Basic

ラインレースカーによる開発実習

マイコン基礎編

本テキストを使用する為に必要なモノ

- 実習機器
  - パソコン
  - PK-LTC
  - ドック
  - 基礎編の部品一式
  - 単三電池×２
  - オーバルコース(A3サイズ)

推奨する動作環境

- OS　　: Windows7/8/8.1/10
- CPU　　: Intel Core i5以上(2.0GHz以上を推奨)
- RAM　　: 4GB以上
- I/F　　: USBポート(3個以上)
- 画面　　: HD1080(1920×1080)以上

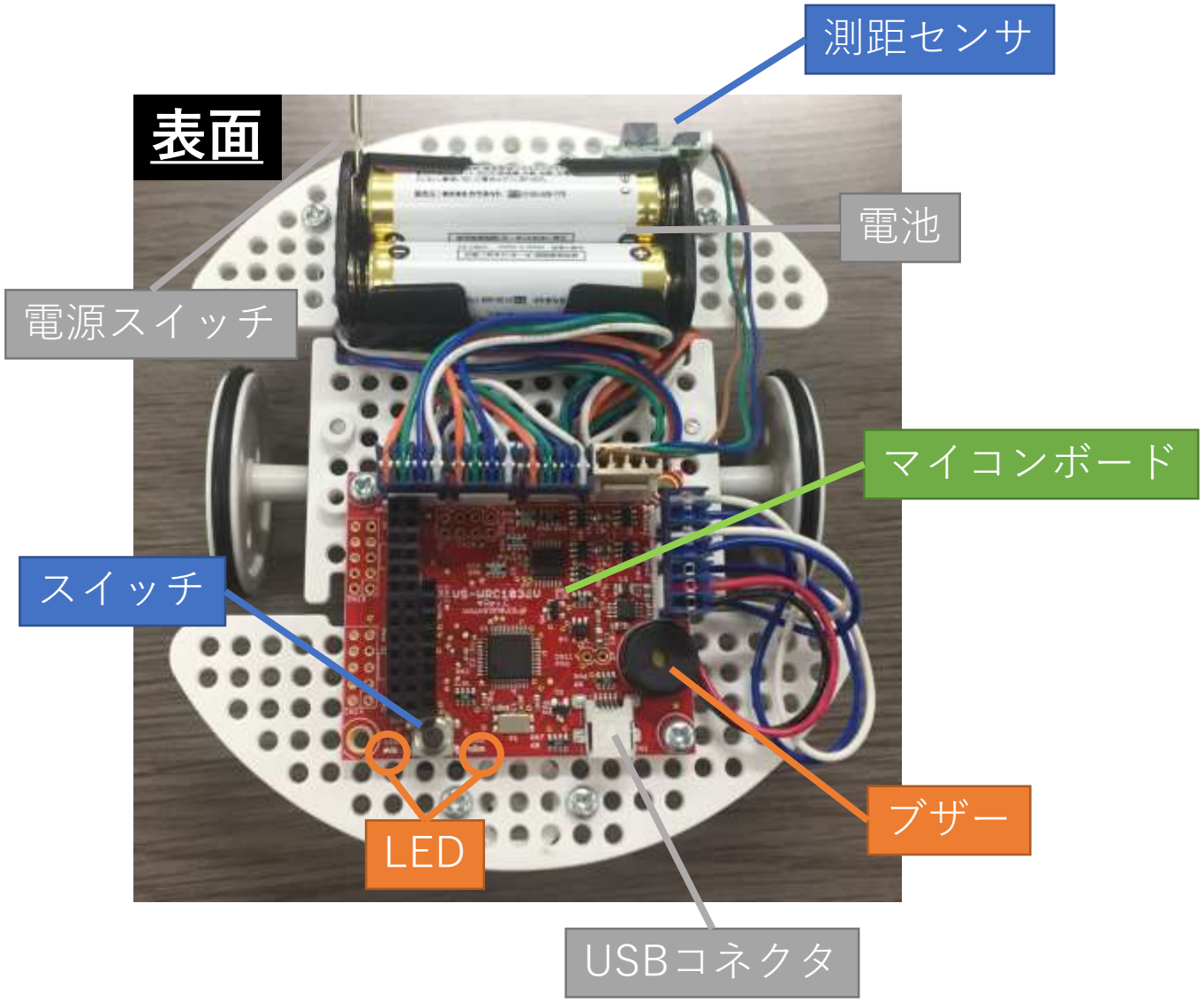
- 使用するソフトウェア
  - LPCXpresso IDE V8.2.2
  - TeraTerm V4.98

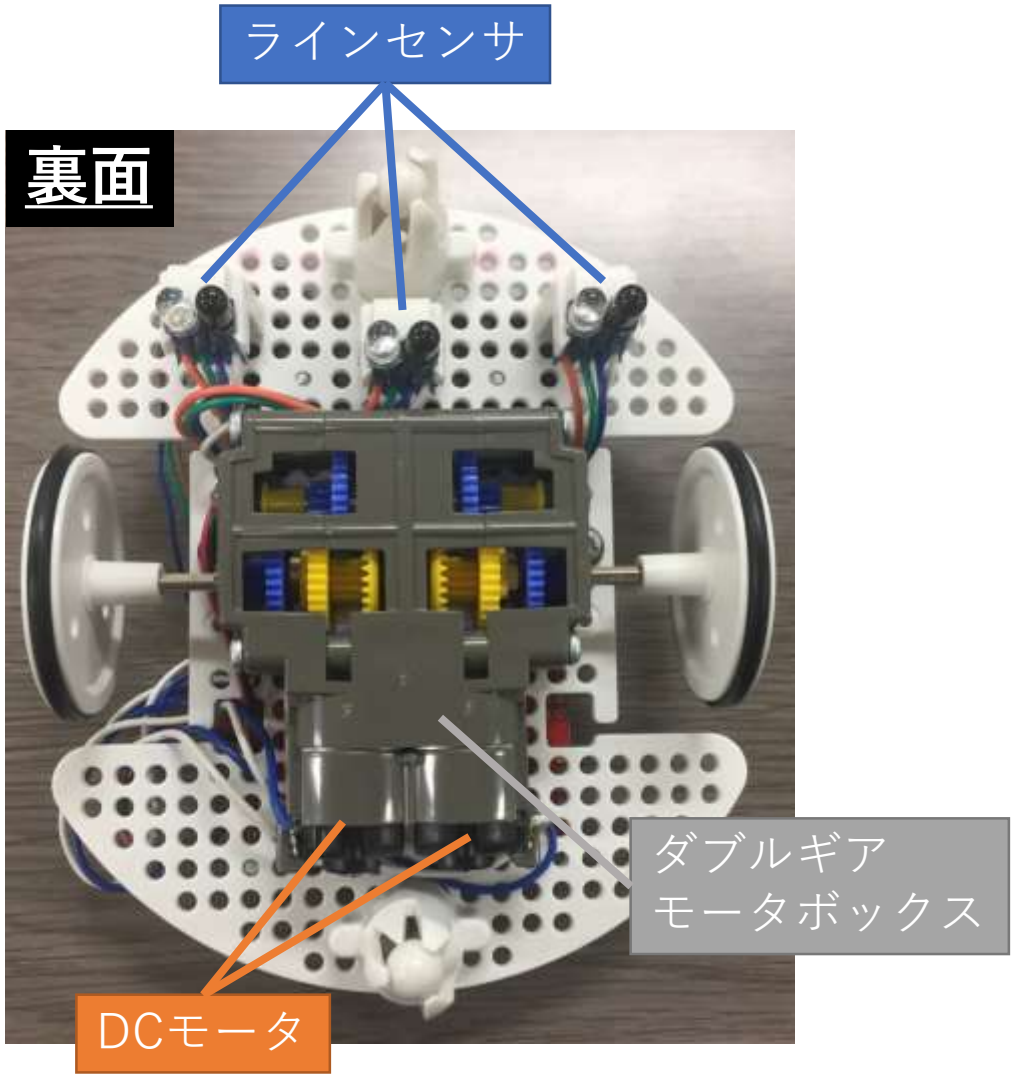
マイコン基礎編の目的と目標

- 目的
  - マイコン制御の基本であるIO制御やAD変換などの知識・技術の習得

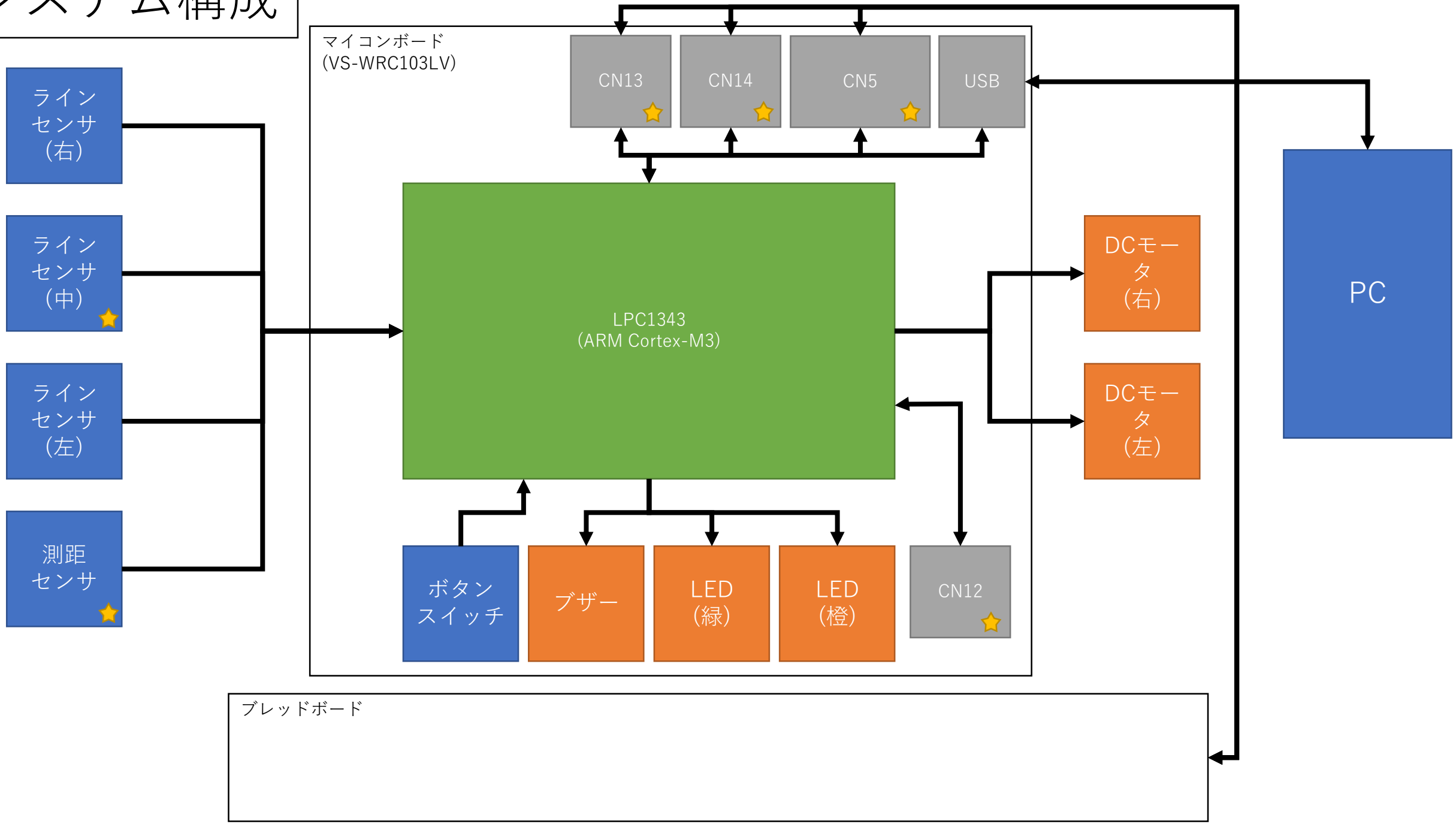
- 目標
  - C言語を用いてGPIOおよびAD変換制御ができるようになること
  - 習得した技術を活用してライントレース制御ができるようになること

PK-LTCの仕様


表面
測距センサ
電池
電源スイッチ
マイコンボード
スイッチ
LED
ブザー
USBコネクタ


裏面
ラインセンサ
ダブルギア
モータボックス
DCモータ

システム構成

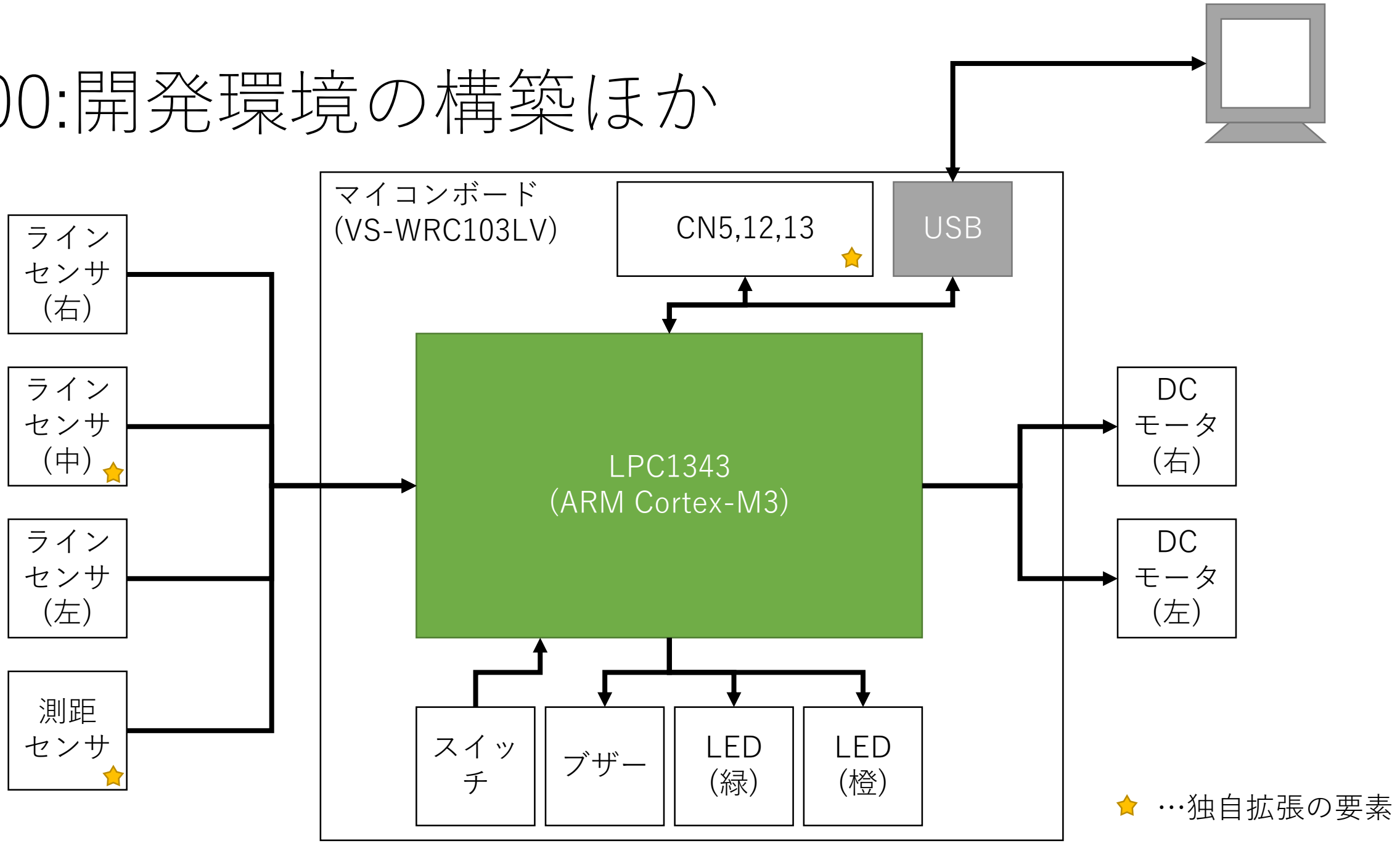

マイコンボード
(VS-WRC103LV)
CN13
CN14
CN5
USB
ライン
センサ
(右)
ライン
センサ
(中)
ライン
センサ
(左)
測距
センサ
LPC1343
(ARM Cortex-M3)
DCモー
タ
(右)
DCモー
タ
(左)
PC
ボタン
スイッチ
ブザー
LED
(緑)
LED
(橙)
CN12
ブレッドボード

LPC1343マイコンの概要

- LPC1343のスペック概要
  - CPU　　:ARM Cortex-M3 72MHz
  - Flash　:32KB …今回は開発環境からの制限で8KBまで使用可
  - SRAM　:8KB
  - GPIO　:42本(共用pinあり)
  - ADC　　:8本 分解能10bit
  - 通信　　:UART,USB,I2Cほか
- VS-WRC103LVからの制約
  - 動作電圧は3.3V
  - プログラムの書き込みはUSB経由

- データシートのDescriptionやFeaturesを読もう！
  - マイコンの機能概要が書かれている。ここを読めばどんなマイコンなのかザックリ分かる
  - 複数の型番のマイコンの事が1つのデータシートに書かれている事が多いので注意すること

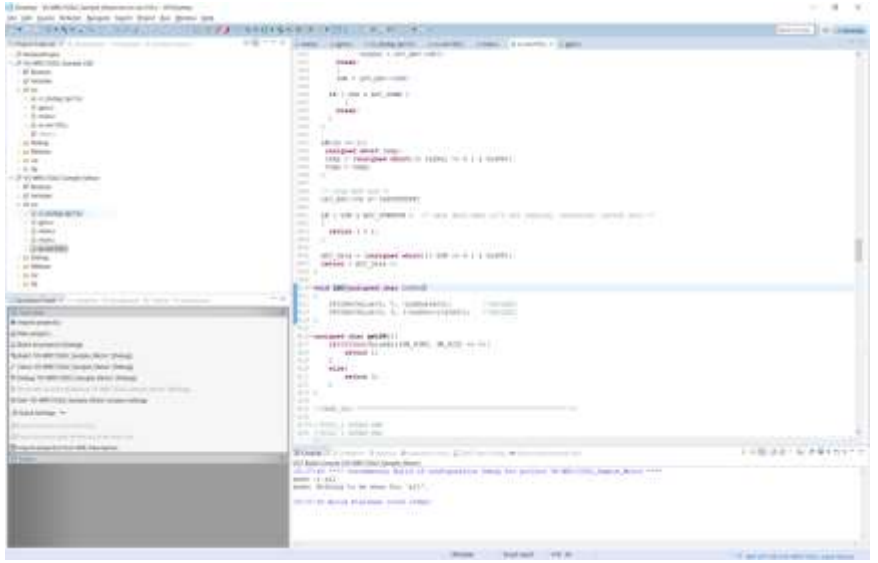
00:開発環境の構築ほか



開発環境の構築・使い方

- VS-WRC103LV_取扱説明書を合わせて使用しながら実施
  - 今回使用するIDEはLPCXpresso 8.2.2

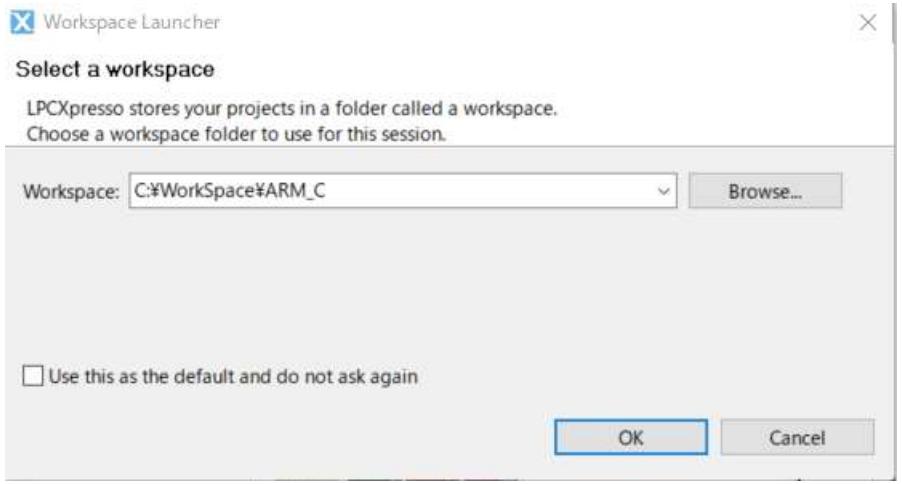
- LPCXpressoのダウンロード
  - 共有フォルダからDLしてください
- LPCXpressoのインストール方法
  - 取説 P17 6-2 LPCXpressoのインストール を参照
    - インストールするだけでは8KB制限がかかってしまうが、今回の開発規模ではプログラムのサイズが8KB以上超える事は稀であると予想される為、アクティベーションは行わない
- LPCXpressoの起動
  - 取説 P20 6-3 LPCXpressoの起動と認証
    - アクティベーションに関する作業は不要
- プロジェクトの作成・ビルド手順
  - 本テキストの次ページ以降を参照
- プログラムの書き込み手順
  - 取説 P33 6-5 VS-WRC103LVへのプログラムの書き込み方法を参照



プロジェクトの作成とビルド手順

ワークスペースの選択

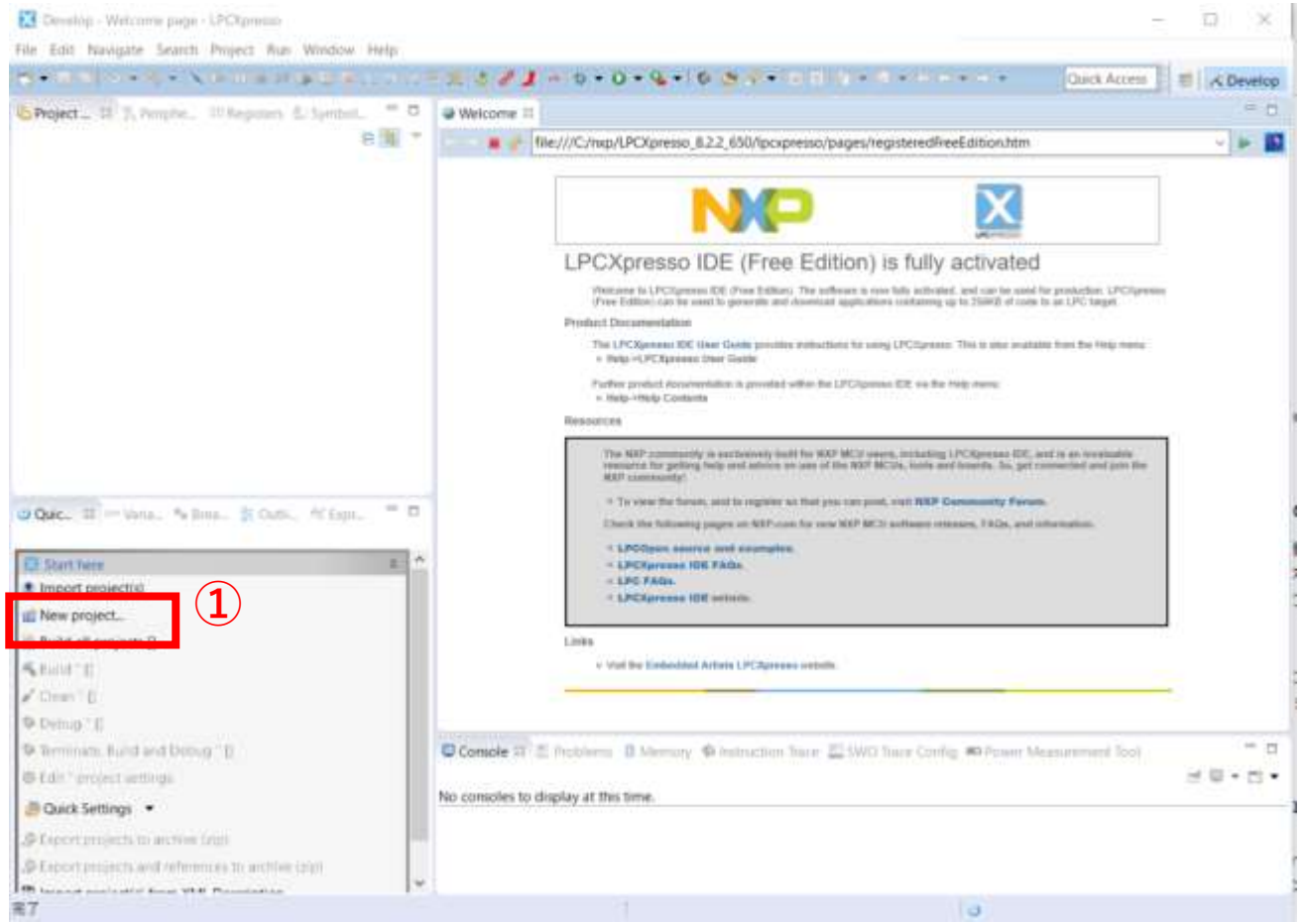
- デスクトップより「LPCXpresso」を起動
- ワークスペースフォルダの変更
  - 変更先： C:¥WorkSpace¥ARM_C
  - 【変更の理由】LPCXpressoのワークスペースはパスに日本語などの全角文字が含まれると正常にビルドできない為

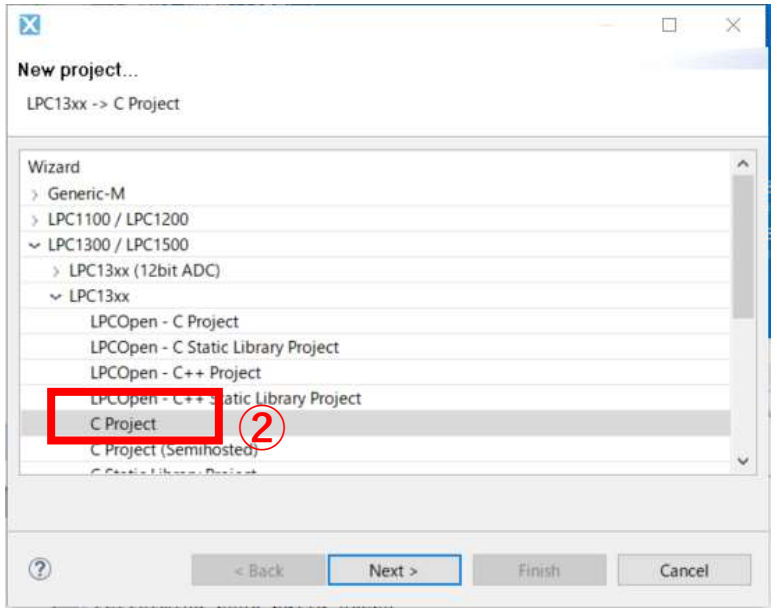


プロジェクトの新規作成１

① QuickStart PanelよりNewProjectを選択

② NewProject…画面よりLPC1300/1500>LPC13xx>C Projectを選択



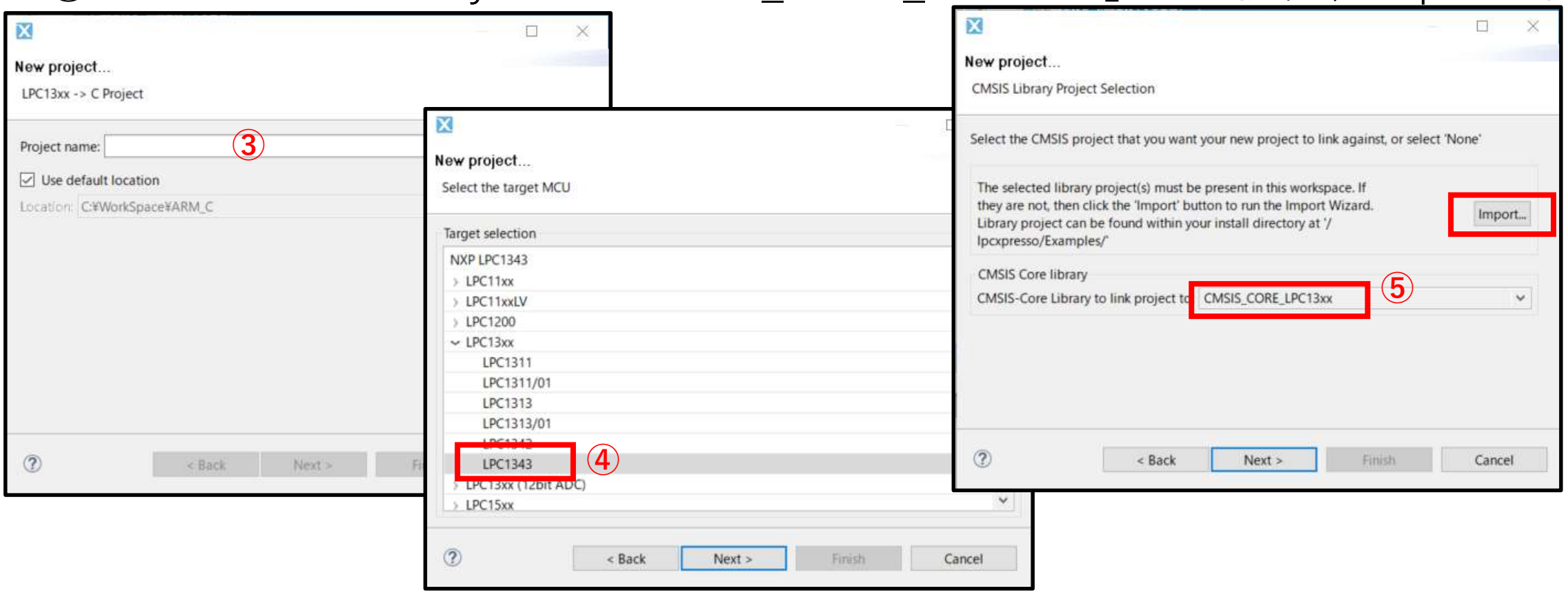


プロジェクトの新規作成２

③ Project nameに「test01」と入力

④ Target selectionよりLPC13xx>LPC1343を選択

⑤ CMSIS Core libraryより「CMSIS_CORE_LPC13xx」を選択後、importを押



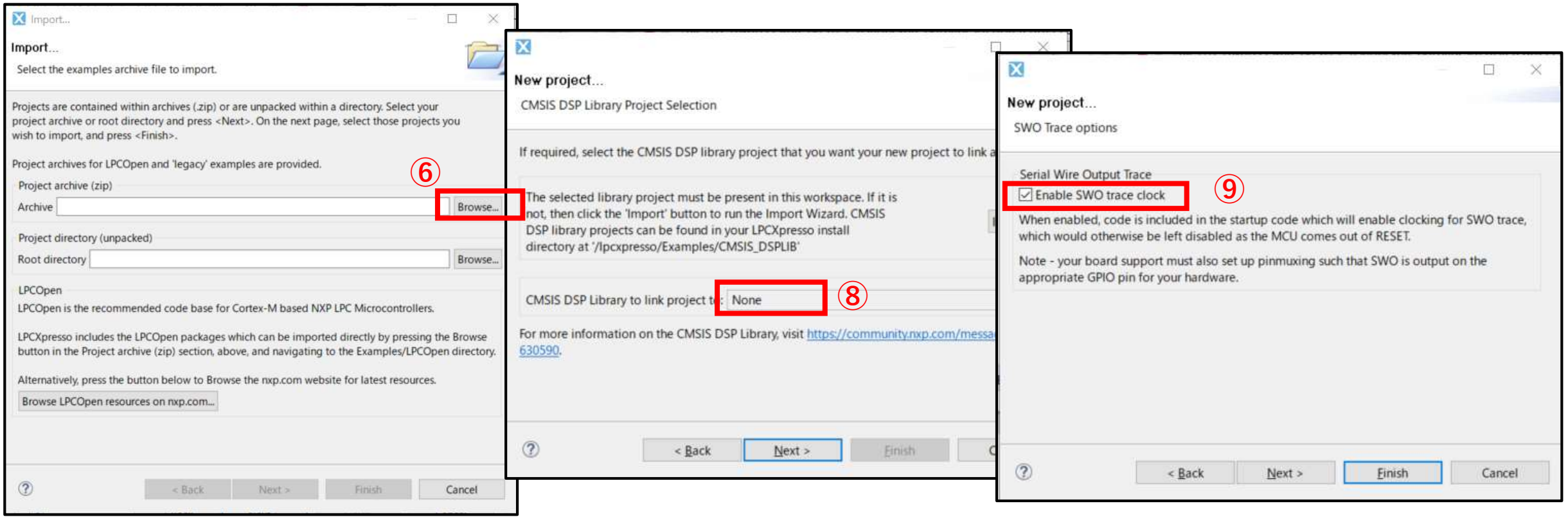
プロジェクトの新規作成３

⑥ Project archive(zip)よりBrowseボタンを押下
- 「C:¥nxp¥LPCXpresso_8.2.2_650¥lpcxpresso¥Examples¥Legacy¥CMSIS_CORE¥ CMSIS_CORE_Latest.zip」を開いて、Next押下

⑦ ProjectよりCMSIS_CORE_LPC13xx(CMSIS_CORE_LPC13xx)を選択し、Finish

⑧ CMSIS DSP Library Project Selectionは「none」でNext

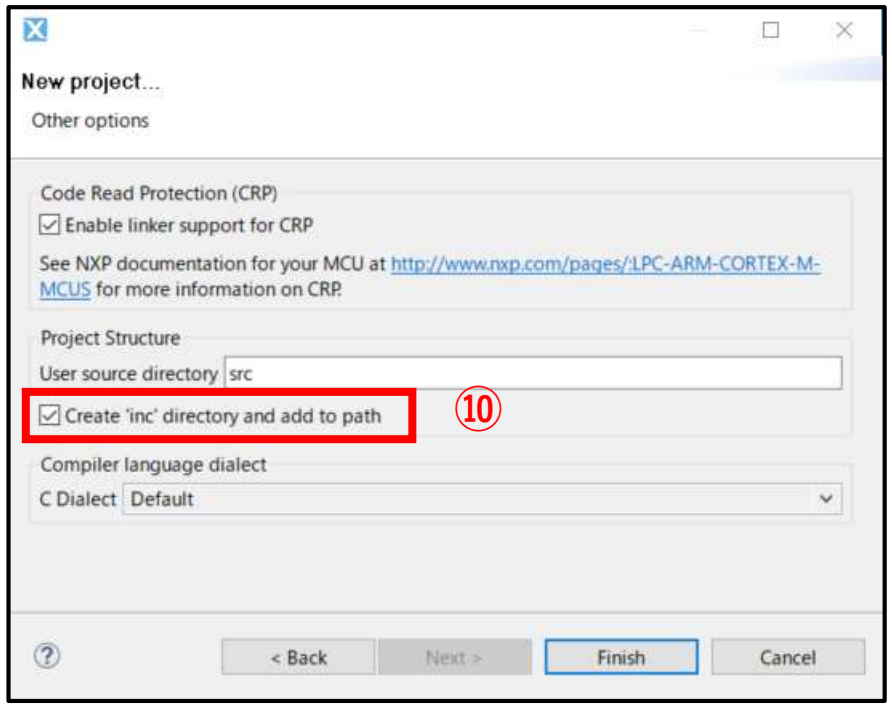
⑨ SWO Trace Output Traceはチェックして Next

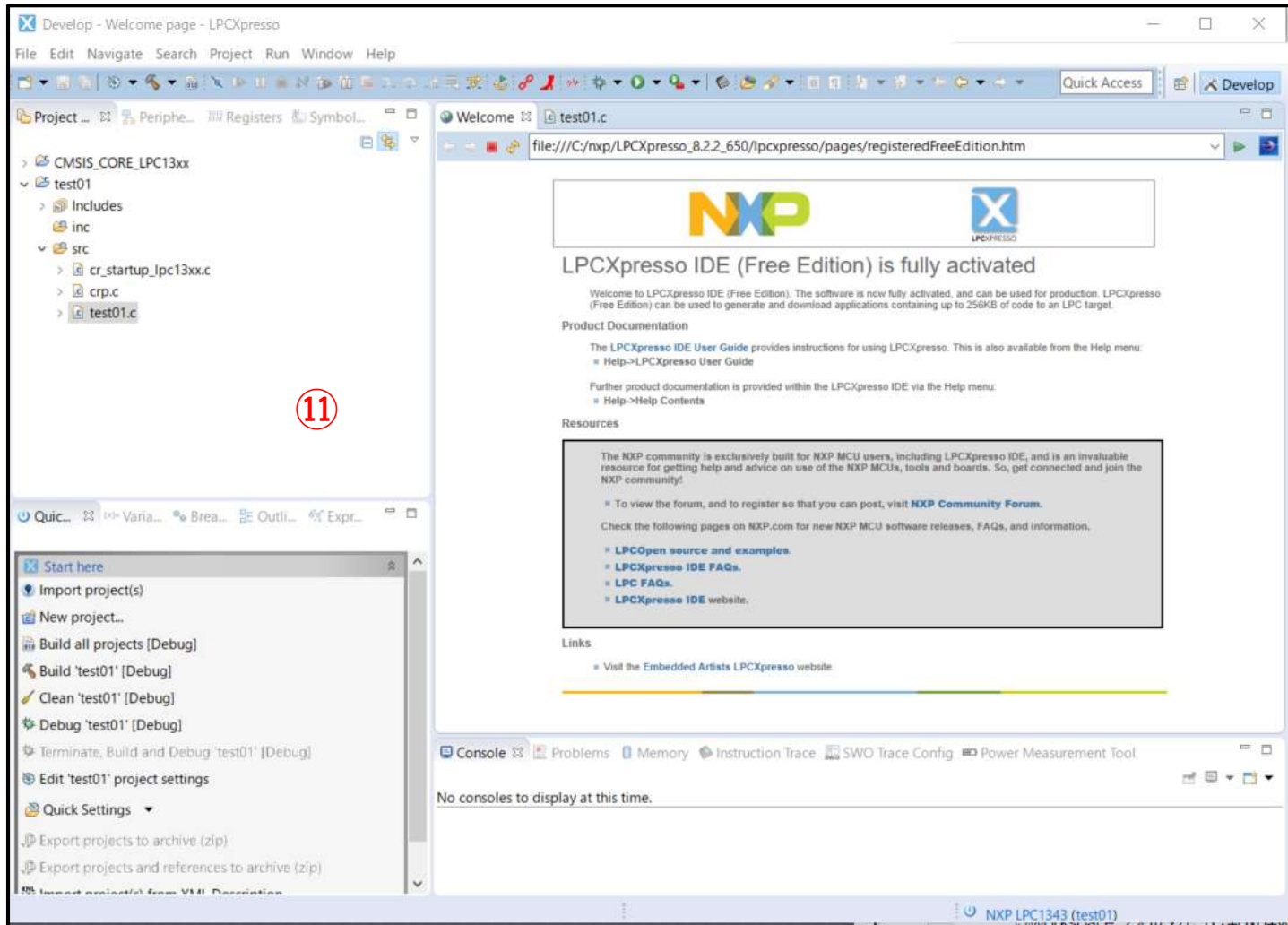


プロジェクトの新規作成４

⑩ Other optionsはincにチェックを入れてFinish

⑪ プロジェクトが完成

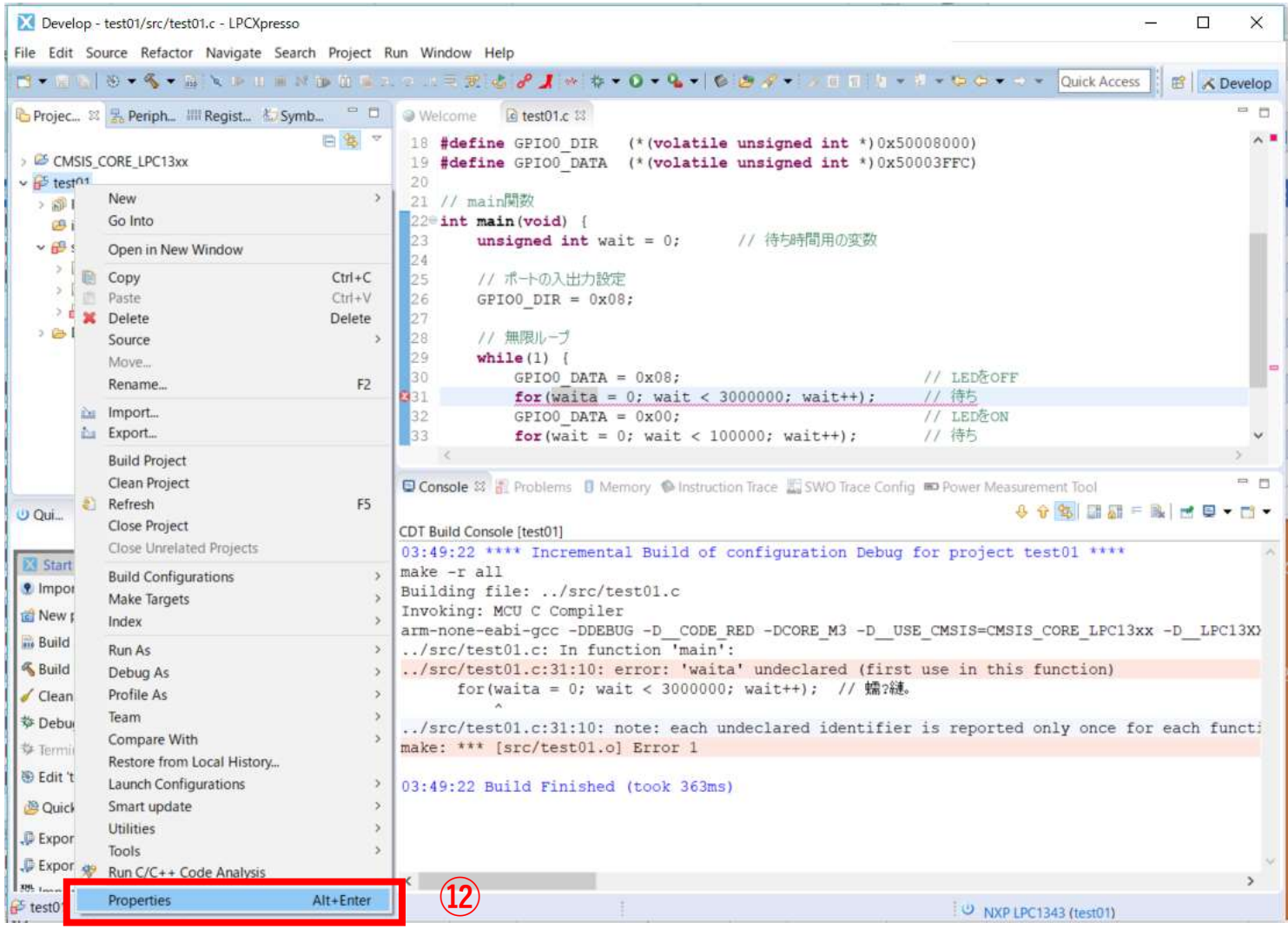




binファイル生成の為の設定１

- プロジェクトの作成は完了しましたが、実際にコードを開発するにあたって、もう1つ設定する項目があります。
- 通常、axfと呼ばれるデバッガ用のファイルしか生成されない。そこで、axfからbinファイルを生成するコマンドを設定する

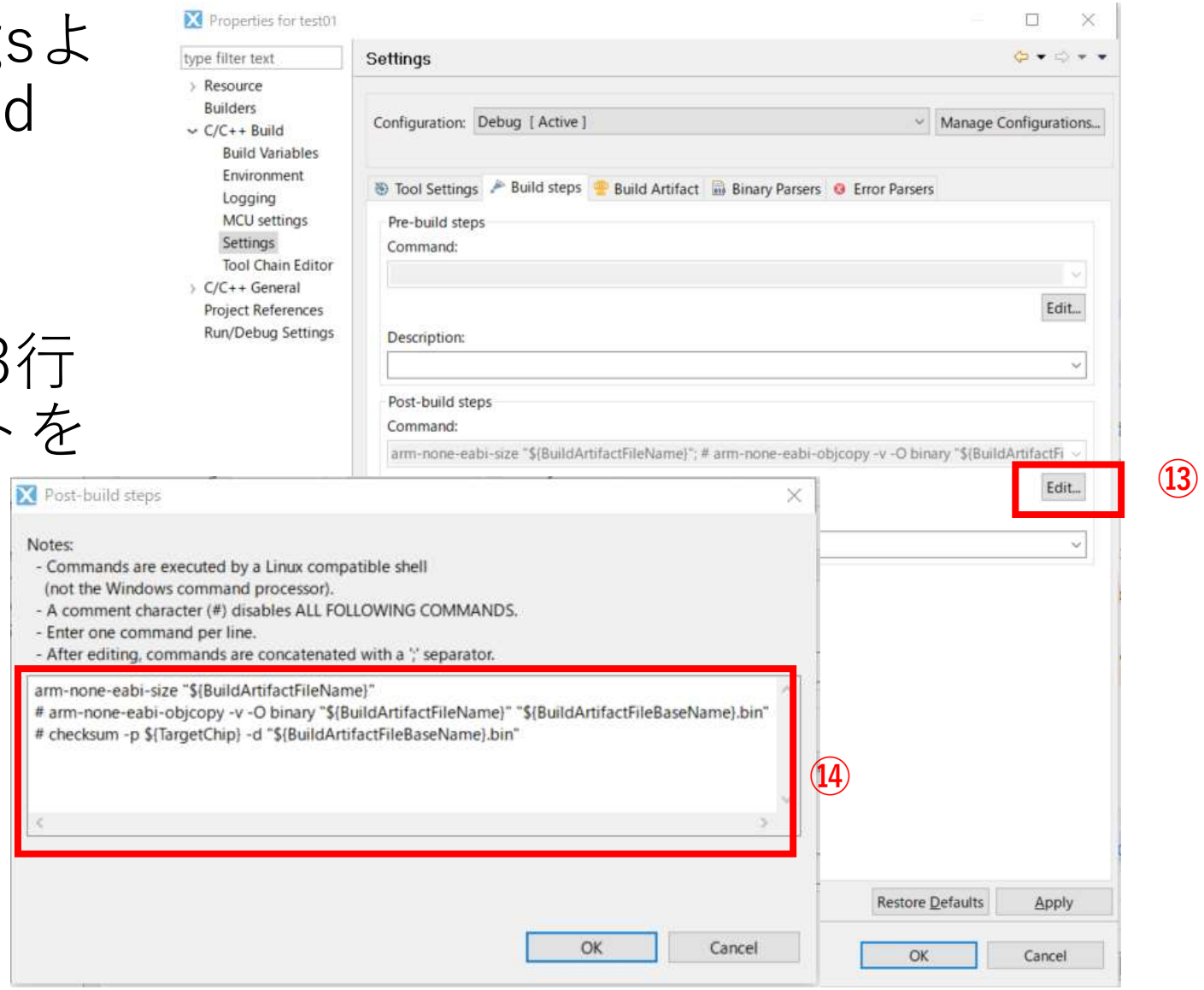
⑫ ProjectExplorerよりtest01プロジェクトを右クリック>Propertiesをクリック



binファイル生成の為の設定

⑬ C/C++ Build > SettingsよりBuild steps > Post-Build steps > Editボタン押下

⑭ 既存の3行のうち、2～3行目の冒頭にある # コメントを削除



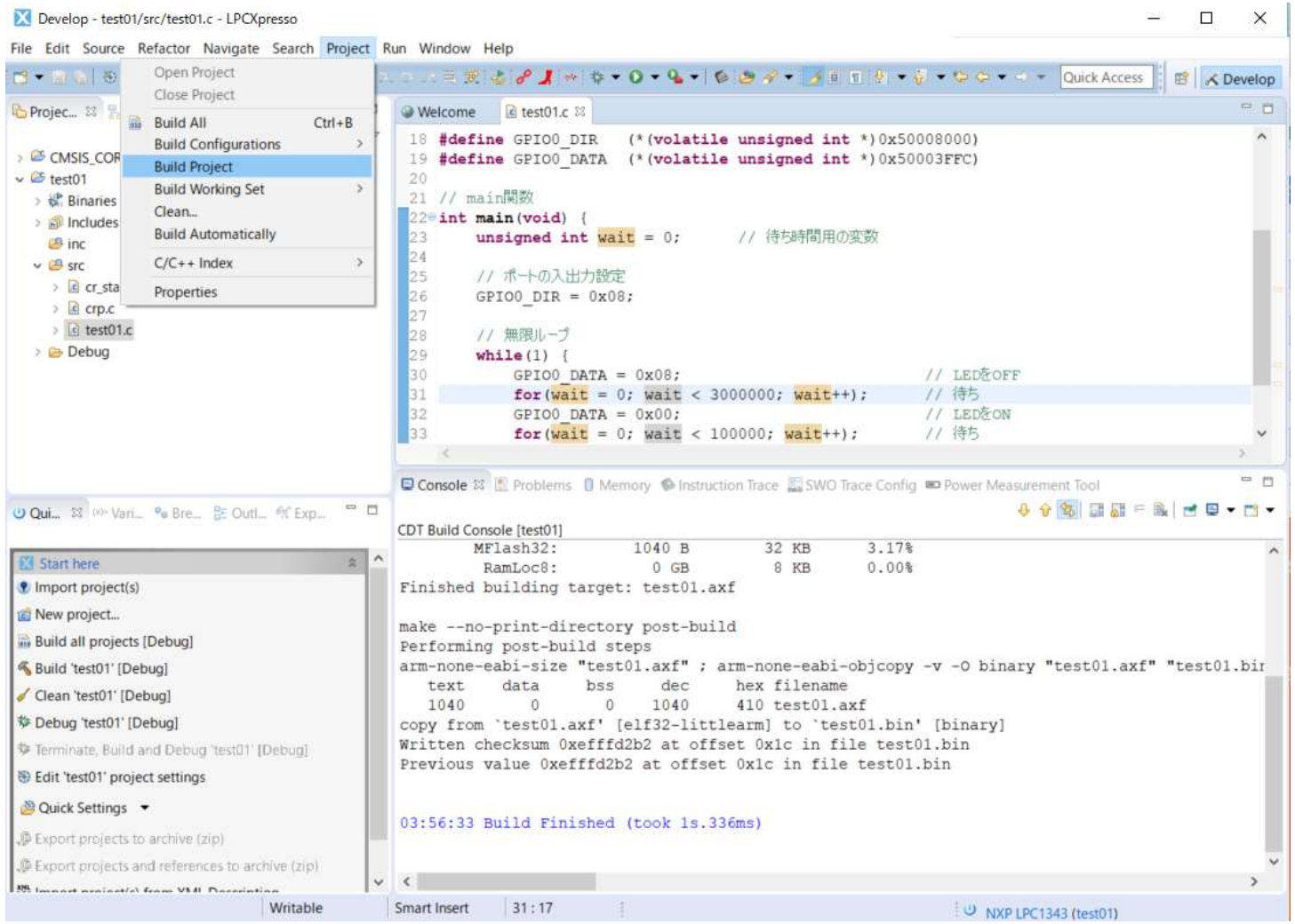
初めてのマイコンプログラミング

- 初めてのマイコンプログラミングと言えばLEDをチカチカと点滅させるのが定番です
  - 組込みにおける「Hello World」です
- test01.cに右のソースコードを打ち込んで動作を確認してみましょう

```
#ifdef __USE_CMSIS
#include "LPC13xx.h"
#endif

#include <cr_section_macros.h>

// レジスタ定義
#define GPIO0_DIR (*(volatile unsigned int *)0x50008000)
#define GPIO0_DATA (*(volatile unsigned int *)0x50003FFC)

// main関数
int main(void) {
    volatile unsigned int wait = 0;// 待ち時間用の変数

    // ポートの入出力設定
    GPIO0_DIR = 0x08;

    // 無限ループ
    while(1) {
        GPIO0_DATA = 0x08;// LEDをOFF
        for(wait = 0; wait < 1000000; wait++);// 待ち
        GPIO0_DATA = 0x00;// LEDをON
        for(wait = 0; wait < 1000000; wait++);// 待ち
    }
    return 0 ;
}
```

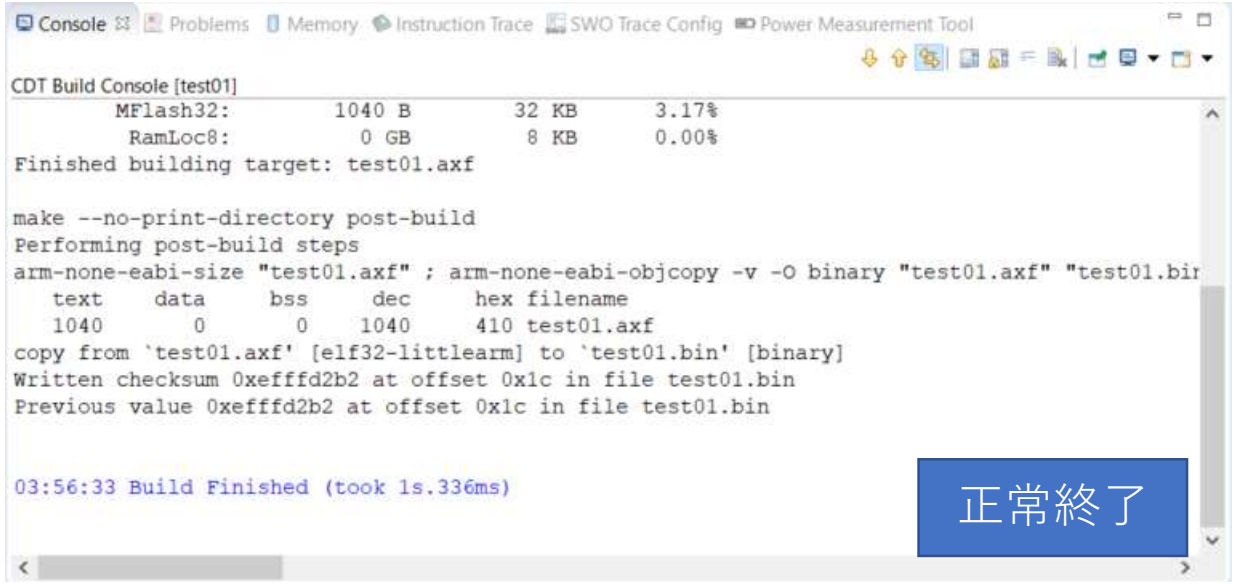
プロジェクトのビルド

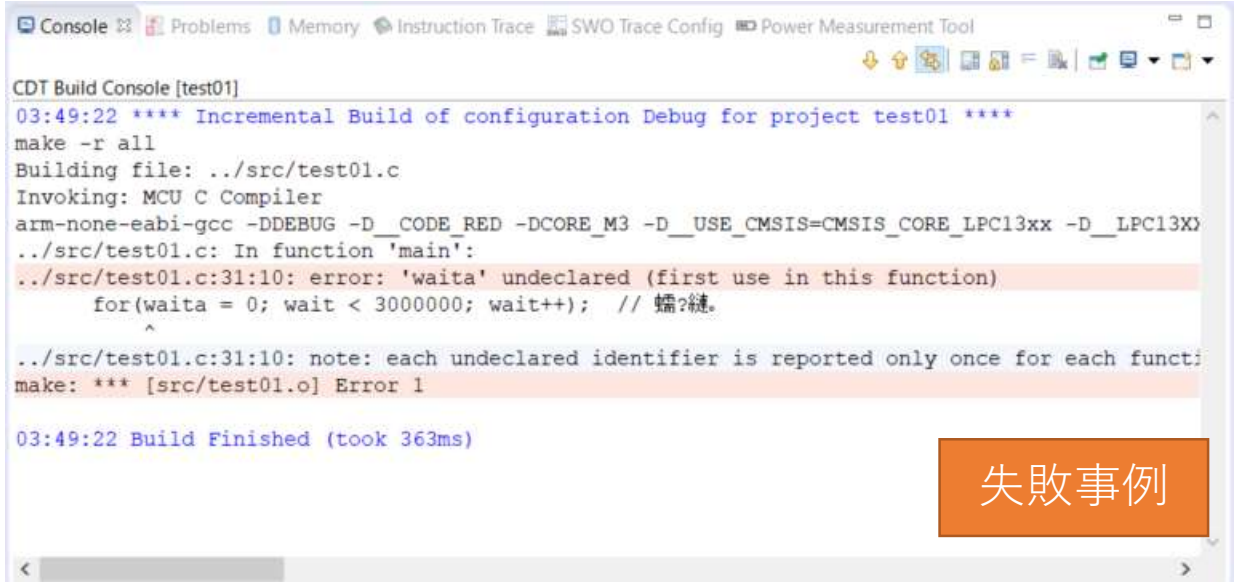
- ProjectExplorerよりtest01プロジェクトを選択
- メニューのProject>Build Projectをクリック
  - これにより該当のプロジェクトのビルドがスタート



ビルド正常終了と失敗事例

- ビルドが正常に終了すると、Consoleの左記のようなログが流れます。Binファイルが生成されて、Build Finishedの文字が見えれば成功です
- ビルドが失敗に終わると、問題点について赤くマークアップされる。





プログラムの書き込み手順

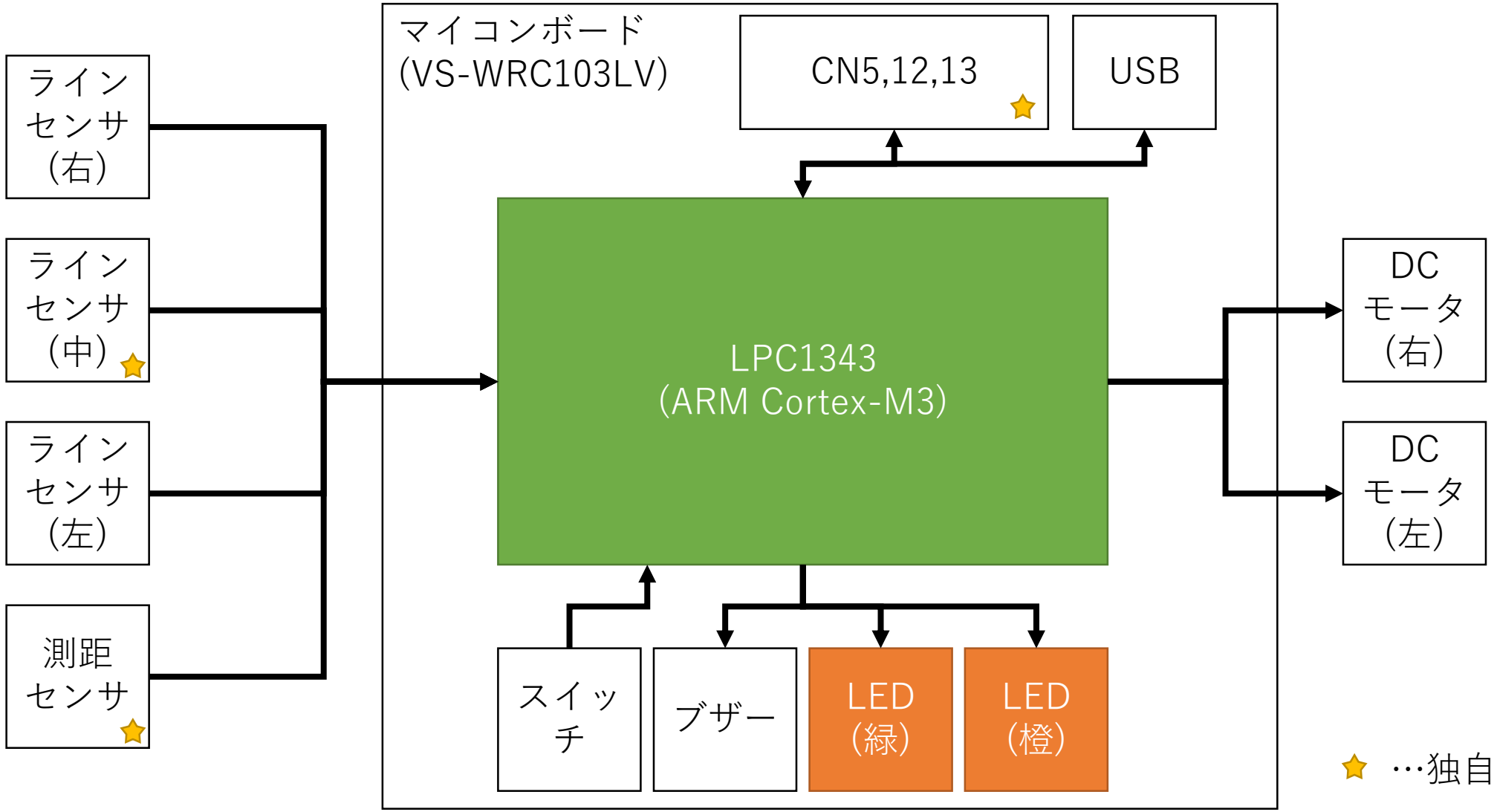
- 取説 P33 6-5 VS-WRC103LVへのプログラムの書き込み方法 を参照

ようこそ！組込みの世界へ

- 基板上のオレンジ色のLEDが点滅することを確認できれば正常に動作した事になります。

- ようこそ！組込みの世界へ！

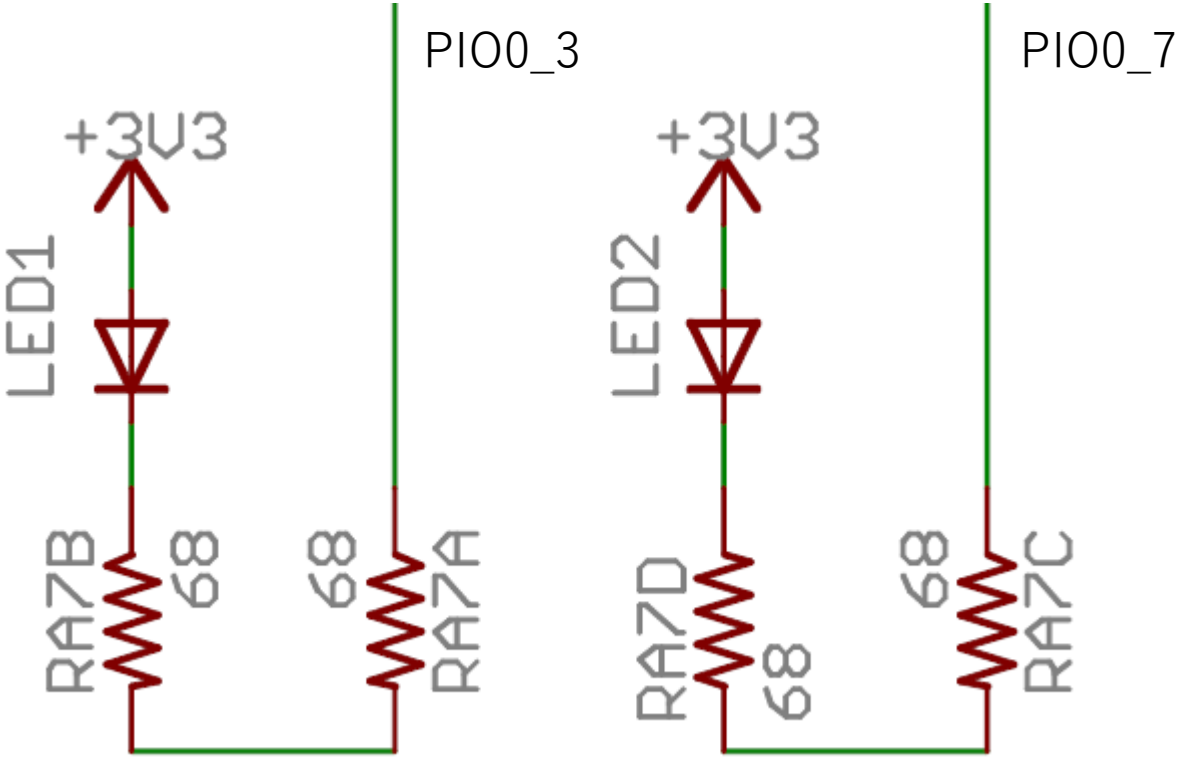
01:GPIOによるLED制御



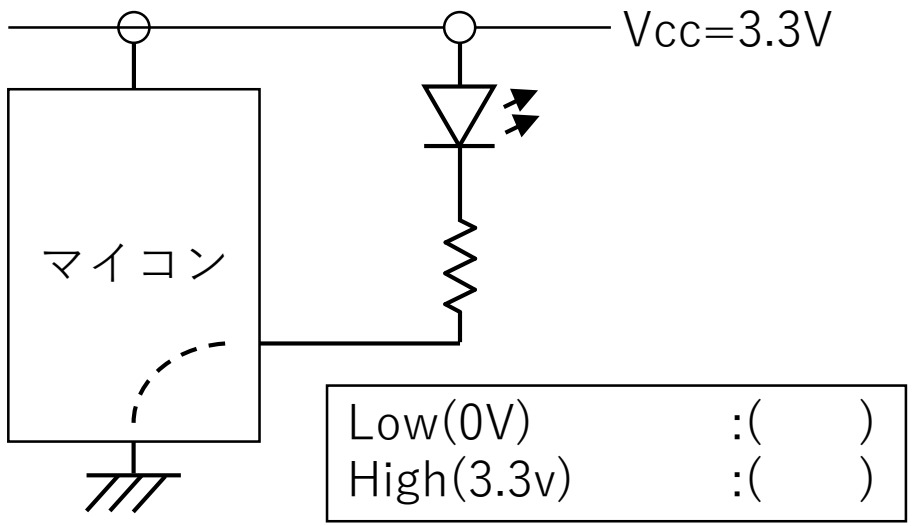
はじめに

- 最初のサンプルプログラムは、マイコンの基本であるLED制御です。
- このLEDと次章で説明するボタンスイッチ、スライドスイッチなどは全てマイコン内蔵のGPIOと呼ばれるIOポートを利用しています。
- よって、ここではGPIOの基本的な使い方について学習します。
- GPIOは、ほとんどのマイコンに付属する最も基本的な機能になります。それゆえに、LEDをはじめ多くのハードウェアの制御に使用されています。

回路図



- GPIOによるLED回路
  - LED1：緑のLED
  - LED2：オレンジのLED
- シンク電流による点灯制御



- LEDに流れる電流値を計算
  - V=R*I → 3.3[V]-2[V]=136[Ω]+I[A]
  - I=(3.3-2)/136=9.55…[mA]
  - LPC1343のシンク電流の最大値
    - 1pinあたり：4mA以下
      - データシートのIOH=-4mAより
    - しかし、IOHS=45mAである為、非保障ながら最大で45mAまで可

回路の簡単な説明

- 基板上には緑と橙(オレンジ)の2つのLEDが搭載されており、緑LEDはPIO0_3に、橙LEDはPIO0_7に繋がっています。
- ここでは、シンク電流によるLED点灯回路が構成されており、IOポートから'0'を出力するとLEDの両端に3.3Vの電位差が発生し、LEDが点灯、逆に'1'を出力するとLEDの両端に電位差が生じずLEDは消灯する仕組みになっています。
  - 緑LEDを点灯させるには、_____から_____を出力する
  - 緑LEDを消灯させるには、_____から_____を出力する
  - 橙LEDを点灯させるには、_____から_____を出力する
  - 橙LEDを消灯させるには、_____から_____を出力する

GPIOとは

- GPIOはGeneral Purpose I/Oの略です。
- GPIOは外部に対して、’1’か’0’の信号を出力する事ができます。
- LPC1343マイコンには、12bitのポートが3セットと4bitのポートが1セット内蔵されており、それぞれPIO0, PIO1, PIO2, PIO3と名前が付けられています。
- またポートをビットごとに表現するにはポート1のビット0の場合、PIO1_0と表します。
- 同じLPC1343マイコンでも、ピン数の少ないHVQFN33パッケージの場合、PIO2およびPIO3は1bitのポート割付になる為、注意が必要です。

Table 146. GPIO configuration

Part	Package	GPIO port 0	GPIO port 1	GPIO port 2	GPIO port 3	Total GPIO pins
LPC1311, LPC1311/01	HVQFN33	PIO0_0 to PIO0_11	PIO1_0 to PIO1_11	PIO2_0	PIO3_2; PIO3_4; PIO3_5	28
LPC1313, LPC1313/01	LQFP48	PIO0_0 to PIO0_11	PIO1_0 to PIO1_11	PIO2_0 to PIO2_11	PIO3_0 to PIO3_5	42
LPC1313	HVQFN33	PIO0_0 to PIO0_11	PIO1_0 to PIO1_11	PIO2_0	PIO3_2; PIO3_4; PIO3_5	28
LPC1342	LQFP48	PIO0_0 to PIO0_11	PIO1_0 to PIO1_11	PIO2_0 to PIO2_11	PIO3_0 to PIO3_3	40
	HVQFN33	PIO0_0 to PIO0_11	PIO1_0 to PIO1_11	PIO2_0	PIO3_2	26
LPC1343	LQFP48	PIO0_0 to PIO0_11	PIO1_0 to PIO1_11	PIO2_0 to PIO2_11	PIO3_0 to PIO3_3	40
	HVQFN33	PIO0_0 to PIO0_11	PIO1_0 to PIO1_11	PIO2_0	PIO3_2	26

GPIOを扱う時の注意点

- 注意点としては、これらのポートのほとんどが他の機能の端子と兼用になっている点です。
- マイコンの回路設計をする際には、ADやPWMなど数の少ない有用な機能を優先的にセンサやアクチュエータに割り当てて、GPIOなどは後の方で残ったものを利用する傾向があります。

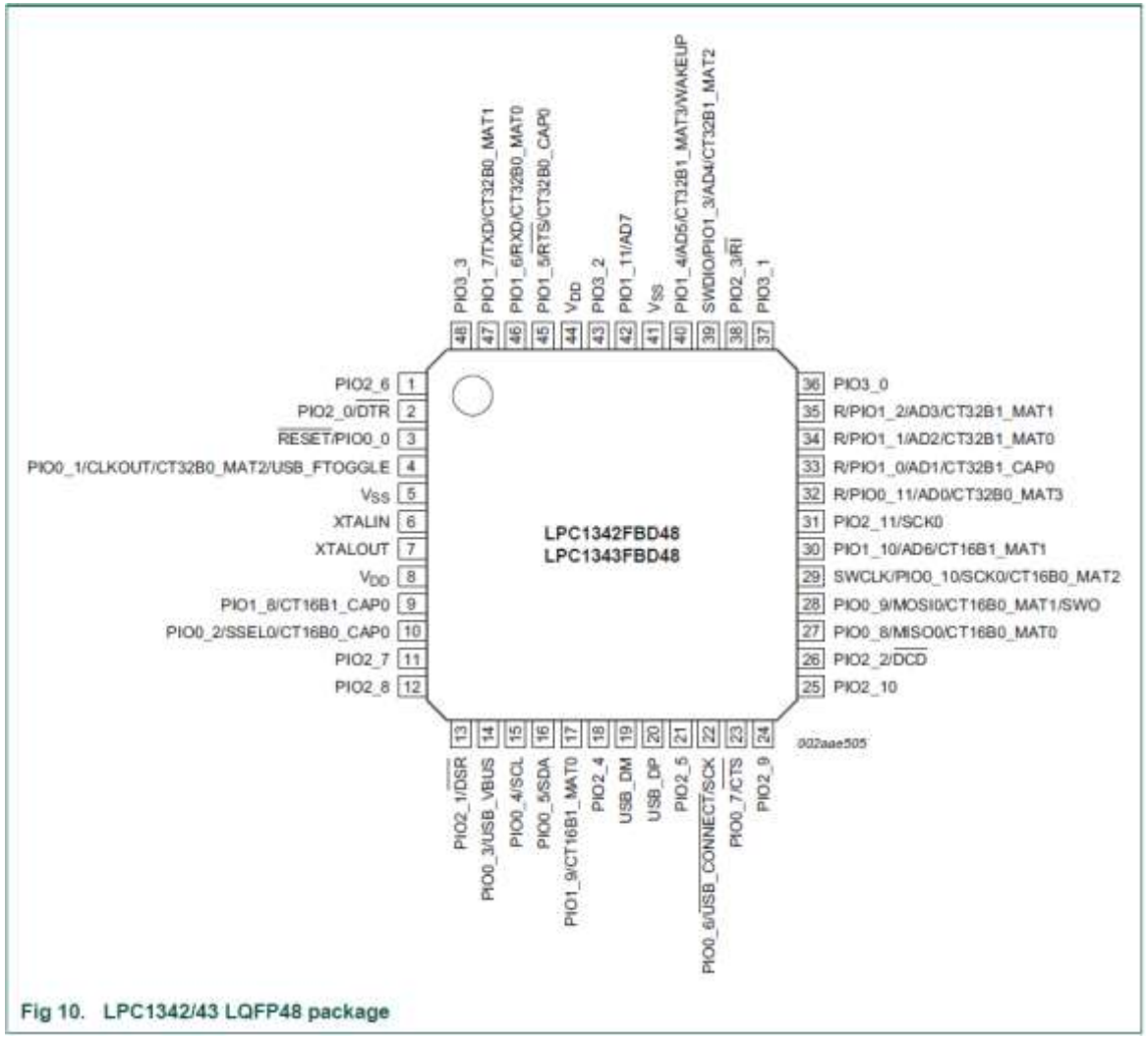


Fig 10. LPC1342/43 LQFP48 package

ユーザマニュアルの参照

- GPIOを制御するプログラムを作成する場合は、ユーザマニュアルの「Chapter 9: LPC13xx General Purpose I/O (GPIO)」を参照します

PIO0のレジスタ構成

名称	アドレス オフセット	R/W	初期値
GPIODIR	0x8000	R/W	0x00
GPIODATA	0x3FFC	R/W	n/a

[base address]
port 0: 0x5000 0000
port 1: 0x5001 0000
port 2: 0x5002 0000
port 3: 0x5003 0000

- ここでいうレジスタとは、汎用レジスタとは異なり、GPIOを制御する為の機能が割りついた専用のレジスタとなります。
- 他のADコンバータやPWMといった機能も、全て制御レジスタが用意されており、このレジスタに値を書き込んだり、逆に値を読み込むことで制御を行います。
- マイコンを制御する場合、レジスタの制御を覚える事は非常に重要なポイントとなります。
- GPIOの各レジスタは、base address + アドレスオフセットで算出が可能です。
  - PIO0のDIRレジスタのアドレス　：0x5000 0000 + 0x8000 = 0x5000 8000
  - PIO0のDATAレジスタのアドレス：0x5000 0000 + 0x3FFC = 0x5000 3FFC

GPIODIR (GPIO data direction register)

- 該当のポートの各ビット(各端子)が入力ポートであるか、出力ポートであるかを設定します。
- 対応するビット、それぞれにバラバラの設定が可能
  - ’0’の場合は入力ポート
  - ’1’の場合は出力ポート
- 初期値は0x00である為、電源投入時は入力ポートの設定。

GPIODATA(GPIO data register)

- 出力ポートの場合
  - 該当のポートの本レジスタに値を書き込むことで、値によってビットが立ったポート端子PIOn_0～11に出力されます。
- 入力ポートの場合
  - 該当のポートの本レジスタの値を読み込むことで、ポート端子の状態を取得する事が出来ます。

例題:test01.c

```
#ifdef __USE_CMSIS
#include "LPC13xx.h"
#endif

#include <cr_section_macros.h>

// レジスタ定義
#define GPIO0_DIR (*(volatile unsigned int *)0x50008000)
#define GPIO0_DATA(*(volatile unsigned int *)0x50003FFC)

// main関数
int main(void) {
    volatile unsigned int wait = 0;// 待ち時間用の変数

    // ポートの入出力設定
    GPIO0_DIR = 0x08;

    // 無限ループ
    while(1) {
        GPIO0_DATA = 0x08;// LEDをOFF
        for(wait = 0; wait < 1000000; wait++);// 待ち
        GPIO0_DATA = 0x00;// LEDをON
        for(wait = 0; wait < 1000000; wait++);// 待ち
    }
    return 0 ;
}
```

- GPIOの制御レジスタの定義
  - #define GPIO0_DIR …
  - #define GPIO_DATA …
- GPIOの制御レジスタへのアクセス

→ 制御レジスタに対して変数のようにアクセスする
  - GPIO0_DIR = 0x08;
  - GPIO_DATA = 0x08;
- 待ち時間の作成→空ループ
  - for(…);　　…何もしない処理
- volatile修飾子
  - コンパイラの最適化を防ぐ

CMSIS-COREの活用

- 今後、新たなレジスタを使用する度に、レジスタをdefineで定義する必要があります。レジスタの数は非常に多岐に渡り、define文で定義するだけでも一苦労です。
- ARM Cortex-MシリーズにはCMSIS-COREと呼ばれる便利な規格が存在する
  - #include<LPC13xx.h> の中身を見てみよう！

CMSIS-COREを活用したLチカ

```
// CMSIS-COREを活用した場合
#ifdef __USE_CMSIS
#include "LPC13xx.h"
#endif

#include <cr_section_macros.h>

int main(void) {
    unsigned int wait = 0;

LPC_GPIO0->DIR = 0x08;// ポート初期化

    while(1) {
    LPC_GPIO0->DATA = 0x08;// LEDをOFF
    for(wait = 0; wait < 100000; wait++);// 待ち
    LPC_GPIO0->DATA = 0x00;// LEDをON
    for(wait = 0; wait < 1000000; wait++);// 待ち
    }
    return 0 ;
}
```

- ファイル名：rei01_01.c
- 制御レジスタを構造体を活用し、ペリフェラル別に構造化してのアクセス
  - LPC_GPIO0->DIR = 0x08;
  - LPC_GPIO0->DATA = 0x08;
- LPC13xx.h内に全て記述済み

- ARM Cortex-M間での移植性を高める為の仕組み
- 今後は、CMSIS-COREで定義されたレジスタを活用する形でコードを書いていく。

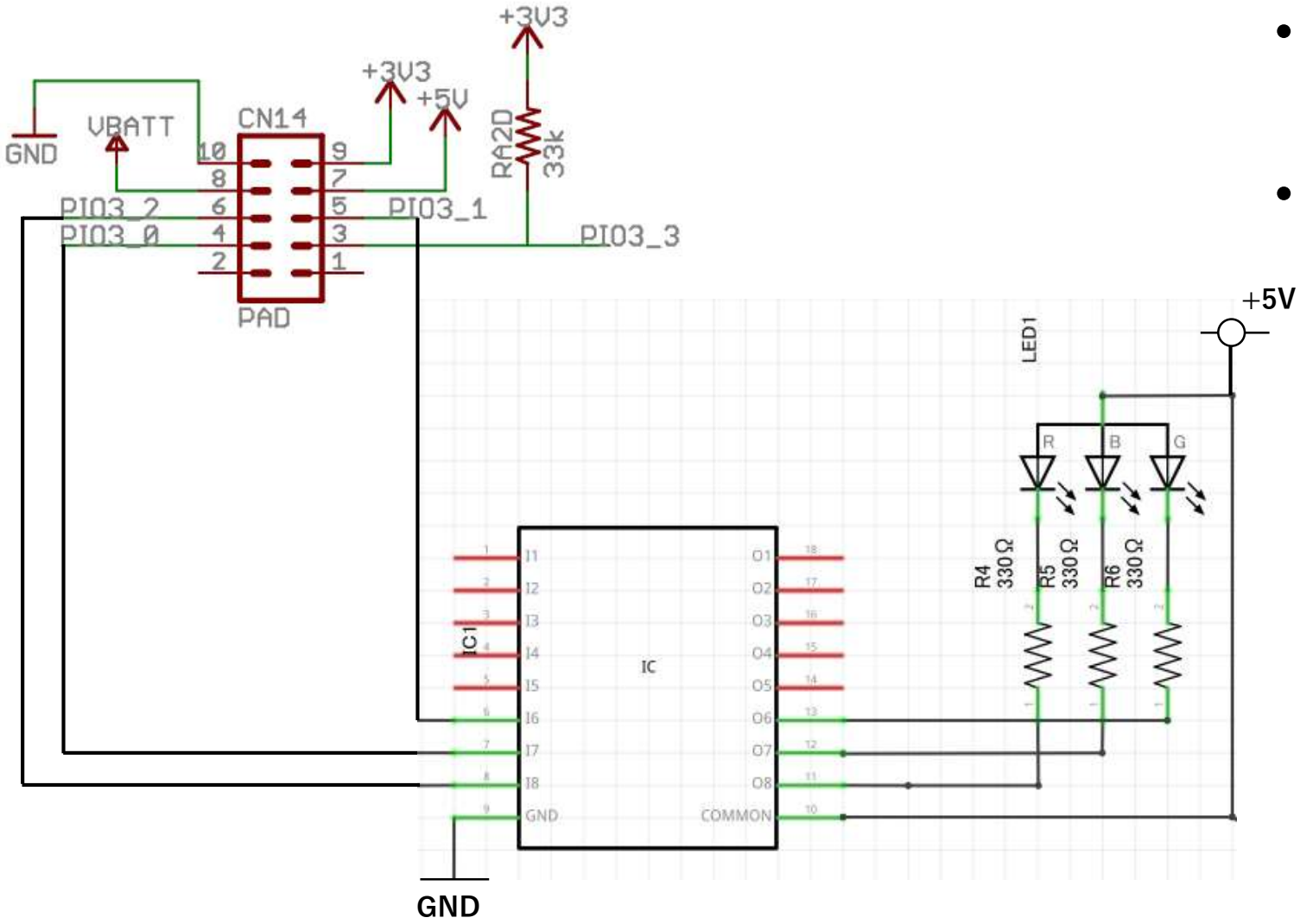
課題

ファイル名は「kadai01_xx.c」とすること。

- 課題01：緑LEDだけを点滅制御するプログラム
- 課題02：橙LEDと緑LEDを**同時**に点滅制御するプログラム
  尚、点滅周期は従来の2倍とする
- 課題03：橙LEDと緑LEDを**交互**に点滅制御するプログラム
- 課題04：LED制御を行う関数を作成
  - void init_led(void);
  - void set_led_orange(unsigned char value);
    - 橙LEDの点灯制御を行う関数
    - 引数が0の時、消灯。1の時、点灯。
  - void set_led_green(unsigned char value);
    - 緑LEDの点灯制御を行う関数
    - 引数が0の時、消灯。1の時、点灯。

ヒント：ビット演算

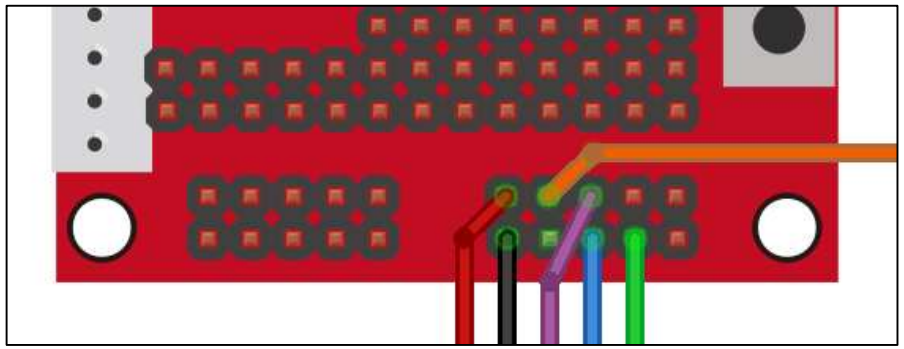
フルカラーLEDを制御しよう！



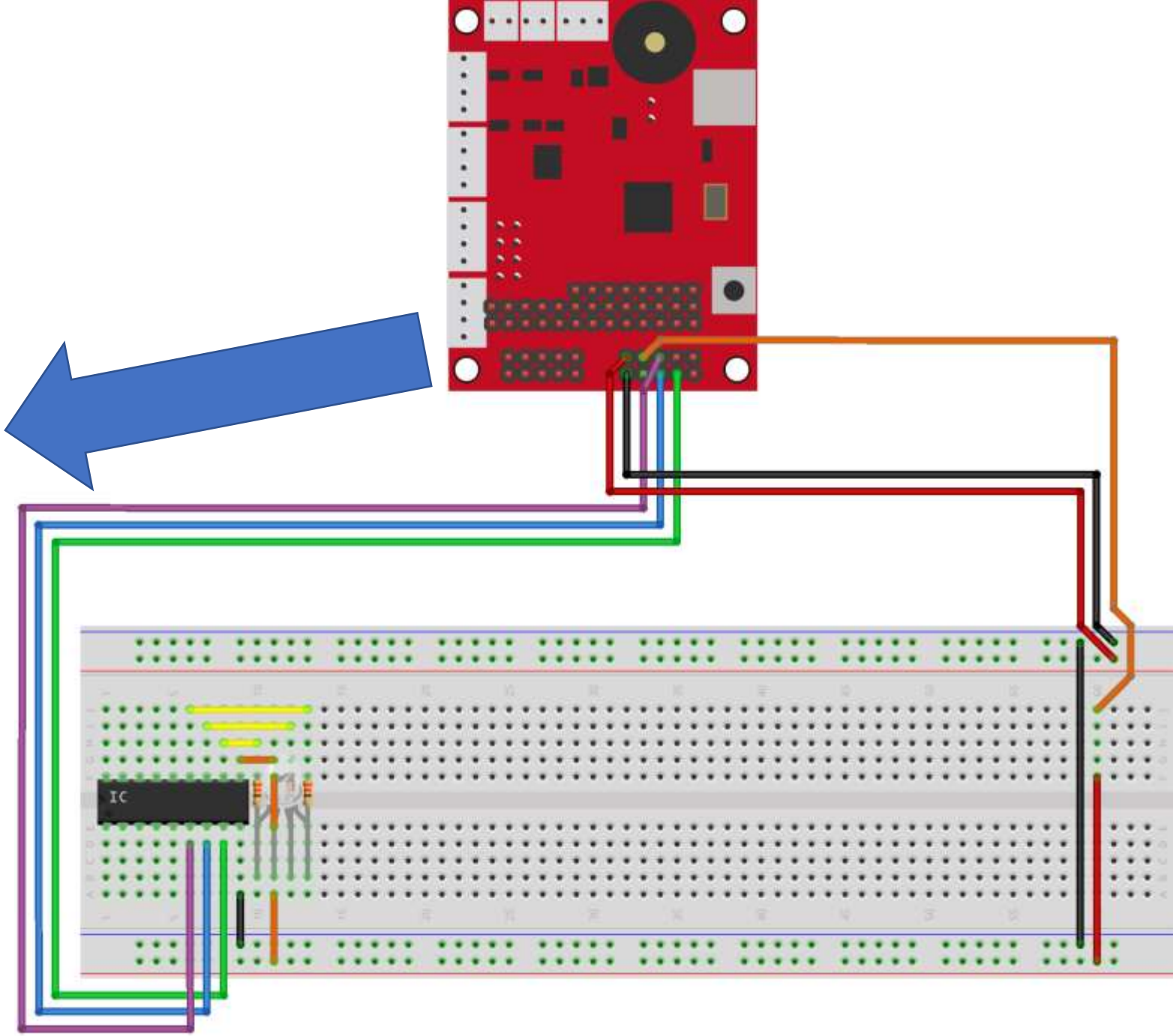
- CN14のPIO3_0～PIO3_2にフルカラーLEDを繋ぎ、点灯制御してみましょう！
- 課題
  - 05:赤→緑→青の順で点灯するプログラムを作成
  - 06:白色に光るプログラムを作成
  - 07:紫→黄→水の順で点灯するプログラムを作成
  - 08:時間があれば…関数化
    - void init_exled(void);
    - void set_exled_full_color(unsigned char color);
      - 引数により…0:消灯、1:赤、2:緑、4:青といった制御を行う

実体配線図

拡大図

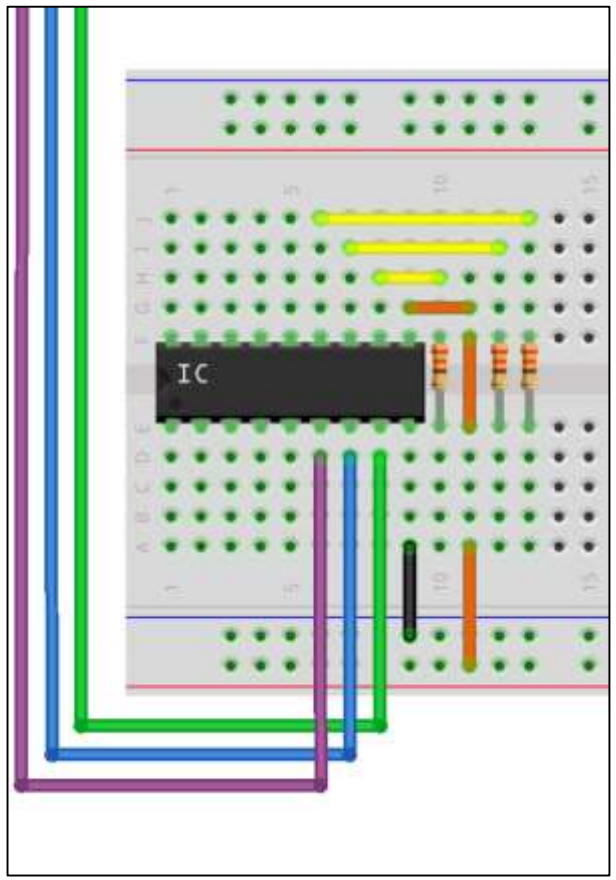


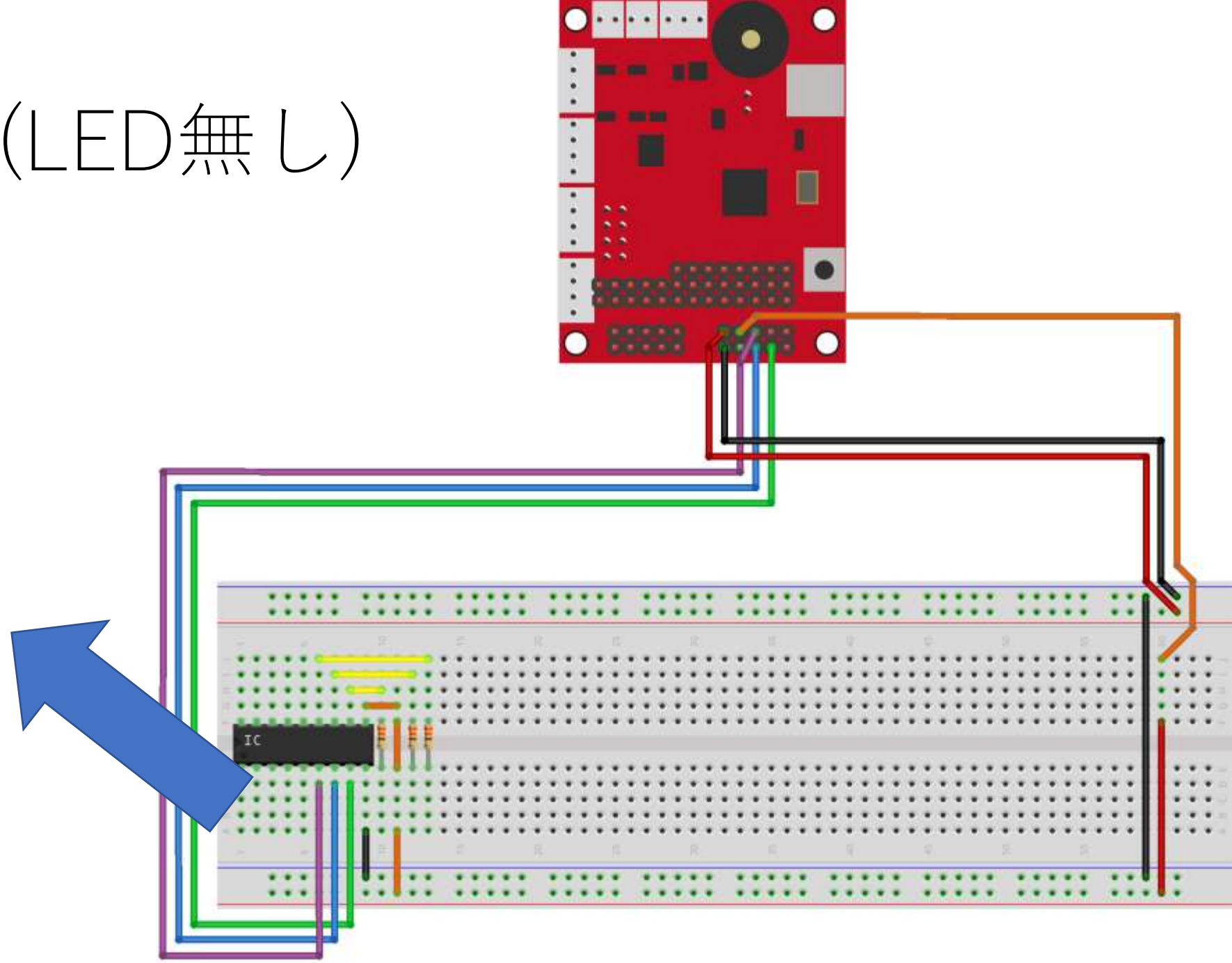
- フルカラーLED＊1
- 330Ω抵抗＊3
- トランジスタアレイ＊1



実体配線図(LED無し)

拡大図

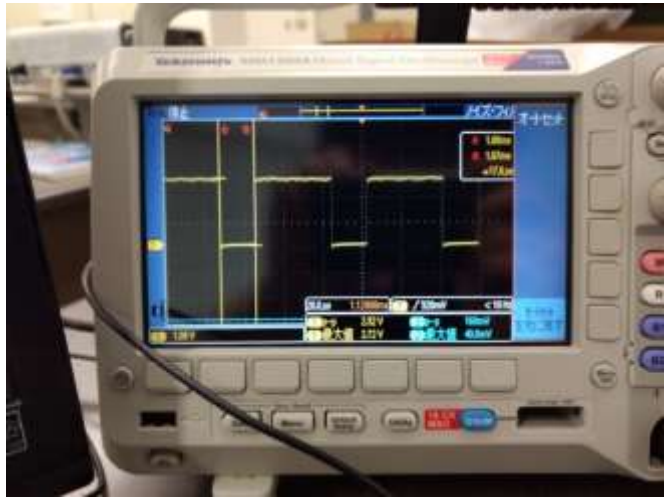




精密な点滅周期を作る

- rei01_01.cなどをベースとして行ってください
- オシロスコープを用いて現在の点滅周期を測定する
- 空ループの回数を調整し、1msのwait関数を作成する
  - ファイル名:kadai01_08.c

```
extern inline void wait_1ms();
inline void wait_1ms(){
        volatile unsigned short i;

        for(i=0; i<_____; i++);
}
```



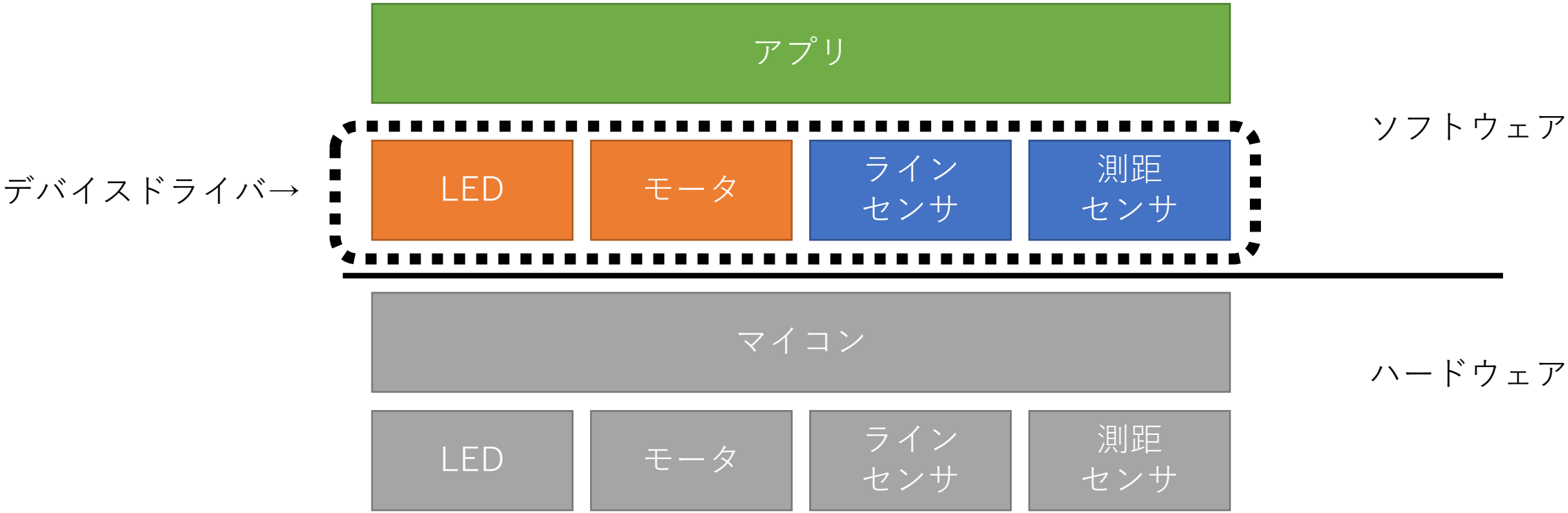
- 1ms以上の任意の時間のwaitする事ができる関数、wait_ms()を作成せよ。Wait時間は引数*1msとする。
  - ファイル名:kadai01_09.c

```
extern inline void wait_ms(unsigned short ms);
inline void wait_ms(unsigned short ms){
        // ココに記述
}
```

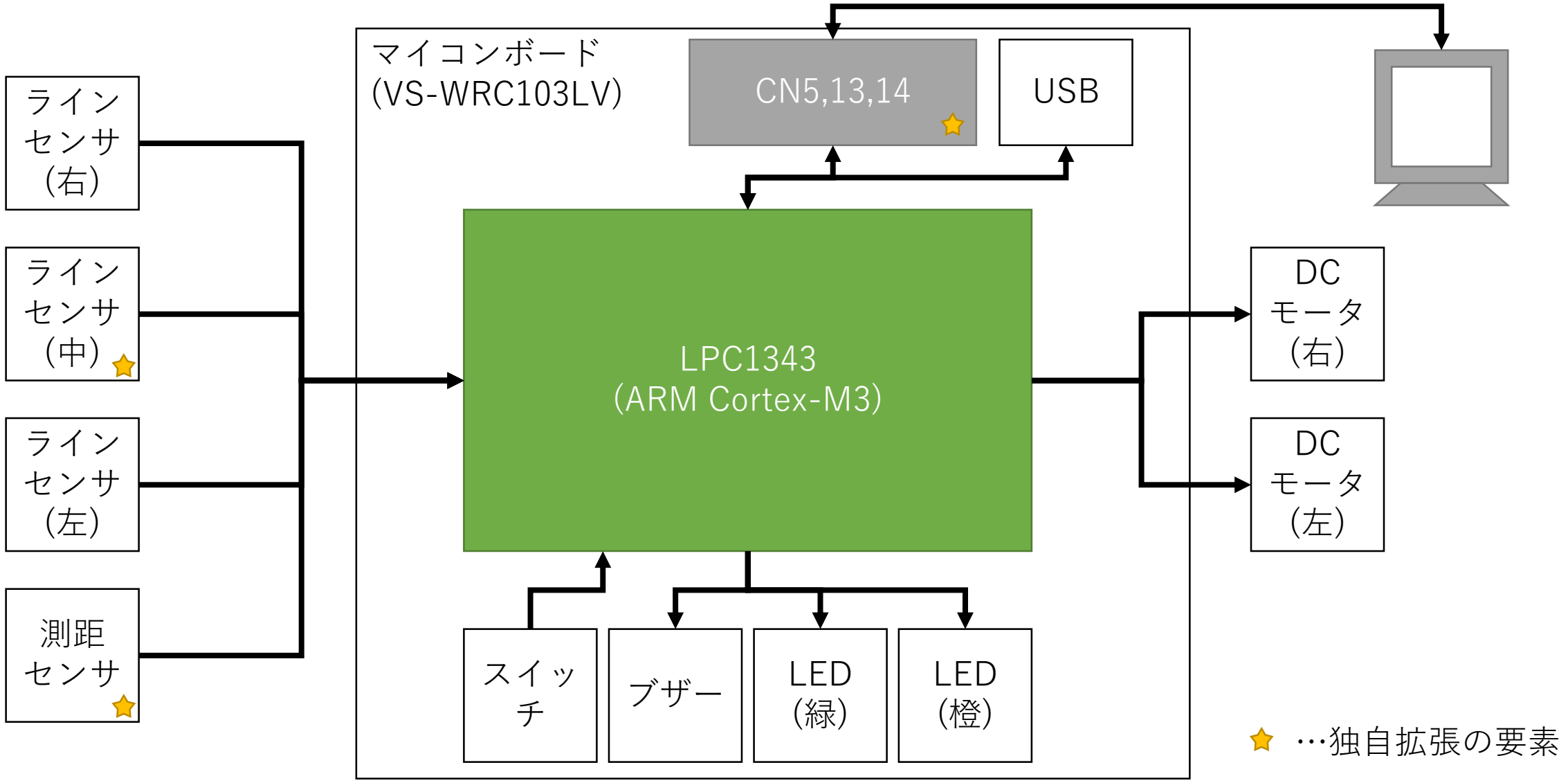
用語：デバイスドライバ

- デバイスドライバとは

デバイスドライバ（略称：ドライバ、ドライバー、デバドラ）とは、画像ディスプレイモニター、プリンターやイーサネットボード、拡張カードやその他周辺機器など、パソコンに接続されているハードウェアなどをOSによって制御可能にするために用意された、ソフトウェアである。
【引用元：Wikipedia】 https://ja.wikipedia.org/wiki/デバイスドライバ

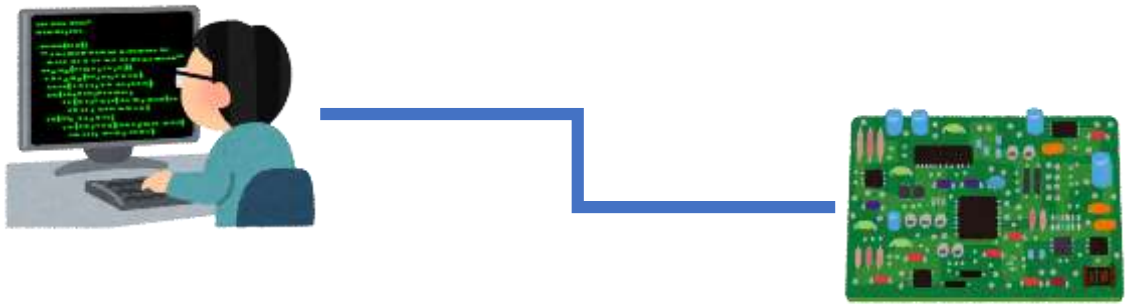


02:シリアル通信によるprintfデバッグ

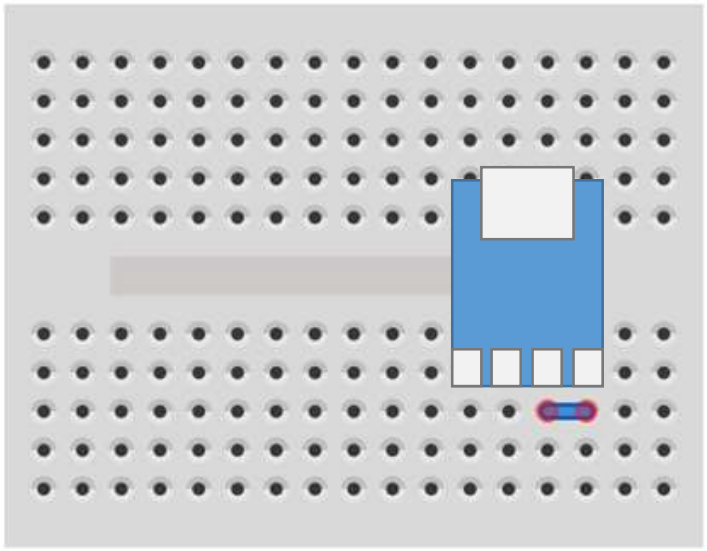


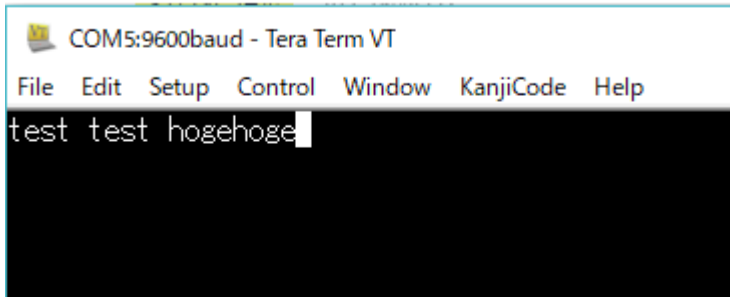
はじめに

- PCやスマホのアプリケーション開発とは異なり、マイコンにはキーボードも無ければ液晶画面も存在しないものが大半である
- そんな中でも、簡単なデバッグ・動作確認を行う為に、シリアル通信を活用し、マイコンに対してデータの入力／出力を行う事が多い。
- 今回は、redlibとシリアル通信用のデバイスドライバを活用し、stdio.hを活用できるようにする
  - つまり、printfやscanfが活用できる！



USB-Serialセットアップ＆動作確認

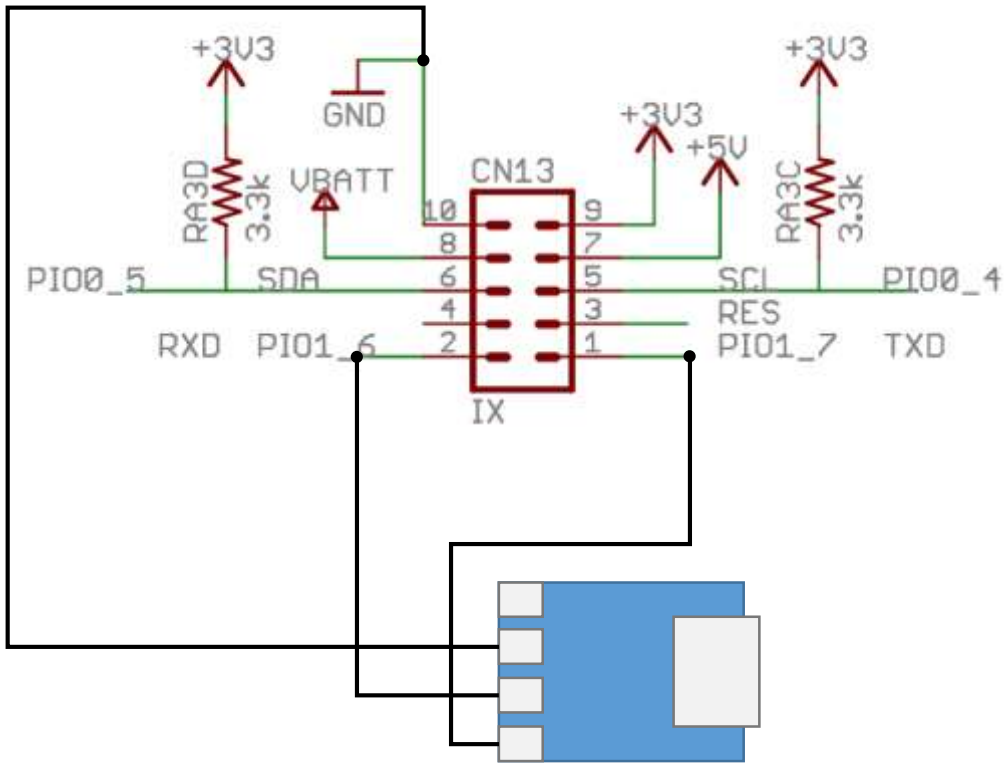


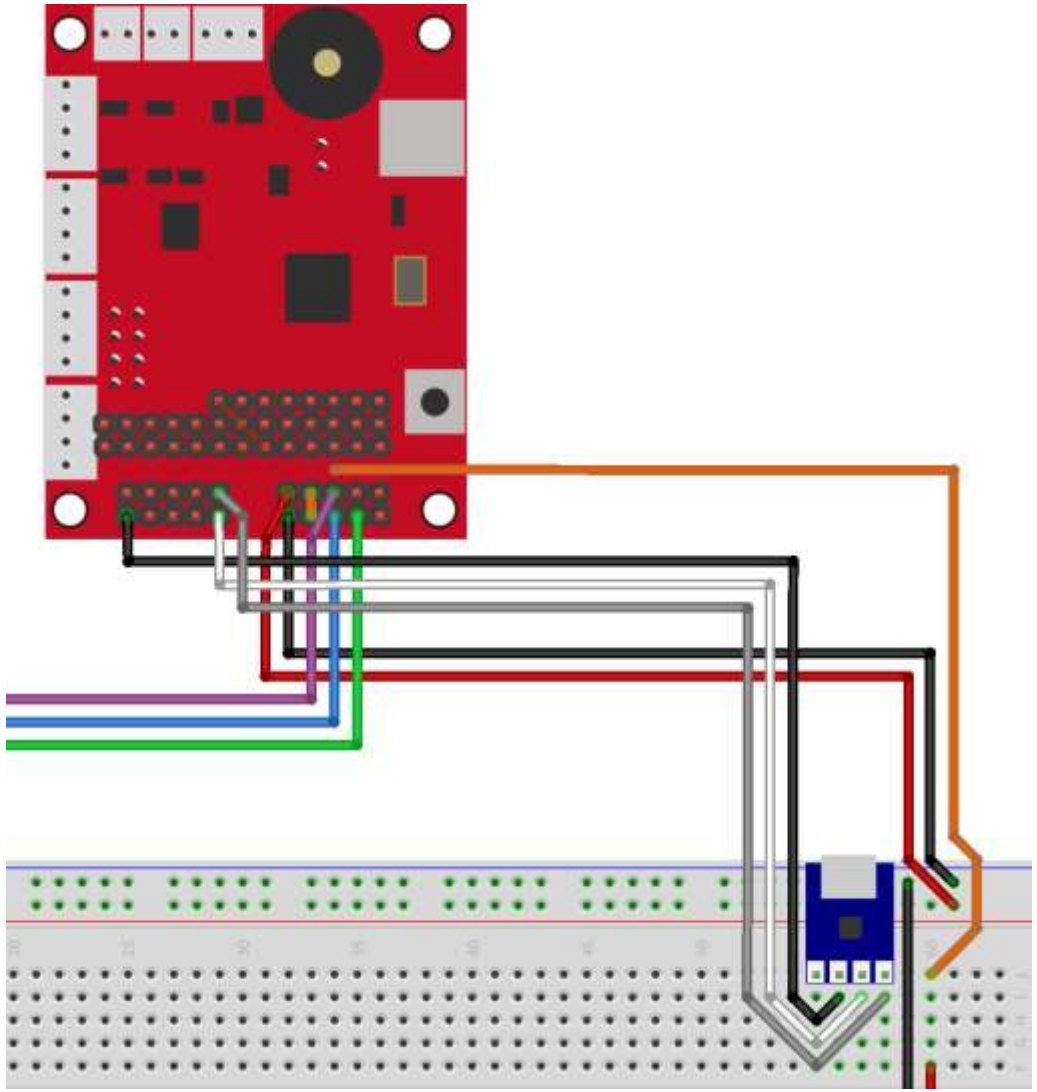


1. USB-Serialの3pin(TXD)と4pin(RXD)をショートさせる
2. USB-SerialとPCを接続
3. (ドライバ導入)
   - 既に導入済みの場合は不要
4. TeraTerm起動
   - シリアル設定はデフォルト
   - ターミナル設定よりLocalEcho：切
5. 何か文字を打ち込みターミナル上に表示されれば成功

回路図＆実装配線図

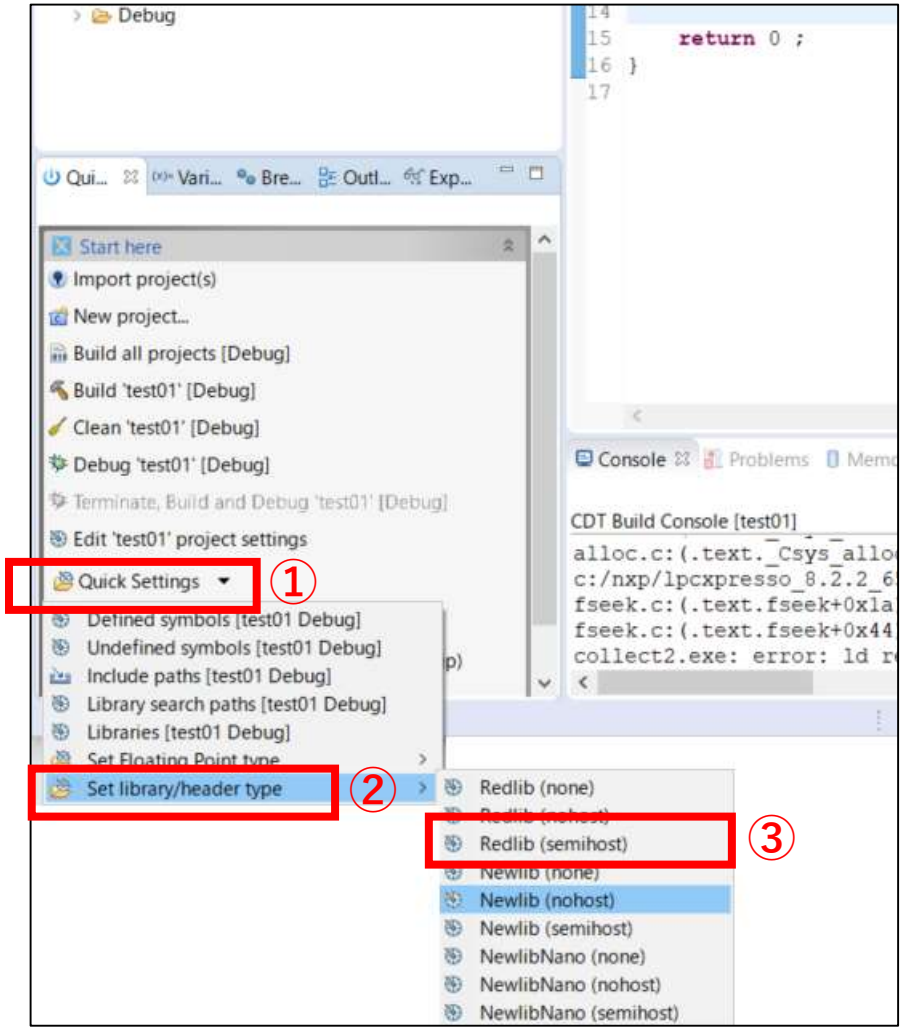
これまでに作った回路に
新たに追加する形で実現すること





- PIO1_6はRxDと兼用ピン
- PIO1_7はTxDと兼用ピン

RedLibとシリアルのデバドラ導入



- シリアル通信のソースコードを共有フォルダからダウンロードする
  - uart.h, uart_io.hをプロジェクトのincに入れる
  - uart.c, uart_io.cをプロジェクトのsrcに入れる
- RedLibを使用する設定を行う
  - Quickstart -> Quick Settings -> Set library/header type -> redlib(semihost) を選択

【RedLib】
CodeRed社が実装した標準Cライブラリ→printfやscanfを提供

【シリアルのデバドラ】
マイコンのメーカーや評価ボードのメーカーの提供コードを利用するか、独自実装する

【システムコール関数】
RedLibのprintfやscanfから呼び出される関数.
シリアルのデバドラとRedLibの橋渡し。自身で実装する

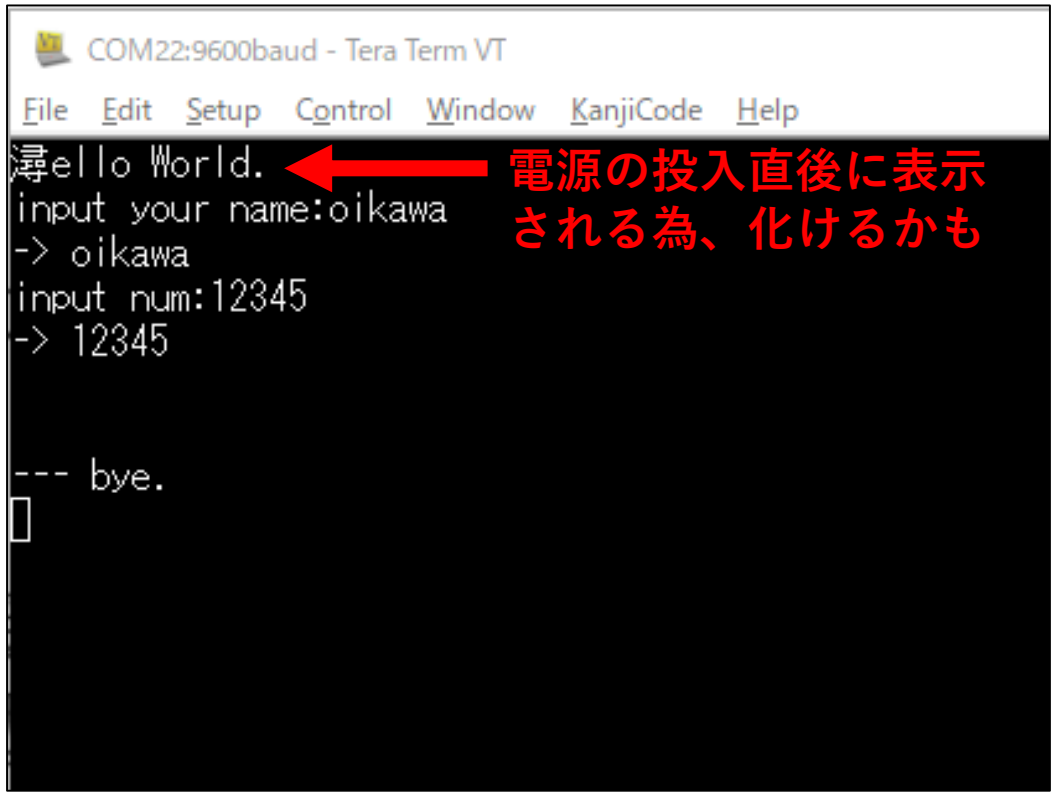
例題:rei02_01.c (ソースコード提供)

```
#ifdef __USE_CMSIS
#include "LPC13xx.h"
#endif

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <cr_section_macros.h>
#include "uart.h"
#include "uart_io.h"

// システムコールの実装
int __sys_write(int iFileHandle, char *pcBuffer, int iLength)
{
    int n;
    for (n = 0; n < iLength; n++) {
        if (pcBuffer[n] == '¥n'){
        uart_putchar('¥r');
        }
        uart_putchar(pcBuffer[n]);
    }
    return iLength;
}

int __sys_readc(){
    return uart_getchar();
}
```

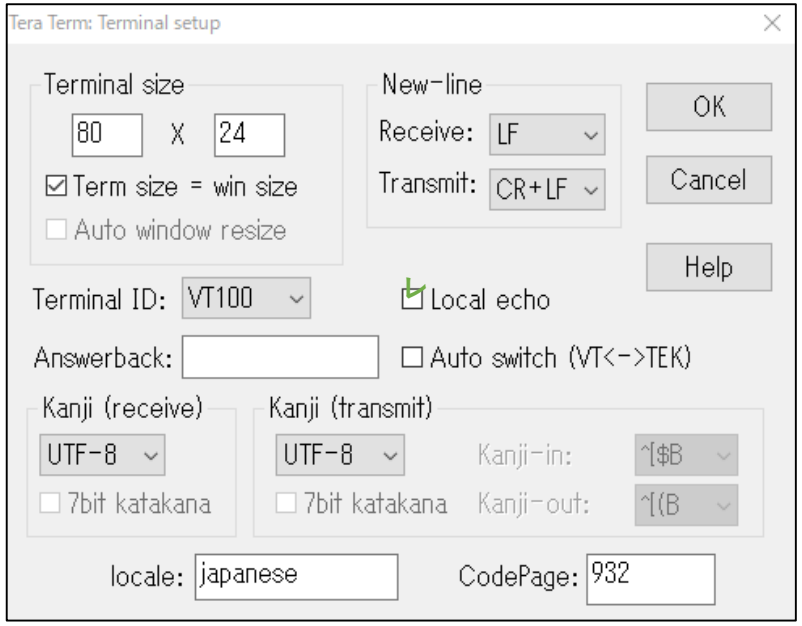
```
// main関数
int main(void) {
    char str[16];
    int a;

    // シリアル通信の初期化
    init_uart_io(9600);

    printf("Hello World.¥n");

    // scanfによる文字列の取得
    printf("input your name:");
    scanf("%s", str);
    printf("-> %s¥n", str);

    // scanfによる整数の取得
    printf("input num:");
    scanf("%d", &a);
    printf("-> %d¥n", a);

    printf("¥n¥n--- bye.¥n");

    return 0 ;
}
```

例題:rei02_01.cの動作確認

TeraTermの表示される結果



TeraTermの設定



課題

ファイル名は「kadai02_xx.c」とすること。

1. printf()関数を用いて、自己紹介をするプログラムを作成してください
   - [出力イメージ]

```
Name:Seigyo Taro
Age:5
From:Kumamoto
```

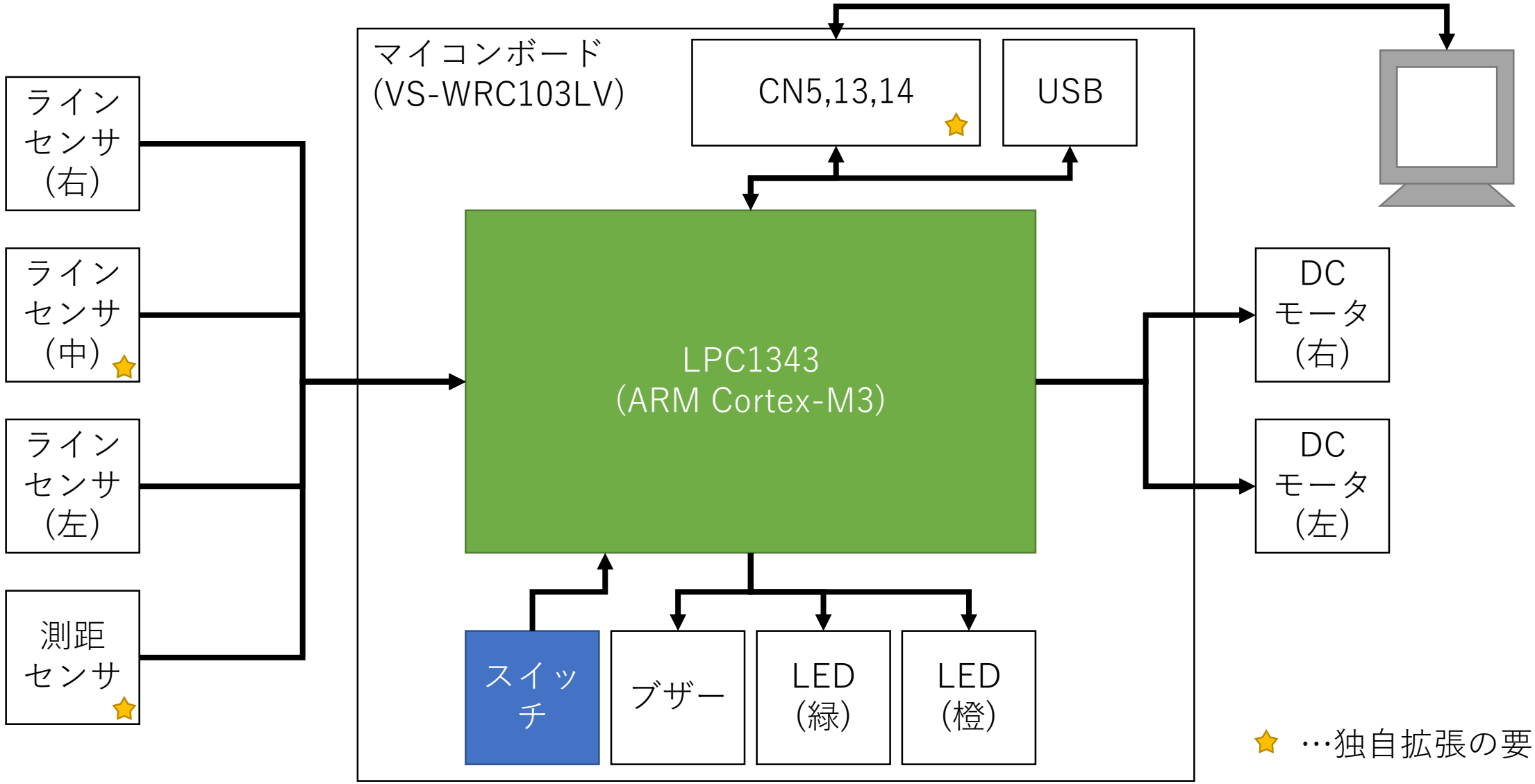
2. シリアル通信でPCより’g’が送られてきたら緑のLEDを点灯するプログラムを作成せよ
3. ’o’が送られてきたらオレンジのLEDを点灯する
4. ’q’が送られてきたらLEDを消灯する

printfデバッグの注意点

- シリアル通信を活用したprintf関数は、他の制御(GPIOやADなど)と比べ非常に遅い。
- シリアル通信を多用すると、システムの動作に大きな影響が出る可能性がある。
  - デバッグ時には使用するがリリース時には，printfの処理を無効化するなどの工夫が必要

```
#ifdef DEBUG
#define debug_printf(fmt, ...) printf(fmt, __VA_ARGS__)
#else
#define debug_printf(...)
#endif
```

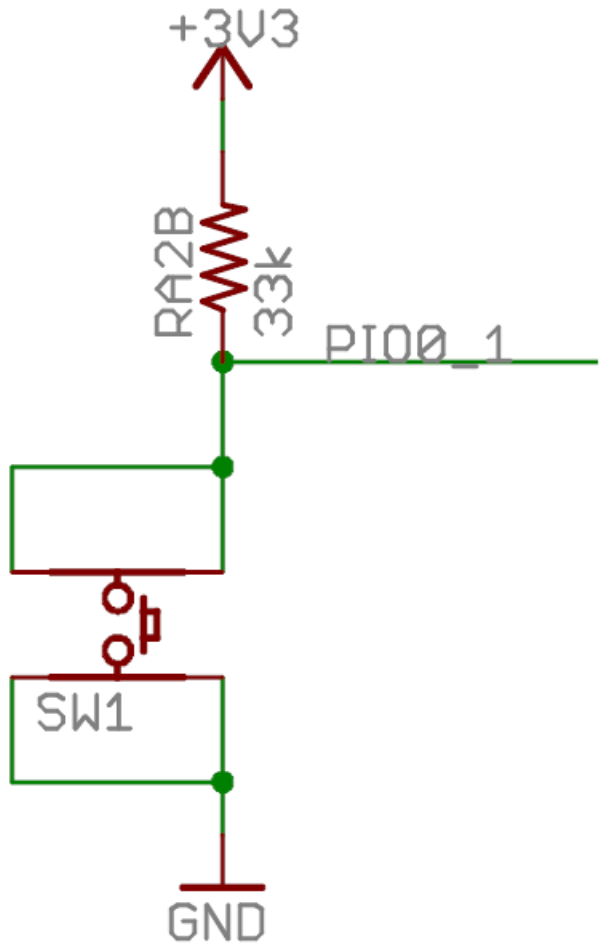
03:GPIOによるスイッチ制御



はじめに

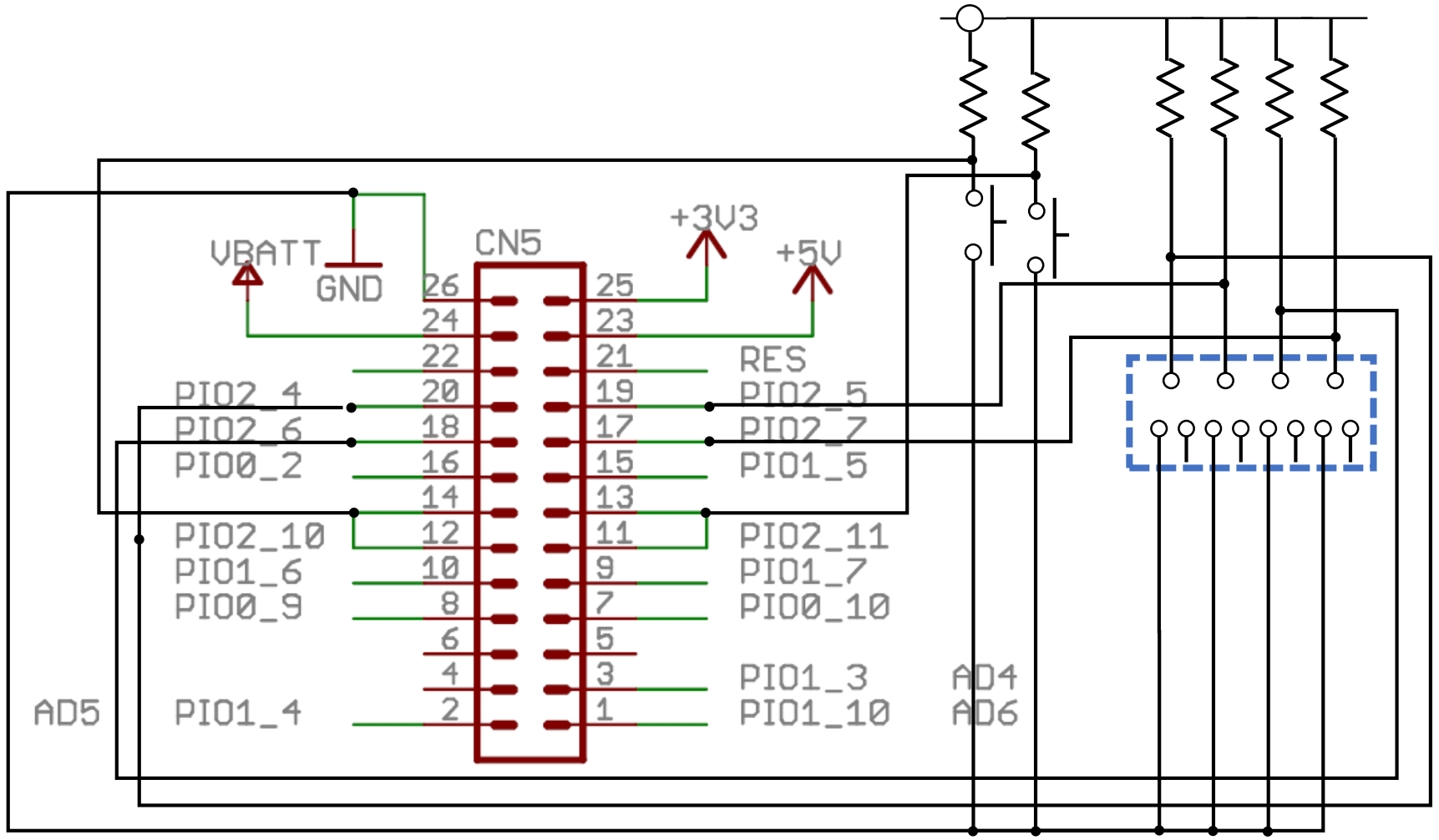
- 前章ではGPIOの出力機能について学びました。この章では、GPIOによる入力機能について学びます。
- GPIOの入力機能も出力機能と同様に汎用性の高い機能になります。
- 今回は、ボタンスイッチとスライドスイッチ、2つのスイッチを扱います。
  - ボタンスイッチについて
    - ボタンを押下中のみONとなり、導通するモーメンタリ方式と、ボタンを押した後に指を離してもON状態が持続するオルタネイト方式の2つ種類があります。
    - モーメンタリ…リモコンやゲームコントローラ、インターフォンなど
    - オルタネイト…生産設備の非常停止ボタンやブラウン管TVなど
      - モーメンタリ方式のスイッチとマイコンを活用して、ソフトウェア的にオルタネイトにする事もあります
  - スライドスイッチについて
    - つまみ部分をずらす事でON/OFFを切り替えます。切替後はON/OFFの状態が常時続きますので、製品のモード設定などによく使われます。

回路図：VS-WRC-103LV



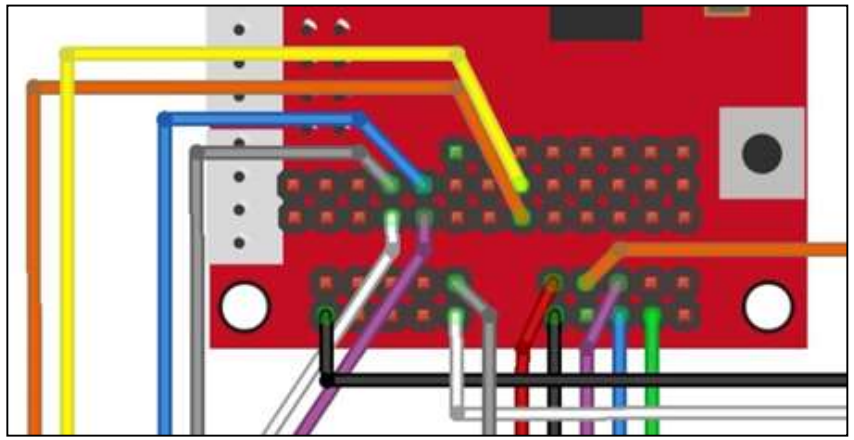
- GPIOによるスイッチ回路
- スイッチのON/OFFとPIO0_1への信号
  - SW1 ON　　:（　　）
  - SW1 OFF　　:（　　）
- チャタリング防止回路は無し
  - 必要があればソフトウェア的に対処する事

追加回路：ブレッドボード

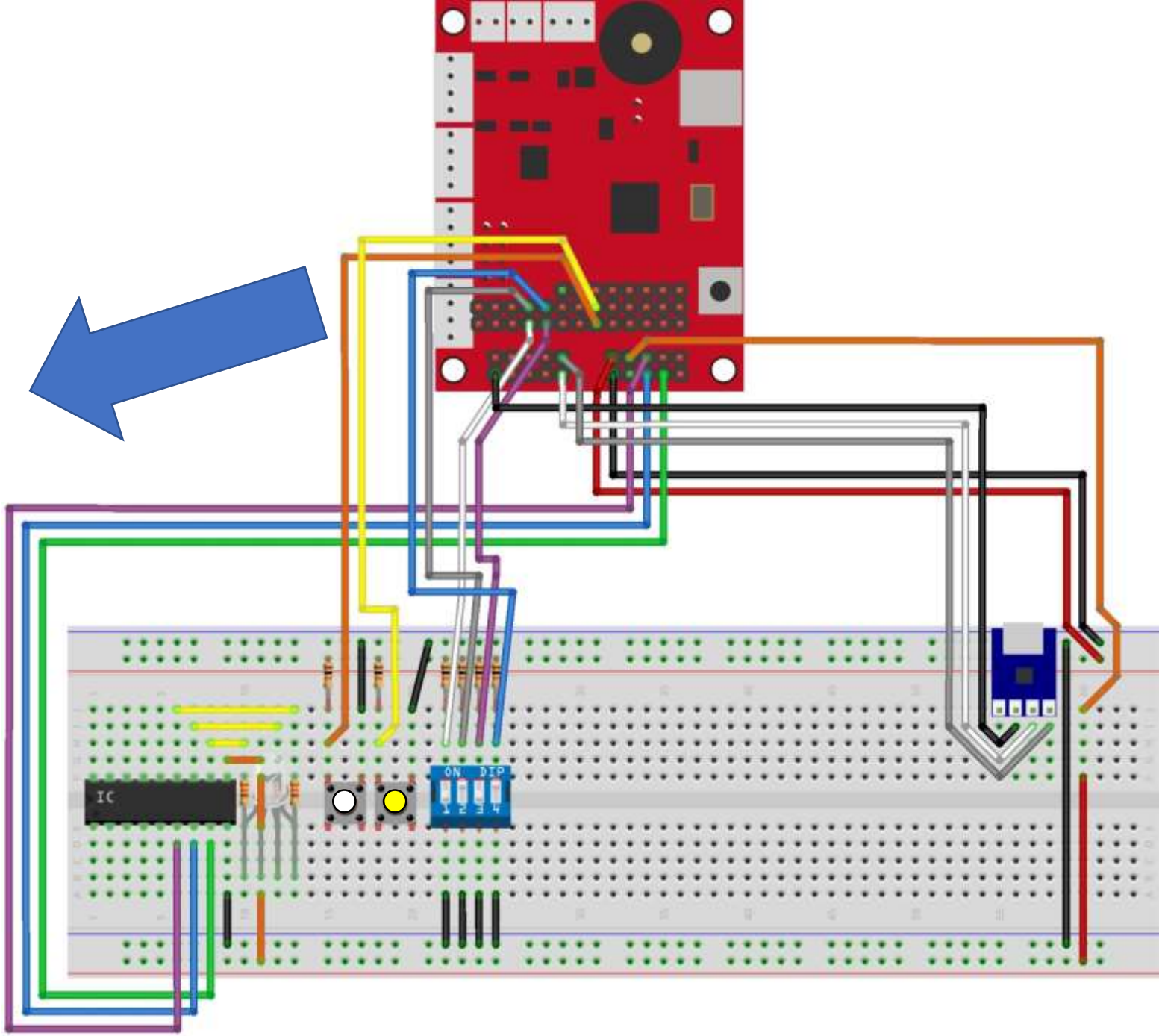


実体配線図

拡大図

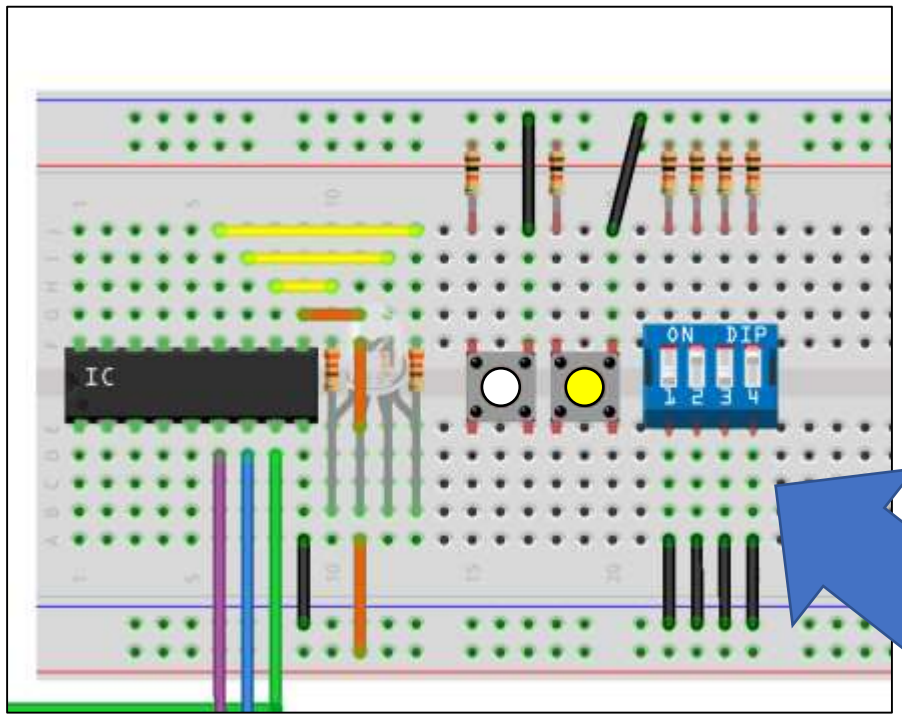


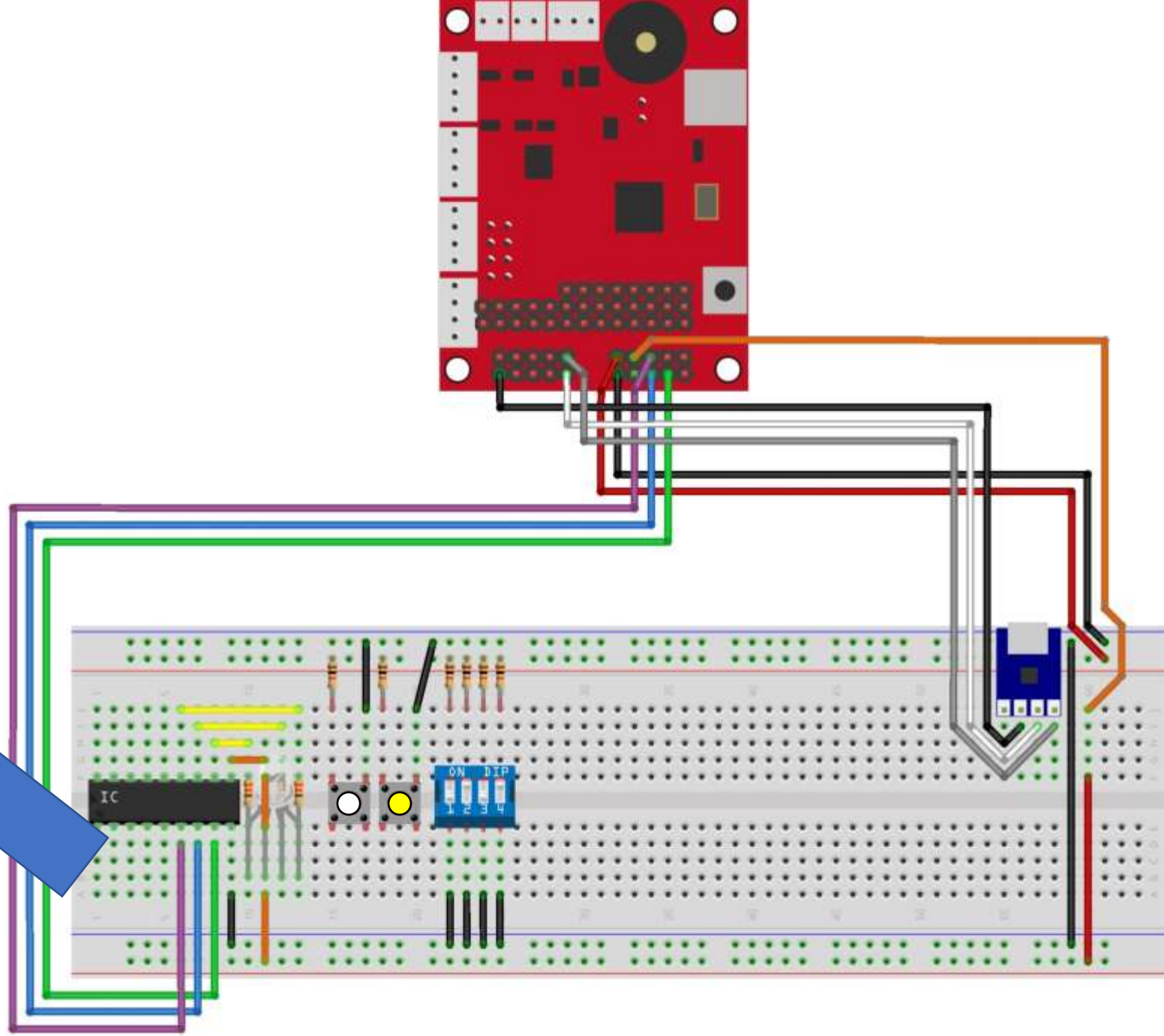
- ボタンスイッチ＊2
  - 黄色・白色
- スライドスイッチ＊1
- 10kΩ抵抗＊6



実体配線図

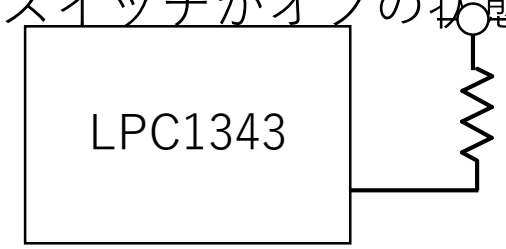
拡大図





回路の簡単な説明

- VS-WRC10LV基板には、ボタンスイッチが1つ実装されており、IOへの割付は以下の通りです。
  - ボタンスイッチ(黒)　　：PIO0_1
- ブレッドボード上には、ボタンスイッチとスライドスイッチが実装されており以下の通りの割付です。
  - ボタンスイッチ(左・白)　　：PIO2_10
  - ボタンスイッチ(右・黄)　　：PIO2_11
  - スライドスイッチ1　　：PIO2_4
  - スライドスイッチ2　　：PIO2_5
  - スライドスイッチ3　　：PIO2_6
  - スライドスイッチ4　　：PIO2_7
- スイッチがオフの状態では、ポートはプルアップされた状態になる為、’1’が入力されます。



- スイッチがオンの状態では、ポートはGNDに接続されますから、’0’が入力されます。

例題:rei03_01.c

```
#ifdef __USE_CMSIS
#include "LPC13xx.h"
#endif

// 過去に作成した関数などは省略

int main(void) {
    init_led();
    LPC_GPIO0->DIR &= ~0x0002;

    while(1) {
        if((LPC_GPIO0->DATA & 0x0002) == 0){
            set_led_orange(LED_ON);
        }else{
            set_led_orange(LED_OFF);
        }
    }
    return 0 ;
}
```

- スイッチ(SW1)を押下している間、橙LEDが点灯するプログラム

- DIRレジスタの設定
  - LPC_GPIO0->DIR &= ~0x0002;
  - GPIO0の2bit目を入力設定に
- DATAレジスタからの読み出し
  - if((LPC_GPIO0->DATA & 0x0002) == 0)
  - GPIO0の2bit目の入力状態のみを抽出
    - SW1が押下であるかを判断

課題

ファイル名は「kadai03_xx.c」とすること。

1. ボタンスイッチ(黒)を押下している間、緑LEDと橙LEDが点灯するプログラムを作成せよ
2. ボタンスイッチ(黒)を押下している間、緑LEDと橙LEDが同時に500msec周期で点滅するプログラムを作成せよ

課題

ファイル名は「kadai03_03.c」とし、順次機能追加して実装すること。

1. スライドスイッチ1～4のON/OFF状態を、随時16進数でTeraterm画面に表示する
2. スライドスイッチ1のON/OFFによりフルカラーLEDを制御する
    1. ON　　：赤→緑→青の順で1秒周期で順次点灯する
    2. OFF　　：青→緑→赤の逆順で1秒周期で順次点灯する
3. スライドスイッチ4のON/OFFにより、順次点灯の開始・停止をする電源ボタンの機能を実装する
4. 各スライドスイッチの入力により、順次点灯の動作が変化するプログラムを作成する

	SSW1	SSW2	SSW3	SSW4
ON	赤→緑→青の 順次点灯		点灯間隔を 500msec	順次点灯の開始
OFF	青→緑→赤の 順次点灯		点灯間隔を 1sec	順次点灯の停止

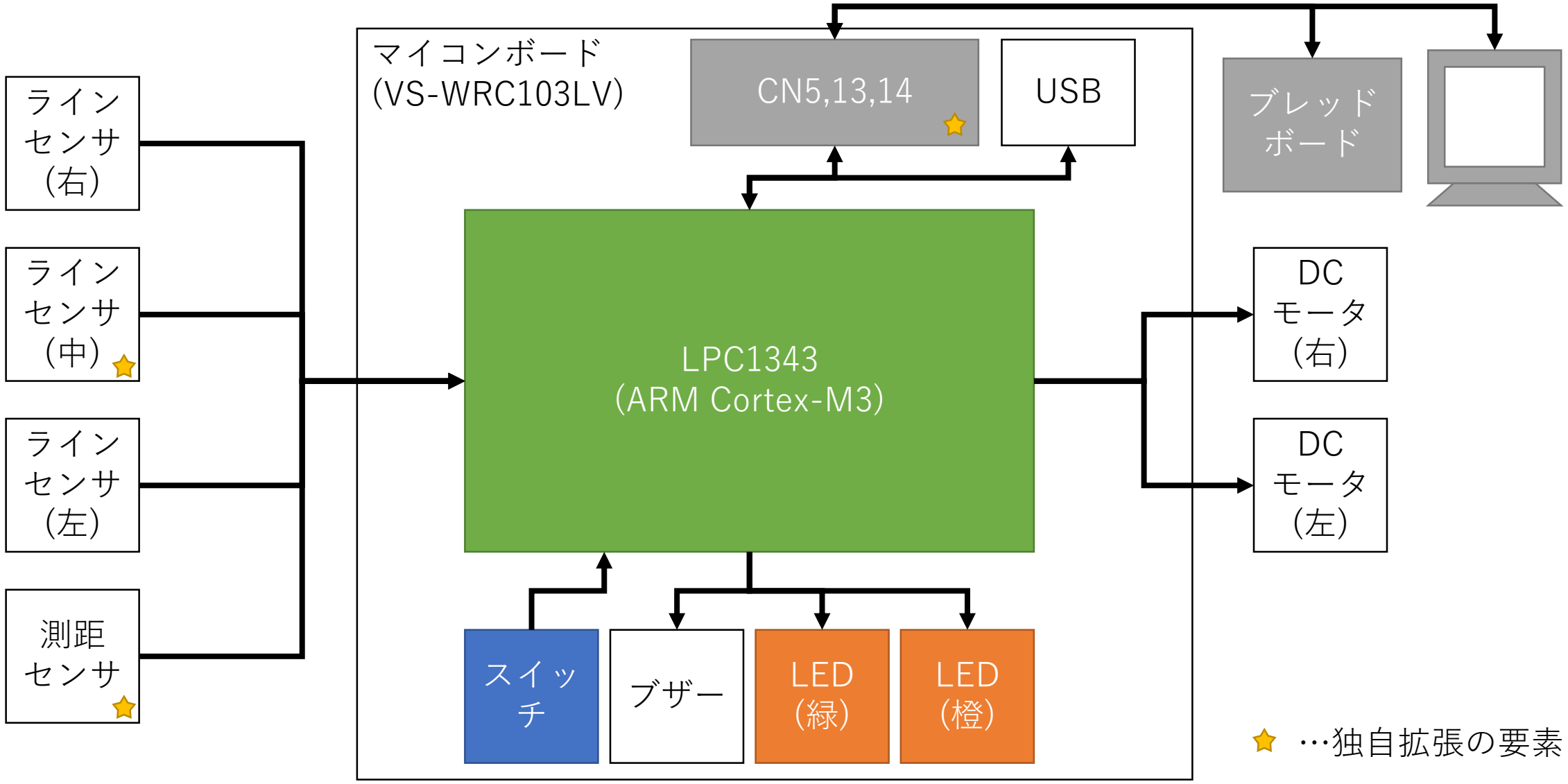
5. [発展]スライドスイッチ2をONにしたら順次点灯のパターンを以下に変更
    1. 赤→黄→青→消灯→紫→水の順で順次点灯する
    2. 5.1の逆順で順次点灯する

事例紹介：ボタン押下後に動作したLTC

```
// main関数
int main(void) {
// 初期化
init_uart_io(9600);
    init_led();
    LPC_GPIO0->DIR &= ~0x0002;

    // 開始時のボタン押下待ち
    set_led_orange(LED_ON);
    set_led_green(LED_ON);
    while((LPC_GPIO0->DATA & 0x0002) != 0);// 押下待ち
    while((LPC_GPIO0->DATA & 0x0002) == 0);// 指を離す待ち
    set_led_orange(LED_OFF);
    set_led_green(LED_OFF);

    // printf表示
    printf("Name:Kumikomi Taro¥n");
    printf("Age:5¥n");
    printf("From:Kumamoto¥n");

    return 0 ;
}
```

- ビュートビルダー2でビジュアルプログラミングしていた際には、以下のフローで動作をしていた
  1. 電源投入
  2. 橙・緑LED点灯
  3. ボタンスイッチ(黒)押下
  4. 橙・緑LED消灯
  5. 書き込んだプログラムが実行
- この動作は左のコードのように実装できる
  - kadai02_01.cに機能追加した事例

04:ファイル分割によるソースコード管理



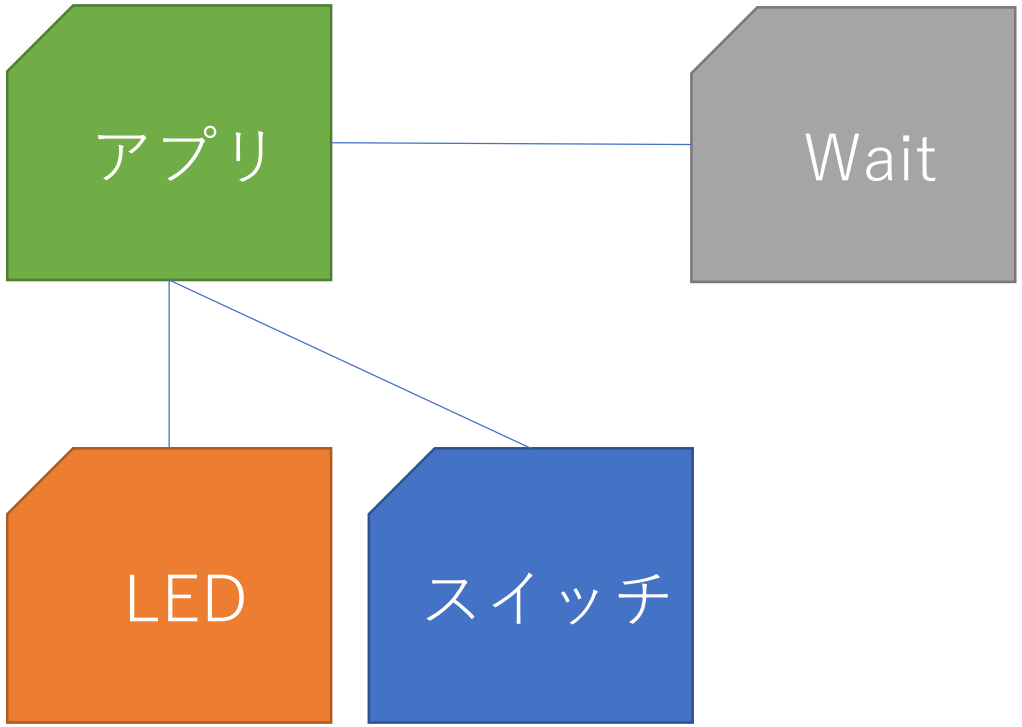
関数が増えてcファイルが長い問題

```c
// 課題03_03：発展課題も実装

#ifdef __USE_CMSIS
#include "LPC13xx.h"
#endif

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <cr_section_macros.h>
#include "uart.h"
#include "uart_io.h"

//-----------------------------------------------------------------
// 定数・グローバル変数定義
//-----------------------------------------------------------------
// 基板上のLED制御向けの定数
#define LED_OFF0// 消灯
#define LED_ON1// 点灯

// ブレッドボード上のフルカラーLED制御向けの定数
#define LED_BLANK0x00// 消灯
#define LED_RED0x01// 赤色
#define LED_GREEN0x02// 緑色
#define LED_BLUE0x04// 青色
#defineLED_PURPLE(LED_RED | LED_BLUE)// 紫色
#defineLED_YELLOW(LED_RED | LED_GREEN)// 黄色
#defineLED_LIGHT_BLUE(LED_BLUE | LED_GREEN)// 水色
#defineLED_WHITE(LED_RED | LED_BLUE | LED_GREEN)// 白色


//-----------------------------------------------------------------
// プロトタイプ宣言
//-----------------------------------------------------------------
// LED制御関数群
void init_led(void);
void set_led_orange(unsigned char value);
void set_led_green(unsigned char value);
void init_ex_led_full_color(void);
void set_ex_led_full_color(unsigned char value);

// Wait関数群
extern inline void wait_1ms(void);
extern inline void wait_ms(unsigned short ms);
```

```c
// スイッチ制御関数群
void init_button(void);
unsigned char get_button_black(void);
void init_ex_slide_switch(void);
unsigned char get_ex_slide_switch(void);
unsigned char is_ex_slide_switch1_on(void);
unsigned char is_ex_slide_switch2_on(void);
unsigned char is_ex_slide_switch3_on(void);
unsigned char is_ex_slide_switch4_on(void);


//-----------------
// メイン関数
//-----------------
int main(void) {
unsigned char i = 1
unsigned int msec =
unsigned char power
const unsigned char
LED_RED, LED_YELLOW
};// 赤・黄・青・消灯
unsigned char flash

init_uart_io(9600);
    init_led();
    init_ex_led_ful
    init_button();
    init_ex_slide_s

    set_led_orange(
    while(1) {
    // ----------
        // 課題03_01
        set_led_green(LED_ON);
    printf("sw1:%02X, ", get_button_black());
    printf("ssw:%02X¥n ", get_ex_slide_switch());
    printf(" ssw1:%x, ", is_ex_slide_switch1_on());
    printf(" ssw2:%x, ", is_ex_slide_switch2_on());
    printf(" ssw3:%x, ", is_ex_slide_switch3_on());
    printf(" ssw4:%x¥n", is_ex_slide_switch4_on());
    printf("------¥n");
        set_led_green(LED_OFF);
```

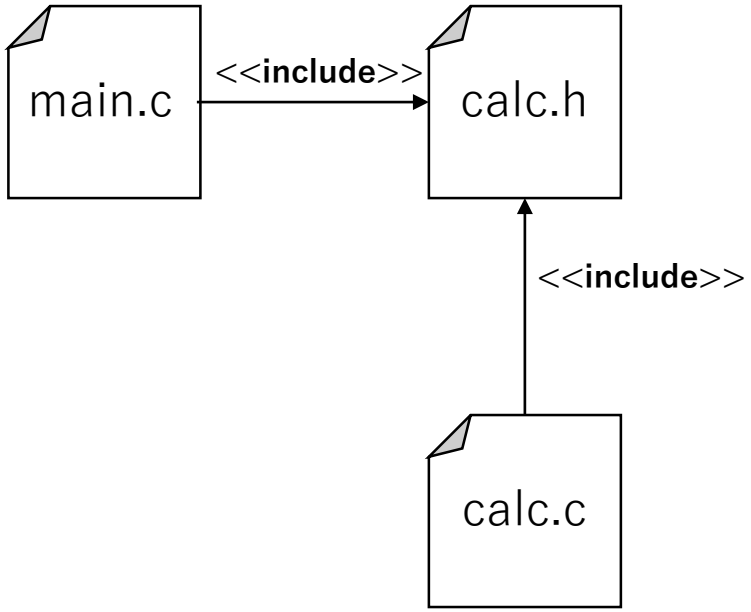
- kadai03_03_ex.cで240行ほどのコードの長さがある
- まだ、LED/printf/Switchのみであり、今後まだまだ増える…!!
- 既に可読性が落ちている

ファイル分割



- 1ファイルのコード量が増えると可読性が落ち、メンテナンスし辛い。追加で開発し辛い。といった側面が現れる
- 対応策としては、コードを複数のファイルに分割し、分散管理する手法
  - チーム開発時にも有効であり、1つのファイルを複数名で同時に編集する場面を避けられる
  - アプリ・システム全体が見通しやすくなる

C言語におけるファイル分割



- C言語の場合、ヘッダファイル(hoge.h)とソースファイル(hoge.c)をワンセットに分割するスタイルが慣習的にある
  - 例：
- ヘッダファイル
  - 定数やグローバル変数、プロトタイプ宣言を記述する
  - includeガードを記述する
- ソースファイル
  - 関数の実体などを記述

C言語におけるファイル分割②

main.c

```
#include <stdio.h>
#include <calc.h>

int main(void){
        int a;
        a = add(1, 2);
        printf("ans:%d¥n", a);

        return 0;
}
```

関数の呼び出し

calc.h

```
#ifdef __CALC_H
#define __CALC_H

int add(int  x, int y);

#endif
```

Includeガード

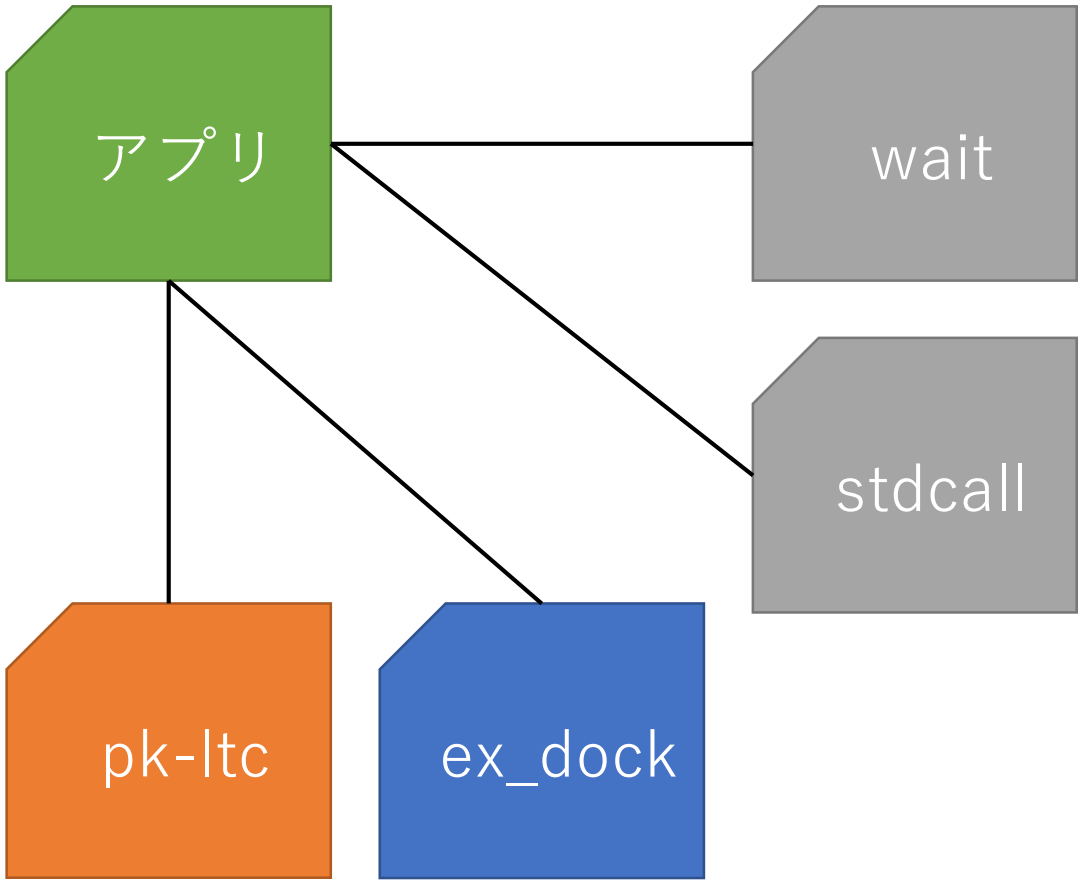
プロトタイプ宣言

calc.c

```
#include "calc.h"

int add(int  x, int y){
        return x + y;
}
```

関数の本体

今回の分割方針



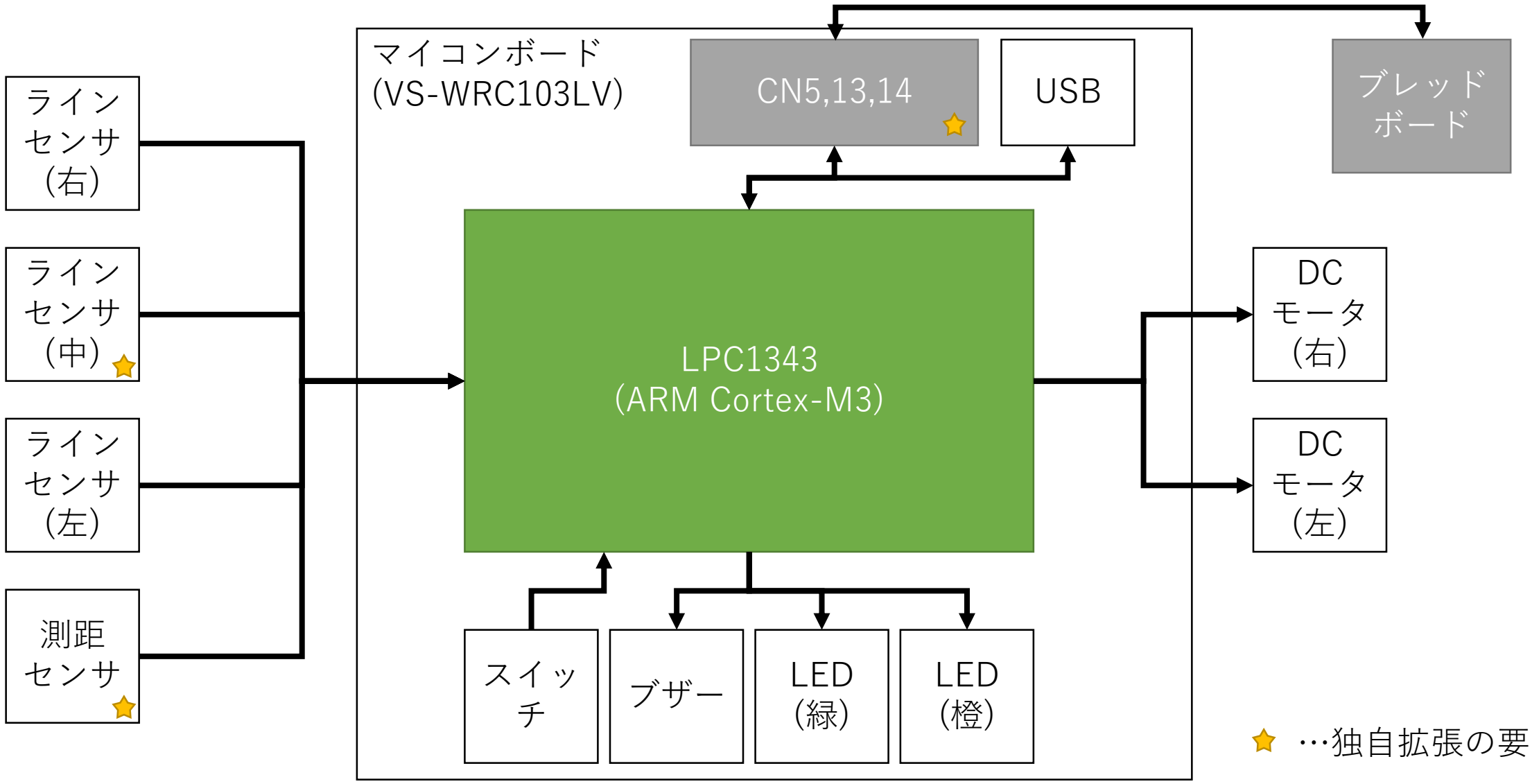
- ファイル分割の際の切り分け方方は、設計者や対象とするシステムによって異なる
- 今回は次のように分割する
  - pk_ltc
    - ラインとレースカーに関する関数や定数などを管理
    - LEDやスイッチ、モータなど
  - ex_dock
    - 学習＆デバッグ用のドック上で構築する回路などの制御に関する関数や定数などを管理
    - フルカラーLEDやスライドスイッチ、温度センサなど
  - wait
    - wait関数を管理
  - Stdcall
    - システムコールを管理

課題

- Kadai03_03.cをベースにファイル分割を行い、ソースコードを整理します
  - test02_kadai.zipをプロジェクトに追加して下さい
    - Quickstart Panel>import project(s)>Project archive(zip)より追加可能
  - 前のページの分割方針に合わせて、既に雛形が出来上がっています
  - **pk_ltc.c**および**ex_dock.c**が未完成となっていますので、こちらを埋めて完成させてください

- 注意事項
  - Redlibの設定は再度行う必要があります
  - 今後、書き込むbinファイルはtest02.binという名前に変わります

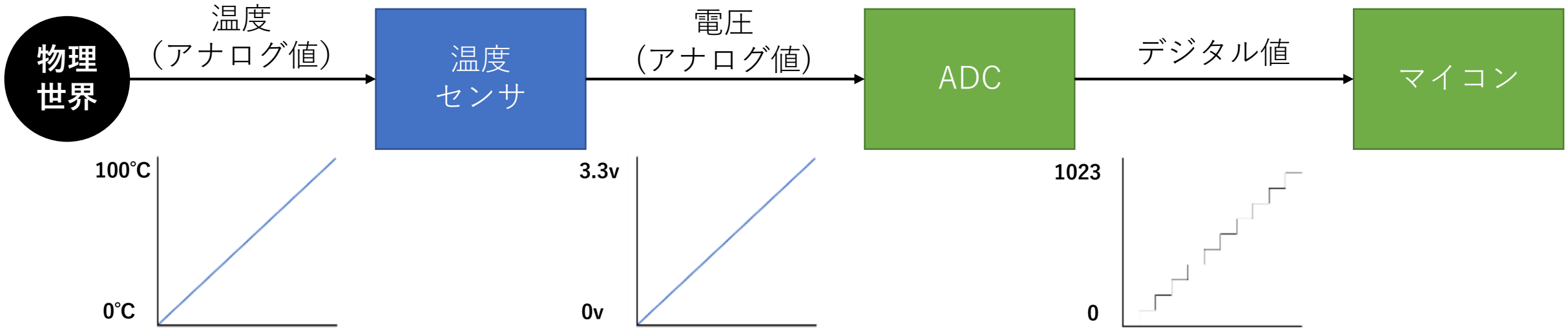
05:ADコンバータ入門



はじめに

- ADコンバータとは、アナログ量をデジタル量に変換する機能です。多くのマイコンに内蔵されています。
- 物理世界のほとんどの物理量はアナログ量であり、温度・湿度・光量・音量・加速度など全て連続的なアナログ値になります。
- 組込み製品ではこれらのアナログ量を扱うことが多々ありますが、そのままでは扱えない問題がある。そこでアナログ量をデジタル量に変換し、マイコンでも扱えるようにする。

アナログ値がデジタル値に変換されるまでの流れ



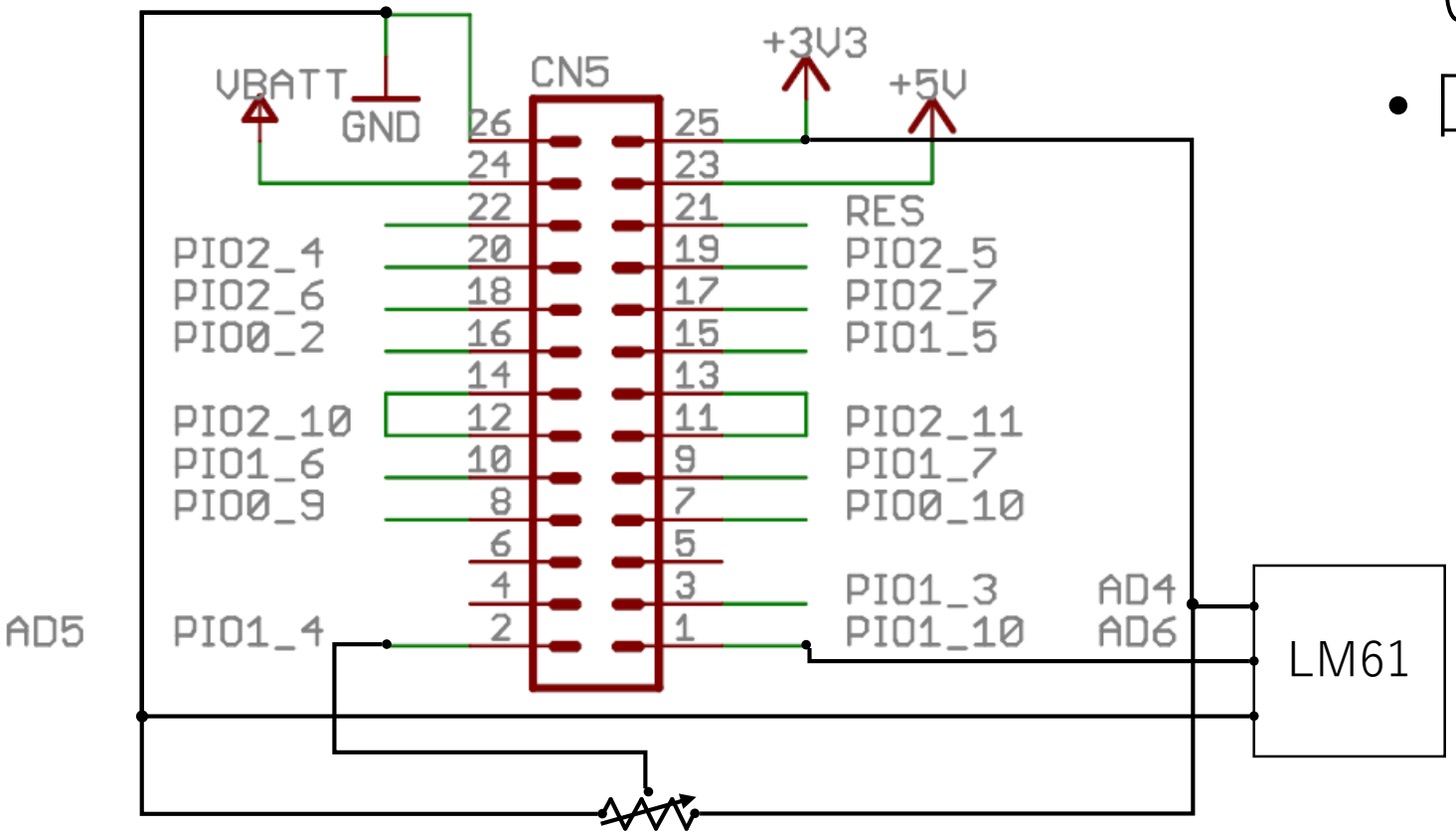
- センサ
  - アナログ量を電圧に変換するモノ
  - 変換できるアナログ量、電圧はセンサによって様々
  - 変換出力は電圧だけではなく、電流や抵抗値のことも
    - この場合、ADCとの間にI→VやR→Vの変換回路を入れる
- ADコンバータ(ADC)
  - 電圧をデジタル値に変換するモノ
  - 分解能が高ければ高い程、より細かくデジタル値で表現できる

図の事例の場合、マイコンが…
　ADCから0を取得したら0℃
　ADCから1023を取得したら100℃
ということになる

分解能について

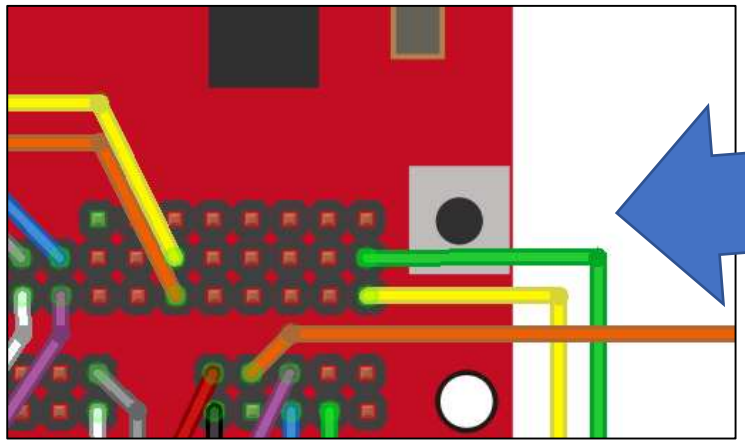
- ADコンバータの性能を指す。分解能が高い程、より細かくアナログ値をデジタル値に変換できる。
- 通常、ビット数で表され、8bit, 10bit, 12bit, 16bit…というように表現される。
- このビット数は、デジタル量で表現できる大きさを表す。
  - 仮に2bitのADコンバータであれば、4段階に分けてアナログ量を表現できる
  - 温度センサであれば、「冷たい・ぬるい・暖かい・熱い」の四段階程度
- 分解能が高ければより細かい正確な測定ができるのか？
  - 高度な測定に必要な要素の1つであるが、他にも必要とされる要素がある為、分解能1つでは高度な測定は実現できない

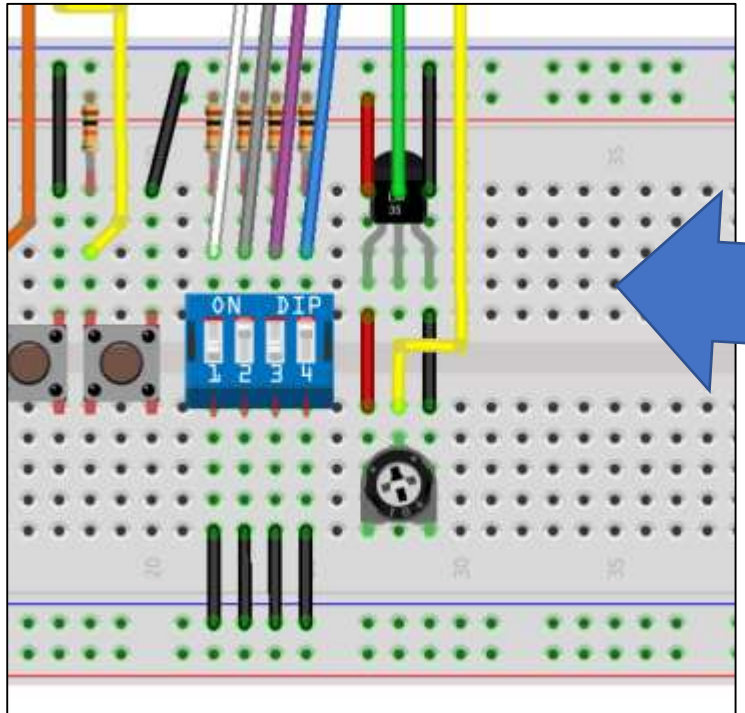
回路図



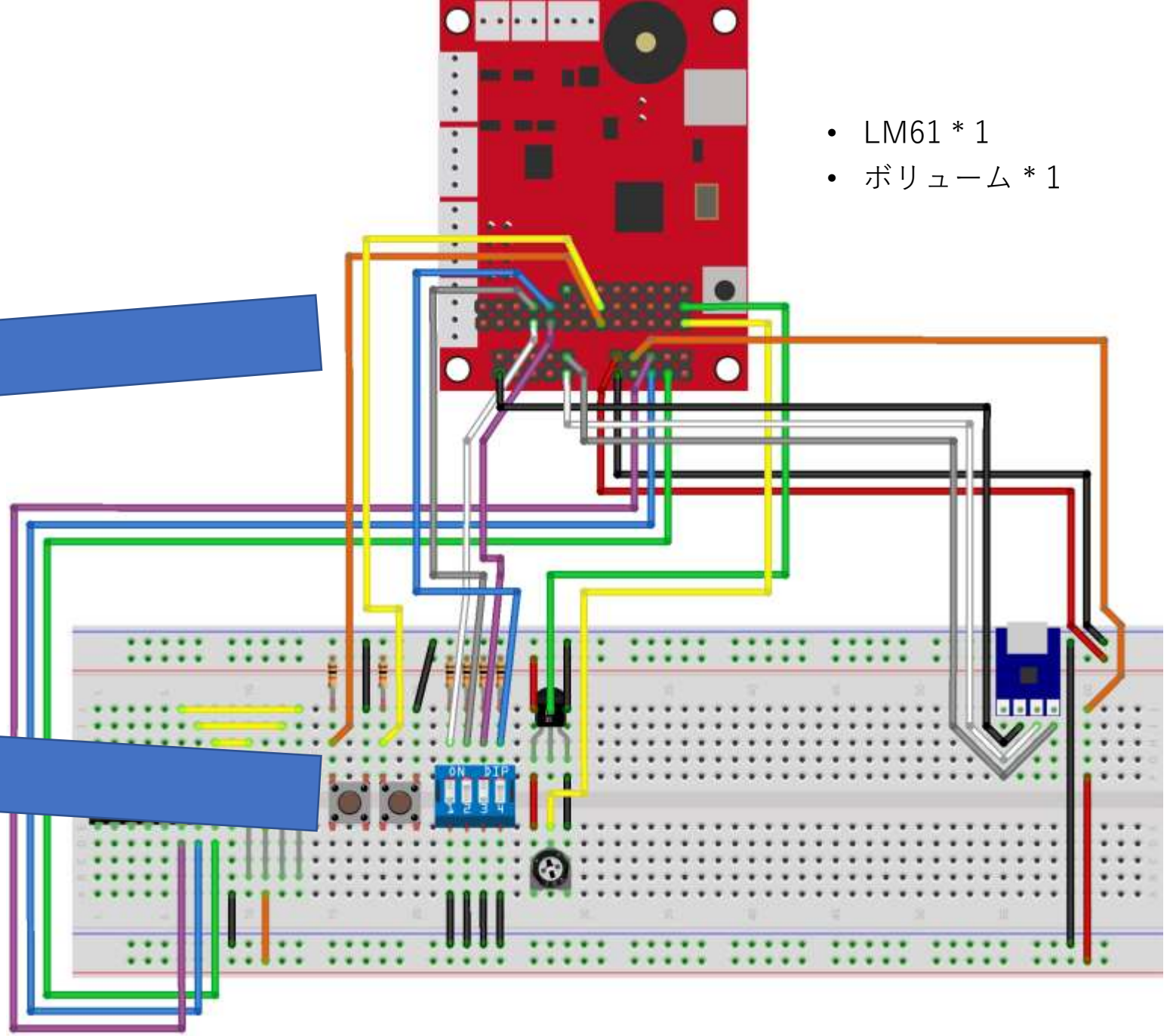
- ボリュームと温度センサについて説明
- 回路図

実体配線図





- LM61 * 1
- ボリューム * 1

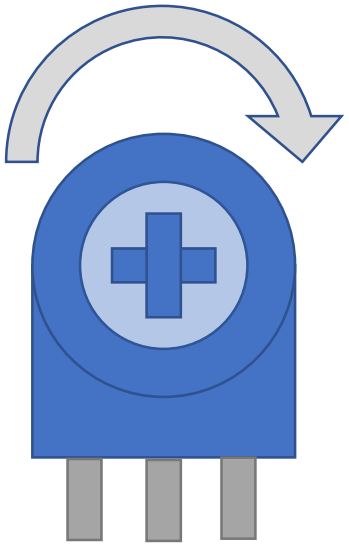


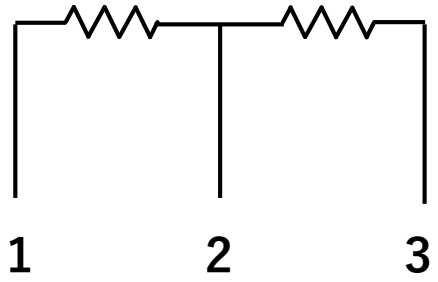
回路の簡単な説明

- ボリューム
  - ボリューム抵抗、可変抵抗などと呼ばれるツマミを回すことで抵抗値を変化させることが出来る抵抗器です
  - 総和(Ra+Rb)は常に一定で、ツマミの位置でRaとRbの値が変化します

ツマミ	1-2間	2-3間
反時計回りに回し切る	1kΩ	0Ω
真ん中に合わせる	500Ω	500Ω
時計回りに回し切る	0Ω	1kΩ

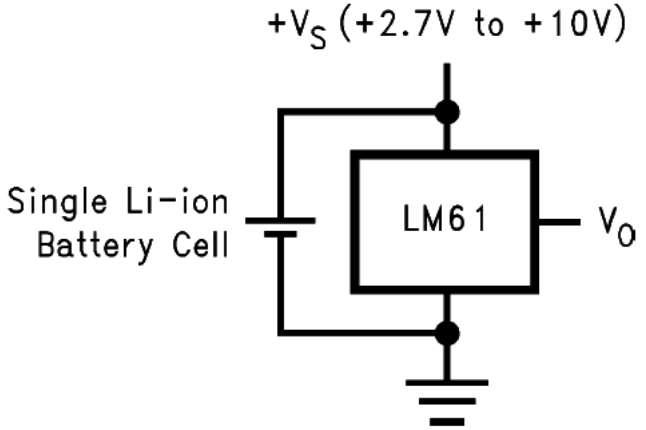
  - 総和はボリュームの型番によって変わります
    - 100Ω, 1kΩ, 10kΩ, 100kΩなどがある
- ボリュームはマイコンのAD5へ接続





回路の簡単な説明

- 温度センサ(LM61)
  - -30～100°Cの温度測定が可能なセンサ
  - -30～100°Cの温度を+300mV～1600mVの電圧に変換
    - 0°Cの時に600mVの電圧が出力される
- 温度センサはマイコンのAD6へ接続

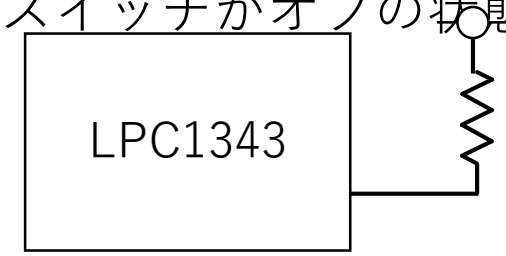


$$V_O = (+10\ mV/℃ \times T\ ℃) + 600\ mV$$

Temperature (T)	Typical V_O
+100˚C	+1600 mV
+85˚C	+1450 mV
+25˚C	+850 mV
0˚C	+600 mV
−25˚C	+350 mV
−30˚C	+300 mV

回路の簡単な説明

- VS-WRC10LV基板には、ボタンスイッチが1つ実装されており、IOへの割付は以下の通りです。
  - ボタンスイッチ(黒)　　：PIO0_1
- ブレッドボード上には、ボタンスイッチとスライドスイッチが実装されており以下の通りの割付です。
  - ボタンスイッチ(左・白)　　：PIO2_10
  - ボタンスイッチ(右・黄)　　：PIO2_11
  - スライドスイッチ1　　：PIO2_4
  - スライドスイッチ2　　：PIO2_5
  - スライドスイッチ3　　：PIO2_6
  - スライドスイッチ4　　：PIO2_7
- スイッチがオフの状態では、ポートはプルアップされた状態になる為、’1’が入力されます。



- スイッチがオンの状態では、ポートはGNDに接続されますから、’0’が入力されます。

例題:rei05_01.c

```
#ifdef __USE_CMSIS
#include "LPC13xx.h"
#endif

// 省略

int main(void) {
    int vol, ls_l, ls_c, ls_r;
    double temp, dist;

    init_syscall();
    init_adc();

    while(1) {
        vol = get_volume();
        ls_l = get_left_line_sensor();
        ls_c = get_center_line_sensor();
        ls_r = get_right_line_sensor();

        temp = get_temperature();
        dist = get_distance();

        printf("vol:%4d, l:%4d, c:%4d, r:%4d,", vol, ls_l, ls_c, ls_r);
        printf("temp:%4.2lf, dist:%4.2lf¥n", temp, dist);

        wait_ms(500);
    }

    return 0 ;
}
```

- 各種センサの値を取得し、シリアル通信で値を確認できるプログラム
  - ボリューム(可変抵抗)
  - ラインセンサ左
  - ラインセンサ右
  - ラインセンサ中央
  - 温度センサ
  - 距離センサ

例題:rei05_01.c(動作確認)

- 各種センサの値を変化させ、シリアル通信で出力されるAD値の値が変化する事を確認しましょう
  - ラインセンサ

	左	中央	右
何もしない時			
指で覆った時			

  - 温度センサ

何もしない時	
ドライヤーで5秒間あてた時	

  - 距離センサ

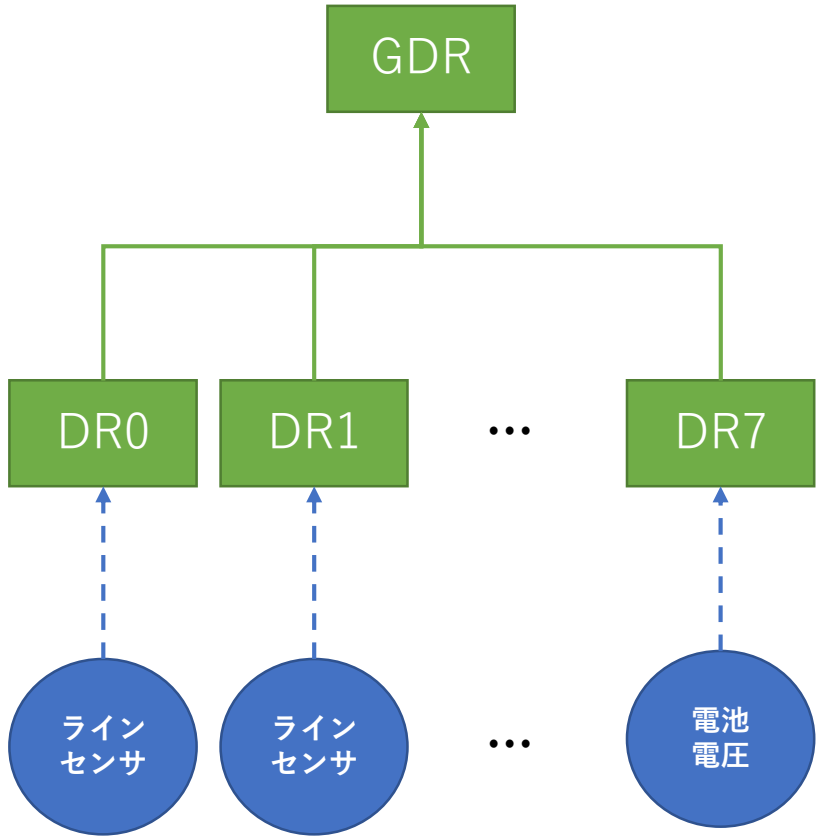
何もしない時	
LTCの前方に手をかざした時	

ADCのレジスタ制御（初期化）

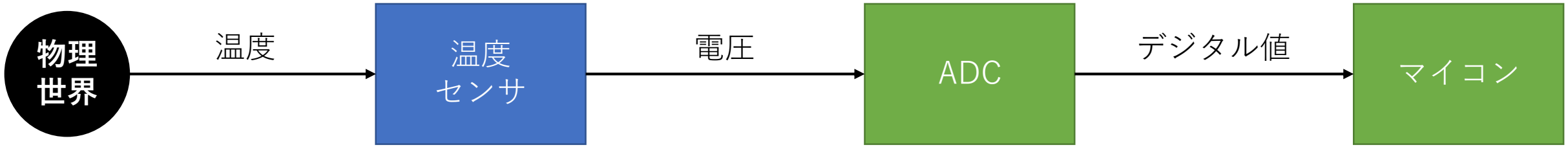
- IOCONの設定(UM Chap7参照)
  - IO Pinの機能割当をADに変換する
  - プルアップモードを無効化
- SYSCONの設定(UM Chap3.5参照)
  - 電源管理機能よりADCへ電源供給する
  - システム及び周辺ブロックへのクロック管理機能より、ADCへクロック供給する
- AD Control Register(UM Chap20.6.1参照)
  - ADCの管理を行うレジスタの初期設定を行う

ADCのレジスタ制御(値取得)

- AD Control Register
  - AD変換の開始・停止制御
- AD Data Register 0～7
  - 各ADチャンネルに対して変換中の確認
  - 変換できたデジタル値の取得
  - オーバーラン発生の有無確認
- AD General Data Register
  - 最新のAD変換の結果が格納される
  - 実行中やオーバーラン発生情報なども含まれる



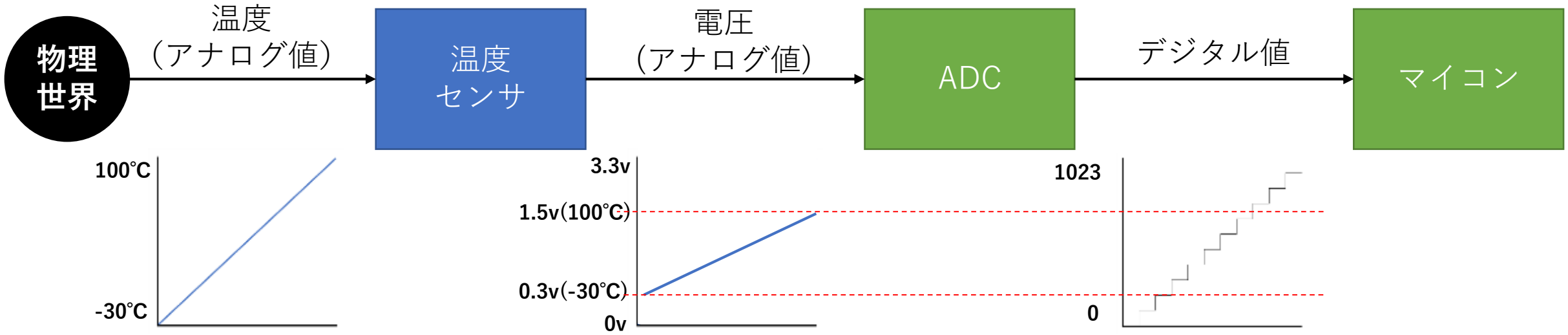
センサからの温度算出方法



- 今回使用するLM61と呼ばれるセンサは-30°C～+100°Cまでの温度範囲を検出できる温度センサIC
- 出力電圧は摂氏温度にリニアに比例(+10mV/°C)
- 内部に600mVのDCオフセットを持つ
  - つまり、出力が600mVの時に0°Cとなる
- 温度と電圧の関係は以下の算出式から求められる
  - Vo = (+10mV/°C * T°C) + 600mV　　(…温度からの電圧算出)
  - T°C = (Vo – 600mV) / +10mV　　(…電圧からの温度算出)

Temperature (T)	Typical V_o
+100°C	+1600 mV
+85°C	+1450 mV
+25°C	+850 mV
0°C	+600 mV
–25°C	+350 mV
–30°C	+300 mV

アナログ値がデジタル値に変換されるまでの流れ



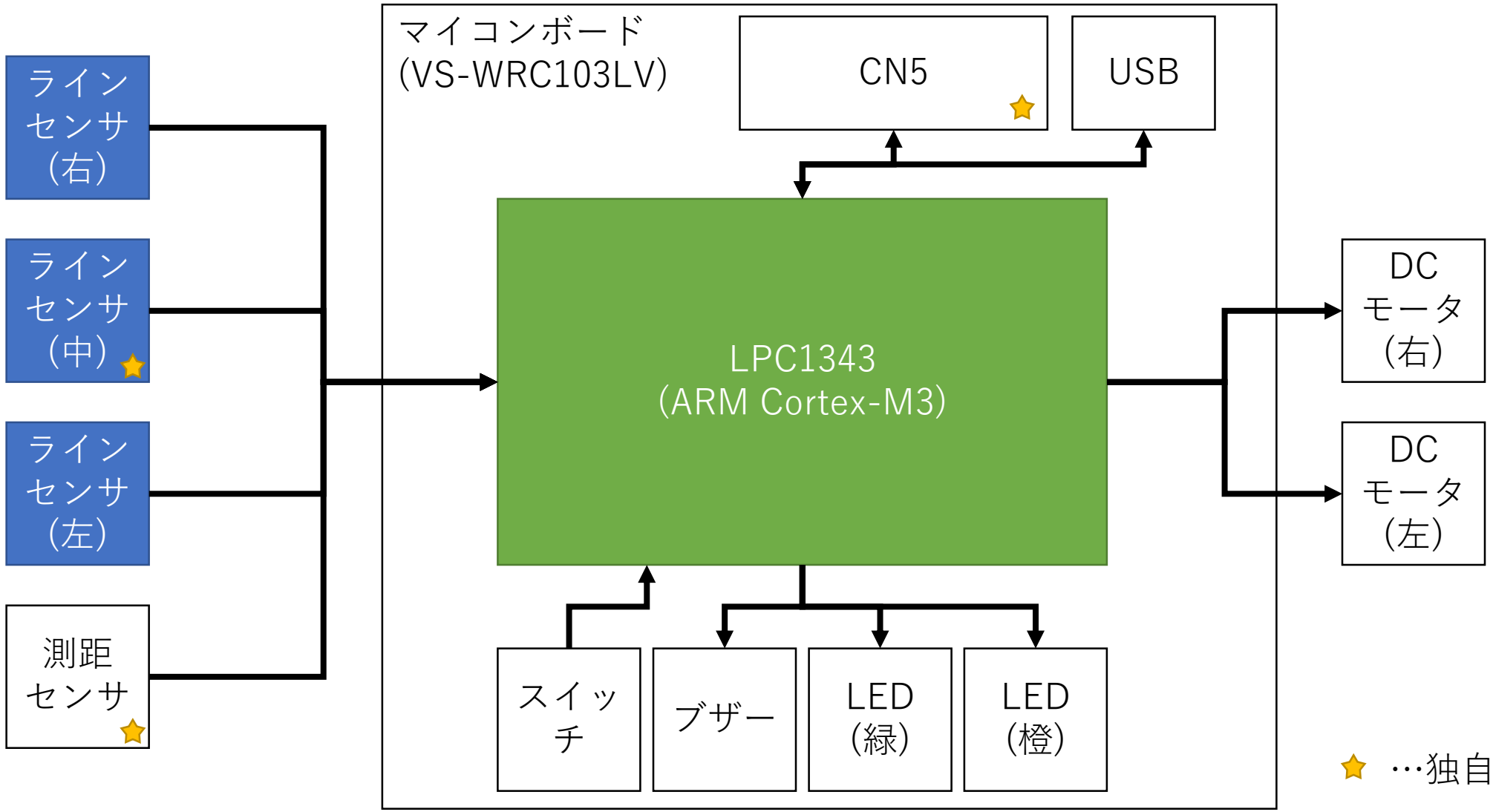
- 今回の場合、リニアリティの取れたセンサである為、温度算出は非常に容易
- 一方で、出力電圧が0.3v～1.6vと差が1.3v程度しかない.
  ADCは0～3.3vの領域を 0 ～1023に変換する為、ADCのスペックの半分程度しか出せていない
  - ADCのスペックを最大限に出したい場合、温度センサとADC間にゲインやオフセット調整の為の回路が必要になる

課題

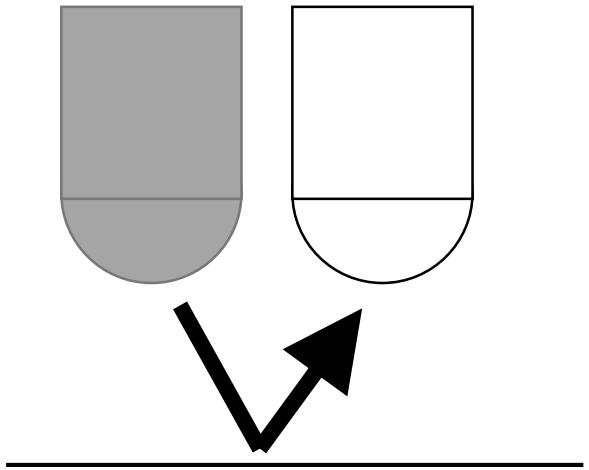
ファイル名は「kadai05_xx.c」とすること。

1. ボリュームの値でLチカ。ボリュームで取得した値が…
   500未満：フルカラーLEDを消灯
   500以上：フルカラーLEDを点灯(赤)
2. ボリュームで取得した値が…
   ～299　　：フルカラーLEDを消灯
   300～599　　：フルカラーLEDを点灯(青)
   600～899　　：フルカラーLEDを点灯(緑)
   900～　　：フルカラーLEDを点灯(赤)

06:ADCによるラインセンサ制御

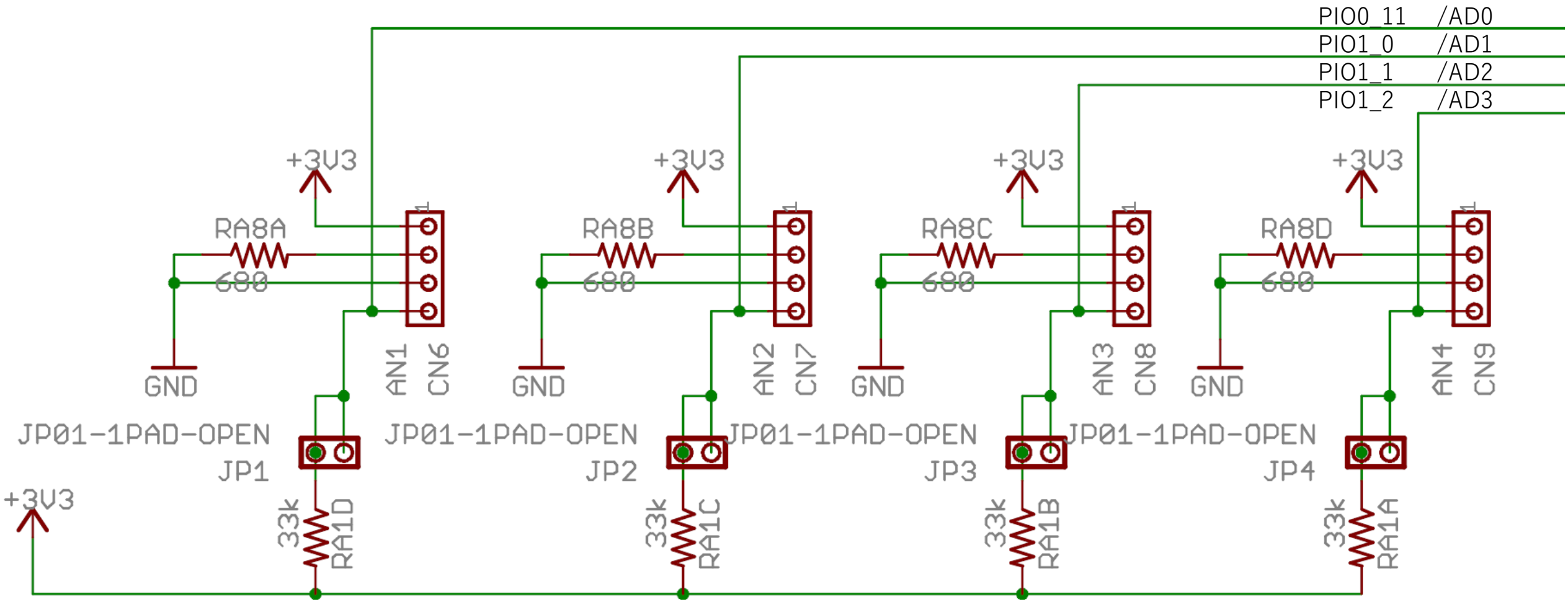


ラインセンサの仕組み

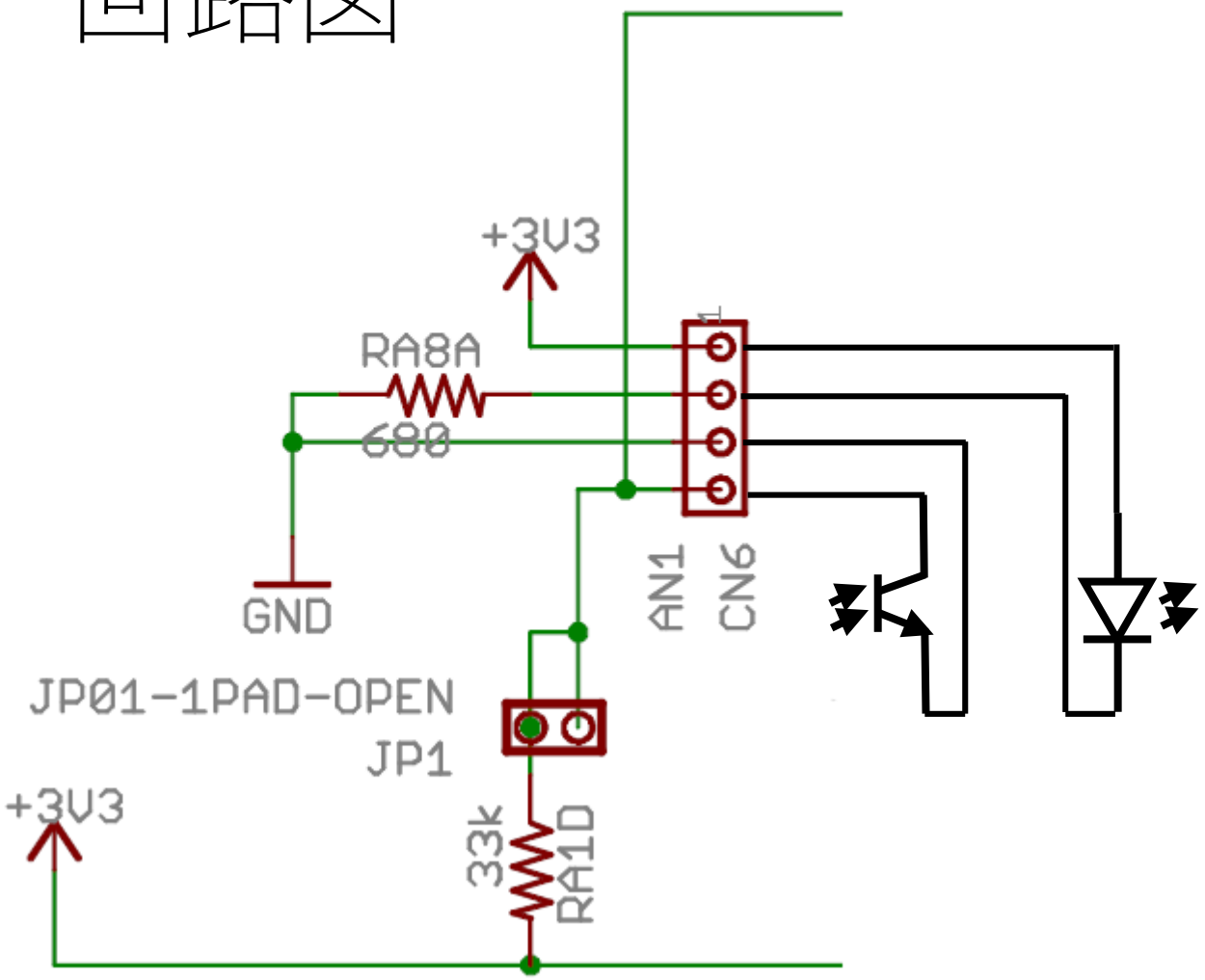


- 赤外線LEDより赤外線光を照射する
- 白い床、黒いラインに光が当たって反射した光を受光素子(フォトダイオードorフォトトランジスタ)で受け取る
- 受光素子は、その光量に合わせた電圧を出力する

回路図①


PIO0_11 /AD0
PIO1_0 /AD1
PIO1_1 /AD2
PIO1_2 /AD3
+3V3
RA8A
680
GND
AN1
CN6
JP01-1PAD-OPEN
JP1
33k
RA1D
RA8B
AN2
CN7
JP2
RA1C
RA8C
AN3
CN8
JP3
RA1B
RA8D
AN4
CN9
JP4
RA1A

回路図



- CN6,7,8の先にはそれぞれラインセンサが接続されている

ラインセンサ左	CN　（AD　）
ラインセンサ中央	CN　（AD　）
ラインセンサ右	CN　（AD　）

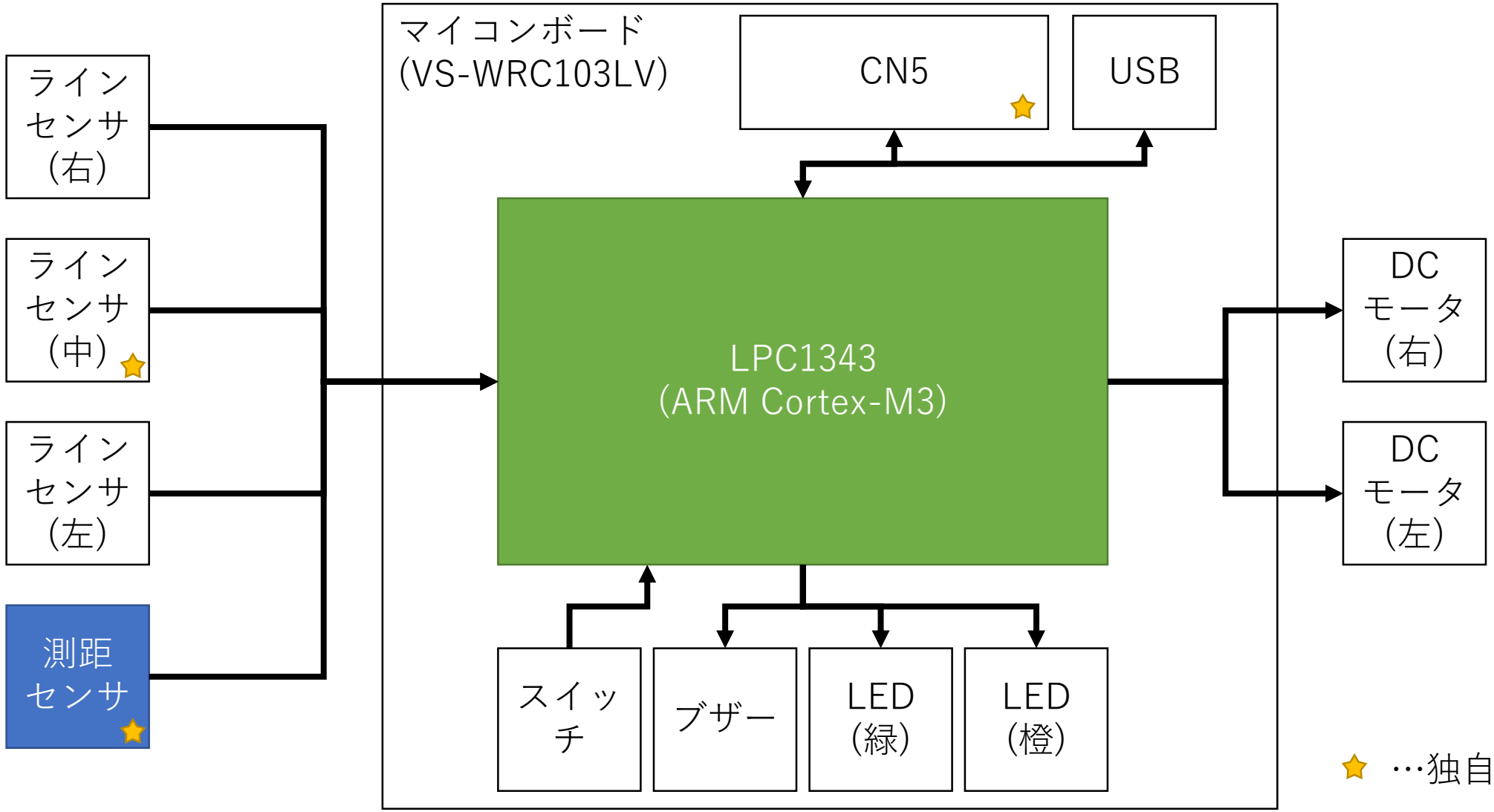
- 回路図に起こすと左図のような形となる
- 受光素子は光量が多ければ、多いだけ電流を流す

- CN9には測距センサが接続されている

デジタル値と物理量

- ラインセンサやボリュームなどは正確な物理量の算出は出来ない
  - ラインセンサ：光量からライン検知
  - ボリューム　：回転角度の検出
  - 変化量から何かを判断したい場合、この手のセンサは扱われる
- 温度センサや測距センサは、デジタル値→電圧→物理量まで算出が可能
- 2種類あるので、扱うセンサがどちらか

07:ADCによる測距センサ制御



★ …独自拡張の要素

測距センサの仕組み

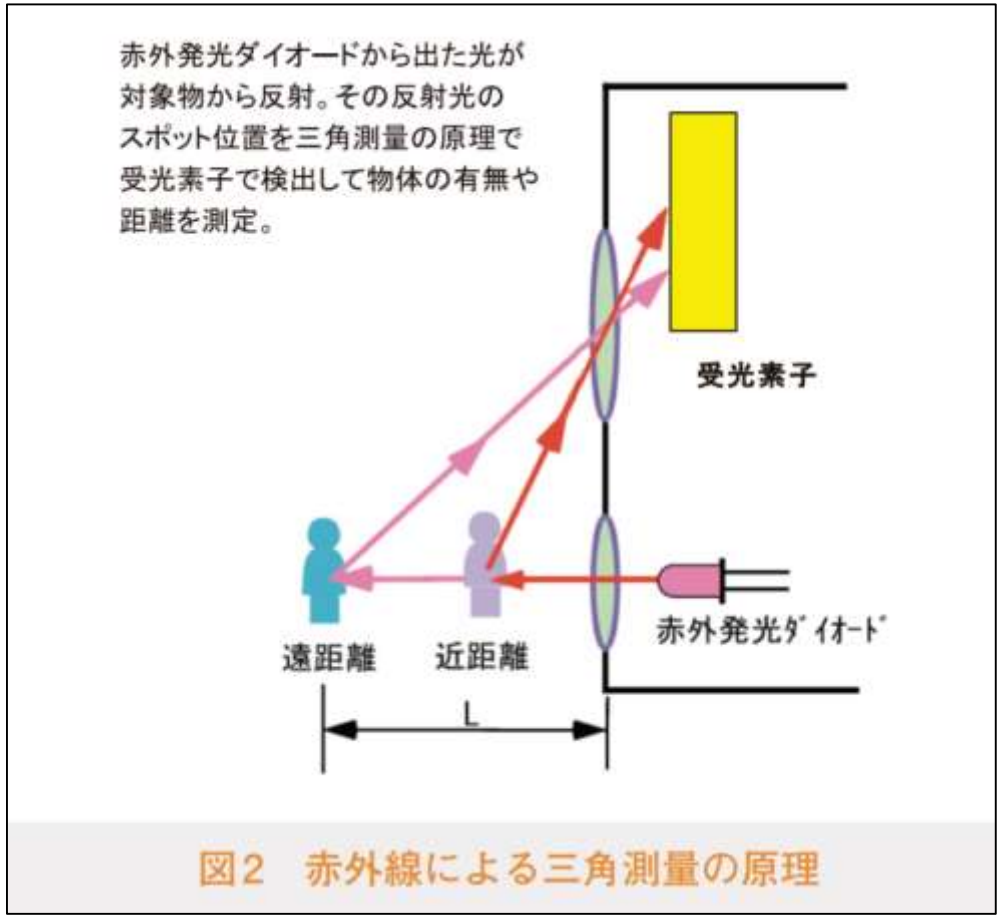


図2　赤外線による三角測量の原理

[仕組みについて]

- 赤外線LEDから照射された光の反射で測量する
- ポイントは受光素子であり、反射した光の角度によりスポット位置が異なる
- この位置から距離を計算によって求める。という仕組み

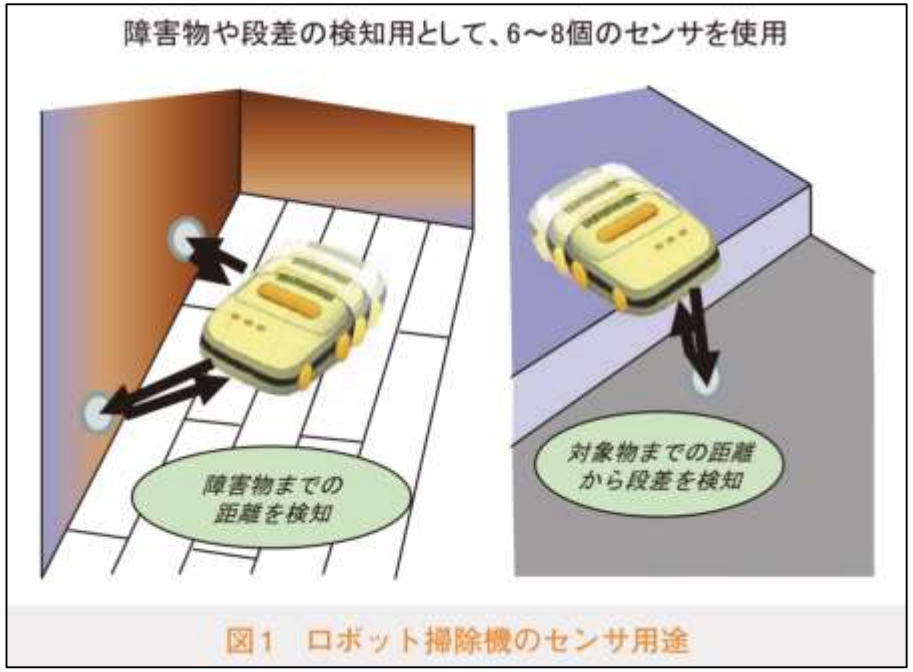
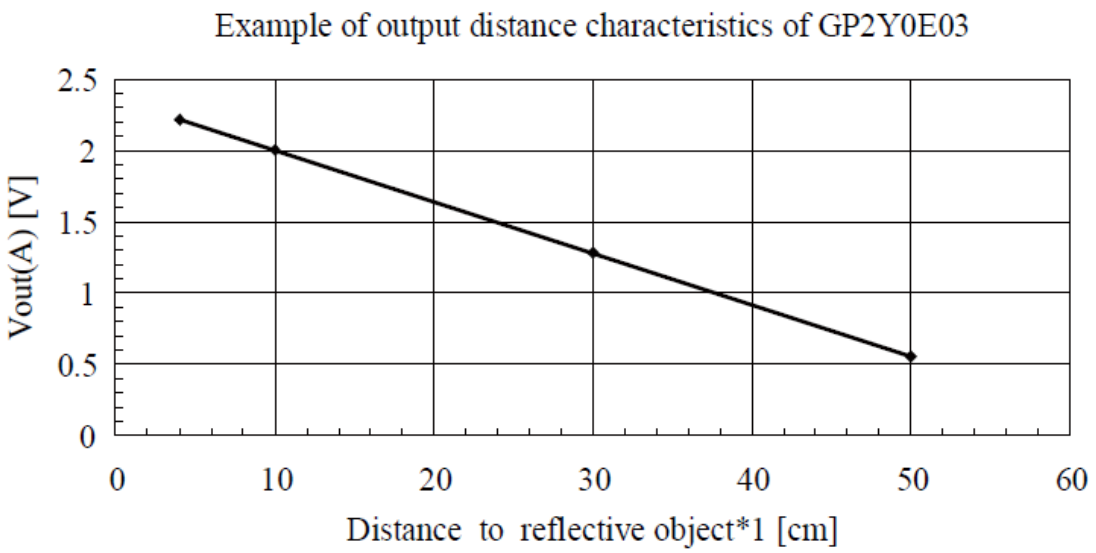


図1　ロボット掃除機のセンサ用途

距離の算出方法

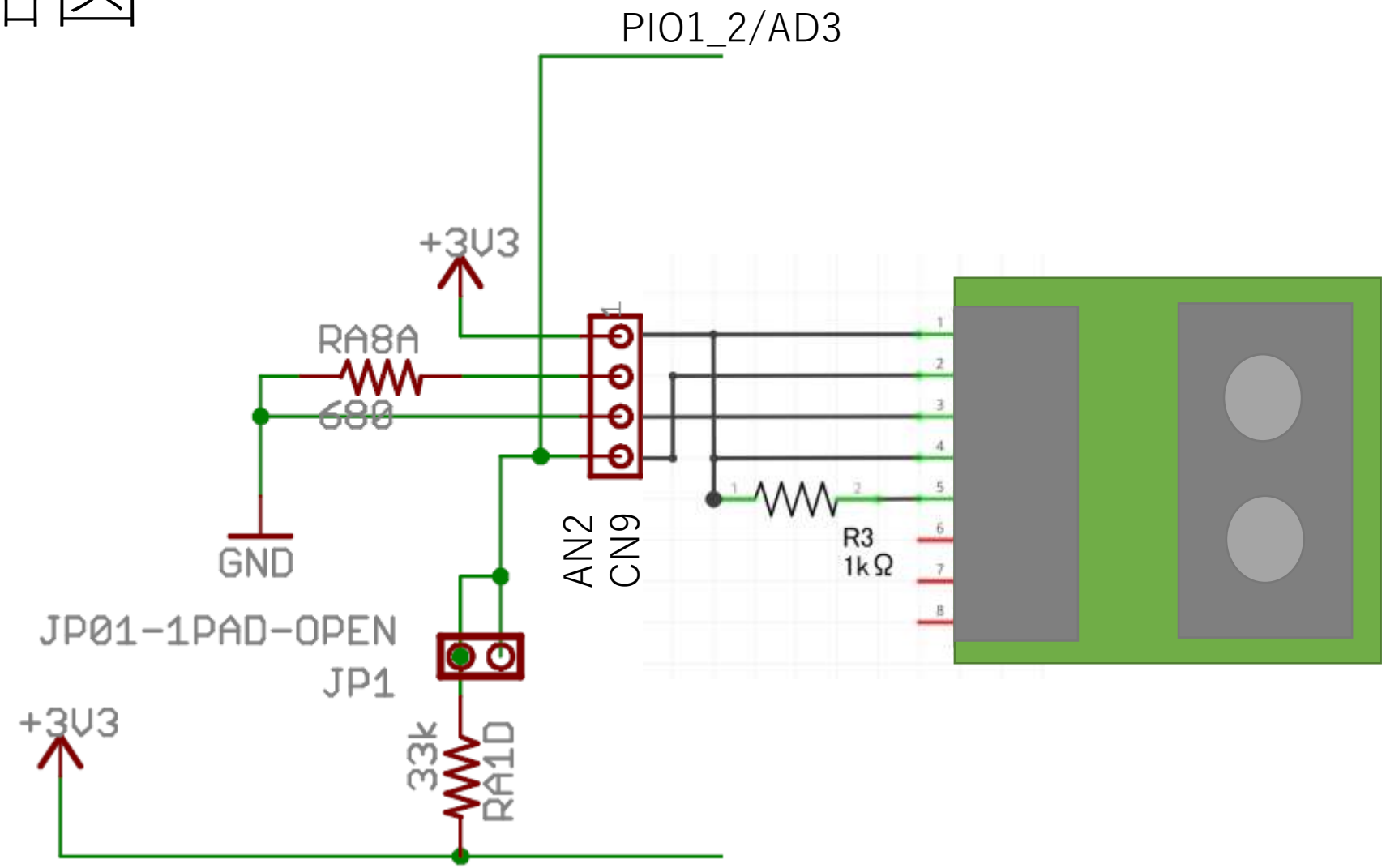


*1 : Using reflective object : White paper (reflective ratio : 90%)

- グラフから以下の事が分かる
  - 2Vの時、物体まで10cm
  - 1.3Vの時、物体まで30cm
  - 10.6Vの時、物体まで50cm
- このグラフの切片と傾きを求める

  y=ax+b　　b=2.35
  2=10a+b　　y=-0.035x+2.35
  0.6=50a+b
  a=-0.035　　y:電圧, x:距離
- YとXを入れ替えると…
  - **x=(-200y+470)/7**
  - 本式により電圧から距離が算出可

回路図


PIO1_2/AD3
+3V3
RA8A
680
GND
AN2
CN9
R3
1kΩ
JP01-1PAD-OPEN
JP1
+3V3
33k
RA1D

課題

ファイル名は「kadai06_xx.c」とすること。

１．温度計
　温度センサの値が...
　　20°C未満　　：フルカラーLEDを点灯(水)
　　20°C以上28°C未満　：フルカラーLEDを点灯(緑)
　　28°C以上35°C未満　：フルカラーLEDを点灯(赤)
２．高温アラーム機能の追加
　１の機能に以下を追加
　　35°C以上40°C未満　：フルカラーLEDを点灯(赤)を500ms周期で点滅
　　40°C以上　　：フルカラーLEDを点灯(赤)を100ms周期で点滅

課題

ファイル名は「kadai07_xx.c」とすること。

１．ラインセンサの組み合わせ

左・中央・右のセンサが覆われていない	：緑・橙LEDを消灯
左のセンサが覆われた	：緑LEDを点滅
右のセンサが覆われた	：橙LEDを点滅
左・右のセンサが覆われた	：緑・橙LEDを点滅
左・中央・右のセンサが覆われた	：緑・橙LEDを点灯

(点滅周期は全て200ms周期)

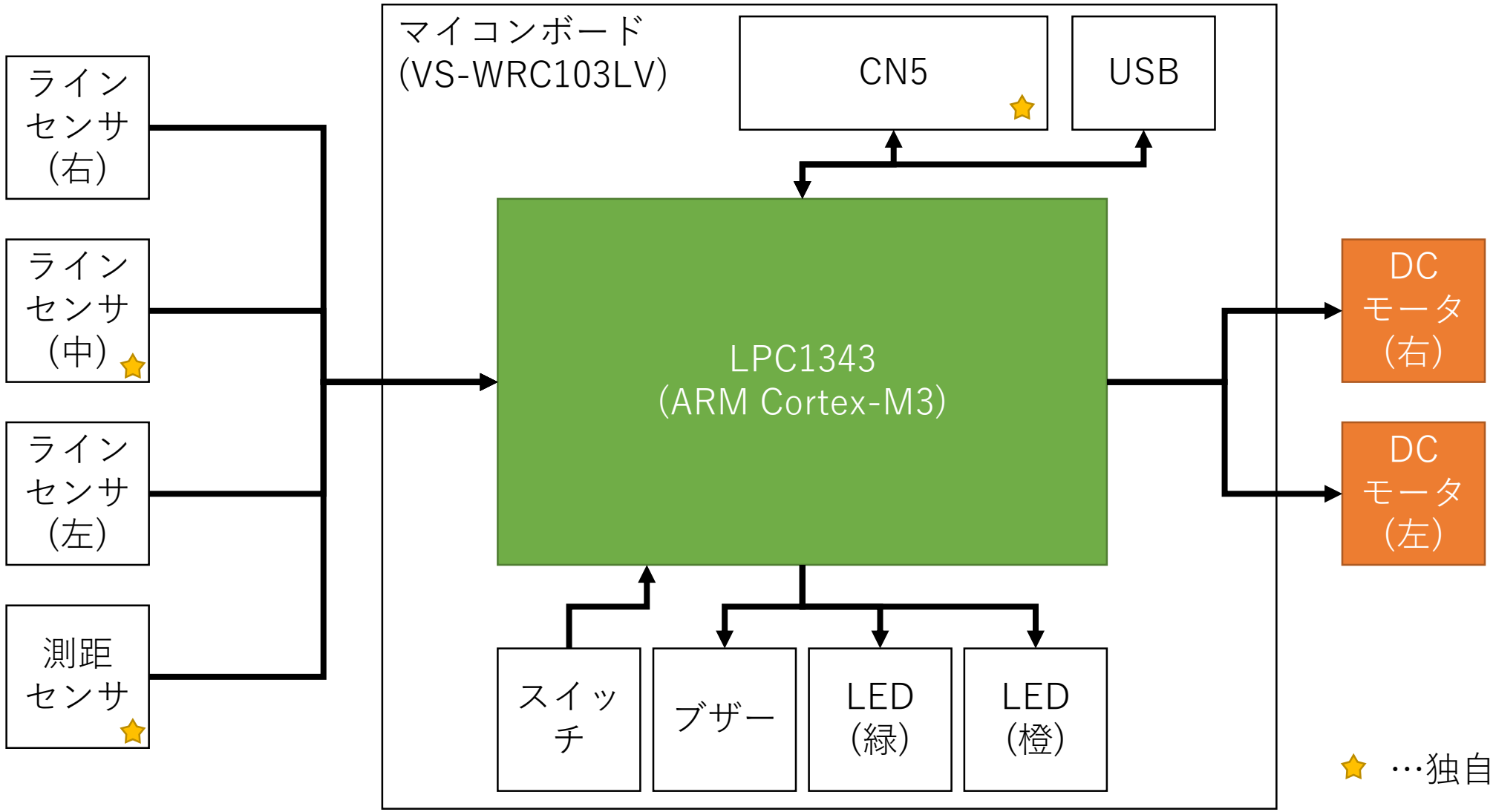
２．測距センサの関数を完成させよう！

－現在はADコンバータで取得したデジタル値を得られるだけ
　デジタル値を距離に変換してください

３．[発展] 侵入検知システム

－20cm以内に物体を検知した時，緑・橙LEDを交互点灯するプログラムを作成

08:GPIOによるDCモータ制御

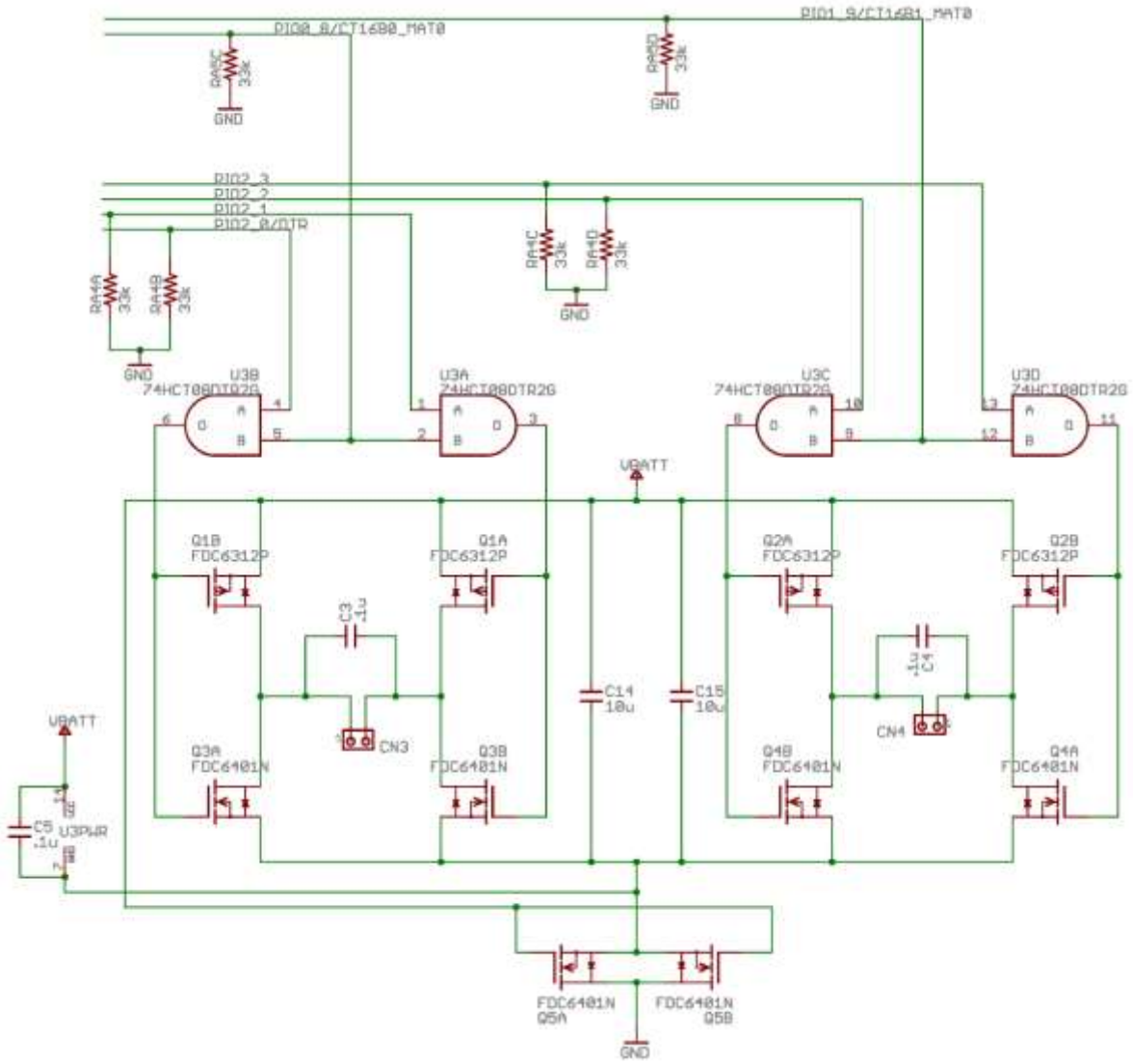


★ …独自拡張の要素

はじめに

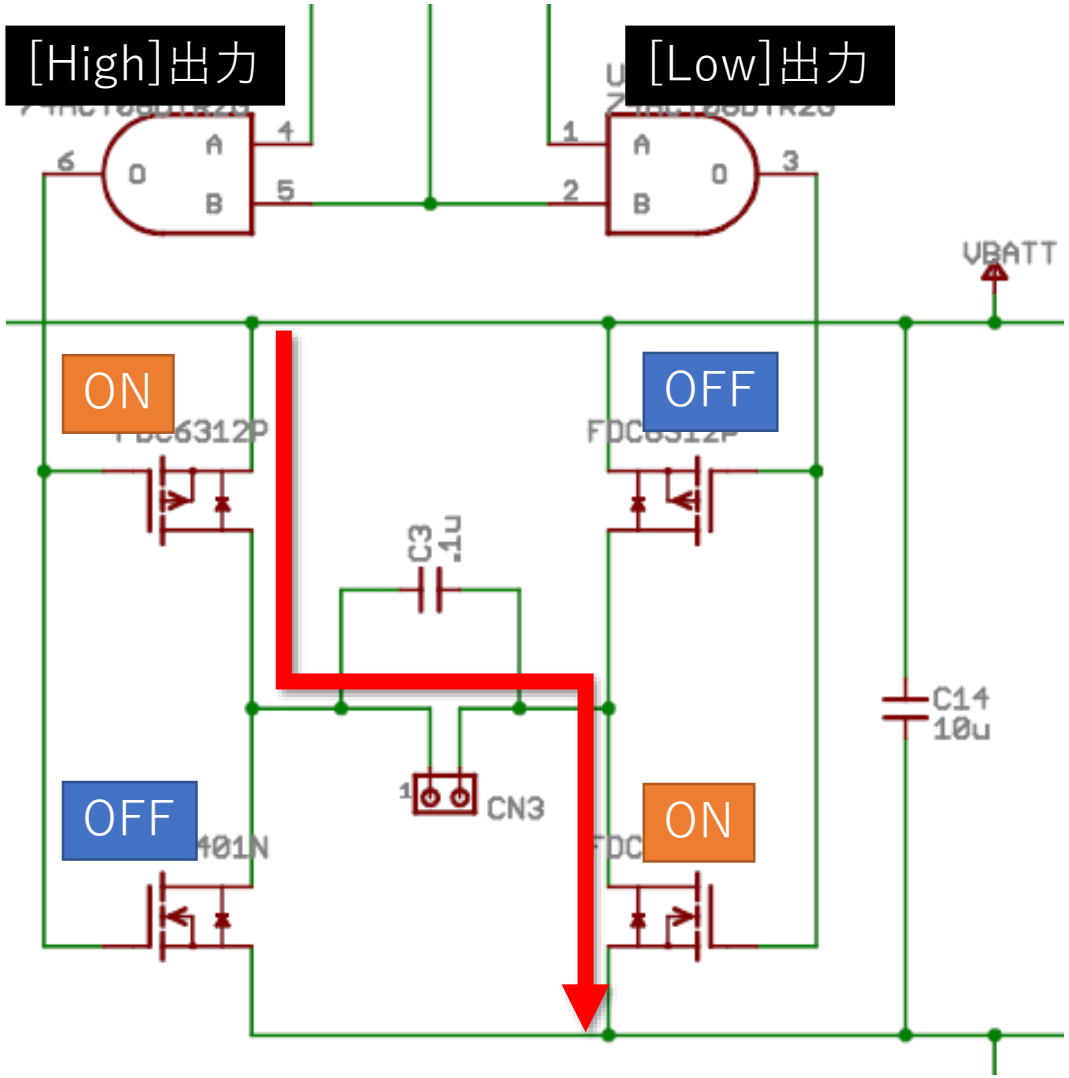
- いくつか種類のあるモータの中でも、この章ではDCモータの制御について学びます。
- ここでは、DCモータ制御の中でも基本的な正転・逆転の方法と速度調整の方法について触れます。
  - 電圧を印加することで回転速度を容易に制御でき、また印加する電圧の方向で回転方法の制御ができる、容易に制御が可能なモータです。
  - 一方で、DCモータ単体での位置決め制御は向きません。
- 今回使うDCモータはブラシモータとも呼ばれ、電動歯ブラシUSBファンやミニ四駆などにも利用される安価なモータです。

回路



- DCモータを制御する典型的なHブリッジ回路
  - 正転/逆転/停止を実現する
  - 1方向に対して回転させる/停止させるならばFETは1個で実現
- PIO2_0～2_3でモータの回転方向の制御
- PIO0_8,PI1_9で速度制御

Hブリッジ回路



	左モータ(CN3)			右モータ(CN4)		
Port	PIO2_0	PIO2_1	PIO0_8	PIO2_2	PIO2_3	PIO1_9
正転	H	L	H or ⎍	H	L	H or ⎍
逆転	L	H	H or ⎍	L	H	H or ⎍
ブレーキ	H	H	H	H	H	H
フリー	-	-	L	-	-	L

例題①：コマンドでモータ制御

```
int main(void) {
    char cmd;

    init_syscall();
    init_motor();
    init_ex_led_full_color();

    set_ex_led_full_color(LED_LIGHT_BLUE);

    printf("-----¥n");
    printf("*** Motor Control TEST ***¥n");
    printf("-----¥n");
    while(1) {
        printf("input command:");
        scanf("%c", &cmd);
        printf("¥n");

        switch(cmd){
        case 'a':
            rotate_motor(MOTOR_LEFT, CW);
            printf("> LEFT CW¥n");
            break;
        case 'b':
             rotate_motor(MOTOR_LEFT, CCW);
             printf("> LEFT CCW¥n");
             break;
        case 'c':
             rotate_motor(MOTOR_LEFT, BRAKE);
             printf("> LEFT BRAKE¥n");
             break;
        case 'd':
                 rotate_motor(MOTOR_LEFT, FREE);
                 printf("> LEFT FREE¥n");
                 break;

        case 'e':
                 rotate_motor(MOTOR_RIGHT, CW);
                 printf("> RIGHT CW¥n");
                 break;
        case 'f':
                 rotate_motor(MOTOR_RIGHT, CCW);
                 printf("> RIGHT CCW¥n");
                 break;
        case 'g':
                 rotate_motor(MOTOR_RIGHT, BRAKE);
                 printf("> RIGHT BRAKE¥n");
                 break;
        case 'h':
                 rotate_motor(MOTOR_RIGHT, FREE);
                 printf("> RIGHT FREE¥n");
                 break;
        }
    }

    return 0 ;
}
```

必ずLTCを浮かして動かすこと

TeraTermからコマンドを
打ち込んで動作確認

コマンド一覧は以下の通り

- ‘a’　　:左モータ 正転
- ‘b’　　:左モータ 逆転
- ‘c’　　:左モータ ブレーキ
- ‘d’　　:左モータ フリー
- ‘e’　　:右モータ 正転
- ‘f’　　:右モータ 逆転
- ‘g’　　:右モータ ブレーキ
- ‘h’　　:右モータ フリー

課題①：コマンドによる前進・後進ほか

- kadai08_01.cの名前で作成
- 前進・後進・右旋回・左旋回・停止・フリーをする関数を作成せよ

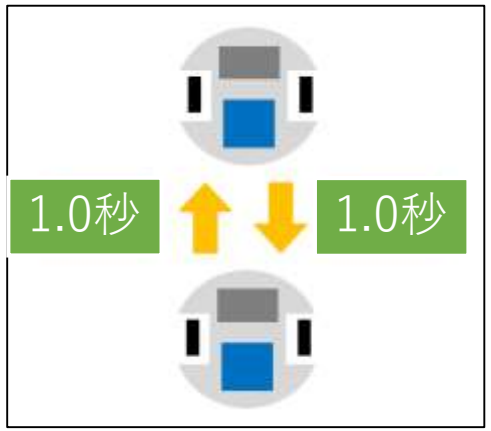
```
void _forward(void);
void _back(void);
void _turn_left(void);
void _turn_right(void);
void _stop(void);
void _free(void);
```

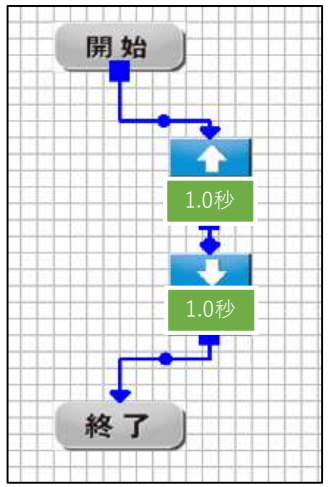
- コマンドは以下の通り

‘i’	:前進	‘m’	:停止
‘j’	:後進	‘n’	:フリー
‘k’	:左旋回		
‘l’	:右旋回		

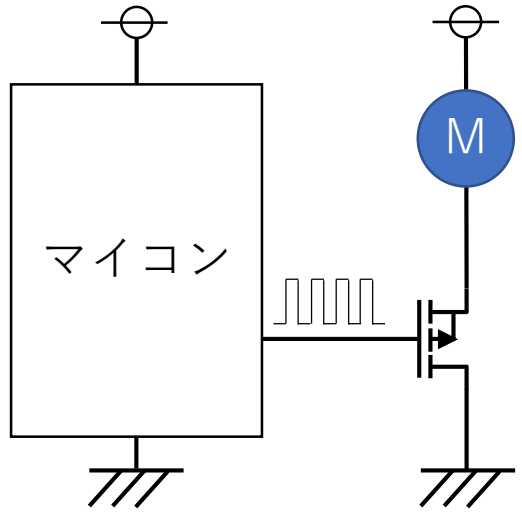
課題②：前進＆後進

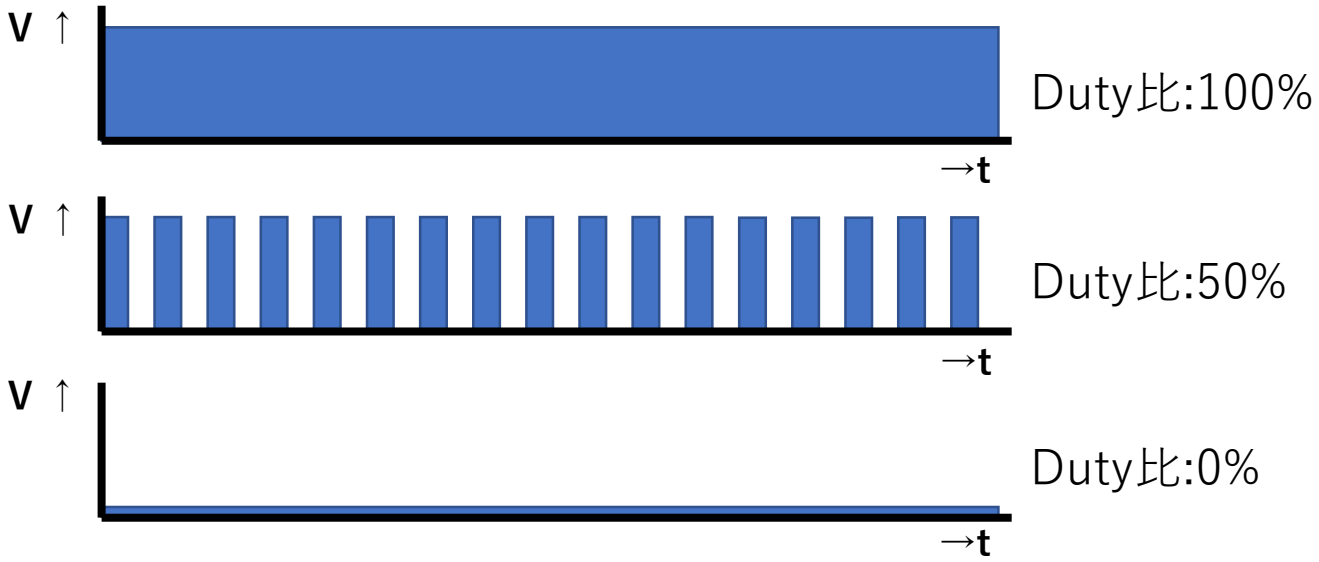
- ここより2人１組でやること
  - 1人しかいない席は、LTCを2台用いて行う
  - 片方の人がドックからライントレースカーを取り外す
    - 1台は机上でのデバッグ・検証用途
    - もう1台は実際に動作させてのデバッグ・検証用途
- ボタン押下後、**1秒の前進**をしたあと、**1秒の後進**し、最後に停止をするプログラムを作成せよ





速度制御とその仕組み





- PWM制御による速度制御を実現
- モータに流す電流を細かくON/OFFさせてTotalで流れる電流量を調整することで速度変化が生じる

- Duty比とは
  - ON時間とOFF時間の比
- 同じ周期でデューティ比の増減で制御を行う

例題②：GPIO+Waitによる速度制御

```
while(1){
    mode = get_ex_slide_switch();
    vol = get_volume();
    if(vol <= 0)
    vol = 1;
    //printf("%x, %d¥n", mode, vol);

    switch(mode)
    {
    case 0x0f:
        _stop();
        break;
    case 0x0e:
        _forward();
        break;
    case 0x0c:
        _forward();
        wait_ms(10);
        _stop();
        wait_ms(10 * vol / 1000);
        break;
    }
}
```

- ドック上に置いた実機で動作確認してください
- 機能は以下の通りです
  - SSW全てOFF
    - 停止状態
  - SSW1:ON
    - 通常の前進
  - SSW1+SSW2:ON
    - ボリュームで速度調整

- プログラム書き込み後、速度調整機能が確認出来たらオシロスコープで、モータの波形を確認してください
  - ボリュームの捻り具合でデューティ比が変わる事を確認

課題③

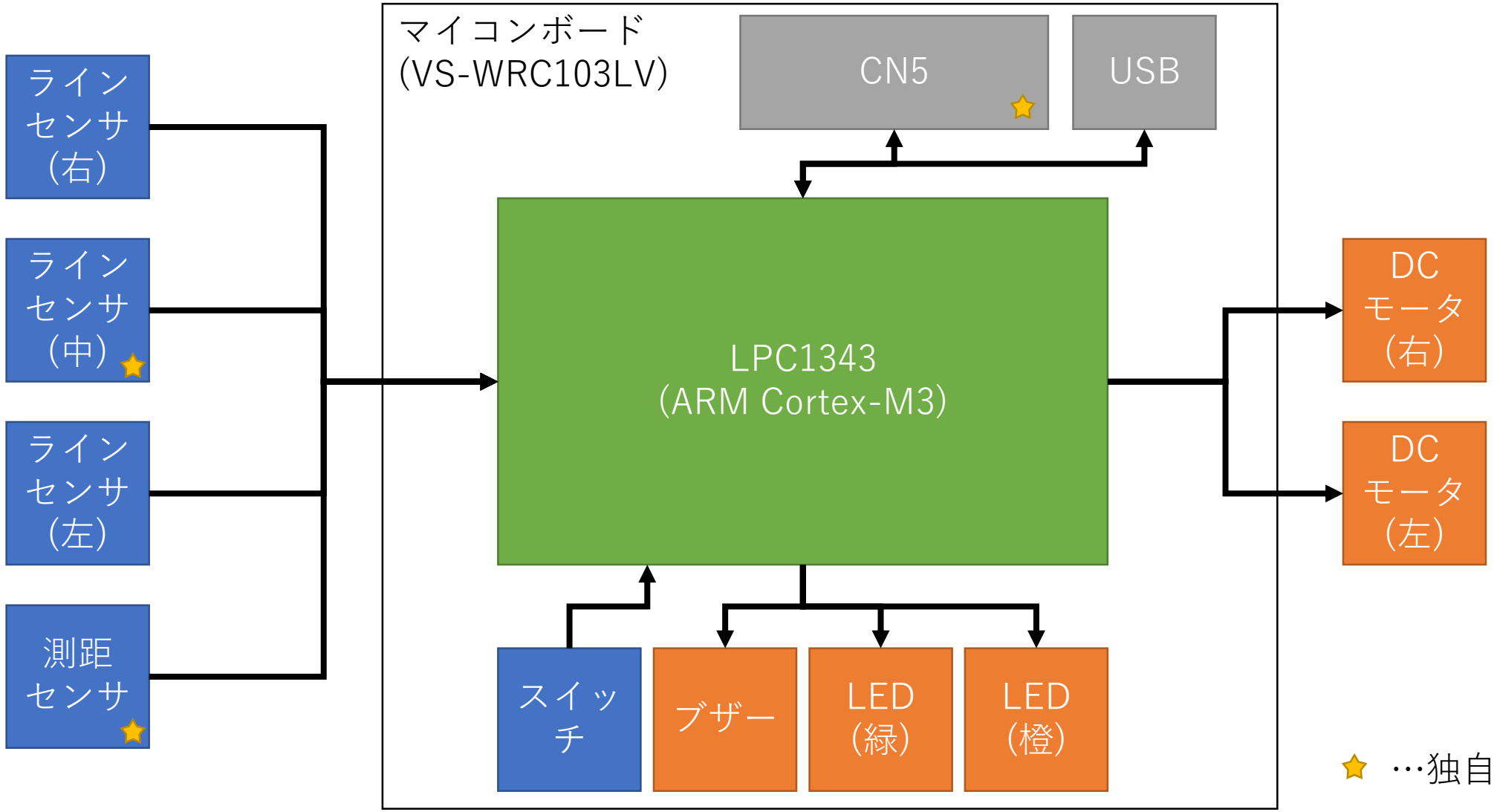
1. rei08_02.cの一部処理をまとめ、下記の関数を作成ください
    1. Whileループに組み込めば、周期20msec、デューティ比50%でモータ制御を行う関数となる

```
void _forward_20ms(void);
void _back_20ms(void);
void _turn_left_20ms(void);
void _turn_right_20ms(void);
```

課題④：LEDの輝度調整

1. rei08_02.cを改変し、フルカラーLEDの赤色の輝度調整ができるように修正してみて下さい
   - LEDはモータに比べて即応性が高い為、waitの時間調整が必要な可能性があります
2. フルカラーLEDの緑色を点灯させた状態で、１を確認してみて下さい
3. その他の色についても試してみて下さい

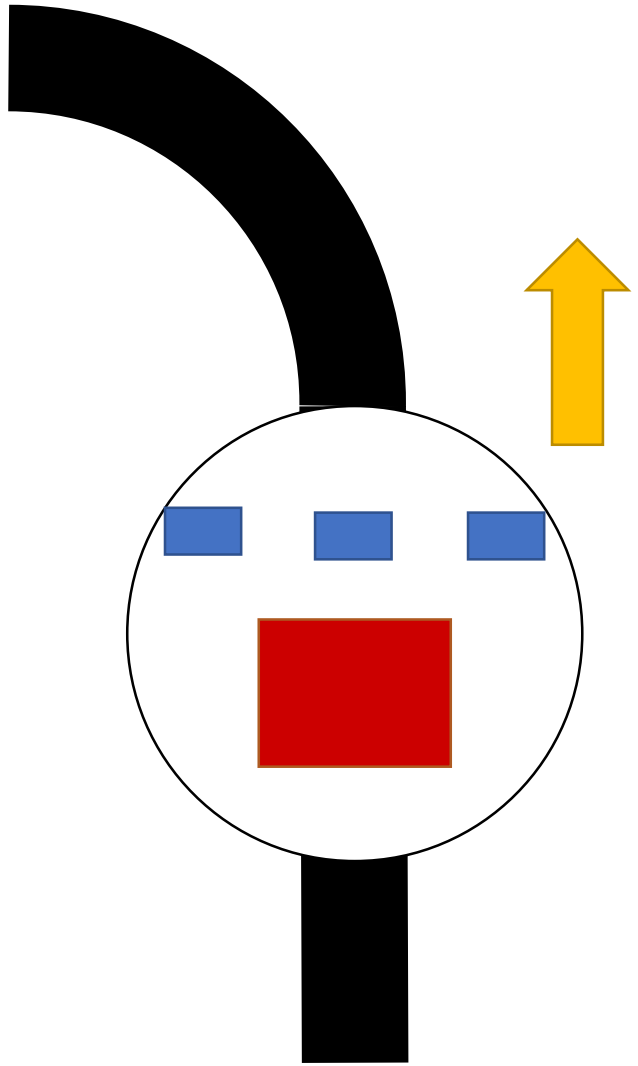
09:C言語によるライントーレス制御



ライントレース制御①

- ラインセンサを用いて線の上を走る
- 今回は中央のラインセンサは使わない

- ライントレースの基本アルゴリズム
  - 左のラインセンサがラインに反応した場合
    - 舵を右に切りながら走行する
  - 右のラインセンサがラインに反応した場合
    - 舵を左に切りながら走行する
  - どちらのセンサを反応しない場合
    - 直進する
  - 以上をひたすら繰り返す

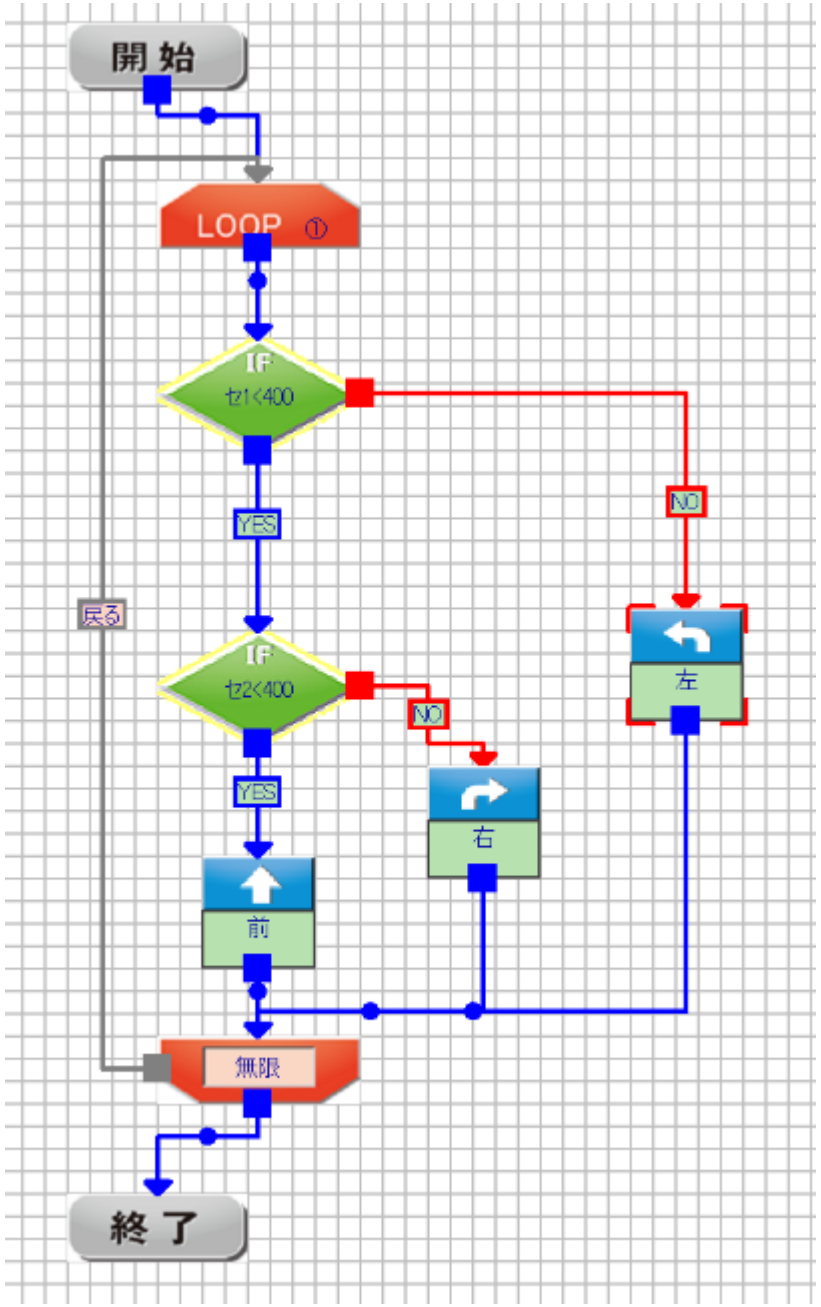


ライントレース制御②

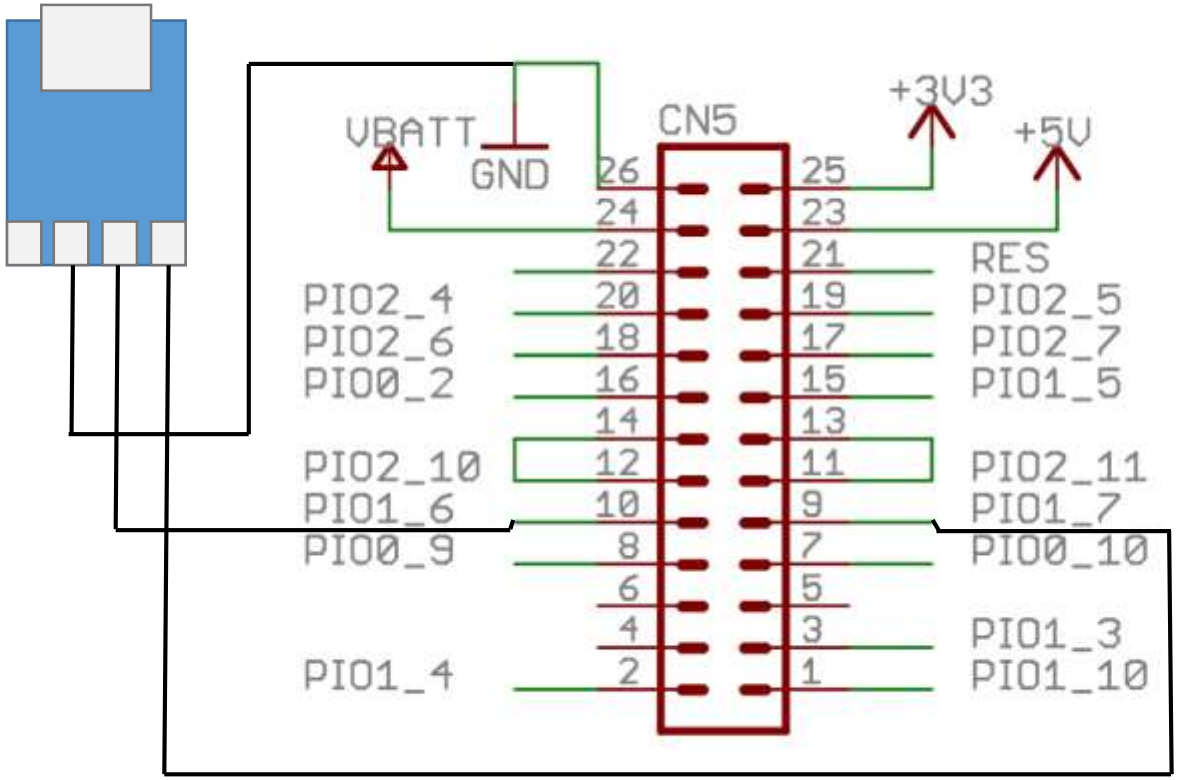
- ライン(白・黒)を識別しよう
- ラインが識別出来たら…
  - ラインセンサの値に合わせてLTCの動作(振舞い)を定めよう！
- Let’s Programing!!
  - ファイル名：kadai09_01.c

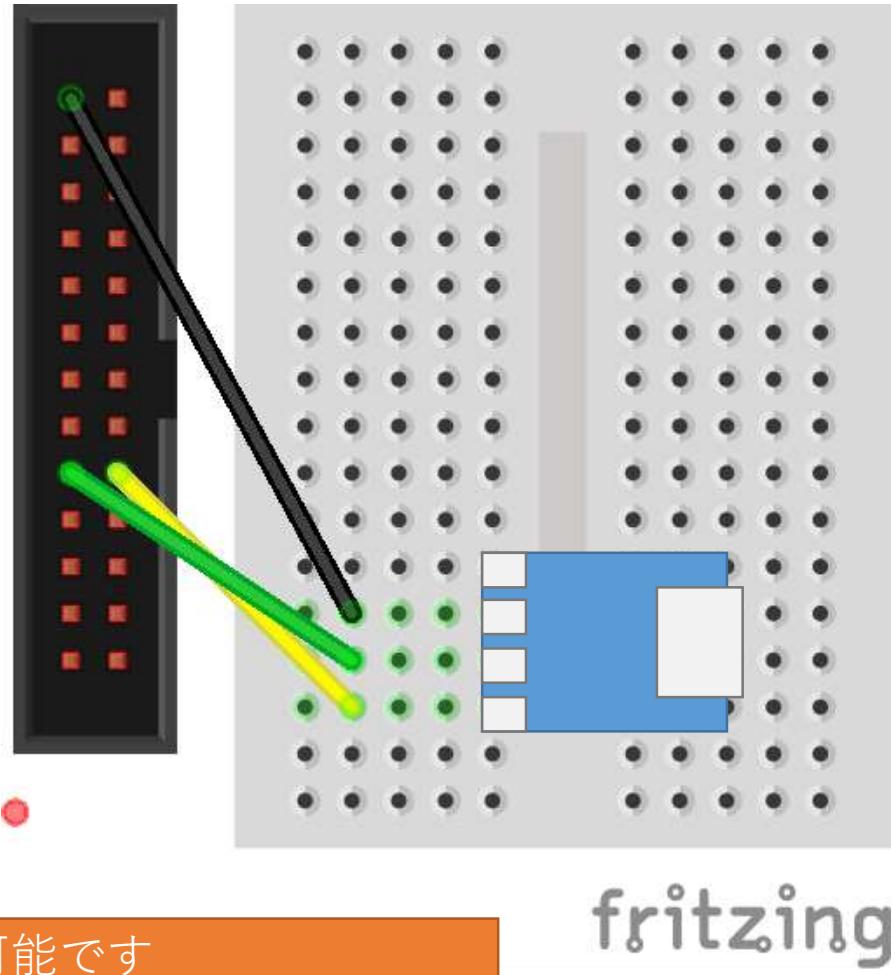
場所	ラインセンサ(左)	ラインセンサ(右)
ライン上(黒)		
それ以外(白)		

ラインセンサ(左)	ラインセンサ(右)	LTCの動作
白	白	
黒	白	
白	黒	



ヒント：ドック無しでシリアル通信





- PIO1_6はRxDと兼用ピン
- PIO1_7はTxDと兼用ピン

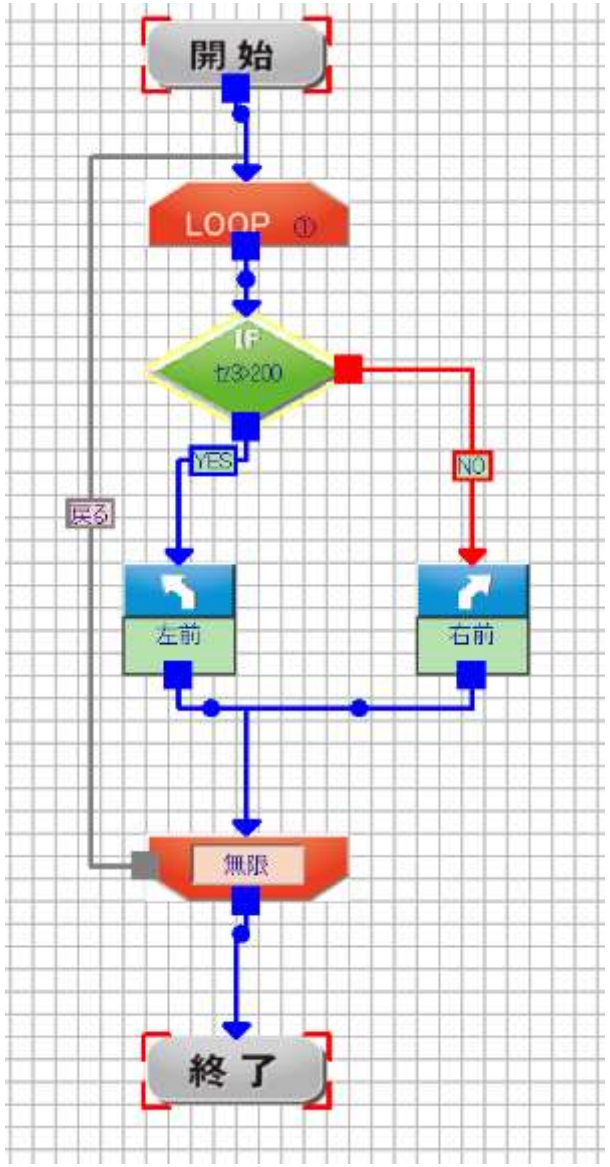
CN5からもシリアル通信に必要なTxD, RxDピンは取得可能です
PK-LTC用の拡張ブレッドボードと併用する事でドックと接続していなくても
シリアル通信が実現可能です

課題①：センサ１つでラインントレース

- ファイル名：kadai09_02.c
- ラインセンサ(中)だけを用いてライントレースするプログラムを実現しよう。

場所	ラインセンサ(中)
ライン上(黒)	
それ以外(白)	

ラインセンサ(中)	LTCの動作
白	
黒	

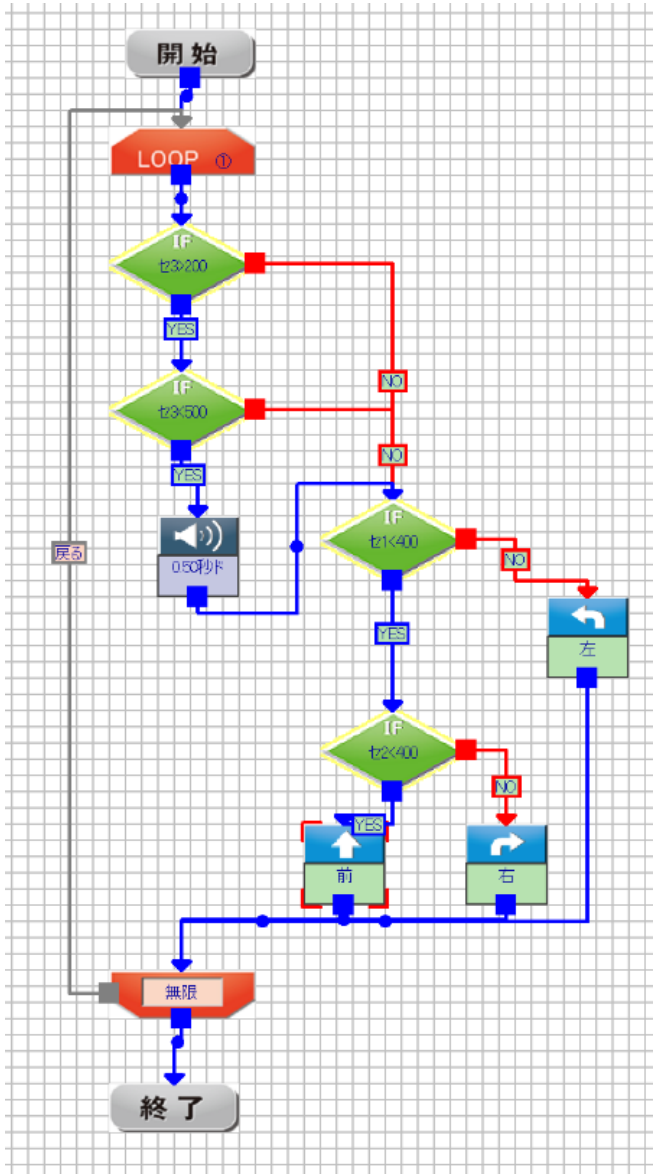


課題②：灰色を検知したら橙LED点灯

- ラインセンサ(中)で灰色のラインを検出したら、橙LEDを点灯させるプログラムを作成してください。ファイル名：kadai09_03.c

場所	ラインセンサ(左)	ラインセンサ(中)	ラインセンサ(右)
ライン上(黒)			
ライン上(灰)			
それ以外(白)			

ラインセンサ(左)	ラインセンサ(中)	ラインセンサ(右)	LTCの動作
白	－	白	
黒	－	白	
白	－	黒	
－	白	－	
－	灰	－	
－	黒	－	

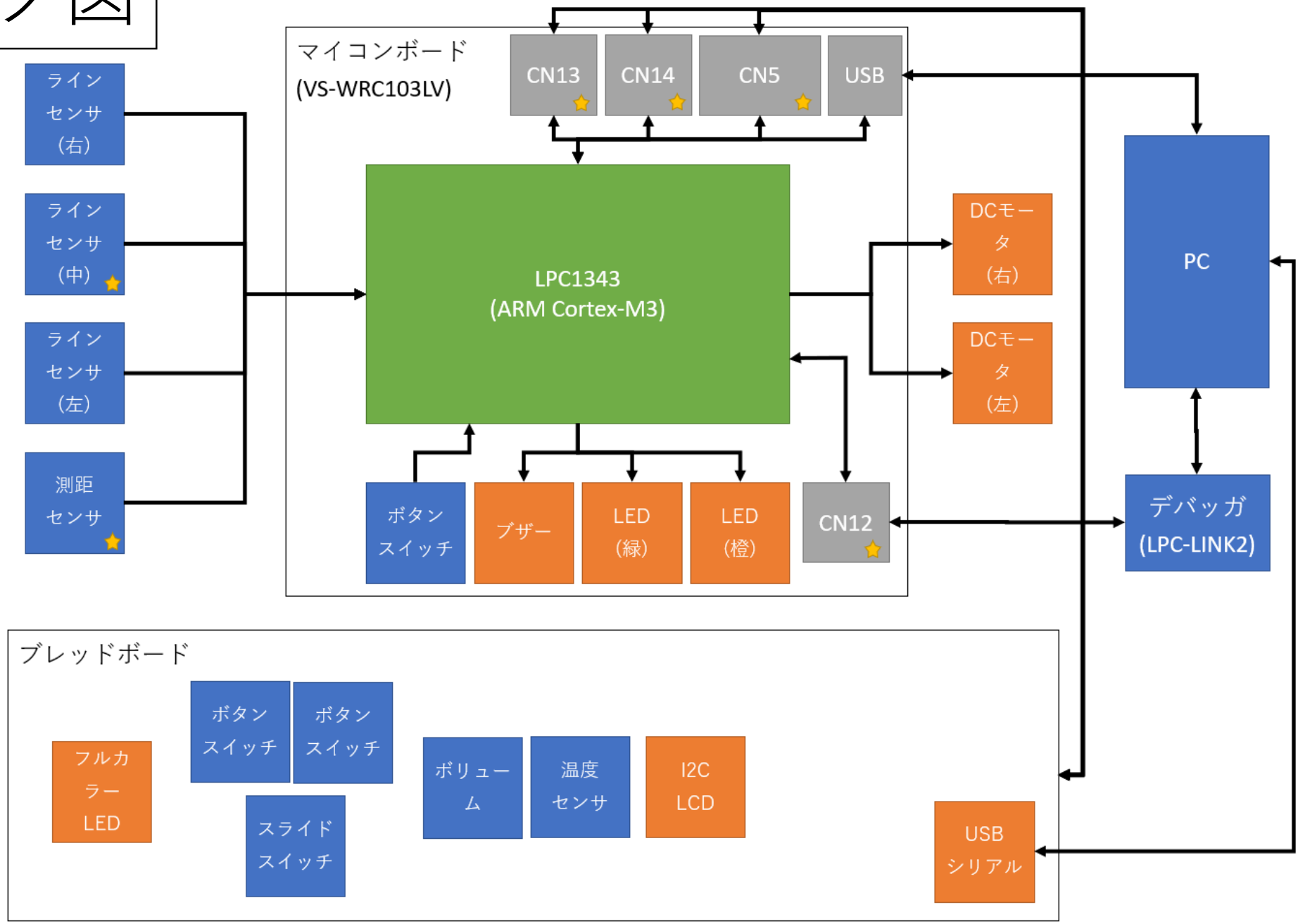


新たな課題の発見

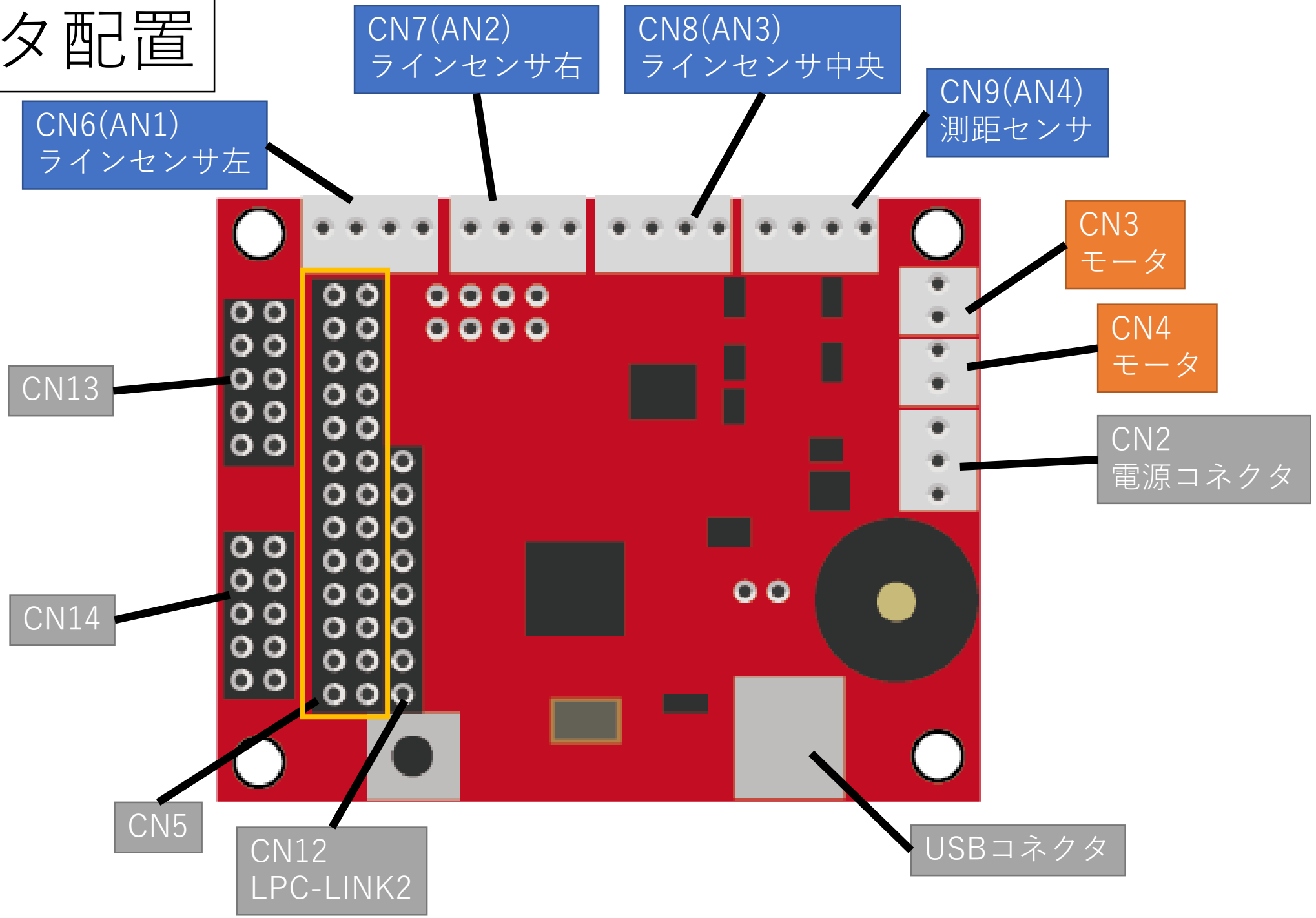
- ファイル名：kadai09_04.c
- 作成したライントレースのプログラムに、200ms周期で緑LEDを点滅させるコードを追加してみて下さい。
  - 挙動はどのように変わるでしょうか？

参考添付

ブロック図

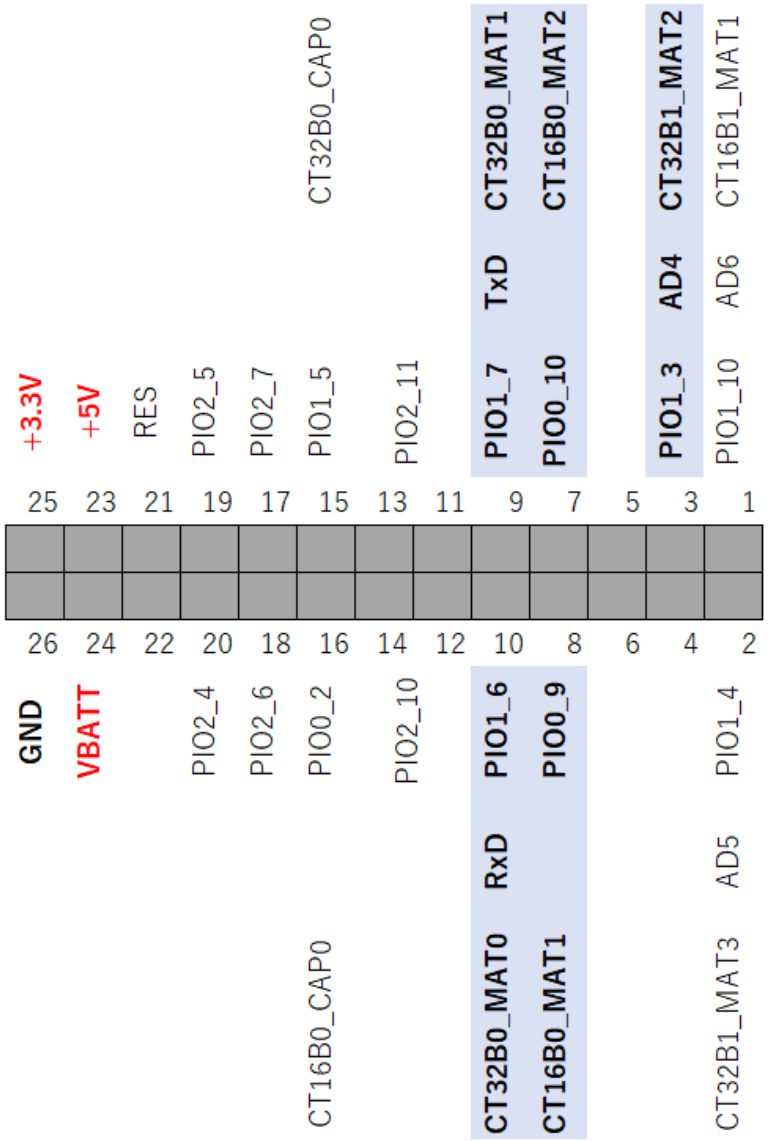

マイコンボード
(VS-WRC103LV)
CN13
CN14
CN5
USB
ライン
センサ
(右)
ライン
センサ
(中)
ライン
センサ
(左)
測距
センサ
LPC1343
(ARM Cortex-M3)
DCモー
タ
(右)
DCモー
タ
(左)
PC
ボタン
スイッチ
ブザー
LED
(緑)
LED
(橙)
CN12
デバッガ
(LPC-LINK2)
ブレッドボード
フルカ
ラー
LED
ボタン
スイッチ
ボタン
スイッチ
スライド
スイッチ
ボリュー
ム
温度
センサ
I2C
LCD
USB
シリアル

コネクタ配置


CN7(AN2)
ラインセンサ右
CN8(AN3)
ラインセンサ中央
CN9(AN4)
測距センサ
CN6(AN1)
ラインセンサ左
CN3
モータ
CN4
モータ
CN13
CN2
電源コネクタ
CN14
CN5
CN12
LPC-LINK2
USBコネクタ

拡張コネクタのピン配置図

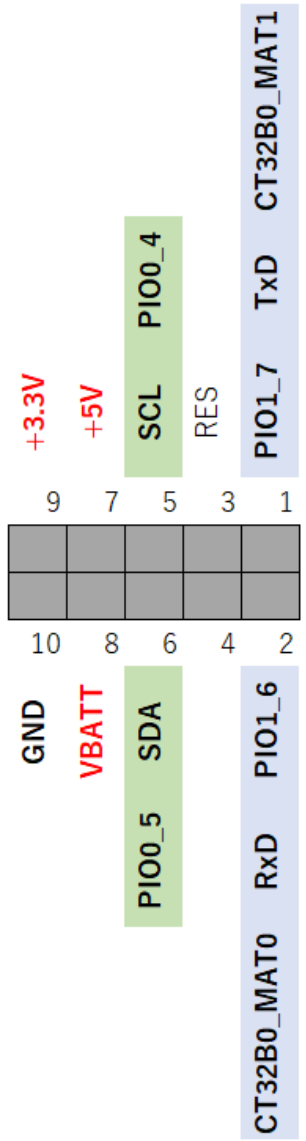
■ CN5



＊ 9,10pinはCN5の1,2pinと被ってる

＊ 3,7,8pinはCN12の2,3,4pinと被っている

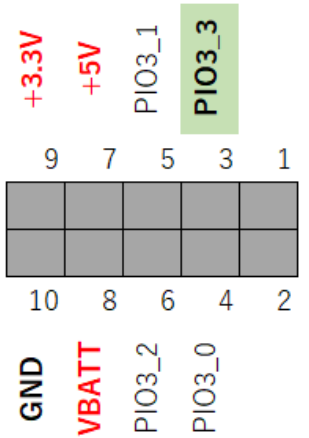
■ CN13



＊ 5,6pinは3.3KΩでプルアップ済み

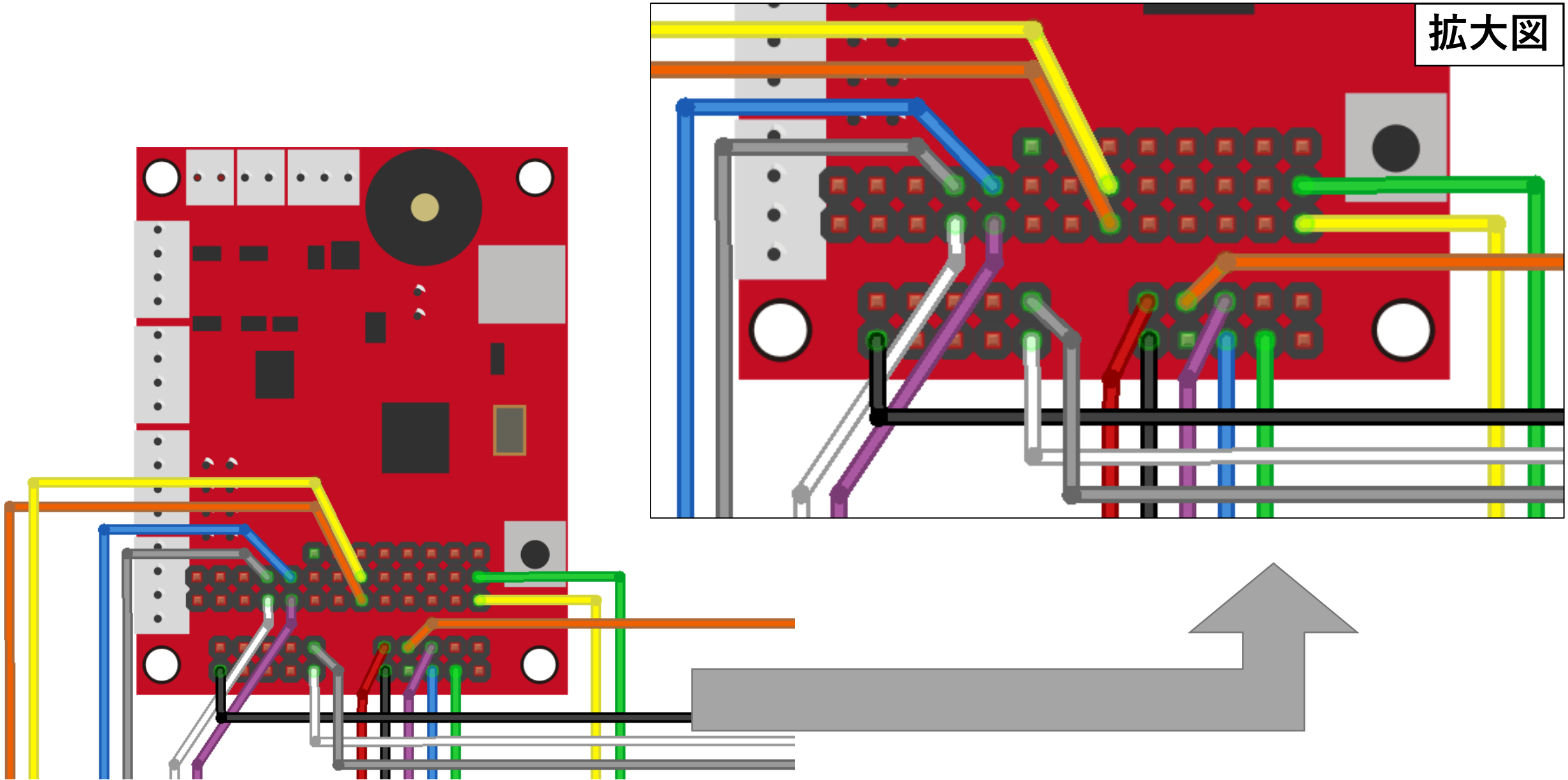
＊ 1,2pinはCN5の9,10pinと被ってる

■ CN14



＊ 3pinは33kΩでプルアップ済み

CN5, 13, 14への再接続時のチートシート



関連リンクほか

- ビュートローバーARM（ヴイストン社）
  - https://www.vstone.co.jp/products/beauto_rover/index.html
  - 本テキストに関する問い合わせについて、ヴイストン社は受け付けておりませんのでご注意ください。
- LPCXPresso(NXP社)
  - http://www.nxp-lpc.com/lpc_boards/lpcxpresso/
- ARMマイコンによる組込みプログラミング入門(オーム社)
  - http://shop.ohmsha.co.jp/shop/shopdetail.html?brandcode=000000001461&search=ARM%A5%DE%A5%A4%A5%B3%A5%F3

- Special Thanks
  - SSEST5(Summer School on Embedded System Technologies)