
pin2 -> LV2 -> (raspberryPi SCL)
pin3 -> LV -> 3.3V
pin4 -> LV-GND
pin5 -> LV3
pin6 -> LV4
pin7 -> HV4
pin8 -> HV3
pin9 -> HV-GND
pin10 -> HV -> 5V
pin11 -> HV2 -> (SCL)
pin12 -> HV1 -> (SDA)
```

■ Reference

1. PCA9306 module -> <http://akizukidenshi.com/download/ds/akizuki/m5452-ae-pca9306.pdf>
2. PCA9306 -> <http://akizukidenshi.com/download/ds/nxp/pca9306.pdf>
3. 4ch module -> https://www.amazon.co.jp/gp/product/B079GRPDCT/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1

■ Cluster HAT

■ Reference

1. 8086 Support <https://8086.support>
2. Clustar HAT <https://clusterhat.com>
3. boot without SD <https://jyn.jp/raspberrypi-usb-only-boot/>
4. network boot <https://qiita.com/khayate/items/c69ab816fe0cdf54ff64>
- 5.

- USB hub controller

USB2.0の4-PORrハブコントローラチップとしては、GL850Gとその互換品FE1.1Sがある。

■ Reference

1. GL850G <http://www.chip1stop.com/dispDetail.do?partId=GENE-0000002>
2. GL850G data sheet
https://static.chip1stop.com/pdf/product/GENE/GL850G_DATASHEET_103.PDF
3. USB hub-module <http://www.aitendo.com/product/16011>
4. FE1.1S data sheet
[http://aitendo3.sakura.ne.jp/aitendo_data/product_img/pc/FE1.1S/FE1.1s%20Data%20Sheet%20\(Rav.%201.0\).pdf](http://aitendo3.sakura.ne.jp/aitendo_data/product_img/pc/FE1.1S/FE1.1s%20Data%20Sheet%20(Rav.%201.0).pdf)

- PCF8574

I2Cと8bitパラレルバスインターフェース

■ Reference

1. <http://www.tij.co.jp/jp/lit/ml/jajb003/jajb003.pdf>
2. https://www.amazon.co.jp/ノープランド品-STK0151000682-ノープランド品-arduinoのラズベリーパイ用-PCF8574-IO拡張ボードI2C開発ボード/dp/B01FLEOXG6/ref=as_li_ss_tl?ie=UTF8&qid=1495561657&sr=8-3&keywords=PCF8574&linkCode=sl1&tag=deko-0f-22&linkId=edccfcefcb4b56a9e35e49e4ee0d7d7fa

- 1602LCD

PCF8574A + HD44780 LCD Module

■ Reference

1. https://www.amazon.co.jp/EasyWordMall-1602-ブラック-インタフェース-モジュール/dp/B010Q000V2/ref=pd_rhf_dp_30?encoding=UTF8&pd_rd_i=B010Q000V2&pd_rd_r=GJ2GD0QDRVCY0XZCYQ06&pd_rd_w=6sus1&pd_rd_wg=ZDEhw&psc=1&refRID=GJ2GD0QDRVCY0XZCYQ06
2.

コラム スイッチとインジケータ

たとえシステムの自動化を、極限まで推し進めたとしても、シャットダウン・スイッチなどのスイッチ類をなくすることはできないだろうから、さらに小型化を目指す、どのようにアクセスできるようにするか等、は開発項目として残っている。

また、これと対になるインジケータの小型化も必要であろう。市販の砲弾型LEDは、他の部品と比べると、その大きさから言って、バランスを欠くからである。

どちらも通常時には、あまり必要としないが、どうしても無くすることはできないのであるから、パーツとしては、着実に、進歩させる必要があるだろう。

電源

バッテリー、DC/DC、充電器等

RaspberryPiはUSBから給電できるとしても、スタンドアロンで動かす必要もあるだろう。この場面では、バッテリーとバッテリー電圧を適切に変換するコンバーターが必要とされる。

1. 電源モジュール

■ モジュールの選択

RaspberryPiにパワーモジュールを接続する必要がある理由はいくつか挙げることができる。以下のような例が挙げられよう。

- モーターなど外部モジュールが、RaspberryPiが供給できるより高い電圧・電流を必要とする場合
RaspberryPi Zero GPIO -> Level shifter -> motor drive module -> motor <- power module
- リチウムポリマー電池をシステムに内蔵する場合 ~ 5V を必要とする場合
LiPo (3.7V) -> DC/DC (5V) -> RaspberryPi Zero pin 2(5V)
- リチウムポリマー電池の充電器もシステムに内蔵する場合
Lithium battery charger(4.2V) -> LiPo (3.7V) -> DC/DC -> RaspberryPi Zero

各GPIOピンに流せる電流は16mAであり、全体で50mAに制限されているので、より大きな電流を必要とする場合には、パワーモジュールを用意しなければならない。

なお、GPIOピンは、3.3Vの電圧で動作することが想定されているので、例えば、5Vの電圧で作動するモーターをI2Cで制御しようとするとき、GPIOピンに接続されたI2Cと、モータードライバー側のI2Cの間の、異なるI2C電圧を調整するために、level shifterが必要である。

*5Vピンをオープンにしたままで、3.3V出力ピンに3.3V電源を接続する構成は、RaspberryPiに損傷を与える可能性がある。

■ MT3608 DC/DCコンバーター

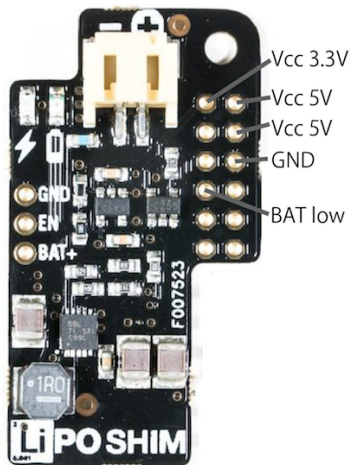
2-24Vの入力に対して、4-14V 2Aの出力を得ることができる。出力電圧はトリマーにより調整する。Shutdownモードにしない時、出力電流が0の時にも、1.6mAの電流が流れるので、バッテリーを入力にした場合にはその消耗に留意する必要がある。また、バッテリー電圧低下検出機能はないので、何らかの方法で、過放電に対処する必要がある。

■ Reference

1. <http://prom-electric.ru/media/MT3608.pdf>
2. https://www.amazon.co.jp/Rasbee-ステップアップ-転換モジュール-2V-24V-5V-28V/dp/B072F25WMG/ref=pd_cp_60_2?_encoding=UTF8&pvc=1&refRID=CD8F6RE6FYVR93N5599S

■ LiPo SHIM (TPS61232 DC/DC converter)

3.7Vのリチウムポリマーバッテリー、あるいは3.3Vの電源から、RaspberryPiの電源に用いるための5V出力を得るための、TPS61232を用いたDC/DCコンバーターである。バッテリー電圧低下検出機能がある。バッテリーとはJSTコネクタにより接続する。



■ Reference

1. LiPo shim specification sheet
2. Zero LiPo at Raspberry Pi GPIO Pinout
3. TPS61232 data sheet

<https://shop.pimoroni.com/products/lipo-shim>
https://pinout.xyz/pinout/zero_lipo
<http://www.ti.com/lit/ds/symlink/tps61232.pdf>

■ AMS1117-3.3ドロップレギュレーター

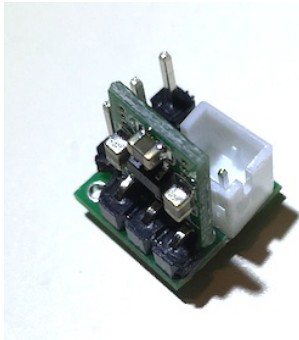
リニア・ドロップ・レギュレータである。3.3V+1.3V以上の入力すなわち4.6V以上の入力電圧から、3.3V/1Aを出力する。

■ Reference

1. <http://www.advanced-monolithic.com/pdf/ds1117.pdf>
2. https://www.amazon.co.jp/gp/product/B01F8PVY2Y/ref=oh_aui_detailpage_o02_s00?ie=UTF8&psc=1

■ LiPo to 3.3V/100mA down & up converter (charge pump regulator)

DC/DC Step-Down& Step-Up Converter 1.8V-5V 3V 3.7V to 3.3V/100mA Power Module
 チャージポンプ方式のコンバーターは、小型化できるが、出力電流に制限がある。



コンバーターと電池プラグで構成したモジュール

■ Reference と購入先

1. N1IF alias HX4002-3.3 datasheet -> <http://www.jiecx.com/files856985665897965/productpdf/2013-7-5/545010255.pdf>
2. 入手先 -> <https://www.ebay.com/itm/271873956073>

■ LiPo to 5V/250mA up converter (charge pump regulator)

DC/DC Step-Down& Step-Up Converter 3.7V-5V to 5V/250mA Power Module
 チャージポンプ方式のコンバーターは、小型化できるが、出力電流に制限がある。

■ Reference と購入先

1. 10A45 datasheet -> <http://www.saiertong.com/files856985665897965/productpdf/2014-1-17/183612173.pdf>
2. <https://ja.aliexpress.com/> -> dc/dc converterで検索

■ LTC3558 LiPo充電+昇降圧DC/DCコンバータモジュール

strawberry-linux社製のLTC3558を使用した、3.7Vリチウムポリマーバッテリーを充電し、3.3Vを出力するモジュールである。

■ 購入先

<https://strawberry-linux.com/catalog/items?code=13558>

2. リチウム電池と充電

■ LiPo電池

リチウムポリマーバッテリーは、高密度エネルギーモジュールであるから、充電や放電には適切な取り扱いのできるモジュールが必要である。入手容易な平型電池の仕様は以下の通り。

公称電圧: 3.7V

バッテリー容量(mAh): 40, 110, 400, 860, 1000, 2000

適合コネクタ: JST 2.0mmピッチ PHコネクタ ベース付ポスト トップ型 2P

■ 購入先

- <https://www.amazon.co.jp/LiPo-バッテリー-400mAh-Soliste-GB231/dp/B010U9HU58>

■ 18650 5V/1A リチウムバッテリー チャージボード

TP4056を用いて、4.2Vの出力電圧で、リチウムポリマーバッテリーを充電し、同時に8205Aにより、バッテリーの過充電・過放電をモニタするモジュールである。

■ 購入先

1. TP4056 charger spec sheet

<https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Prototyping/TP4056.pdf>

2. S8205A バッテリー保護IC

http://datasheet.sii-ic.com/jp/battery_protection/S8205A_B_J.pdf

3. 入手先

https://www.amazon.co.jp/gp/product/B01N949XR0/ref=oh_aui_detailpage_o03_s00?ie=UTF8&psc=1

■ Adafruit Lilon/LiPoly charger

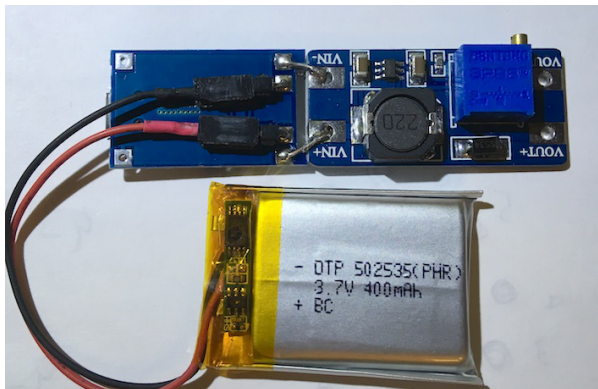
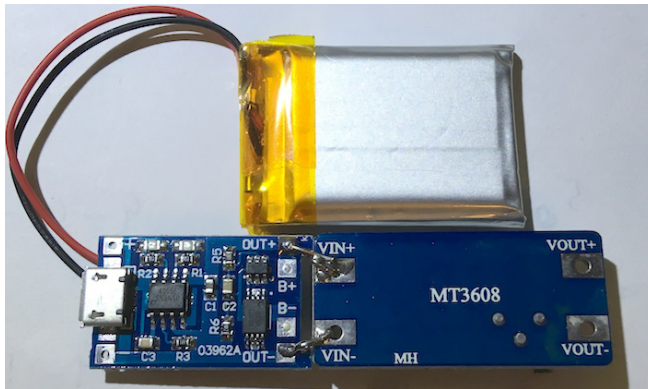
MCP73831を使用していると思われる、3.7V/4.2Vのリチウムポリマー、あるいはリチウムイオンバッテリーを充電するモジュールである。バッテリーとはJSTコネクタにより接続する。

■ 購入先

<https://www.adafruit.com/product/1904>

■ Combination of LiPo battery, DC/DC converter, and LiPo charger

3.7V 400mAhのLiPoとTP4056チャージャー、MT3608 DC/DCコンバーターを組み合わせた例（表面と裏面）。



コラム 電源の重要性は重要

RaspberryPiもArduinoも小型化された結果、システム全体を可搬とする、あるいは自立することが容易になっている。この時に電源を如何に構成するかが、重要な問題となる。幸いにLiPoに代表されるように、バッテリーの進化も著しいので、これを積極的に取り入れるべきであろう。

また、電池電圧をモニタして、電池の容量が失われる前に、適切にシステムをシャットダウンすること、どのようなタイミングで、バッテリーを充電すべきか、そして、全体のパッケージとのバランスをどのように取るか等、まだ、解決すべき問題は多々残っている。

各種パーツ

システムを制作するためのパーツや工具、配線材料等とその購入ガイド

RaspberryPi上にシステムを組み上げるためには、各種配線や基盤、そのための工具等が必要となる。ここでは、ブレッドボードではなくて、ピンヘッダーとジャンパーボードの組合せを用いる。

1. 配線材料

■ GPIO ピンヘッダー

RaspberryPi Zeroには外部接続のためのピンが備えられていない。必要な場合にはピンを半田づけ、あるいは、特殊な形状を持つピンを、治具を用いて圧入する、方法のどちらかを選ぶことになる。このうち、治具による圧入方式を取るのが、Hammer header for RaspberryPi Zeroである。

■ ジャンパーケーブルとジャンパーボード

ピンヘッダーには、通常のピンヘッダーの他に、細ピンヘッダーと通称される、ブレッドボードに直接差し込むための、より細いピンヘッダーがある。QIコネクタは、細ピンヘッダーからは抜け落ちるので、使用することはできない。

ピンヘッダー：ピンサイズ：0.64sq：ピン穴サイズ： ϕ 1.02

細ピンヘッダー：ピンサイズ：0.5sq：ピン穴サイズ： ϕ 0.8

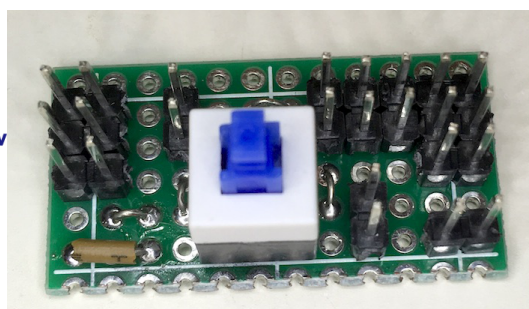
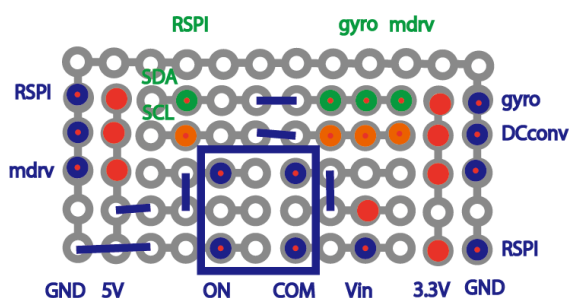
● ジャンパーケーブル

RaspberryPi Zeroと外部モジュールの接続には、RaspberryPi-Zeroに、maleピンヘッダを設置した後に、取付け取り外しが自由な、female-femaleタイプのジャンパーケーブルを用いるのが、便利である。また、ジャンパーケーブルを自作する必要がある場合には以下の方法がある。

1. メス型ピンヘッダーを1pin分だけ切り離して、これにケーブルをハンダ付けする
2. AWG24程度のワイヤケーブルとQIコネクタと呼ばれるコネクタを用いて制作する

● ジャンパーボード

システムを組み上げる場合には、ジャンパーケーブルの分岐が必要となる場合があり、このためのジャンパーボードがあると便利である。なお、5Vと3.3Vの電源を混在させる時には、Level shifterの利用を考慮すること。



5V、3.3V、Ground、I2C、スイッチを含む、ジャンパーボードの例（スイッチはボタンをリリースした時にONとなる配線）

- 購入先

圧入式I/Oピン -> <https://shop.pimoroni.com/products/gpio-hammer-header>
 ブレッドボード・ジャンパーワイヤー -> https://www.amazon.co.jp/gp/product/B00P9C2KE8/ref=oh_aui_detailpage_o07_s00?ie=UTF8&psc=1
 ジャンパ線 コネクタ -> https://www.amazon.co.jp/dp/B017IPVXF0/ref=sspa_dk_detail_5?psc=1
 ジャンパー線コネクタ・ハウジング -> https://www.amazon.co.jp/SODIAL-R-1001Pメスジャンパーケーブルプラグコネクタ-2-54ミリメートル-Arduino用/dp/B00YM3QG5Y/ref=pd_bxgy_328_img_2?_encoding=UTF8&psc=1&refRID=6JESM0TC5SJPXT1HP91A
 ピンヘッダー -> https://www.amazon.co.jp/gp/product/B008QUVM4E/ref=od_aui_detailpages00?ie=UTF8&psc=1
 ワイヤー -> <http://akizukidenshi.com/catalog/g/gP-06756/>

- 参考URL

QIコネクタの圧着方法 1 -> <http://blog.digit-parts.com/archives/51796900.html>
 QIコネクタ圧着方法 2 -> <https://www.youtube.com/watch?v=lrrFPDUSqKk>

- 電子部品基盤

各種部品を取付けるための基盤には、片面、両面スルーホール、
 ブレッドボード型基盤、などの種類がある。それぞれ以下のような特徴

- 片面基盤

ハンダ付けのためのランド（基盤の表面にある、メッキされた領域）が基盤の片面だけにあり、ランドのない側に、部品および部品を接続するためのジャンパー線を配置する。

- 両面スルーホール

基盤のホール（部品のリードを差し込むために空けられている）内面がメッキされていて、基盤両面のランドが電氣的に接続されている。部品を密度高く実装することができる。

- ブレッドボード型基盤

基盤内部に接続ラインが設けられていて、ブレッドボードと同じ使い方ができる。内部のラインを電源ラインやGNDラインとして使うことができ、部品の実装が簡単になる。

- 購入先

片面ブレッドボード型基盤 -> <http://akizukidenshi.com/catalog/g/gP-08242/>
 両面16ホールユニバーサル基盤 -> <http://akizukidenshi.com/catalog/g/gP-02515/>
 両面ブレッドボード型基盤 -> <http://akizukidenshi.com/catalog/g/gP-12356/>

- ICチップ接続基盤

市販されているICチップを使う場合は、チップの小型化が進んでいるので、一旦専用の基盤にチップを取付けて、この基盤の端子と他の部品をケーブルで接続する。ピンヘッダーのサイズに注意する必要がある。

*MSOP(μ -SOP): パッケージのリードピッチが0.65mm
 *SOT23: パッケージのリードピッチが0.95mm
 *SOP8: パッケージのリードピッチが1.27mm

- 購入先

SOT23をDIPに変換する基板	http://akizukidenshi.com/catalog/g/gP-03659/
SOP8をDIPに変換する基板	http://akizukidenshi.com/catalog/g/gP-05154/
MSOPをDIPに変換する基板	http://akizukidenshi.com/catalog/g/gP-08210/
ICパッケージの種類	http://www.chip1stop.com/tutorialContents.do?page=045

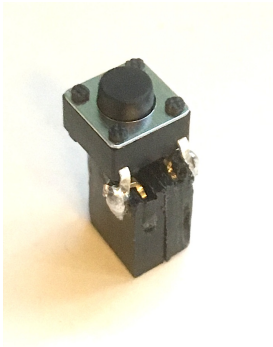
- スイッチ類

- タクトスイッチ

ヘッドを押している間だけ、動作するモメンタリータイプのスイッチである。

- タクトスイッチのモジュール化

femaleピンヘッダを組み合わせて、4ピン分の四角形のヘッダを作り、対向する2つのピンに接続されるヘッダピンに、タクトスイッチをハンダ付けすると、RaspberryPiのピンに差し込むだけでスイッチ回路を形成できる。どこにも接続されていないピンヘッダは、スイッチの支えとして利用できる。



■ オルタネイトスイッチ

ヘッドを押す度に、スイッチの状態が変化するタイプのスイッチである

■ 購入先

- タクトスイッチ

https://www.amazon.co.jp/gp/product/B00QLTT8ZC/ref=oh_aui_detailpage_o03_s00?ie=UTF8&psc=1

- オルタネイトスイッチ

https://www.amazon.co.jp/gp/product/B00LGJ3EPM/ref=oh_aui_detailpage_o09_s00?ie=UTF8&psc=1

■ 抵抗とコンデンサー

抵抗は、プルアップ用やLEDの電流制限用などが、主な用途となるであろう。必要な抵抗値の抵抗を個別に購入するよりも、セットで購入した方が安価である。コンデンサーは、抵抗よりもさらに利用頻度が低いので、セット購入するかどうかは検討すべきである。

LEDの電流制限用途であれば、CRD(定電流ダイオード)の利用も考えられる。

■ 購入先

抵抗セット

https://www.amazon.co.jp/gp/product/B077N1BV3Y/ref=oh_aui_detailpage_o01_s00?ie=UTF8&psc=1

CRD

<http://akizukidenshi.com/catalog/c/ccrd>

■ RaspberryPi Zero ケース

RaspberryPi Zeroを物理的に保護しあるいは、電氣的に絶縁するためのケースである。

■ 購入先

- ケース

https://www.amazon.co.jp/gp/product/B01N8PGOTA/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1

■ 熱収縮チューブ

ピンとケーブルの接続部分の絶縁や保護、複数のケーブルの取りまとめ等に用いる。1mmから複数のサイズを揃えたキットが便利である。ヒートガンによりチューブを熱収縮させるのが基本であるが、ハンダごての先、ライターの炎などで、収縮させることも可能である。

■ 購入先

- チューブ

https://www.amazon.co.jp/gp/product/B0713SHBY4/ref=oh_aui_detailpage_o02_s00?ie=UTF8&psc=1

■ 工具

■ ワイヤストリッパー、圧着工具

ケーブルを自作する場合には、ケーブルを適当な長さに切断した後、ケーブルの両端の被覆を剥ぎ取り、これにコネクタをハンダ付け、あるいは圧着する必要がある。作業数が多い場合には、被覆を剥ぎ取るためのワイヤストリッパー、圧着のための圧着ペンチなどの、専用工具が便利である。

QIコネクタとケーブルを使って、ジャンパーケーブルを作る場合には、圧着工具が必須である。QIコネクタ圧着方法については、Reference 3を参照するとよい。

■ ニッパー

ケーブルの切断、ハンダ付け後の余分なピンの切断、その他の作業に必要である。

■ ラジオペンチ

ハンダ付け時の部品の保持、リード線の曲げ、その他の作業に必要である。

■ ルーペ

微細な部品を扱う場合に必要である。

■ ピンセット

微細な部品を扱う場合に必要である。非磁性である方が使いやすい。

■ 購入先

ワイヤーストリッパー

https://www.amazon.co.jp/gp/product/B000CEAXM4/ref=oh_aui_detailpage_o01_s00?ie=UTF8&psc=1

圧着ペンチ

https://www.amazon.co.jp/gp/product/B002AVVO7K/ref=oh_aui_detailpage_o02_s00?ie=UTF8&psc=1

ニッパー

<http://www.fujiya-kk.com/ja/products/155/>

ラジオペンチ

<http://www.fujiya-kk.com/ja/products/223/>

ルーペ

<https://www.monotaro.com/g/00688759/>

ピンセット

<http://www.ymt21.com/products/brands/dumont.html>

■ ハンダ付け

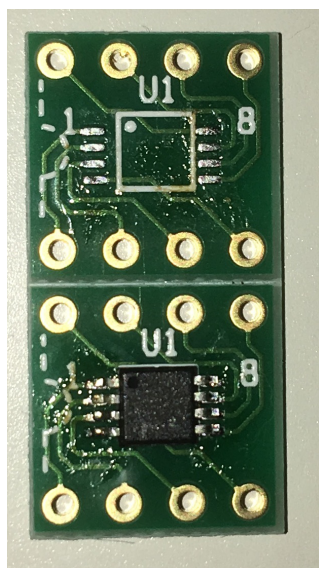
ハンダごては15～30W程度のサイズが適当である。使用時、コテ先は常にハンダで濡れた状態にしておく必要があり、このためには、金属製コテクリーナーが便利である。コテ先の酸化が進んでしまった場合には、コテ先用のリフレッシュ剤により、酸化膜を取り除くことができる。

ハンダは、コテ先の太さと対象物のサイズに合わせて選ぶ。ハンダメッキ（基盤のランド等に予めハンダを薄く付けておくこと）をするためには、ハンダ吸取線が有効である。

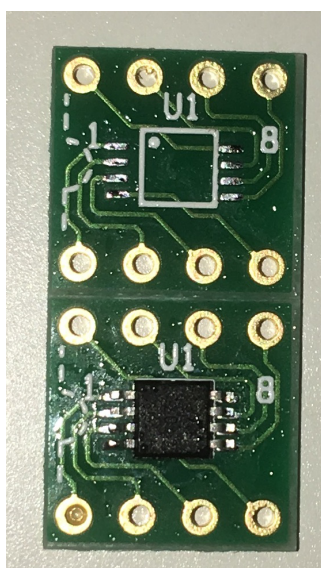
ハンダ付けの際の、部品の保定に、ヒートクリップ（本来はハンダ付けの熱からの部品の保護に使用する）が役立つ場合がある。

ハンダ吸い取り線で余分なハンダを取り除いた後も、フラックスが基盤に残留することがある。残留したフラックスは、パーツクリーナーで除去することができる。

MSOPパッケージ（8ピン 0.65mmピッチ）用DIP変換基盤へのIC接続例



フラックス除去前



フラックス除去後

■ 購入先

ハンダコテ	http://www.goot.jp/handakote/ks-30r/
金属製コテクリーナー	http://www.goot.jp/handakanren/st-40/
コテ先のリフレッシュ剤	http://www.goot.jp/handakanren/bs-2/
ヤニ入りハンダ0.65mm	http://akizukidenshi.com/catalog/g/gT-09556/
ハンダ吸取線	http://akizukidenshi.com/catalog/g/gT-11554/
ヒートクリップ	http://www.goot.jp/handakanren/h-2sl/

2. パーツ等の購入先

各種パーツ（部品）の入手には、計画を適切に進めるために、通信販売を利用すべきである。高性能なパーツが安価で大量に供給されていることを考えると、以下のような点に留意すべきである。

- 時間を要する不良品の交換手続きを避けるため、余裕をもった数を揃える
- 発注から到着までの時間を考慮して、設計構想が完了する以前に、パーツを準備しておく
- 適切なパーツの探索は習慣的に行なう
- 開発時間を短縮するために、モジュール化されたものを組み合わせる
- 小型化・軽量化を目ざし、必要電力の増加を避ける

■ 購入先

安価なパーツを探索する場合はAmazon	www.amazon.co.jp
リチウム電池が購入できるのは千石電商	http://www.sengoku.co.jp/index.php
基盤などの基幹部品が豊富なのは秋月電子	http://akizukidenshi.com/catalog/default.aspx
RaspberryPI Zeroが確実に購入できるのはPimoroni	https://shop.pimoroni.com
各種モジュールが提供されるStrawberry Linux	http://strawberry-linux.com
各種モジュールを探しだせるebay	www.ebay.com
ICチップを1個から購入できるChip One Stop	http://www.chip1stop.com
またはRS	https://jp.rs-online.com/web/generalDisplay.html?id=services/ordering-methods
または、マウサー日本	https://www.mouser.jp/Cart/
その他Raspberryに関連するモジュールはSwitch Science	https://www.switch-science.com
安価なパーツはaitendo	http://www.aitendo.com
Alibabaグループの通販サイト	ja.aliexpress.com

コラム 組み立てという人間的なふるまい

手を使って、何かをつくりあげるといえるのは、手というものの複雑さと、人間にとっての手の存在の大きさを知れば、人間と切り離せない何かを表しているのであろう。

さらに手先を使って、細かなものを組み上げることに、尋常ならざる喜びを秘かに抱いているような人間は、私のことでもある、何かを作りあげていることさえ忘れてしまいがちである。これもまた、手先を動かす作業に、分類し、分割し、成果の再利用を目指す、別の作業を組み込めば、それぞれの作業を固定化できるので、効率化が図れるのである。

さらにパーツを、買う、集める、眺める、調べる、というような楽しみも生まれるので、何かを組み立てるといえるのは、捨て難い喜びとなるのである。

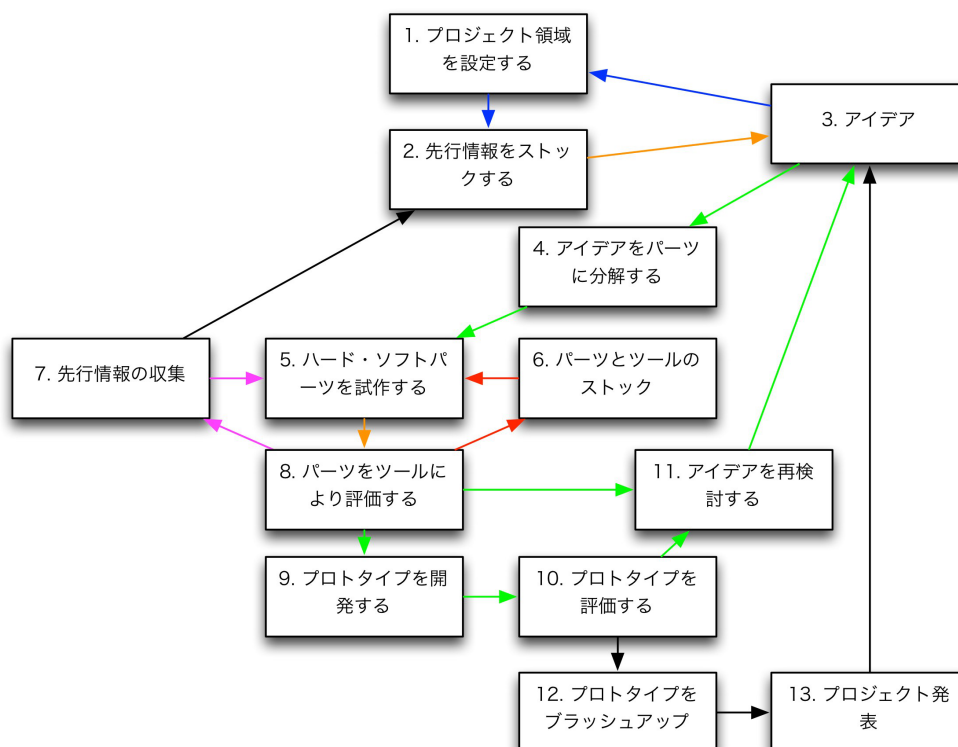
プロジェクトのマネジメント

プロジェクトとは、自分のアイデアを実現する過程であり、その展開はシステム化できる

アイデアは実現しなければ、意味を持たない。アイデアを実現するためには、手順を踏む必要があり、その手順がどのように構成されるかを知って、準備を整えると手順を効率的にすすめることができるだろう。そこで、手順そのものを、システム化することで、プロジェクトの開発が効率化される。さらには、システムに手順途中の失敗への対処を組み込むことで、失敗した手順を、次のプロジェクトに活かすことができる。

そこで、アイデア実現の作業全体を、プロジェクトを構成する要素の集合とみなし、要素間の関連が手順であると考えて、グラフ化してみる。

- プロジェクトの構成要素
- プロジェクトを構成する要素の関連図



- プロジェクトを構成する手順とその作業内容

1. プロジェクト領域の設定

最初に、プロジェクトが発散しないようにカテゴライズする必要がある。どんなに優れたアイデアと思われても、実現しなければ無意味である。実現可能性を考慮した、コストや開発期間を考慮しておく。

2. 先行情報のストック

自分のアイデアに関連しそうな情報をピックアップしておく。あるいは、興味を引いた情報もピックアップして、ストックしておく。

3. アイデア

自分のアイデアを生み出す。

4. アイデアのパーツへの分解

生まれたアイデアを、それを構成する要素に分解する。このとき、構成要素すなわちパーツを、ハードウェア的に実現するか、ソフトウェア的に構築するか、想定しておく。

5. ハードウェア・ソフトウェアパーツの試作

分解された要素をハードウェアあるいはソフトウェアパーツとして試作してみる。

6. パーツとツールのストック

試作したパーツやツールは、この後のアイデアに再利用可能かも知れないので、ストックしておく。

7. 先行情報の収集

試作パーツがプロジェクトを構成する要素として不十分な場合には、先行情報を検索し、利用できる情報を発見したならば、これを試作パーツにフィードバックして、パーツの完成度を高める。ハードウェアの新規購入を含む。

8. パーツをツールによる評価

試作したパーツが、プロトタイプに適用できるかどうか評価する。必要な場合にはツールを用いて評価する。この時、ツールも開発する必要があるかも知れない。

9. プロトタイプの開発

試作パーツを組み合わせて、プロトタイプを開発する。

10. プロトタイプの評価

プロトタイプを評価して、それが、自分のアイデアにマッチしたものであるかどうかを確認する。

11. アイデアの再検討

プロトタイプの評価結果がアイデアとマッチしない場合には、アイデアそのものについて再検討を加える。

12. プロトタイプのブラッシュアップ

プロトタイプの外観の整理やメンテナンス用パーツの取付けなどの、ブラッシュアップ作業を加える

13. プロジェクトの発表

プロトタイプをプロジェクトの成果として発表する。

■ プロジェクトの開発を構成するループ

アイデアを実現するための作業手順とは、プロジェクトの要素である作業が、直線的に連結されたものではない。実際の作業とは、評価と再試行がループしている手順が、幾つもの、折り重なったものである。

作業のループは、よりよい成果を得るために、評価と試行が繰り返される、必要手順であるが、繰り返しからの適切な脱出判断も同時に重要である。特に、作業の手戻り（プロジェクトが進展してから錯誤に気付いて、作業をやり直す必要に迫られること）を避けるためには、ループの存在による手順の停滞の可能性や、ループからの脱出条件が如何に有るべきか、を認識しつつ作業を進捗させる必要がある。

上図（プロジェクトを構成する要素の関連図）によれば、以下のようなループが見出される。

1. 情報蓄積とアイデア創造のループ

興味のある領域について情報を収集・蓄積する、アイデア創造のための、定常的なループである。

項目番号、3 → 1 → 2 → 3 のループ

2. パーツの試作を繰り返すループ

パーツを試作した後、これを評価した結果が不十分な場合には、外部情報を検索し、これをパーツの試作作業にフィードバックさせる、作業ループである。

項目番号、5 → 8 → 7 → 5 のループ

3. パーツの評価結果をアイデアの修正にフィードバックするループ

試作したパーツの評価結果が不十分な場合、アイデアのパーツへの分割が不適切だった可能性があるため、パーツへの分割が可能であるように、アイデアを修正する。

項目番号、8 → 11 → 3 → 4 → 5 → 8 のループ

4. プロトタイプを評価してアイデアを修正するループ

プロトタイプの評価結果は、最初のアイデアが実現されたと看做すには、不十分であることが多い。この時には、アイデアを再構築する必要がある。アイデアに固執するとプロジェクトが完成しない。あるいは、アイデアを放棄する場合があるかも知れない。

項目番号、10 → 11 → 3 → 4 → 5 → 8 → 9 → 10 のループ

5. 試作パーツを再利用する、長期的なループ

パーツをスクラッチから試作するのではなく、パーツのストックから再利用する、より長期的な作業ループである。アイデアの実現のパーツとして不十分であっても、ストックしておけば、再利用の可能性が生じる。ストック量が大きくなれば有効性が増す。

項目番号、8 → 6 → 5 → 8 のループ

6. 外部情報を収集し再利用する、より長期的なループ

関連すると思われる外部情報をスキャンし蓄積しておいて、パーツの開発やアイデアの創造に利用するための、情報再利用ループである。ドキュメントの整備を伴う。

項目番号、8 → 7 → 2 → 3 → 4 → 5 → 8 のループ

7. プロジェクト発表とアイデア創造のモチベーションのループ

アイデアを生み出し続けるためには、プロジェクトを発表し、その評価をアイデア創造のモチベーションとする。

項目番号、13 → 3 → 4 → 5 → 8 → 9 → 10 → 12 → 13 のループ