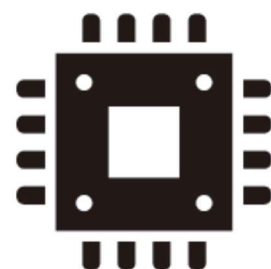
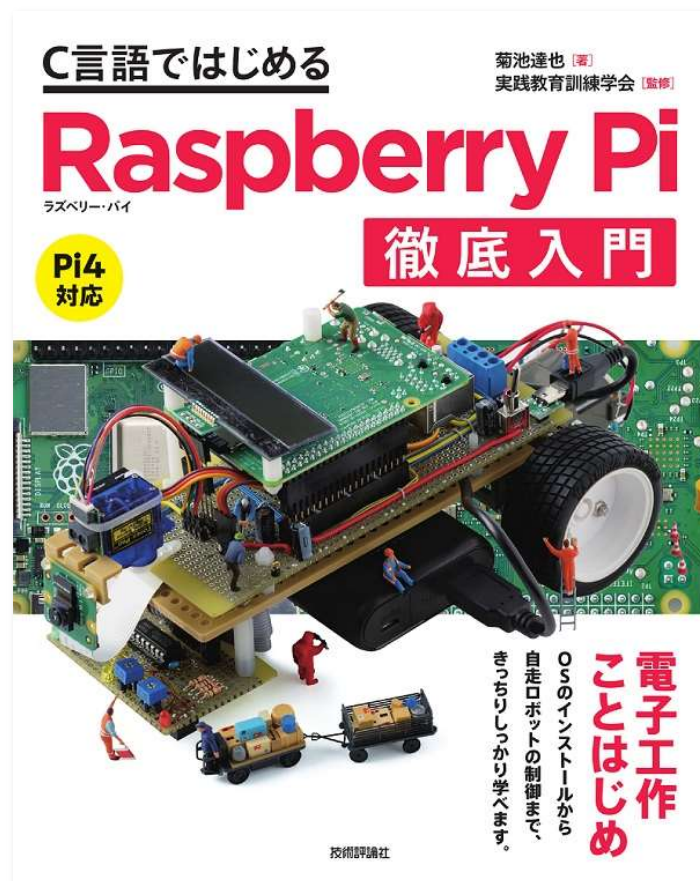


# C 言語ではじめる Raspberry Pi 徹底入門

## 章末問題 Chapter 4 ～ Chapter 9



# 目 次

目 次 .....	2
はじめに .....	4
解答例のソースコードのダウンロードの方法 .....	4
Chapter 4 章末問題 .....	7
【問題 4-1】 LED を2つの点灯パターンで交互に点灯させるプログラム .....	7
【問題 4-2】 2 進数でカウントアップのプログラム .....	8
【問題 4-3】 左シフトを繰り返す LED 点灯のプログラム .....	8
【問題 4-4】 左シフトと右シフトを繰り返す LED 点灯のプログラム .....	9
【問題 4-5】 キーボードから入力された数値を 2 進数で LED に表示するプログラム .....	9
Chapter 5 章末問題 .....	10
【問題 5-1】 タクタイルスイッチで LED を点灯させるプログラム(その1) .....	11
【問題 5-2】 タクタイルスイッチで LED を点灯させるプログラム(その 2) .....	11
【問題 5-3】 タクタイルスイッチで LED を点滅させるプログラム(その 1) .....	12
【問題 5-4】 タクタイルスイッチで LED を点滅させるプログラム(その 2) .....	12
【問題 5-5】 複数の割込み処理をするプログラム .....	13
【参考 5-1】 割込み処理でオルタネート動作させるプログラム .....	14
【参考 5-2】 ビットフィールドで LED と SW の変数を表現するプログラム .....	18
Chapter 6 章末問題 .....	21
【問題 6-1】 ソフトウェア方式 PWM でデューティ比を変えるプログラム .....	21
【問題 6-2】 ソフトウェア方式 PWM で約 440Hz の矩形波を発生させるプログラム ..	22
【問題 6-3】 ハードウェア方式 PWM で約 440Hz の矩形波を発生させるプログラム .....	22
【問題 6-4】 電子ピアノのプログラム .....	23

【問題 6-5】 電子オルゴールのプログラム .....	24
Chapter 7 章末問題 .....	29
【問題 7-1】 温度測定部の関数をライブラリに登録する .....	29
【問題 7-2】 温度を LCD に表示するプログラム (その1) .....	30
【問題 7-3】 摂氏と華氏に切り替えて LCD に表示するプログラム .....	30
【問題 7-4】 測定した温度データをファイルに保存するプログラム .....	31
【参考 7-1】 起動したときに LCD に IP アドレスを表示させる .....	33
Chapter 8 章末問題 .....	35
【問題 8-1】 D/A 変換処理の関数をライブラリに登録する .....	35
【問題 8-2】 A/D 変換処理の関数をライブラリに登録する .....	36
【問題 8-3】 D/A 変換して出力した電圧を A/D 変換するプログラム .....	36
【問題 8-4】 A/D 変換した値を LCD に表示するプログラム .....	37
【問題 8-5】 A/D 変換した値を横棒グラフで LCD に表示するプログラム .....	37
【問題 8-6】 可変抵抗で LED の明るさを制御するプログラム .....	38
【問題 8-7】 可変抵抗で圧電サウダの周波数を制御するプログラム .....	39
【問題 8-8】 電圧を測定してファイルに保存するプログラム .....	40
【問題 8-9】 正弦波の信号を出力するプログラム .....	41
Chapter 9 章末問題 .....	43
【問題 9-1】 静止画のエフェクトを確認するプログラム .....	44
【問題 9-2】 人を検知したら撮影するプログラム .....	44
参考文献/参考資料 .....	46

# はじめに

「C 言語ではじめる Raspberry Pi 徹底入門」(以下、「徹底入門」)の学習内容を利用して、読者が応用力を習得するために章末問題を作成しました。そのため、章末問題を解くためには、「徹底入門」の知識、開発環境、ハードウェア等が必要になります。

章末問題には、複数の割込み処理、電子オルゴールにおける楽譜の配列化、測定した時刻とデータのファイル化、人感センサの活用などの応用的な内容が含まれています。問題の解説については、基本的にソースコードにコメントを記述しましたが、新たに追加した技術的な内容については解説を加えています。「徹底入門」で習得した知識で、理解できると思います。また、問題とは別に、ビットフィールドの使い方などの参考事例も載せました。

解答例のソースコードは著者の GitHub の Web サイトからダウンロードできますが、「答は一つではない」ので、ご自身の着眼点やアイデアを大切に、チャレンジしてください。章末問題の使い方は自由ですが、「徹底入門」の各 Chapter を学んだ直後に取り組むと、より学習効果が期待できるでしょう。

## 解答例のソースコードのダウンロードの方法

Raspberry Pi(以下、ラズパイ)の Web Browser で、著者の GitHub の Web サイトを開きます。著者の GitHub の Web サイトの[View on GitHub]をクリックします(図 1-101)。

<https://tamamori2020.github.io/raspi-gh1/>

○図 1-101: 著者の GitHub のホームページ



図 1-102 に示す Web サイトの①[Clone or download] (①)⇒[Download ZIP] (②)をクリックします。ダウンロードが完了すると、ラズパイのダウンロードフォルダに「raspi-gh1-master.zip」ファイルが保存されます。

○図 1-102: 著者の GitHub のダウンロードのページ

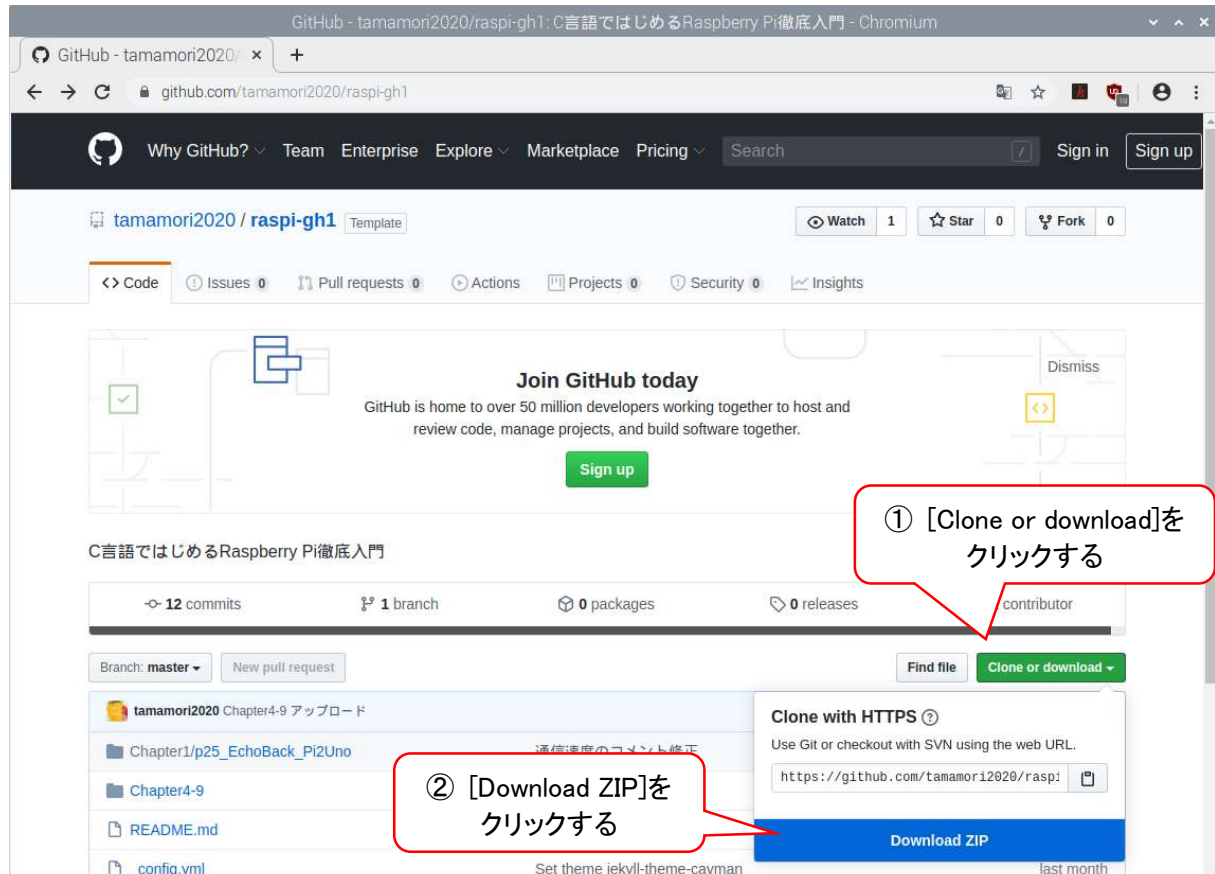


図 1-103 のようにダウンロードフォルダを開いて、raspi-gh1-master.zip (①)を右クリック⇒[ここでファイルを展開] (②)をクリックします。展開が終了すると、raspi-gh1-master フォルダが作成されるので、ダブルクリックするといくつかフォルダを表示されます。Chapter4-9 フォルダ内のすべての内容を/home/pi/MyApp にコピーします。

○図 1-103: ZIP ファイルの展開の方法



Chapter4-9 フォルダには、[未完成ライブラリ]と[完成ライブラリ]のフォルダがあります。7 章以降の章末問題では、「徹底入門」で作成したライブラリ libMyPi.a に関数を追加していきます。

章末問題でライブラリの登録の課題を行う場合は、「徹底入門」で読者が作成したファイル MyPi.c と MyPi.h を使用してください。ファイルが無い場合は、[未完成ライブラリ]のファイルをフォルダ/home/pi/MyApp にコピーして使用してください。

章末問題のソースコードを直ぐに動作確認したい場合は、[完成ライブラリ]のファイルを/home/pi/MyApp にコピーして使用してください。なお、[完成ライブラリ]にあるライブラリ libMyPi.a は、Pi 3/3B+および Pi 4B で動作を確認していますが、将来的にラズパイや WiringPi のバージョンアップ等により動作しない場合があります。本章末問題は、「徹底入門」の前付に準じます。

#### 必ずお読みください

本書では、次のようなシステム構成を前提に解説しています。

- 本体 : Raspberry Pi 3/3B+および4B
- OS : Raspbian (Buster) 2019-09-30 版 (NOOBS Ver.3.2.1)
- ディスプレイ : HDMI 仕様
- キーボード・マウス : USB 仕様
- SD カード : microSD SDHC 16GB スピードクラス10
- ネットワーク環境 : Raspberry Pi に内蔵した無線 LAN (Wi-Fi) でインターネットに接続

本書は、Windows PC やインターネットの一般的な操作を一通りできる方を対象にしているため、基本的な操作方法などは解説していません。C 言語や電子回路の基本的な知識についても、習得されていることを前提にしています。電子部品や工具などを使用する際は、ケガなどをされないようにご注意ください。また、電子部品は静電気に弱いので取り扱いにもご注意ください。

#### 本書のサポートページについて

本書で利用するソースコードなどが記述されたサンプルファイルと追加情報や、正誤表、本書発行後にバージョンアップなどにより設定手順などに変更が生じた場合の補足情報は、本書サポートページに掲載しています。

- 本書サポートページ  
<https://gihyo.jp/book/2020/978-4-297-11299-8>

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いたご利用は、必ずお客様自身の責任と判断によって行ってください。これらの情報によるご利用の結果について、技術評論社および著者はいかなる責任も負いません。

本書記載の情報は、2020年1月現在のものを掲載していますので、ご利用時には変更されている場合もあります。また、ソフトウェアに関する記述は、とくに断わりのないかぎり、2020年1月時点での最新バージョンをもとにしています。それぞれバージョンアップされる場合があり、本書での説明とは内容などが異なってしまうこともあり得ます。本書ご購入の前に、必ずバージョン番号などをご確認ください。

本書の配線図は「Fritzing」(<https://fritzing.org/>)を使って描いたものを使用しています。

以上の注意事項をご承諾いただいたうえで、本書をご利用願います。これらの注意事項をお読みいただくに、お問い合わせいただいても、技術評論社および著者は対処しかねます。あらかじめ、ご承知おきください。

本書で記載されている会社名、団体名、製品名、サービス名などは、一般に関係各社・団体の商標または登録商標です。なお、本書では™、®などは明記していません。また、本書では、通称やそのほかの名称で記述する場合がありますが、あらかじめご了承ください。

著者のサポートページにも有益な情報をアップしていますので、是非ご参照ください。

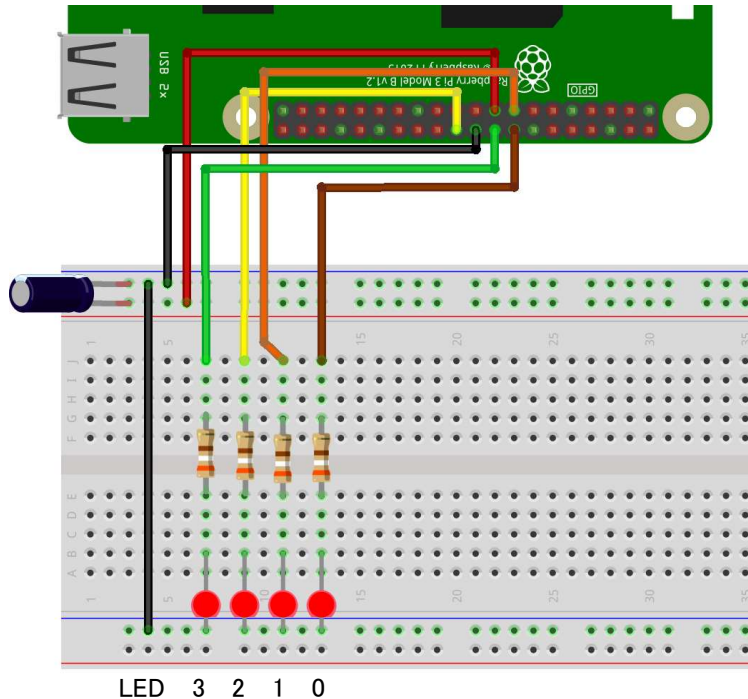
<http://raspi-gh1.blogspot.com/>



# Chapter 4 章末問題

「徹底入門」の図 4-21 (P. 94) の配線図を使用します(図 4-101)。

○図 4-101: 章末問題の配線図



## 【問題 4-1】 LED を2つの点灯パターンで交互に点灯させるプログラム

表 4-101 に示すように、LED0～LED3 を 2 つの点灯パターンで 0.5 秒間隔で交互の点灯させ、10 回繰り返します。繰り返しが終了したら、LED を消灯します。ファイル名は「List4\_Ex01.c」とします。

○表 4-101: 点灯パターン

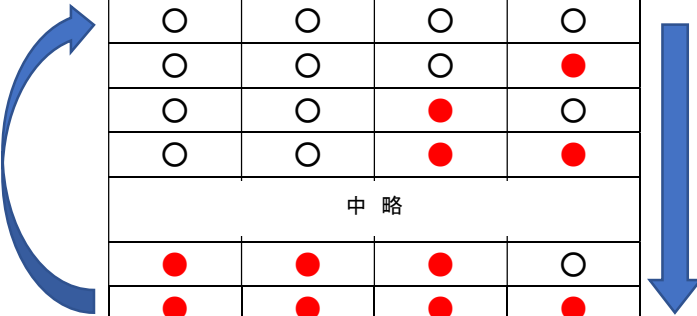
パターン	LED3	LED2	LED1	LED0
1	○	●	○	●
2	●	○	●	○
10 回、繰り返し終了				
3	○	○	○	○

## 【問題 4-2】 2 進数でカウントアップのプログラム

表 4-102 に示すように、2 進数でカウントアップする点灯パターン 0000、0001、0010～1110、1111 を 0.5 秒間隔で LED に繰り返し点灯させるプログラムを作成します。ファイル名を「List4\_Ex02.c」とします。

○表 4-102: 点灯パターン

LED3	LED2	LED1	LED0
○	○	○	○
○	○	○	●
○	○	●	○
○	○	●	●
中 略			
●	●	●	○
●	●	●	●

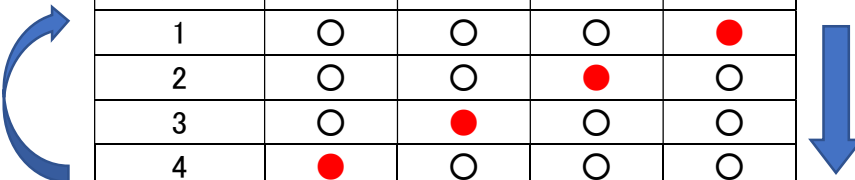


## 【問題 4-3】 左シフトを繰り返す LED 点灯のプログラム

表 4-103 に示すように、左シフトする点灯パターン 1～4 を 0.5 秒間隔で LED に繰り返し点灯させるプログラムを作成します。ファイル名は「List4\_Ex03.c」とします。

○表 4-103: 点灯パターン

パターン	LED3	LED2	LED1	LED0
1	○	○	○	●
2	○	○	●	○
3	○	●	○	○
4	●	○	○	○

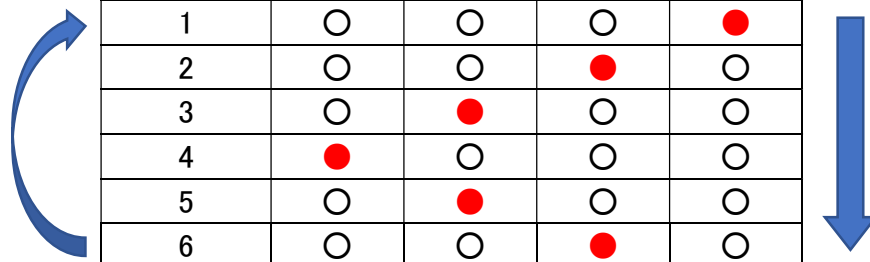




## 【問題 4-4】 左シフトと右シフトを繰り返す LED 点灯のプログラム

表 4-104 に示すように、左シフトと右シフトのパターン 1 ～ 6 を 0.5 秒間隔で LED に繰り返し点灯させるプログラムを作成します。ファイル名を「List4\_Ex04.c」とします。

○表 4-104: 点灯パターン



パターン	LED3	LED2	LED1	LED0
1	○	○	○	●
2	○	○	●	○
3	○	●	○	○
4	●	○	○	○
5	○	●	○	○
6	○	○	●	○

## 【問題 4-5】 キーボードから入力された数値を 2 進数で LED に表示するプログラム

ターミナルに「0から15までの値を入力してください>>>」と表示させます。キーボードから0～15の数値を入力して「Enter」が押されたら、2進数でLEDに表示させます。また、0～15以外が入力されたら「入力の値が範囲外です」と表示して、すべてのLEDを消灯します。ファイル名を「List4\_Ex05.c」とします。

### ● 実行例

0から15までの値を入力してください>>>5

LED の表示

○●○●

0から15までの値を入力してください>>>20

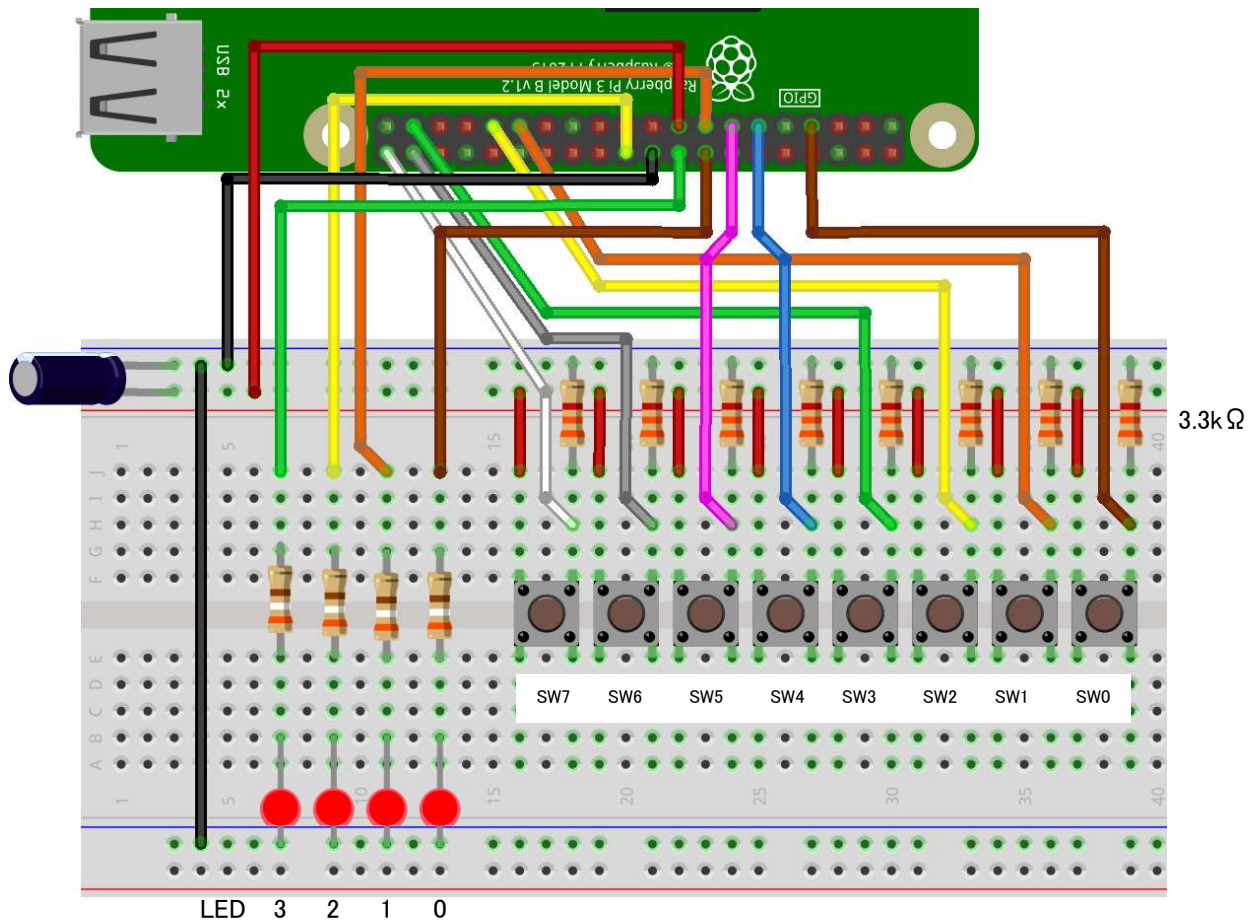
○○○○

入力の値が範囲外です

# Chapter 5 章末問題

「徹底入門」の図 5-19 (P. 116) の配線図に 6 個のタクタイルスイッチ回路を追加します(図 5-101)。各スイッチと GPIO の対応を表 5-101 に示します。Raspberry Pi 4B では、WiringPi のバージョン 2.52 において pullUpDnControl 関数が無効です。そのため、抵抗  $3.3k\Omega$  でプルダウンします。Pi 3/3B+では pullUpDnControl 関数が有効なので抵抗は不要です。

○図 5-101: 章末問題の配線図



○表 5-101: タクタイルスイッチと GPIO の対応表

タクタイルスイッチ番号	信号名
SW0	GPIO4
SW1	GPIO5
SW2	GPIO6
SW3	GPIO26
SW4	GPIO17
SW5	GPIO27
SW6	GPIO20
SW7	GPIO21

## 【問題 5-1】 タクタイルスイッチで LED を点灯させるプログラム(その1)

---

SW0～SW3 のタクタイルスイッチを押している間、表 5-102 に対応する LED が点灯するプログラムを作成します。タクタイルスイッチを離すと LED は消灯します。ファイル名は「List5\_Ex01.c」とします。

○表 5-102:タクタイルスイッチと LED の対応表

タクタイルスイッチ番号	LED 番号
SW0	LED0
SW1	LED1
SW2	LED2
SW3	LED3

## 【問題 5-2】 タクタイルスイッチで LED を点灯させるプログラム(その 2)

---

SW0～SW7 のタクタイルスイッチを押している間、表 5-103 に対応する LED が点灯するプログラムを作成します。タクタイルスイッチを離すと LED は消灯します。ファイル名は「List5\_Ex02.c」とします。

○表 5-103:タクタイルスイッチと LED の対応表

タクタイルスイッチ番号		LED 番号
SW0	SW4	LED0
SW1	SW5	LED1
SW2	SW6	LED2
SW3	SW7	LED3

### 【問題 5-3】 タクタイルスイッチで LED を点滅させるプログラム(その 1)

---

SW0 を押すと LED0 が 1 秒間隔で点滅し、SW1 を押すと LED0 の点滅が一時停止し、SW0 を押すと再スタートするプログラムを、割込み処理を使わないで作成します。ファイル名は「List5\_Ex03.c」とします。

なお、解答例のソースコードでは、LED0 の 1 点滅毎に SW1 の状態をチェックしているため、SW1 を約 2 秒間長押ししないと一時停止しません。

#### ● ヒント

SW1 が押されたら LED0 の点滅のループを break 文で抜けるのも一案です。

### 【問題 5-4】 タクタイルスイッチで LED を点滅させるプログラム(その 2)

---

問題 5-3 のプログラムでは、SW1 を長押ししないと一時停止しない欠点がありました。割込み処理を使用して、SW1 を長押ししなくても一時停止するプログラムに改良してください。ファイル名は「List5\_Ex04.c」とします。

## 【問題 5-5】 複数の割込み処理をするプログラム

表 5-104 に示すように、SW0～SW7 のタクトイルスイッチと割込み処理を利用して、LED0～LED2 の点灯パターンを設定します。表 5-104 の入力の 1 はスイッチを押した状態を表し、出力の 1 は LED の点灯を表しています。スイッチを離しても、LED は点灯状態を保持します。

なお、表 5-104 は 8 入力 3 出力のエンコーダの真理値表になります。ファイル名は「List5\_Ex05.c」とします。

○表 5-104: 8 個のタクトイルスイッチと 3 個の LED の対応表

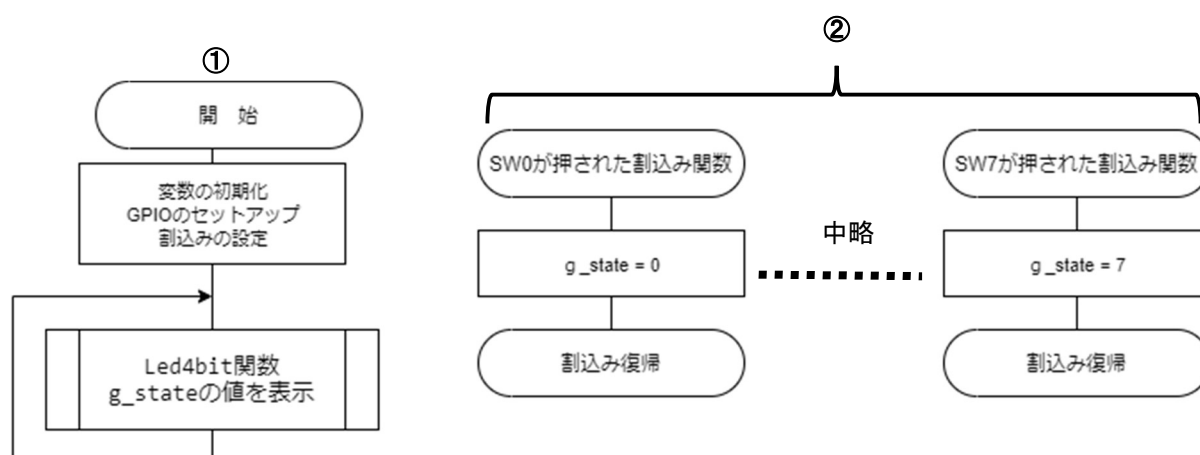
入力								出力		
SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0	LED2	LED1	LED0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

### ● フローチャート

フローチャートを図 5-102 に示します。main 関数、SW0 割込み関数から SW7 割込み関数のフローチャートは独立していますが、グローバル変数 `g_state` を共有しています。

- ① main 関数では、`g_state` の値を LED に繰り返し表示します。`g_state` の値を変更するのは SW0 から SW7 の割込み関数の処理になります。
- ② 各 SW が押されたら、対応する割込み関数は立ち上がりエッジで動作します。各割込み関数では、表 5-104 に示した出力の値を `g_state` に代入します。

○図 5-102: 複数の割込み処理のフローチャート



## 【参考 5-1】 割込み処理でオルタネート動作させるプログラム

「徹底入門」のリスト 6-4 でバウンス対策したプログラムを作成しましたが、割込み処理と wiringPi の millis 関数を利用して、対策プログラムを作成することができます。

millis 関数の戻り値(32bit)は wiringPiSetup 関数を実行してからの経過時間(ミリ秒単位)です。値は符号なしの 4 バイトです。経過時間は約 49 日後にオーバーフローしてゼロに戻り、カウントを再開します。

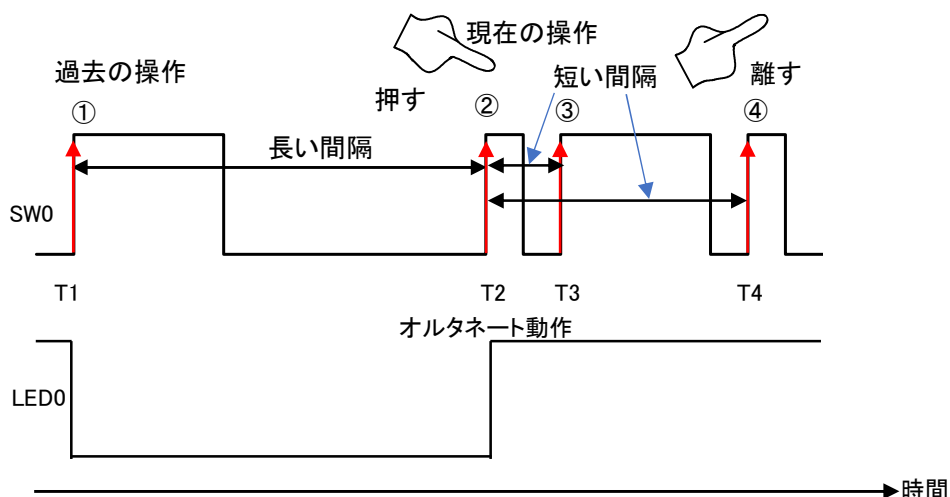
```
unsigned int millis (void) ;
```

図 5-103 を利用して、割込み処理と millis 関数を利用したバウンス対策の考え方を説明します。割込み処理の起動に「立ち上がりエッジ」を設定します。

過去(前回)に SW0 を操作したときの「①立ち上がりエッジ」の時間 T1 を millis 関数で取得しておきます。現在、操作した「②立ち上がりエッジ」の時間 T2 を同様に millis 関数で取得し、時間 T1 との時間間隔を計算します。求まった時間間隔がバウンス現象より長い場合、「スイッチが押された」と判断してオルタネート動作をします。

今度は T2 を記憶します。立ち上がりエッジ③や④の場合のように短い間隔を検出しときは、バウンス現象と判断してオルタネート動作をしません。また、時間 T3 と T4 を記憶しません。なお、本課題のソースコード「List5\_Ref01.c」の実験値から 300 ミリ秒以上の時間間隔のとき、オルタネート動作するように設定してます。

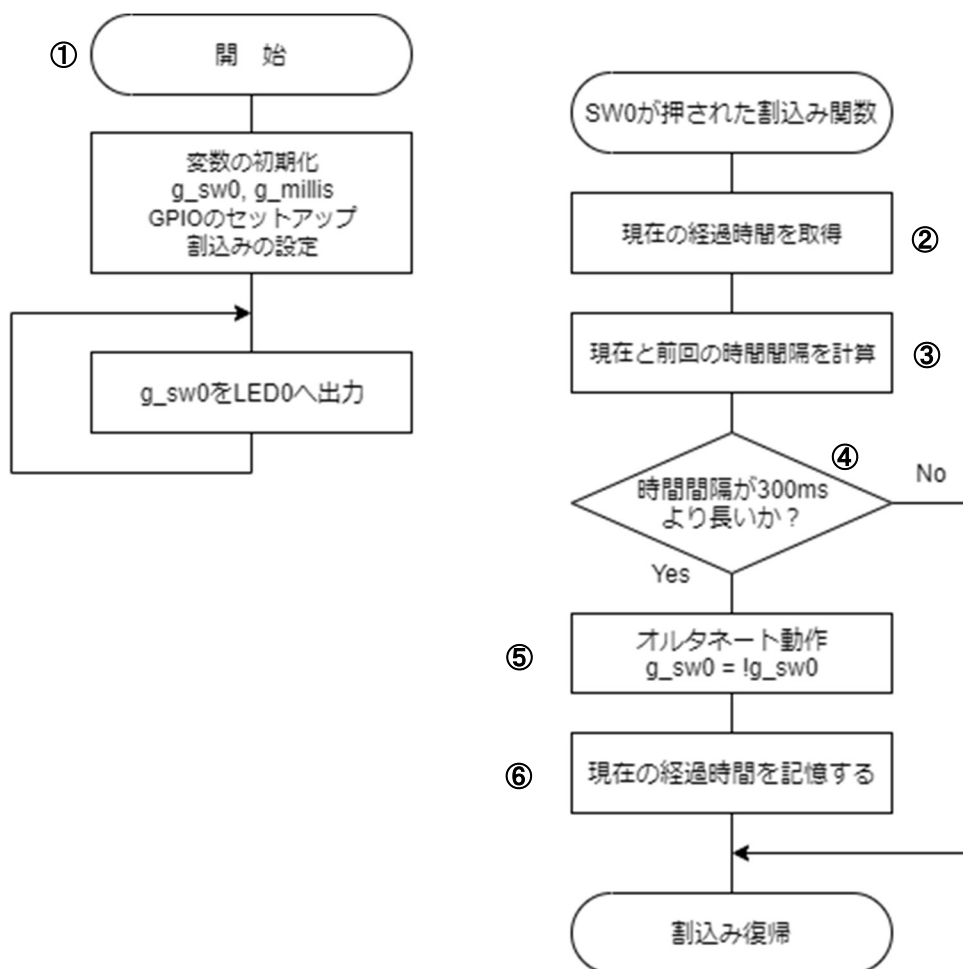
○図 5-103: 割込みを利用したバウンスの対策



## ● フローチャート

割込み処理を使用したバウンス対策のフローチャートを図 5-104 に示します。

図 5-104: 割込みを利用したバウンス対策のフローチャート



- ① main 関数では、グローバル変数(g\_sw0、g\_millis)、GPIO のセットアップ、割込みの設定などの初期化を行います。main 関数ではオルタネート動作を記憶するグローバル変数 g\_sw0 の値を LED0 へ出力します。変数 g\_millis は、過去のスイッチを操作したときの時間を記憶します。
- ② SW0 が押されたら、現在の経過時間を取得します。
- ③ 現在と過去の経過時間から時間間隔を求めます。
- ④ 時間間隔が 300 ミリ秒以下の場合は、バウンスによるパルス信号と判断して、流れ線「No」により割込み処理を終了します。
- ⑤ 時間間隔が 300 ミリ秒より長い場合は、オルタネート動作とします。
- ⑥ ②で取得した経過時間を変数 g\_millis に記憶します。



## ● ソースコード

ソースコードはリスト 5-101 のとおりです。ファイル名を「List5\_Ref01.c」とします。

○リスト 5-101: List5\_Ref01.c

```
#include <stdio.h>           //入出力
#include <stdlib.h>          //一般ユーティリティ
#include <string.h>          //文字列操作
#include <stdbool.h>         //論理型および論理値
#include <limits.h>          //整数型の大きさ
#include <wiringPi.h>        //wiringPi

#define LED0      23        //GPIO23 を LED0 と定義
#define SW0       4         //GPIO4 を SW0 と定義
#define INTERVAL  300      //バウンス対策時間 ms

/* グローバル変数 */
static volatile bool g_sw0;           //SW0 操作の変数
static volatile unsigned int g_millis; //経過時間 ms
/* プロトタイプ宣言 */
void IntSw0(void);                   //SW0 の割込み関数

int main(void){
    wiringPiSetupGpio();              //BCM の GPIO 番号を使用
    pinMode(LED0, OUTPUT);            //LED0 を出力に設定
    pinMode(SW0, INPUT);              //SW0 を入力に設定
    pullUpDnControl(SW0,PUD_DOWN);   //SW0 にプルダウン抵抗をつける

    wiringPiISR(SW0, INT_EDGE_RISING, (void*)IntSw0); //立ち上がりエッジ
    g_millis = millis();
    g_sw0 = false;

    while(1){
        digitalWrite(LED0,g_sw0);
    }
    return EXIT_SUCCESS;
}

void IntSw0(void){
    unsigned int diff,now;
    now = millis();
    diff = now - g_millis;
    if(diff > INTERVAL){
        g_sw0 = !g_sw0;
        g_millis = now;
    }
}
```

- ① グローバル変数 `g_sw0` はオルタネート動作を記憶し、`g_millis` は前回のスイッチを操作したときの時間を記憶します。`g_sw0` と `g_millis` は `main` 関数と `IntSw0` 関数で共有するため、グローバル変数としています。各変数を修飾する `static` 記憶クラス指定子は、関数内と関数外で働きが異なります。関数内ではプログラムが終了するまで消滅しません。関数外で定義したグローバル

変数の有効範囲を List5\_Ref01.c のファイル内だけに限定させます。本事例では、ファイルは一つですが、複数のソースファイルで構成される場合においては他のファイルからグローバル変数がアクセスされることを抑止します。

また、コンパイラが最適化のために無駄と解釈された変数は削除される可能性があります。そのため、削除されたくない変数には `volatile` 修飾子として宣言して、コンパイラに対して最適化の抑止を指示します。特に、割込み処理で使用する変数がループ文で変化しない場合、無駄と解釈される可能性があります。そのため `volatile` 修飾することにより、割込み処理など要因で変数の値が変化することを明示します。なお、コンパイラのオプションで最適化を無効にしている場合は必要ないかもしれませんが、ソースコードで明示することは可読性に有効です。

- ② SW0 の割込み関数のプロトタイプ宣言です。
- ③ SW0 を押したときに、立ち上がりエッジで割込みが発生するように設定します。
- ④ `millis` 関数により経過時間を取得して `g_millis` に代入します。`g_sw0` の初期値を `false` にします。ヘッダファイル `stdbool.h` では、`false` は 0、`true` は 1 にマクロ定義されています。オルタネート動作の論理反転に利用します。
- ⑤ `g_sw0` を永久ループで LED0 へ出力します。
- ⑥ 割込み関数が発生すると、現在の経過時間を `millis` 関数で取得して、変数 `now` に代入します。前回のスイッチ操作した経過時間である `g_millis` との時間間隔を計算します。
- ⑦ 時間間隔が `INTERVAL` (300 ミリ秒) より長い場合は、`g_sw0` を論理反転させてオルタネート動作の処理をします。また、`g_millis` の更新します。時間間隔が 300 ミリ秒以下のときはバウンスングによるパルス信号と解釈して割込み処理を終了します。

## ● 動作確認

List5\_Ref01.c をビルドして実行します。バウンスング対策したオルタネート動作を確認します。

List5-4\_Alt02.c と比較して、割込み処理を使用すると `main` 関数がシンプルになり、複数のタクトイルスイッチへの対応も容易です。なお、タクトイルスイッチを押す(ワンプッシュ)する時間を 300 ミリ秒としていますので、スイッチを長押する操作では離れたときのバウンスングによって誤動作することがあります。また、マルチタスク OS 環境では CPU の負荷状態にも影響を受ける場合があります。

## 【参考 5-2】ビットフィールドで LED と SW の変数を表現するプログラム

例えば、SW0 の情報は 1 ビットですが、その情報を保存するために int 型を用いるのは、メモリ容量が少ないマイコンではメモリの利用効率から好ましくはありません。そのため、一般にはビットフィールドを使用してメモリを節約します。Raspberry Pi のメモリ容量は十分ですが、ビットフィールドの使用例を紹介しましょう。

### ● ビットフィールドとは

ビットフィールドは、1 バイト(8 ビット)の変数の定義に使用します。8 ビット未満のビットも定義することができます。ビットフィールドは構造体(struct)のメンバーとして宣言します。

リスト 5-102 に SW0 から SW7 のビットフィールドの使用例を示します。構造体を利用してビットフィールドを B0 から B7 で定義します。LSB は SW0 です。さらに、共有体(union)で、8 ビットのフィールドを 1 バイトとして参照できるようにします。このビットフィールドに各 SW の値を格納するプログラムを別途作成します。

- ビット SW.BIT.B0 SW0 のビット参照。
- バイト SW.BYTE SW0 から SW7 のバイト参照。

○リスト 5-102:ビットフィールドの使用例

```
union {
    unsigned char BYTE;
    struct{
        unsigned char B0:1;    //SW0
        unsigned char B1:1;    //SW1
        unsigned char B2:1;    //SW2
        unsigned char B3:1;    //SW3
        unsigned char B4:1;    //SW4
        unsigned char B5:1;    //SW5
        unsigned char B6:1;    //SW6
        unsigned char B7:1;    //SW7
    }BIT;
}SW;
```

### ● ソースコード

【問題 5-2】の問題をビットフィールドで記述したソースコードをリスト 5-103 に示します。ファイル名を「List5\_Ref02.c」とします。

○リスト 5-103: List5\_Ref02.c

```
#include <stdio.h>           //入出力
#include <stdlib.h>          //一般ユーティリティ
#include <wiringPi.h>        //wiringPi

const int ledGpio[4] = {23,22,25,24};           //LED GPIO を配列で定義
const int swGpio[8] = {4,5,6,26,17,27,20,21};   //SW GPIO を配列で定義
```

```

/* LED のビットフィールド */
union {
    unsigned char BYTE;
    struct {
        unsigned char B0:1;    //LED0
        unsigned char B1:1;    //LED1
        unsigned char B2:1;    //LED2
        unsigned char B3:1;    //LED3
        unsigned char :4;      //空の 4bit
    }BIT;
}LED;

/* SW のビットフィールド */
union {
    unsigned char BYTE;
    struct{
        unsigned char B0:1;    //SW0
        unsigned char B1:1;    //SW1
        unsigned char B2:1;    //SW2
        unsigned char B3:1;    //SW3
        unsigned char B4:1;    //SW4
        unsigned char B5:1;    //SW5
        unsigned char B6:1;    //SW6
        unsigned char B7:1;    //SW7
    }BIT;
}SW;

/* プロトタイプ宣言 */
void Led4bit(int ledData);
void Sw8bit(void);

int main (void){
    int i;
    wiringPiSetupGpio();        //BCM の GPIO 番号を使用
    for(i=0;i<4;i++){          //LED0-LED3 を出力に設定
        pinMode(ledGpio[i], OUTPUT);}
    for(i=0;i<8;i++){          //SW0-SW7 を入力に設定
        pinMode(swGpio[i], INPUT);}
    for(i=0;i<8;i++){          //SW0-SW7 をプルダウン抵抗をつける
        pullUpDnControl(swGpio[i],PUD_DOWN);}

    while(1){
        Sw8bit();              //SW 入力
        LED.BIT.B0 = SW.BIT.B0 | SW.BIT.B4;
        LED.BIT.B1 = SW.BIT.B1 | SW.BIT.B5;
        LED.BIT.B2 = SW.BIT.B2 | SW.BIT.B6;
        LED.BIT.B3 = SW.BIT.B3 | SW.BIT.B7;
        Led4bit(LED.BYTE);      //LED 出力
    }
    return EXIT_SUCCESS;
}

void Led4bit(int ledData)
{
    int i;

```

```

        for (i=0; i<4; i++){
            //LSB の LED0 から順番に出力する
            digitalWrite(ledGpio[i], ledData&1); //LSB だけの値にして出力
            ledData = ledData >> 1; //1bit 右シフト
        }
    }

void Sw8bit(void){
    int i,k;
    SW.BYTE=0;
    for(i=7; i>=0; i--){
        k=digitalRead(swGpio[i]);
        SW.BYTE = SW.BYTE << 1;
        SW.BYTE = SW.BYTE + k;
    }
    // printf("%d¥n",SW.BYTE); //デバッグ用
}

```

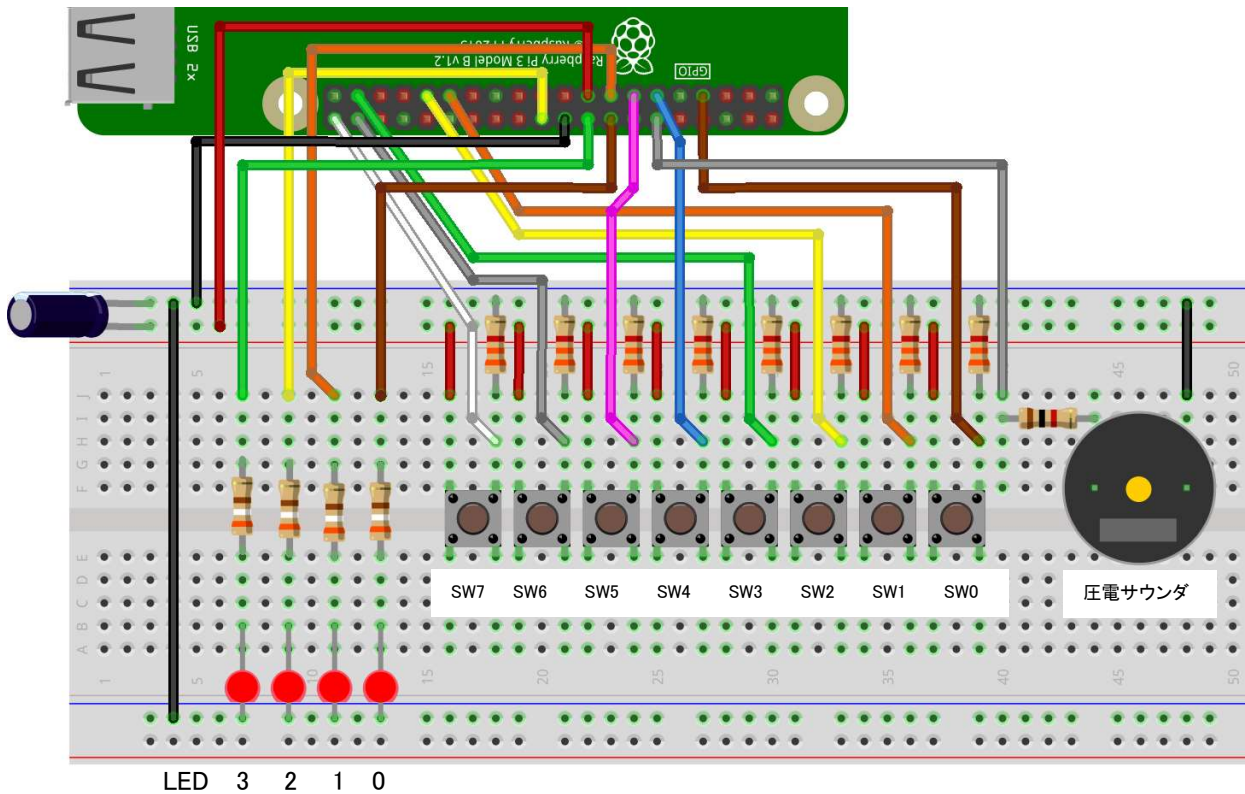
### ● 動作確認

List5\_Ref02.c をビルドして実行します。List5\_Ex02.c と同じ動作になることを確認します。なお、ビットフィールドのメモリ上の割付け順序はコンパイラに依存しますので、本ソースコードを他の処理系に移植する場合は注意してください。

# Chapter 6 章末問題

「徹底入門」の図 6-12 (P. 130) の配線図において、Pi 4B 用にプルダウン抵抗をタクタイルスイッチに実装しました(図 6-101)。

○図 6-101: 章末問題の配線図



## 【問題 6-1】ソフトウェア方式 PWM でデューティ比を変えるプログラム

ソフトウェア方式 PWM を利用して、LED0 (GPIO23) を表 6-101 に示すタクタイルスイッチに対応したデューティ比で点灯させるプログラムを作成します。タクタイルスイッチが押されていないときのデューティ比は 0%とします。ファイル名は「List6\_Ex01.c」とします。

○表 6-101: タクタイルスイッチを押したときのデューティ比

スイッチ番号	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
デューティ比	100%	87.5%	75%	62.5%	50%	37.5%	25%	12.5%

- 条件
- ・ PWM の周波数  $f_{\text{pwm}}$  100Hz

## 【問題 6-2】 ソフトウェア方式 PWM で約 440Hz の矩形波を発生させるプログラム

---

タクトイルスイッチ SW0 を押している時だけ、ソフトウェア方式 PWM で GPIO18 から約 440Hz の矩形波を出力して圧電サウンダを鳴らすプログラムを作成します。ファイル名は「List6\_Ex02.c」とします。

- 条件
  - ・ デューティ比      約 50%
- ヒント
  - ・ 440Hz の周期[ms]を求め、ソフトウェア方式の PWM 信号の最小変化幅(100  $\mu$ s)で割った値が `softPwmCreate` 関数の引数 `pwmRange` になります。

## 【問題 6-3】 ハードウェア方式 PWM で約 440Hz の矩形波を発生させるプログラム

---

タクトイルスイッチ SW0 を押している時だけ、ハードウェア方式の PWM で GPIO18 から約 440Hz の矩形波を出力して圧電サウンダを鳴らすプログラムを作成します。ファイル名は「List6\_Ex03.c」とします。

- 条件
  - ・ デューティ比                      約 50%
  - ・ PWM 信号の階調(range)      100
- ヒント
  - ・ Pi 3/3B+ と Pi 4B では内部クロックの周波数が異なりますので、適切な分周の値を設定します。詳しくは「徹底入門」の 6.5 ハードウェア方式の PWM(P.125-127)を参照してください。



## 【問題 6-4】 電子ピアノのプログラム

ハードウェア方式の PWM で GPIO18 から表 6-102 に示す音名の矩形波を出力して、電子ピアノとなるプログラムを作成します。PWM 信号の 1 周期の階調(range)を 100 としたとき、表 6-102 の分周値(divisor)を pwmSetClock 関数の引数とすると、音名の周波数が出力されます。なお、分周値は  $\text{Pi } 3/3B+$  と  $\text{Pi } 4$  で異なります。各タクトイルスイッチと音名の対応は図 6-102 のとおりです。ファイル名は「List6\_Ex04.c」とします。

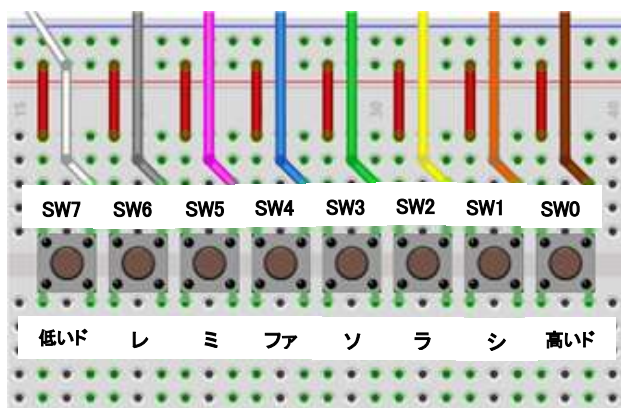
### ● 条件

- ・ デューティ比 約 50%
- ・ PWM 信号の階調(range) 100
- ・ PWM 信号の波形のモード PWM\_MODE\_MS
- ・ 1 本指による操作とし、和音には対応しなくてよい。

○表 6-102:タクトイルスイッチと音名

スイッチ	階名	周波数 (Hz)	周期 (ms)	分周値(divisor)	
				Pi 3/3B+	Pi 4
SW7	(低い)ド	261.63	3.82	734	2064
SW6	レ	293.66	3.41	654	1839
SW5	ミ	329.63	3.03	582	1638
SW4	ファ	349.23	2.86	550	1546
SW3	ソ	392.0	2.55	490	1378
SW2	ラ	440.0	2.28	436	1227
SW1	シ	493.83	2.02	389	1093
SW0	(高い)ド	523.25	1.91	367	1032

○図 6-102:タイルスイッチと音名の対応



## 【問題 6-5】 電子オルゴールのプログラム

SW0 を押したら、ハードウェア方式の PWM で GPI018 から電子オルゴールが動作するプログラムを作成します。SW1 を押したら、電子オルゴールを停止します。また、SW0 を押したら最初から再生します。ファイル名は「List6\_Ex05.c」とします。

- 条件
  - ・ デューティ比 約 50%
  - ・ PWM 信号の階調(range) 100
  - ・ PWM 信号の波形のモード PWM\_MODE\_MS

- ヒント

音名または休符を発生する `g` 関数を作成して、利用します。`g` 関数への引数(オクターブ、階名、時間)を配列にして楽譜を作成して、1 音符毎に `g` 関数で音名を鳴らして、曲を再生します。

```
void g(int oct, int solfa, int length);
```

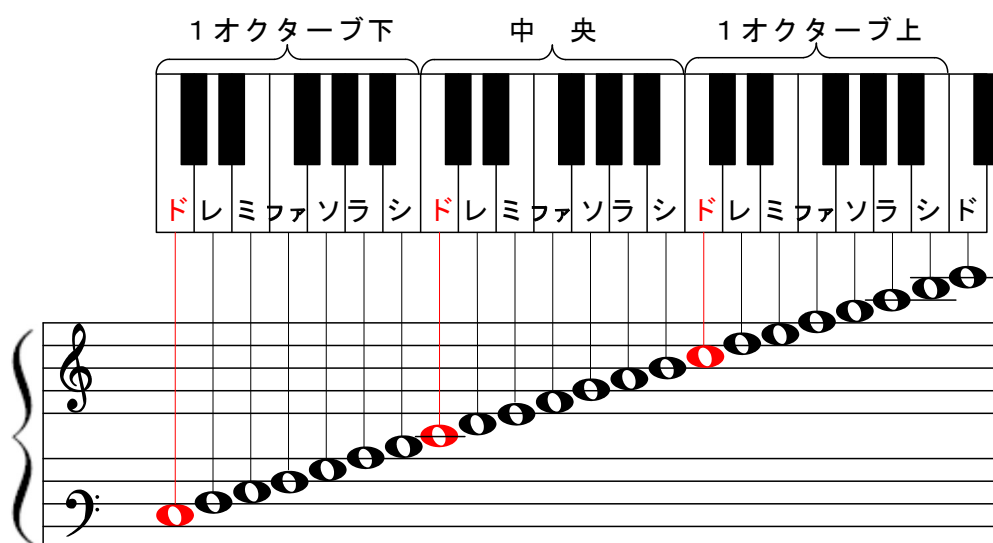
`g` 関数には `oct`、`solfa`、`length` の 3 つの引数があります。これらの引数の役割について説明します。

- ・ 引数 `oct` はオクターブのことです。3 オクターブから 1 つを選択します(表 6-103)。引数 `oct` と引数 `solfa`(階名)により、音名(鍵盤)が決まります(図 6-103)。ここでの階名とは七音(ドレミファソラシ)に呼び分けた名称のことで、音名とは個々の音の絶対的な高さを表す名称です。

○表 6-103:オクターブと引数 `oct`

オクターブ	1 オクターブ下	中央	1 オクターブ上
マクロ名	OL	OC	OH
引数 <code>oct</code>	3	4	5

○図 6-103:オクターブと音名(鍵盤)の位置



- 引数 solfa は階名のことです。休符は階名ではありませんが、0 に割り当てています(表 6-104)。

○表 6-104: 階名と引数 solfa

階名	休符	ド	ド#	レ	レ#	ミ	ファ	ファ#	ソ	ソ#	ラ	ラ#	シ
マクロ名	REST	DO	DOS	RE	RES	MI	FA	FAS	SO	SOS	RA	RAS	SI
引数	0	1	2	3	4	5	6	7	8	9	10	11	12

- 引数 length は音の時間の長さ(ミリ秒)です。4 分音符(quarter note)の時間の長さは、1 分を楽譜のテンポ(BPM:Beats Per Minute)で割った値になります。例えば、テンポが 120 のとき、1 分 ÷ 120 より 4 分音符の時間の長さは 0.5 秒になります。

### ● 音名の周波数の計算方法

引数オクターブと引数階名から、式 6-101 により音名の周波数を計算し、ラズパイの内部クロックの分周値 divisor を求めます。

式 6-101 の  $f$  は周波数、 $d$  は整数で、各音名の周波数を表 6-105 に示します。式 6-101 の  $d$  に 0 を代入すると、周波数  $f$  は、440Hz となりラの音になります。440Hz(ラ)は調律の際の音高(ピッチ)の標準になっています。1 オクターブは 12 音階になっていますので、 $(^{12}\sqrt{2})^d$  を 12 回乗算しますと 1 オクターブ上のラになり周波数  $f$  は 880Hz となります。

$$f = 440 \times (^{12}\sqrt{2})^d \quad [\text{式 6-101}]$$

○表 6-105: 中央のオクターブの音名の周波数

音名	ド	ド#	レ	レ#	ミ	ファ	ファ#	ソ	ソ#	ラ	ラ#	シ
d	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2
f Hz	261.6	277.2	293.7	311.1	329.6	349.2	370.0	392.0	415.3	440.0	466.2	493.9

g 関数のソースコードはリスト 6-101 のとおりです。

```

○リスト 6-101: void g(int oct, int solfa, int length)
#define PI4_CLK    54000000    //Pi4B の内部クロック Hz
#define PI3_CLK    19200000    //Pi3B/3B+の内部クロック Hz
#define RANGE      100        //PWM range

#define OL        3            //1 オクターブ下
#define OC        4            //中央のオクターブ
#define OH        5            //1 オクターブ上

#define REST      0            //休符
#define DO        1            //ド
#define DOS       2            //ド#
#define RE        3            //レ
#define RES       4            //レ#
#define MI        5            //ミ
#define FA        6            //ファ
#define FAS       7            //ファ#
#define SO        8            //ソ
#define SOS       9            //ソ#
#define RA        10           //ラ
#define RAS       11           //ラ#
#define SI        12           //シ

void g(int oct, int solfa, int length){
int d = 0;        //整数
int f = 0;        //周波数
int divisor;      //分周値 pwmSetClock 関数の引数
int value;        //Ton 時間 pwmWrite 関数の引数
    if(solfa !=0){
        d = solfa - 10;
        if (oct == OL){
            d -= 12;
        }
        if(oct == OH){
            d+=12;
        }
        f = 440 * pow(2,(double)d/12)+0.5;    //べき乗計算の pow 関数
//      どちらかを選択
        divisor = PI4_CLK / (f*RANGE);        //Pi 4B のとき
//      divisor = PI3_CLK / (f*RANGE);        //Pi 3/3B+のとき
        pwmSetClock(divisor); //内部クロックの分周の設定
        value = RANGE/2;    //デューティ比 50%
    }else{
        value = 0;
    }
    pwmWrite(BUZZER, value);
    delay(length);
}

```

- ① 内部クロック、オクターブ、階名のマクロ定義です。
- ② 変数 d(整数)と変数 f(周波数)は、音名の周波数の計算(式 6-101)に使用します。
- ③ 変数 divisor と変数 value は PWM 信号の発生に使用します。
- ④ solfa が休符以外のときは、音名の周波数を計算します。
- ⑤ 表 6-104 でマクロ定義した引数 solfa の値から、式 6-101 の d(表 6-105)を求めます。
- ⑥ g 関数の引数 oct が OL のときは1オクターブ下のため d の値から 12(1オクターブの音階)を減算し、oct が OH のときは1オクターブ上のため d の値に 12 を加算します。OL と OH の値は、表 6-103 で定義してあります。
- ⑦ べき乗の pow 関数を使用して、式 6-101 を構成します。小数点第一位を四捨五入のため、「+0.5」を式に入れています。pow 関数を使用するため、数学のヘッダーファイル math.h をインクルードします。さらに、gcc コマンドに数学オプションの「-lm」を追加します。geany のビルドコマンドの設定を次のようにします。

```
gcc -Wall -o "%e" "%f" -lwiringPi -lpthread -g -O0 -lm
```

- ⑧ 音名の周波数から分周値(divisor)を求めて内部クロックを分周し、PWM 信号をデューティ比 50%に設定します。divisor の値は Raspberry Pi 3B/3B+と Raspberry Pi 4B では、異なりますので注意してください。value は PWM 信号のパルスオンの時間 Ton のことです。
- ⑨ 休符の場合は、value を 0 にして PWM 信号の出力信号を 0V にします。
- ⑩ length の時間(ミリ秒)の長さで、1音または休符を出力します。

## ● 楽譜の作成と再生方法

例えば、中央のオクターブの「低いド」から1オクターブ上の「高いド」までを鳴らして、休符する楽譜を考えます。楽譜は g 関数の引数の順で配列[オクターブ、階名、時間の長さ]を作成するのが簡単です(リスト 6-102)。楽譜の終わりに休符を入れて PWM 信号の出力を止めます。変数 qN は 4 分音符(quarter note)のことです。

楽譜を再生するには、for 文などで配列の先頭から順番に配列の要素を g 関数の引数に代入して、1音毎に鳴らします。for 文の中で SW1 を検出するようにすれば、再生を停止させることが可能です。なお、参考例として「さくらさくら」(図 6-104)の楽譜を載せますが、お気に入りの曲で電子オルゴールを作成してみましょう。

### ○リスト 6-102: 簡単な楽譜の例

```
int music[9][3] = {
    {OC, DO, qN},           //ド
    {OC, RE, qN},           //レ
    {OC, MI, qN},           //ミ
    {OC, FA, qN},           //ファ
    {OC, SO, qN},           //ソ
    {OC, RA, qN},           //ラ
    {OC, SI, qN},           //シ
    {OH, DO, qN},           //ド(1オクターブ上)
    {OC, REST, qN}};
```

図 6-104 : さくらさくらの楽譜

さくらさくら

72 BPM

さくら さくら のやまも さとーも みわたす かぎーり かすみか

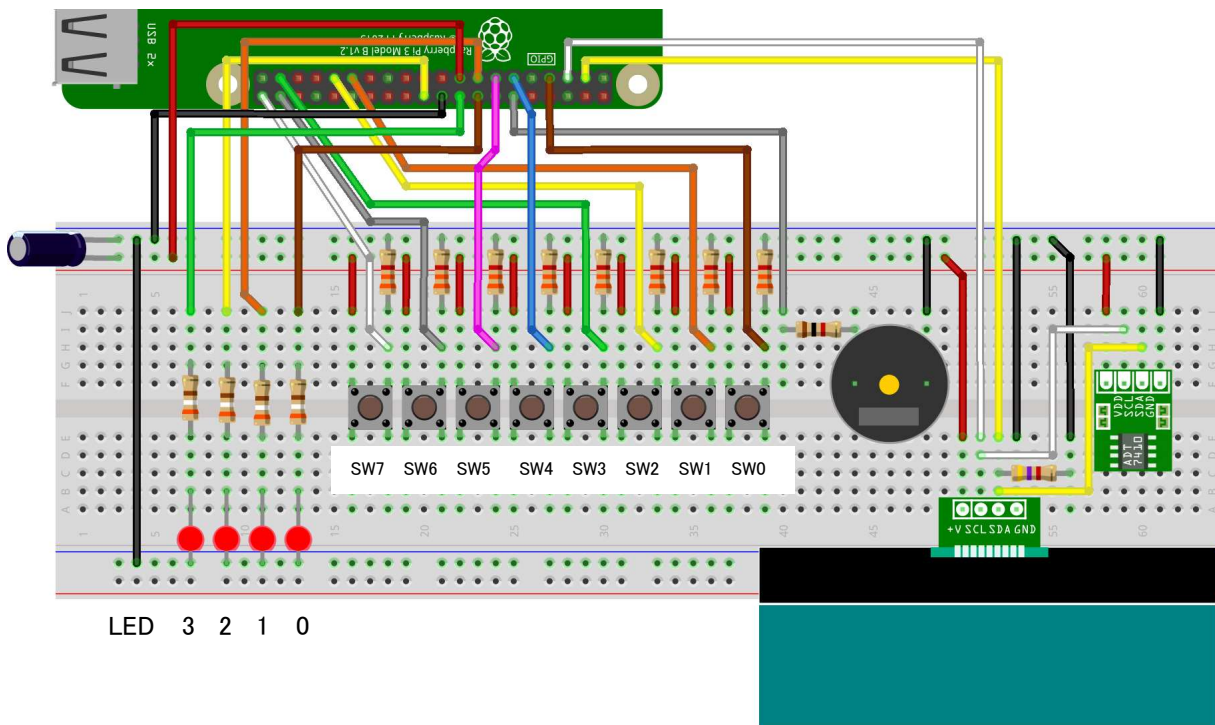
8

くもーか あさひに に おーう さくら さくら はなざーか り

# Chapter 7 章末問題

「徹底入門」の図 7-23 (P. 161) の配線図において、Pi 4B 用にプルダウン抵抗をタクトイルスイッチに実装しました(図 7-101)。

○図 7-101: 章末問題の配線図



## 【問題 7-1】 温度測定部の関数をライブラリに登録する

「徹底入門」の List7-3\_Temp01.c (P. 163) の温度測定のスーソースコードを Adt7410Read 関数として作成して、ライブラリ (libMyPi.a) に追加します。追加の方法は、ライブラリファイルの作成方法に準じますので、「徹底入門」の Chapter 7 (P. 155) を参照してください。下記に Adt7410Read 関数の仕様を示します。

登録したライブラリを使用して List7-3\_Temp01 (P. 164) と同じように温度センサの測定値を 0.5 秒毎に小数点第一位までをターミナルに表示するプログラムを作成します。ファイル名は「List7\_Ex01.c」とします。なお、技術評論社のサポートページからダウンロードしたサンプルソースコードには、既に Adt7410Read 関数は登録されています。

- 仕様
- ・ 関数名 double Adt7410Read(int fd)
- ・ 引数 fd wiringPiI2CSetup 関数で取得したファイルハンドル
- ・ 戻り値 double 型、温度データ 単位℃




## 【問題 7-2】 温度を LCD に表示するプログラム(その1)

---

問題 7-1 の List7\_Ex01.c を改造して、図 7-102 に示すように温度センサの測定値を 0.5 秒毎に小数点第一位までを LCD に表示するプログラムを作成します。ファイル名は「List7\_Ex02.c」とします。

- ヒント
  - sprintf 関数を使用すると、実数を文字列へ変換できます。

○図 7-102: LCD の表示例



25.1 C

## 【問題 7-3】 摂氏と華氏に切り替えて LCD に表示するプログラム

---

タクトイルスイッチ SW0 を押す毎に、温度の単位を摂氏 (°C) と華氏 (°F) に切り替えて、LCD に表示するプログラムを作成します(図 7-103)。ファイル名は「List7\_Ex03.c」とします。

- ヒント
  - 華氏 (°F) は、式 7-101 より摂氏 (°C) より求めることができます。


$$F = \frac{9}{5}C + 32 \quad [\text{式 7-101}]$$

○図 7-103: LCD の表示例



27.0 C

(a) 摂氏



80.6 F

(b) 華氏

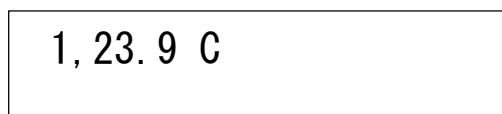
## 【問題 7-4】 測定した温度データをファイルに保存するプログラム

SW0 を押すと 0.5 秒毎に、1 から始まる測定の連番と小数点第一位までを測定した摂氏度をカンマ (,) で区切って LCD に表示します(図 7-104)。また、連番、摂氏度、改行をテキストモードでファイルに記録し、記録中は LED0 を点灯させます。SW1 を押すと測定を停止させ、ファイルをクローズし、LED0 を消灯させてプログラムを終了します。ファイル名は「List7\_Ex04.c」とします。

測定したデータを保存するファイルは CSV 形式とし、ファイル名を年月日時間秒、拡張子を「.csv」とします。例えば、プログラムを実行した時刻が 2020 年 2 月 6 日 10 時 48 分 28 秒であれば、20200206104828.csv となります。CSV は、Comma Separated Value の略で、「カンマ区切り」または「コンマ区切り」と呼ばれ、値をカンマ (,) で区切ったものです。改行はレコードの区切りを表します(表 7-101)。

表計算ソフト(LibreOffice Calc)で CSV ファイルを開くと、図 7-105 の「テキストのインポート」のダイアログが表示されるので、「区切りのオプション」の「コンマ」にチェックを入れて「OK」をクリックします。連番と摂氏度のセルを選択してグラフ(散布図)を作成し、タイトルや凡例を入力します(図 7-106)。グラフを含めて保存する場合は、ファイルを ODF 表計算ドキュメント(.ods)形式で保存します。

○図 7-104: LCD での温度の表示例

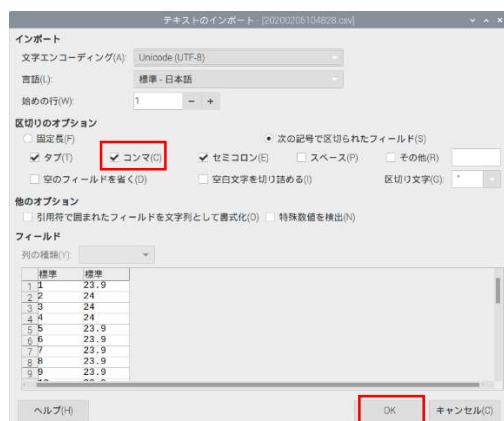


○表 7-101: CSV ファイルのレコードフォーマット

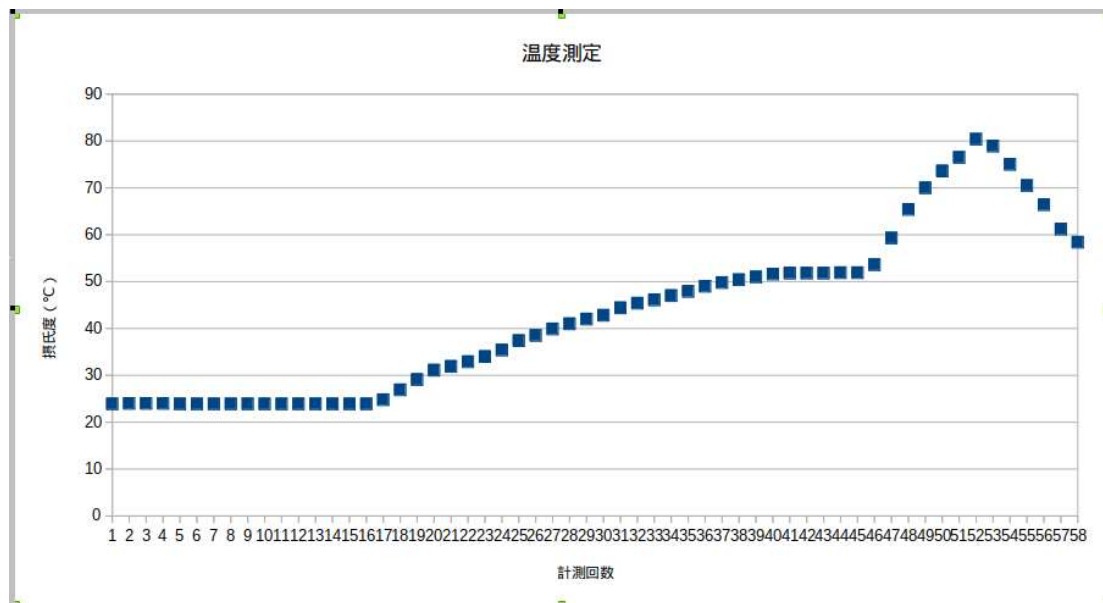
連番	カンマ	摂氏度	改行
1	,	23.9	↵
2	,	24.5	↵
3	,	25.3	↵

...(略)

○図 7-105: テキストのインポートのダイアログ



○図 7-106: 表計算ソフトによる温度グラフの例



● ヒント ファイル名を年月日時間秒とする方法

日付及び時間を取得するために使用したヘッダファイル `time.h` で定義されている関数や構造体を利用します。`strftime` 関数は日時情報を指定された書式で、文字配列に格納します。`strftime` 関数の変換指定子は、Y(年)、m(月)、d(日付)、H(24 時制)、M(分)、S(秒)を表します。

```
time_t now;
char fileName[32];
now = time(NULL);
strftime(fileName, sizeof(fileName), "%Y%m%d%H%M%S.csv", localtime(&now));
fp = fopen (fileName, "w");
```



LibreOffice のメニューが英語の表記の場合は、次の手順で日本語に変換できます。

```
$ sudo apt-get update
$ sudo apt-get upgrade -y
$ sudo apt-get install libreoffice-l10n-ja
$ sudo apt-get install libreoffice-help-ja
```

## 【参考 7-1】 起動したときに LCD に IP アドレスを表示させる

OS が起動した時に LCD に IP アドレスを図 7-107 のように表示させます。

○図 7-107:IP アドレスの表示例



192.168.11.2

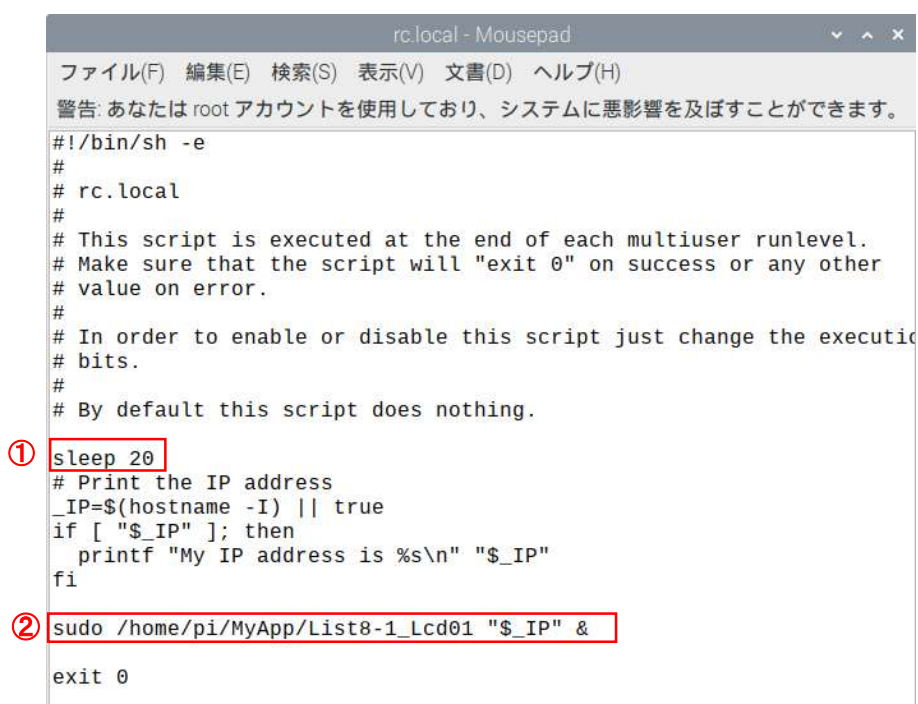
OS が起動する時に、「/etc/rc.local」に記載されたスクリプトやプログラムが自動的に管理者権限で実行されます。図 7-108 に示すように「rc.local」では IP アドレスを変数「\$\_IP」に代入するスクリプトが既に記述されているので、変数「\$\_IP」を List8-1\_Lcd01 (LCD に文字や数字を表示する) への引数とします。「rc.local」の編集には管理者権限が必要となりますので、sudo コマンドを使用してエディタ mousepad で編集します。

```
$ sudo mousepad /etc/rc.local
```

「rc.local」の 2 箇所(①と②)に追記します(図 7-108)。rc.local ファイルを保存して OS を再起動させると、IP アドレスが LCD に表示されます。

- ① sleep 20 を実行して、システムが安定するまで 20 秒間、時間待ちをします。
- ② List8-1\_Lcd01 を管理者権限で実行し、引数&\_IP を渡して LCD に IP アドレスを表示します。  
&は、プログラムをバックグラウンドで実行させる指示です(「徹底入門」(P. 288))。

○図 7-108:rc.local ファイルの編集箇所





### <i2cset コマンド>

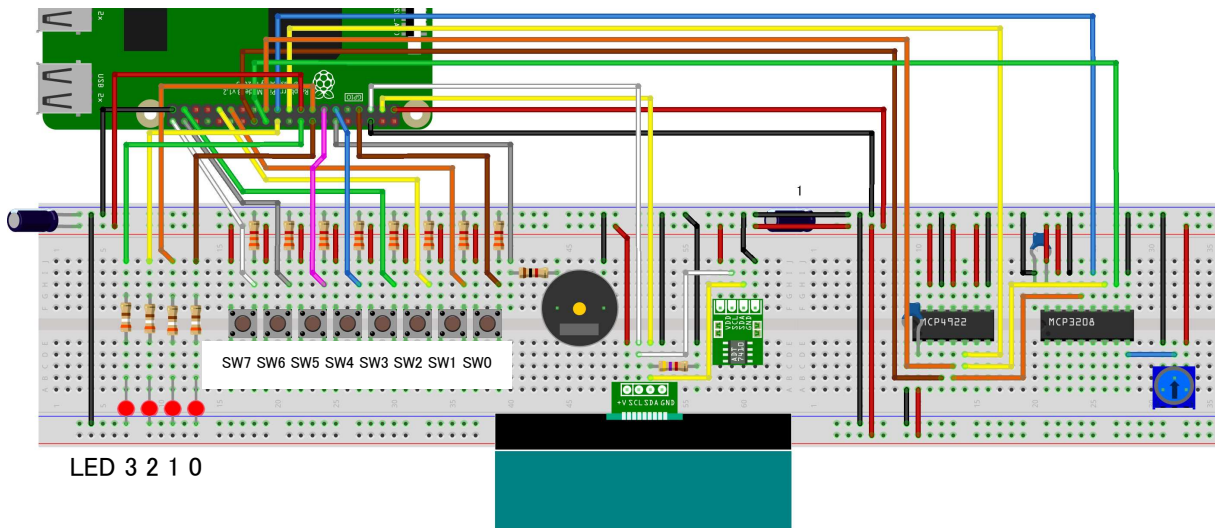
ターミナルから i2cset コマンドで I<sup>2</sup>C バスのデバイスヘデータを転送することができます。デバックに便利です。LCD AQM1602 の初期設定と「ABC」を表示する例を示します。

```
$ i2cset -y 1 0x3e 0x00 0x38 // Function set
$ i2cset -y 1 0x3e 0x00 0x39 // Function set
$ i2cset -y 1 0x3e 0x00 0x14 // Internal OSC frequency
$ i2cset -y 1 0x3e 0x00 0x73 // Contrast set
$ i2cset -y 1 0x3e 0x00 0x56 // Power/ICON/Contrast control
$ i2cset -y 1 0x3e 0x00 0x6c // Follower control
$ i2cset -y 1 0x3e 0x00 0x38 // Function set
$ i2cset -y 1 0x3e 0x00 0x01 // Clear Display
$ i2cset -y 1 0x3e 0x00 0x0c // Display ON/OFF control
$ i2cset -y 1 0x3e 0x40 0x41 // A
$ i2cset -y 1 0x3e 0x40 0x42 // B
$ i2cset -y 1 0x3e 0x40 0x43 // C
```

# Chapter 8 章末問題

「徹底入門」の図 8-20 (P. 189) の配線図において、Pi 4B 用にプルダウン抵抗をタクトイルスイッチに実装しました(図 8-101)。

○図 8-101: 章末問題の配線図



## 【問題 8-1】 D/A 変換処理の関数をライブラリに登録する

「徹底入門」の List8-1\_Dac01.c (P. 181) の D/A 変換処理のソースコードを Mcp4922Write 関数として、ライブラリ (libMyPi.a) に登録します。下記に Mcp4922Write 関数の仕様を示します。登録したライブラリを使用して List8-1\_Dac01 と同じように、ターミナルから 16 進数で 0 から fff までのデジタル値を入力し、MCP4922 の VOUTA と VOUTB の両方からアナログ電圧を出力させ、デジタルマルチメータで出力電圧を確認します。ファイル名は「List8\_Ex01.c」とします。

### ● 仕様

- 関数名 `int Mcp4922Write(int ssNo, unsigned char dacCh, unsigned short dacCode)`
- 引数 `ssNo` スレーブセレクト 0=SS0, 1=SS1
- 引数 `dacCh` 0=VOUTA, 1=VOUTB
- 引数 `dacCode` 12bit DAC コード 0000-0fff
- 戻り値 `wiringPiSPIDataRW` 関数の戻り値

## 【問題 8-2】 A/D 変換処理の関数をライブラリに登録する

---

「徹底入門」の List8-2\_Adc01.c (P. 191) の A/D 変換処理のソースコードを Mcp3208RW 関数として、ライブラリ (libMyPi.a) に登録します。下記に Mcp3208RW 関数の仕様を示します。登録したライブラリを使用して List8-2\_Adc01 と同じように、可変抵抗 VR1 からの入力電圧を 0.5 秒毎に A/D 変換して、ターミナルに ADC コード (16 進数) と電圧値 (小数点第三位) を表示します。ファイル名は「List8\_Ex02.c」とします。

- 仕様

- ・ 関数名 unsigned short Mcp3208RW(int ssNo, unsigned char adcChannel)
- ・ 引数 ssNo スレーブセレクト 0=SS0, 1=SS1
- ・ 引数 adcChannel MCP3208 のチャンネル番号 0 から 7
- ・ 戻り値 12bitAD 変換データ 0000-0fff

## 【問題 8-3】 D/A 変換して出力した電圧を A/D 変換するプログラム

---

DAC MCP4922 の  $V_{OUTB}$  (10 ピン) と ADC MCP3208 の CH1 (2 ピン) をジャンパーワイヤで接続します。次に、ライブラリ (libMyPi.a) に登録した Mcp4922Write 関数と Mcp3208RW 関数を使用して、DAC MCP4922 から出力したアナログ電圧を ADC MCP3208 でデジタル値に変換します。

DAC MCP4922 の  $V_{OUTB}$  より 0.5 秒間隔で、実行例のように 0, 0xff, 0x1ff, 0x2ff … 0xffff までアナログ電圧を 0.5 秒間隔で出力させ、ADC MCP3208 の CH1 でデジタル値に変換して、ターミナルに表示します。ファイル名は「List8\_Ex03.c」とします。

- 実行例

```
DA-VoutB= 0H AD-CH1= 0H 0.000V
DA-VoutB= FFH AD-CH1= FDH 0.204V
DA-VoutB=1FFH AD-CH1=1FEH 0.411V
      ... (中略)
DA-VoutB=EFFFH AD-CH1=EFAH 3.089V
DA-VoutB=FFFFH AD-CH1=FF9H 3.294V
DA-VoutB= 0H AD-CH1= 0H 0.000V
DA-VoutB= FFH AD-CH1= FDH 0.204V
```



## 【問題 8-4】 A/D 変換した値を LCD に表示するプログラム

可変抵抗 VR1 を使用して、0 から 3.3V 範囲で任意の電圧を A/D コンバータ MCP3208 の CH7 に入力します。入力電圧を 0.5 秒毎に A/D 変換して、LCD に ADC コード（16 進数）と「徹底入門」の式 8-5 (P. 188) で計算した電圧を小数点第三位までを表示します（図 8-102）。ファイル名は「List8\_Ex04.c」とします。

○図 8-102: LCD の電圧の表示例

CH7=7FAH 1.645V

## 【問題 8-5】 A/D 変換した値を横棒グラフで LCD に表示するプログラム

可変抵抗 VR1 を使用して、0 から 3.3V 範囲で任意の電圧を A/D コンバータ MCP3208 の CH7 に入力します。入力電圧を 0.5 秒毎に A/D 変換し、17 等分して\*を横棒グラフで LCD に表示します（図 8-103）。17 等分した最下位の電圧を入力したときは\*は無しで、次の位から\*を 1 つ表示し、最高位では\*を 16 個表示します。ファイル名は「List8\_Ex05.c」とします。

○図 8-103: 横棒グラフの表示例



(a) AD7 = 0 V 付近のとき

\*\*\*\*\*

(b) AD7 = 1.66 V 付近のとき

\*\*\*\*\*

(c) AD7 = 3.3 V 付近のとき

- ヒント

Arduino には、ある範囲にある数値を別の範囲の数値に変換する map 関数<sup>1</sup>があります。map 関数の戻り値は、旧範囲の任意の値(value)が新範囲に変換された値です。

- 旧範囲の任意の値(value)
- 旧範囲の最小値(fromLow)、最大値(fromHigh)
- 新範囲の最小値(toLow)、最大値(toHigh)
- 戻り値は変換された値

○リスト:map 関数のソースコード

```
long map(long value, long fromLow, long fromHigh, long toLow, long toHigh)
{
    return(value - fromLow)*(toHigh - toLow)/(fromHigh - fromLow) + toLow;
}
```

- 使用例

```
map(adcCode, 0, 0xffff, 0, 16);
```

## 【問題 8-6】可変抵抗で LED の明るさを制御するプログラム

---

ソフトウェア方式の PWM を利用して、LED0 (GPIO23) の明るさを制御します。制御の方法は、可変抵抗 VR1 と A/D コンバータ MCP3208 を用いて、LED0 への PWM 信号のデューティ比を 0 から 100%まで可変します。0.1 秒毎に可変抵抗の電圧を A/D 変換して、LED の明るさを制御します。また、PWM 信号の周波数を 100Hz とします。ファイル名は「List8\_Ex06.c」とします。

---

<sup>1</sup> <https://www.arduino.cc/reference/en/language/functions/math/map/>

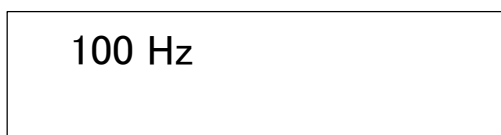
## 【問題 8-7】 可変抵抗で圧電サウンダの周波数を制御するプログラム

SW0 が押されている間、GPIO18 に接続されている圧電サウンダを周期 10ms から  $20\mu\text{s}$  まで可変するパルス信号で鳴らすプログラムを作成します。周期の変更は、可変抵抗 VR1 と A/D コンバータ MCP3208 を用います。パルス信号はハードウェア方式の PWM で作成します。なお、周期から周波数を計算して、LCD に表示します（図 8-104）。0.1 秒毎に可変抵抗の電圧を A/D 変換して、周期を制御します。ファイル名は「List8\_Ex07.c」とします。

### ● 条件

- ・デューティ比                      約 50%
- ・PWM 信号の最小変化幅       $10\mu\text{s}$
- ・PWM 信号の波形のモード    PWM\_MODE\_MS

○図 8-104: 周波数の表示例



### ● ヒント

周期を 10ms から  $20\mu\text{s}$  は、周波数にすると 100Hz から 50kHz の範囲になります。PWM 信号の周波数は、pwmSetRange 関数への引数 range で変更できます。A/D 変換した値を計算して、range に代入します。なお、題意より、PWM 信号の最小変化幅  $10\mu\text{s}$  なので、range が 2 のとき、周期は  $20\mu\text{s}$  となり周波数は 50kHz となります。range、周期、周波数の関係を表 8-101 に示します。

○表 8-101: range、周期、周波数の関係

range	周期 (sec)	周波数 (Hz)
2	0.00002	50,000
3	0.00003	33,333
...中略		
500	0.005	200
...中略		
999	0.00999	100
1000	0.01	100

## 【問題 8-8】 電圧を測定してファイルに保存するプログラム

SW0 を押すと、1 秒毎に時刻と VR1 の電圧（小数点第三位まで）、改行をテキストモードでファイルに記録します。記録中は LED0 を点灯させます。また、LCD の 1 行目に時刻、2 行目に電圧値を表示します(図 8-105)。SW1 を押すと測定を停止して、ファイルをクローズし、LED0 を消灯させて、プログラムを終了します。ファイル名は「List8\_Ex08.c」とします。

測定したデータを保存するファイルは、CSV 形式とし、ファイル名を年月日時間秒、拡張子を.csv とします。例えば、プログラムを実行する時間が 2020 年 2 月 24 日 12 時 03 分 54 秒であれば、20200224120354.csv とします。CSV は、Comma Separated Value の略で、「カンマ区切り」とも呼ばれ、値をカンマ(,)で区切ったものです。改行はレコードの区切りを表します(表 8-102)。ファイル名を年月日時間秒とする方法については、章末問題の問題 7-4 を参照してください。

表計算ソフト(LibreOffice Calc)で CSV ファイルを開いて、図 8-106 のグラフ(線)を作成し、タイトルや凡例を入力します。グラフを含めて保存する場合は、ファイルを ODF 表計算ドキュメント(.ods)形式で保存します。

○図 8-105:時刻と電圧の表示例

12:03:56
0.006 V

○表 8-102:CSV ファイルのレコードフォーマット

時刻	カンマ	電圧値	改行
12:03:56	,	0.006	↵
12:03:57	,	0.007	↵
12:03:58	,	0.006	↵

・・・(以下、略)

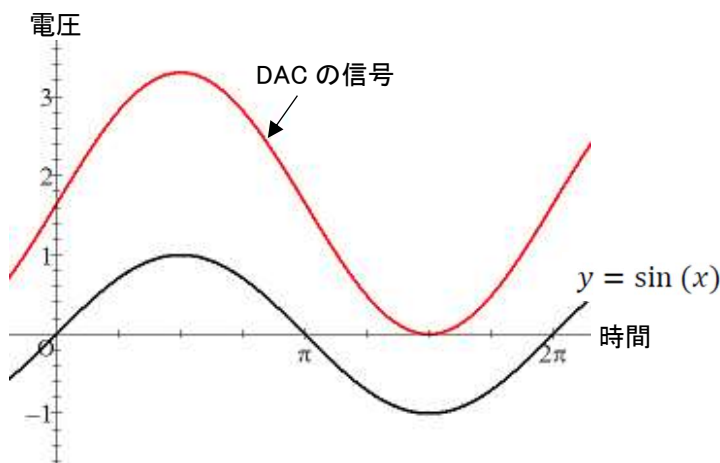
○図 8-106: 電圧測定のグラフ例



## 【問題 8-9】 正弦波の信号を出力するプログラム

D/A コンバータ (MCP4922) の  $V_{OUTA}$  から正弦波(sin)の信号を出力します。図 8-107 に示すように、信号の出力電圧の範囲を 0V~3.3V とします。正弦波の 1 周期を 360 等分して計算し、信号は繰り返し出力するようにします。オシロスコープで MCP4922 の  $V_{OUTA}$  を測定します。ファイル名は「List8\_Ex09.c」とします。なお、オシロスコープが無い場合は、図 8-108 に示す測定波形と実行例の解説を参照してください。

○図 8-107: 正弦波と DAC の出力波形



### ● ヒント

- sin 関数を利用するには数学のヘッダファイル math.h をインクルードします。
- gcc コマンドに数学オプションの「-lm」を追加します。  
`gcc -Wall -o "%e" "%f" -lwiringPi -lpthread -g -O0 libMyPi.a -lm`
- sin 関数の角度は弧度法になります。その単位を rad(ラジアン)と呼びます。正弦波の 1 周期は 60 分法では  $360^\circ$  ですが、弧度法では  $2\pi$  [rad]となります。弧度  $\theta$  と 60 分法の角度  $\theta^\circ$  には式 8-101 の関係があります。

$$\text{弧度 } \theta = \pi \times \text{角度 } \theta^\circ \div 180^\circ \quad [\text{rad}] \quad [\text{式 8-101}]$$

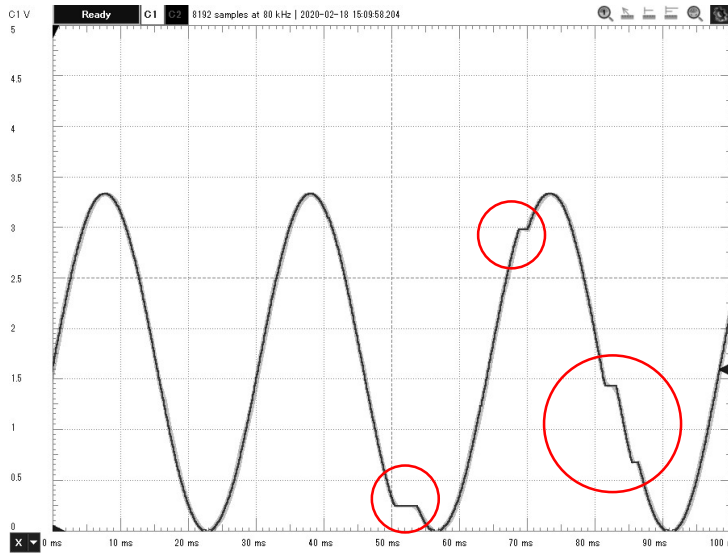
- 円周率( $\pi$ )は PI とマクロ定義します。  
`#define PI 3.141593F` F は float 型の定数を示すサフィックスです。
- 図 8-107 に示すように sin 関数は、-1 から+1 の間で振幅します。正弦波を 0V から 3.3V の間で振幅させるには、振幅を最大電圧 3.3V の半分とし、オフセット電圧も 3.3V の中間とします。正弦波信号の出力電圧を V、弧度を  $\theta$  [rad]とすると、題意の正弦波の数式は式 8-102 となります。

$$V = 1.65 \times \sin(\theta) + 1.65 \quad [\text{V}] \quad [\text{式 8-102}]$$

- 実行例

正弦波の周波数は 30～40[Hz]程度になります。マルチタスク OS でプログラムを実行しているため、大きな負荷が発生すると List8\_Ex09 のプログラムを一時的に中断させるため、周波数が変化したり、出力が止まったりすることがあります(図 8-108)。

○図 8-108: List8\_Ex09 を実行中に Web Browser のアプリを起動したとき



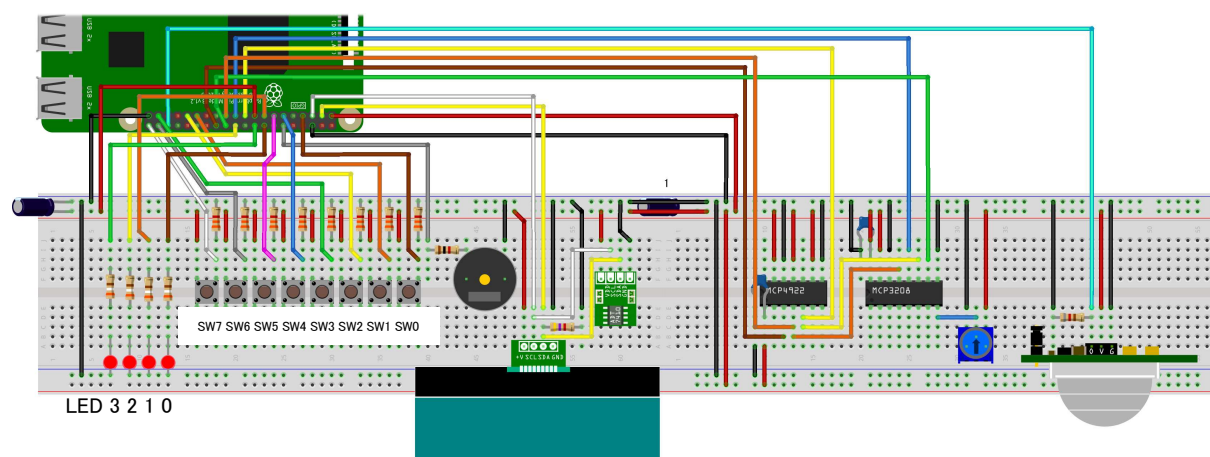
# Chapter 9 章末問題

「徹底入門」の図 8-20 (P. 189) の配線図において、Pi 4B のタクタイルスイッチ用にプルダウン抵抗と焦電センサモジュールを実装しました(図 9-101)。Pi カメラモジュールを Raspberry Pi 本体に取り付けます(図 9-102)。問題 9-2 は、表 9-101 に示す焦電センサモジュールを使用して人を検知する課題内容です。

○表 9-101: 準備するもの

名称	個数	備考	参考品 (秋月電子通商)
焦電センサモジュール	1	人を検知するために使用します。	M-06835

○図 9-101: 章末問題の配線図



○図 9-102: Pi カメラモジュール



## 【問題 9-1】 静止画のエフェクトを確認するプログラム

カメラ制御オプションのエフェクトを使用して、その効果をプレビューで確認しましょう。  
表 9-102 に示すように、各タクトイルスイッチを押すと対応したエフェクトで加工された静止画を約 5 秒間プレビューします。raspistill コマンドと「徹底入門」の表 9-5 (P. 202) に示すエフェクト・オプション「-ifx」を使用します。ファイル名を「List9\_Ex01.c」とします。

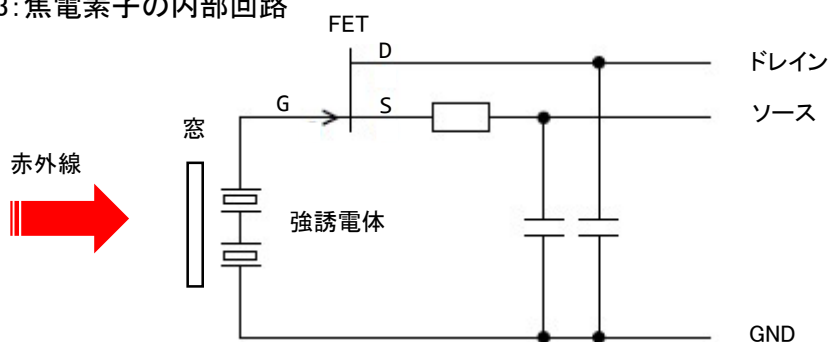
○表 9-102: タクトイルスイッチとエフェクト

タクトイルスイッチ	エフェクト
SW0	none(エフェクトなし)
SW1	negative(反転)
SW2	solarise(ソラライズ)
SW3	posterise(ポスタライズ)
SW4	sketch(スケッチ)
SW5	denoise(ノイズ除去)
SW6	emboss(エンボス)
SW7	oilpaint(油絵)

## 【問題 9-2】 人を検知したら撮影するプログラム

人を検知して自動的に照明が点灯したり、いなくなると消灯したりする人感センサが省エネや防犯などに役立っています。人感センサには、赤外線、超音波、可視光などが用いられます。赤外線タイプのセンサでは焦電素子 (Pyroelectric sensor) を使用して、人体から放射される赤外線を検出します。赤外線の放射量は温度に比例しています。焦電素子に内蔵された強誘電体は赤外線を受けると、赤外線の変化に比例して表面に電荷が誘起されます。この現象を「焦電効果」と呼びます。図 9-103 に示す焦電素子の窓はフィルタになっており、目的の波長を検出します。内部の回路は、2 個の強誘電体を使用したデュアル型で、FET (電界効果トランジスタ) により検出信号を出力します。

○図 9-103: 焦電素子の内部回路

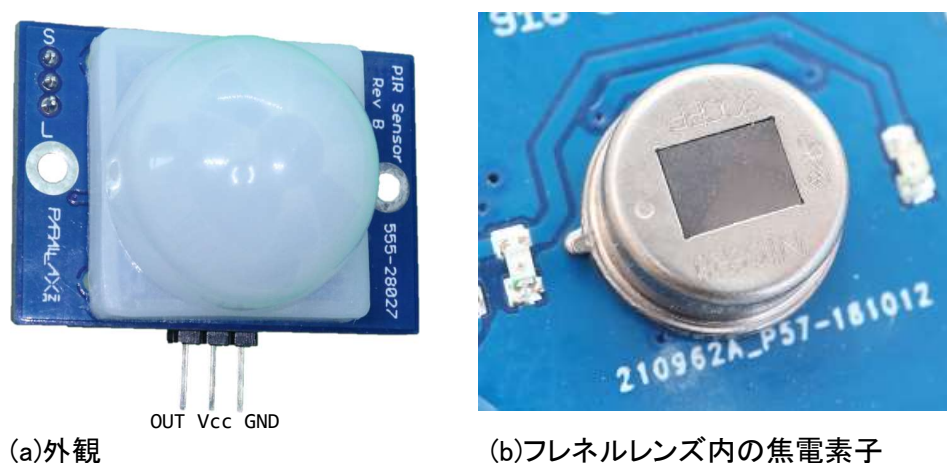




焦電センサモジュールは焦電素子と周辺回路を内蔵させたものです。赤外線効率良く集光するために、センサにはドーム型フレネルレンズ<sup>2</sup>が取り付けられています。今回、Parallax社の焦電センサモジュールを使用します（表 9-103）。検出範囲は短距離と長距離に分かれ、モジュール本体のジャンパーピンで切り替えます。人が焦電センサモジュールに近づくと、赤外線の変化を受けて強誘電体に電荷が発生して、HIGH 信号を出力します。その後、人の動きが止まった場合は赤外線の変化が無いので強誘電体の電荷も変化しません。そのため、出力信号はLOW になります。焦電センサは人の動きを検知するのです。

焦電センサモジュールのジャンパーピンを切り替えて、検出範囲を Short range (S) に設定します。出力信号 (OUT) をラズパイの GPIO16 に接続し、立ち下りエッジを急峻にするため抵抗 820  $\Omega$  でプルダウンします。GPIO16 を割込みに設定し、出力信号 (OUT) の立ち上がりエッジを検出して、静止画を撮影します。静止画のファイル形式は「徹底入門」の List9-1\_Cam01.c (P. 207) を参照してください。なお、main 関数では、5 秒毎にターミナルに” hello, world” と繰り返し表示します。 ファイル名を「List9\_Ex02.c」とします。

○図 9-103: Parallax 社の焦電センサモジュール



(a)外観

(b)フレネルレンズ内の焦電素子

○表 9-103: Parallax 社の焦電センサモジュールの仕様

項 目	内 容
動作電源電圧	3V~6V
検出範囲	Shorter range 約 4.6m Longer range 約 9m
出力信号	動き有り: HIGH、赤色 LED 点灯 動き無し: LOW、赤色 LED 消灯

#### ● 仕様

- main 関数で 5 秒毎にターミナルに” hello, world”を表示します。
- date 関数を利用してファイル名を日時、JPGE フォーマットで保存します。  
2020 年 2 月 27 日 15 時 01 分 37 秒に記録した場合は「20200227-150137.jpg」になります。

<sup>2</sup> フランスの物理学者オーギュスタン・ジャン・フレネル (Augustin Jean Fresnel, 1788 年 5 月 10 日 - 1827 年 7 月 14 日) の発明に由来する。

# 参考文献/参考資料

- [1] Peter Prinz, Tony Crawford, 島敏博 監修, 黒川利明 翻訳, “C クイックリファレンス 第2版 (日本語),” オライリージャパン (Nov. 2016)
- [2] IPA, “【改訂版】組込みソフトウェア開発向け コーディング作法ガイド [C 言語版] ESCR Ver.3.0,” 独立行政法人情報処理推進機構 (Jun 2018) <https://www.ipa.go.jp/sec/publish/tn18-004.html>
- [3] 福岡知的クラスタ(第1期)組込みソフト開発プロジェクト, “組込み現場の「C」プログラミング 標準コーディングガイドライン,” 技術評論社 (Oct. 2007)
- [4] MISRA-C 研究会, “組込み開発者における MISRA-C:2004—C 言語利用の高信頼化ガイド,” 日本規格協会 (Oct. 2006)

シーげんご      ラズベリーパイ      てっていにゅうもん  
C言語ではじめるRaspberry Pi 徹底入門

しょうまつもんだい      Chapter 4 ~ Chapter 9  
章末問題

2020年5月31日 初版

©2020 菊池達也