TOPPERS基礎実装セミナー

（Raspberry Pi Pico/Pico2(W)版：ハードウェア設計）

基礎編：1日目

TOPPERSプロジェクト

教育ワーキング・グループ

TOPPERS

本ドキュメントに関して

1. 著作権に関しての表記

<TOPPERS基礎ハードウェア設計セミナー(Raspberry Pi Pico(W)/Pico2(W)版：ハードウェア設計）1日目>

Copy right (C) 2015-2025 by TOPPERSプロジェクト教育WG
Copy right (C) 2015-2025 by 竹内良輔

上記著作権者は、以下の（1)～(3）の条件を満たす場合に限り、本ドキュメント（本ドキュメントを改変したものを含む。以下同じ）を使用・複製・改変・再配布（以下、利用と呼ぶ）することを無償で許諾する。
(1) 本ドキュメントを利用する場合には、上記の著作権表示、この利用条件および以下の無保証規定が、そのままの形でドキュメント中に含まれていること。
(2) 本ドキュメントを改変する場合には、ドキュメントを改変した旨の記述を、改変後のドキュメント中に含めること。ただし、改変後のドキュメントがTOPPERSプロジェクト指定の開発成果物である場合には、この限りではない。
(3) 本ドキュメントの利用により直接的または間接的に生じるいかなる損害からも、上記著作権者およびTOPPERSプロジェクトを免責すること。また、本ドキュメントのユーザまたはエンドユーザからのいかなる理由に基づく請求からも、上記著作者およびTOPPERSプロジェクトを免責すること。
本ドキュメントは、無保証で提供されているものである。上記著作権者およびTOPPERSプロジェクトは、本ドキュメントに関して、特定の使用目的に対する適合性も含めて、いかなる保障もしない。また、本ドキュメントの利用により直接的または間接的に生じたいかなる損害に関しても、その責任を負わない。

---

2. 本ドキュメントに関するご意見・ご提言・ご感想・ご質問等がありましたら、TOPPERSプロジェクト事務局までE-Mailにてご連絡ください。

3. 本ドキュメントの内容は、内容の改善や適正化の目的で予告無く改定することがあります。

本ドキュメントでは、Microsoft社のClip Art Galleryコンテンツを使用しています。

TRONは”The Real-time Operating system Nucleus”の略称です。ITRONは”Industrial TRON”の略称です。μITRONは”Micro Industrial TRON”の略称です。TOPPERS/JSPはToyohashi Open Platform for Embedded Real-Time System/Just Standard Profile Kernelの略称です。」

本ドキュメント中の商品名及び商標名は、各社の商標または登録商標です。

TOPPERS

スケジュール

- １日目

1.	はじめに	0.2時間
2.	開発環境の説明	0.5時間
3.	デバッガとLEDの点灯確認	1.5時間
4.	ブザーとキーを動かそう	1.0時間
5.	LCDを動かそう	1.0時間
6.	カップラーメンタイマソフト説明	1.0時間
7.	まとめ	0.1時間

はじめに

ハードウェア知識の必要性

- 近年、ハードウェアロジックのSoC化、FPGAを使ったロジック設計により、ボード上に複雑な回路設計を行う設計が減ってきている
- 多機能なワンチップマイコンと中低速のシリアルインターフェイスで、いろいろな周辺モジュールを制御することができるようになった、しかも、ブレッドボード上の実装でも十分な品質で通信が可能となった
- 変わりに、SoCの複雑なペリフェラル制御をソフトウェア制御する要求が増えてきている
- Arduinoの普及でホビーとしての組込みが認知されてきた、また、Arduinoコネクタに新規の技術を乗せた種々のシールドが販売され、簡単に新しいハードウェアを試せるようになった

ハードウェア知識の必要性

- 逆に、Arduinoシールドに対応したエバレーション・ボードが多く出されている
- Arduino IDE以外で、シールドを制御するためにハードウェア制御よりの知識が必要となる
- 回路設計者はドライバ等のソフトウェア設計技術が要求され、ソフトウェア設計者も品質の高いプログラム実装を行うためにハードウェアペリフェラルの設計知識が要求されている
- 注意：この講座はTOPPERS BASE PLATFORMを使用するため基礎２終了後に受講してください

まず、ブレッドボードを使ってハードウェア設計にチャレンジしてみよう

開発環境の説明

1. マイコンボードの概要
2. ハードウェア環境の構築
3. ソフトウェア環境の構築
4. プログラムの開発方法

マイコンボードの概要

マイコンボードと搭載プロセッサついて解説する

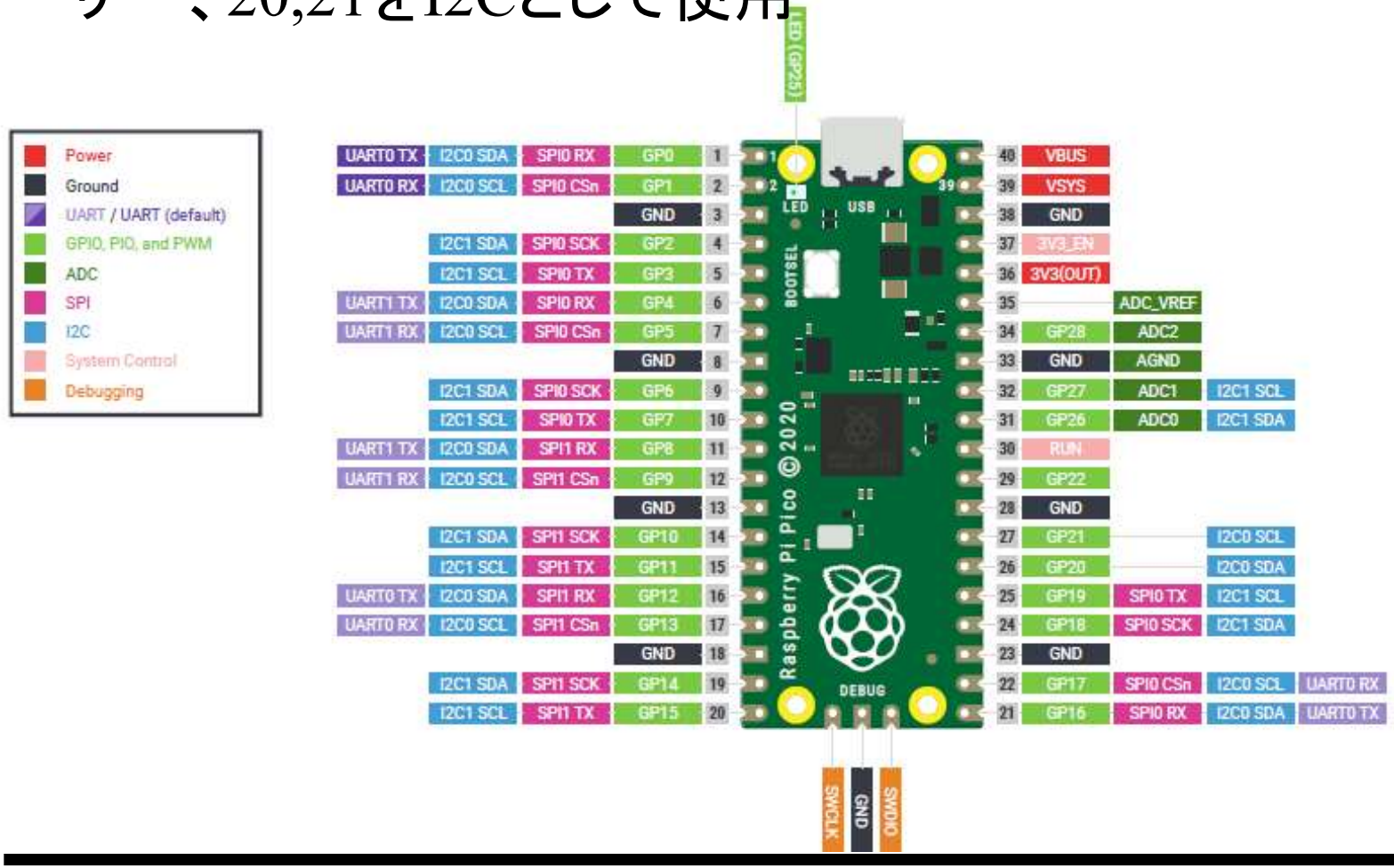
- RP2040はARM Cortex-M0+ 2コアとしたSoCで、ラズベリーパイ財団が独自に開発しました
- 製品型番：RP2040/SC0914
- パッケージ：56pin QFNパッケージ
- CPU：デュアルコア ARM Cortex-M0+ @133MHz
- 内部RAM：264KB SRAM
- Flashメモリ：なし、Quad SPIは
- I/O電圧：3.3V
- 電源電圧：3.3V

RP2040概要

- インターフェースとして以下を持ちます
  - 1×USB 1.1（デバイス・ホストモードに対応）
  - 30×GPIO（内4本がアナログ対応）
  - 2×UART
  - 2×I2C
  - 2×SPI
  - 16×PWM（8×A/Bチャネル、Bは入力対応）
  - 8×PIO
  - 5×ADC 12bit 500kサンプル/秒（ひとつは内部温度センサーに接続）
  - 1×タイマー（4×アラート機能付き）
  - 1×RTC
  - 1×ウォッチドッグタイマー
  - 1×内部温度センサー
  - 1×3ピンARMシリアルワイヤデバッグ(SWD)ポート

PICOのピン設定

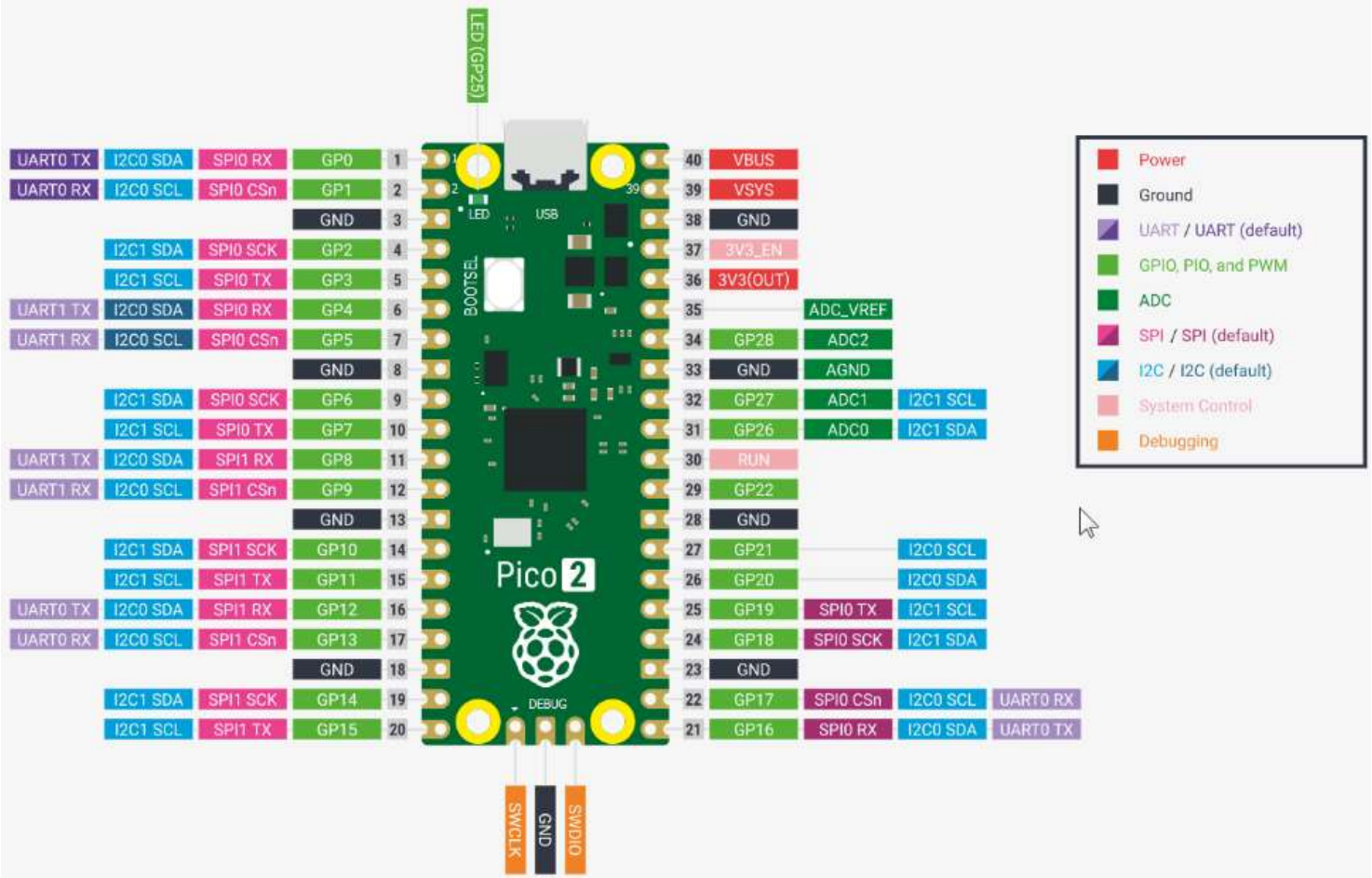
- Raspberry Pi Picoのピンレイアウトは以下の通り
- GP0,1をUART、19をLED、8,9をプッシュスイッチ、18をブザー、20,21をI2Cとして使用



TOPPERS

Pico2のピン設定

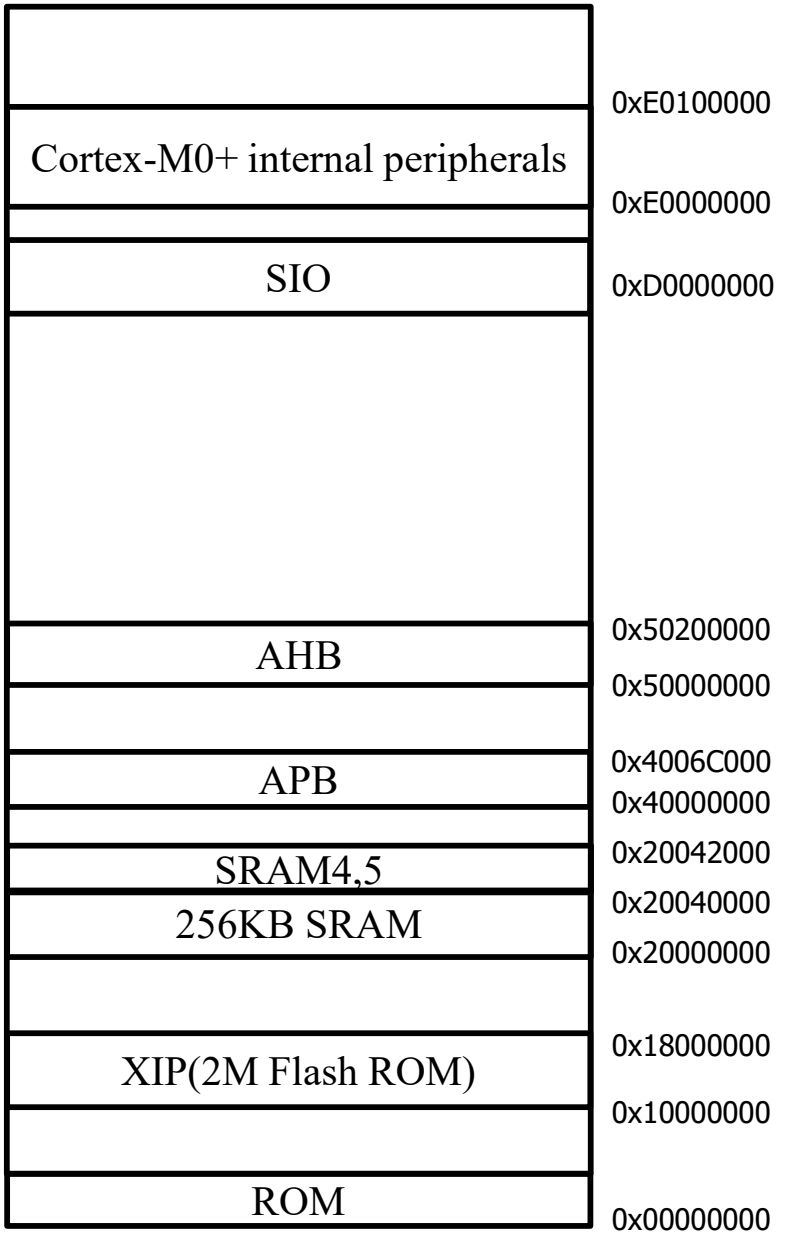
- Raspberry Pi Pico2のピンレイアウトはPicoと同様
- GPIOピン、UART、I2CもPicoと同様です



TOPPERS

RP2040：メモリマップ

- 2MBのFlash ROM領域にプログラムを置き、256KBのSRAM領域をデータとしてプログラム開発を行う
- 今回、演習用にROMモニタを使用する
- Flash ROM領域にROMモニタプログラムを置き、SRAM中の0x2003F000～0x2003FFFFの4KBをROMモニタのデータ領域とし
- 0x2000000～0x2003EFFFをダウンロード領域とする



TOPPERS

Cortex-Mのレジスタ構成

- 実行モードはスレッドモード（特権モードと非特権モード）、ハンドラモード３つ
- CPSRレジスタは３つのレジスタに分解された
  - APSR,IPSR,EPSR
- FIQ割込みはなくなり、割込み発生時、スタック上に使用するレジスタをハードウェアでPUSHする
- 割込み復帰はBX命令のエクセプションモードで行われる

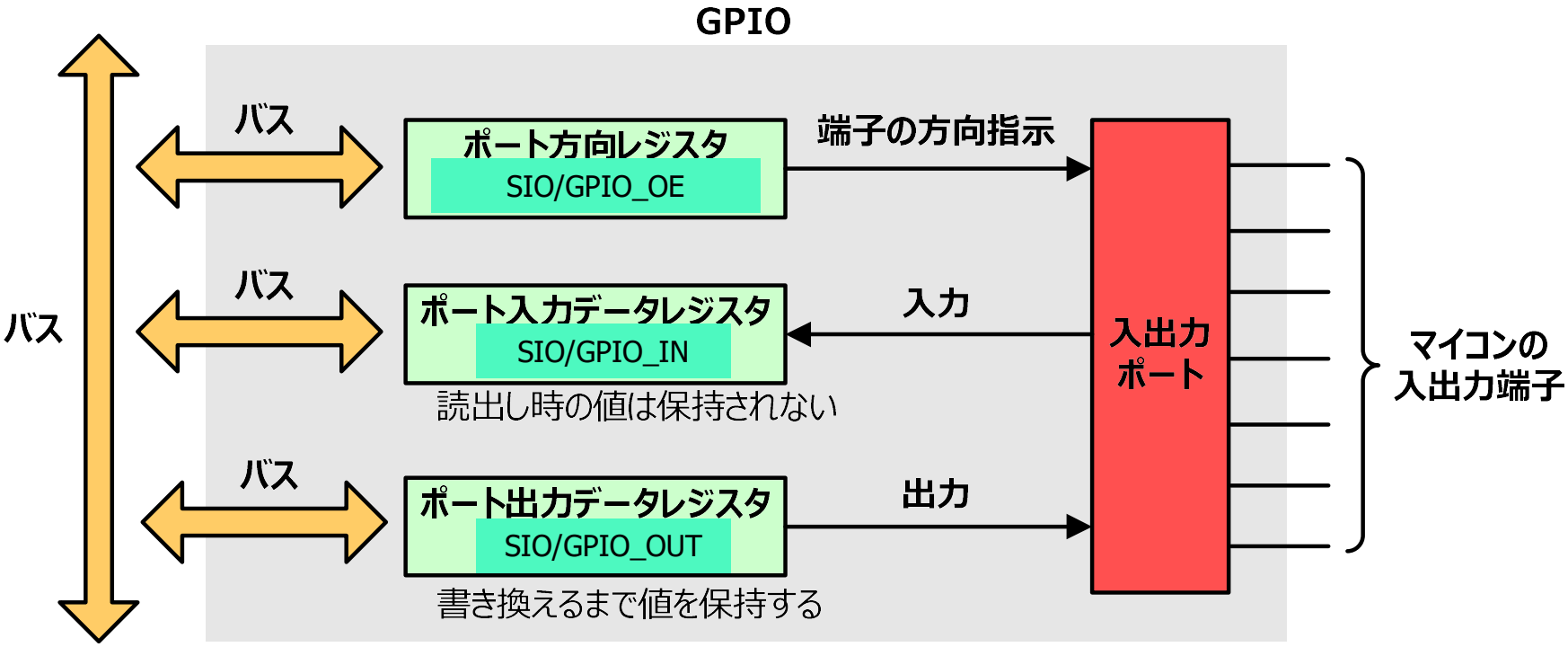
ユーザー	特権モード
R0	R0
R1	R1
R2	R2
R3	R3
R4	R4
R5	R5
R6	R6
R7	R7
R8	R8
R9	R9
R10	R10
R11	R11
R12	R12
R13	R13_svc
R14	R14
PC	PC

特権モード移行時R13（SP）のみ変更となる

ARSP	APSR
IPSR	IPSR
EPSR	EPSR

TOPPERS

GPIO（General Purpose Input / Output）の説明



- 汎用I/Oポート。LSI端子を介してデジタル信号の入出力を行なう。
- CPUと外部装置でデータの受け渡しをするために、プログラムで扱うデジタル値（0/1）と信号（LOW/HIGH）の変換を相互に実施する。

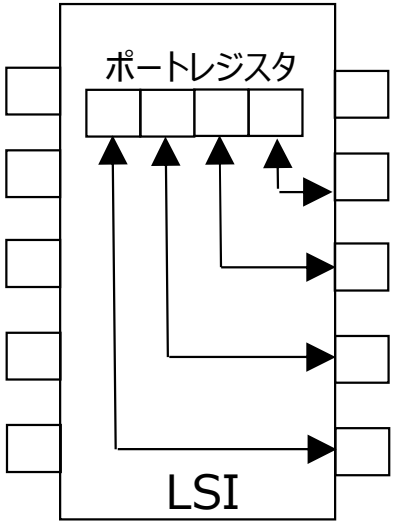
GPIOの説明

・General Purpose Input / Output（汎用入出力）
・LSIの端子（ポート）を介してデジタル信号の入出力を行なうデバイス

入力ポート
- ポートからの入力が、ポートレジスタに反映される

出力ポート
- ポートレジスタの設定内容が、ポートから出力される

コントロールレジスタ
- 各ポートを出力ポートとして使うか、入力ポートとして使うかなどを設定する
- ビット単位もしくは、あるグループ単位毎に設定可能



開発環境の説明

1. マイコンボードの概要
2. ハードウェア環境の構築
3. ソフトウェア環境の構築
4. プログラムの開発方法

ハードウェア環境の構築

- Pico/Pico2以外の必要機材は以下の通り
  - 部品の詳細は「TOPPERS BASE PLATFORM開発環境編」に記載

①ジャンパーワイヤー
②ジャンパーコード
③圧電ブザー
④LED
⑤タクトスイッチ
⑥LCD(I2C接続)
⑦ブレッドボード
⑧抵抗、コンデンサ

開発環境の説明

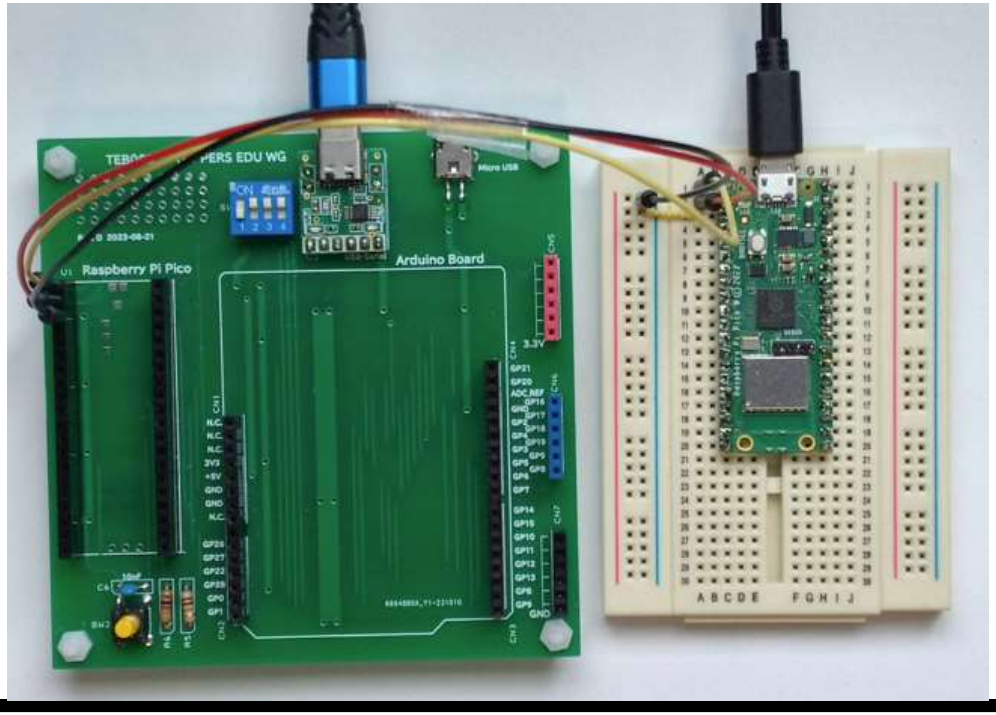
1. マイコンボードの概要
2. ハードウェア環境の構築
3. ソフトウェア環境の構築
4. プログラムの開発方法

ソフトウェア環境の構築

- 開発環境
  - MSYS2　　：プログラム開発環境
  - GCC-ARM　　：クロスコンパイラ
  - RISC-Vコンパイラ　　：クロスコンパイラ(Pico2/RISC-V)
  - Tera Term　　：シリアル通信ターミナル
  - TOPPERS BASE PLATFORM 1.5.0
- 各ツールの提供Webからダウンロード、インストール
- その他のソフトウェア
  - テキストエディタ
  - PDFリーダー（Acrobat Reader等）

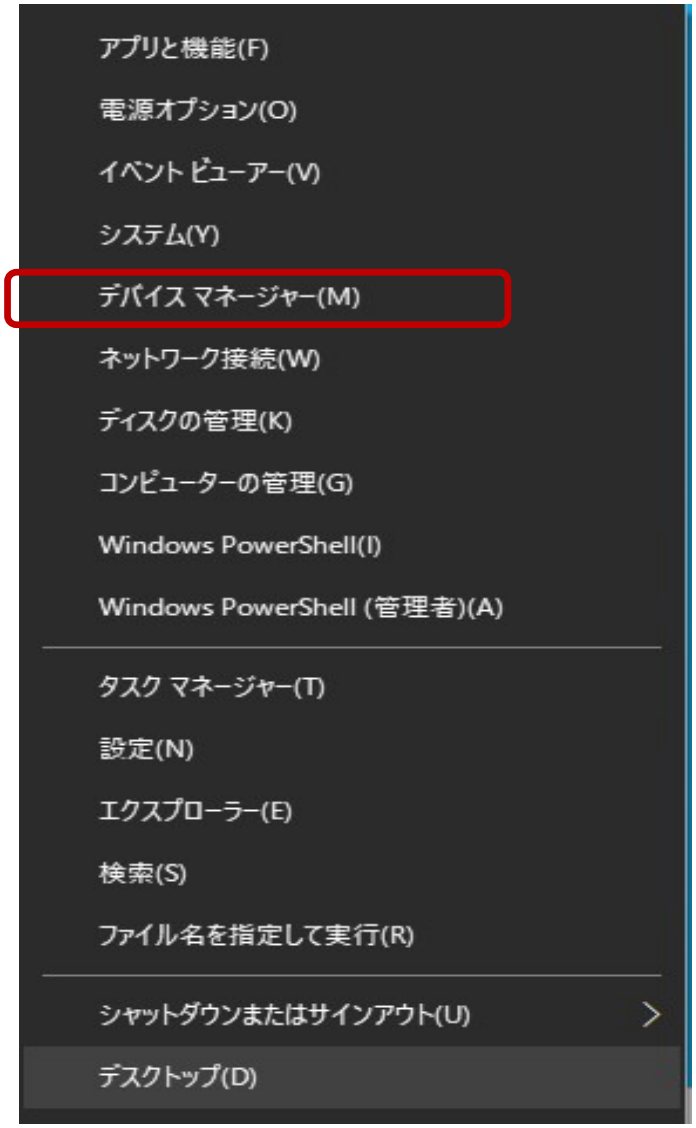
USBシリアルの接続

- TEB003のUSBシリアルを使って、パソコンと通信
  - ジャンパーケーブル３本をTEB003のCN1の1,2,3ピンに接続
    - Pico 1ピン(RX)　- TEB003 CN1 1ピン
    - Pico 2ピン(TX)　- TEB003 CN1 2ピン
    - Pico 3ピン(GND)　- TEB003 CN1 3ピン



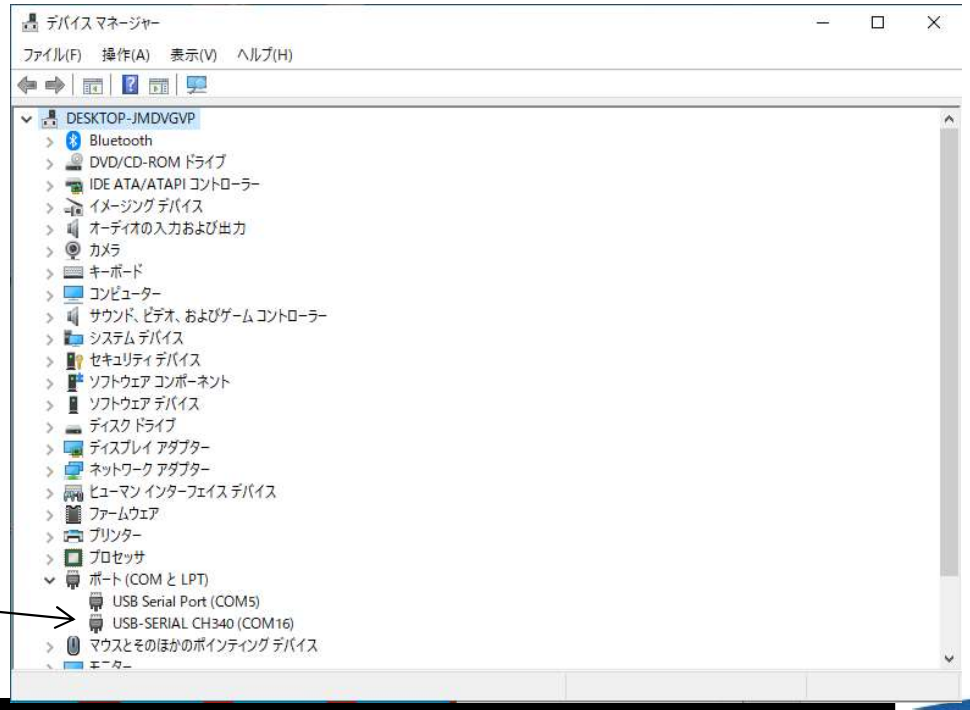
USB-シリアルのCOMポート設定

- TEB003をPCに接続
- COMポートの確認
  - “スタート”のマウスの右クリックで管理用メニューを表示し、“デバイスマネージャー”を起動する



USB-シリアルのCOMポートの確認

- “ポート（COM と LPT）”を開く
  - USB Serial Port の項目にCOMポートが出現する
  - 番号は環境によって異なる
  - 番号をメモしておく
  - 後述のデバッガの設定に必要

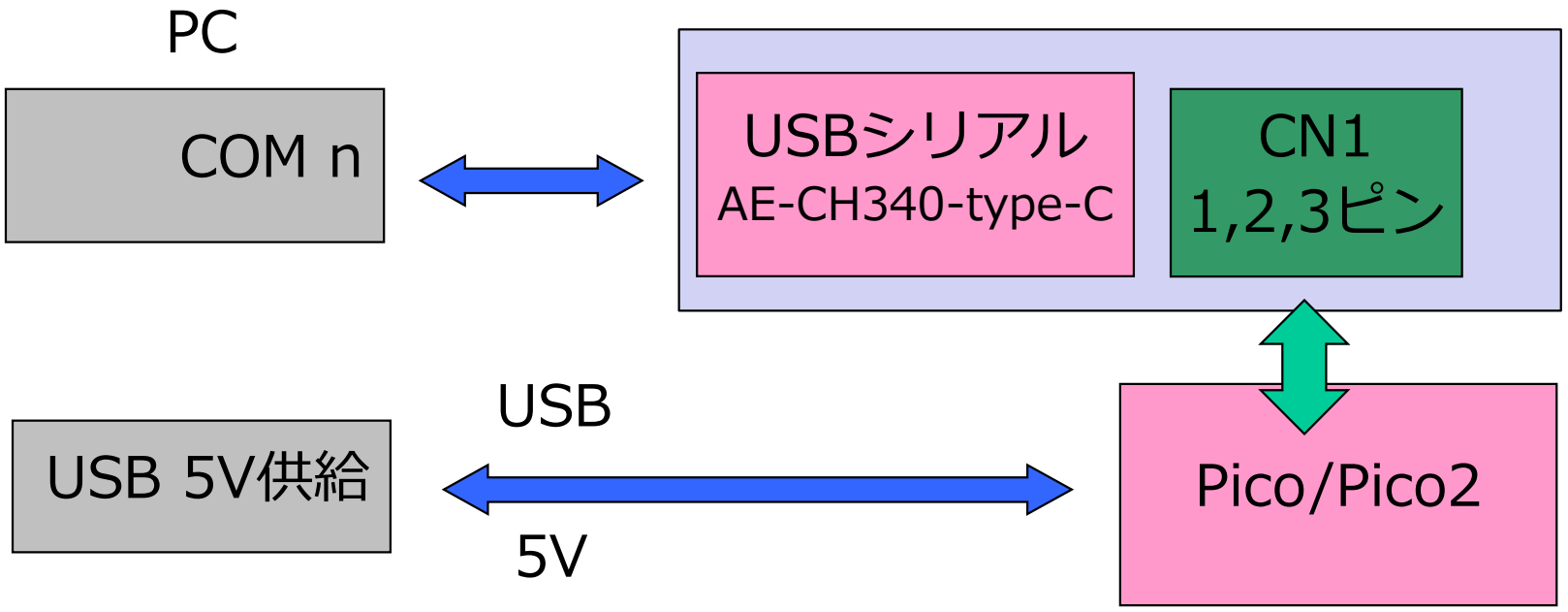


COM番号

TOPPERS

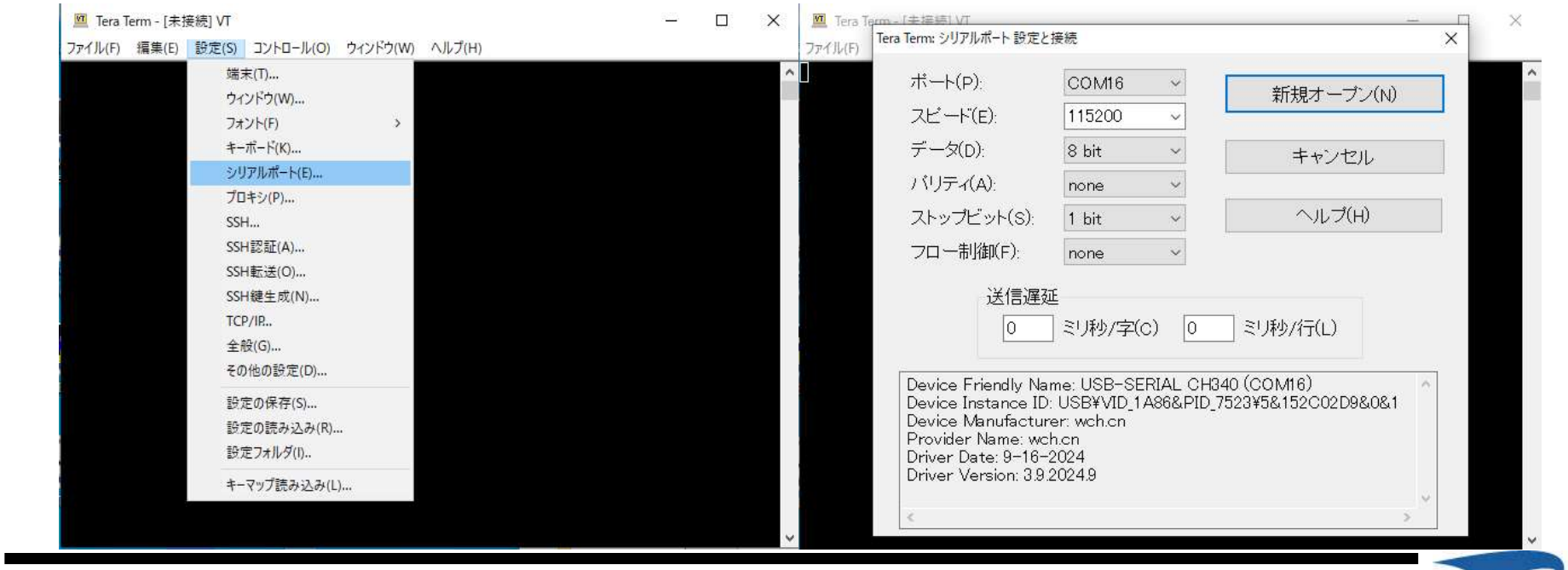
USBシリアルを使ったデバッグ・シリアル通信

- USBシリアル変換ケーブルをPCと接続するとPicoのUART0 USBシリアル変換器を介してPCと接続される
- PicoのUSBを接続すると5Vが供給され、Picoが起動
  - Picoへの5V供給はPCからでなくても良い



USB-シリアル：TeraTermの設定

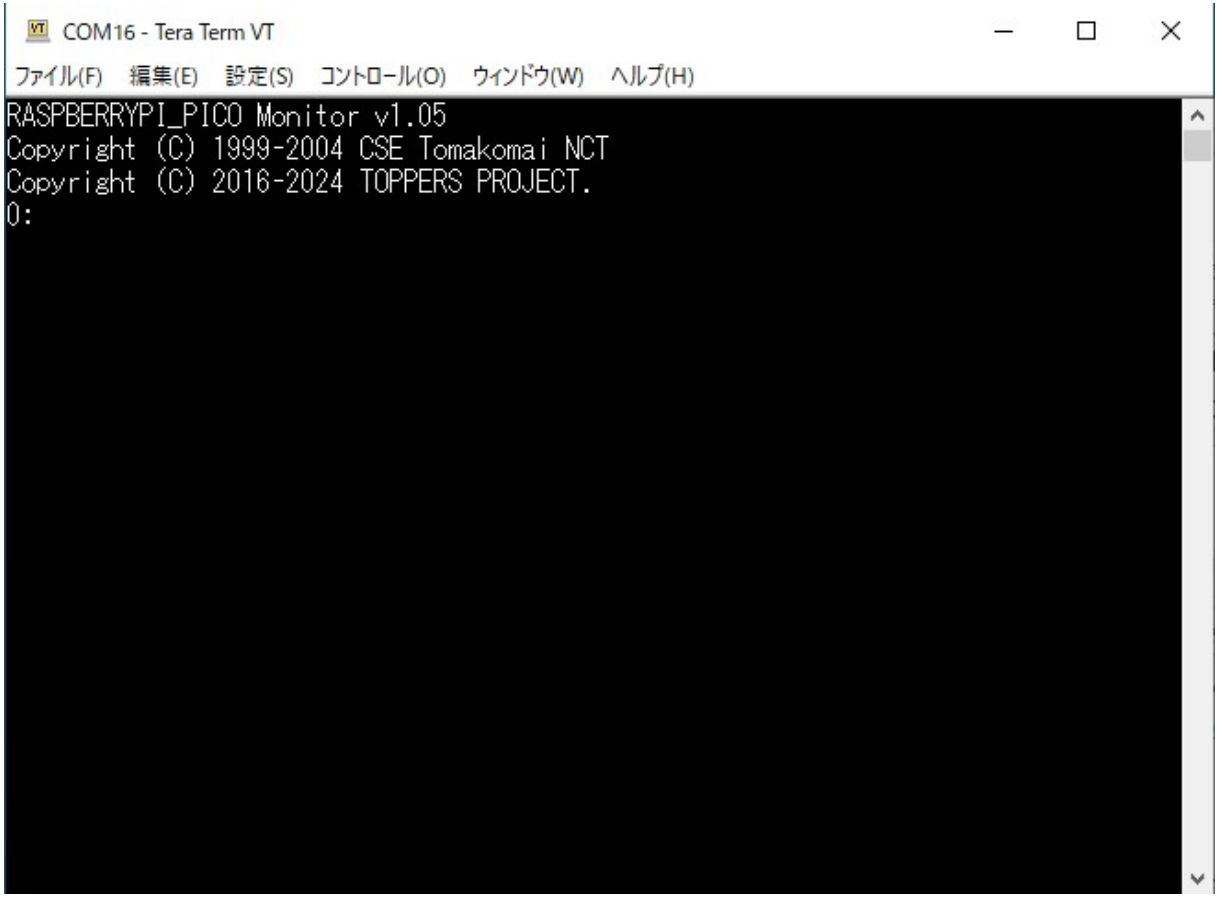
- 割当てCOMポートの変更が必要な場合
  - 設定からシリアルポート(E)…を開く
  - デバイス・ドライバで確認したCOMポードを選択
  - スピード(E)を15200に設定



TOPPERS

USB-シリアル：PicoのUSBを接続

- PicoのUSBをPC等に接続すると、電源が供給され、基礎１でROMモニタを書き込んでいればROMモニタのバナー表示が表示される



TOPPERS

開発環境の説明

1. マイコンボードの概要
2. ハードウェア環境の構築
3. ソフトウェア環境の構築
4. プログラムの開発方法

フラッシュメモリへの書き込み

- 教材でのプログラムの開発は基礎１実装セミナーの開発環境を用いて行う

- MakefileでコンパイルスイッチのDBGENVがROMに設定されていれば、ROM実行用のプログラムを生成する、それ以外（デフォルト）ならばRAM実行用のプログラムを生成する
  - ROM実行用のプログラムは、UF2ファイルをフラッシュROMに書込み実行する
  - RAM実行用のプログラムは、あらかじめROMモニタを書き込んでおいて、Tera Termを経由して、プログラムをダウンロードして実行する
- 本教材では、RAM実行でプログラムを実行する形で説明を行う

デバッガとLEDの点灯確認

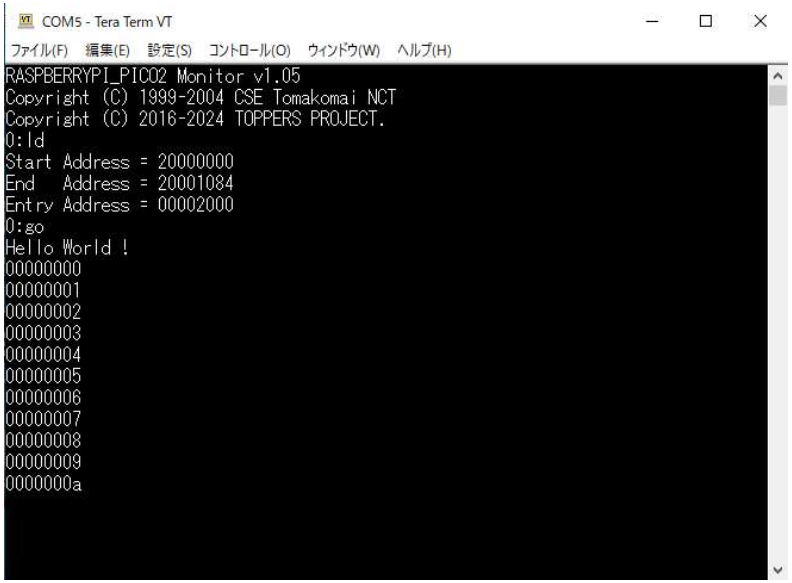
1. 開発環境のビルドと実行
2. LED点滅プログラムの確認
3. ブレッドボードの実装方法
4. LEDを配置する

sys_printf文

- 基礎１実装セミナーの開発環境では、コンパイルスイッチDEBUGを定義することで、sys_printf文の記載を、シリアル端末に表示することができる
  - DEBUGが未定義の場合、表示用のすべてのプログラムがリンク対象外となり、メモリの削減が可能
- プログラムが思い通り動作しない場合、変数等を表示させて確認を行える

```
void
main(void){
    int count = 0;

    sys_printf("Hello World !¥n");
    for (;;) {
        sys_printf(“%08x¥n“, count);
        count++;
        busy_wait();
    }
}
```



基礎1sampleプログラムでのログ表示

ポーリングプログラミング：プログラムファイル

- プログラムファイルの置き場所
  - 教材ディレクトリ/baseh/program
- プログラム一覧
  - led_blink　　：LED点滅プログラム
  - switch_push　：PUSHスイッチプログラム
  - lcd_test　　：キャラクタLCD表示テストプログラム
  - cup_timer　　：カップラーメンタイマー

ビルドと実行：開発環境の起動

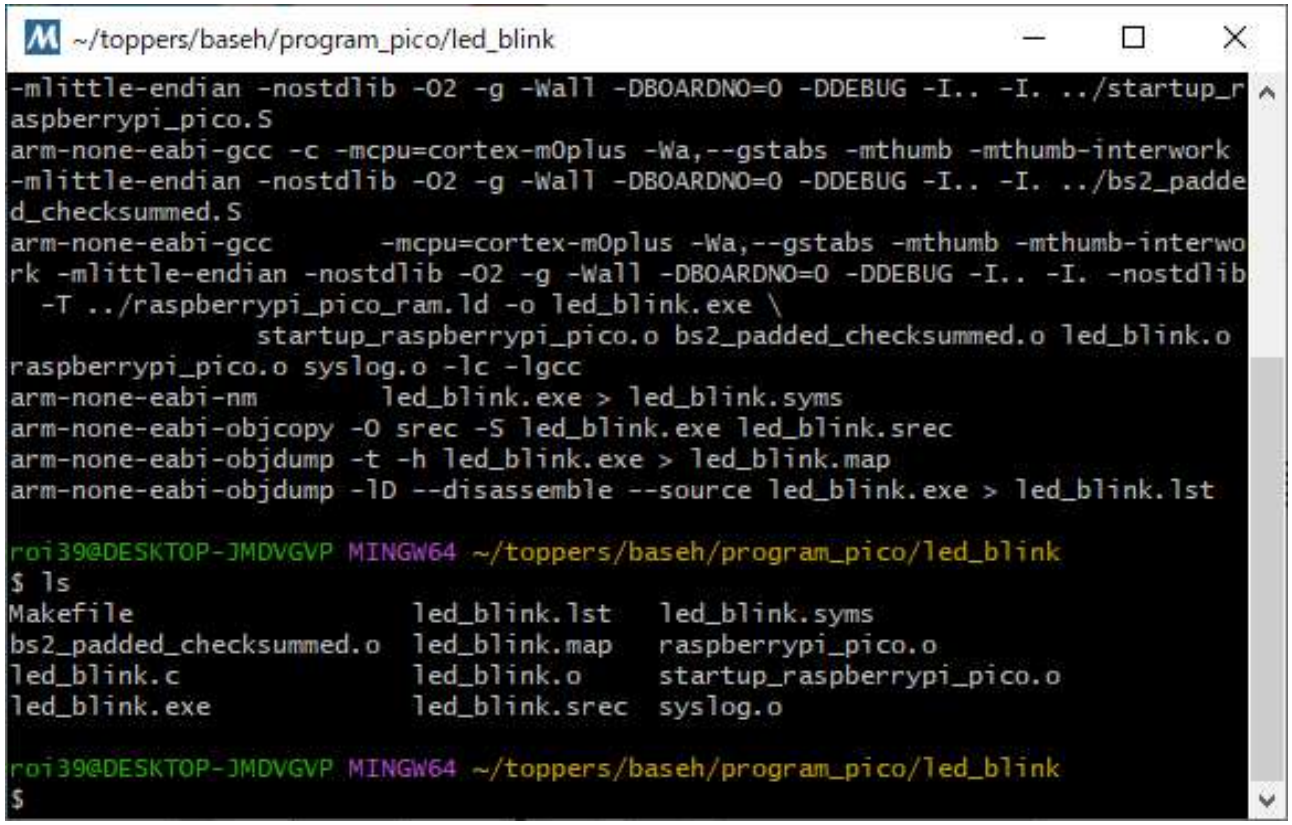
開発環境の動作確認のため，
サンプルプログラムをコンパイルする

- MSYS2のホームディレクトリにtoppersディレクトリを作り、教材プログラムbasehをコピーする
  - MSYS2を起動し、toppers/baseh/led_blinkに移動する
  - make DEBUG=1(return)でサンプルプログラムをビルドする

make BOARDNO=0	BOARDNOは省略可能、Picoをビルド
make BOARDNO=1	Pico2:Cortex-M33をビルド
make BOARDNO=2	Pico2:RISC-Vをビルド

ビルドと実行：開発環境の起動

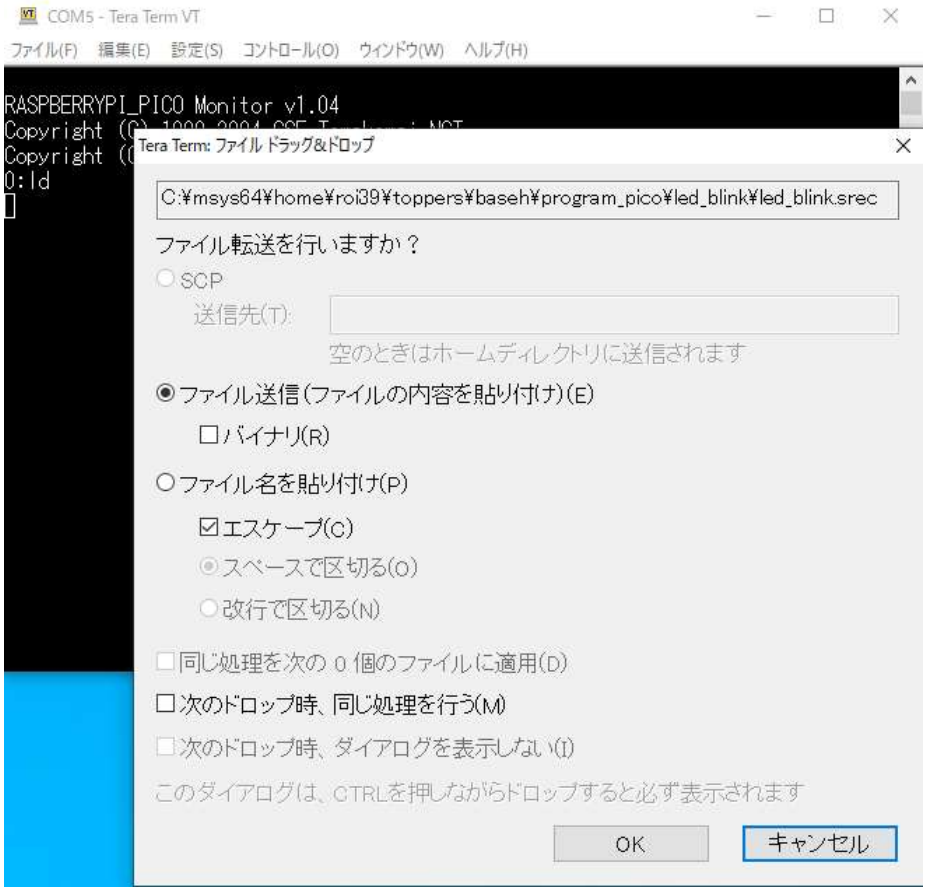
- 実習用のプロジェクトファイルを選択
  - /program/led_blink/led_blink.srec



TOPPERS

ダウンロード

- Raspberry Pi Picoには、ROMモニタを書き込んでおく
- USBコネクタでPCに接続し、リセットボタンで、ROMモニタのプロンプトを表示
- ldコマンド後、led_blink.srecをTera Termにドロップしてダウンロード開始

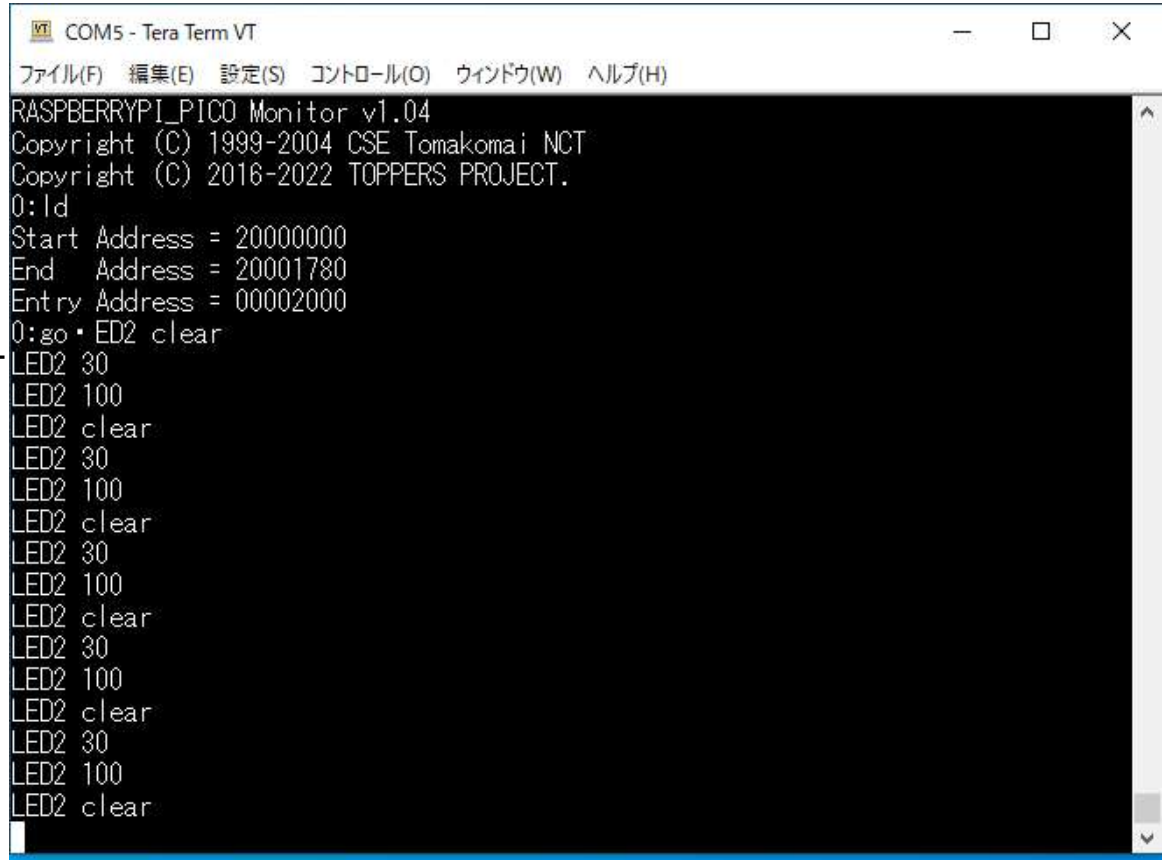


TOPPERS

実行

- ダウンロード終了後、goコマンドでプログラムを実行
- Tera TermにLEDのログ、ボード上のLEDが点滅する

LED2は
以下の表示で30%点灯
  LED2 30
以下の表示で100%点灯
  LED2 100
以下の表示で消灯
  LED2 clear



デバッガとLEDの点灯確認

1. 開発環境のビルドと実行
2. LED点滅プログラムの確認
3. ブレッドボードの実装方法
4. LEDを配置する

LEDプログラム：led_brink概要

- 次のパターンでLEDを点灯させる
  - LED全てOFF → LED2 30%ON → LED2 100%ON → LED全てOFF

- 学習内容
  - LEDデバイスの確認
  - LEDプログラムの確認
  - スタートアップルーチン

LEDデバイスの確認

- ボード上のユーザーLEDはGPIO25に接続している
- GPIO19：ジャンパーピンにてLEDにセット
  - LED1⇔GPIO25：使用しない
  - LED2⇔GPIO19：PWMを設定する

LEDプログラム：led_blink概要

- 生成ファイル
  - led_blink.c　　メイン関数
  - Makefile　　メイクファイル
- 共通ファイル
  - ../Makefile.common　　Makefileの環境設定ファイル
  - ../sil.h　　SIL関数定義
  - ../startup_rp2040.S　　スタートアップモジュール：アセンブラ
  - ../sys_defs.h　　システム定義ファイル
  - ../SysConfig.c　　C言語スタートアップルーチン
  - ../syslog.c　　ログ表示ドライバファイル
  - ../t_syslog_h　　ログ表示インクルードファイル

LEDプログラム：led_blink概要

- Picoファイル
  - ../bs2_padded_checksummed.S パディングソースファイル
  - ../rp2040.h rp2040マップドIO定義
  - ../rp2040_ram.ld PicoのRAM実行リンクスクリプト
  - ../rp2040_rom.ld PicoのROM実行リンクスクリプト
  - ../startup_rp2040.S アセンブラのスタートアップルーチン
- Pico2ファイル
  - ../rp2350.h rp2350マップドIO定義
  - ../rp2350_ram.ld Cortex-m33のRAM実行リンクスクリプト
  - ../rp2350_rom.ld Cortex-m33のROM実行リンクスクリプト
  - ../rp2350rv_ram.ld RISC-VのRAM実行リンクスクリプト
  - ../rp2350rv_rom.ld RISC-VのRAM実行リンクスクリプト
  - ../startup_rp2350.S Cortex-M33アセンブラのスタートアップルーチン
  - ../startup_rp2350rv.S アセンブラのスタートアップルーチン
  - ../riscv_int.c RISC-V用割込み関数
  - ../entry.S RISC-V用割込みハンドリングソースファイル

led_blink:main関数

- 後述するスタートアップルーチンから呼び出される

PWMリセット

LEDをPWMで初期化

ウェイト

```
void
main(void){

    /* ハードウェア初期化 */
    pwm_init();
    led_init();
    for (;;) {
        sys_printf("LED2 clear¥n");
        led_out_sil(0x00);
        busy_wait();

        sys_printf("LED2 30% ¥n");
        led_out_sil(30);
        busy_wait();
        busy_wait();

        sys_printf("LED2 100% ¥n");
        led_out_sil(100);
        busy_wait();
    }
}
```

LED全消灯

LED2 30%点灯

LED2 100%点灯

TOPPERS

led_blink:ポートレジスタ書込み関数

- LED点灯・消灯制御
  - led_dataはPWMのデューティー
  - calc_count関数でデューティーをタイマーカウントに変換し設定
  - ポートのアドレスはrp2040.h/rp2350.h”(sys_defs.hに定義)を参照

```
/*
 *  LED接続ポート書き込み
 */
/*
 * アクセス関数使用版
 */

void
led_out_sil(uint32_t led_data){
    uint32_t ct = get_pwm_count(LED_PIN);
    uint32_t base = TADR_PWM_BASE + (get_pwm_channel(LED_PIN) * PWM_CHANNEL_SIZE);
    uint32_t count;

    led_data = (led_data * MAX_PWM_COUNT) / 100;
    count = calc_count(ct, sil_rew_mem((uint32_t *)(base+TOFF_PWM_CH_CC)), led_data);
    sil_wrw_mem((uint32_t *)(base+TOFF_PWM_CH_CC), count);
}
```


TOPPERS

led_blink：初期化関数led_init

- ポートをPWM設定し、ポートに対応したPWMモジュールの設定を行う

```
void
led_init(void){
    pwm_setup(LED_PIN, 8);
}
```

```
void
pwm_setup(uint8_t gpio, uint8_t div){
    uint32_t ct = get_pwm_count(gpio);
    uint32_t base, count;

    pinmux_setup(gpio, IO_BANK0_GPIO_CTRL_FUNCSEL_VALUE_PWM);

    base = TADR_PWM_BASE + (get_pwm_channel(gpio) * PWM_CHANNEL_SIZE);
    sil_wrw_mem((uint32_t *)(base+TOFF_PWM_CH_TOP), MAX_PWM_COUNT);
    sil_wrw_mem((uint32_t *)(base+TOFF_PWM_CH_CSR), PWM_CH_CSR_DIV_FREE_RUNNING);
    count = calc_count(ct, sil_rew_mem((uint32_t *)(base+TOFF_PWM_CH_CC)), MAX_PWM_COUNT);
    sil_wrw_mem((uint32_t *)(base+TOFF_PWM_CH_CC), count);
    sil_wrw_mem((uint32_t *)(base+TOFF_PWM_CH_DIV), (((uint32_t)div << 4) | 0));
    sil_wrw_mem((uint32_t *)(base+TOFF_PWM_CH_CSR+REG_ALIAS_SET), PWM_CH_CSR_EN);
}
```

ピンをPWM設定する

PWM設定

TOPPERS

led_blink:ビジーウェイト関数busy_wait

- forループにより任意時間待ちを生成
- ループ回数はforループ1回の実行時間から求める
  - 実際には適当にためして決定する

```
#define DELAY_LOOP (250*1000)

void
busy_wait(void)
{
    volatile int i;
    for(i = 0; i < DELAY_LOOP; i++)
        sil_dly_nse(1000);
}
```

最適化を抑制するため
volatile修飾子を付ける

TOPPERS

led_blink：startup_rp2040.S

- Reset_Handler
  - リセットで最初に実行される「スタートアップルーチン」
  - CPU制御レジスタの設定
    - 組み込み関数を使用している（sys_defs.hをインクルード）
  - リセット後のアセンブラ記述でC言語用にRAMを初期化する
  - main関数の呼び出し

led_blink : startup_rp2040.S Reset関数

- リセットで最初に実行される「スタートアップルーチン」
- 制御レジスタの設定と変数、ハードウェアの初期化を行いmain関数を実行する
- ハードウェアの初期化はhardware_init_hookをコール

```
#define TOPPERS_MACRO_ONLY
#include "sys_defs.h"
～中略～
    .section .text.Reset_Handler
    .weak   Reset_Handler
    .type    Reset_Handler, %function
Reset_Handler:
～中略～
      b LoopCopyDataInit
CopyDataInit:
      ldr  r3, =_sidata
      ldr  r3, [r3, r1]
      str  r3, [r0, r1]
      adds r1, r1, #4
LoopCopyDataInit:
～中略～
      bl  main
LoopForever:
      b LoopForever
```

C言語用のRAM初期化
ハードウェアの初期化

main関数を呼び出す

TOPPERS

led_blink : startup_rp2040.S 固定ベクタ

- Cortex-M0+はリセット後，フラッシュROMの先頭番地から始まる固定ベクタの＋４に置かれた番地から実行を開始する
- ベクタテーブルはアセンブラ言語（C言語でもよい）の配列として作成
- リンク用に特定のセクション名を設定する
- セクションはリンカのセクション設定にアドレスと共に配置
- リセット後，Reset_Handler（スタートアップルーチン）にジャンプ

Reset_Handlerのアドレス

```
    .section  .isr_vector,"a",%progbits
    .type     g_pfnVectors, %object
    .size     g_ptnVectors, .-g_pfnVectors
g_ptnVectores:
    word   _estack
    word    Reset_Handler
    word    NMI_Handler
    word    HardFault_Handler
    word    0
    word    0
    word    0
    word    0
    word    0
    word    0
    word    0
    word    0
    word    SVC_Handler
```


TOPPERS

led_blink : SysConfig.c C言語初期化関数

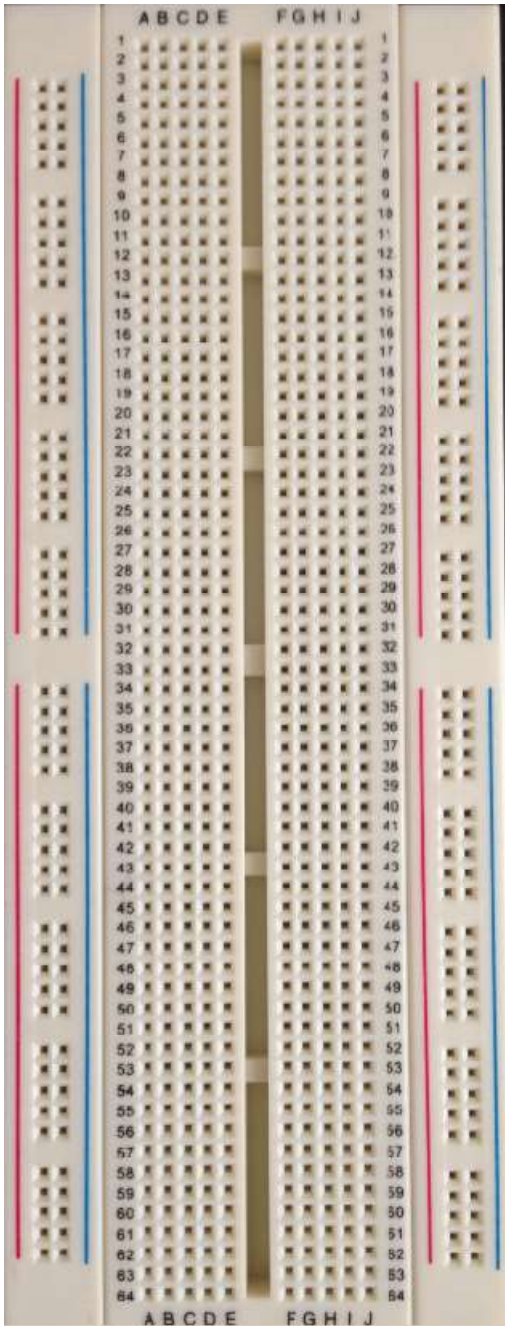
- RP2040/RP2350用ハードウェアとUARTの初期化関数
  - SystimInit
- エクセプション発生時のエラーハンドラ
  - NMI_Handler等
- UARTキャラクタ出力関数
  - sys_putc：キャラクタ１文字をUARTに出力

デバッガとLEDの点灯確認

1. 開発環境のビルドと実行
2. LED点滅プログラムの確認
3. ブレッドボードの実装方法
4. LEDを配置する

ブレッドボードの説明（1）

- ブレッドボードとは、**半田を使用せずに**各種電子部品やジャンパ線をボードの穴に差し込むだけで電子回路を手軽に実現できる基板



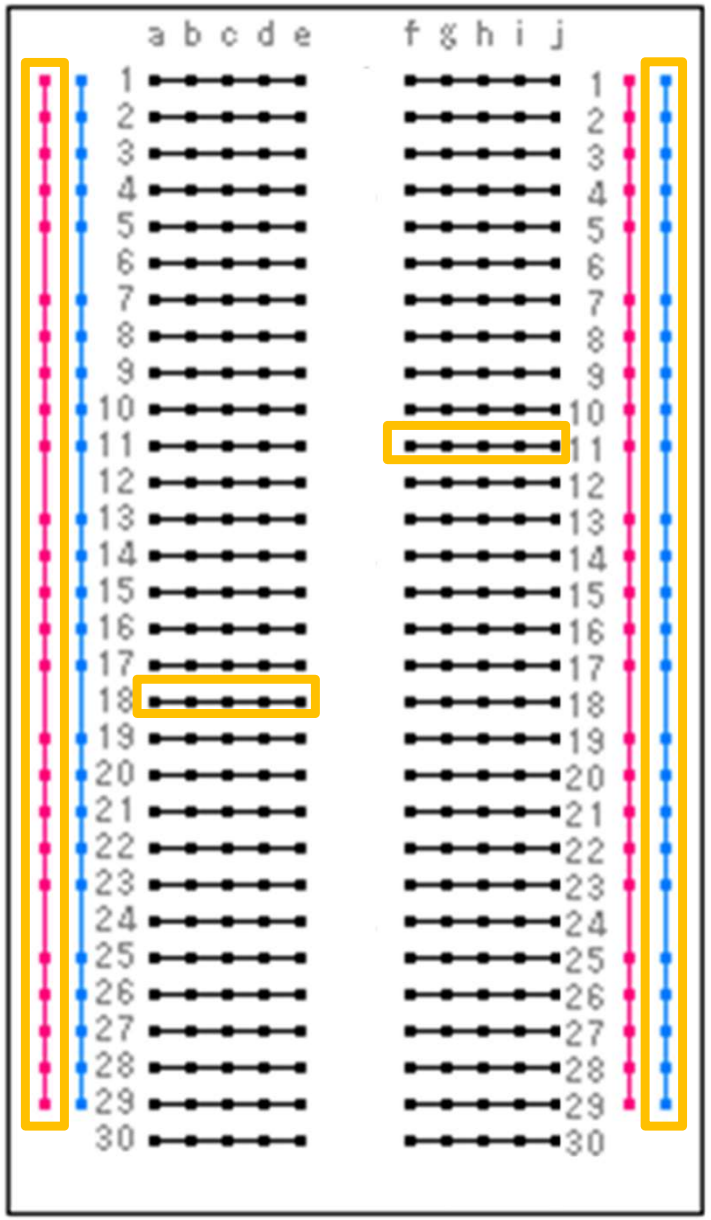
TOPPERS

ブレッドボードの説明（2）

それぞれのボードの穴の接続は
予め決まっている
→**穴を線で繋いだ部分が導通**
※黒は横方向、赤青は縦方向

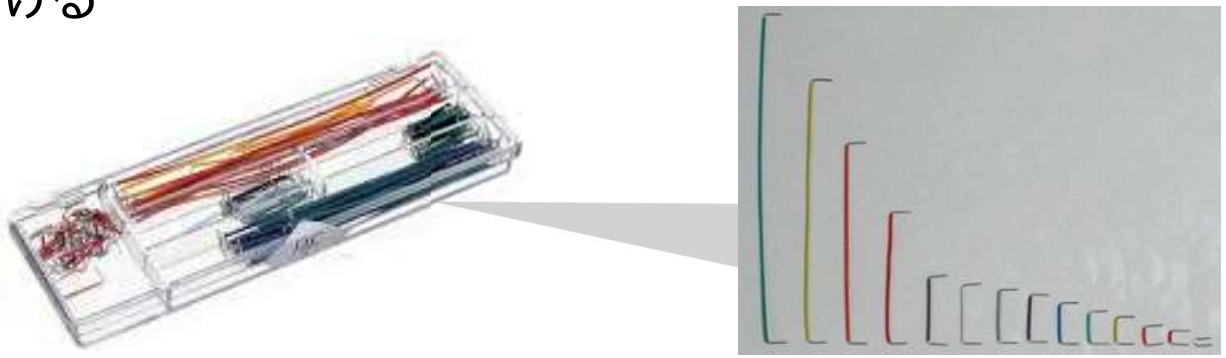
赤青4本の線は電源として使用
・赤:**プラス側**（3,5V・・・）
・青:**マイナス側**（0V,GND）

※本演習では、Pico/Pico2ボード
側から電源を取っていきます



ブレッドボードの説明（3）

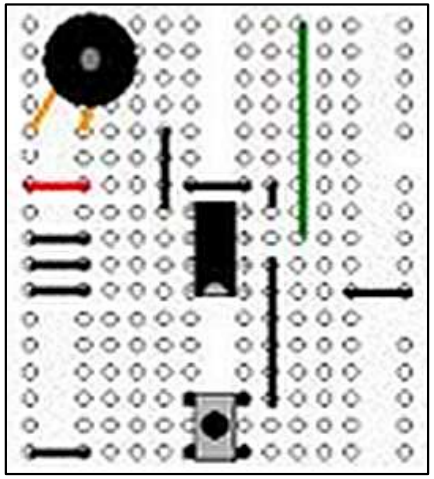
- ジャンパ線は長さごとに色が違うものが用意されているため、接続距離によって使い分ける



ジャンパ線

- 穴同士をジャンパ線や電子部品で繋ぐことで電子回路を実現する
- 電子部品によっては、方向性があるものもあるので注意

ex) LED：リード足の
長い方がアノード(＋)、短い方がカソード(－)

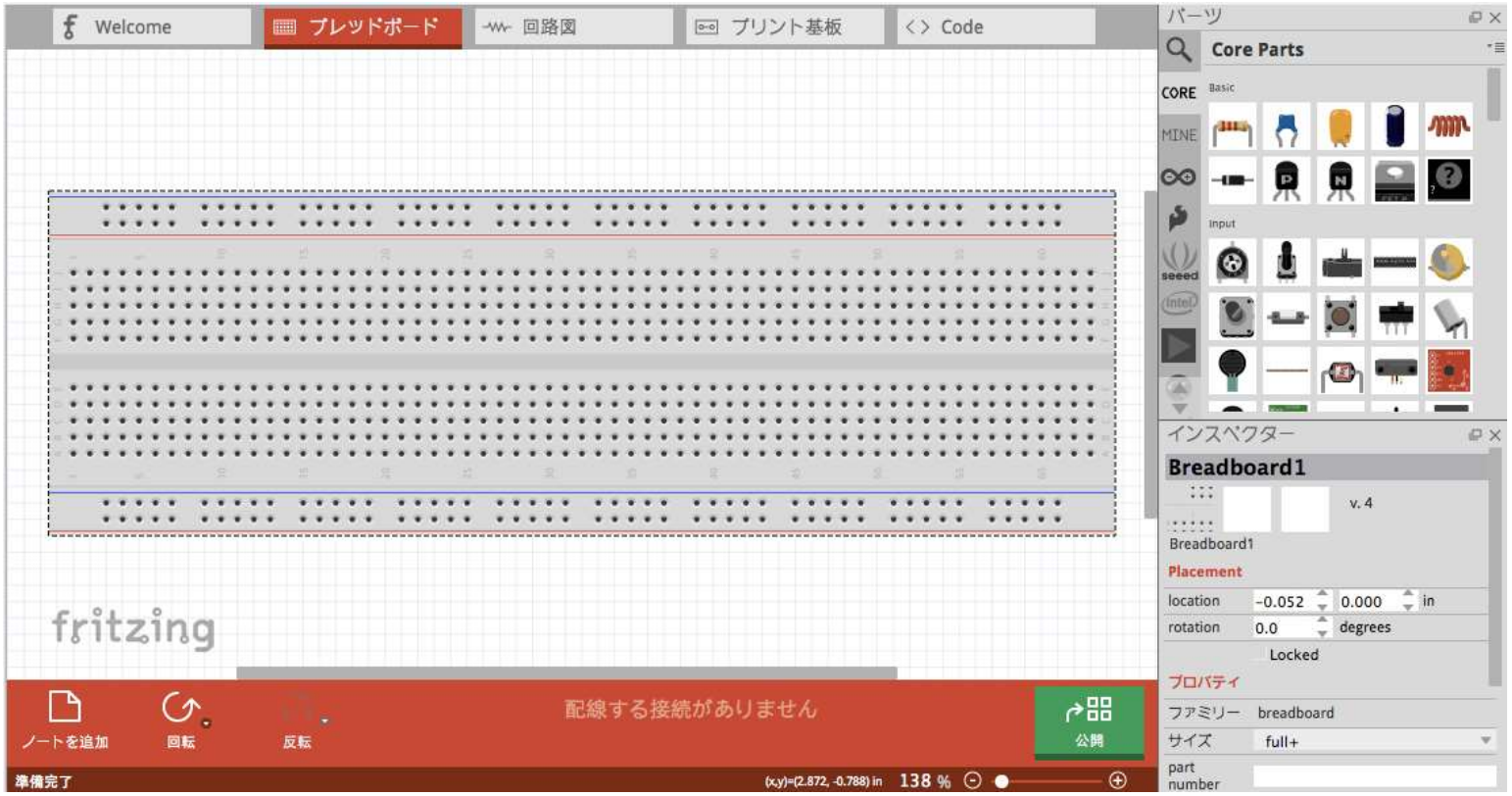


実装例

TOPPERS

ブレッドボードの説明（4）

- ブレッドボードの回路作図ツール：**fritzing**
  http://www.friting.org/

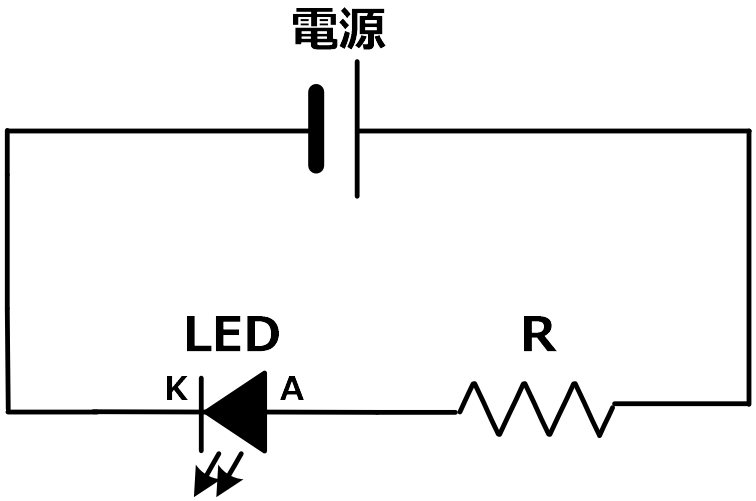


・素子やジャンパ線を選択して、自由にブレッドボード上に配置できます
・ドラッグ＆ドロップで直感的に操作可能です

TOPPERS

ブレッドボードの説明（５）：例１

- **例１）**電源ONにより、LEDが点灯する電子回路

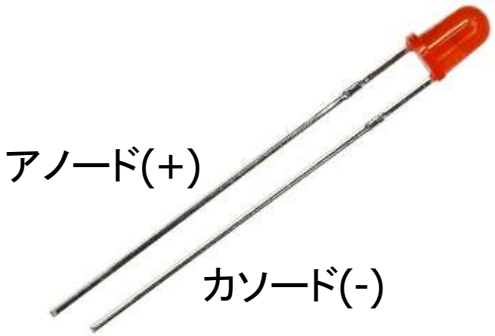


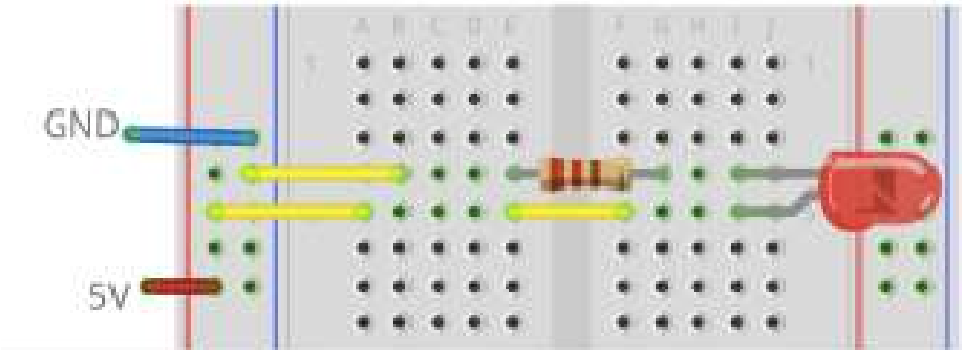
オームの法則
V=R･I

V	R	I
5.0V	200Ω	0.025A

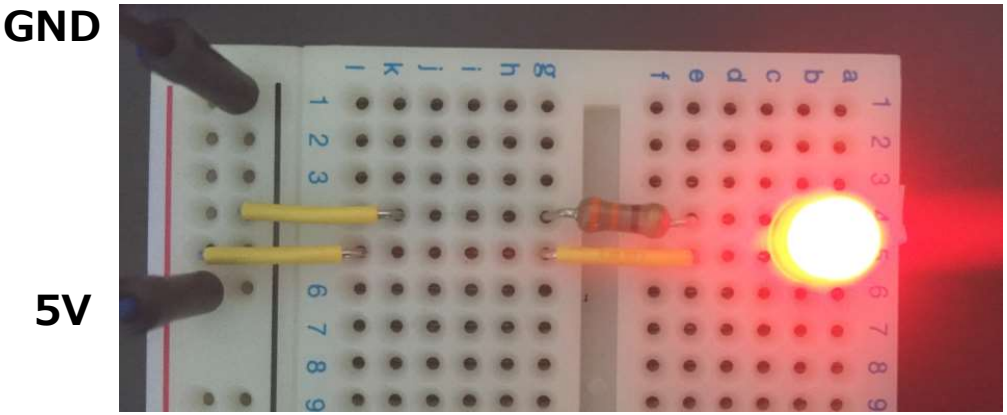
5VでLEDの標準電流を25mAとすると抵抗は200Ω

ブレッドボードの説明（６）：例１　実装例





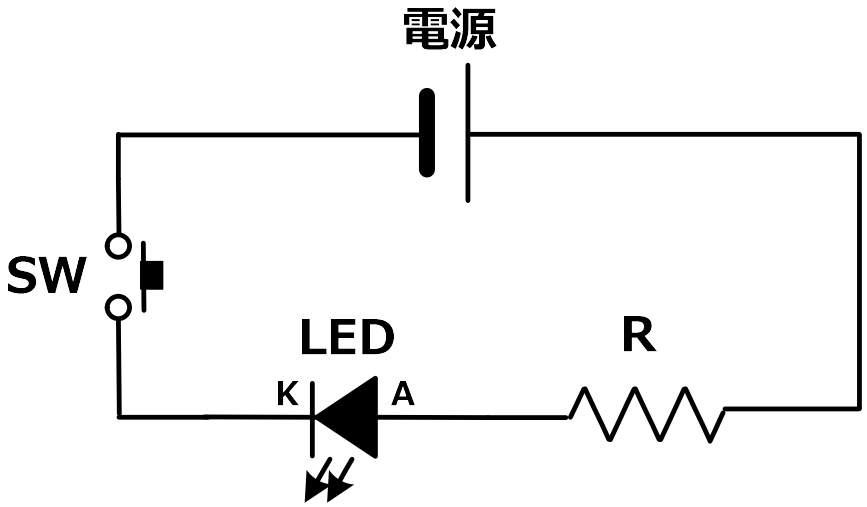
電源：ON



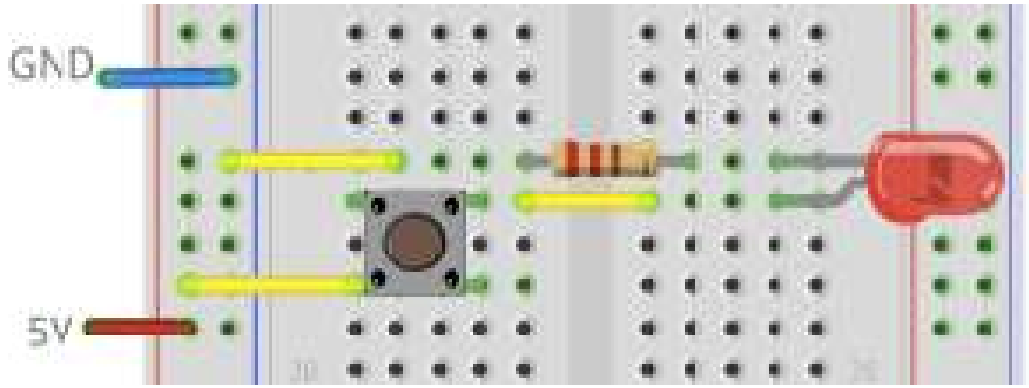
※ブレッドボードの縦4,5をLEDで接続

ブレッドボードの説明（7）：例2

- **例２）**電源ON後、キー(SW)押下によりLEDが点灯する電子回路



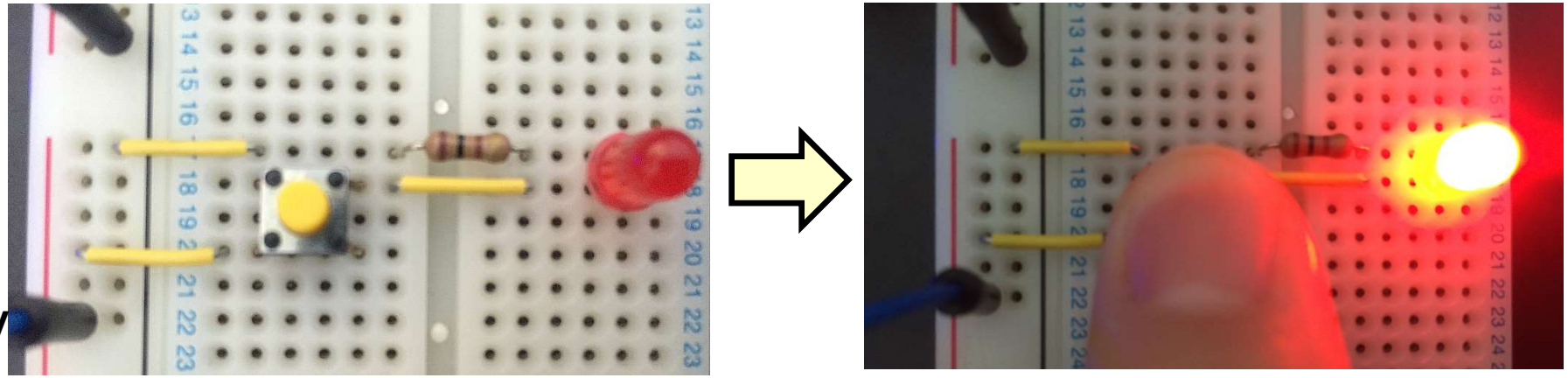
ブレッドボードの説明（8）：例2　実装例



GND

電源：ON, SW：OFF

電源：ON, SW：ON



3.3V

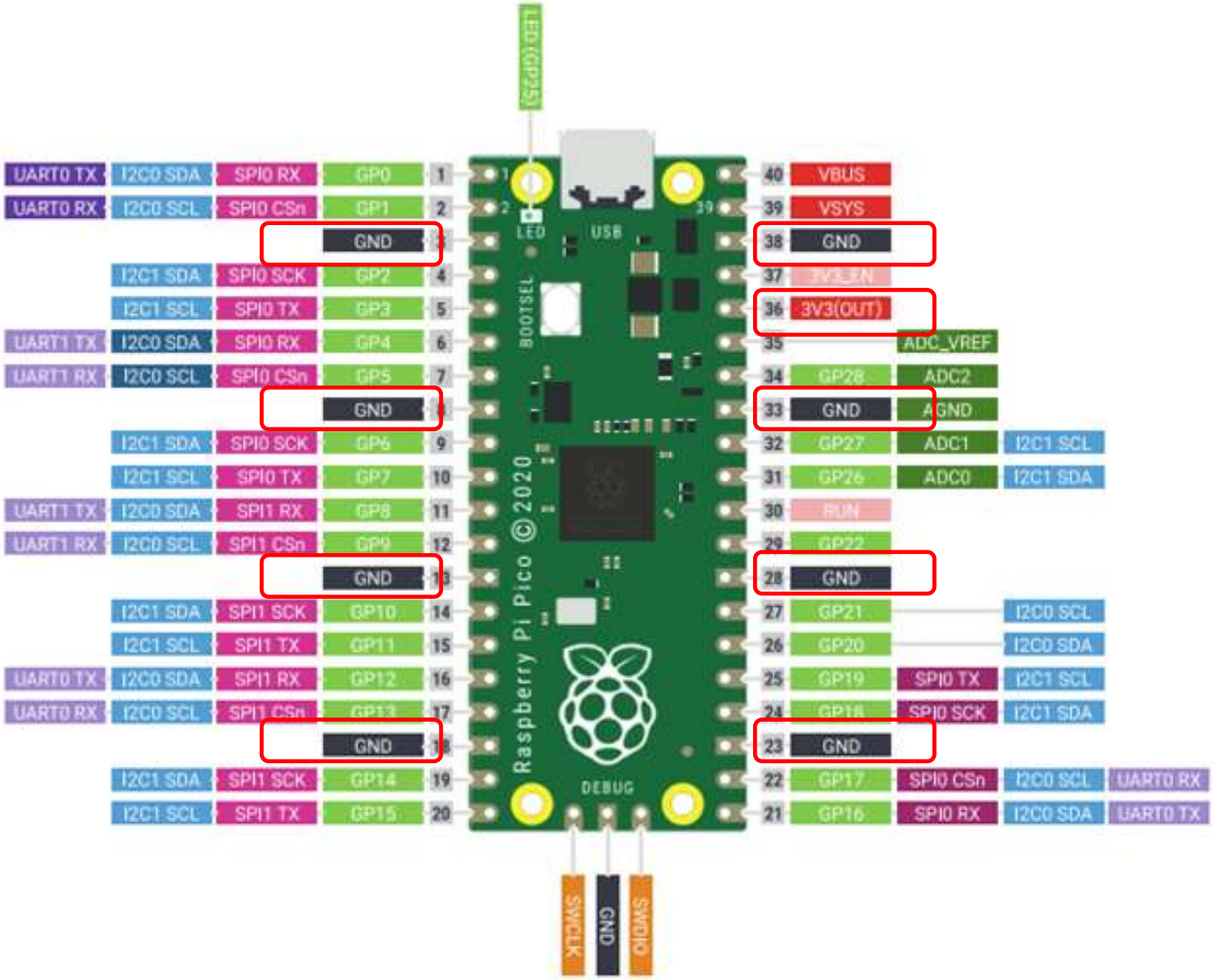
TOPPERS

デバッガとLEDの点灯確認

1. 開発環境のビルドと実行
2. LED点滅プログラムの確認
3. ブレッドボードの実装方法
4. LEDを配置する

Picoボードの電源/GNDの供給について

- 信号電圧は3.3Vなので、+には3.3Vを供給する



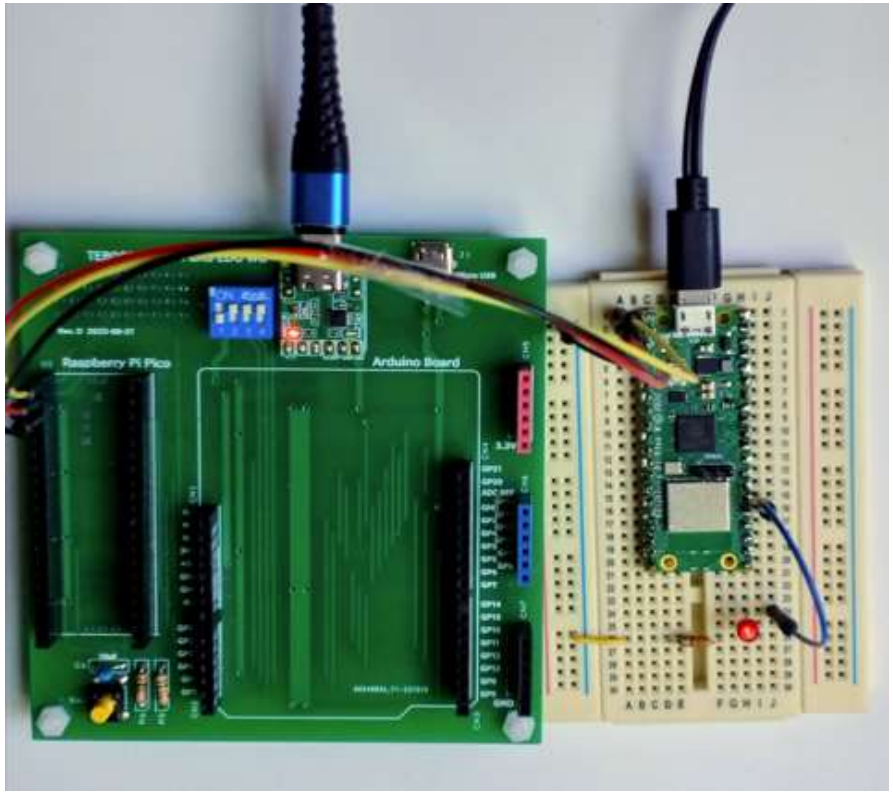
TOPPERS

ブレッドボードに実装

- ブレッドボードにLED、抵抗（100Ω）を実装して、ジャンパーケーブルでつなぐ

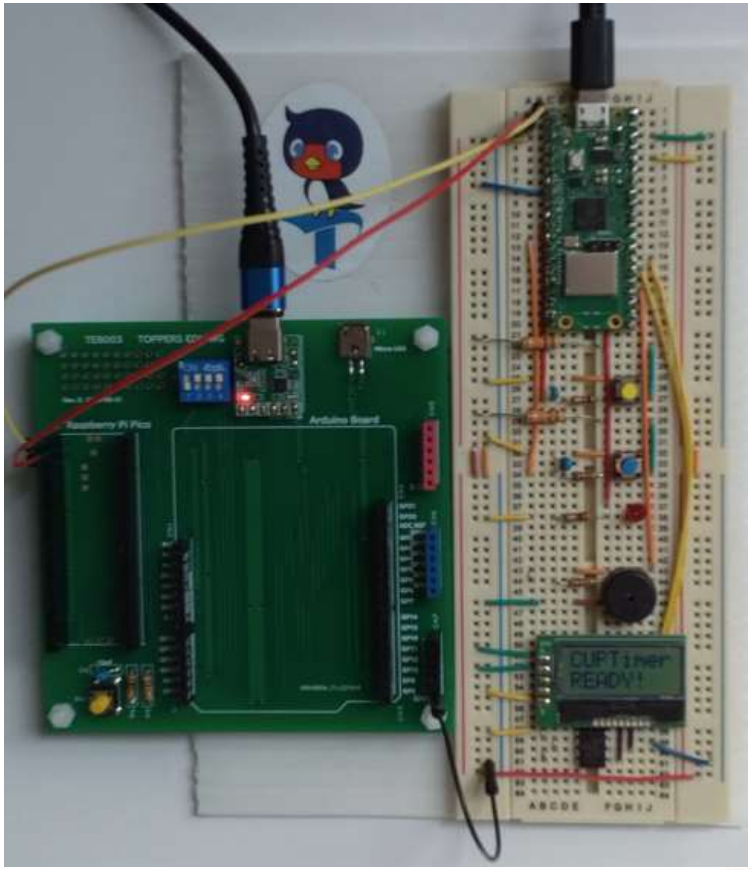
GND(GND) – 抵抗(100Ω) – LED(カソード)

GP19 ――――――――――― LED(アノード)



TOPPERS

ブザーとキーを動かそう



TOPPERS

ブザーとキーを動かそう

1. ブレッドボードの実装方法
2. 回路図（ブザー/キー）の確認
3. GPIOの確認
4. switch_pushプログラムの確認

ブザー/キープログラム：switch_push概要

- キーを押すと同時に、ブザーを鳴らしてLEDを点灯させる

  ① キー(SW1)を押下（ブザーを鳴らしてLED1を点灯）
  ② キー(SW2)を押下（ブザーを鳴らしてLED2を点灯）

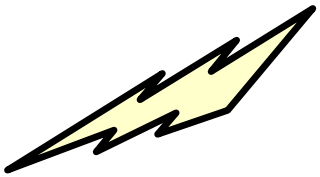
- 学習内容
  - ブレッドボードの実装方法
  - 回路図（ブザー/キーデバイス）の確認
  - GPIOの確認
  - switch_pushプログラムの確認

ブザーとキーを動かそう

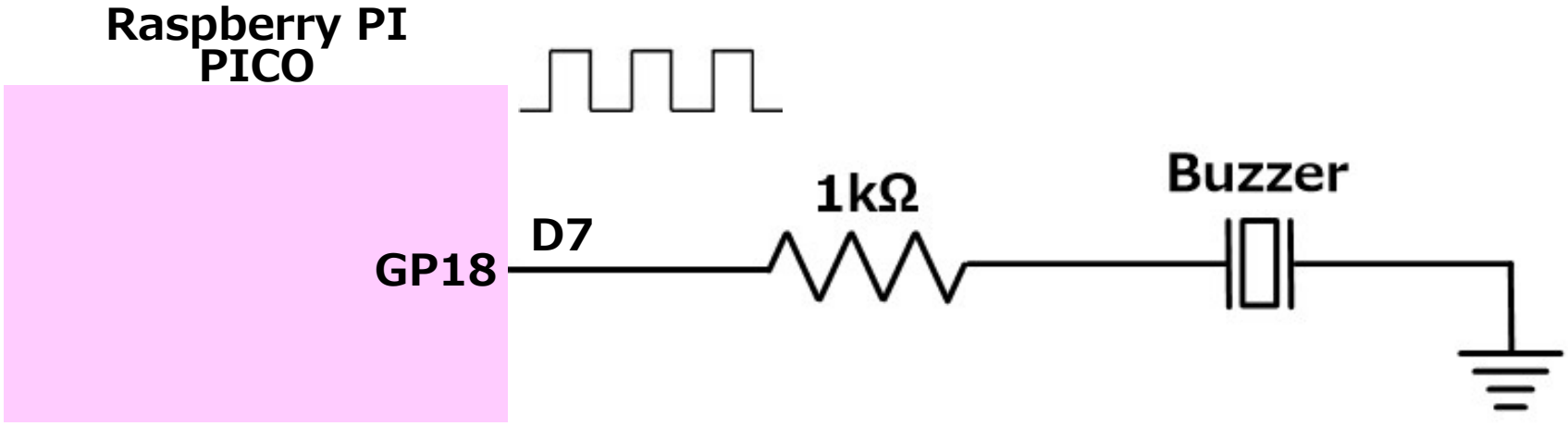
1. ブレッドボードの実装方法
2. 回路図（ブザー/キー）の確認
3. GPIOの確認
4. switch_pushプログラムの確認

回路図（ブザー）の説明（1）

- 発信器が内蔵された電子部品でパルス（PWMなど）を入れることで音を出す
- パルスの波形を変更することで、音調を変更することが可能



- ブザーの回路図



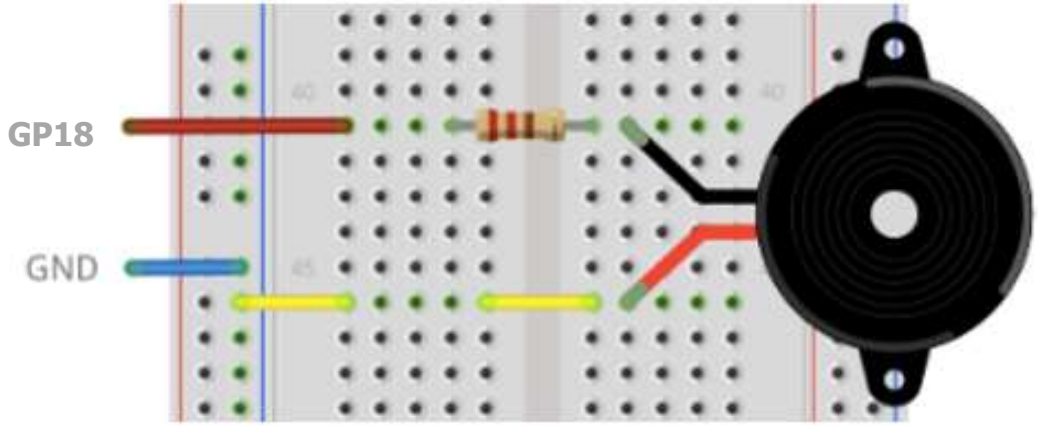
TOPPERS

回路図（ブザー）の説明（2）

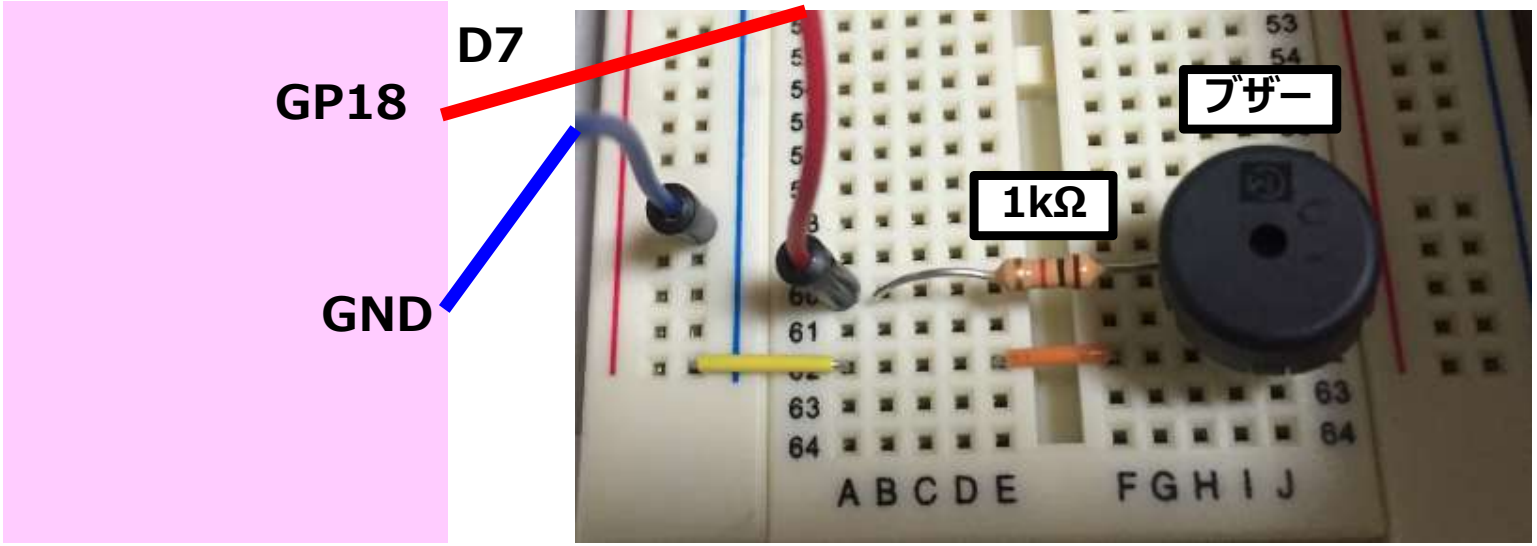
回路図を元にブレッドボード上へ実装してみましょう

・まずは”fritzing”でパーツ配置
・次に実際にブレッドボード上に実装してみましょう

回路図（ブザー）の説明（3）：実装例



Pico/Pico2ボード

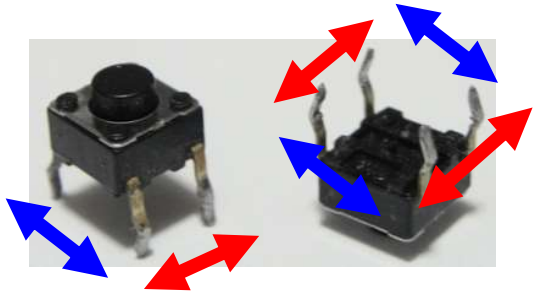


TOPPERS

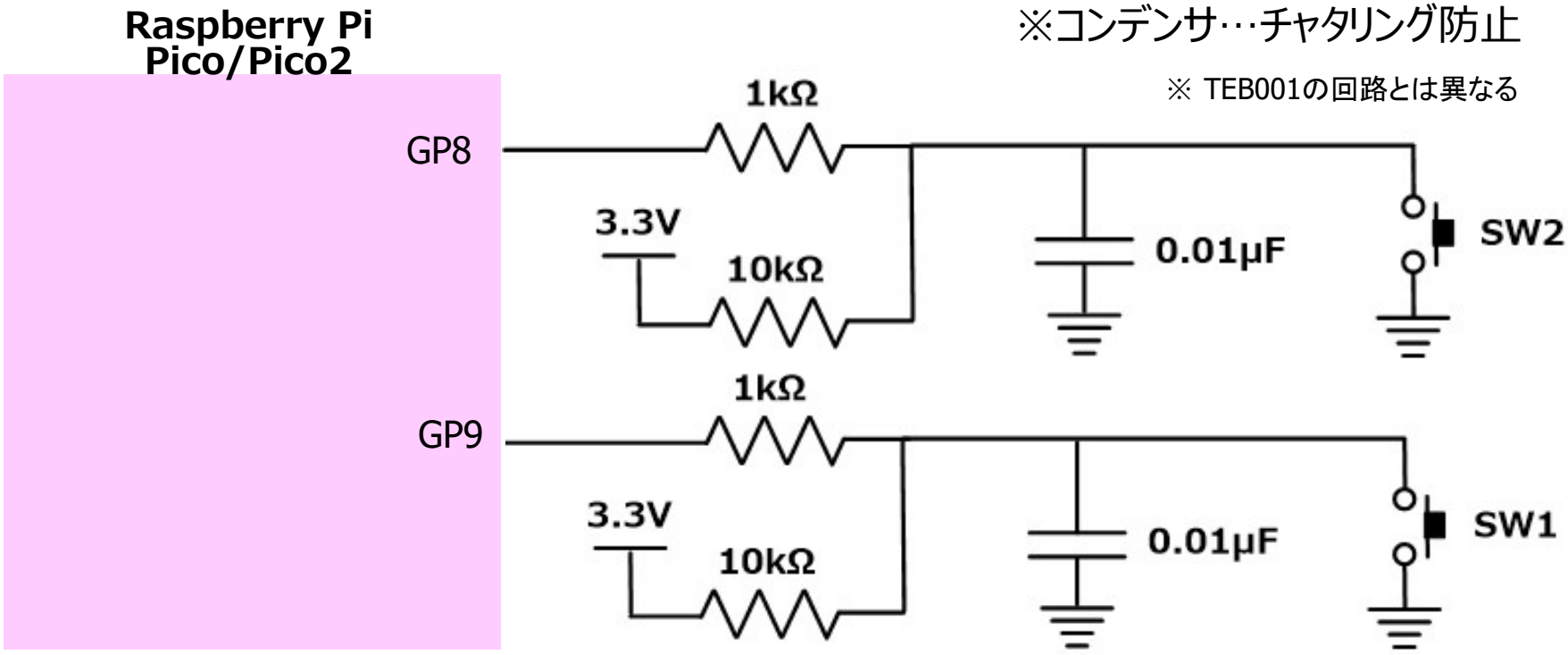
回路図（キー）の説明（1）

- キーを押すことでON/OFFが切り替わる

※**初期状態**では**青矢印の方向が導通**している
※**スイッチを押す**ことで、**赤矢印の方向に導通**する



- キーの回路図



※コンデンサ…チャタリング防止

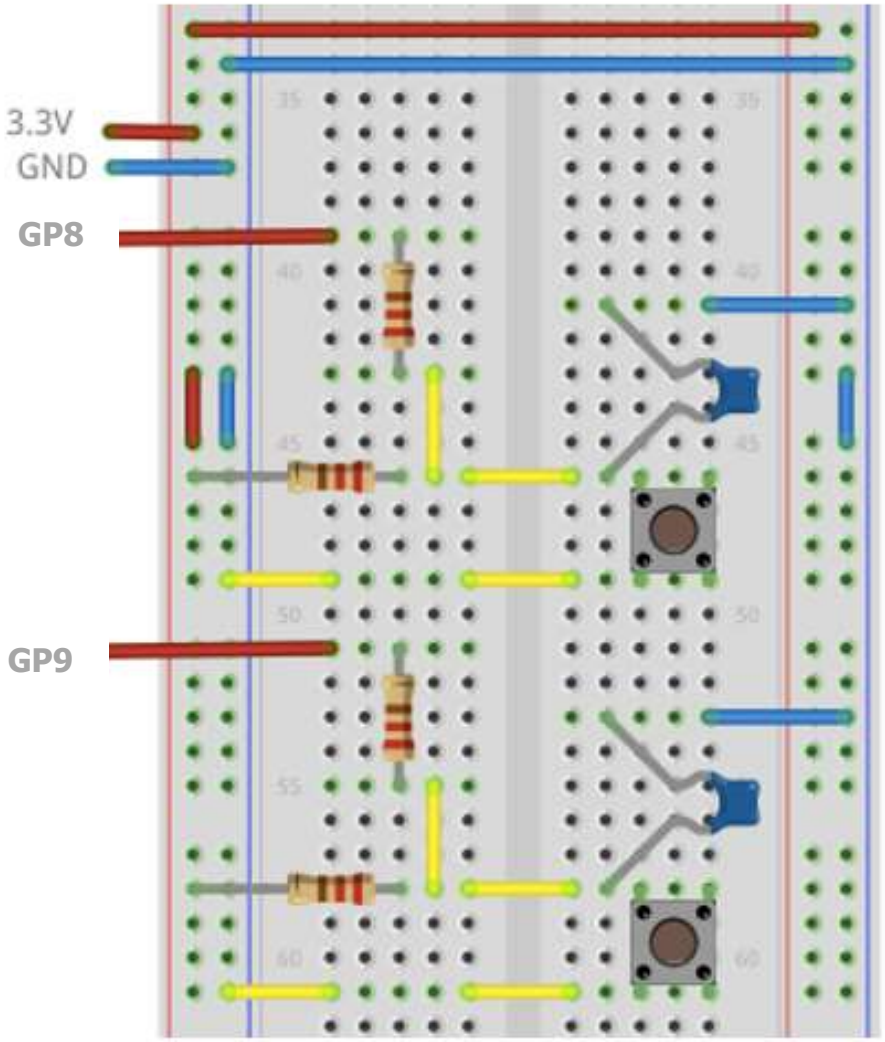
※ TEB001の回路とは異なる

回路図（キー）の説明（2）

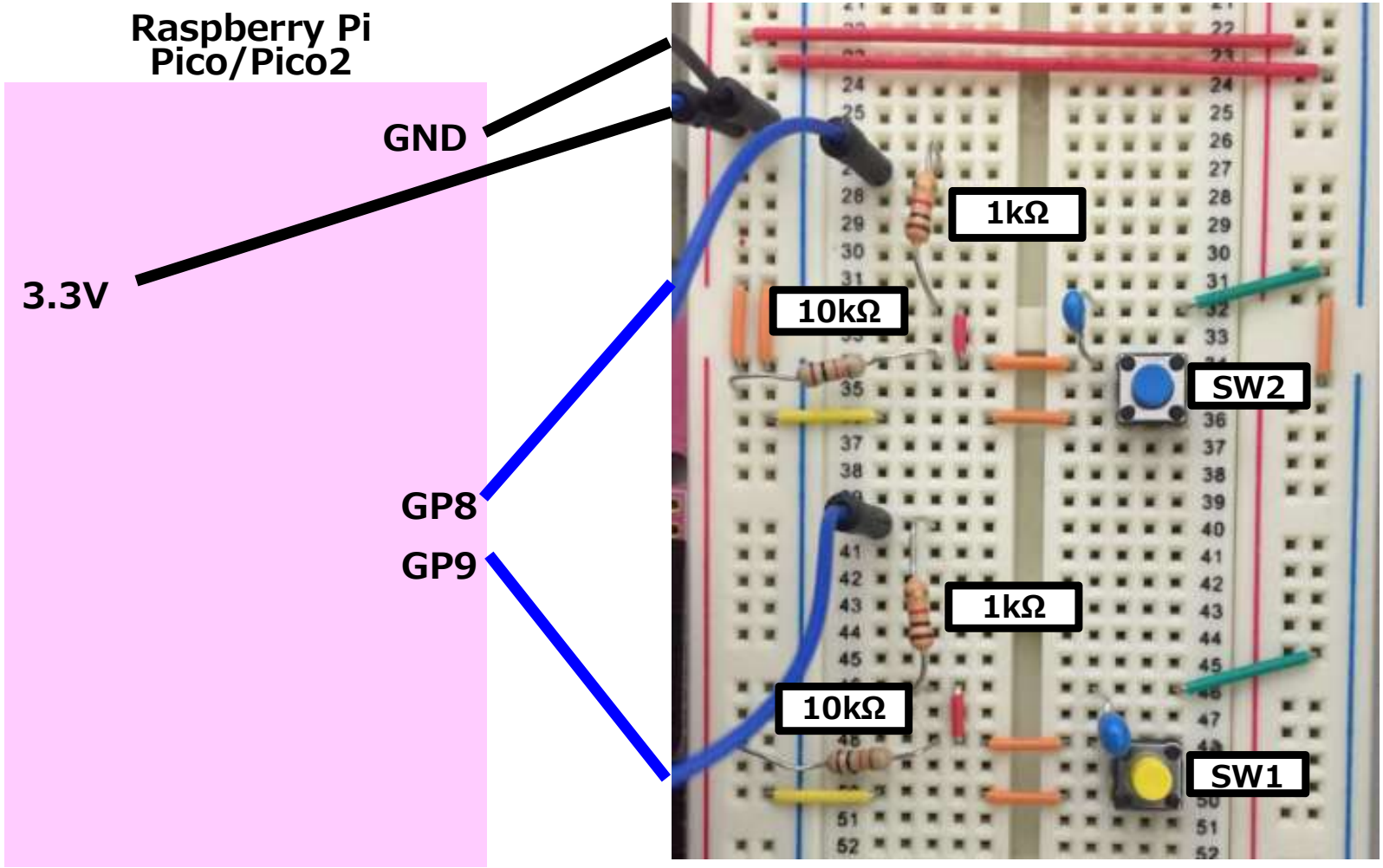
回路図を元にブレッドボード上へ実装してみましょう

・まずは”fritzing”でパーツ配置
・次に実際にブレッドボード上に実装してみましょう

回路図（キー）の説明（3）：実装例



回路図（キー）の説明（4）：実装例

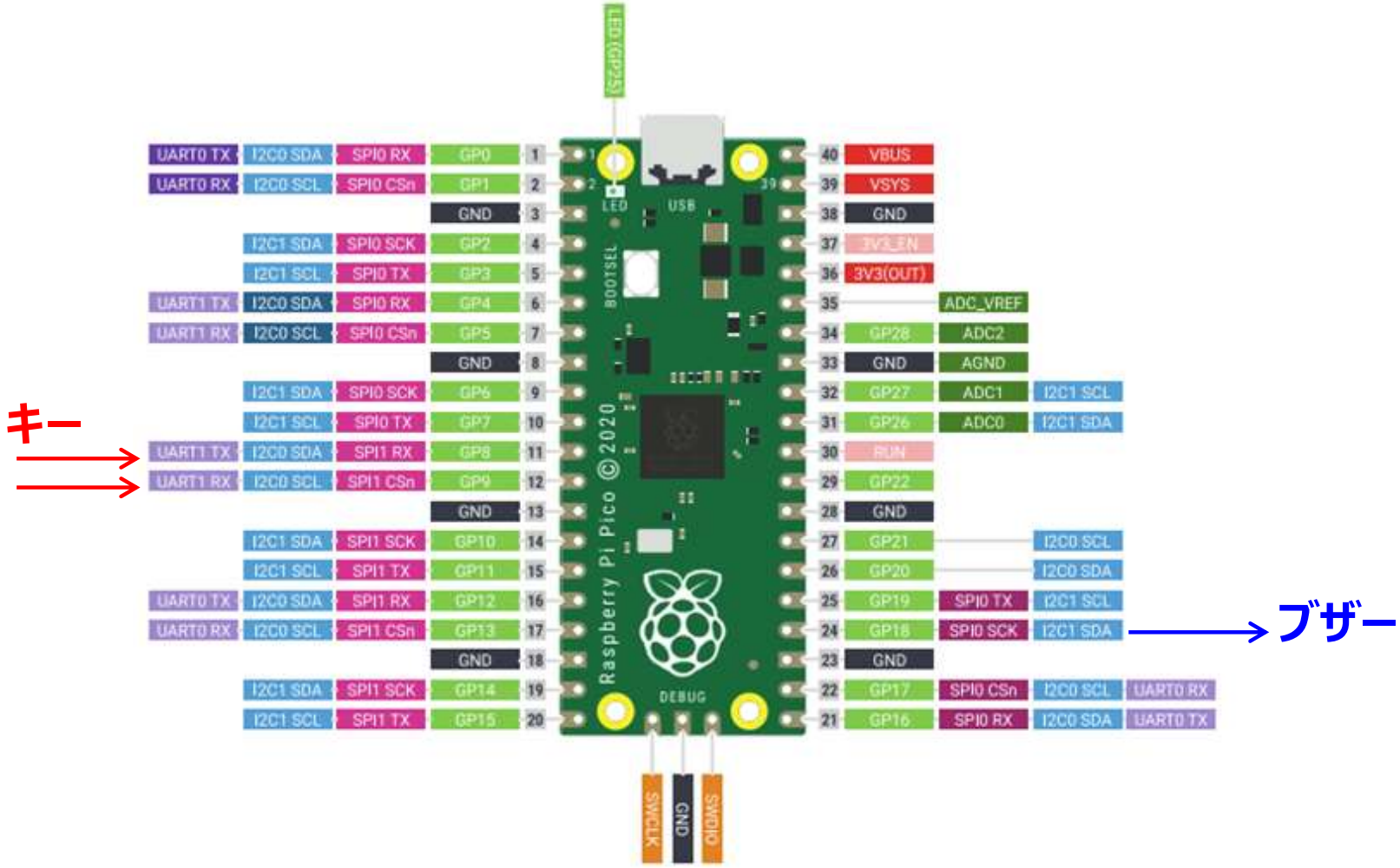


ブザーとキーを動かそう

1. ブレッドボードの実装方法
2. 回路図（ブザー/キー）の確認
3. GPIOの確認
4. switch_pushプログラムの確認

ブザー/キー：GPIOポートの割当て

- ブザー（24ピン）　　⇔ **GP18**：SIOポートのビット18
- キー　（11,12ピン）　⇔ **GP8,GP9**　：SIOポートのビット8,9



TOPPERS

ブザー：GPIO/PWMのプログラム設定

・各デバイスのGPIO設定とPWM設定を実施する　**ブザーのGPIO設定**

device.h

```
#define BUZZER_PIN      PINPOSITION18
```

switch_push.c

```
/*
 *  BUZZER初期化
 */
void buzzer_init(void){
    pwm_setup(BUZZER_PIN, 8);
}
```

switch_push.c

```
/*
 *  BUZZER-DUTY設定関数
 */
void
buzzer_duty(uint16_t duty){
    uint32_t ct = get_pwm_count(BUZZER_PIN);
    uint32_t base = TADR_PWM_BASE + (get_pwm_channel(BUZZER_PIN) * PWM_CHANNEL_SIZE);
    uint32_t count;

    duty = (duty * MAX_PWM_COUNT) / 100;
    count = calc_count(ct, sil_rew_mem((uint32_t *)(base+TOFF_PWM_CH_CC)), duty);
    sil_wrw_mem((uint32_t *)(base+TOFF_PWM_CH_CC), count);
}
```


TOPPERS

キー：GPIOのプログラム設定

・各デバイスのGPIO設定を実施する　　**キーのGPIO設定**

device.h

```
#define PSW1_PINNO      PINPOSITION8
#define PSW2_PINNO      PINPOSITION9
#define PSW1            (1<<PSW1_PINNO)
#define PSW2            (1<<PSW2_PINNO)
#define PSW_MASK        (PSW1 | PSW2)
```

switch_push.c

```
/*
 * PUSHスイッチ接続ポート初期化
 */
void
push_switch_init(void){
    uint32_t pinpos, mask;

    for(pinpos = PSW1_PINNO ; pinpos <= PSW2_PINNO ; pinpos++){
        uint32_t offset = pinpos * 4;
        pinmux_setup(pinpos, IO_BANK0_GPIO_CTRL_FUNCSEL_VALUE_SIO);

        sil_wrw_mem((uint32_t *)(TADR_SIO_BASE+TOFF_SIO_GPIO_OE_CLR), (1<<pinpos));
        mask = (sil_rew_mem((uint32_t *)(TADR_PADS_BANK0_BASE+TOFF_PADS_BANK0_GPIO+offset))
              & PADS_BANK0_GPIO_PMASK);
        sil_wrw_mem((uint32_t *)(TADR_PADS_BANK0_BASE+TOFF_PADS_BANK0_GPIO+offset
              +REG_ALIAS_XOR), mask);
        sil_wrw_mem((uint32_t *)(TADR_SIO_BASE+TOFF_SIO_GPIO_OUT_CLR), (1 <<pinpos));
    }
}
```


TOPPERS

ブザーとキーを動かそう

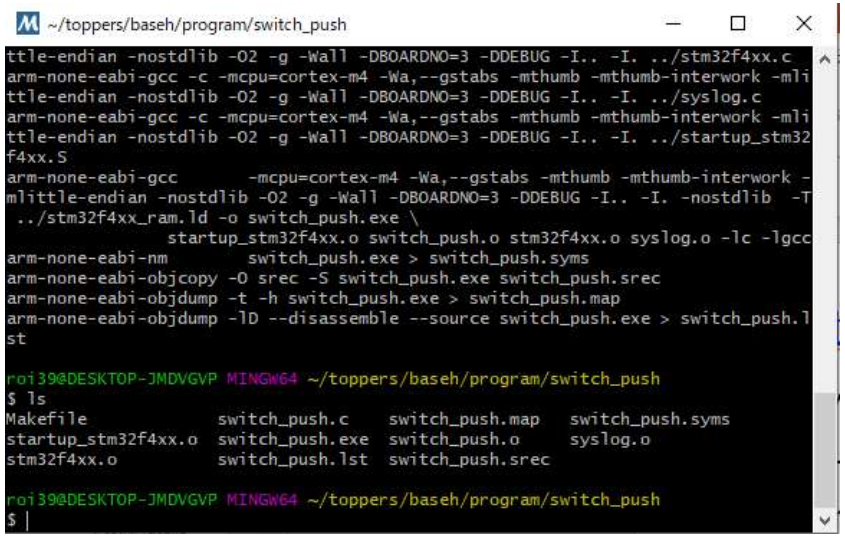
1. ブレッドボードの実装方法
2. 回路図（ブザー/キー）の確認
3. GPIOの確認
4. switch_pushプログラムの確認

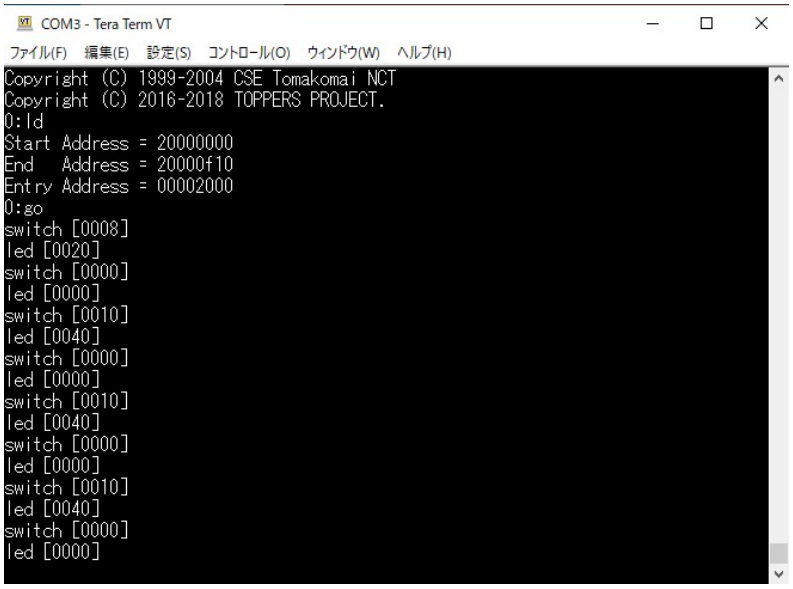
switch_pushプログラム：概要

- 構成ファイル
  - Makefile　　メイクファイル
  - switch_push.c　　メイン関数
- 下のディレクトリのファイルはled_blinkと同様

switch_pushプログラム:ビルド

- MSYS2にてcdコマンドでprogram/switch_pushに移動
  - cd ../switch_push(return)
- makeコマンドでビルド(Pico用)
  - make DEBUG=1(return)
- ボードリセット後、switch_push.srecをダウンロードして実行

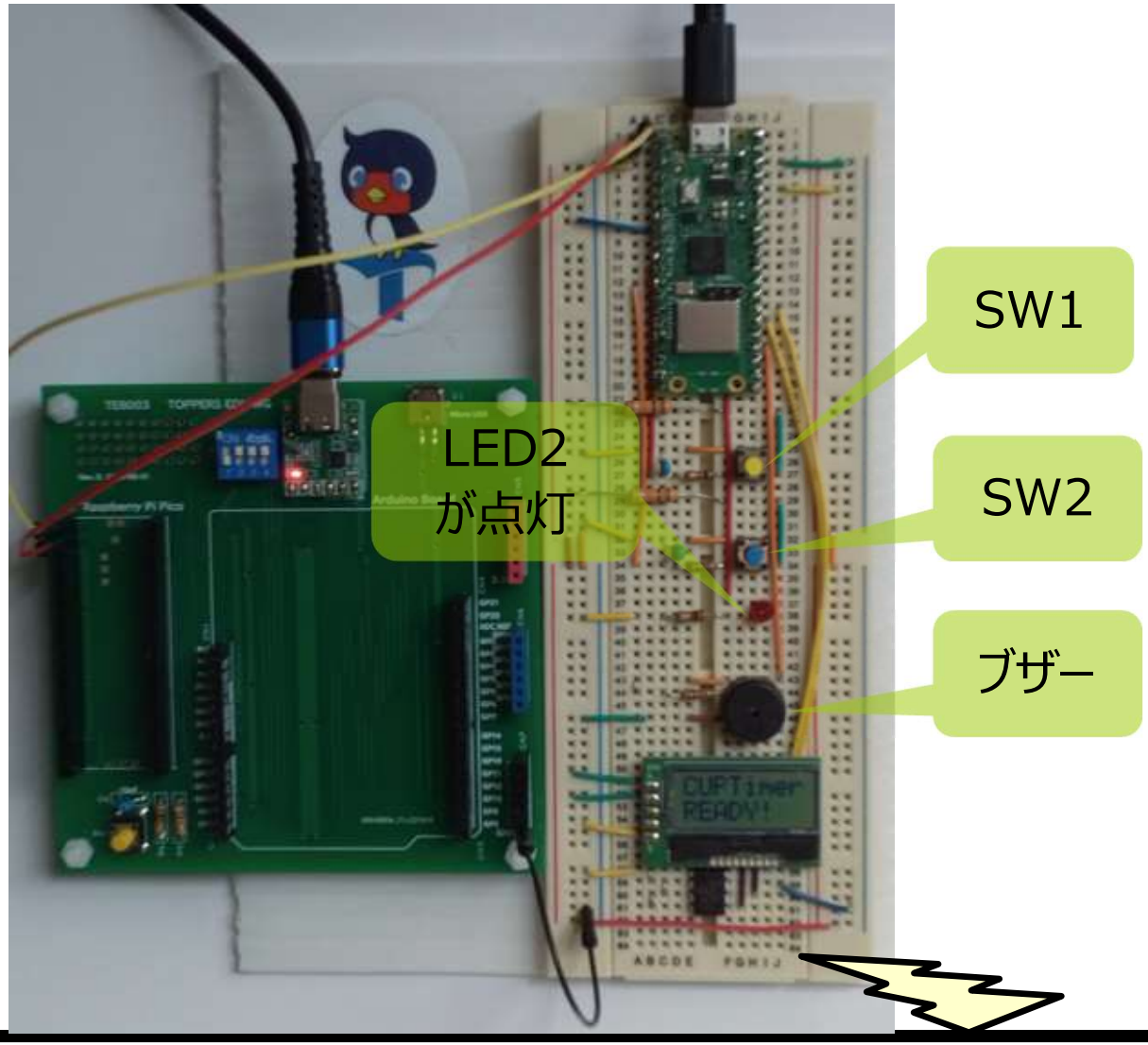




TOPPERS

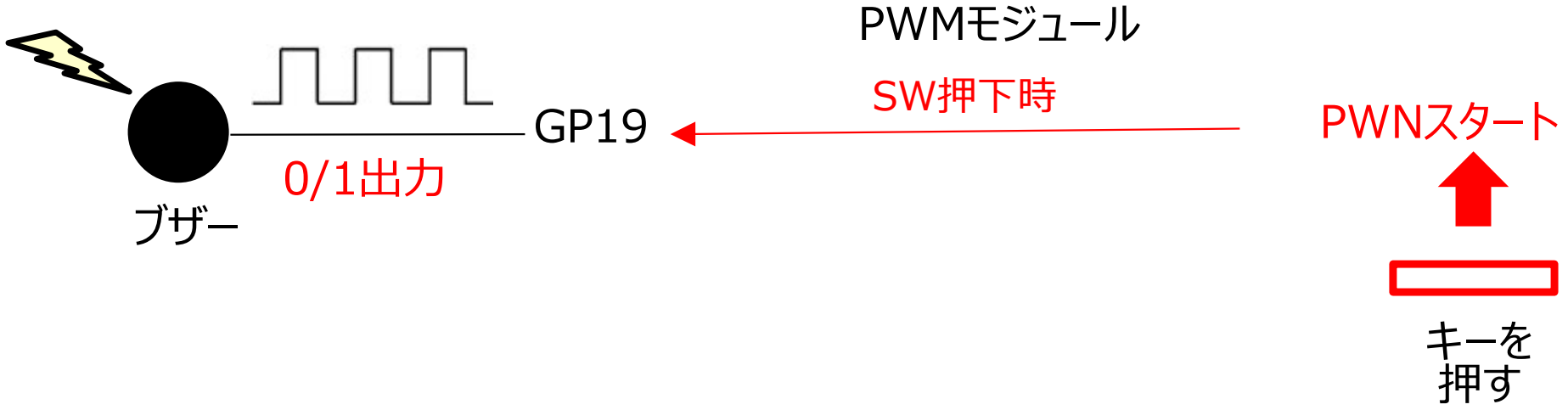
switch_pushプログラム：実行

- キー(SW1/SW2)を押下するごとにブザーが鳴り、LEDが点灯



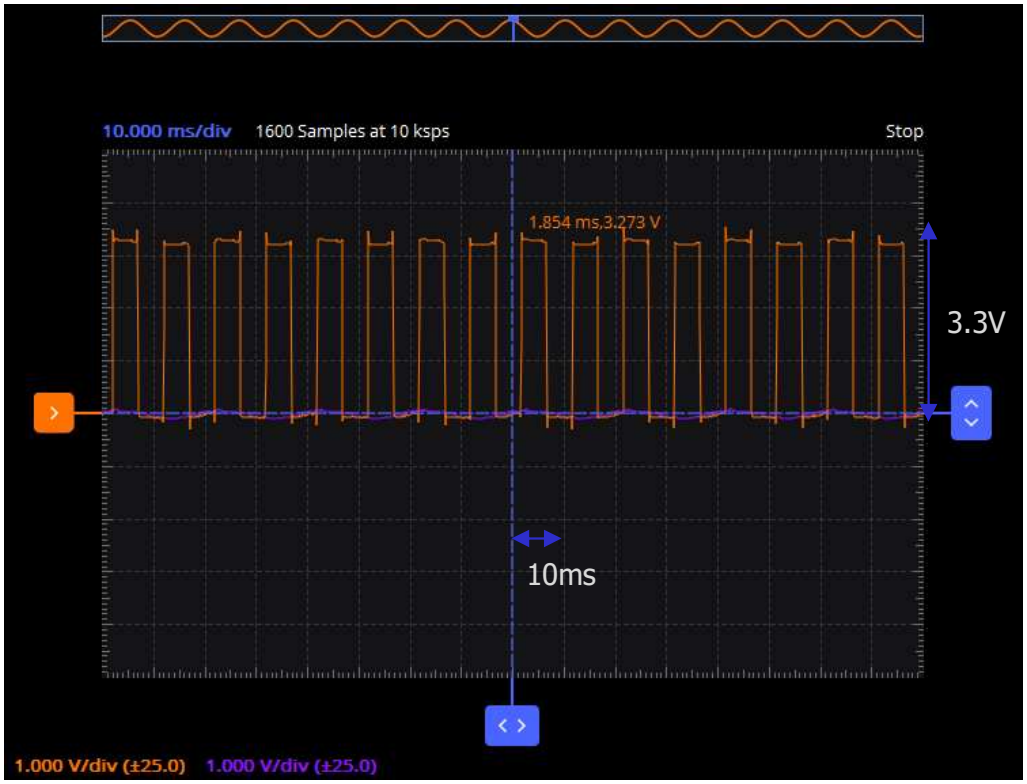
switch_pushプログラム：ブザー制御の実現

- 今回のブザー制御には、PWM制御を用いる
- Pico/Pico2はGPIOに対応したPWMモジュールを持ち、ブザーのGP18に対応させる
- 出力（0/1）を切替えながら、パルスを作り出す
  →10msごとに50%で、チャネル信号をLOW/HIGHさせる



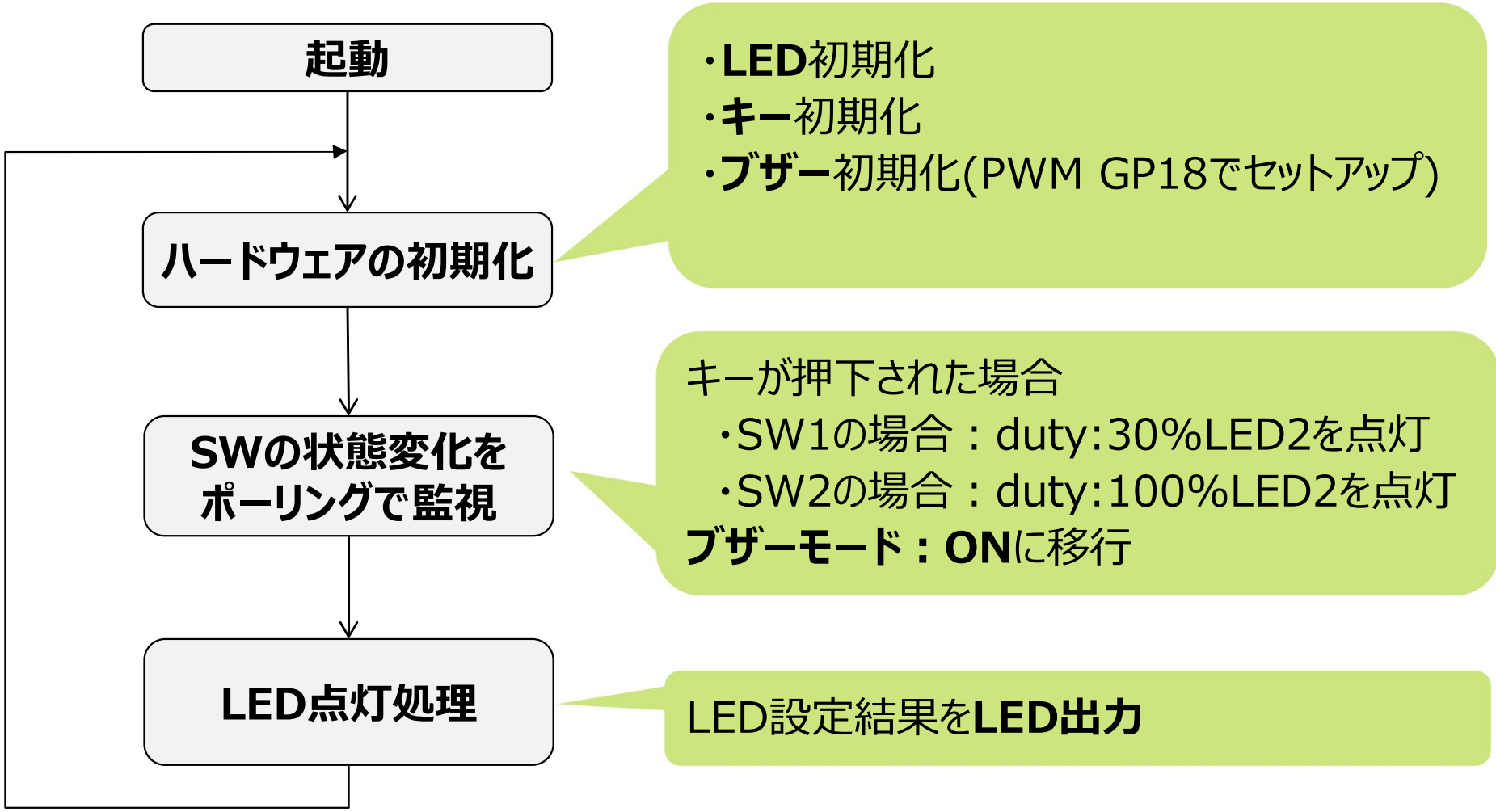
ブザー信号を測定

- ブザー信号をオシロスコープで測定
  - ベースクロック：10000Hz
  - PERIODを100として、DUTYを50%
  - 10msの半分をHight、半分をLow

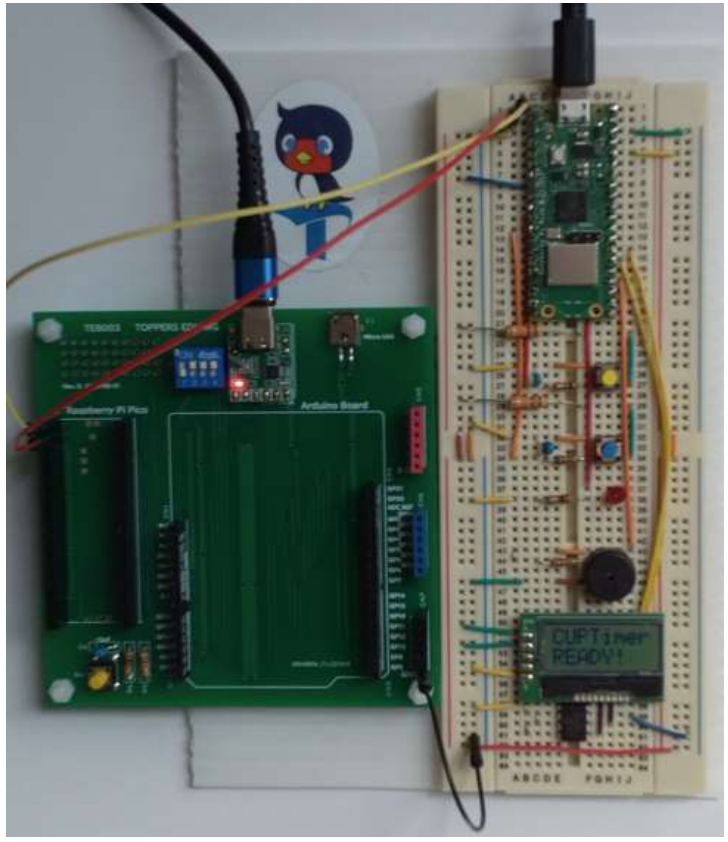


switch_pushプログラムの説明(1):メイン関数

switch_push/main.c



LCDを動かそう



TOPPERS

LCDプログラム：lcd_test概要

- プログラム実行後、LCD上に”00000000”が表示される。

①SW1を押下すると、”1”カウントダウンする。LED1が点灯する。
②SW2を押下すると、”1”カウントアップする。LED2が点灯する。

- 学習内容
  - 回路図（LCD）の確認
  - I2C通信について
  - lcd_testプログラムの確認

LCD

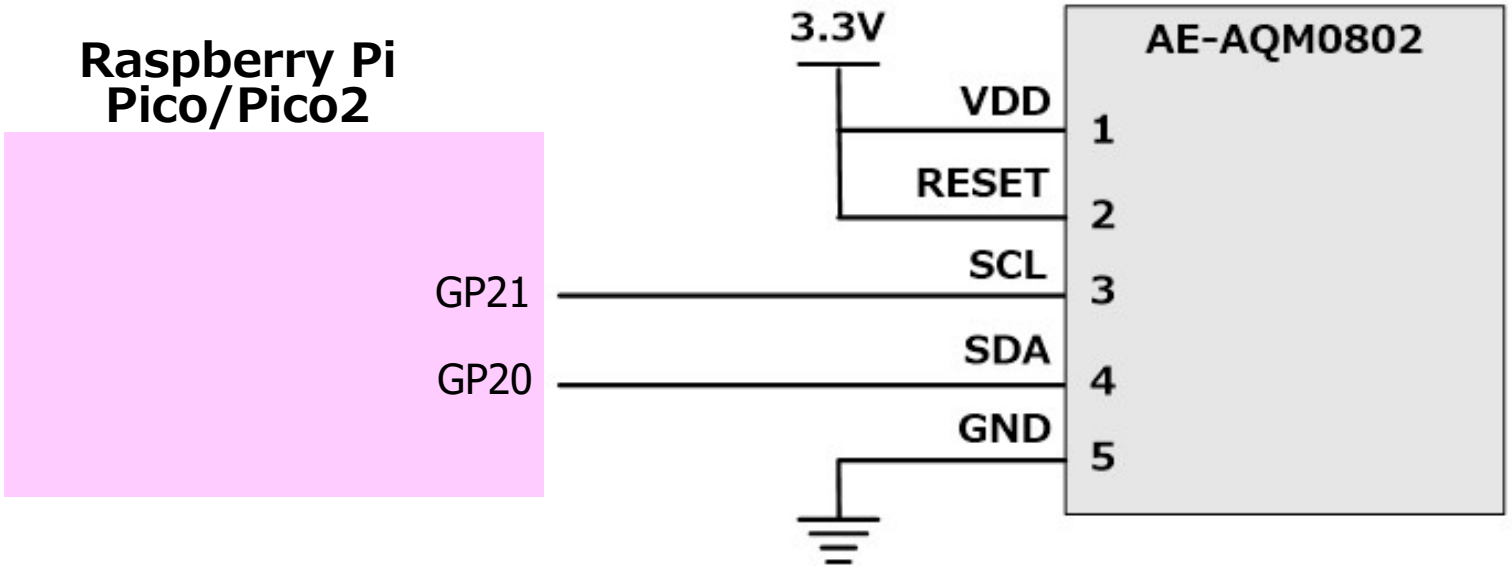
LCDを動かそう

1. 回路図（LCD）の確認
2. I2C通信について
3. lcd_testプログラムの確認

回路図（LCD）の説明（1）

- LCD（Liquid crystal display）とは、画面表示装置である。
- 種類はキャラクタ/グラフィック/タッチパネル式など様々である。

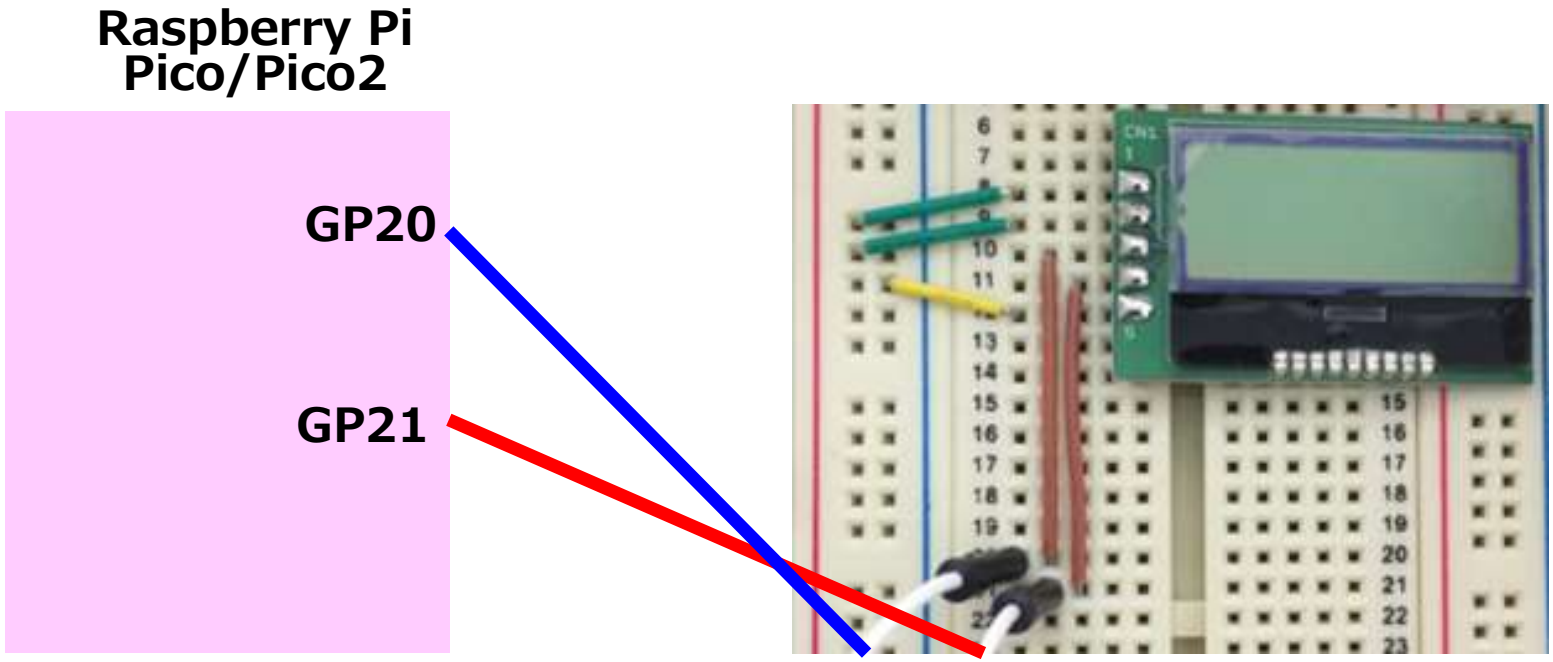
- LCDの回路図



回路図（LCD）の説明（２）

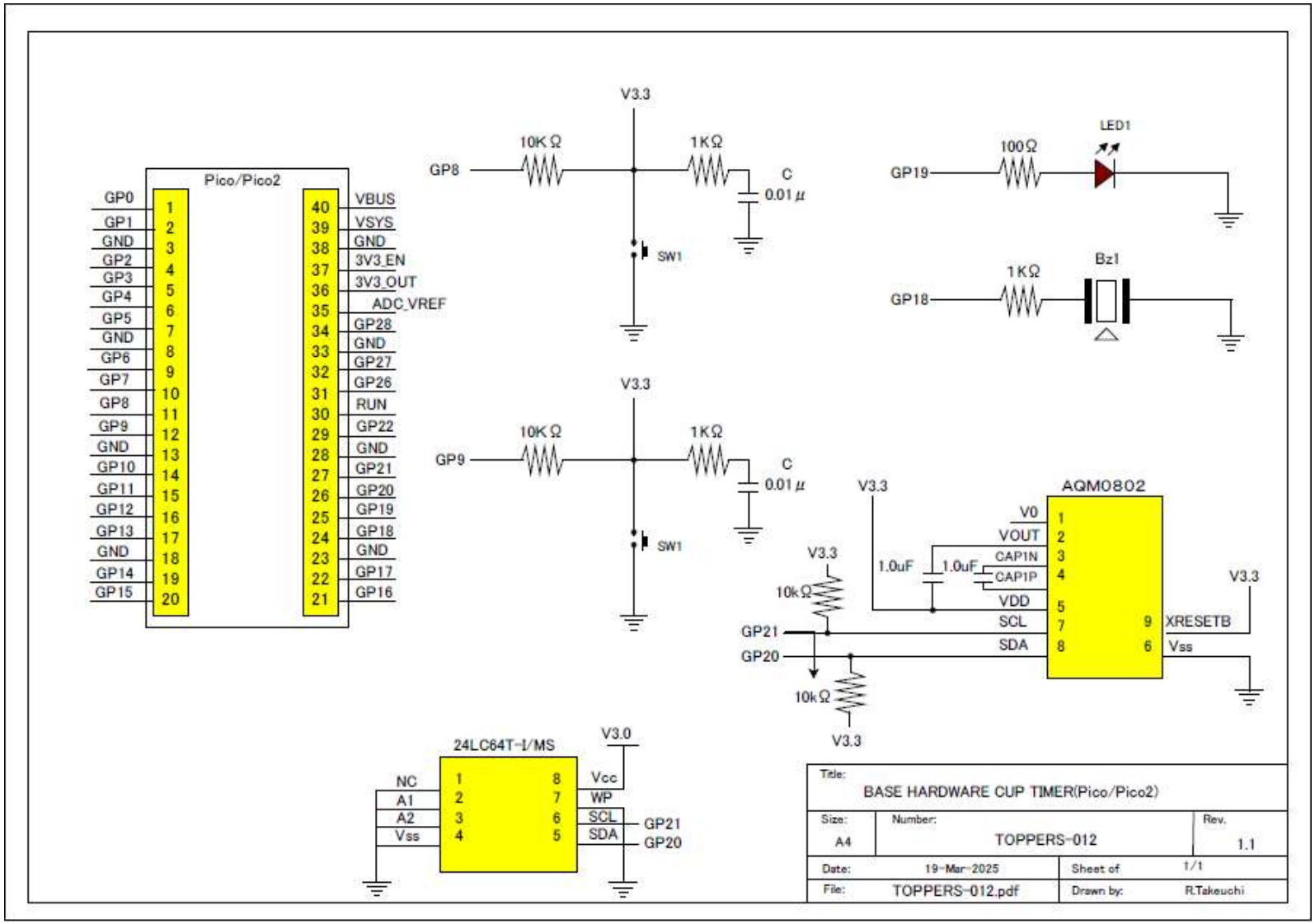
回路図を元にブレッドボード上へ実装してみましょう

回路図(LCD)の説明(3):実装例



カップラーメン回路図

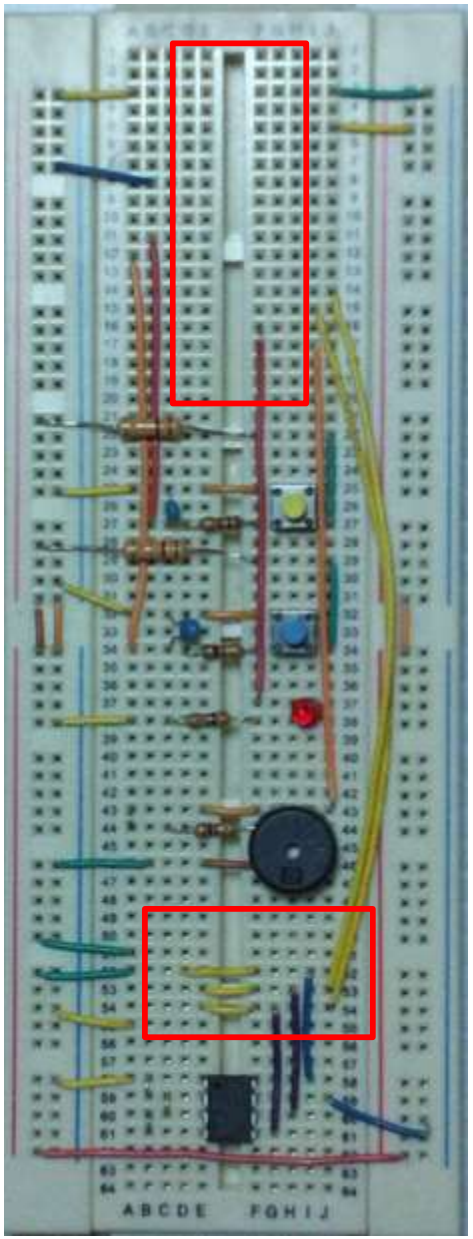
- LCDを含めすべてを実装した回路図を作成



TOPPERS

カップラーメンブレッド・ボード1

- 回路図を元に、ブレッド・ボード上に配線する

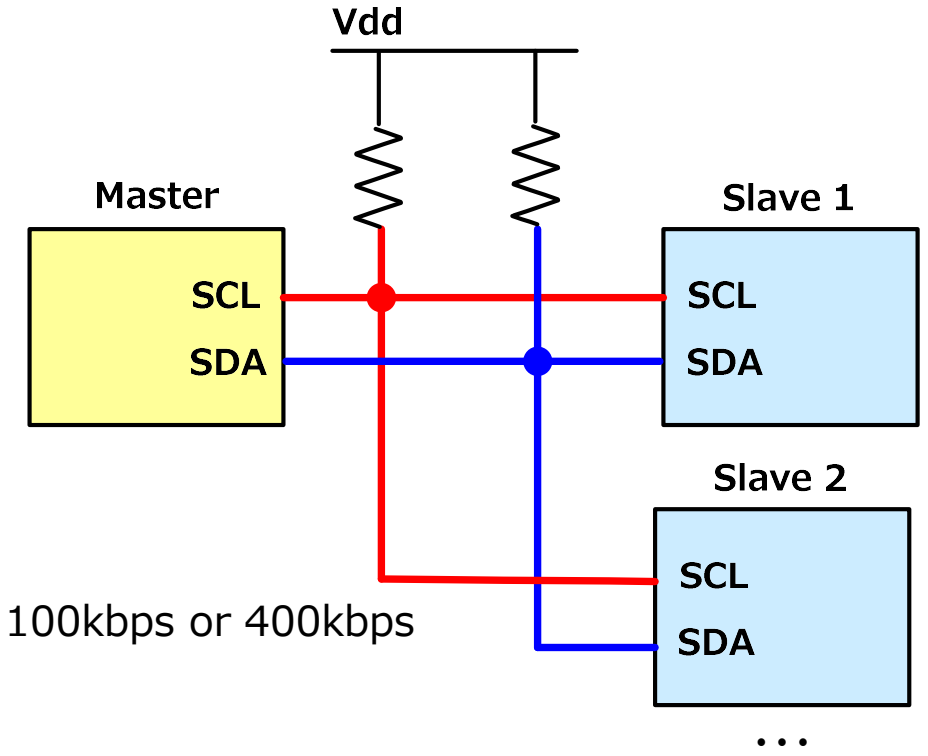


LCDを動かそう

1. 回路図（LCD）の確認
2. I2C通信について
3. lcd_testプログラムの確認

I2C通信(1):概要

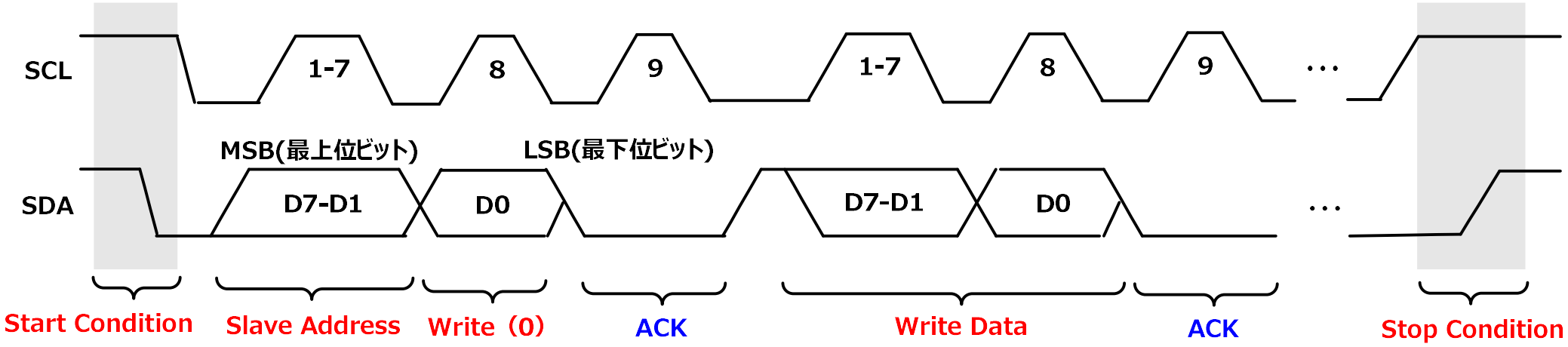
- I2C(Inter-Integrated Circuit)は周辺デバイスとのシリアル通信方式



- クロック信号(SCL)とデータ信号(SDA)の2線で通信を実施する
- 各スレーブがスレーブアドレスを持ち、データの中にアドレスを含む
- 通信を開始するのは全てマスタ側でSCLを基準にデータがSDA上で転送
- スレーブアドレスが一致したデバイスのみ、送受信を継続する仕組み

I2C通信（2）：マスタからスレーブへ送信の場合

スレーブアドレスが7bitの時のタイミングチャート

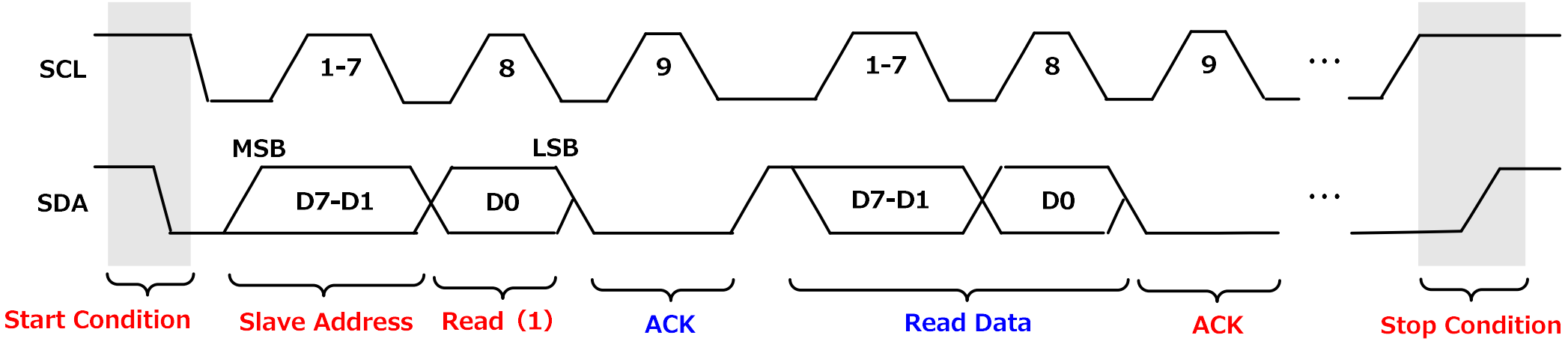


※赤：マスタ側、青：スレーブ側

- SCLがHighの間にマスタがSDAをLowとするとStart Conditionになる
- マスタはSCLがHighの間にSDAをHighにすることでStop Conditionとなる（それぞれ通常のデータ時はSCLがLowの時しか変化しない）
- スレーブ側はデータの取り出しが完了するまでBUSYとしてSCLを強制的にLowとすることで、見かけ上クロックがなくなるので、マスタ側は次のデータ出力を待つことになる

I2C通信(3):スレーブからマスタへ受信の場合

スレーブアドレスが7bitの時のタイミングチャート



※赤：マスタ側、青：スレーブ側

- スレーブ側からのデータ入力に応じて、マスタ側はACKを自動返信する

LCDを動かそう

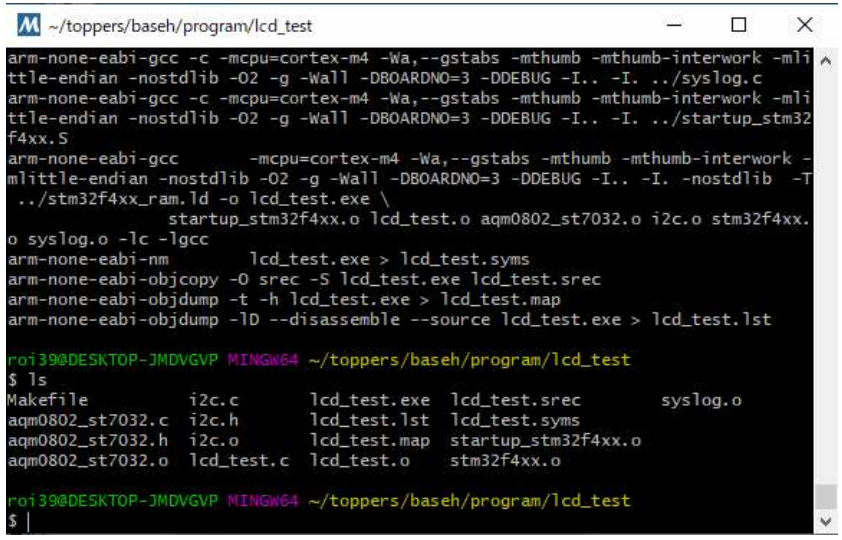
1. 回路図（LCD）の確認
2. I2C通信について
3. lcd_testプログラムの確認

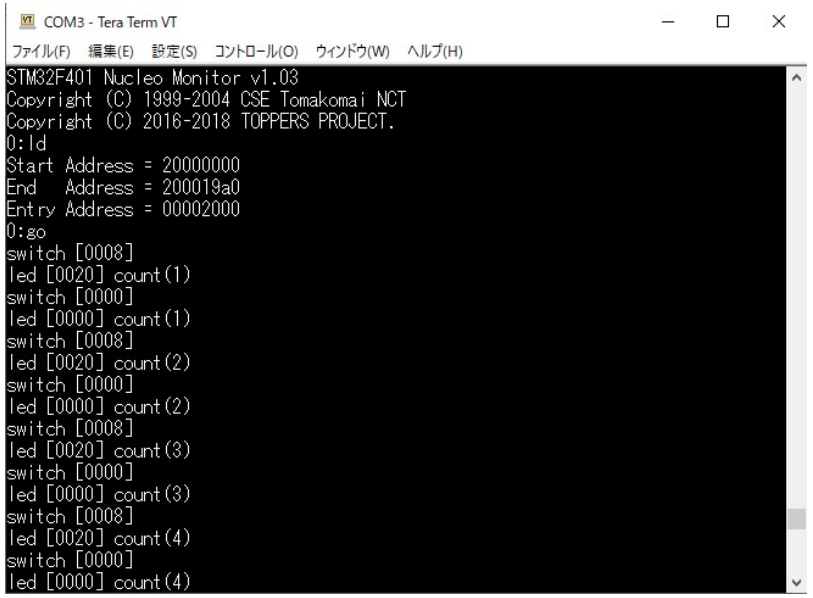
lcd_testプログラム：概要

- 構成ファイル
  - ../seviecall　　μITRON仕様の仮サービスコール
  - ../../asp/gdic/aqm0802_st7032
    AQM0802はTOPPERS BASE PLATFORMのGDICドライバを使用
  - ../../asp/pdic
    ドライバはTOPPERS BASE PLATFORMのPDICドライバを使用
  - lcd_test.c　　lcd_testメイン関数
  - Makefile　　メイクファイル
  - kernel_cfg.h　　仮サービスコールのセマフォIDの定義

lcd_testプログラム:ビルド

- MSYS2にてcdコマンドでprogram/lcd_testに移動
  - cd ../lcd_test(return)
- makeコマンドでビルド(Pico用)
  - make (return)
- ボードリセット後、lcd_test.srecをダウンロードして実行

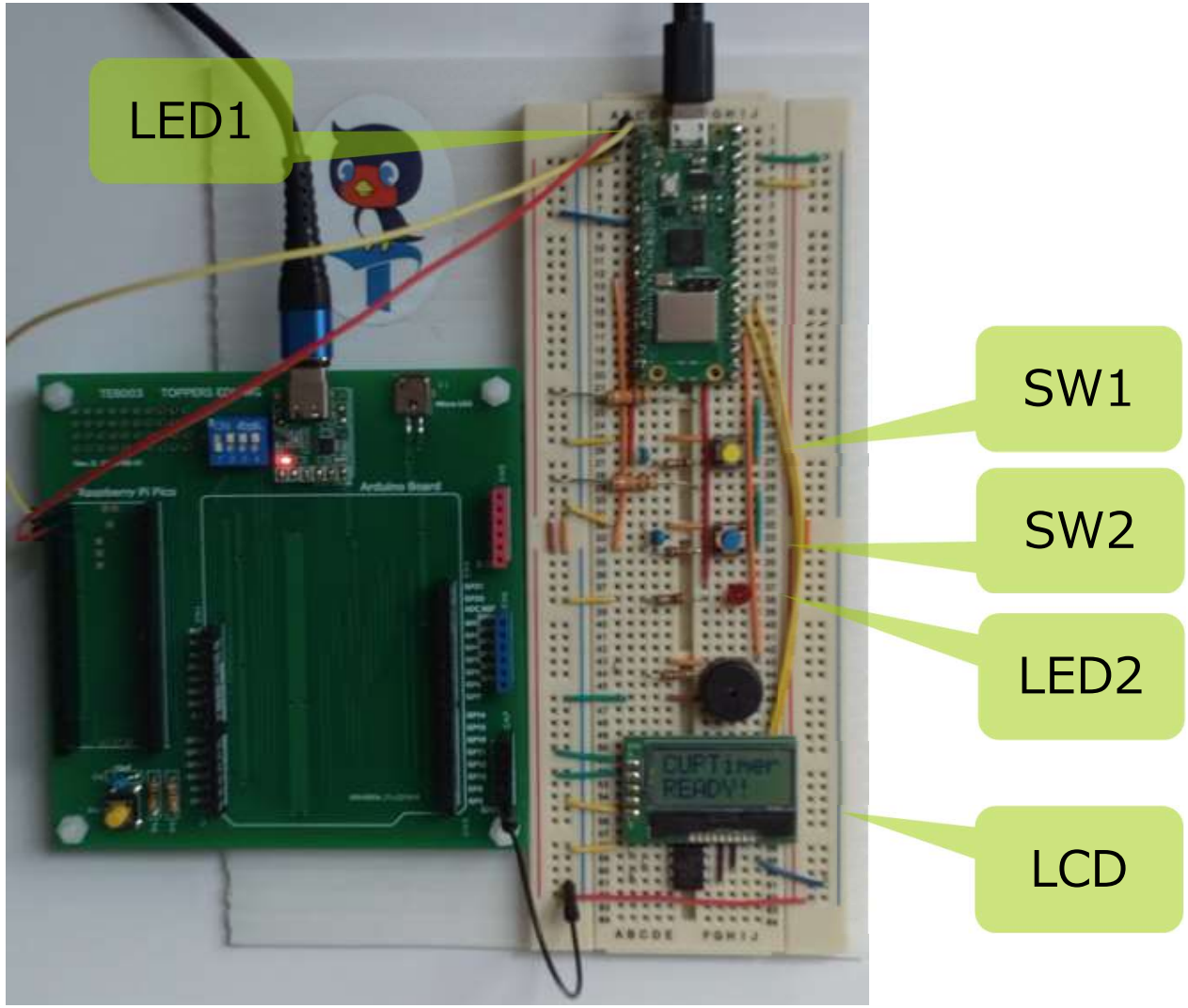




TOPPERS

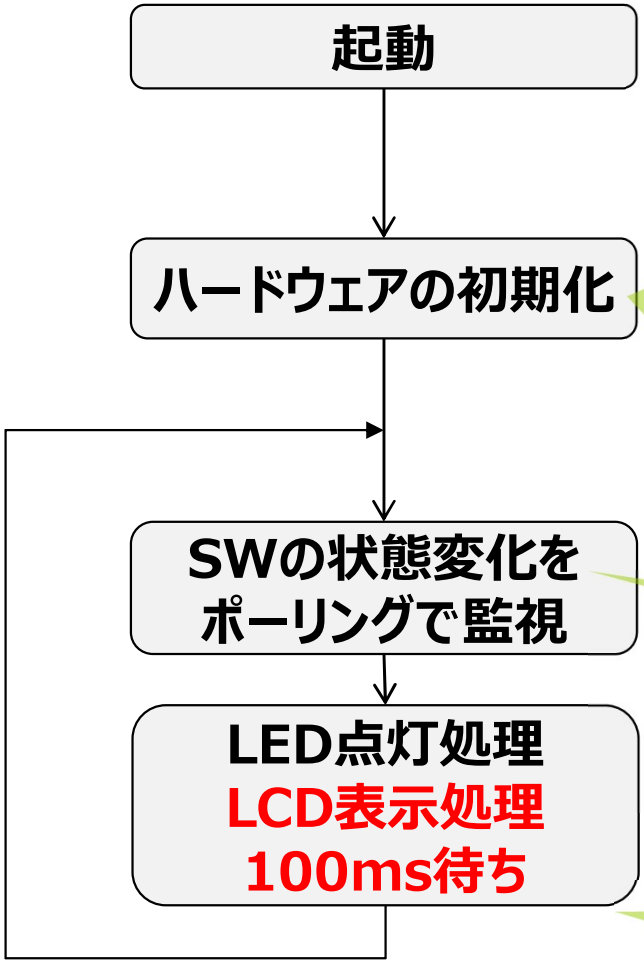
lcd_testプログラム：実行

- キー(SW1/SW2)を押下するごとにLCD上で演算、LEDも点灯
  ・・・SW1：+1 カウントアップ、SW2：-1 カウントダウン



lcd_testプログラムの説明：メイン関数

lcd_test/lcd_test.c



・LED初期化
・キー初期化
・**I2Cの初期化**
　※ポート:GIO,方向レジスタ:出力,初期信号:Hi
・AQM0802-PDICドライバの初期化
　※LCDの初期化
　※ **I2C通信によるCMD送信**
　→**スレーブアドレス（0x3E）** ※7bitアドレスモード
・ブザー初期化

・**SWの状態取り込み**

・**LCDの場所(0,0)を指定して、**
　LEDの値をLCD表示に反映する

配線

- PICOピンとブレッドボードの配線を再確認

RASPBERRY PI PICOピン	ブレッドボード
GP21	I2C1_SCL
GP20	I2C1_SDA
GND	-
GP19	LED2
GP18	BUZZER
GP8	PSW1
GP9	PSW2
+3.3V	+

カップラーメンタイマソフト説明

カップラーメンタイマの仕様

- 概要
  - カップラーメンの完成を知らせるタイマ
  - 30秒刻みで、お知らせ時間を設定
  - タイマをスタートしてから、設定した時間が経過したことブザーでLEDで知らせる
- スイッチとLEDの使い方
  - LCD　　：タイマの状態を表示
  - Buzzer：タイムアップ時ブザーで知らせる
  - LED2　：SW1/SW2が押されたことを表示
  - SW2　　：タイマを起動・停止
  - SW1　　:設定時間を３０秒延長
    タイマー停止中はシステム時間の表示切替

カップラーメンタイマの詳細仕様

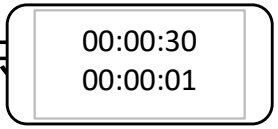
- 電源をONの間（プログラムが動いている間）、LCDに状態表示
- SW2がONになると設定時間を30秒にして、タイマを起動する
- SW2が再度ONになると、タイマを停止する
- タイマが動いている間にSW1がONになると、ONの間LED2を点灯させ、設定時間を30秒延長する
- タイマが動いている間は、led1を点灯する
- 設定時間が経過すると、led1を0.25秒間隔で点滅させて、15秒後に消灯する。その間ブザーを鳴らす。
- led1とLED2の点灯、消灯はLED2の輝度で表わす

カップラーメンタイマの拡張仕様

LCD表示を追加する

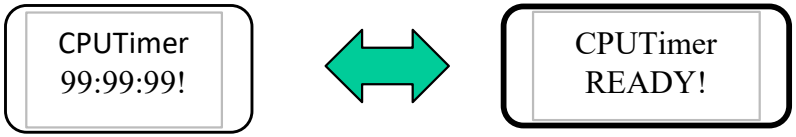
- LCDにタイマ停止の状態を表示する

- タイマが動いている間はタイマ設定時間と経過時間を表示する



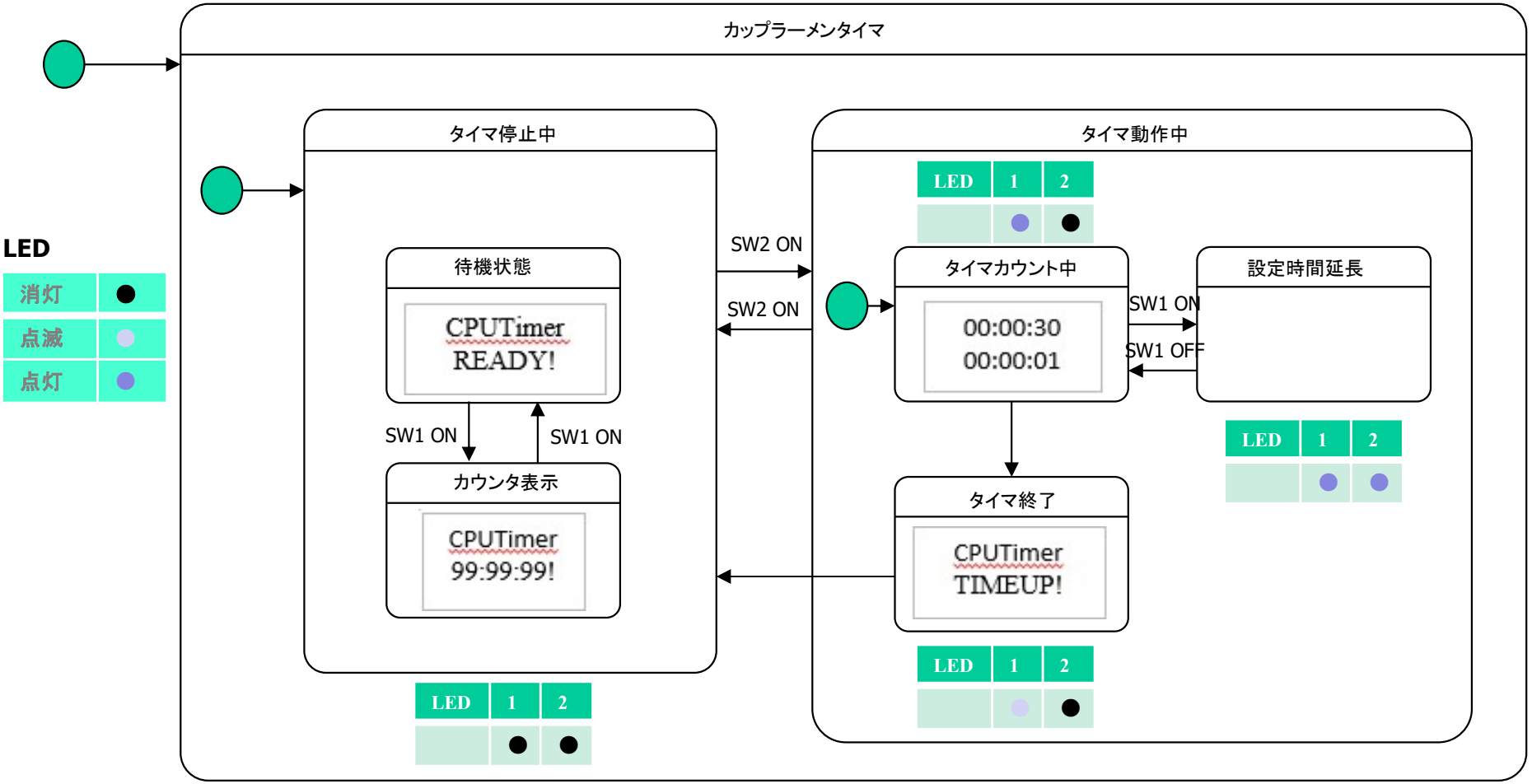
- タイマ終了を表示で知らせる。

- タイマ停止の状態でSW1がＯＮになると、ONの間LED2を点灯させ、カウンターの表示とＲＥＡＤＹ表示を切り替える



カップラーメンタイマの状態遷移図

- 状態：タイマ停止中、タイマ動作中



カップラーメンタイマプログラム cup_timer

- ハードウェアの操作に関してはこれまでに学習した内容を用いる
  - ブザーとキーを動かそう
    - ハードウェアの初期化
    - ＬＥＤ点灯と消灯
    - ＳＷの状態変化を監視
    - ブザーを鳴らす
    - タイマー割込みによる時間をカウントする
  - LCDを動かそう
    - ハードウェアの初期化
    - ＬＣＤ表示処理

作成の手順

- Step1 ：
  - デバイスドライバの作成
- Step2 ：
  - タイマによる基本時間（0.25秒）の作成
- Step3 ：
  - スイッチプロセスの作成
- Step4 ：
  - タイマプロセスの状態遷移判定と各状態の大枠作成
- Step5：
  - タイマプロセスの状態遷移判定と各状態の詳細
- Step6：
  - 拡張仕様（LCD表示）の作成

Step1：デバイスドライバの作成

LED, スイッチ, ブザー、タイマ、LCDの操作関数を作成

lcd_test の初期化関数と操作関数を流用

i2c.c/h aqm0802_st7032.c/h

- LED1と2の状態で輝度変えて表示する関数

```
void set_led_state(uint32_t led, uint8_t state)
```

- プッシュＳＷの状態を個別に読み込む関数

```
uint8_t get_push_state(uint16_t sw)
```

- ブザーモードの制御

```
void set_buzzer_mode(uint8_t mode)
```


TOPPERS

Step1：デバイスドライバの作成

set_led_stateの記載例

LED1を1、LED2を2とし、輝度は両方ONで100%

その他は(LED1+LED2設定)×25

```
void
set_led_state(uint32_t led, uint8_t state){
    uint16_t duty;

    if(state == ON)
        current_led |= led;
    else
        current_led &= ~led;

    if(current_led == LED_MASK)
        duty = 100;
    else
        duty = current_led * 25;
    led_out_sil(duty);
}
```

DUTYはLED2に設定しているので実際はLED2しか点灯しない

TOPPERS

Step1 : デバイスドライバの作成

LED, スイッチ, ブザー、タイマ、ＬＣＤの操作関数を作成
lcd_test の初期化関数と操作関数を流用
i2c.c/h aqm0802_st7032.c/h

- LCDのロケート設定

```
void  LCD_locate(h, x, y)
```

- LCDのデータ送信

```
void LCD_puts(h, buffer)
```

Step2 :タイマによる基本時間（0.25秒）の作成

- グローバル変数
  - 1msec毎にインクリメントする変数（BaseTime）を用意
  - TIMR0:Arарm0をタイマとして使用する

- タイマ割込みハンドラ
  - 1msec周期で実行され，BaseTimeをインクリメント

- メイン関数
  - BaseTimeを用いて250msec周期の処理を実現
    - 各ハードウェアの初期化と割込みの許可
    - 無限ループ内でBaseTimeをチェックする

Step2 : プログラム例 タイマ割込みハンドラ

- グローバル変数 BaseTime
  - uint32_t (32bit)とする

```
volatile uint32_t BaseTime;
```

- タイマ割込み
  - タイマ割り込み初期化

```
void
timer0_init(void);
```

Step2 : プログラム例 タイマ割込みハンドラ

- タイマ割込み
  - タイマ割込みハンドラ

```
void
Timer0_IRQHandler(void){
    uint64_t delay_time;
    uint32_t ints = sil_rew_mem((uint32_t *)(TADR_TIMER_BASE+TOFF_TIMER_INTS));

    BaseTime += TIMER_GRANULE;
    /*
     *  割込みクリア
     */
    sil_wrw_mem((uint32_t *)(TADR_TIMER_BASE+TOFF_TIMER_INTR), ints);
    delay_time = sil_rew_mem((uint32_t *)(TADR_TIMER_BASE+
                TOFF_TIMER_TIMERAWL)) + wrap_time;
    sil_wrw_mem((uint32_t *)(TADR_TIMER_BASE+
                TOFF_TIMER_ALARM0+ALARMNO*4), (uint32_t)delay_time);
}
```


TOPPERS

Step2 : プログラム例 メイン関数

- 250msec周期の処理

```
#define TIMER_GRANULE 250     /* 時間単位          */

void
main(void){
    uint32_t timer250 = TIMER_GRANULE;
    .....
    BaseTime = 0;

    for (;;) {
      if (timer250 <= BaseTime) { /* 250m周期で実行 */
            /* 現在時刻の250msec後を設定 */
              timer250 += TIMER_GRANULE;
      }
    }
}
```

タイマ割込みハンドラで250ms周期でインクリメントされる

TOPPERS

Step3 :スイッチプロセスの作成

- イベントの定義とビット割り当て
  - 開始　　：EVT_TIMER_TRIG　　0x01
  - 時間アップ　：EVT_TIMER_COUNT　0x04

- グローバル変数
  - スイッチの状態を保持するための変数（Push1, Push2）
  - イベント通知用変数（Event）

- スイッチプロセス
  - SW1,SW2の変化をチェックし，イベント通知用変数をセットする
  - SW1の状態に応じて，LED2を点灯・消灯させる

Step3 : プログラム例 SW2の状態取得

- SW2の状態をチェックし，状態が変化していれば，イベント変数に対応するイベントをセットする

```
void
switch_process(void){
    uint8_t  sw;
    sw = get_push_state(PUSH2);
    if (Push2 != sw) {
        if (sw == ON) {
            Event |= EVT_TIMER_TRIG;
        }
        Push2 = sw;
    }
```

開始イベント

TOPPERS

Step3 : プログラム例 SW1の状態取得

- SW1の状態をチェックし，状態が変化していれば，イベント変数に対応するイベントをセットする。
- SW1がONの間LED2を点灯する

```
sw = get_push_state(PUSH1);
if (Push1 != sw) {
    if (sw == ON) {
        Event |= EVT_TIMER_COUNT;
        set_led_state(LED2, ON);
    } else {
        set_led_state(LED2, OFF);
    }
    Push1 = sw;
}
```

時間UPイベント

LED2:ON

LED2:OFF

TOPPERS

Step4 : タイマプロセスの大枠とLED1点灯の実現

タイマプロセスの状態遷移判定と各状態の大枠作成
led1の点灯の実現（実際はLED2の点灯）

- タイマプロセスの状態を示す列挙型（TIMER_MODE）
  - 計時終了 ： STOP_TIMER_MODE
  - 計時中 ： ACT_TIMER_MODE
  - タイムアウト： TIMEOUT_MODE
- タイマプロセスのローカル変数（static変数）
  - タイムアウト時間を保持する変数（maximum_time）
  - 現在の計時時間を保持する変数（current_time）
  - カップラーメンタイマのモードを保持する変数（timer_mode）
  - ブザーのオン・オフ管理変数（buzzer_mode）
  - アラーム時間設定用変数（alarm）
  - LED1の点灯状態を保持する変数（led1）

Step4：列挙型とローカル変数

- タイマプロセスの状態を示す列挙型（TIMER_MODE）

```
enum TIMER_MODE {
    STOP_TIMER_MODE,   /* 計時終了モード      */
    ACT_TIMER_MODE,    /* 計時中モード        */
    TIMEOUT_MODE       /* タイムアウトモード */
};
```

- タイマプロセスのローカル変数（static変数）
  - タイマプロセスは250msec周期で呼び出され，終了するため変数の値を保存するために static変数とする

```
static uint32_t maximum_time;
static uint32_t current_time;
static uint8_t timer_mode = STOP_TIMER_MODE;
static uint8_t buzzer_mode = OFF;
static uint8_t led1 = OFF;
static uint8_t alarm = 0;
```

Step4 : 定数マクロの定義

- 各定数をマクロで定義する
  - タイムアウト時間の初期値
  - 時間アップ（延長）単位
  - タイムアウト時のLED2の点滅時間

```
#define INIT_TIME          30   /* スタート時のタイムアウト時間      */
#define TIME_UNIT          30   /* 時間アップ単位                    */
#define TIMEOUT_BLINK      15   /* タイムアウト時の点滅時間          */
```

Step4 : プログラム例 タイマプロセス状態遷移判定

- イベント通知用変数を参照してカップラーメンタイマのモード変数（状態）を設定する

```
if (Event != 0) {                /* switch_processより通知 */
      if (Event & EVT_TIMER_TRIG) {
         if(timer_mode == STOP_TIMER_MODE){
            timer_mode = ACT_TIMER_MODE;    /* 開始 */
         }else{
             timer_mode = STOP_TIMER_MODE;    /* 停止 */
      }
      if (Event & EVT_TIMER_COUNT) {    /* 時間アップ */
         if (timer_mode == ACT_TIMER_MODE) {

         }
      }
      Event = 0;
 }
```


TOPPERS

Step4 : プログラム例 各状態毎の処理

- timer_modeによりモードを判定して実行

```
switch (timer_mode) {
    case ACT_TIMER_MODE:     /* 計時中モード */
       計測中の状態設定
       break;
   case STOP_TIMER_MODE:  /* 計時終了モード*/
       停止設定時の状態設定
       break;
  case TIMEOUT_MODE:  /*タイムアウトモード */
       タイムアウト時の状態設定
       break;
   default:
       break;
   }
```


TOPPERS

Step4 : プログラム例 設定をデバイスに反映

- 状態設定後、設定をLCD、ブザー、LEDに反映

```
    LCD_locate(&CLcdHandle, 0, 0);
    LCD_puts(&CLcdHandle, &lcd_buf[0][0]);
    LCD_locate(&CLcdHandle, 0, 1);
    LCD_puts(&CLcdHandle, &lcd_buf[1][0]);

    set_buzzer_mode(buzzer_mode);
    set_led_state(LED1, led1);   /* LED1の設定 */
}
```

Step5 : タイマプロセスの状態遷移判定と各状態の詳細

- 状態遷移判定での処理
  - 開始
    - current_time を0で初期化
    - maximum_time を設定し30秒でタイムアウトするよう設定
  - 停止
    - alarm を0にしてled1を消灯
  - 時間アップ
    - 計時中モードであれば maximum_time を30秒延長

Step5：タイマプロセスの状態遷移判定と各状態の詳細

- 各状態（モード）での処理
  - 計時中モード
    - タイムアウトしていればモードをタイムアウトモードに移行させる
    - led1を点灯させる
    - ブザーをONにする
  - タイムアウトモード
    - 15秒経過したらled1の点滅を停止し，計時終了モードに移行する
  - 計時終了モード
    - led1を消灯させる
    - ブザーをOFFにする

Step5：プログラム例 状態遷移判定の詳細

- 状態遷移と共に変数を設定する

```
if (Event & EVT_TIMER_TRIG){
    if(timer_mode == STOP_TIMER_MODE){
        timer_mode = ACT_TIMER_MODE;
        current_time = 0;
        maximum_time = INIT_TIME * TO_SEC;
    }else{
        timer_mode = STOP_TIMER_MODE;
    }
}
if (Event & EVT_TIMER_COUNT) {      /* 時間アップ */
    if (timer_mode == ACT_TIMER_MODE) {
        maximum_time += TIME_UNIT * TO_SEC;
    }
    else if(timer_mode == TIMEOUT_MODE)
        timer_mode = STOP_TIMER_MODE;
    else{
        if(base_disp_mode == OFF)
            base_disp_mode = ON;
        else
            base_disp_mode = OFF;
    }
    Event = 0;
}
```


TOPPERS

Step5 : プログラム例 各状態の詳細

- 計時中モード

```
case ACT_TIMER_MODE:
    if (current_time >= maximum_time) {
      /* タイムアウト */
       alarm = TIMEOUT_BLINK*TO_SEC;
       timer_mode = TIMEOUT_MODE;
    }
    current_time++;
    led1 = ON;
    break;
```

Step5 : プログラム例 各状態の詳細

- 計時終了モード

```
case TIMEOUT_MODE:
    if(base_disp_mode == OFF)
        strcpy((char *)&lcd_buf[1][0], title2);
    else
        set_time(&lcd_buf[1][0], BaseTime/TIMER_GRANULE);
    alarm = 0;
    buzzer_mode = OFF;
    led1 = OFF;
    break;
```

Step5 : プログラム例 各状態の詳細

- タイムアウトモード

```
case TIMEOUT_MODE:
    if(alarm > 0){
        if(led1 == ON)
            led1 = OFF;
        else
            led1 = ON;
        buzzer_mode = ON;
        alarm--;
    }else
        timer_mode = STOP_TIMER_MODE;
break;
```

Step6：拡張仕様（LCD表示）の作成

- LCD表示文字と変数の定義

```
static const char title1[] = "CPUTimer";
static const char title2[] = "READY!   ";
static const char title3[] = "TIMEUP! ";

uint8_t lcd_buf[2][12];
```

Step6：拡張仕様（LCD表示）の作成

- 各状態（モード）での処理
  - 計時中モード
    - 設定されたタイムアウト時間を表示する
    - 計時中の時間を表示する
  - タイムアウトモード
    - title1とtitle3 を表示する
  - 計時終了モード
    - title1とtitle2 を表示する
    - SW1がONになるtitle2を、カウンター表示に切り替える

Step6 :プログラム例

- 各状態（モード）での処理

```
case ACT_TIMER_MODE:
    set_time(&lcd_buf[0][0], maximum_time);
    set_time(&lcd_buf[1][0], current_time);

case TIMEOUT_MODE:
    strcpy((char *)&lcd_buf[0][0], title1);
    strcpy((char *)&lcd_buf[1][0], title3);

case STOP_TIMER_MODE:
   strcpy((char *)&lcd_buf[0][0], title1);
   if(base_disp_mode == OFF)
      strcpy((char *)&lcd_buf[1][0], title2);
   else
      set_time(&lcd_buf[1][0], BaseTime/TIMER_GRANULE);
```


TOPPERS

Step6 :プログラム例

- timer_processにLCD表示処理を追加

```
LCD_locate(&CLcdHandle, 0, 0);
LCD_puts(&CLcdHandle, &lcd_buf[0][0]);
LCD_locate(&CLcdHandle, 0, 1);
LCD_puts(&CLcdHandle, &lcd_buf[1][0]);
```

Step6 :プログラム例

- 時間を時：分：秒の文字列に変換する

```
void set_time(uint8_t *buf, uint32_t time)
{
    int wk = time/TO_SEC;
    set_value(&buf[6], wk % 60);
    wk = wk / 60;
    buf[5] = ':';
    set_value(&buf[3], wk % 60);
    wk = wk / 60;
    buf[2] = ':';
    set_value(&buf[0], wk);
}
```

まとめ

著者リスト

- はじめに
- 開発環境の説明
- デバッガとLCDの点灯確認
  - 竹内 良輔（教育WG主査）
- ブザーを動かそう
- LCDを動かそう
  - 田中裕貴（(株)リコー）
- カップラーメンタイマソフト説明
  - 森田浩（(株)アドバンスドデータコントロール）