デバッグキャンプ

SWEST25
s4c, s5c

本セッションについて

- SWESTに参加者でも若手技術者や学生向けのチュートリアル系のセッションです
- デバッグのやり方って，あまり学ぶ機会無いよね？と言う事で，SWEST実行委員な人々のノウハウを集約して座学＆実習形式のセッションを企画してみました
- 当日，参加者の皆さんのノウハウなども共有できたらと思っています

本セッションを担当する愉快な仲間たち

9/1(金) 12:30〜13:10 セッションs4c
デバッグキャンプ

講師
及川達裕

四国職業能力開発大学校

講師
山本健太

デンソークリエイト

講師
大栄豊

オーバス(デンソー)

コーディネータ
細合 晋太郎

東京大学

裏で色々助けてもらいました！あざます！！１

対象とする参加者

- マイコンボードを扱う人，扱いたい人
  - OSなどを含めないベアメタルな開発
  - ハードウェアも含めたデバッグ

- 研究でボードやロボットを作る必要のある学生さん
- 普段あまりボードに触れる事のない若手の組込み技術者など

概要

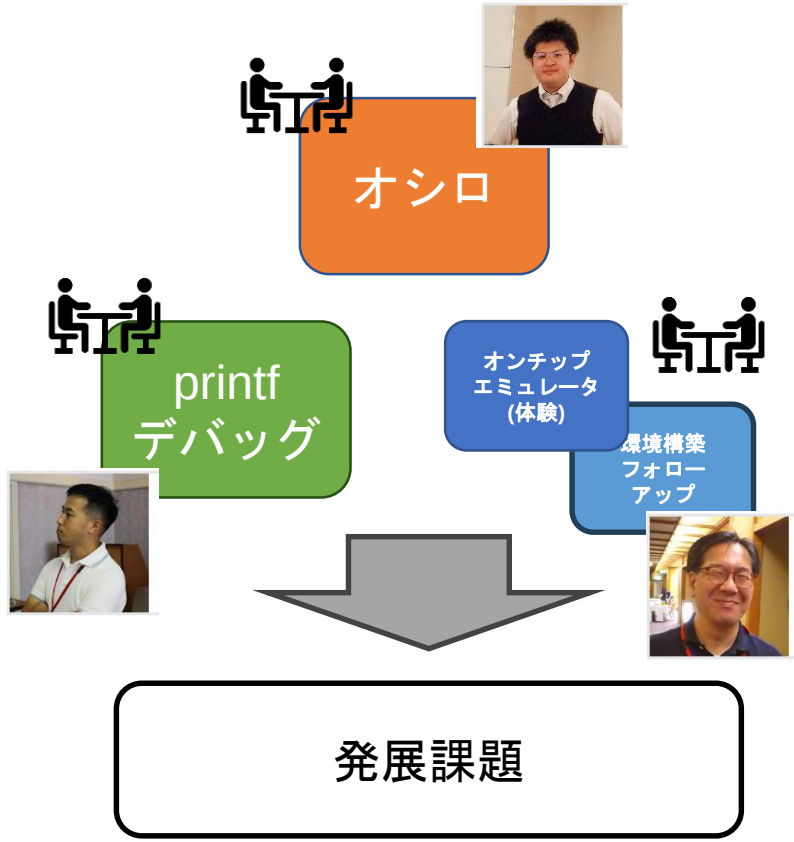
- 目的
  - Printデバッグを深堀し，Printデバッグの一歩先を知ろう！
- 目標
  - Printデバッグについて一歩踏み込んだノウハウを得られること
  - オンチップ・エミュレータを活用したデバッグとの使い分けを理解できること
  - オシロスコープを利用したデバッグ

- 時間　　：40分+70分の構成
- 対象　　：企業の若手技術者・学生(B3～M2程度)
- 場所　　：水明館・如月の間
- その他　：実機ありのハンズオンあり

実施形態

- s4c：座学(40min)
  - 聴講形式，デバッグの方法の紹介
  - 演習で使用するXIAOGYANの紹介

- s5c：演習(70min)
  - 演習テーマ別に3つの島を用意します.
    1. テーマ①：Printデバッグ
    2. テーマ②：オンチップ・エミュレータによるデバッグ体験
       ＋環境構築フォローアップ
    3. テーマ③：オシロスコープを利用したデバッグ
  - まとめ



ターゲットボード：XIAOGYAN

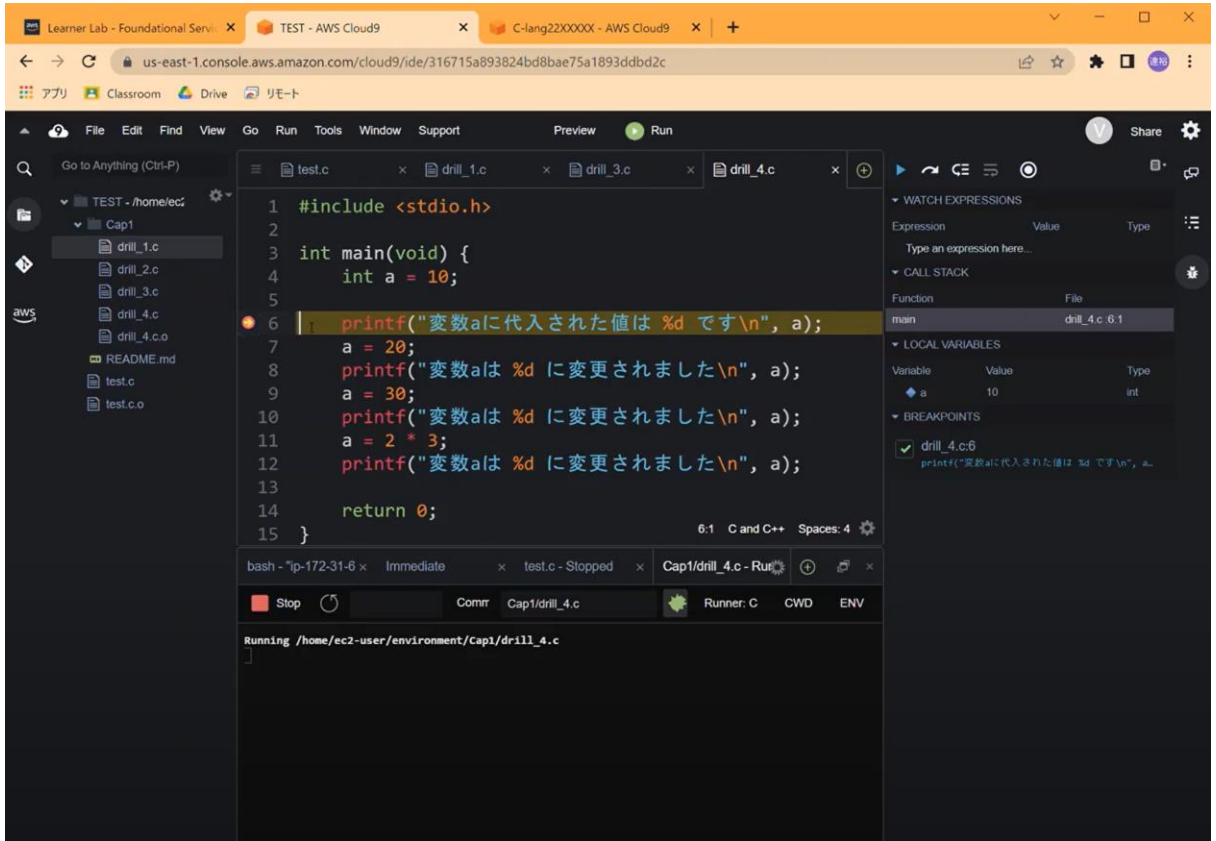
- IoT ALGYANの8周年イベントに合わせて開発されたボードです
  - https://github.com/algyan/XIAOGYAN
- マイコンはSeeeduino XIAO ESP32C3
- 周辺機器としてロータリエンコーダやLEDマトリックスなどを完備

座学編

① Printデバッグ

(PC上で動く)C言語のデバッグ方法

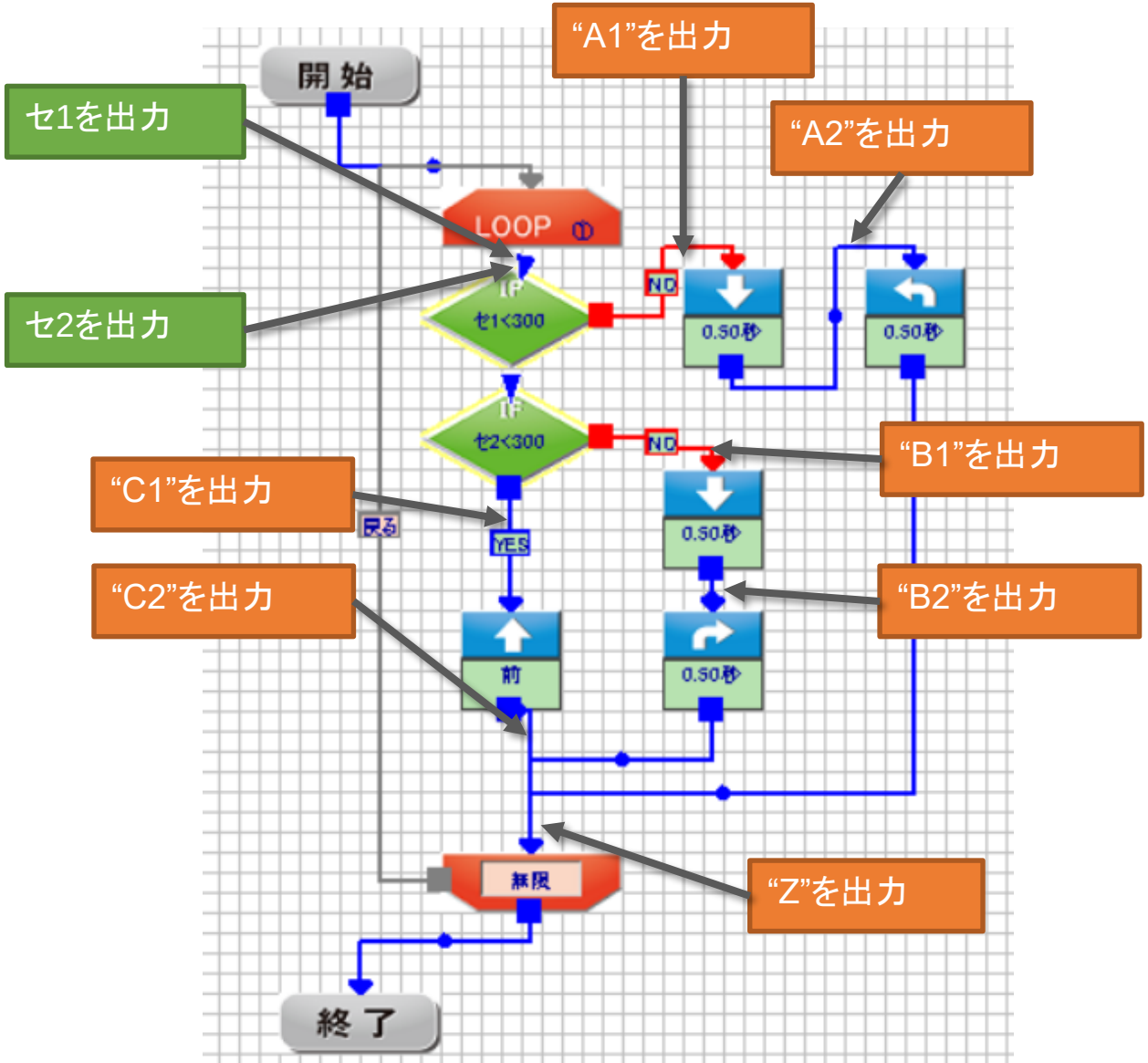
- C言語を初めて学んだ際には，どのような方法でデバッグしましたか？



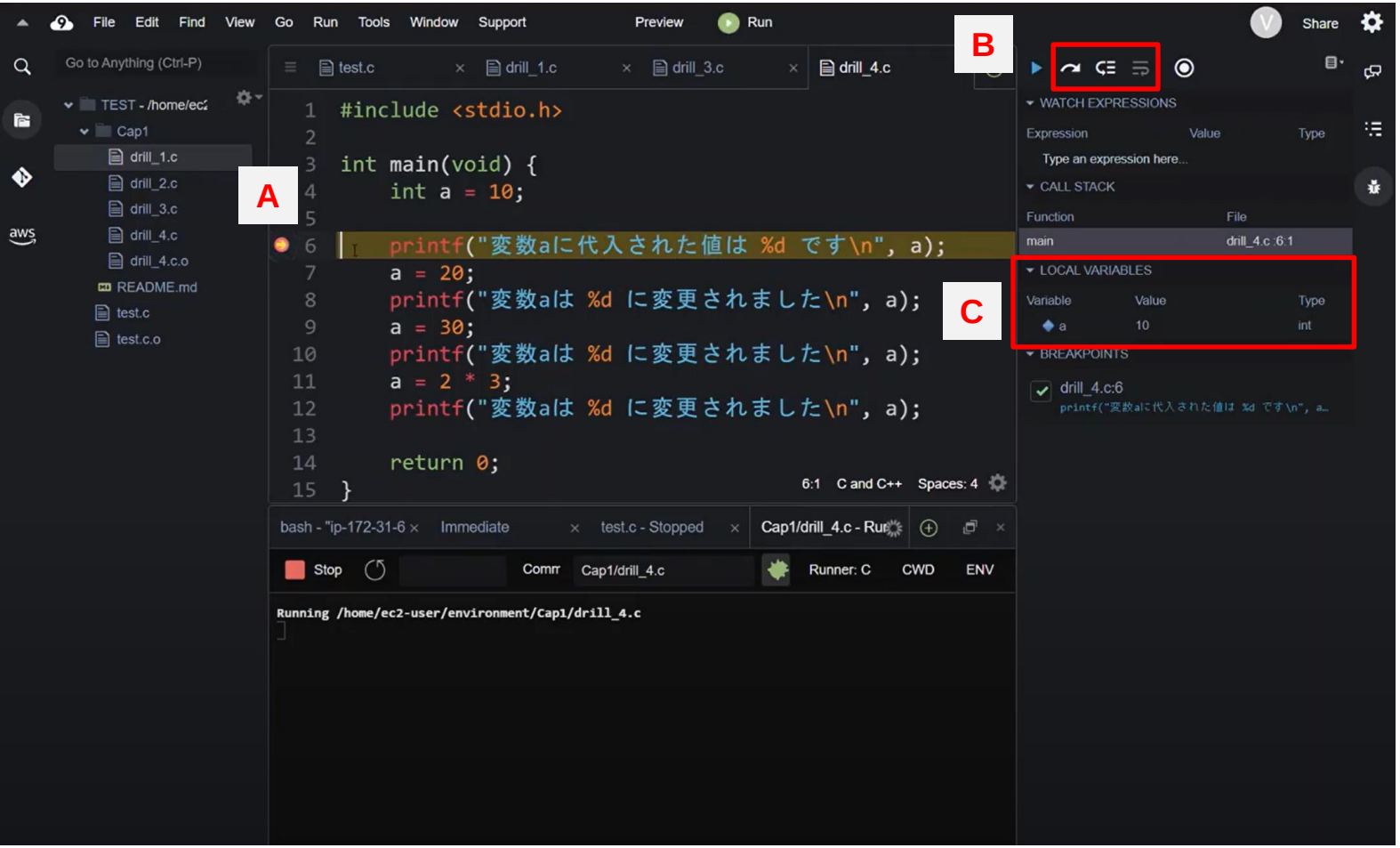
1. Printf関数を使ったデバッグ
2. ブレイクポイントを用いたデバッグ

①Printf関数を使ったデバッグで出来ること

- 変数の中身を確認する
  - 例えば、ADから取得したラインセンサ値、距離センサ値の確認
  - カウンタの値の確認
  - 変化しやすい値は、その値によって処理が分岐することが主である
- 内部フローを確認する
  - 特定のif文の中に入ったら’a’を表示する、if文に入らなかったら’z’を表示する。など
  - 自身の作成したプログラムが意図したタイミングで分岐や繰り返しをしているかを判断



②オンチップ・エミュレータを用いたデバッグでできること



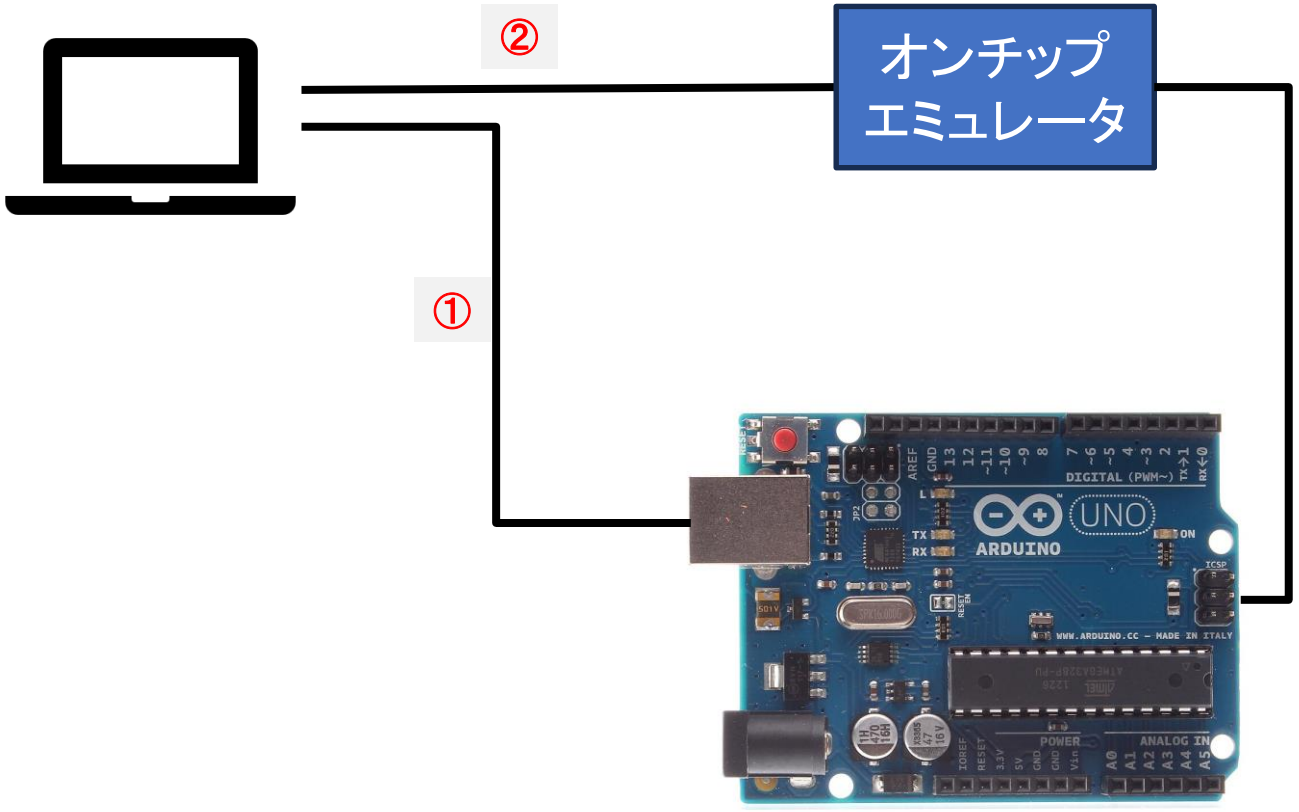
参考：AWS Cloud9におけるC言語のデバッグ画面

A) ブレイクポイント
- プログラムを任意の行で一時停止できる機能

B) ステップ実行
- 一時停止したコードを1行ずつ実行する事ができる

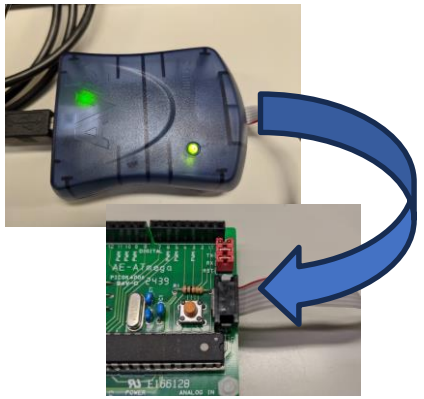
C) 変数ウォッチ
- 一時停止している時の任意の変数の値を確認する事ができる

マイコンのプログラムのデバッグ方法



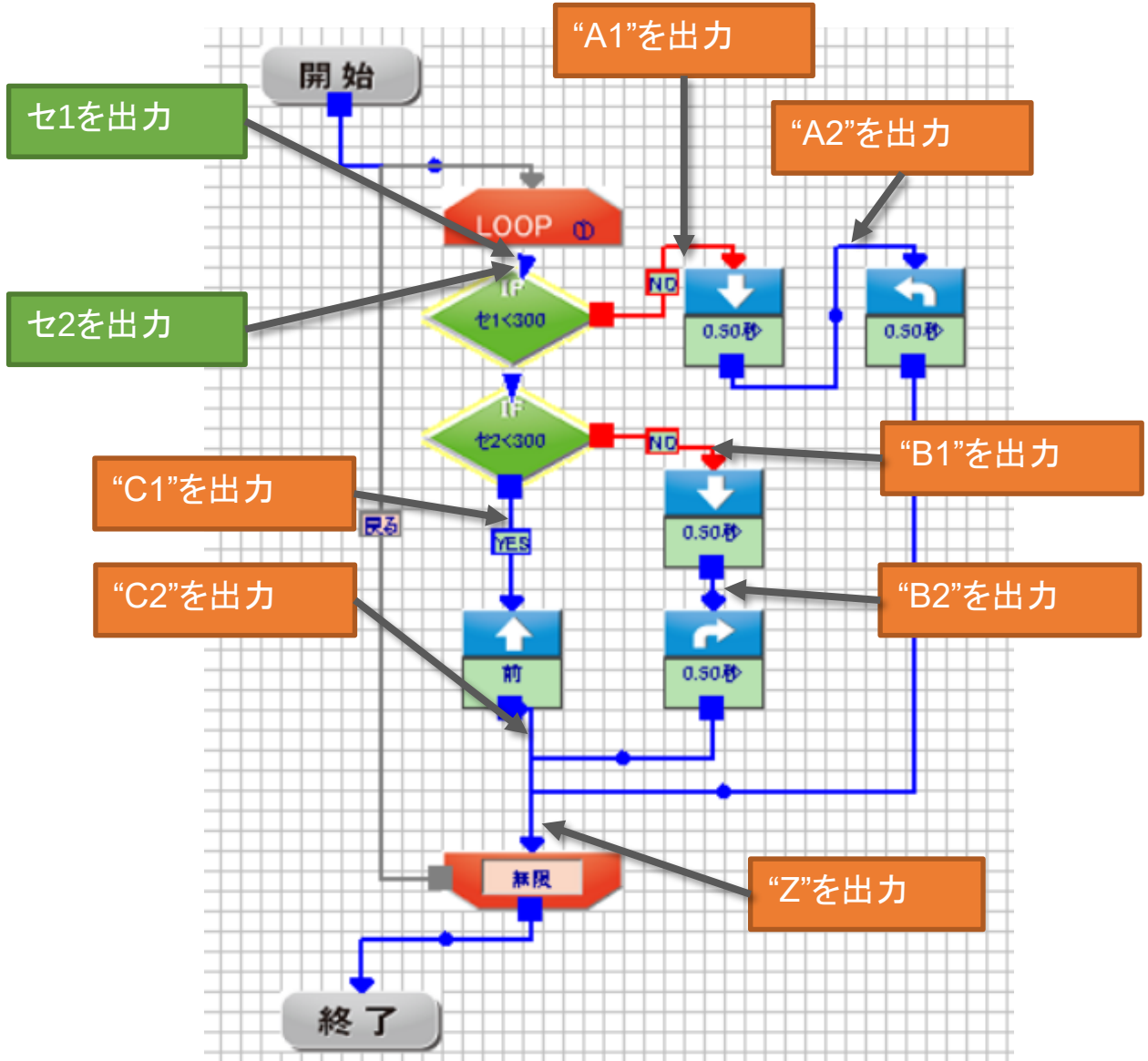
oomlout - ARDU-03_02, CC 表示-継承 2.0, https://commons.wikimedia.org/w/index.php?curid=28380408による

1. 通信を用いたPrintデバッグ
2. オンチップエミュレータによるデバッグ
   - ブレイクポイントを用いたデバッグが出来るようになる
   - ただし，専用のデバッガが必須
     - Arduinoの場合はAVR ISPなど



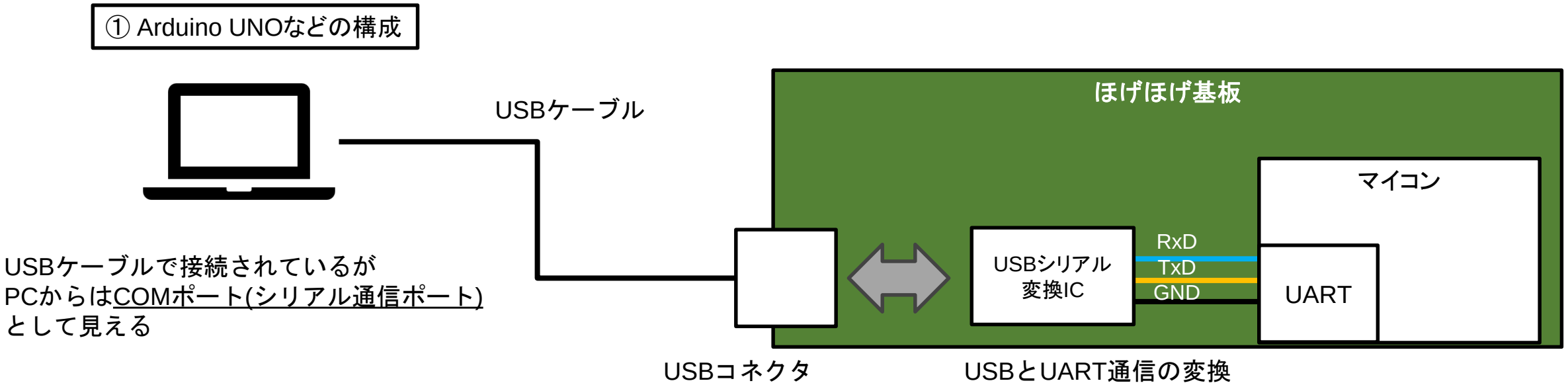
通信によるPrintデバッグで出来ること

- 変数の中身を確認する
  - 例えば、ADから取得したラインセンサ値、距離センサ値の確認
  - カウンタの値の確認
  - 変化しやすい値は、その値によって処理が分岐することが主である
- 内部フローを確認する
  - 特定のif文の中に入ったら’a’を表示する、if文に入らなかったら’z’を表示する。など
  - 自身の作成したプログラムが意図したタイミングで分岐や繰り返しをしているかを判断

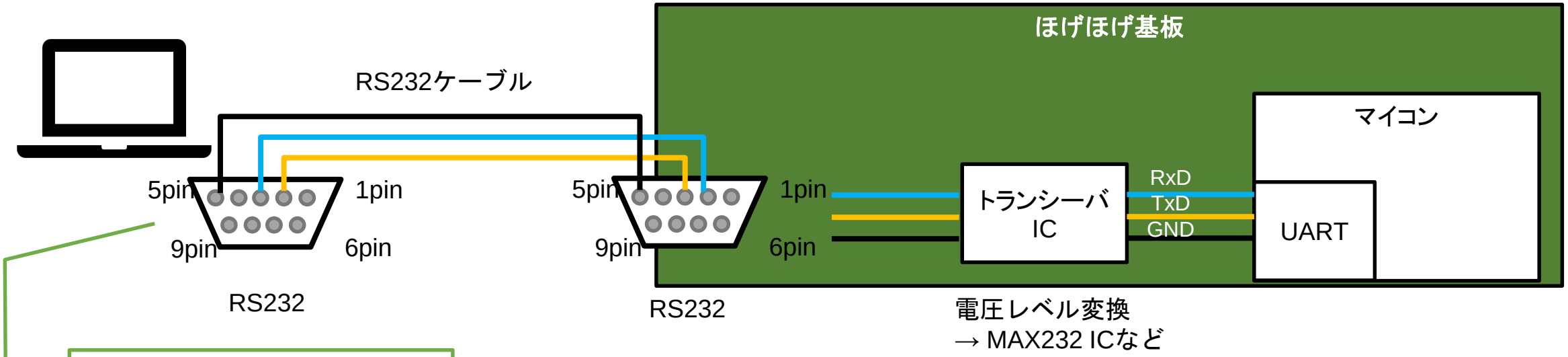


通信って？

- シリアル通信(UART)を用いる事が多い
  - どんなマイコンでも持っていることが多いIFである
  - 扱いが比較的容易な通信である
  - 一方で通信速度はあまり速くない(9600bps～115200bpsが主)



② 一昔前 or FA業界などでよくある構成



最近のPCにはRS232コネクタが
無いのでUSB-RS232変換
ケーブルを使う

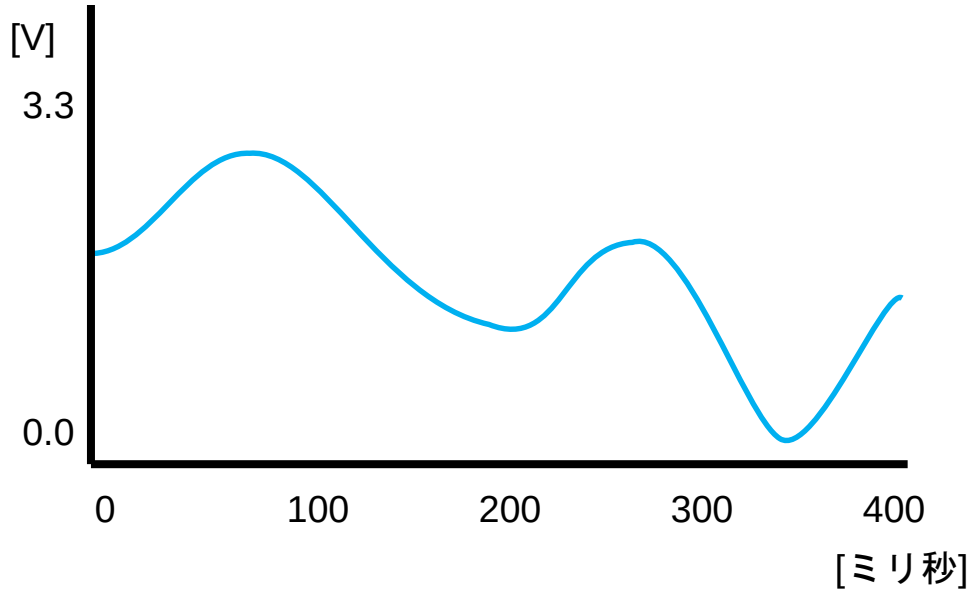
USB-シリアル変換ケーブル

RS232コネクタ(オス)

RS232コネクタ(メス)

通信によるPrintデバッグの良いところ

- 組込みシステムは「時間軸」があることが特徴
- ブレイクポイントを用いたデバッグでは，処理を止めてしまうので時間軸をぶった切る事になる
  - マイコンを動作させながら特定の値の変化を見ることも大切



センサ値などはまさに動作させながら挙動を追いたい

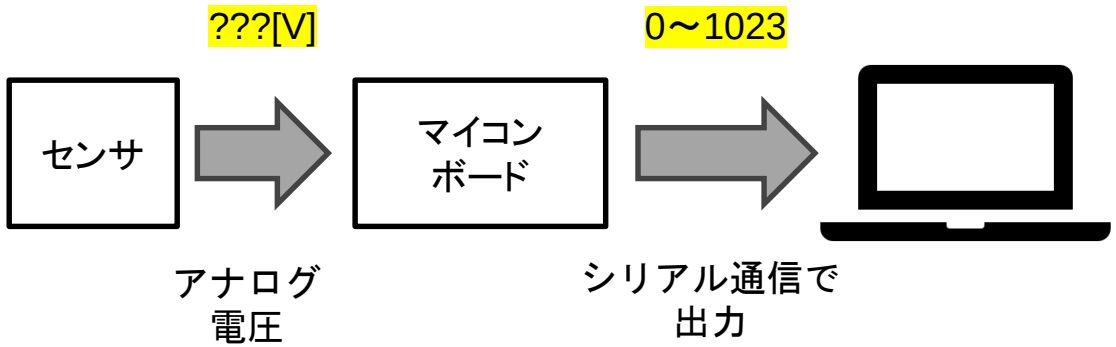
ログから動作を追う！

通信によるPrintデバッグの問題点①

- 例) UART通信で，0～1023のセンサ値を確認したい.

コード例

```
void setup() {
  Serial.begin(9600);                  // シリアル通信の初期化
}

void loop() {
  int value = analogRead(A0);          // アナログ入力(センサ値の取得)

  Serial.print("sensor = ");            // センサ値の出力
  Serial.println(value);               //

  delay(100);                          // 100msの待ち
}
```



通信によるPrintデバッグの問題点②

- **Printの処理には時間がかかること**
- 例) UART通信で，0～1023のセンサ値を確認したい.
  - 9600bpsで通信した場合，最大で4msec近く時間がかかる
    9600bit/sec = _____byte/sec
    1秒あたり_____byteのデータを送信可能
    今回は1～4文字(byte)のデータを送りたい為，最大で_____msec/回の時間がかかる事が予期される
- 吐き出す情報量が増えれば増えるほど，処理時間がかかる
  - 他の処理への影響が発生しやすい

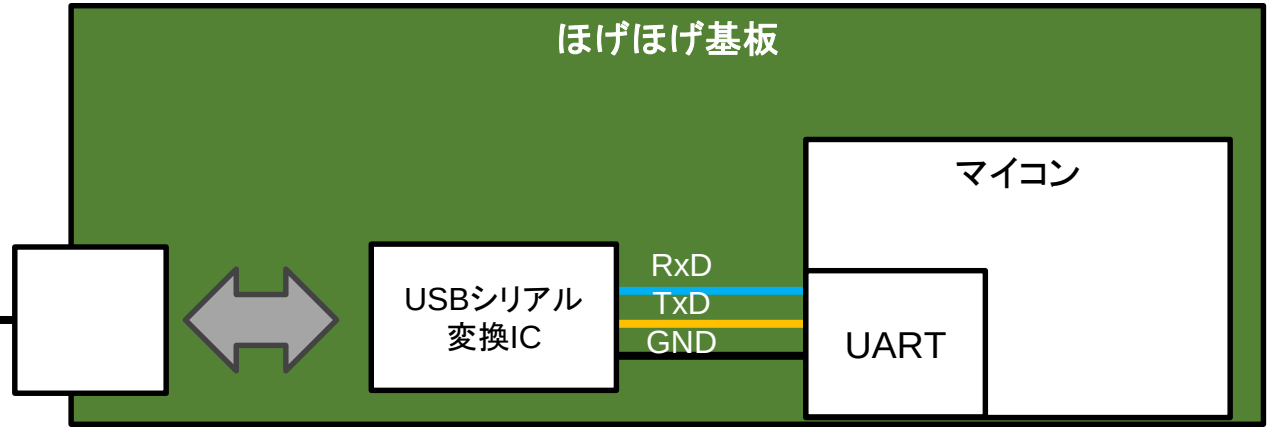
では，どうしたらいいのか？

- Printにかかる処理時間を考慮・想定した上で活用する
  - UARTであれば，通信速度から見積もることは可能

- でも…近頃のボードは，単純なUARTばかりとも言えない事も?
  - USB CSC, Bluetoothシリアルなど

① Arduino UNOなどの構成

USBケーブル



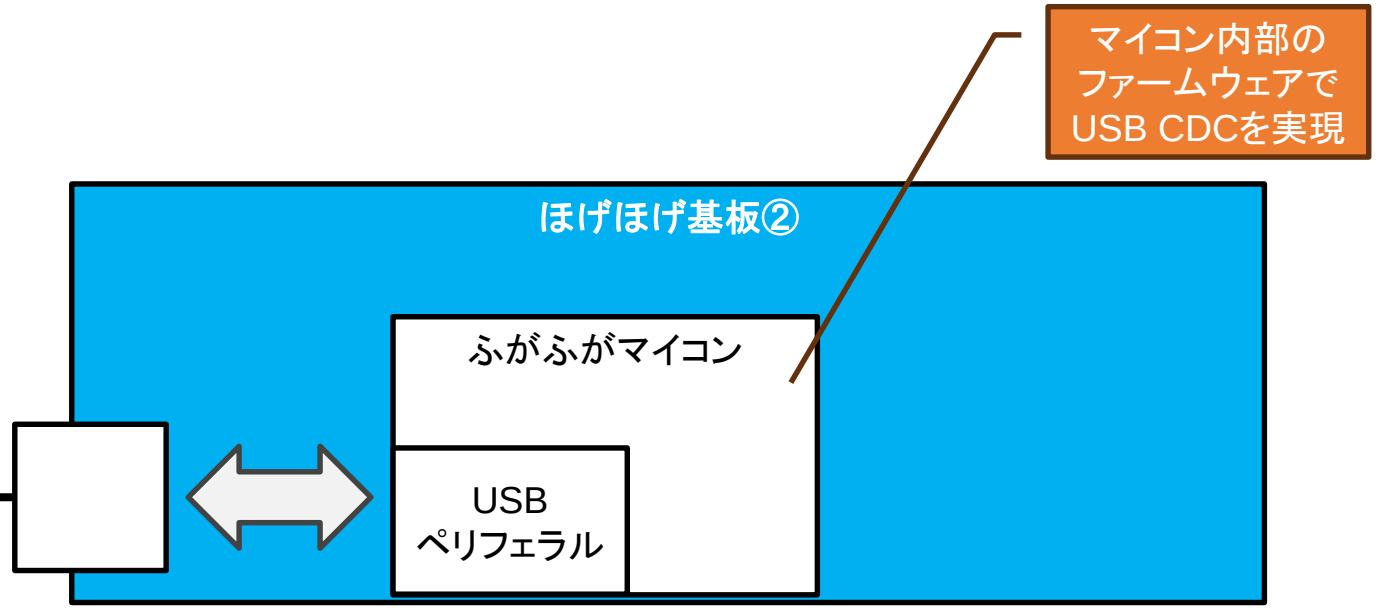
USBケーブルで接続されているが
PCからはCOMポート(シリアル通信ポート)
として見える

USBコネクタ

USBとUART通信の変換

③ USB CDCな構成

USBケーブル



USBケーブルで接続されているが
PCからはCOMポート(シリアル通信ポート)
として見える

USBコネクタ

ソフトウェア的にシリアル通信を実現
①の構成に比べてCPUリソースが食われる＝処理時間が発生

① IO制御+オシロで時間計測

- 以下の処理を記述して，LEDのON/OFFの時間をオシロスコープで計測すると，Printにかかる処理時間が見えてくる
  - 実測ベースではあるが，処理時間を把握する事が可能になる

① LEDをONする
② Printする
③ LEDをOFFする

- ただし，以下のような場面もある
  - オシロスコープがない
  - 都合の良いIOがない

② ソフトウェアで経過時間をカウントする

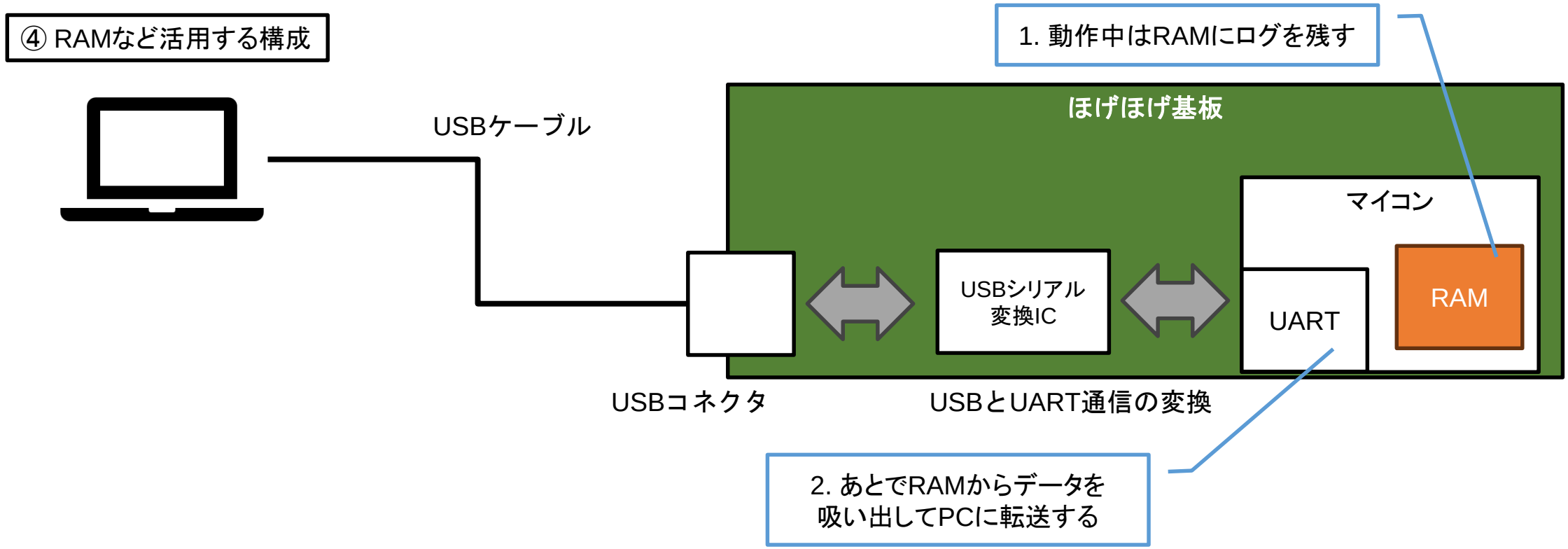
- 経過時間を計測して，その差分から**②(Printする)**にかかった処理時間を計測する手もあります

```
① time1 = micros();
② Printする
③ time2 = micros();
④ time_diff = time2 -time1;        // 経過時間
⑤ time_diffをPrintする
```

- micros関数
  - Arduino系のライブラリを使用すると活用できる関数
  - ボードがプログラム実行を開始した時から現在までの経過時間を<u>マイクロ秒単位</u>で返す
  - 同等の関数が無い場合は，タイマなどを活用して自作しましょう！

では，どうしたらいいのか？(別案)

- Printにかかる処理時間を考慮・想定した上で活用する
  - そうは言っても，処理時間がシステムに致命的な影響を及ぼすことも
    → ログを残す為にかかる時間を極力短くする



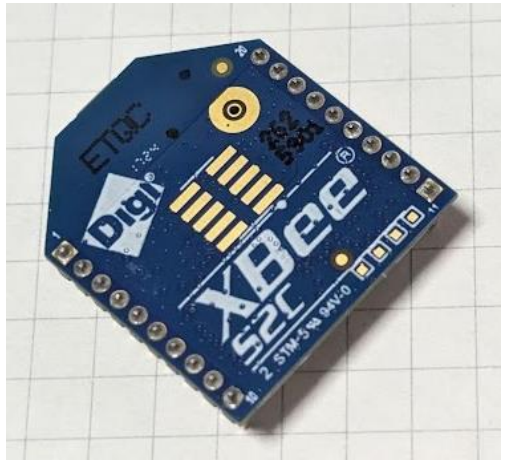
オマケ：デバッグ対象が移動体の場合

- 移動体である為，**有線接続が厳しい** といった場面も

- こんな時はシリアル通信の透過モードのある無線デバイスを使うのも1つの手.
  - XBee, TWE-LITE

XBeeについて

- DIGI社が開発した無線通信機能とマイコンを搭載した小型のモジュールです。



モデル	電波出力	屋内飛距離	屋外飛距離
S2C(通常)	3.1mW	---	---
S2C(ブースト時)	6.3mW	60m	1200m

オマケ

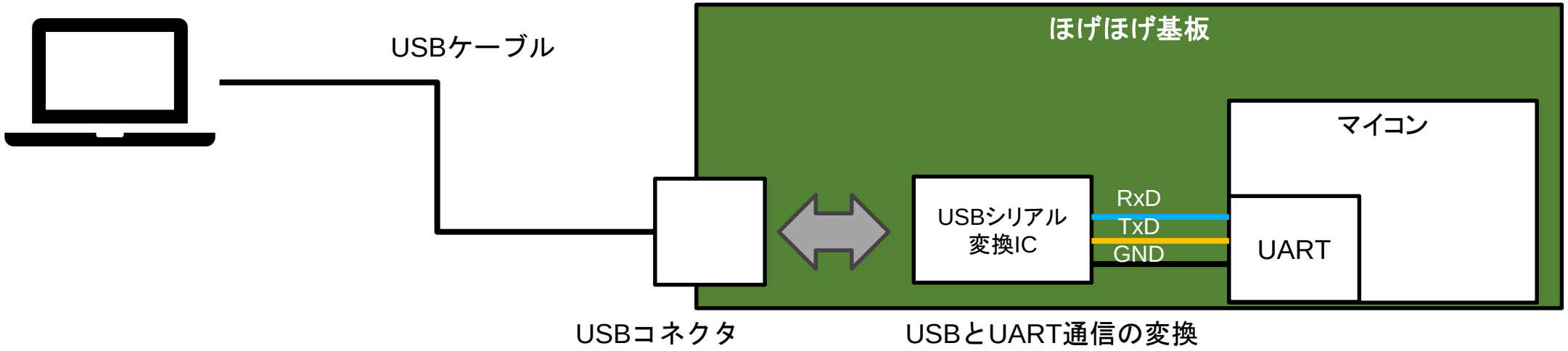
透過モード

- 今回，XBeeを使用する理由
- シリアル通信を送信機が他のプロトコルに変換し，受信機がまたシリアル通信に戻す通信のこと
  - マイコン視点からすると，特にプログラムを変更することなくシリアル通信を他の通信プロトコルに変換できるメリットがある
- XBeeに限らず，TWE-LITEやBluetoothなどにも透過モードを持つ製品あり

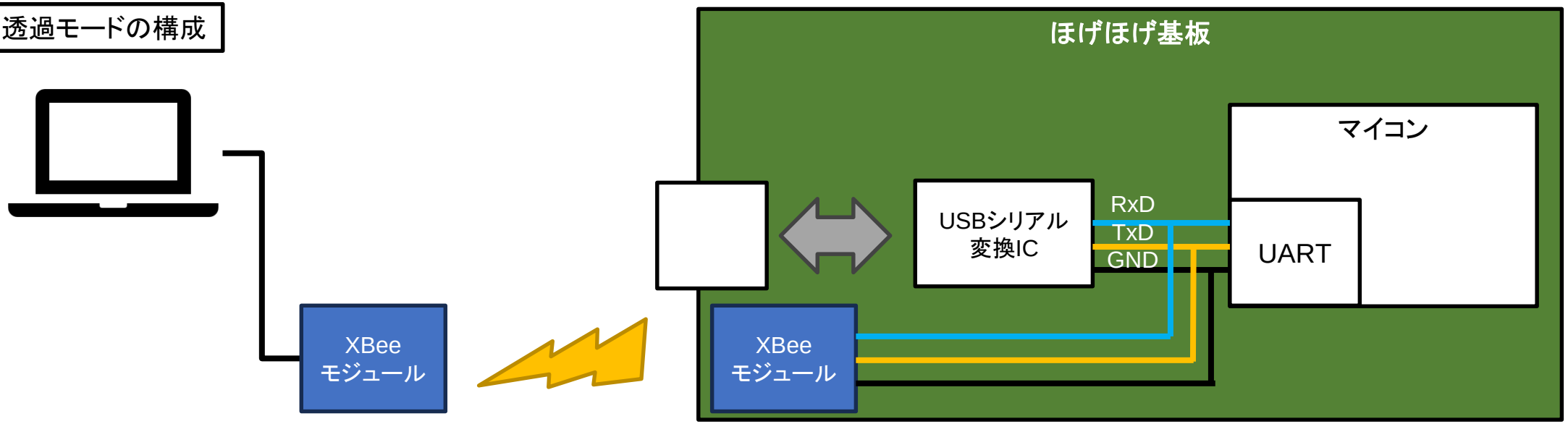
回路イメージ

オマケ

① Arduino UNOなどの構成

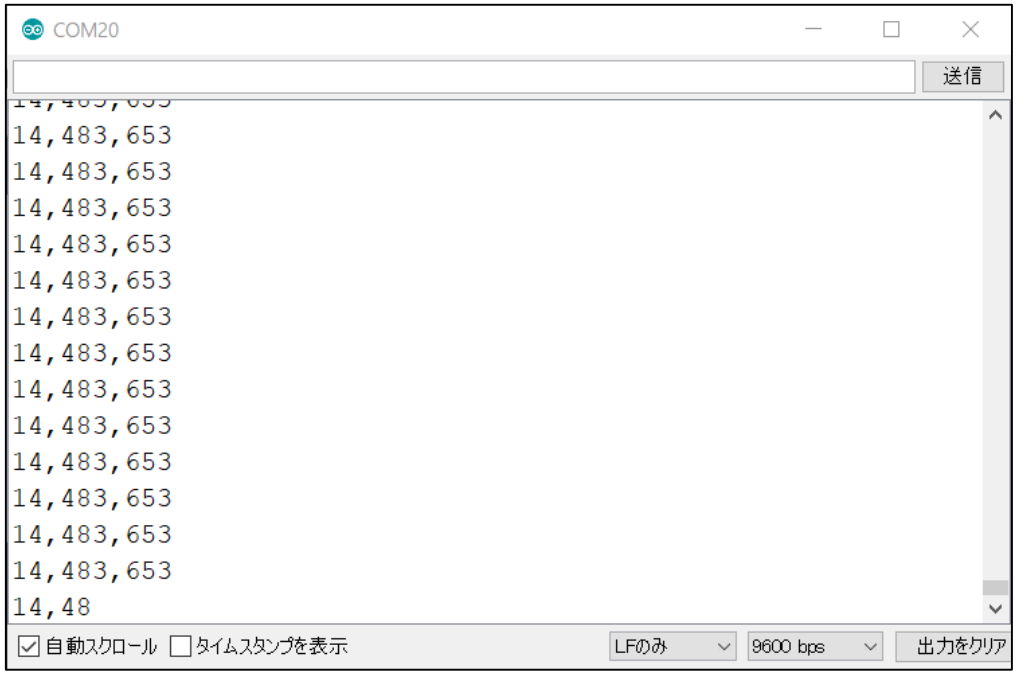


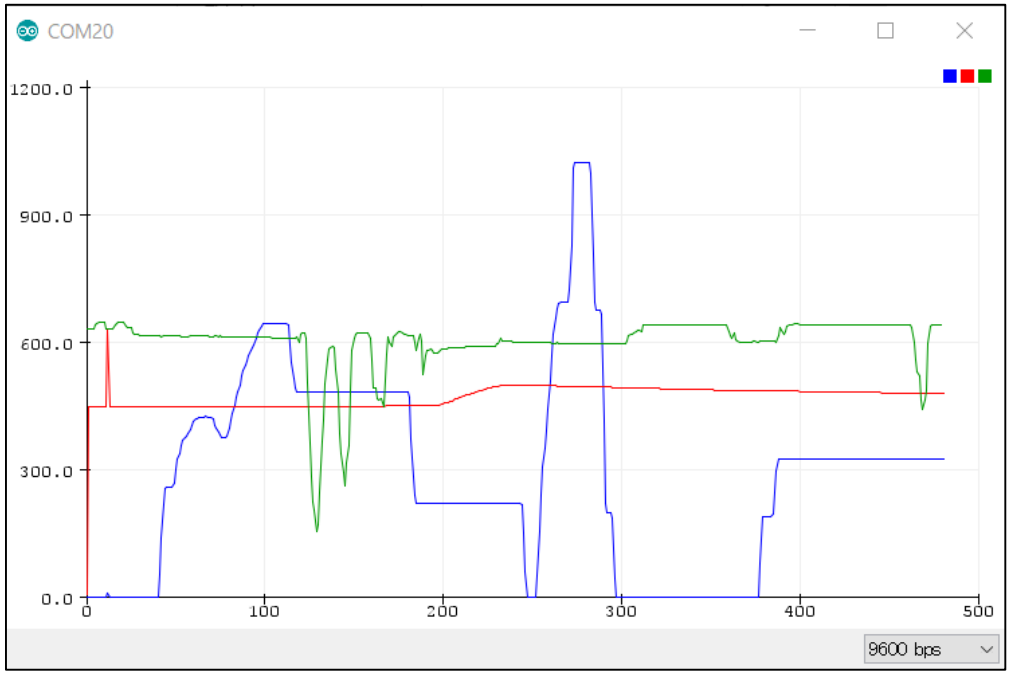
⑤ 透過モードの構成



出力する書式にも気を配ろう！

- CSV形式で吐き出すと…
  - ログを保存して，あとでExcelで加工しながら解析 → 一通りの挙動のログを取ってからじっくりとデバッグ
  - リアルタイムにグラフ描画できる → リアルタイムに挙動を見ながらデバッグ
    - 例) Arduino IDEのシリアルプロッタ機能

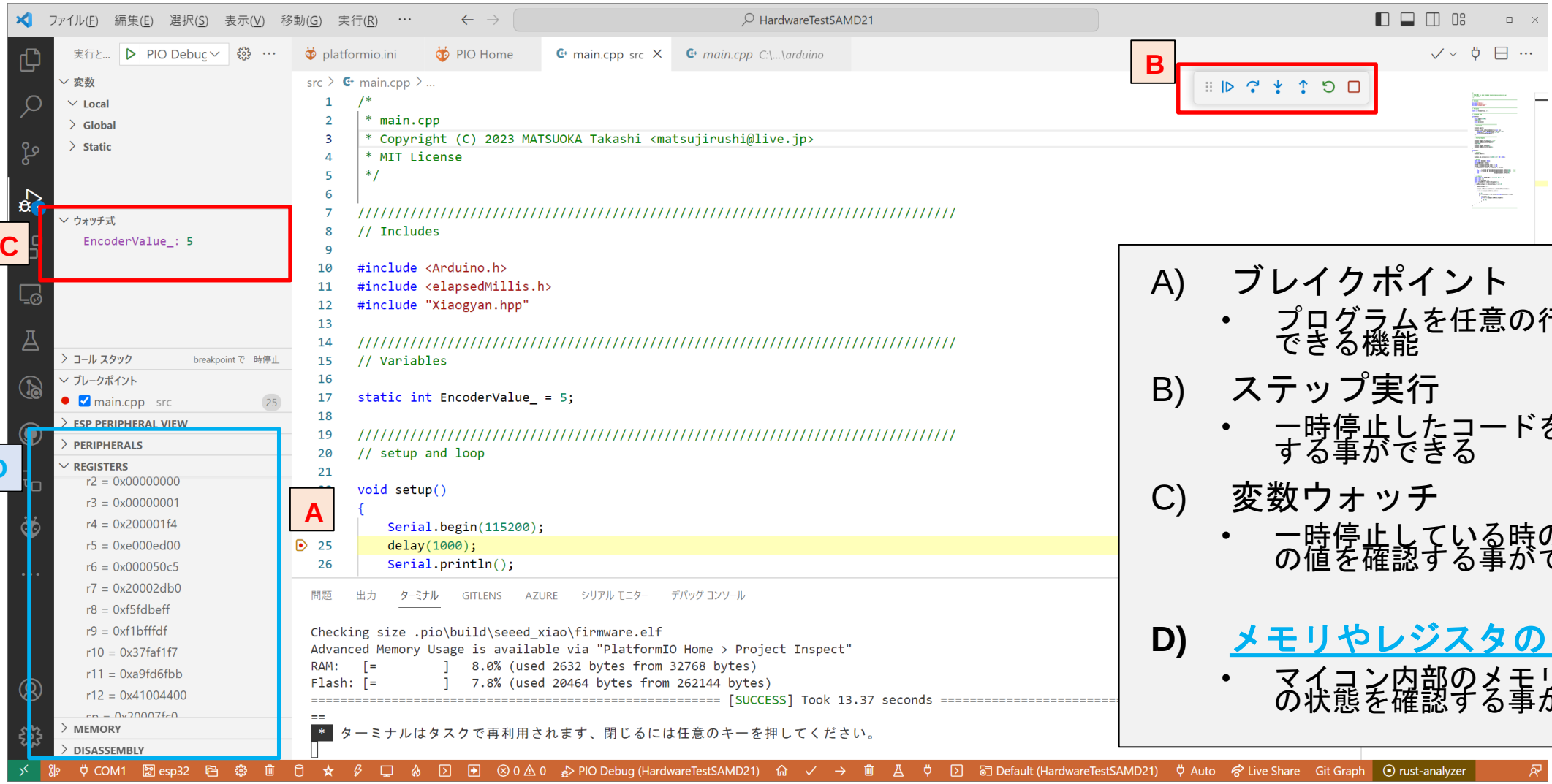




② オンチップ・エミュレータを活用したデバッグ

オンチップ・エミュレータを用いたデバッグでできること

細かいロジックを追いたい時は
ブレイクポイントを貼って
デバッグした方が効率的



A) ブレイクポイント
- プログラムを任意の行で一時停止できる機能

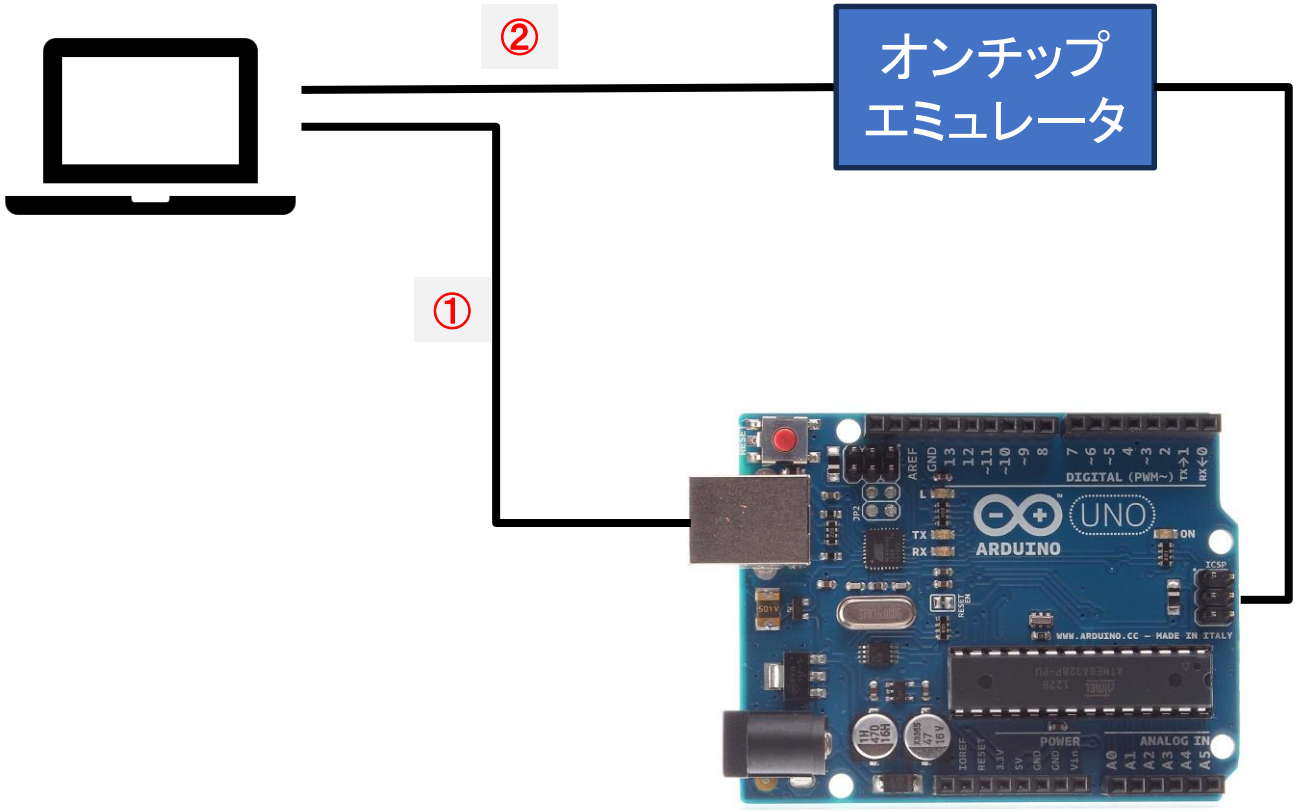
B) ステップ実行
- 一時停止したコードを1行ずつ実行する事ができる

C) 変数ウォッチ
- 一時停止している時の任意の変数の値を確認する事ができる

D) メモリやレジスタのウォッチ
- マイコン内部のメモリやレジスタの状態を確認する事ができる

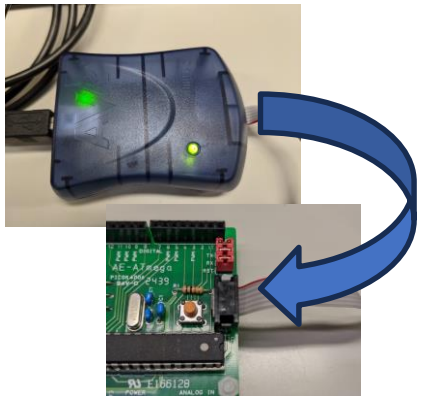
参考： JTAGデバッガを用いたVisual Stduio + Seeeduino XIAO(無印)のデバッグ画面

マイコンのプログラムのデバッグ方法



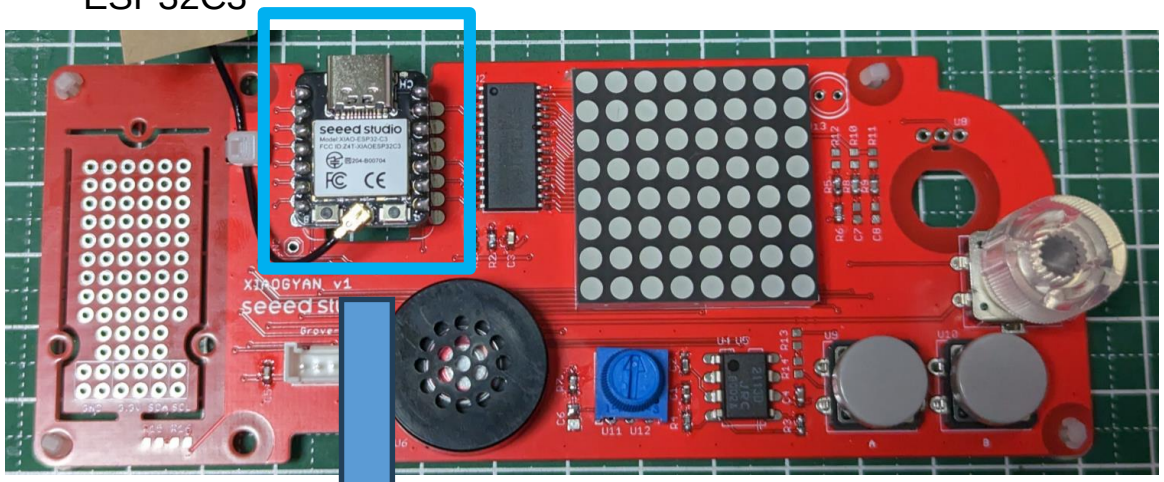
oomlout - ARDU-03_02, CC 表示-継承 2.0,
https://commons.wikimedia.org/w/index.php?curid=28380408による

1. 通信を用いたPrintデバッグ
2. オンチップエミュレータによるデバッグ
   - ブレイクポイントを用いたデバッグが出来るようになる
   - ただし，専用のデバッガが必須
     - Arduinoの場合はAVR ISPなど

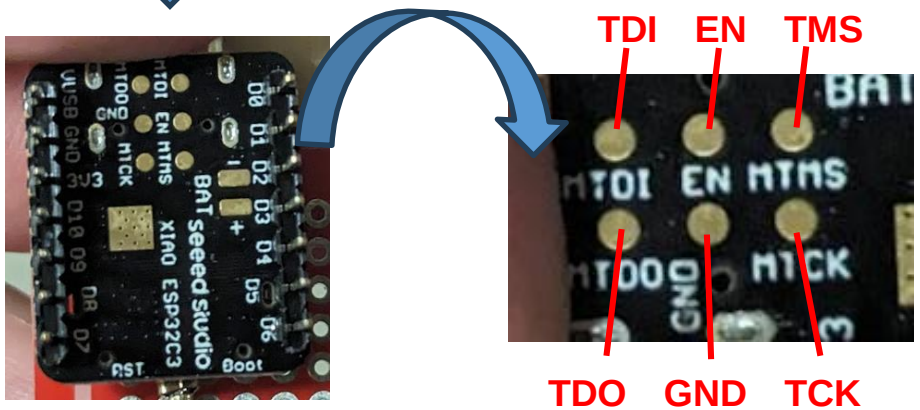


Seeeduino XIAO ESP32C3に オンチップ・エミュレータを接続する

Seeeduino XIAO
ESP32C3



裏返すと…



JTAG
デバッガ

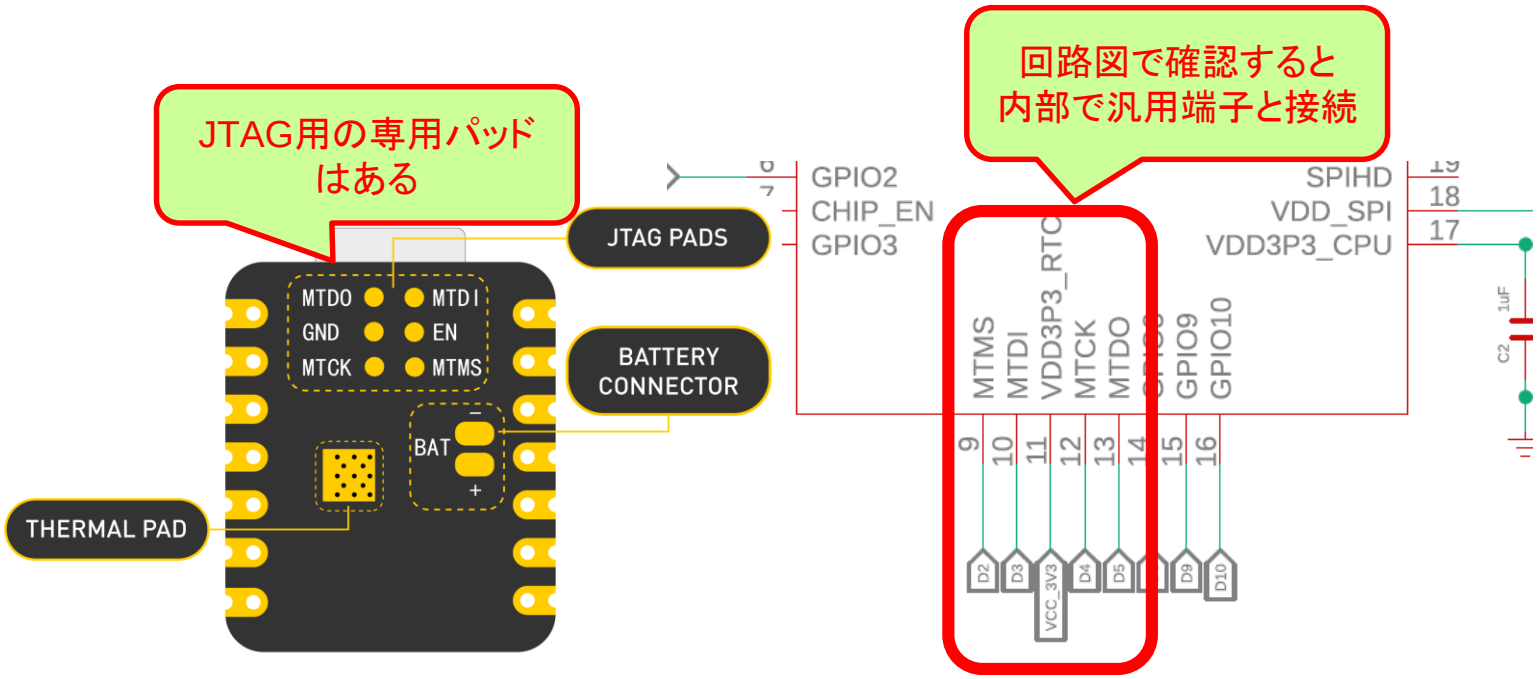
JTAG
インタフェース

- XIAOGYANボードに搭載されるマイコンボード
  - Seeeduino XIAO ESP32C3ボード
- ボード裏のJTAGインタフェース用のPADからJTAGデバッガに繋ぐ

これでデバッグ！の予定だんですが…

これでデバッグ！と思ったんですが…

- 残念ながらXIAO ESP32C3は汎用端子とデバッグ端子が重複していて、利用不可でした。

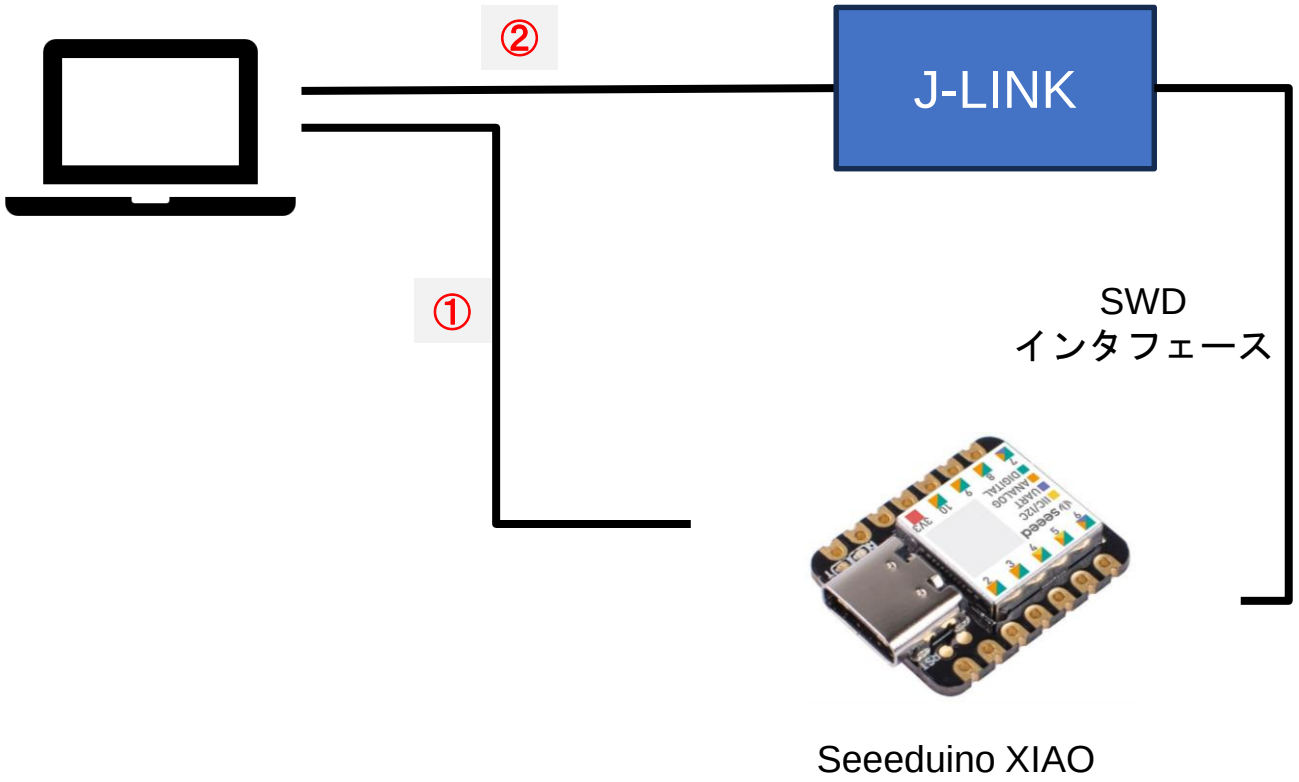


XIAOGYANボードで使えるマイコンは？

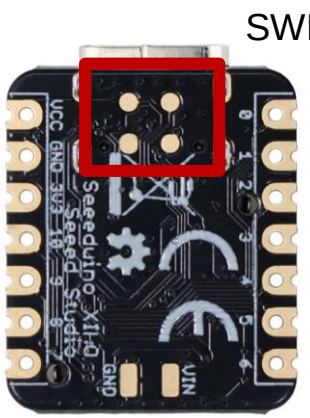
ボード		デバッガ ピンアサイン	PlatformIO	XIAOGYAN ライブラリ	メモ
XIAO	無印(SAMD21)	○(SWD)	○	△	
	RP2040	○(SWD)	△	△	
	ESP32C3	※（JTAG）	○	◎	※汎用端子とデバッグ端子が重複
	ESP32S3	○（JTAG）	○	○	
	nRF52	○（SWD）	△	△	

- 無印XIAO(SWD) or XIAO ESP32S3(JTAG)で、対応できそう！

今回の構成



- オンチップ・エミュレータは，SEGGER Microcontroller社のJ-LINKを使用
- Seeeduino XIAO(無印)の裏面のSWDインタフェースよりJ-LINKに接続



画像引用元： https://jp.seeedstudio.com/Seeeduino-XIAO-Arduino-Microcontroller-SAMD21-Cortex-M0+-p-4426.html

オンチップ・エミュレータの選定

- 開発ツール、デバッグツールは、信頼性第１！
  ⇒不確かなツールを使うと、何が正しいのか不明になる
  - マイコンメーカー推奨機器の採用
  - 業界標準（デファクト）の利用　etc.

- ARM系デバッガ
  - CMSIS-DAP
  - J-LINK、ST-LINK　etc.

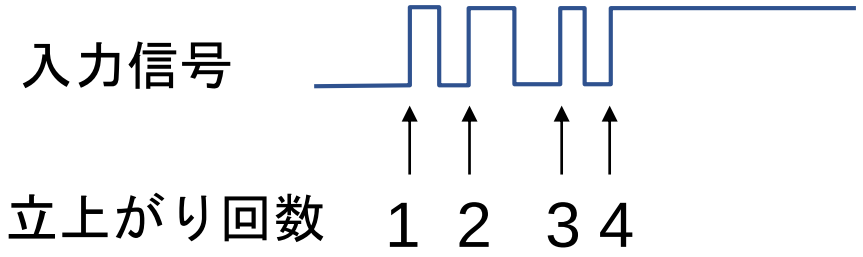
③ オシロスコープを活用したデバッグ

オシロスコープを活用したデバッグでできること

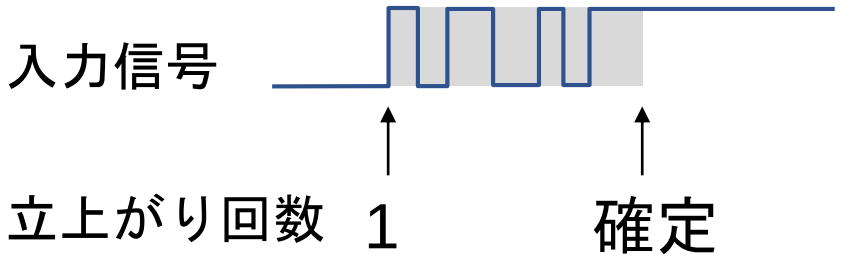
- 入力機器からの信号を確認・検証
- マイコンの動作をIOピンから確認

入力機器からの信号を確認・検証
～スイッチのチャタリング問題～

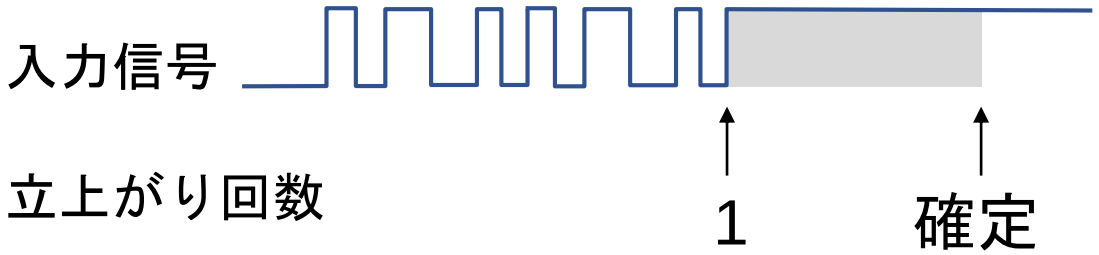
※ チャタリング ： スイッチを切り替えるタイミングで、機械接点の影響により
ON/OFF が繰り返される現象



- 対策1：一定時間カウントを止める



- 対策2：一定時間、入力が安定したら、変化を確定する
これだと、チャタリングが長引いても問題ない

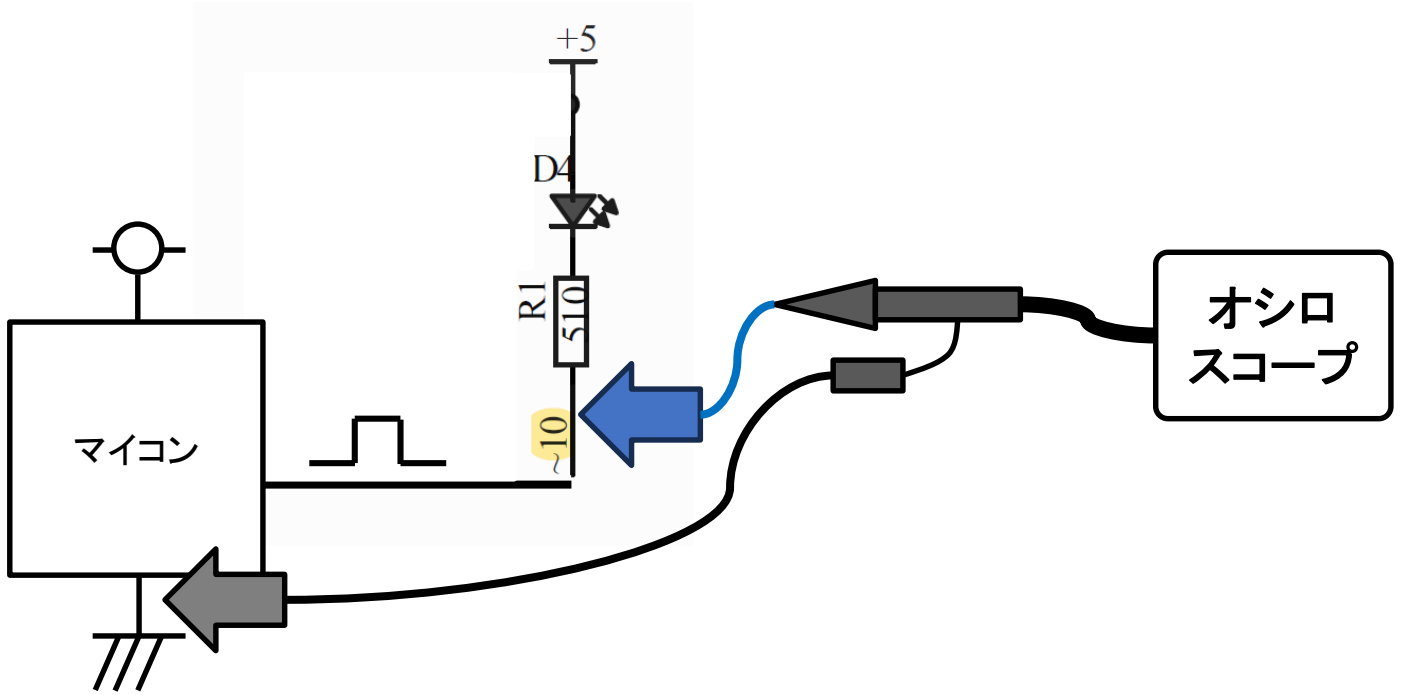


一定時間って？どれぐらい？
→ オシロスコープで実測しよう

マイコンの動作をIOピンから確認
～IO制御+オシロで時間計～

- 以下の処理を記述して，LEDのON/OFFの時間をオシロスコープで計測すると，Printにかかる処理時間が見えてくる
  - 実測ベースではあるが，処理時間を把握する事が可能になる

① LEDをONする
② Printする
③ LEDをOFFする



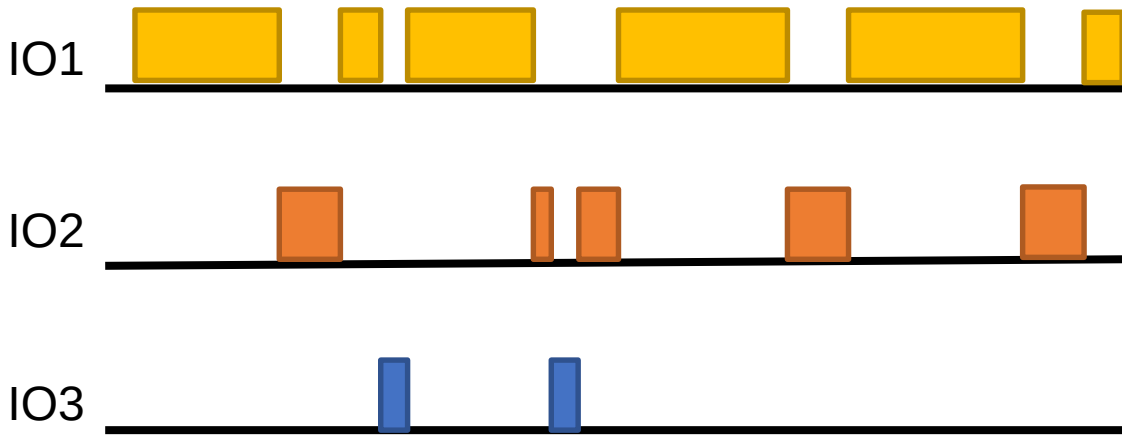
マイコンの動作をIOピンから確認
～タイマなどの割り込みの動作状況の可視化～

- 空きIOから，main関数や割り込み関数が動いている間，HIGH パルスを出力すれば，各関数の稼働状況やディスパッチ順番が可視化できる
  - ただし，タスクの数だけチャンネル数が要る
  - 使い方としては，ロジックアナライザ向き

```
int main関数() {
    IO1をHIGHにする
        :
        :
        :
        :
    IO1をLOWにする
    delay(100);
    return 0;
}
```

```
int タイマ割込み関数() {
    IO2をHIGHにする
        :
        :
        :
        :
    IO2をLOWにする
}
```

```
int 外部入力割込み関数() {
    IO3をHIGHにする
        :
        :
        :
        :
    IO3をLOWにする
}
```



- 特定のタスクの処理時間が極端に短くなっていないか？
- 特定の割り込み関数が，そもそも起動しているか？
- 特定の割り込み関数は，意図した回数だけ起動しているか？　　などなど

オマケ

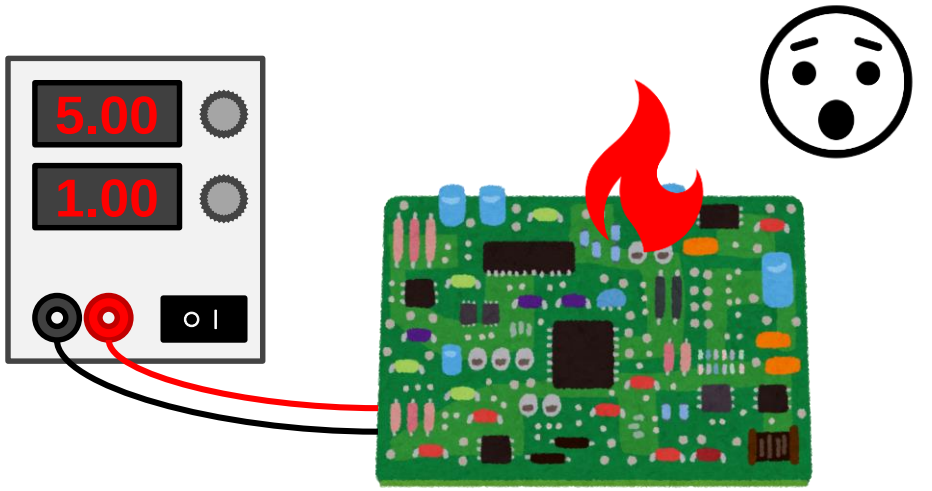
最近は，プロトコルアナライザ付きのオシロスコープも

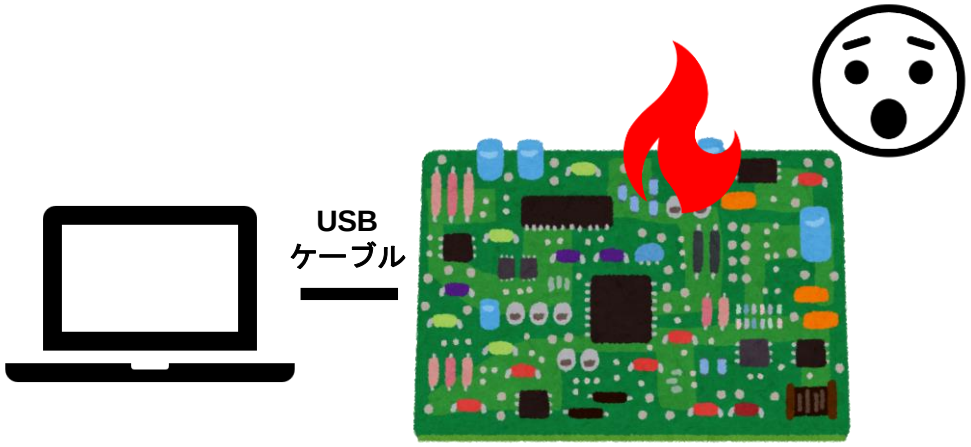
- UARTやI2C, SPIなどの波形をオシロで見ると…
- デコード&画面表示する機能(プロトコルアナライザ)が搭載されたオシロも存在
  - どちらかと言うとロジックアナライザが得意とする機能？

UARTを計測した様子

④ 火入れの前の
ショートチェック

ハードウェアが完成したら
即・電源投入してませんか？





- その回路，本当に正しくできてますか？
  - 部品の極性を間違えていた
  - はんだがブリッジしていた
  - 配線がショートしていた　　などなど

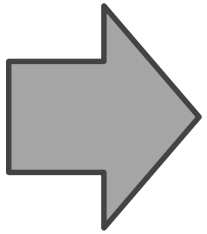
「組込み百物語」の貴重な一話が爆誕！

- 締切・納期は明日…
  - この回路，今からもう一回作るの…？
- 燃えたこの部品，めっちゃ高い…
- PCのUSBポートが死にました…

最低限の電源系のショートチェック！

- 回路図や実体配線図のチェックは勿論ですが…
- 電源系のショートチェックをしましょう！

- ショートチェックとは？
  - A～B間の抵抗値を測定し，ショートしていない事を確認すること
- 電源系とは？
  - Vcc～GND間(例：5v～GND間)
  - Vdd～Vcc間（例：3.3v～5v間)



なぜ電源系なのか？

電流が大量に流れれば…
- 基板の配線パターンの焼損
- 電子部品の焼損

に繋がるから

演習編

演習編で行うこと

XIAOGYANチュートリアル

島ごとに分かれての演習

1. Printデバッグ
2. オンチップ・エミュレータを活用したデバッグ体験
3. オシロスコープを活用したデバッグ
4. 火入れの前のショートチェック

早く終わった人向け

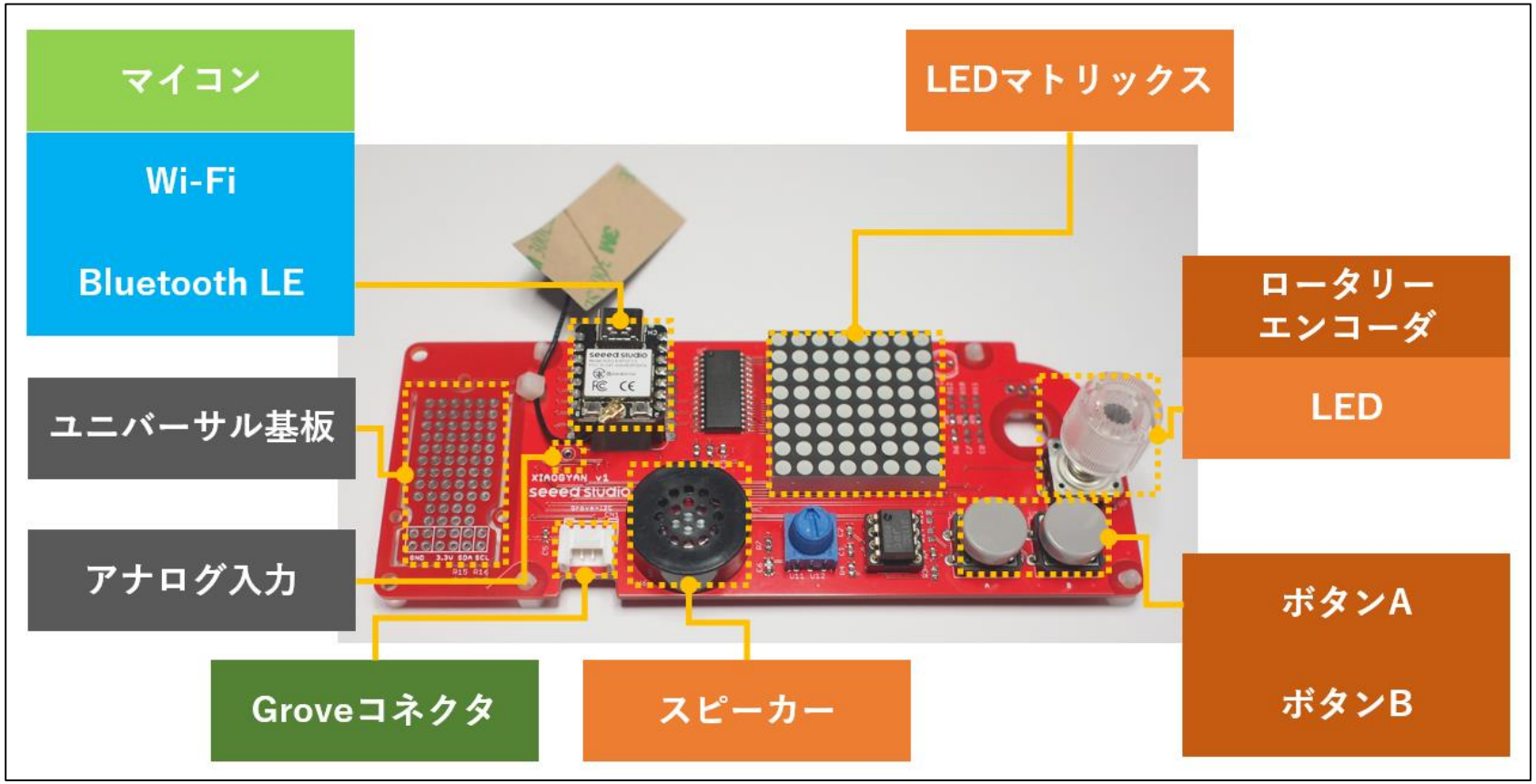
発展課題

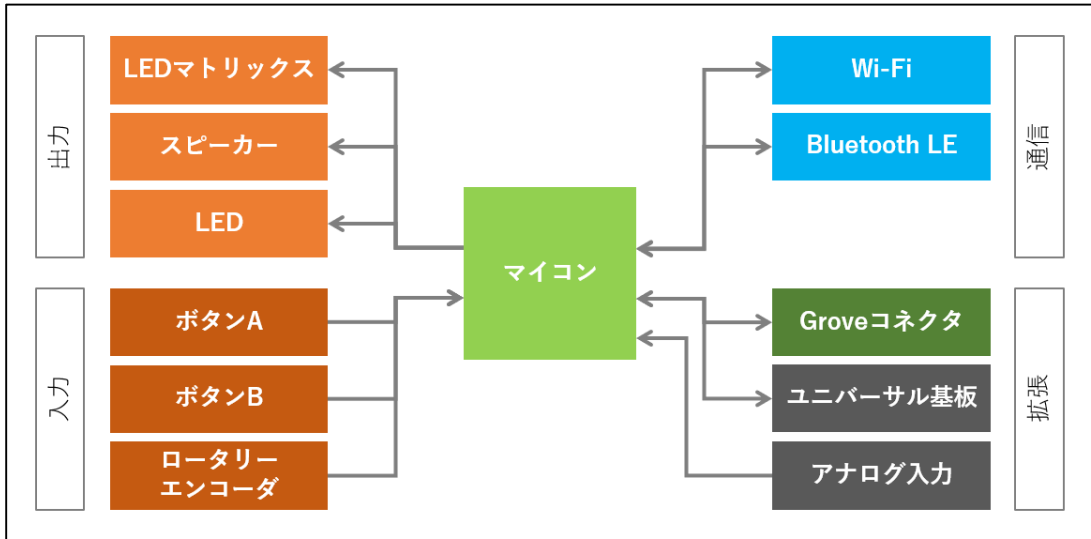
0. XIAOGYANチュートリアル

10分程度

XIAOGYANについて

- IoT ALGYANの8周年イベントに合わせて開発されたボードです
  - https://github.com/algyan/XIAOGYAN
- マイコンはSeeeduino XIAO ESP32C3
- 周辺機器としてロータリエンコーダやLEDマトリックスなどを完備





画像引用元：https://github.com/algyan/XIAOGYAN/blob/main/manuals/guidebook/README.md

XIAOGYANを動かしてみよう！

- まずは，HardwareTest を実際にボードで動かしてみましょう
  - https://github.com/algyan/XIAOGYAN/blob/main/manuals/software/README.md

1. Printデバッグ(演習編)

20分程度

問1-1：問題内容

■要件

ボードのボタンを押下するとそれぞれ対応したビープ音が鳴ること

- Aボタン押下：低音
- Bボタン押下：中音
- A&Bボタン押下：高音

■不具合

A&Bボタン押下時に低音が鳴る

Printデバッグを使って原因を突き止めてみよう

問1-1：ヒント

```
// Buttons
static bool buttonA = false;
static bool buttonB = false;
bool preButtonA = buttonA;
bool preButtonB = buttonB;
buttonA = Xiaogyan.buttonA.read() == LOW;
buttonB = Xiaogyan.buttonB.read() == LOW;
if (preButtonA != buttonA || preButtonB != buttonB)
{
    if( buttonA ) {
        Xiaogyan.speaker.setTone(262);  // C4
    } else if( buttonB ) {
        Xiaogyan.speaker.setTone(294);  // D4
    } else if(buttonA &&  buttonB ) {
        Xiaogyan.speaker.setTone(330);  // E4
    } else {
        Xiaogyan.speaker.setTone(0);    // OFF
    }
 }
```

ボタン押下時の処理は左記のコードで実施しています

それぞれのボタンを押した場合にどのようなパスを通るか、Printを挿入して調べてみよう

問1-1：答え

```
// Buttons
static bool buttonA = false;
static bool buttonB = false;
bool preButtonA = buttonA;
bool preButtonB = buttonB;
buttonA = Xiaogyan.buttonA.read() == LOW;
buttonB = Xiaogyan.buttonB.read() == LOW;
if (preButtonA != buttonA || preButtonB != buttonB)
{
    if( buttonA ) {
        Xiaogyan.speaker.setTone(262);  // C4
    } else if( buttonB ) {
        Xiaogyan.speaker.setTone(294);  // D4
    } else if(buttonA &&  buttonB ) {
        Xiaogyan.speaker.setTone(330);  // E4
    } else {
        Xiaogyan.speaker.setTone(0);    // OFF
    }
}
```

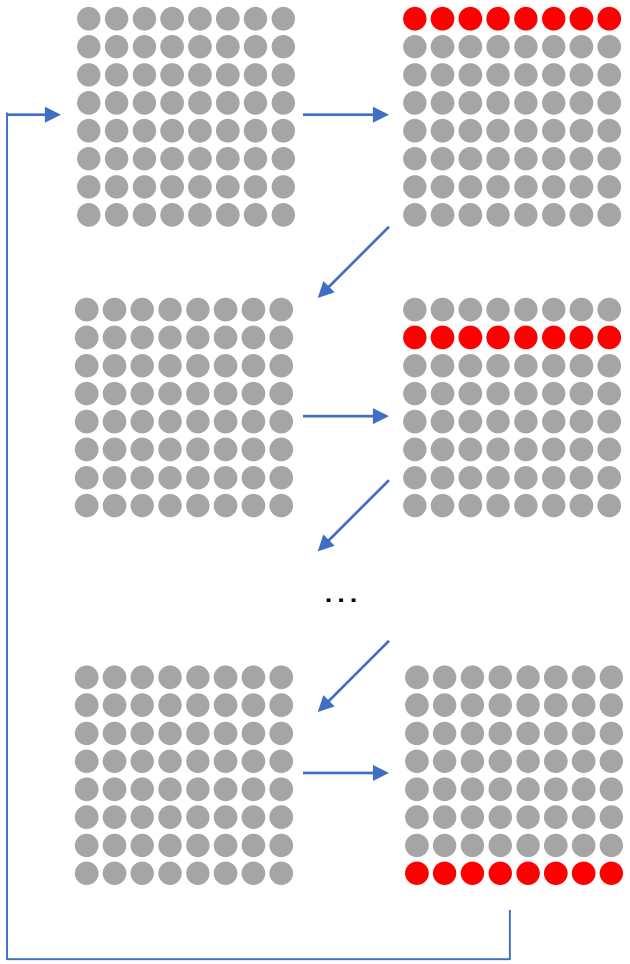
修正

```
// Buttons
static bool buttonA = false;
static bool buttonB = false;
bool preButtonA = buttonA;
bool preButtonB = buttonB;
buttonA = Xiaogyan.buttonA.read() == LOW;
buttonB = Xiaogyan.buttonB.read() == LOW;
if (preButtonA != buttonA || preButtonB != buttonB)
{
    if( buttonA && !buttonB ) {
        Xiaogyan.speaker.setTone(262);  // C4
    } else if( !buttonA && buttonB ) {
        Xiaogyan.speaker.setTone(294);  // D4
    } else if( buttonA && buttonB ) {
        Xiaogyan.speaker.setTone(330);  // E4
    } else {
        Xiaogyan.speaker.setTone(0);    // OFF
    }
}
```

A,Bの両方が押されている場合でも、「Aが押されている」という判定式がTrueになってしまうため、期待した結果にならない

AとBの両方の状態を元に判定することで、Aだけ押されている、Bだけ押されている、AとBの両方が押されている状態を正しく判定できるようにしている

問1-2：問題内容



■要件

LEDを以下のように点灯/消灯させること

- 左から順番に1行目を点灯させる
- 左から順番に1行目を消灯させる
- 1行目を消灯させたら次の行で同様の処理を実施する
- 8行目まで実施したら、すぐに1行目から同じ処理を繰り返す

■問題

8行目の処理が完了した後、1行目の処理が実施されるまでに間があいてしまう

Printデバッグを使って原因を突き止めてみよう

問1-2：ヒント

```
if (ledMatrixElapsed >= (EncoderValue_ + 1) * 10)
{
    ledMatrixElapsed = 0;

    Xiaogyan.ledMatrix.drawPixel(x, y, COLOR_MAP[colorIndex]);

    if (x++ >= Xiaogyan.ledMatrix.width())
    {
        x = 0;
        if (++colorIndex >= std::extent<decltype(COLOR_MAP)>::value)
        {
            colorIndex = 0;
            if (y++ >= Xiaogyan.ledMatrix.height())
            {
                y = 0;
            }
        }
    }
}
```

LEDの点灯/消灯は左記のコードで実施しています

行を示すｙの値がどのように変化しているか、Printを挿入して調べてみよう

＃y=0の時に1行目を指しているとすると、
yはどの範囲でループしてほしい？

問1-2：答え

```
if (ledMatrixElapsed >= (EncoderValue_ + 1) * 10)
{
    ledMatrixElapsed = 0;

    Xiaogyan.ledMatrix.drawPixel(x, y, COLOR_MAP[colorIndex]);

    if (x++ >= Xiaogyan.ledMatrix.width())
    {
        x = 0;
        if (++colorIndex >= std::extent<decltype(COLOR_MAP)>::value)
        {
            colorIndex = 0;
            if (y++ >= Xiaogyan.ledMatrix.height())
            {
                y = 0;
            }
        }
    }
}
```

右辺の値(行数)は8

yは、値を更新した次のループでLEDに反映されます。
yを0に戻す判定は++がyの後についているため、判定後にyの値がインクリメントされます。
そのため、y=7で判定した場合、次のループではy=8となってしまい、LEDの表示対象範囲の外になってしまっている

修正

```
if (ledMatrixElapsed >= (EncoderValue_ + 1) * 10)
{
    ledMatrixElapsed = 0;

    Xiaogyan.ledMatrix.drawPixel(x, y, COLOR_MAP[colorIndex]);

    if (++x >= Xiaogyan.ledMatrix.width())
    {
        x = 0;
        if (++colorIndex >= std::extent<decltype(COLOR_MAP)>::value)
        {
            colorIndex = 0;
            if (++y >= Xiaogyan.ledMatrix.height())
            {
                y = 0;
            }
        }
    }
}
```

++を前につけることで、判定前にインクリメントが実施されるようになる。
これにより、y=8になった直後に判定に引っかかるようになり、
期待したタイミングでy=0に戻る

おまけ

```
if (ledMatrixElapsed >= (EncoderValue_ + 1) * 10)
{
    ledMatrixElapsed = 0;

    Xiaogyan.ledMatrix.drawPixel(x, y, COLOR_MAP[colorIndex]);

    if (x++ >= Xiaogyan.ledMatrix.width())
    {
        x = 0;
        if (++colorIndex >= std::extent<decltype(COLOR_MAP)>::value)
        {
            colorIndex = 0;
            if (y++ >= Xiaogyan.ledMatrix.height())
            {
                y = 0;
            }
        }
    }
}
```

問1,2が終わった方は考えてみてください

実はｘも8までカウントされてしまっています

しかし、ｙと異なり目視では不具合があることがわかりにくいと思います

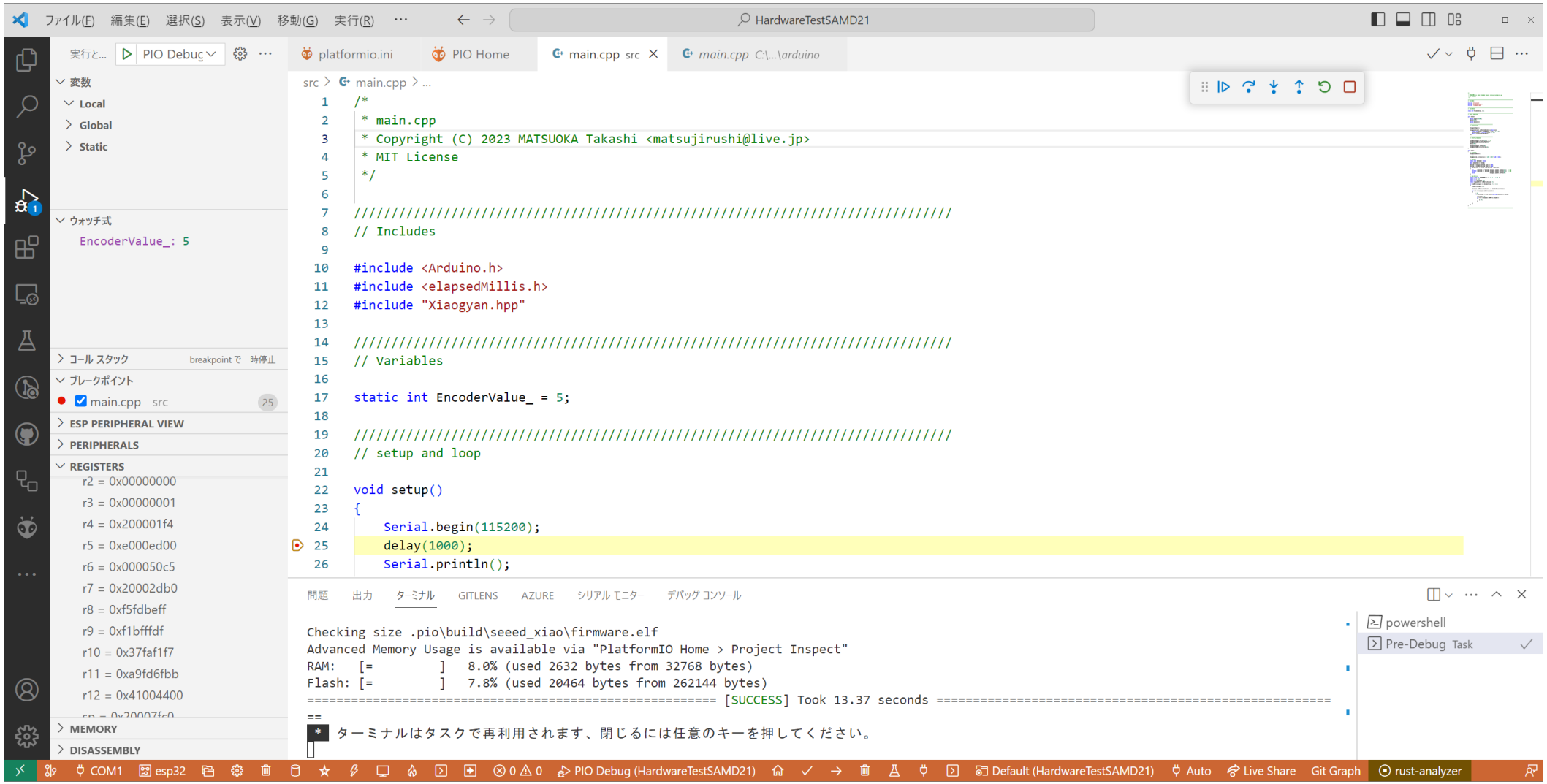
このような不具合を見つける場合、どのようなデバッグを実施すると良いか考えてみてください

2. オンチップ・エミュレータを活用したデバッグ(演習編)

10分程度

体験

デバッガ動作中の画面


ファイル(F) 編集(E) 選択(S) 表示(V) 移動(G) 実行(R)
HardwareTestSAMD21
PIO Debug
platformio.ini
PIO Home
main.cpp src
main.cpp C:\...\arduino
変数
Local
Global
Static
ウォッチ式
EncoderValue_: 5
コール スタック
breakpoint で一時停止
ブレークポイント
main.cpp src 25
ESP PERIPHERAL VIEW
PERIPHERALS
REGISTERS
r2 = 0x00000000
r3 = 0x00000001
r4 = 0x200001f4
r5 = 0xe000ed00
r6 = 0x000050c5
r7 = 0x20002db0
r8 = 0xf5fdbeff
r9 = 0xf1bfffdf
r10 = 0x37faf1f7
r11 = 0xa9fd6fbb
r12 = 0x41004400
MEMORY
DISASSEMBLY
src > main.cpp > ...
1 /*
2 * main.cpp
3 * Copyright (C) 2023 MATSUOKA Takashi <matsujirushi@live.jp>
4 * MIT License
5 */
6
7 //
8 // Includes
9
10 #include <Arduino.h>
11 #include <elapsedMillis.h>
12 #include "Xiaogyan.hpp"
13
14 //
15 // Variables
16
17 static int EncoderValue_ = 5;
18
19 //
20 // setup and loop
21
22 void setup()
23 {
24 Serial.begin(115200);
25 delay(1000);
26 Serial.println();
問題 出力 ターミナル GITLENS AZURE シリアル モニター デバッグ コンソール
Checking size .pio\build\seeed_xiao\firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM: [=] 8.0% (used 2632 bytes from 32768 bytes)
Flash: [=] 7.8% (used 20464 bytes from 262144 bytes)
========== [SUCCESS] Took 13.37 seconds ==========
==
* ターミナルはタスクで再利用されます、閉じるには任意のキーを押してください。
powershell
Pre-Debug Task
COM1 esp32 0 0 PIO Debug (HardwareTestSAMD21) Default (HardwareTestSAMD21) Auto Live Share Git Graph rust-analyzer

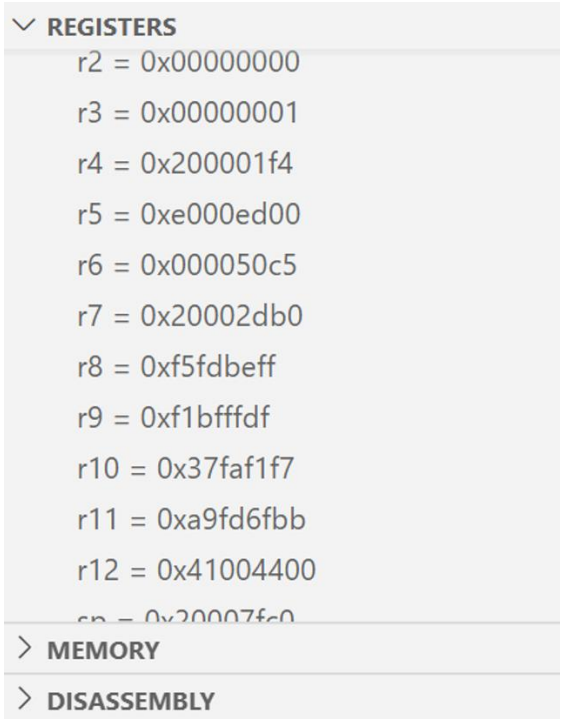
デバッガの操作（基本）

-  : 実行（ブレークポイントまで）
-  : ステップ実行
-  : ステップイン実行
-  : ステップアウト実行
-  : リセット
-  : 停止

デバッガの操作

- 変数
  - スコープに応じた関数内などの変数の値を表示
- ウオッチ式
  - 変数や評価式の値を表示
- コールスタック
  - 関数の呼び出しネストを表示
- ブレークポイント
  - 設定済みのブレークポイント一覧を表示

デバッガの操作



- REGISTERS
  - 各レジスタの値を表示
- MEMORY
  - メモリの値を表示
- DISASSEMBLY
  - ソース⇔アセンブラ命令表示の切り替え

3. オシロスコープを活用したデバッグ(演習編)

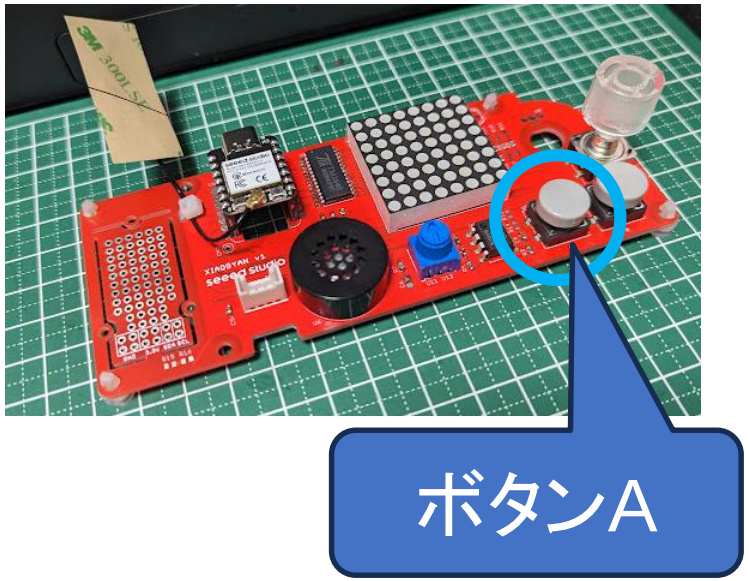
20分程度

例3-1：ソフトウェアで経過時間をカウントする

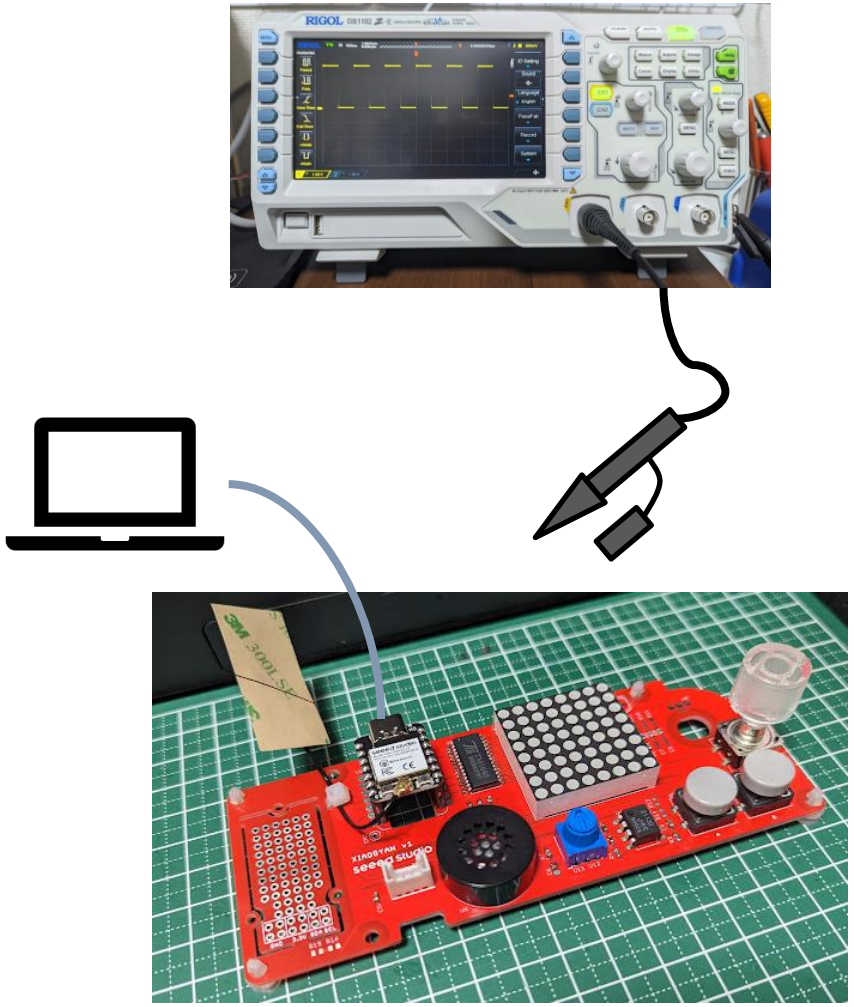
```
if (preButtonA != buttonA || preButtonB != buttonB)
{
    if( buttonA && !buttonB) {
        Xiaogyan.speaker.setTone(262);    // C4

        // [Printデバッグ]② ソフトウェアで経過時間をカウントする
        // -> 経過時間を計測して，その差分から②(Printする)にかかった処理時間を計測する
        unsigned long time1, time2, time_diff;
        ①time1 = micros();
        ②Serial.println("ButtonA Pushed!!");
        ③time2 = micros();
        ④time_diff = time2 - time1;
        ⑤Serial.printf("%ld[us]\n", time_diff);

    } else if (!buttonA &&  buttonB) {
        Xiaogyan.speaker.setTone(294);    // D4
    } else if ( buttonA &&  buttonB) {
        Xiaogyan.speaker.setTone(330);    // E4
    } else {
        Xiaogyan.speaker.setTone(0);
    }
}
```

① time1 = micros();
② Printする
③ time2 = micros();
④ time_diff = time2 -time1;　　// 経過時間
⑤ time_diffをPrintする

- OscilloDebugを実行ください
- ボタンAを押下すると，シリアル通信で”ButtonA Pushed!!”と出力されます
- このシリアル通信の処理時間を続けて出力します．



例3-2：Printの処理時間をオシロで計測



さきほど，紹介した「IO制御＋オシロで計測」を実際にやってみましょう！

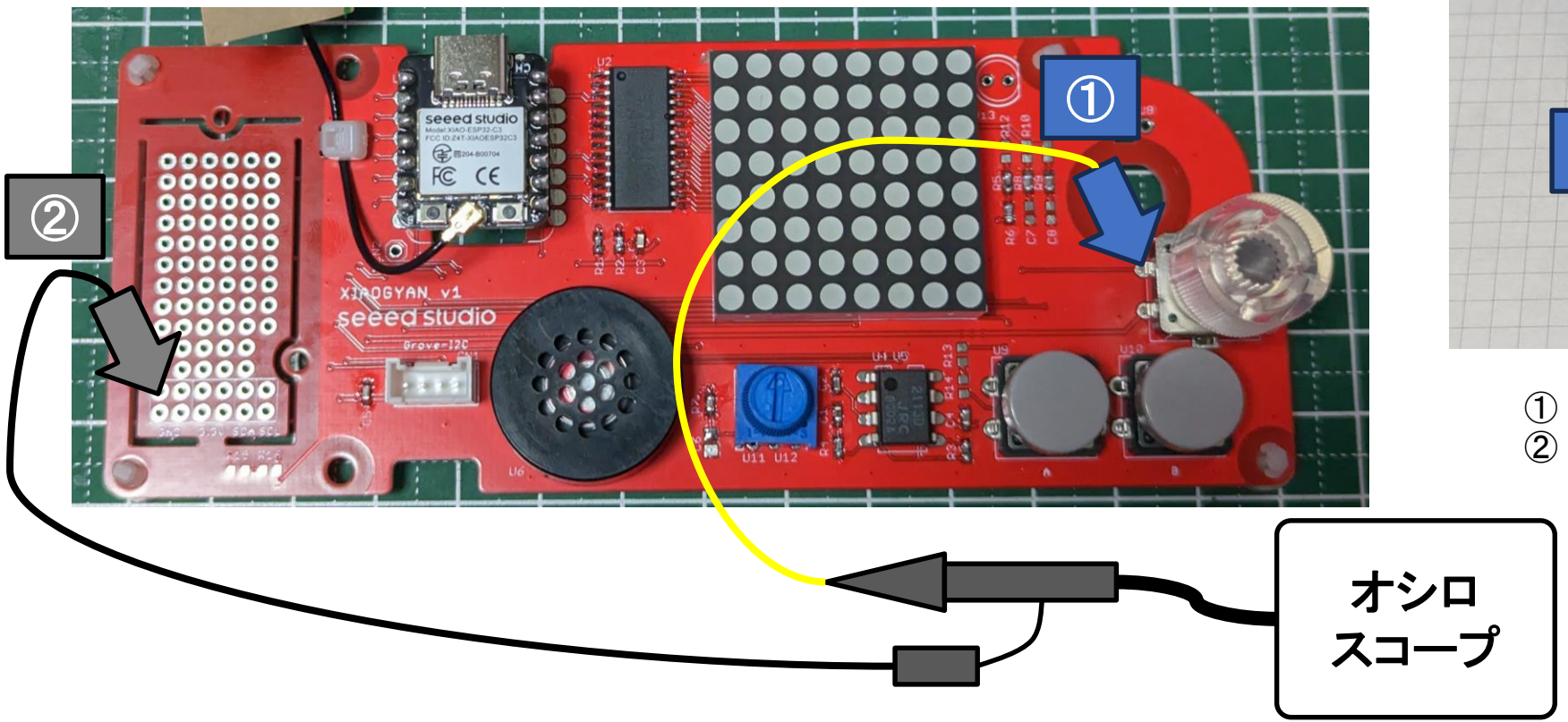
- 以下の処理を記述して，LEDのON/OFFの時間をオシロスコープで計測すると，Printにかかる処理時間が見えてくる
  - 実測ベースではあるが，処理時間を把握する事が可能になる

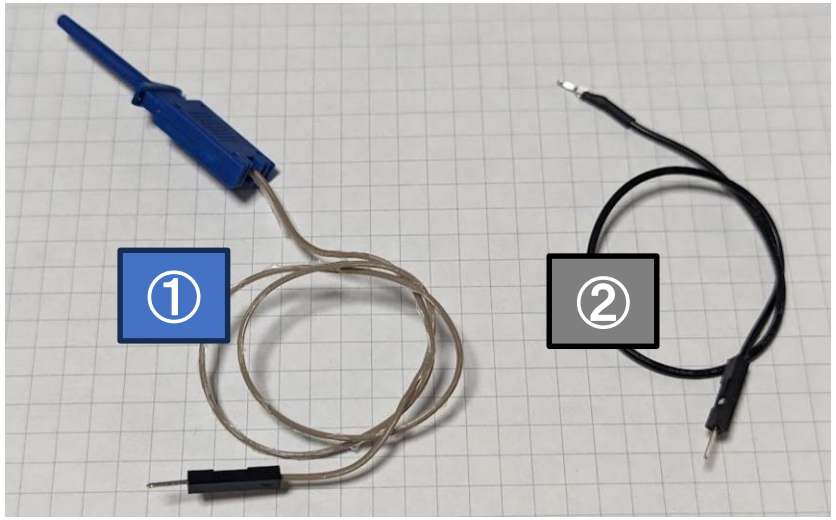
① LEDをONする
② Printする
③ LEDをOFFする

ハードウェアの繋ぎ方

- IO(D6:LEDピン)を上げ下げして，ソフトウェアの処理時間を計測してみましょう！

【計測ポイント】





①：テストクリップ(秋葉原のジャンク品)
②：スルーホール用テストワイヤ(TP-200)

演習用のプログラム

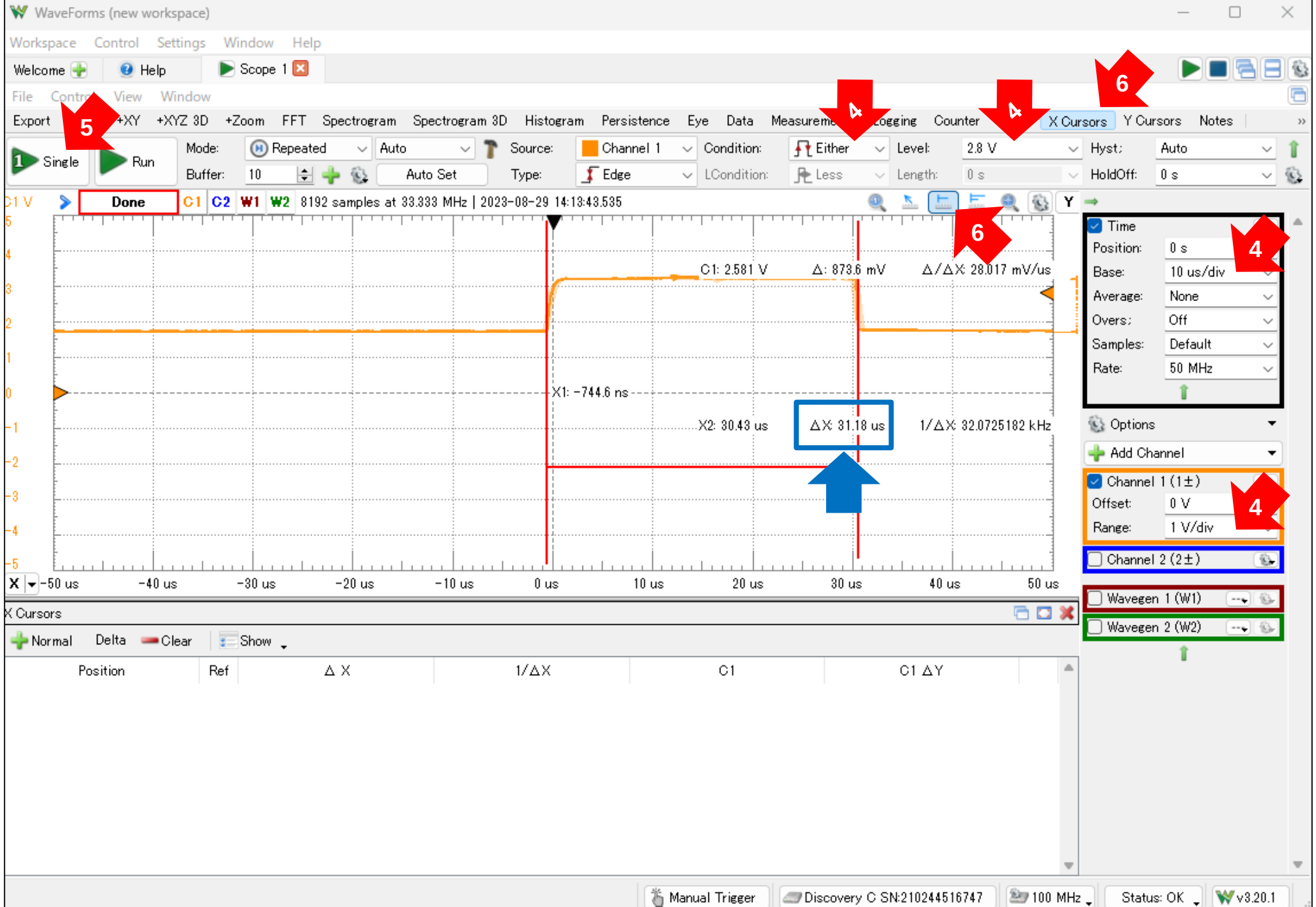
```
if (preButtonA != buttonA || preButtonB != buttonB)
{
    if( buttonA && !buttonB) {
        Xiaogyan.speaker.setTone(262);    // C4

        // [オシロデバッグ]IO制御+オシロで時間計測
        // -> LEDのON/OFFの時間をオシロスコープで計測すると，Printにかかる処理時間が見える
        unsigned long time1, time2, time_diff;
        time1 = micros();
①       Xiaogyan.led.write(HIGH);
②       Serial.println("ButtonA Pushed!!");
③       Xiaogyan.led.write(LOW);
        time2 = micros();
        time_diff = time2 - time1;
        Serial.printf("%ld[us]\n", time_diff);

    } else if (!buttonA &&  buttonB) {
        Xiaogyan.speaker.setTone(294);    // D4
    } else if ( buttonA &&  buttonB) {
        Xiaogyan.speaker.setTone(330);    // E4
    } else {
        Xiaogyan.speaker.setTone(0);
    }
}
```

① LEDをONする
② Printする
③ LEDをOFFする

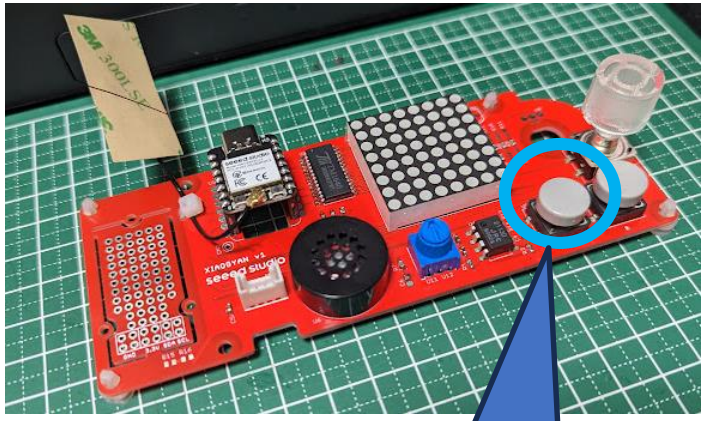
- OscilloDebugを実行ください
  - ソースコードのコメント切り替え下さい
- ボタンAを押下すると，シリアル通信で”ButtonA Pushed!!”と出力されます
- このシリアル通信の処理時間を続けて出力します.
- 合わせて，LEDのON/OFFをオシロスコープで確認してみましょう！

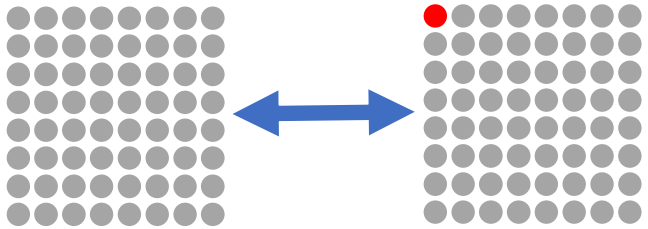
計測の手順

1. プログラムを書き込む
2. 電源を落とす
3. 基板とオシロの配線を行う
4. オシロの設定を以下の通りにする
    1. CH1・レンジ　　:　1V/div
    2. TIME・ベース　　:　10us/div
    3. Condition　　:　Either
    4. Level　　:　2.8～3.0V
5. Singleで計測
    1. Singleボタン押下後，基板のAボタン押下
    2. 画面にパルスが表示される
6. 計測後，”Quick Measure: Vertical” or ”X Cursors” でパルス幅を計測


WaveForms (new workspace)
Workspace Control Settings Window Help
Welcome Help Scope 1
File Control View Window
Export +XY +XYZ 3D +Zoom FFT Spectrogram Spectrogram 3D Histogram Persistence Eye Data Measurements Logging Counter X Cursors Y Cursors Notes
5
4
4
6
Single Run
Mode: Repeated Auto
Buffer: 10
Auto Set
Source: Channel 1
Type: Edge
Condition: Either
LCondition: Less
Level: 2.8 V
Length: 0 s
Hyst: Auto
HoldOff: 0 s
C1 V
Done C1 C2 W1 W2 8192 samples at 33.333 MHz | 2023-08-29 14:13:43.535
6
C1: 2.581 V
Δ: 873.6 mV
Δ/ΔX: 28.017 mV/us
X1: -744.6 ns
X2: 30.43 us
ΔX: 31.18 us
1/ΔX: 32.0725182 kHz
5 4 3 2 1 0 -1 -2 -3 -4 -5
X -50 us -40 us -30 us -20 us -10 us 0 us 10 us 20 us 30 us 40 us 50 us
Time
Position: 0 s
Base: 10 us/div
Average: None
Overs: Off
Samples: Default
Rate: 50 MHz
4
Options
Add Channel
Channel 1 (1±)
Offset: 0 V
Range: 1 V/div
4
Channel 2 (2±)
Wavegen 1 (W1)
Wavegen 2 (W2)
X Cursors
Normal Delta Clear Show
Position Ref Δ X 1/ΔX C1 C1 ΔY
Manual Trigger
Discovery C SN:210244516747
100 MHz
Status: OK
v3.20.1

問3-3：問題内容





■要件

Aボタンを押下したら，LEDマトリクスの左上が点灯する．もう一度押下すると，消灯するプログラム

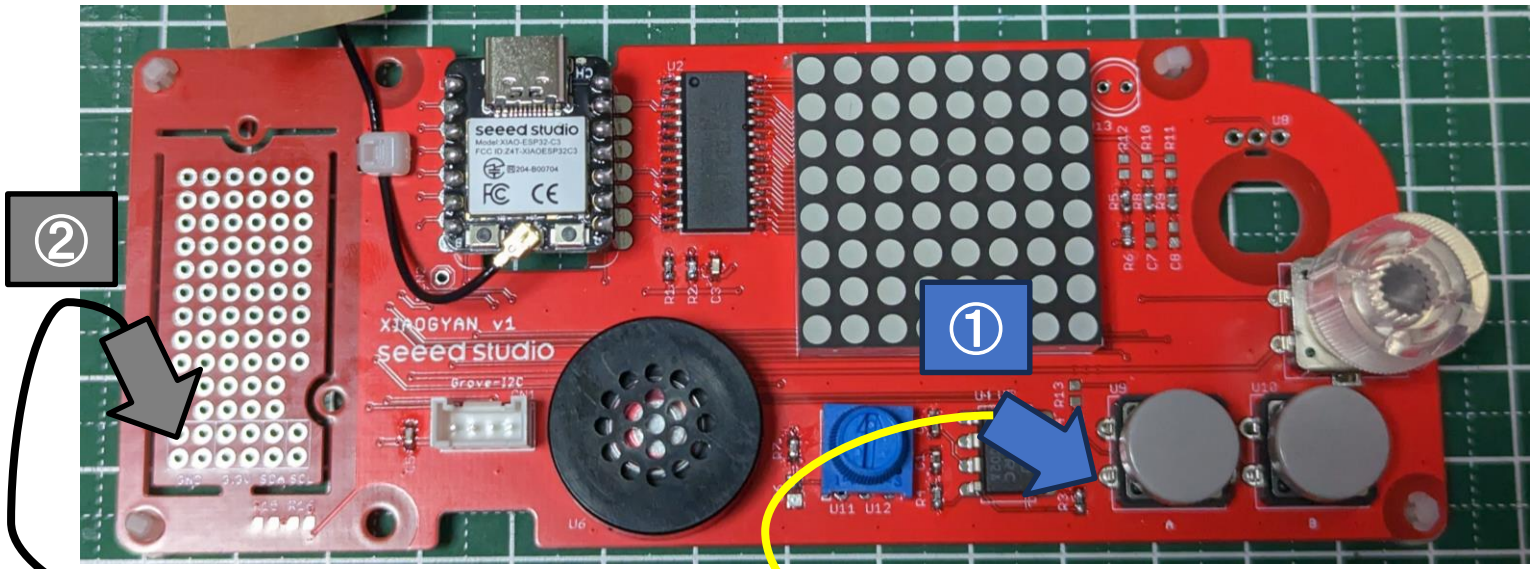
■不具合

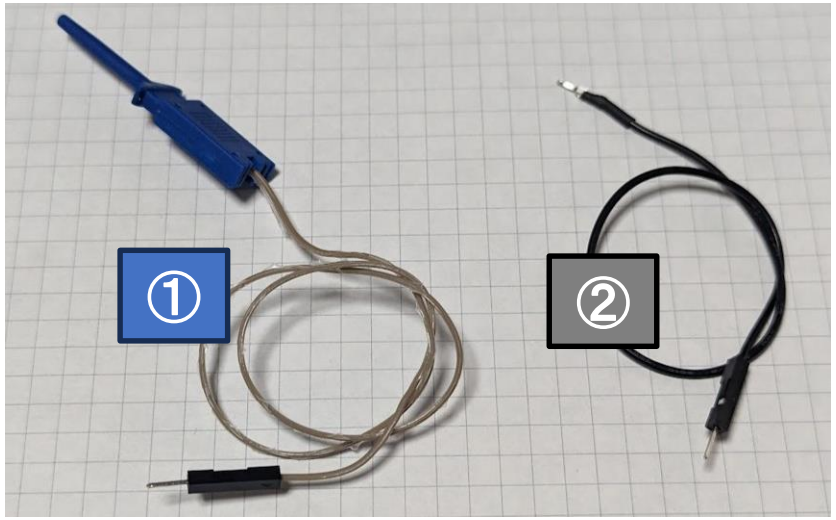
Aボタンを押したはずなのに，点灯と消灯が切り替わらない時がある

ハードウェアの繋ぎ方

・スイッチのチャタリングを計測して，ボタンの誤動作を改良に繋げましょう！

【計測ポイント】





①：テストクリップ(秋葉原のジャンク品)
②：スルーホール用テストワイヤ(TP-200)

オシロ
スコープ

演習用のプログラム：

```
if (preButtonA != buttonA || preButtonB != buttonB)
{
    if( buttonA && !buttonB) {
        // 問3-3：スイッチのチャタリング
        // -> 1度押下しただけなのに，複数押下となっている問題
        Xiaogyan.speaker.setTone(262);    // C4
        count++

        Serial.printf("%3d:ButtonA Pushed!!\n", count);

    } else if (!buttonA &&  buttonB) {
        Xiaogyan.speaker.setTone(294);    // D4
    } else if ( buttonA &&  buttonB) {
        Xiaogyan.speaker.setTone(330);    // E4
    } else {
        Xiaogyan.speaker.setTone(0);
    }
}
```

LEDの点灯/消灯は左記のコードで実施しています

行を示すｙの値がどのように変化しているか、Printを挿入して調べてみよう

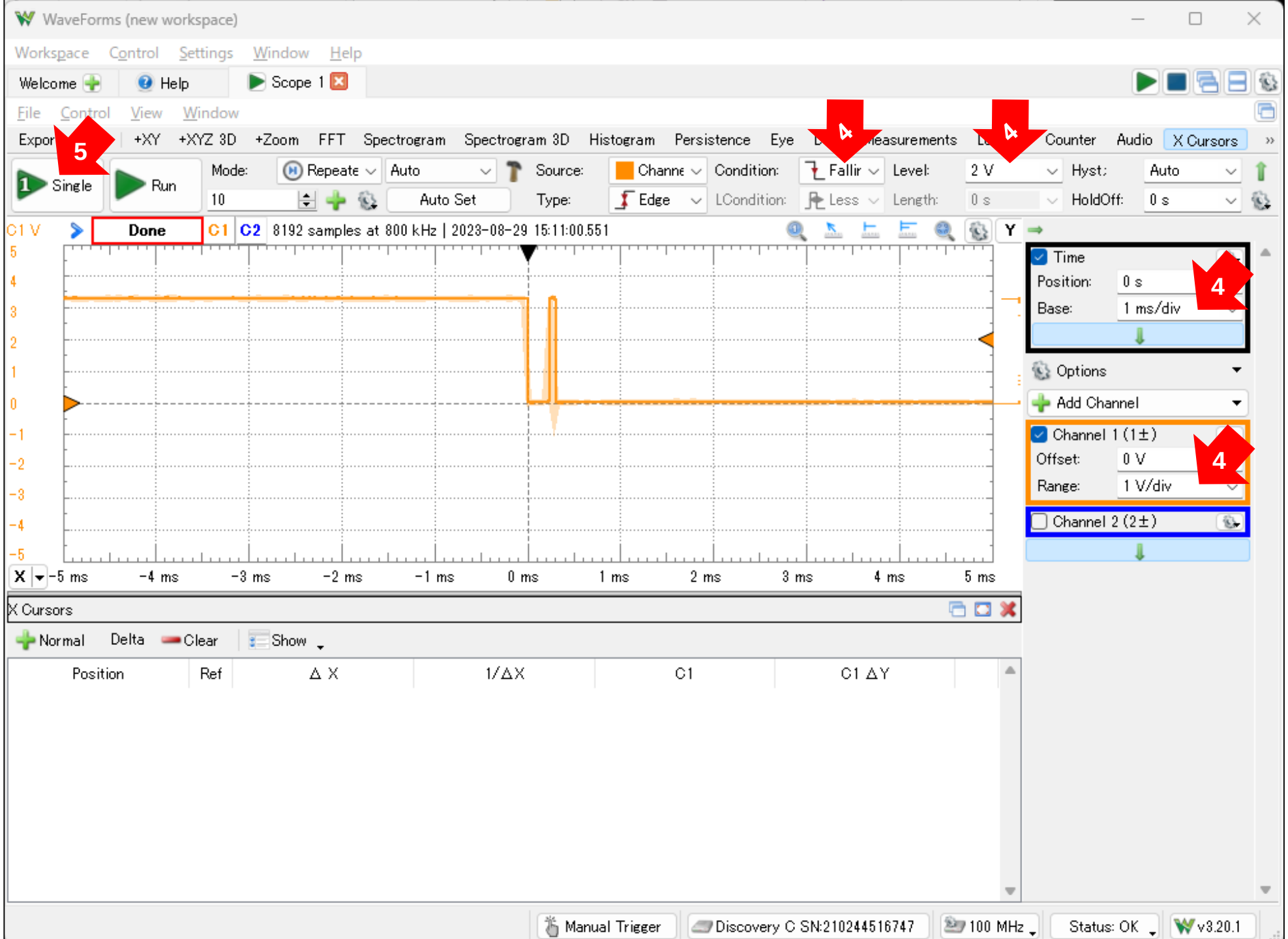
＃y=0の時に1行目を指しているとすると、
　yはどの範囲でループしてほしい？

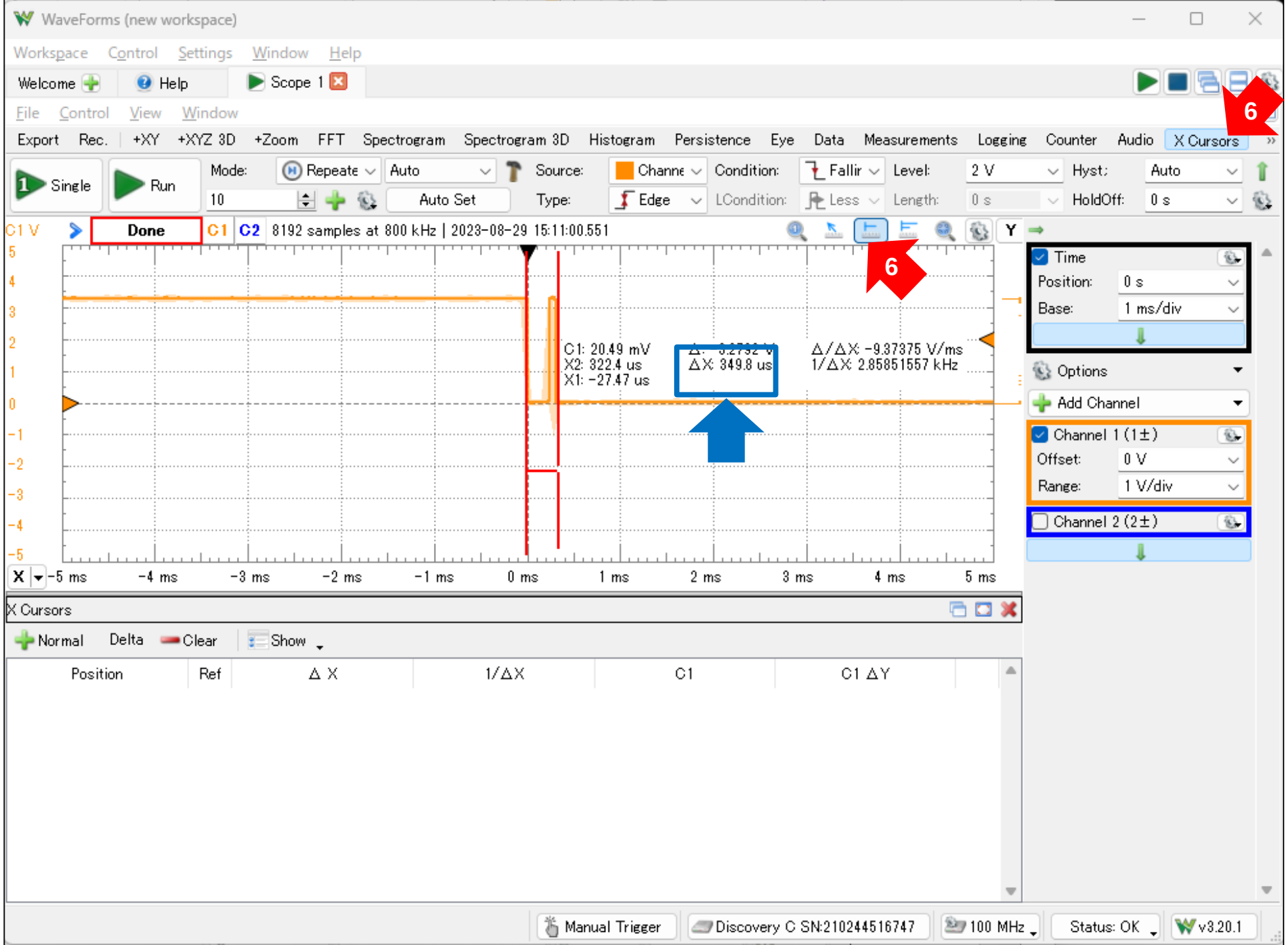
計測の手順

ボタンの押し方やタイミングによって…
・チャタリングが発生したり，しなかったり
・チャタリングが発生する期間が異なったり

する為，何度か試しましょう！

1. プログラムを書き込む
2. 電源を落とす
3. 基板とオシロの配線を行う
4. オシロの設定を以下の通りにする
    1. CH1・レンジ　　：　1V/div
    2. TIME・ベース　　：　10us/div
    3. Condition　　：　Fallir
    4. Level　　：　2.0V
5. Singleで計測
    1. Singleボタン押下後，基板のAボタン押下
    2. 画面にパルスが表示される
6. 計測後，”Quick Measure: Vertical” or ”X Cursors” で
   パルス幅を計測

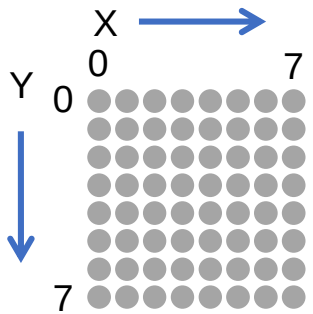

WaveForms (new workspace)
Workspace Control Settings Window Help
Welcome Help Scope 1
File Control View Window
Export +XY +XYZ 3D +Zoom FFT Spectrogram Spectrogram 3D Histogram Persistence Eye Measurements Counter Audio X Cursors
5
Single
Run
Mode: Repeate
Auto
10
Auto Set
Source: Channe
Type: Edge
Condition: Fallir
LCondition: Less
Level: 2 V
Length: 0 s
Hyst: Auto
HoldOff: 0 s
4
4
C1 V
Done
C1
C2
8192 samples at 800 kHz | 2023-08-29 15:11:00.551
5
4
3
2
1
0
-1
-2
-3
-4
-5
X
-5 ms
-4 ms
-3 ms
-2 ms
-1 ms
0 ms
1 ms
2 ms
3 ms
4 ms
5 ms
Y
Time
Position: 0 s
Base: 1 ms/div
4
Options
Add Channel
Channel 1 (1±)
Offset: 0 V
Range: 1 V/div
4
Channel 2 (2±)
X Cursors
Normal
Delta
Clear
Show
Position
Ref
Δ X
1/ΔX
C1
C1 ΔY
Manual Trigger
Discovery C SN:210244516747
100 MHz
Status: OK
v3.20.1


WaveForms (new workspace)
Workspace Control Settings Window Help
Welcome Help Scope 1
File Control View Window
Export Rec. +XY +XYZ 3D +Zoom FFT Spectrogram Spectrogram 3D Histogram Persistence Eye Data Measurements Logging Counter Audio X Cursors
6
Single Run
Mode: Repeate
Auto
10
Auto Set
Source: Channe
Type: Edge
Condition: Fallir
LCondition: Less
Level: 2 V
Length: 0 s
Hyst: Auto
HoldOff: 0 s
C1 V
Done
C1 C2
8192 samples at 800 kHz | 2023-08-29 15:11:00.551
6
C1: 20.49 mV
X2: 322.4 us
X1: -27.47 us
ΔX: 349.8 us
Δ/ΔX: -9.37375 V/ms
1/ΔX: 2.85851557 kHz
5 4 3 2 1 0 -1 -2 -3 -4 -5
-5 ms -4 ms -3 ms -2 ms -1 ms 0 ms 1 ms 2 ms 3 ms 4 ms 5 ms
Time
Position: 0 s
Base: 1 ms/div
Options
Add Channel
Channel 1 (1±)
Offset: 0 V
Range: 1 V/div
Channel 2 (2±)
X Cursors
Normal Delta Clear Show
Position Ref Δ X 1/ΔX C1 C1 ΔY
Manual Trigger
Discovery C SN:210244516747
100 MHz
Status: OK
v3.20.1

発展課題

20分程度

発展課題



- エンコーダをクリクリして，LEDマトリクスを塗りつぶすプログラムを作りましょう！
  1. 塗りつぶし位置
     1. エンコーダを時計回しすると左上から右へ、右端まで移動したら１段下の左端に移動しながら塗りつぶしする
     2. エンコーダを反時計回しすると前述の逆に移動して塗りつぶしをする

A　：●
B　：●
A&B：●

  2. 塗りつぶしの色
     1. Aボタンのみを押すと赤色で塗る
     2. Bボタンのみを押すと緑色で塗る
     3. AボタンとBボタンの同時押しすると消す（黒色で塗る）

EncoderMovePixelの動作

- EncoderMovePixelをビルドして動かしてみましょう！
- 動かしたときのおかしな動作をリストアップしましょう！
  1. エンコーダ操作
     1. 回す方向
     2. 回す早さ
  2. ボタン操作
     1. Aボタン、Bボタン、A&B同時押し
     2. 何も押さない
  3. エンコーダとボタンの組み合わせ操作
- 見つけたおかしな動作を修正しましょう！

ヒント1

```
15  // Variables
16
17  static int EncoderValue_ = 5;
18  static int Color = 0;
19
20  ////////////////////////////////////////////////////////////////
21  // setup and loop
22
23  void setup()
24  {
25      Serial.begin(115200);
26      delay(1000);
27      Serial.println();
28      Serial.println();
29
30      ////////////////////////////////////
31      // Initialize
32
33      Xiaogyan.begin();
34
35      Xiaogyan.encoder.setRotatedHandler([](bool cw){
36          const int value = EncoderValue_ + (cw ? -1 : 1);
37          EncoderValue_ = constrain(value, 0, 19);
38          Serial.println(EncoderValue_);
39      });
40
```

エンコーダを左右に回すときにカウント処理は左記のコードで実施しています

- エンコーダ値の初期値はいくつが適切？
- 回す方向に対し、増減はどうする？
- 最小値・最大値はいくつ？

ヒント2

```
66      buttonA = Xiaogyan.buttonA.read() == LOW;
67      buttonB = Xiaogyan.buttonB.read() == LOW;
68      if (preButtonA != buttonA || preButtonB != buttonB)
69      {
70          if      ( buttonA && !buttonB) Color = 1;    // Red
71          else if (!buttonA &&  buttonB) Color = 2;    // Green
72          else if ( buttonA &&  buttonB) Color = 0;    // Amber
73          else                           ;             // Black
74      }
75
76      // Led Matrix
77      static elapsedMillis ledMatrixElapsed{ 0 };
78
79      if (ledMatrixElapsed >= (EncoderValue_ + 1) * 10)
80      {
81          ledMatrixElapsed = 0;
82
83          int PositionX = EncoderValue_ % 8;
84          int PositionY = EncoderValue_ / 8;
85
86          Xiaogyan.ledMatrix.drawPixel(PositionX, PositionY, Color);
87      }
```

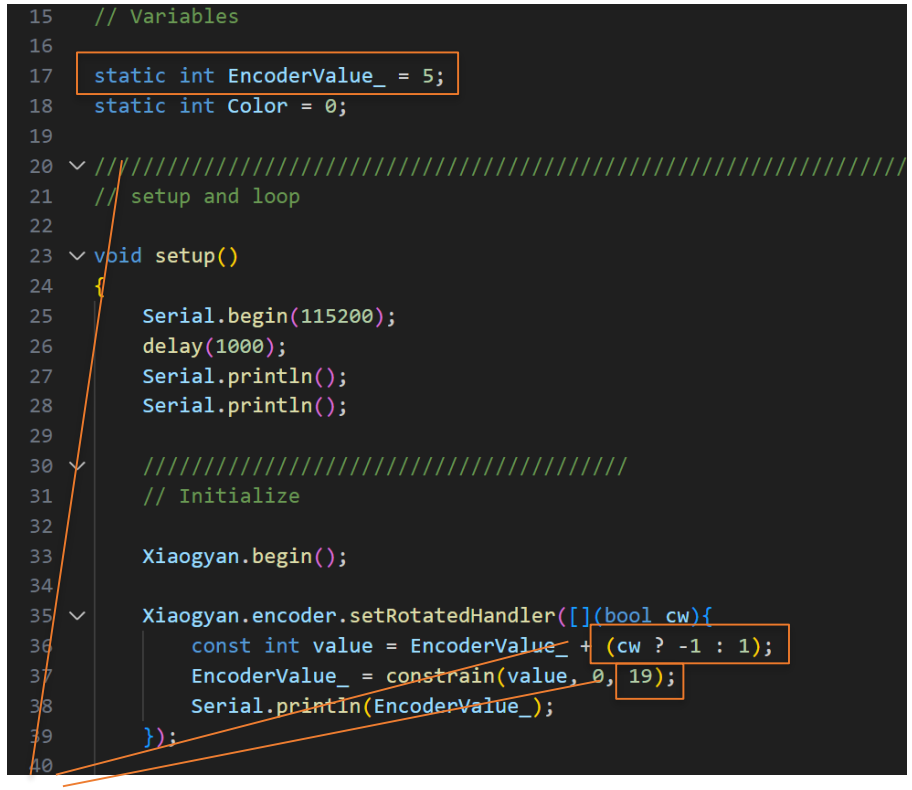
ボタン判定処理、LED表示処理は左記のコードで実施しています

- LEDはきれいに塗りつぶしできますか？
- ボタンを押さずにエンコーダを回すときの動作は？

発展課題：答え（おかしな動作）

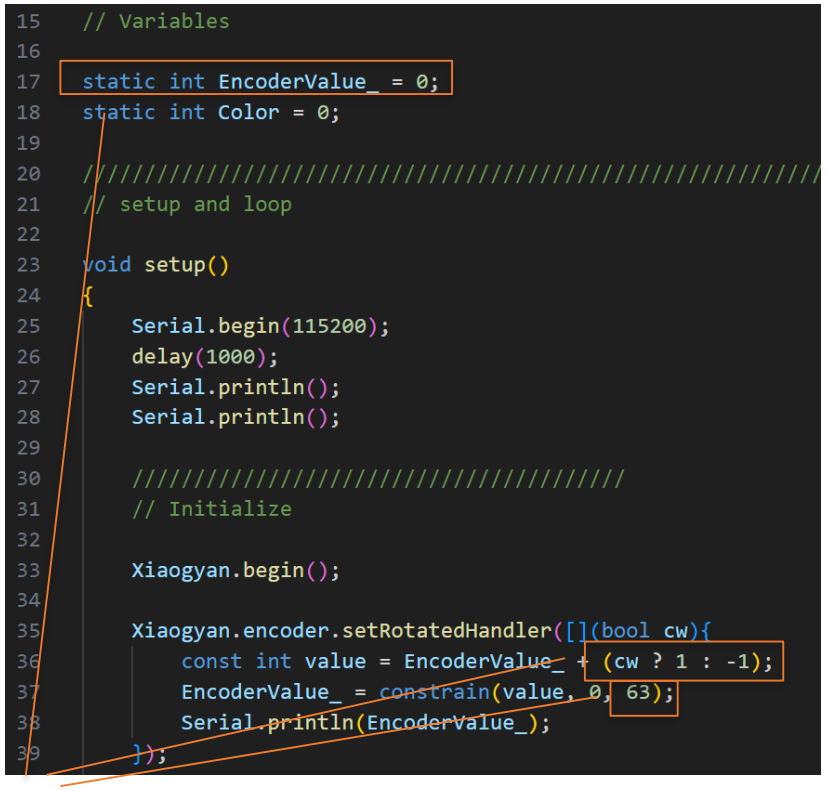
- LEDの塗りつぶしの開始位置が左上になっていない
- LEDの塗りつぶしが3行目の途中までしか実施できない
- エンコーダを回したときのポジション移動が逆になっている
- エンコーダを早く回すとLEDの塗りつぶしがスキップする
- ボタンを放していても、LEDの塗りつぶしが実施される

発展課題：答え（コード修正）



```
// Variables

static int EncoderValue_ = 5;
static int Color = 0;

//////////////////////////////////////////////////////////////////
// setup and loop

void setup()
{
    Serial.begin(115200);
    delay(1000);
    Serial.println();
    Serial.println();

    ////////////////////////////////////
    // Initialize

    Xiaogyan.begin();

    Xiaogyan.encoder.setRotatedHandler([](bool cw){
        const int value = EncoderValue_ + (cw ? -1 : 1);
        EncoderValue_ = constrain(value, 0, 19);
        Serial.println(EncoderValue_);
    });
```

修正



```
// Variables

static int EncoderValue_ = 0;
static int Color = 0;

//////////////////////////////////////////////////////////////////
// setup and loop

void setup()
{
    Serial.begin(115200);
    delay(1000);
    Serial.println();
    Serial.println();

    ////////////////////////////////////
    // Initialize

    Xiaogyan.begin();

    Xiaogyan.encoder.setRotatedHandler([](bool cw){
        const int value = EncoderValue_ + (cw ? 1 : -1);
        EncoderValue_ = constrain(value, 0, 63);
        Serial.println(EncoderValue_);
    });
```

・EncoderValue_の初期値が「5」になっています。
・setRotateHandlerの引数cw判定（3項演算子）で、時計回りで「-1」、反時計回りで「1」となっています。
・constrain()関数の第3引数で、最大値ガードが19になっています。

・EncoderValue_の初期値を「0」に修正します。
・setRotateHandlerの引数cw判定（3項演算子）で、時計回りで「1」、反時計回りで「-1」に修正します。
・constrain()関数の第3引数で、最大値ガードを「63」に修正します。

発展課題：答え（コード修正）

```
66      buttonA = Xiaogyan.buttonA.read() == LOW;
67      buttonB = Xiaogyan.buttonB.read() == LOW;
68      if (preButtonA != buttonA || preButtonB != buttonB)
69      {
70          if      ( buttonA && !buttonB) Color = 1;   // Red
71          else if (!buttonA &&  buttonB) Color = 2;   // Green
72          else if ( buttonA &&  buttonB) Color = 0;   // Amber
73          else                           ;            // Black
74      }
75
76      // Led Matrix
77      static elapsedMillis ledMatrixElapsed{ 0 };
78
79      if (ledMatrixElapsed >= (EncoderValue_ + 1) * 10)
80      {
81          ledMatrixElapsed = 0;
82
83          int PositionX = EncoderValue_ % 8;
84          int PositionY = EncoderValue_ / 8;
85
86          Xiaogyan.ledMatrix.drawPixel(PositionX, PositionY, Color);
87      }
```

修正

```
66      buttonA = Xiaogyan.buttonA.read() == LOW;
67      buttonB = Xiaogyan.buttonB.read() == LOW;
68      if (preButtonA != buttonA || preButtonB != buttonB)
69      {
70          if      ( buttonA && !buttonB) Color = 1;   // Red
71          else if (!buttonA &&  buttonB) Color = 2;   // Green
72          else if ( buttonA &&  buttonB) Color = 0;   // Black
73          else                           ;            // DO nothing
74      }
75
76      // Led Matrix
77      static elapsedMillis ledMatrixElapsed{ 0 };
78
79  //    if (ledMatrixElapsed >= (EncoderValue_ + 1) * 10)
80      {
81          ledMatrixElapsed = 0;
82
83          int PositionX = EncoderValue_ % 8;
84          int PositionY = EncoderValue_ / 8;
85
86          if (buttonA || buttonB)
87              Xiaogyan.ledMatrix.drawPixel(PositionX, PositionY, Color);
88      }
```

・LED表示更新周期がEncoderValue_の値で可変するようになっています。
・drawPixel()関数が無条件に実行されるようになっています。
・else if (buttonA && button)で設定するColorの値およびelseでの処理に対するコメントが間違っています。

・LED表示更新周期をEncoderValue_の値に依存せず最速で行うようにするために、if文の条件判定を無くします(コメント化)。
・drawPixel()関数がAボタンまたはBボタンを押されているときだけ実行されるように条件判定を追加します。
・適切なコメントに修正します。

引用元

- イラスト類
  - いらすとや：https://www.irasutoya.com/