ハードウェア I/O 2
day2 15:00 ～

KEK, High Energy Accelerator Research Organization
帯名 崇
(takashi.obina@kek.jp)

このセッションでのおはなし

最初に全員でできる実習(GPIO の割り込みでEPICS レコードを動かす) を行います。もし時間が余った人は2, 3 を行います。

1）　共通：GPIOの割り込みでレコードプロセス
2）　Arduinoでアナログ入力（10名交代で）
3）　時間があれば（SNL/Sequencer でLED 点滅）

将来のトピックス

- I2C 経由でデータ取得

実習１）GPIOの割り込みでレコードプロセス

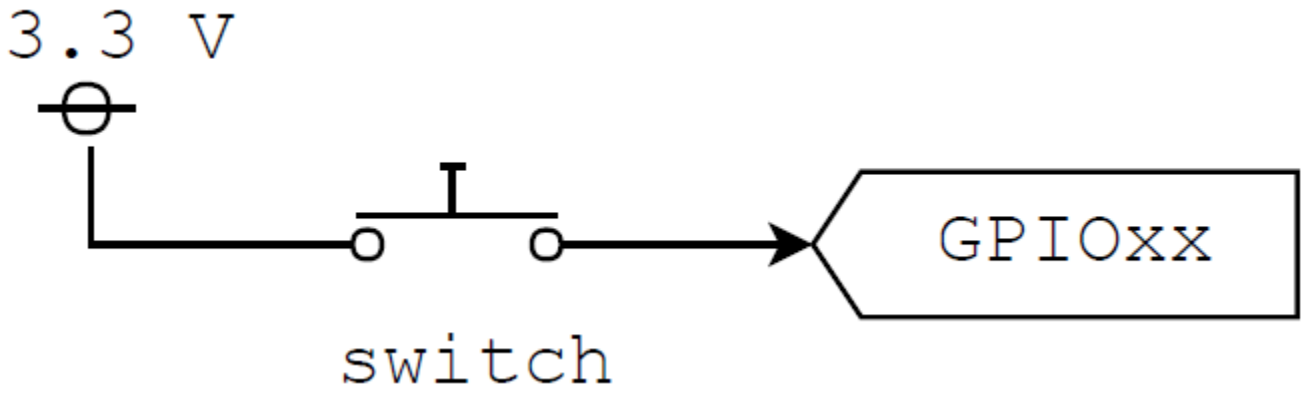
これまでのサンプルで SCAN フィールドは何があったでしょうか？
何が使えるかは<appTop>/dbd/<appName>.dbd ファイルにかいてあります。

```
menu(menuScan) {
    choice(menuScanPassive, "Passive")
    choice(menuScanEvent, "Event")
    choice(menuScanI_O_Intr, "I/O Intr")
    choice(menuScan10_second, "10 second")
    choice(menuScan5_second, "5 second")
    choice(menuScan2_second, "2 second")
    choice(menuScan1_second, "1 second")
    choice(menuScan_5_second, ".5 second")
    choice(menuScan_2_second, ".2 second")
    choice(menuScan_1_second, ".1 second")
}
```

実習１）GPIOの割り込みでレコードプロセス

目的：外部からの電圧入力を検出してEPICS レコードを動かしたい

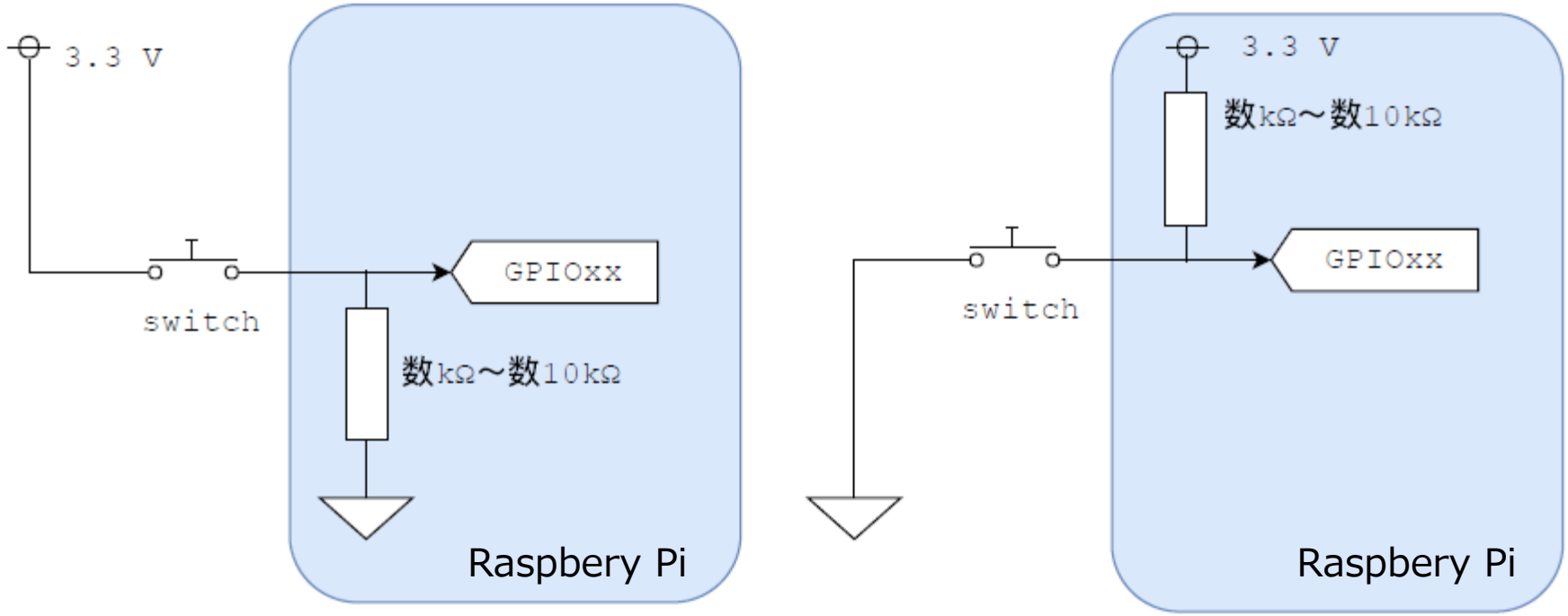
～ソフトで定義した 1 second とかではなく～



例えば、こんな回路を考える

入力がOpenのときは High or Low?

Openだと、入力はどちらになるのかわかりません。
Raspberry Pi の内部ではプルアップまたはプルダウンされています
（これはピンによって異なる）



raspi-gpioコマンド

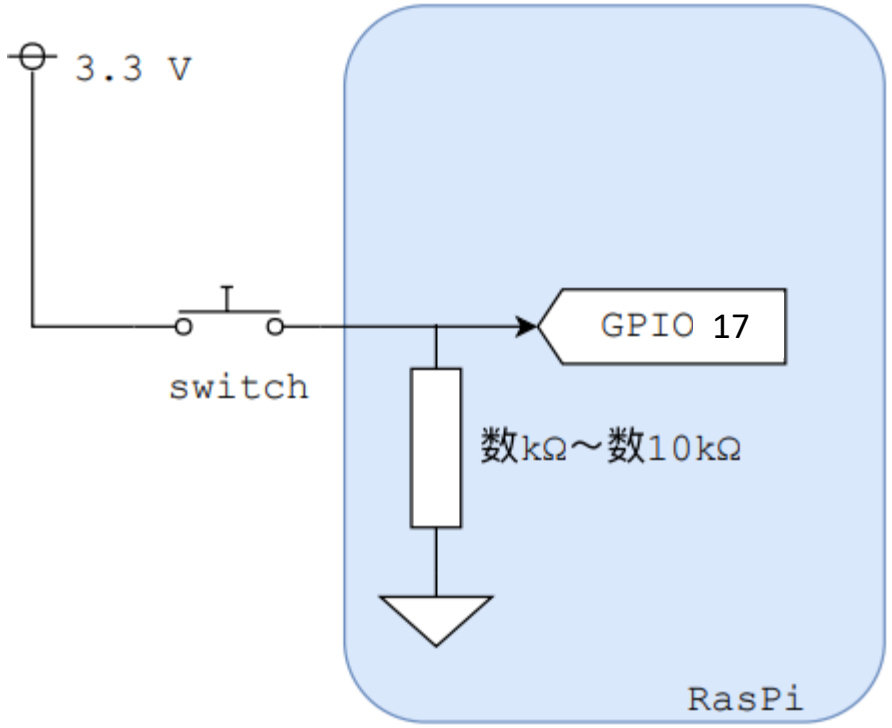
- 以前はWiringPiというパッケージにgpioコマンドがありましたが、今は非推奨となっています
- 替わりに raspi-gpio コマンドを使います。ここで level=1となっているところがpull-up。

```
epics@raspberrypi:~ $ raspi-gpio get
BANK0 (GPIO 0 to 27):
GPIO 0: level=1 fsel=0 func=INPUT
GPIO 1: level=1 fsel=0 func=INPUT
GPIO 2: level=1 fsel=0 func=INPUT
GPIO 3: level=1 fsel=0 func=INPUT
GPIO 4: level=1 fsel=0 func=INPUT
GPIO 5: level=1 fsel=0 func=INPUT
GPIO 6: level=1 fsel=0 func=INPUT
GPIO 7: level=1 fsel=0 func=INPUT
GPIO 8: level=1 fsel=0 func=INPUT
GPIO 9: level=0 fsel=0 func=INPUT
GPIO 10: level=0 fsel=0 func=INPUT
GPIO 11: level=0 fsel=0 func=INPUT
GPIO 12: level=0 fsel=0 func=INPUT
GPIO 13: level=0 fsel=0 func=INPUT
```

```
GPIO 14: level=0 fsel=0 func=INPUT
GPIO 15: level=1 fsel=0 func=INPUT
GPIO 16: level=0 fsel=0 func=INPUT
GPIO 17: level=0 fsel=0 func=INPUT
GPIO 18: level=0 fsel=0 func=INPUT
GPIO 19: level=0 fsel=0 func=INPUT
GPIO 20: level=0 fsel=0 func=INPUT
GPIO 21: level=0 fsel=0 func=INPUT
GPIO 22: level=0 fsel=0 func=INPUT
GPIO 23: level=0 fsel=0 func=INPUT
GPIO 24: level=0 fsel=0 func=INPUT
GPIO 25: level=0 fsel=0 func=INPUT
GPIO 26: level=0 fsel=0 func=INPUT
GPIO 27: level=0 fsel=0 func=INPUT
```

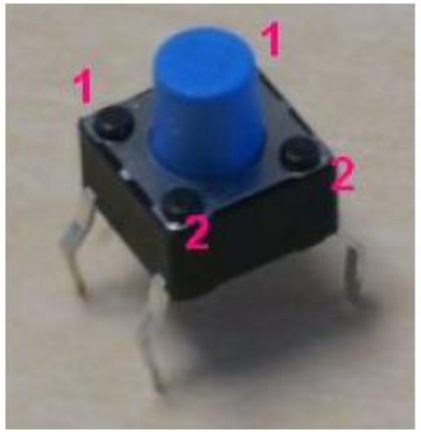
つくりたい回路

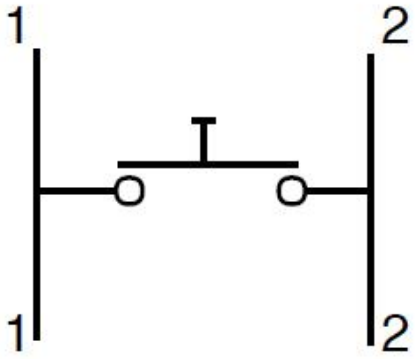
ここでは、プルダウンのピン（GPIO17を例にします）をつかって下の回路を構成しましょう



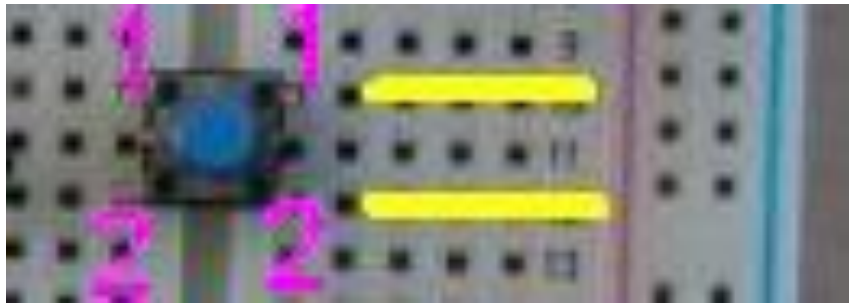
プッシュスイッチ

押している間だけ、1と2の間が導通します

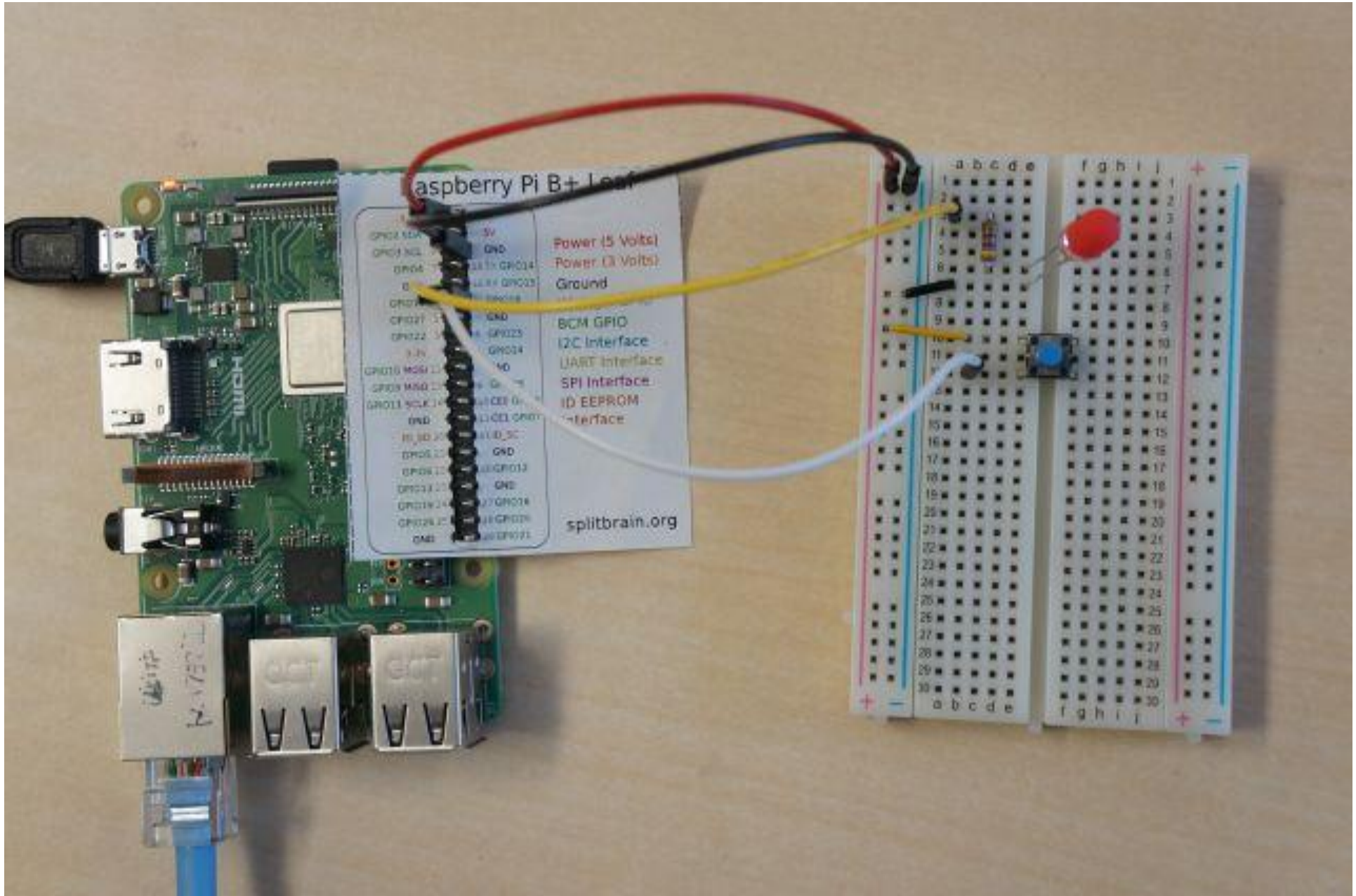




どちらが1なのか2なのか、分かりづらいかもしれません。お勧めはテスターでのチェックですが、下の写真のように「中央の溝」をまたぐように置いてみると、簡単に挿せる方向があると思います（90°回すと足のピンが届かない）。このパーツの場合はこれで向きを確認するのが簡単です。



配線例


Power (5 Volts)
Power (3 Volts)
Ground
BCM GPIO
I2C Interface
UART Interface
SPI Interface
ID EEPROM
splitbrain.org

dbファイルに追記

昨日LED点灯したファイルをそのまま使います。dbファイルは残っていれば18番ピンを追記するだけ。

```
record(bo, "$(head):GPIO17:OUT") {
        field(DTYP, "devgpio")
        field(OUT,  "@17")
        field(SCAN, "Passive") #Default
        field(ZNAM, "OFF")
        field(ONAM, "ON")
}
record(bi, "$(head):GPIO18:IN") {
        field(DTYP, "devgpio")
        field(INP,  "@18")
        field(SCAN, "I/O Intr")
        field(ZNAM, "OFF")
        field(ONAM, "ON")
}
```

その後

いつものように、makeして st.cmd 実行
ボタンを押したらレコードが動くことをcamonitorで確認

その後の実習項目

- 今回追加したレコードをGUI で表示し、ボタンを押すのに同期して表示が変わることを確認する。
- 自分のRaspberry Pi だけでなく、ほかの人のレコードも表示してみる

時間の余った人へ

追加実習項目

- 負論理のスイッチを作成する。プルアップしているピンを使うこと。
- 単にロジックを反転したいだけならばPU/PDどちらでも構わず、ソフトで反転するので十分。実際には(どちらかといえば)プルアップ入力の方が使われることが多い。理由を考える。
- devGpio デバイスサポートのソースコードを読んでみる
- 現在は信号の立ち上がり/立下りエッジの両方を検出してレコードが動いています。場合によっては立ち上がりエッジだけを検出したいこともある（例えばリモコンのボタンとか）ので、ソフトウェア（レコードのリンク）で対処する方法を考えてみる。
- デバイスサポートに手を入れてどちらかのエッジだけで割り込みをかける方法を考えてみる（可能 or 無理まで含め）
- gpio コマンド(wiring pi) を使ってGPIOピンを(EPICSを使わずに)制御してみる。
- コマンドラインからLEDを制御する"だけ"ならば、gpioコマンドで十分。わざわざ EPICS IOC にする意義（御利益）は何なのかを考える。

実習２：アナログ入力信号を測定したい

ADCの入力をどうやって RPi まで運んでいくか？

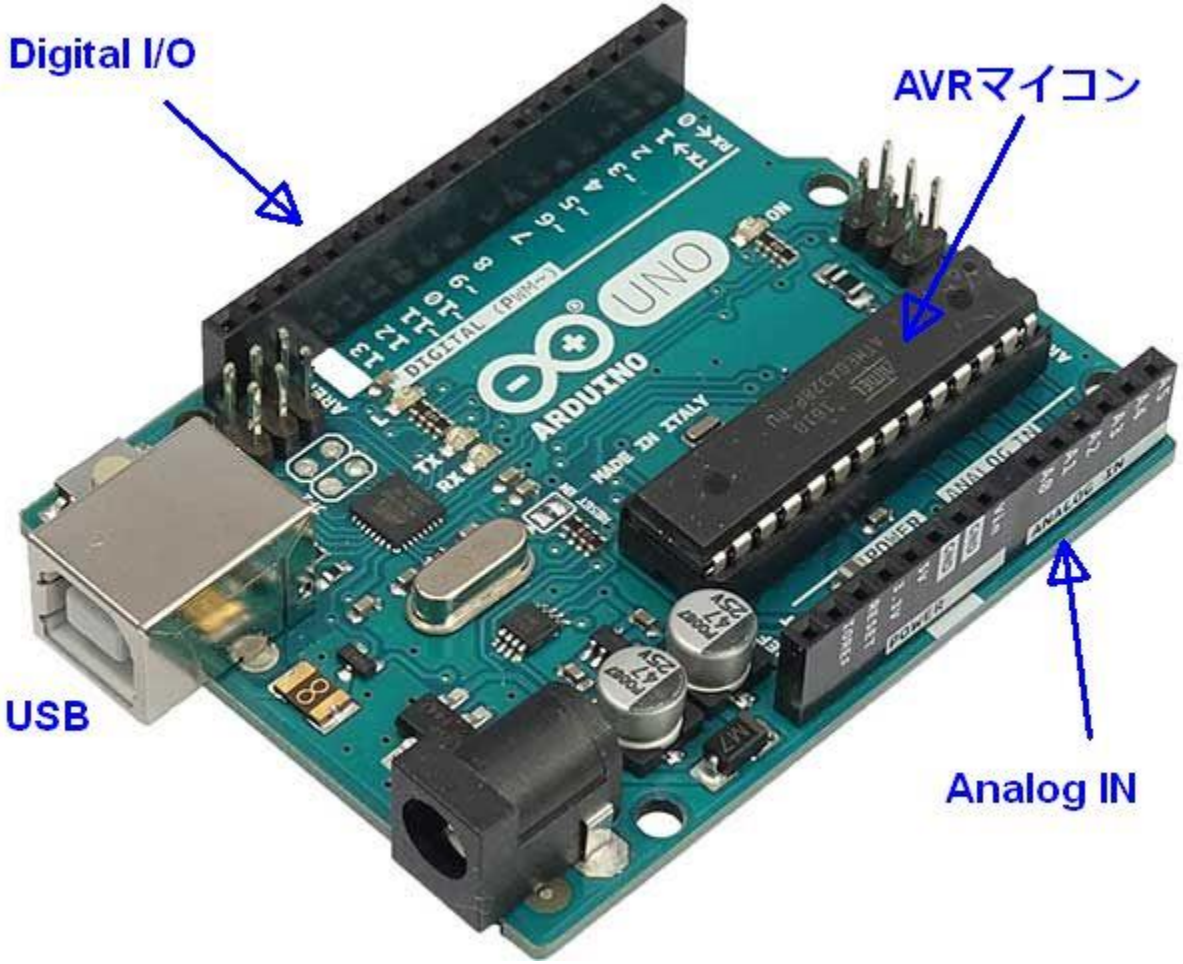
- ADC on Arduino <==[UART(RS232/USB)]==> RPi
- ADC + I2C ==[I2C]==> RPi
- ADC + SPI ==[SPI]==> RPi
- ADC + I2C ==[I2C]== [マイコン] UART(RS232/USB)]==> RPi

など、色々な構成が考えられる

まずはArduinoを使ってアナログデータを取る方法を説明

Arduino

Arduinoを使ってアナログ入力



通信仕様、コマンド

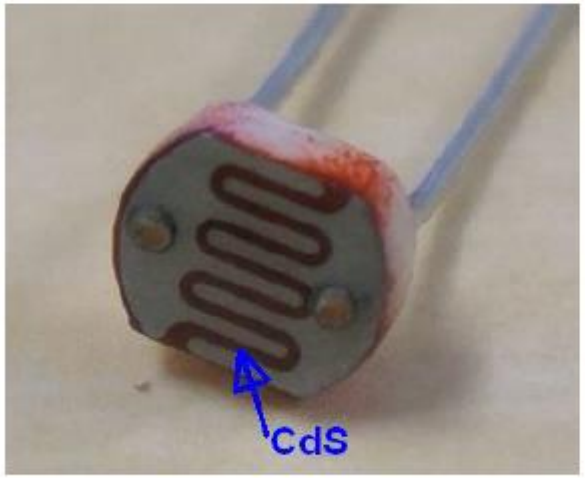
実習用の Arduino には、定期的にアナログ入力値やデジタル入出力を読んでシリアル経由で値を送るプログラムが既に書き込まれています。条件とコマンドは以下の通り。

item	spec
通信速度	115200 bps
parity	none
デリミタ	*

func	command
version	?
analog input	S
analog output	s
digital input	R

例：analog 1 ch を読み込むには “S1“を送る

アナログ照度センサー



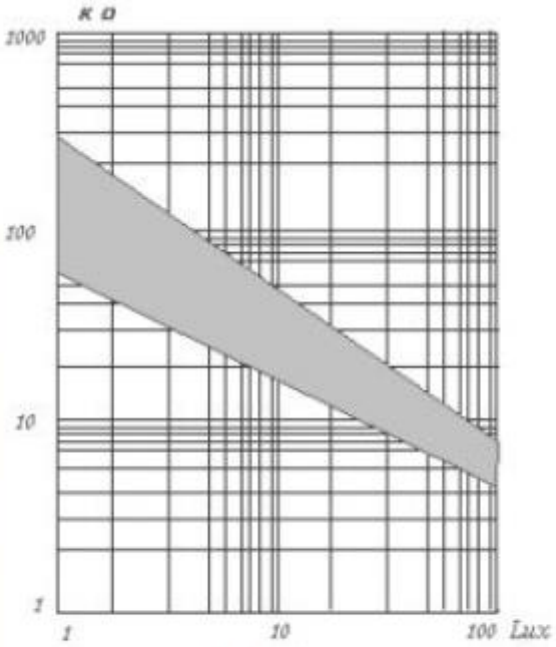


Fig. 3

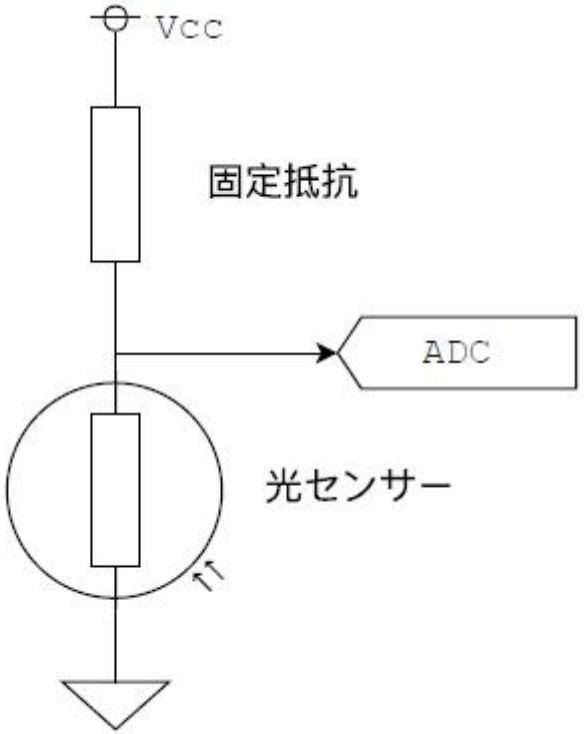


図 7　CdS センサーの写真と特性，回路例。

Terminalでデバッグ

- GNU Screen 使用
  - ？コマンドを送り、バージョン番号とデリミタが返ってくることを確認

- 最後に“CTRL-a k”（killコマンド）を送るか、USB を引き抜くかしてターミナルを切断する

EPICS Application 作成

主な手順は以下の通り:

- configure/RELEASE 編集
- st.cmd に /dev/ttyACM0 あるいは /dev/serial/by-id/usb-…. を記述
- st.cmd に実行フラグを付ける

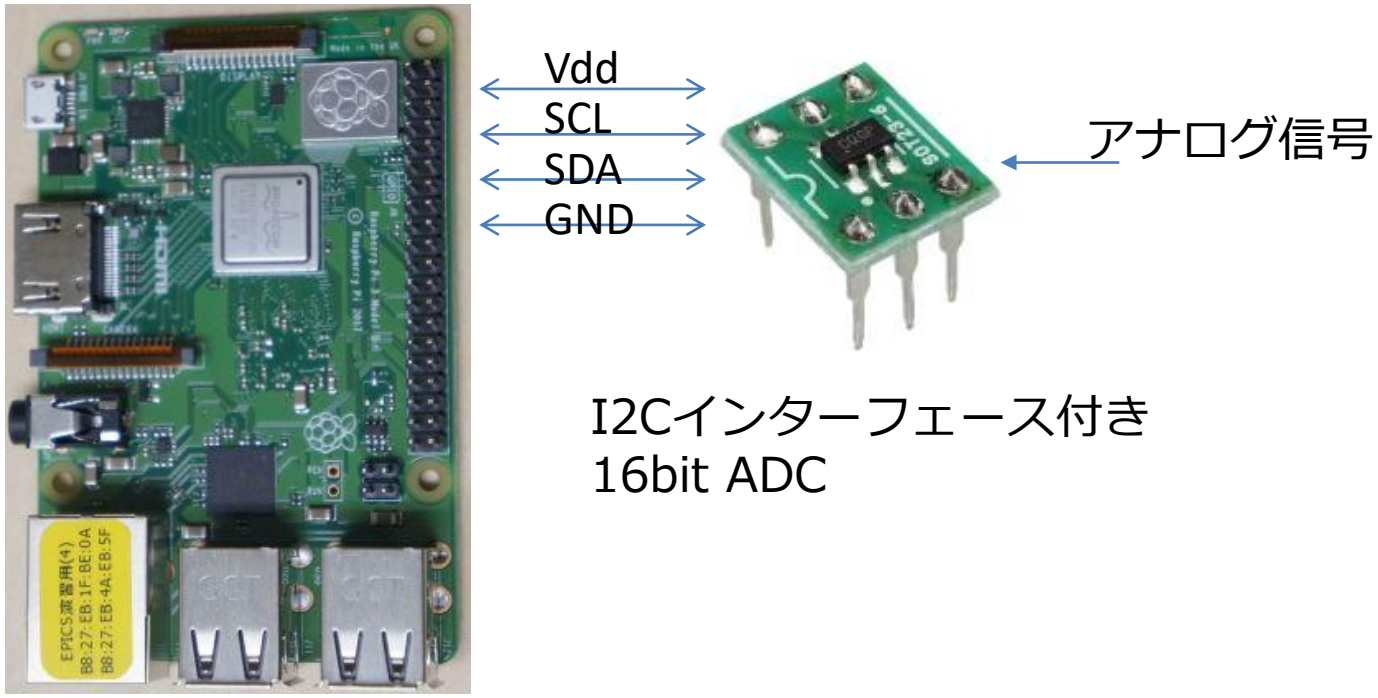
エラー無く起動したら

- dbl でリスト確認
- $(head):Arduino:LUM を camonitor , 手をかざす
- 該当するdbファイルと protocol ファイルを見る
- 他のコマンド(温度センサーなど)をみる

I2CでRPiに取り込む

いままで説明してきたのは Arduino でアナログ信号取り込み

- ADCが10bit（0 – 1024）しかないのはさみしい
- 例えば、16bit ADC を I2C バス経由で取り込むことが出来る



I2Cインターフェース付きセンサー

Vdd
SCL
SDA
GND

I2Cインターフェース付き
温度センサー

他にもいろいろなセンサーがある

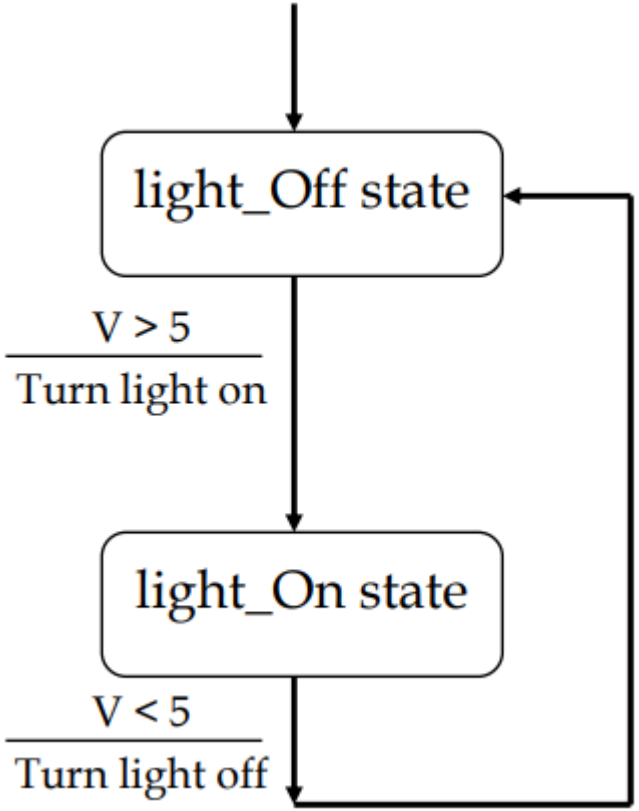
実習３：SNL/Sequencerを使ってLED制御

ここでは EPICS 標準配布物に含まれる State Notation Language (SNL)
および Sequencer について簡単に説明し、一定周期でLEDを点滅させるために使用する。

SNLは状態記述言語と呼ばれる言語の１つで、プログラマは「状態(state)」および「状態間の遷移条件」記述することで制御プログラムを見通し良く、また信頼性・保守性の高いプログラムを書くことができる。

SNLで書きやすいプログラム

2つの“State”がある：Light_ON と Light_OFF
レコードの値によって2つのステートを行き来する




```
program sncExample
double v;
assign v to "{user}:aiExample";
monitor v;

ss ss1 {
  state low {
    when (v > 5.0) {
      printf(“Changing to high¥n");
    } state high
  }
  state high {
    when (v <= 5.0) {
      printf("Changing to low¥n");
    } state low
  }
}
```

１秒間隔でON/OFFする

delay関数を使う: 引数の時間[sec]が経過するとTrueを返す関数。

```
assign out to "{head}:GPIO17:OUT";

ss ss1 {
    state LEDoff {
        when (delay(0.5)) {
            out = 1;
            pvPut(out);
        } state LEDon
    }
    state LEDon {
        when (delay(0.5)) {
            out = 0;
            pvPut(out);
        } state LEDoff
    }
}
```

ちょっと変更して、周期と幅を変更できるように

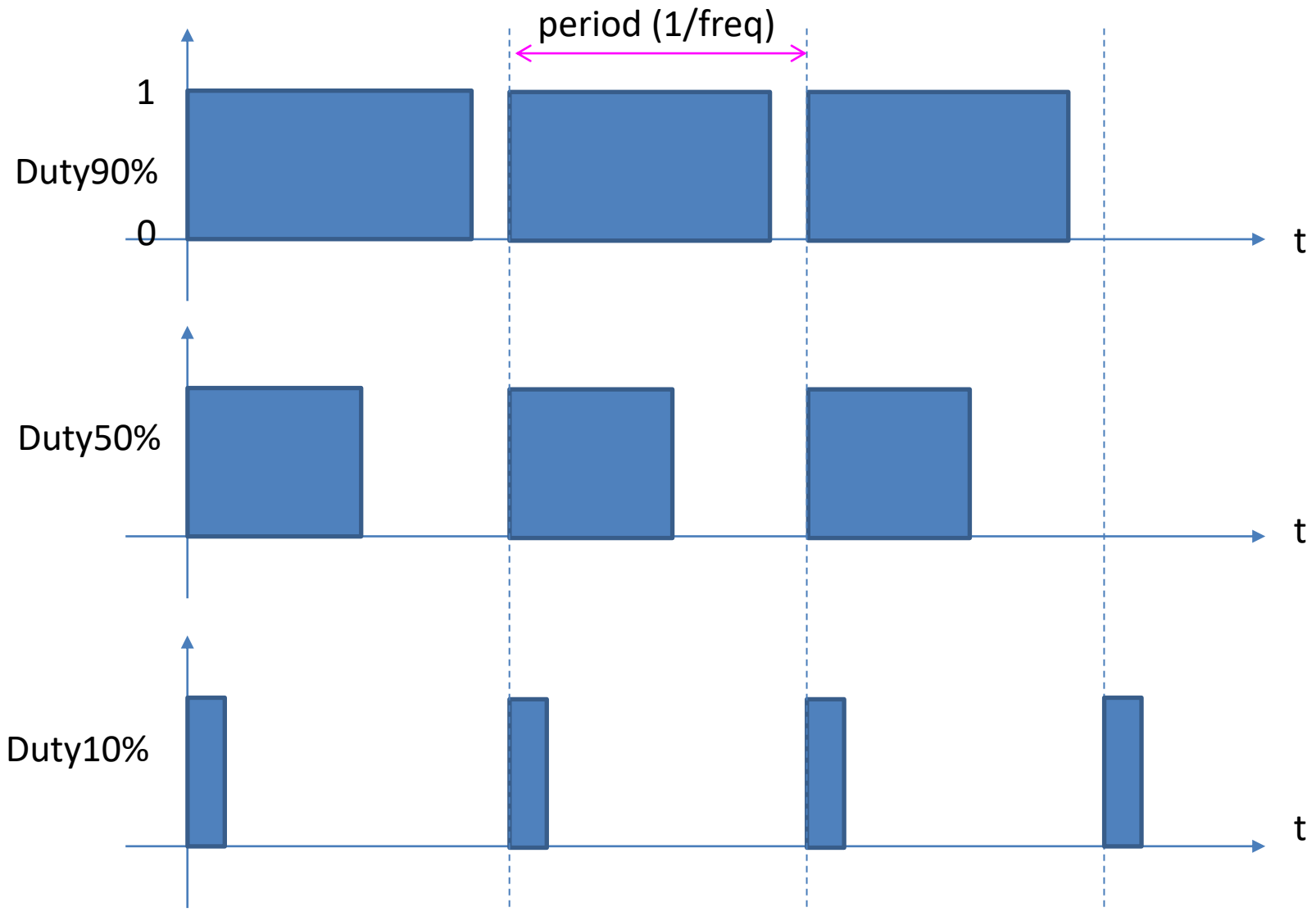
これがテンプレート(pwm)に入っています

```
program sncPwm

assign out to "{head}:GPIO17:OUT";
assign  freq to "{head}:GPIO17:PWM_FREQ";
assign  duty to "{head}:GPIO17:PWM_DUTY";

ss ss1 {
    state LEDoff {
        when (delay(1/freq*(100-duty)/100.0)) {
            out = 1;
            pvPut(out);
        } state LEDon
    }
    state LEDon {
        when (delay(1/freq*(duty)/100.0)) {
            out = 0;
            pvPut(out);
        } state LEDoff
    }
}
```

PWM



共通：CSSに関しての追加演習

昨日触っていますが、(おそらく) 役立つ Tips をいくつか紹介

Data Browser について

- waveform データをアーカイブからとってきて(alan)表示するには?
- カーソルの表示
- グラフに注釈(annotation)を付けたい (例えば室温のデータ2日分取ってきて、どの時刻に何が起きたか記述)。
- 水平・垂直軸の拡大/縮小
- 水平軸を動かす：マウスで軸をつかんで左右に動かす
- 垂直軸を動かす：複数のデータをプロットしておき、そのうち1つのデータの縦方向位置を動かしてみる。
- プロットしたデータを保存する。保存形式やオプションを変えてみる。

OPI について

- Sampleをいくつか触ってみる
- OPIにDataBrowserを貼り付ける