Raspberry Pi Pico-W で作るポータブルゲーム機
（情報工学工房資料）

2024 年度
梶本研究室
ver.1 2024．3

Change Log

内容

Raspberry Pi Pico-W で作るポータブルゲーム機 1
（情報工学工房資料） 1
1 はじめに 2
2 LED の点滅 3
2.1 書き込みに失敗したら 5
3 外付けスイッチ 6
4 加速度センサ 7
5 液晶モニタの導入 9
5.1 Adafruit ILI9341 の使用 10
5.2 TFT_eSPI ライブラリ／LovyanGFX ライブラリの使用（アドバンスト） 12
5.2.1 TFT_eSPI を用いる場合 12
5.2.2 LovyanGFX ライブラリを用いる場合 13
6 音出力 16
6.1 より綺麗な音楽を鳴らす（アドバンスト） 17
6.1.1 波形の再生 17
6.1.2 ファイルからの読み込み 18
6.1.3 コアを分ける 18
6.1.4 DA 変換用の IC とオーディオアンプを用いてスピーカを駆動する 19
6.1.5 SD カードによる読み込みを可能とする 19
7 振動出力 20
8 モバイル化 21
8.1 基板を外注する 21
8.2 外観を整える 21

8.3　バッテリー駆動（アドバンスト）.. 21
9　その他.. 22
9.1　RC サーボモータ駆動（PWM）.. 22
9.2　無線通信... 22

1　はじめに

Raspberry Pi Pico/Pico-W を活用して自分だけのポータブルゲーム機を作る。ポータブルゲーム機に最低限必要な機能はおそらく 5 章の液晶モニタの導入まで。それ以降は機能を追加するために好きなだけ利用する。

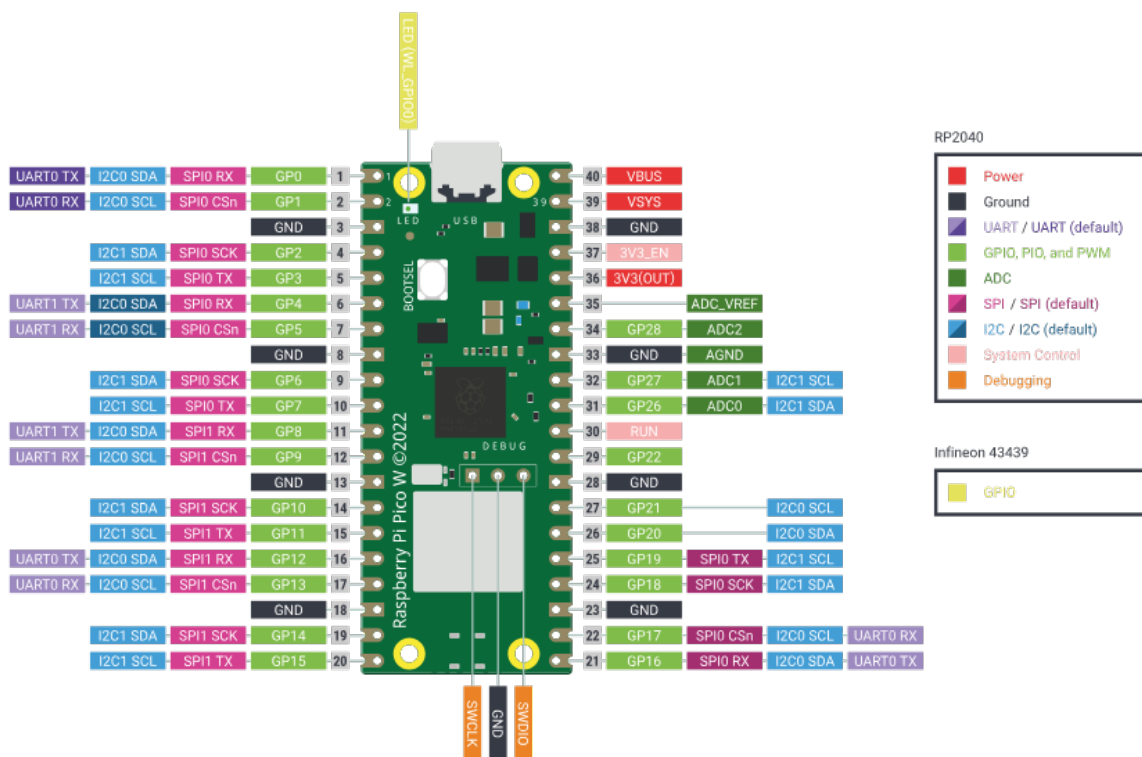


図 1　Raspberry Pi Pico W ピン配置

https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html

2　LED の点滅

開発環境を整え、外付け LED を駆動する。

Pico の足がはんだ付けされていなければする（固定のためにブレッドボードを使うと良い)。

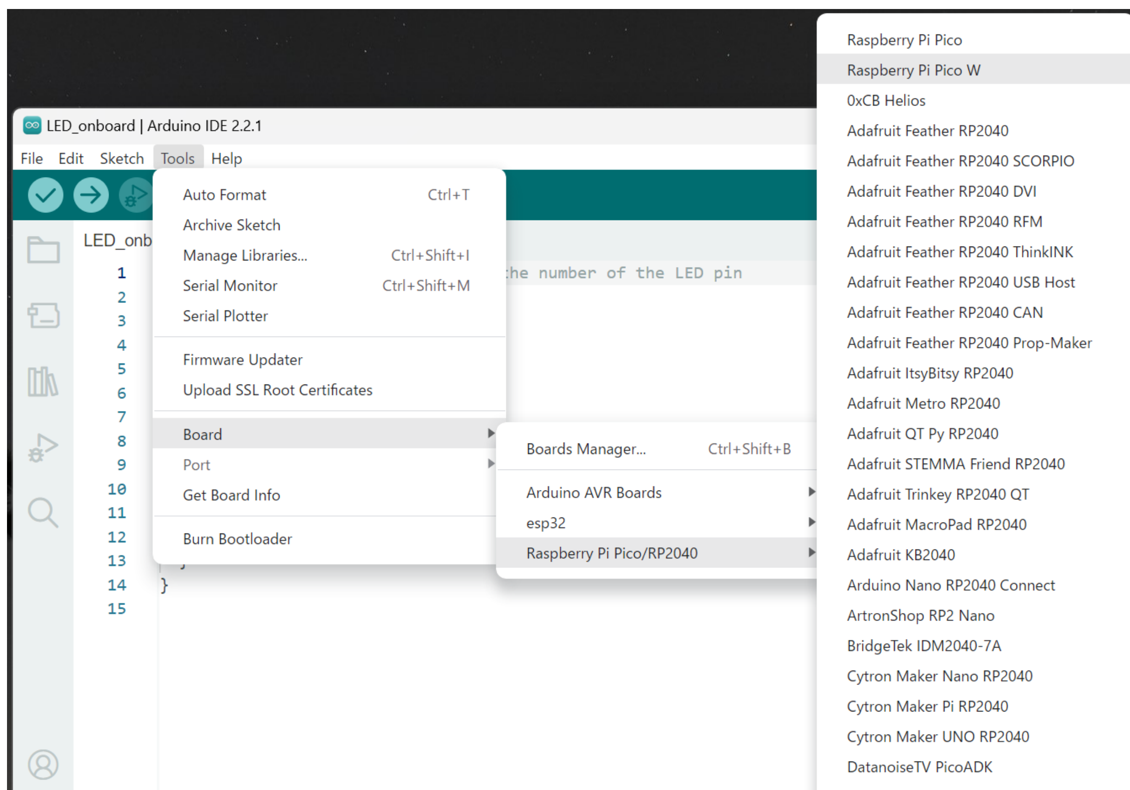
Arduino IDE を用い、Pico のライブラリを導入する。この資料では Philhower 版を用いる。例えば下記のページを参照して環境を構築すること。

https://logikara.blog/raspi-pico-arduinoide/

（こちらの説明にあるように、**ボードマネジャでは Raspberry Pi Pico /RP2040 を選ぶ**）

- Arduino IDE https://www.arduino.cc/en/software
- 追加のボードマネジャの URL
  https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json
- この版の Arduino 環境に関する最もオフィシャルなドキュメント
  https://arduino-pico.readthedocs.io/en/latest/

ボードの選択を行う。下図のように、Board→Raspberry Pi Pico/RP2040->Raspberry Pi Pico W とする。この画面が現れない場合、ボードマネジャの設定を見直す。



その後 Pico の基板に搭載された LED を点滅させる。このサンプルコードはファイル→スケッチ例→02.Digital→BlinkWithoutDelay をそのまま使えるので実際に書き込み、動作を確認する。図 2 のソースでもよい。

最初シリアルポート名が「UF2」となっていることがある。気にせず選ぶ。書き込むと「COM3」などの通常の表記に変わる。

```
const int ledPin =  LED_BUILTIN;// the number of the LED pin

void setup() {
  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  if (millis()%2000<1000) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```

図 2　オンボード LED の点滅例

次に LED を外付けする。図 3 を見ながら配線する。**ブレッドボードを真似るのではなく回路図から読み解くこと**。便宜上 PicoW の GND（グランド）端子に複数の名前が割り当てられている（GND1,GND2,...）がこれらは内部で接続されている。

- LED の極性はテスタで確認すること。
- LED に直列に接続する抵抗（制限抵抗）はここでは 1kΩとしているが、異なる電圧の場合のためにも計算できる必要がある。抵抗値の計算方法は下記リンクなどに記載されているので自習する。照明用でなければ大体 1～10mA 流す。
  http://www.ops.dti.ne.jp/~ishijima/sei/letselec/letselec11.htm
- 抵抗のカラーコードは読めるとよい。
- ここではブレッドボードを使用。外す際には USB コネクタ等に力が加わると破損するのでマイナスドライバ等を用いる。
- ブレッドボードの仕組みについて自習する。
  https://iot.keicode.com/electronics/what-is-breadboard.php

[課題1]　LED を流れる電流を計測する（抵抗間の電圧を計測し、オームの法則により算出する)。その値は前述の計算方法で求めた予想と比較して妥当か。

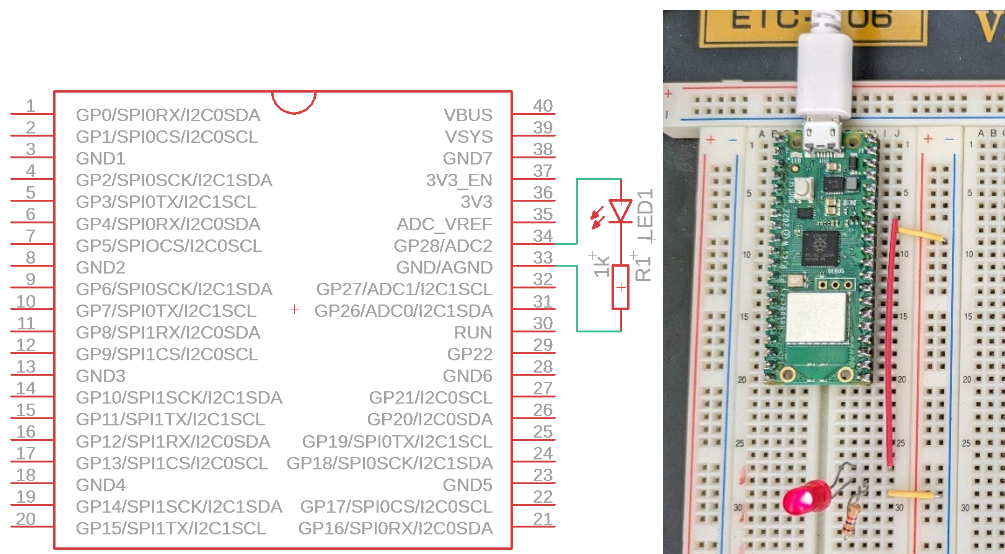


図 3　外付け LED 点滅回路

［課題2］ Arduino 環境が初めてであれば、プログラムで使われている関数を全て調べる。これは今後のすべてのプログラムで共通して行う。

Arduino 日本語リファレンス http://www.musashinodenpa.com/arduino/ref/

2.1 書き込みに失敗したら

- 書き込んでいるが LED が点灯しない場合：ボードの選択を間違えている場合が多い。「PICO」ではなく「PICO W」であることに注意。
- 書き込みがスタートせず Arduino IDE が止まる場合：PICO W のリセットを行う必要がある場合が多い。USB ケーブルを外し、ボード上のボタンを押しながら USB ケーブルを接続する。するとポート名が COM*ではなく UF2 という名前に変わる。こうしたことは SPI 通信のために自分でポートを設定する場合に多発するようである。
https://karakuri-musha.com/inside-technology/arduino-raspberrypi-picow-tips-cannotwriteprogram0101/

3　外付けスイッチ

外部からスイッチ入力を受け付けるようにする。図 4 参照。タクトスイッチは押下時どこが導通するかテスタでチェックしてから使用する。

回路図は簡略化のためにネット名が表記され、同じネット名は接続されていることを表す。例えば図 4 の「GND」同士、「+3V3」同士は接続される。電源と GND はブレッドボードの電源線（赤と青の線）を利用して配線する。写真のブレッドボードは電源線（右端の赤線）が二つに分かれていることに注意。

[課題3]　スイッチ回路に抵抗が必要な理由を考察する。ボタンを押したときに LOW になる回路はどのようにすればよいか。

[課題4]　タクトスイッチを押すと LED が光るようにする(digitalRead 関数)

[課題5]　割込みプログラミングについて自習し、割込みを用いたプログラムに改変する。スイッチを押すたびに LED が点滅するようにする（attachInterrupt 関数）

参考：http://gammon.com.au/interrupts

[課題6]　スイッチが押されるたびに、押された回数をシリアル通信で PC に伝えるようにする。シリアル通信の方法は自習する。PC 側は Arduino IDE のシリアルモニタ機能を用いるが RealTerm, TeraTerm などの通信ソフトを用いてもよい（文字列ではなくバイナリデータを確認する場合などに必須となる）。

RealTerm http://realterm.sourceforge.net/

TeraTerm http://sourceforge.jp/projects/ttssh2/

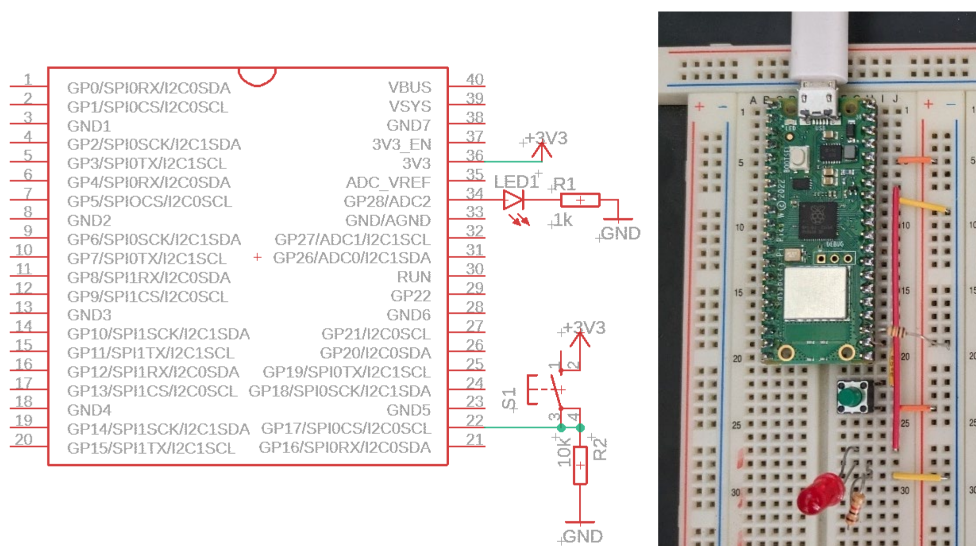


図 4　タクトスイッチ回路

4　加速度センサ

KXR94-2050 モジュール（http://akizukidenshi.com/catalog/g/gM-05153/ ）を用いる。このモジュールは 3 軸加速度をアナログ出力するので、3 つの AD ポートで受け取れば良い。

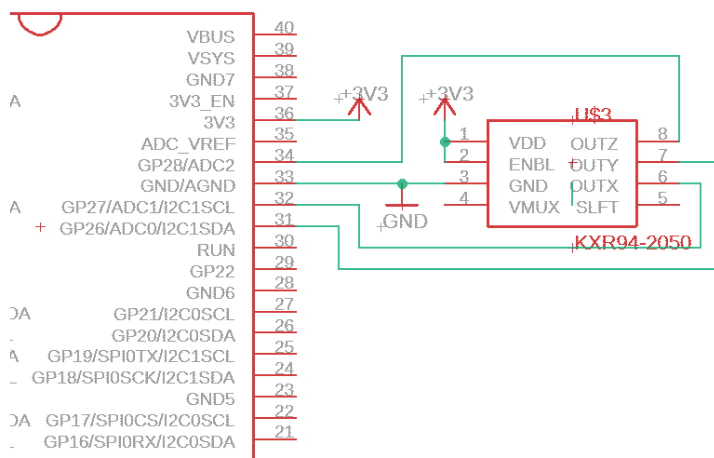
関数は analogRead。関数の引数は GPIO の番号となる。例えば AD0 は 0 番ではなく 26 番となる（これは後述する SPI 等でも同様）。

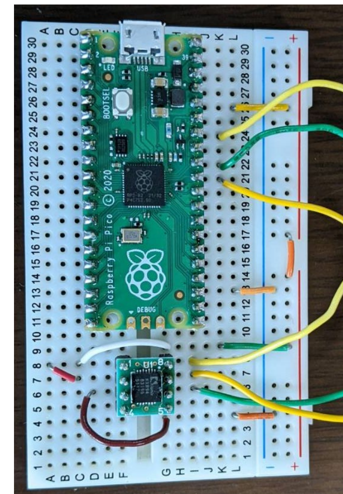
この加速度センサは 3.3V 電源に接続した場合、オフセット 1.65V、1G 加わった際に 0.66V の変化がある。つまり重力加速度の範囲では大体 0.99V～2.31V 出力が変化することになる。 Pico の AD 変換は 0～3.3V を 0～最大値(bit 数依存)に割り当てて出力する。**なおここで使用している Arduino 環境では 10bit 出力がデフォルトなので analogReadResolution 関数で 12bit に設定する**。これによって最大値は 4095 となる。下記のドキュメントを参照。

https://arduino-pico.readthedocs.io/en/latest/analog.html

またここでは出力値を Arduino IDE のシリアルプロッタで確認する。カンマ区切りまたはタブ区切り(“¥t”)で記載することにより複数の変数を描画することができる（図 6)。

[課題7] PC で読み取った値が妥当であることを確認する。重力加速度が存在するため、ある軸のセンサ値は、ある向きで+1G、逆向きで-1G となるはずである。実際にどのような値で変化しているか。それは妥当か。





```
const int ADC0 = 26; //GP26
const int ADC1 = 27; //GP27
const int ADC2 = 28; //GP28

int ax, ay, az;

void setup() {
  Serial.begin(115200);
  analogReadResolution(12); //Set the ADC resolution to 12bit
}

void loop (){
  ax = analogRead(ADC1);
  ay = analogRead(ADC0);
  az = analogRead(ADC2);
  String message = String(ax) + ", " + String(ay) + ", " + String(az);
  Serial.println(message);
  delay(100);
}
```

図 5　加速度センサ回路とサンプルプログラム

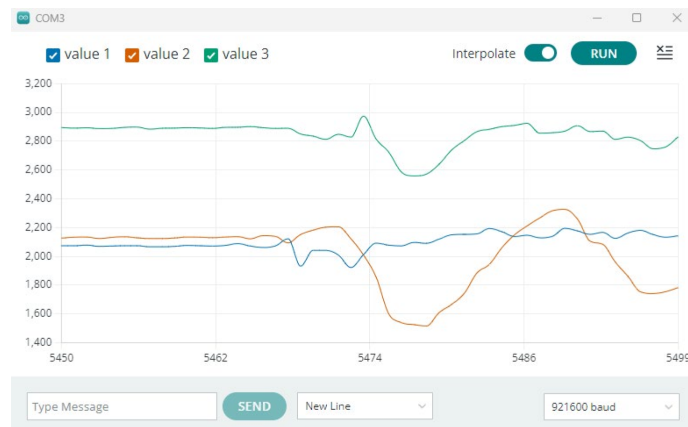


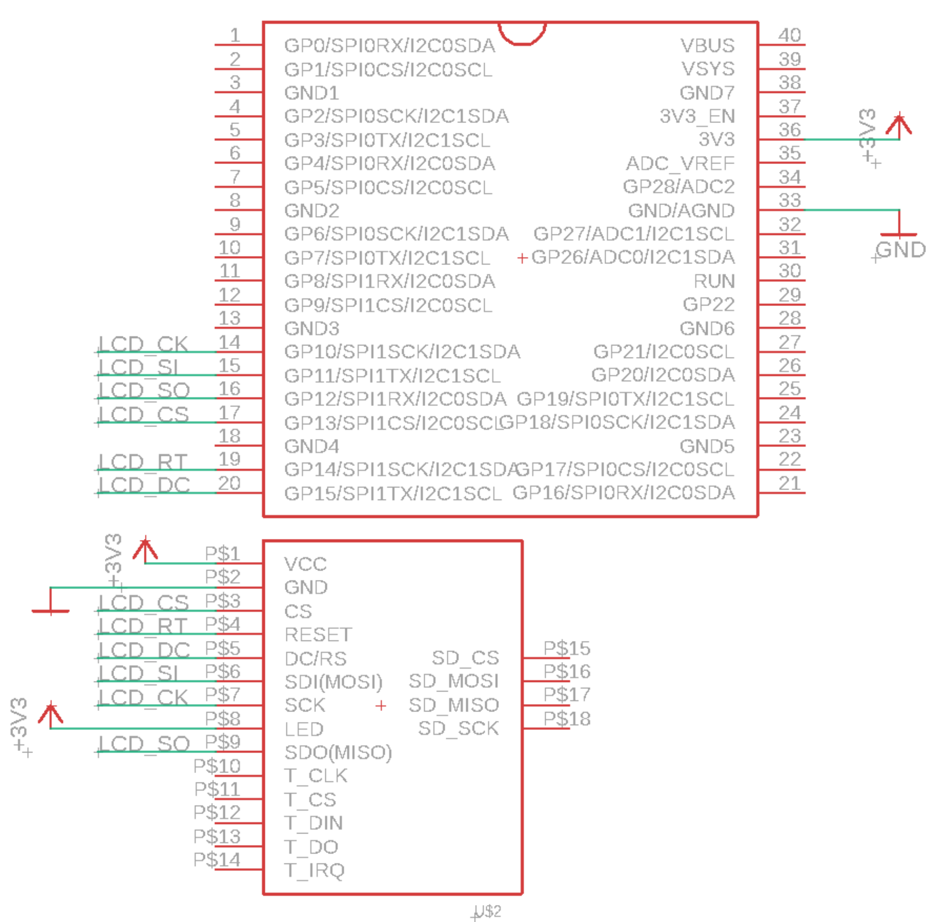
図 6　シリアルプロッタによる観察

5　液晶モニタの導入

本章ではポータブルゲーム機のために液晶モニタを使用する。ここで使う液晶モニタはタッチパネル機能付きで、SD カードリーダも付属している。

- ILI9341 搭載 2.8 インチ SPI 制御タッチパネル付 TFT 液晶 MSP2807
  https://akizukidenshi.com/catalog/g/g116265/
- データシート等
  http://www.lcdwiki.com/2.8inch_SPI_Module_ILI9341_SKU:MSP2807
- Raspberry Pi Pico による液晶ゲーム制作
  http://www.ze.em-net.ne.jp/~kenken/picogames/index.html
- LCD モジュールを使う(ILI9341)
  https://tamanegi.digick.jp/computer-embedded/module/ili9341/

ブレッドボードで回路を作成する。バックライト LED は 3.3V に接続している（これは適当な出力ピンに繋いでバックライトを制御可能としても良い）。以降ボード上の LED は使わないため外して構わない。



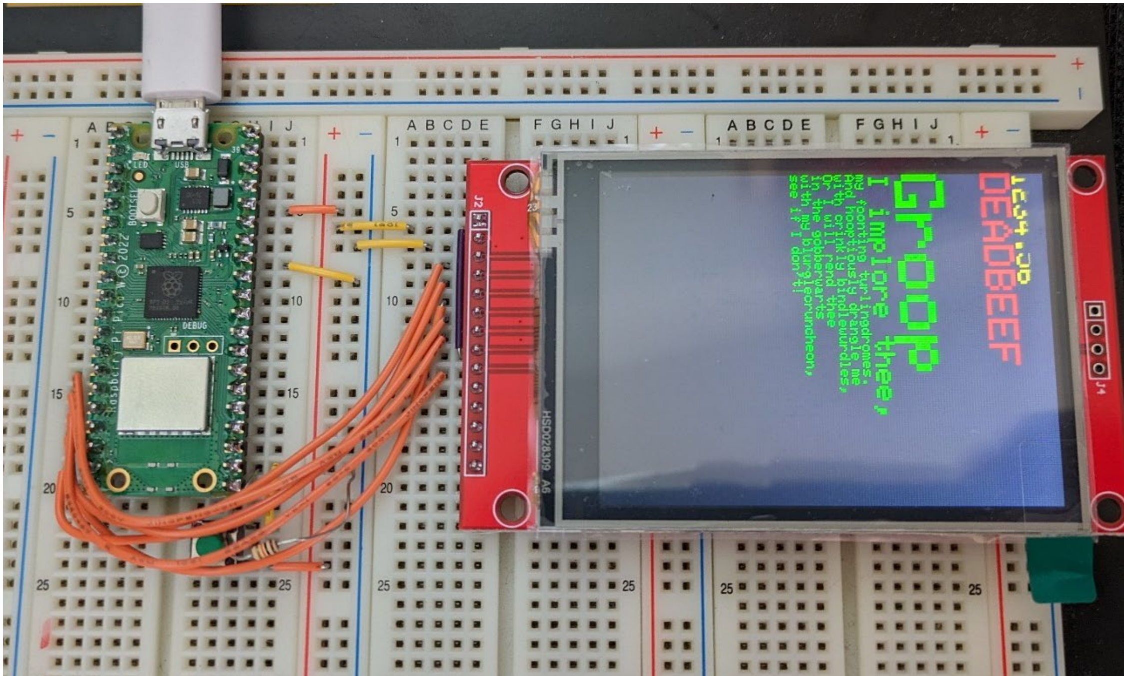


図 7　LCD の接続

5.1 Adafruit ILI9341 の使用

この液晶を駆動するためにライブラリを導入する。ここでは最も導入が容易な Adafruit ILI9341 を用いる。ArduinoIDE のライブラリマネジャ（Tools->Manage Libraries）で「ILI9341」を検索。最上部に出てくる Adafruit ILI9341 をインストールする。この際関連するライブラリのインストールを許可するかどうか聞いてくるので許可しておく。

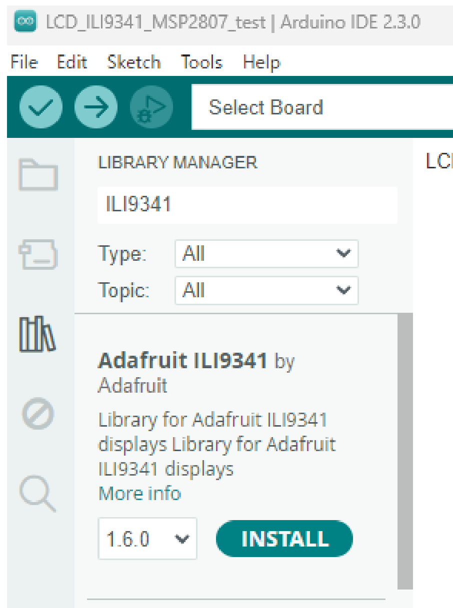


図 8　ライブラリマネジャ

Adafruit ILI9341 Arduino Library https://github.com/adafruit/Adafruit_ILI9341 に行き、examples/graphicstest/graphicstest.ino を丸ごとコピー&ペーストする。使用しているピン番号が異なるため、図 9 の灰色で示した部分の修正を行う（回路図が示すデジタル IO ポートの番号と見比べること)。

```
17  #include "SPI.h"
18  #include "Adafruit_GFX.h"
19  #include "Adafruit_ILI9341.h"
20
21  // ***** Modified for Raspberry Pi Pico W *****
22  #define TFT_DC 15
23  #define TFT_CS 13
24  #define TFT_MOSI 11
25  #define TFT_CLK 10
26  #define TFT_MISO 12
27  #define TFT_RST 14
28
29  Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK, TFT_RST, TFT_MISO);
30  // **********
31
32  void setup() {
33    Serial.begin(9600);
34    Serial.println("ILI9341 Test!");
```

図 9　LCD 用グラフィックステストプログラムの修正部分

書き込んで動作させてみる。シリアルポートで状態を出力しているのでシリアルモニタでも確認する。このプログラムには画像表示のほぼすべての機能（ビットマップ画像の扱いは除く）が関数として網羅されているので、基本的にこのプログラムの関数群を使ってゲーム画面を構築することができる。なお描画用の関数は下記の Adafruit_GFX ライブラリで定義されている。SPI 通信関係は同ライブラリ中の Adafruit_SPITFT が担っている。

https://github.com/adafruit/Adafruit-GFX-Library/blob/master/Adafruit_GFX.h

[課題8]　サンプルプログラムを改変し、適当な球を描画する。

[課題9]　球が動くようにしてみる。壁で跳ね返るようにしてみる。

[課題10]　Adafruit ILI9341 Arduino Library のサンプルの一つ、pictureEmbed を実行してみる。上述の修正点のほかに、図 10 の修正（コメントアウト）が必要。

```
44  void loop(void) {
45    for(uint8_t r=0; r<4; r++) {
46      tft.setRotation(r);
47      tft.fillScreen(ILI9341_BLACK);
48      for(uint8_t j=0; j<20; j++) {
49        tft.drawRGBBitmap(
50          random(-DRAGON_WIDTH , tft.width()),
51          random(-DRAGON_HEIGHT, tft.height()),
52  //#if defined(__AVR__) || defined(ESP8266)
53          dragonBitmap,
54  //#else
55          // Some non-AVR MCU's have a "flat" memory model and don't
56          // distinguish between flash and RAM addresses.  In this case,
57          // the RAM-resident-optimized drawRGBBitmap in the ILI9341
58          // library can be invoked by forcibly type-converting the
59          // PROGMEM bitmap pointer to a non-const uint16_t *.
60  //        (uint16_t *)dragonBitmap,
61  //#endif
62          DRAGON_WIDTH, DRAGON_HEIGHT);
63        delay(1); // Allow ESP8266 to handle watchdog & WiFi stuff
64      }
65      delay(3000);
66    }
67  }
```

図 10　pictureEmbed サンプルプログラムの修正（コメントアウト）部分

5.2 TFT_eSPI ライブラリ／LovyanGFX ライブラリの使用（アドバンスト）

Adafruit ILI9341 Arduino Library は簡単に使える反面、速度が速いとは言えず、特にアニメーションを表示する際にちらつきが目立ちやすい。

より高速なライブラリとして、TFT_eSPI または LovyanGFX を利用する。導入の難易度に大きな違いはない。海外を含めると TFT_eSPI がやや使用例が多いように見える。ただしインストールされた定義ファイルを自分で直接編集する必要がありプログラムの可搬性が落ちる（回避方法はある）。LovyanGFX はこの問題がなく、より高速とされ、日本語のドキュメントがある。

いずれにしてもライブラリのソースを見てある程度内容を理解できる必要があるため、これらのライブラリの使用は自信のある人向け。

5.2.1　TFT_eSPI を用いる場合

ArduinoIDE のライブラリマネジャ（Tools->Manage Libraries）で「TFT_eSPI」を検索、一番上に出てくるライブラリをインストールする。

- TFT_eSPI ライブラリ https://github.com/Bodmer/TFT_eSPI

このライブラリの特徴として、定義用のファイルを自分で編集する必要がある。

Arduino のライブラリをインストールしているフォルダを探す。多くの場合、

C:¥Users¥ユーザ名¥Documents¥Arduino¥libraries であるが、Arduino IDE の Preference から調べることができる。

TFT_eSPI フォルダの User_Setup.h を開き、図 11 を参考に今回のセットアップに合うように修正する。

```
170 #define TFT_MISO  12  // Automatically assigned with ESP8266 if not defined  /****Changed!***/
171 #define TFT_MOSI  11  // Automatically assigned with ESP8266 if not defined  /****Changed!***/
172 #define TFT_SCLK  10  // Automatically assigned with ESP8266 if not defined  /****Changed!***/
173
174 #define TFT_CS    13  // Chip select control pin D8  /****Changed!***/
175 #define TFT_DC    15  // Data Command control pin  /****Changed!***/
176 #define TFT_RST   14  // Reset pin (could connect to NodeMCU RST, see next line)  /****Changed!***/
```

```
348 #define TFT_SPI_PORT 1 // Set to 0 if SPI0 pins are used, or 1 if spi1 pins used  /****Changed!***/
```

```
359 // #define SPI_FREQUENCY   1000000
360 // #define SPI_FREQUENCY   5000000
361 // #define SPI_FREQUENCY  10000000
362 // #define SPI_FREQUENCY  20000000
363 // #define SPI_FREQUENCY  27000000
364 #define SPI_FREQUENCY  40000000 /***Changed!!***/
365 // #define SPI_FREQUENCY  55000000 // STM32 SPI1 only (SPI2 maximum is 27MHz)
366 // #define SPI_FREQUENCY  80000000
```

図 11　TFT_eSPI フォルダの User_Setup.h 修正部分

サンプルプログラムを動作させてみる。TFT_eSPI ライブラリのページの examples→320x240→TFT_Clock を開き、コピーして動作させてみる。

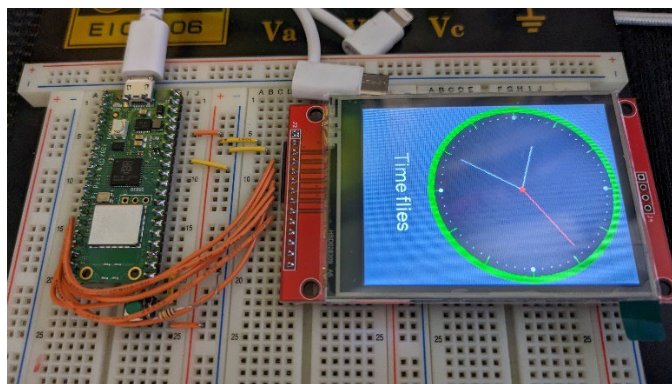


図 12　TFT_Clock https://github.com/Bodmer/TFT_eSPI/tree/master/examples/320%20x%20240/TFT_Clock

5.2.2　LovyanGFX ライブラリを用いる場合

ArduinoIDE のライブラリマネジャ（Tools->Manage Libraries）で「LovyanGFX」を検索、一番上に出てくるライブラリをインストールする。

- LovyanGFX ライブラリ https://github.com/lovyan03/LovyanGFX
- LovyanGFX 関数の解説 https://lang-ship.com/blog/work/lovyangfx-1/#toc32

このライブラリでは自分でヘッダファイルを用意し、その中でピン番号や液晶モニタの

種類を指定する。そのヘッダファイルをメイン関数中で読み込むことによって動作する。

ここでは下記のサンプルのヘッダファイルを書き換えて自分の環境に合わせたヘッダファイルを作成する。図 13 に修正したものを示す。日本語コメントを付けた部分が修正点。
https://github.com/lovyan03/LovyanGFX/blob/master/src/lgfx_user/LGFX_RP2040_096_Waveshare_sample.hpp

```
 6  #define LGFX_USE_V1
 7
 8  #include <LovyanGFX.hpp>
 9
10  // LGFX for Waveshare RP2040-LCD-0.96
11  // https://www.waveshare.com/wiki/RP2040-LCD-0.96
12
13  class LGFX : public lgfx::LGFX_Device
14  {
15    lgfx::Panel_ILI9341 _panel_instance; //使用する液晶に合わせて変更。ここではILI9341
16    lgfx::Bus_SPI       _bus_instance;
17    lgfx::Light_PWM     _light_instance;
18
19    public:
20    LGFX(void)
21    {
22      {
23        auto cfg = _bus_instance.config();
24        cfg.spi_host   = 1;
25        cfg.spi_mode   = 0;
26        cfg.freq_write = 80000000;  //もし通信速度が速すぎて動かない場合10000000等に変更
27        cfg.pin_sclk   = 10; //ボードの設計に合わせて変更。SPI通信のクロック信号のポート
28        cfg.pin_miso   = -1; //今回の液晶にはMISO端子は不要なため-1のままにする
29        cfg.pin_mosi   = 11; //ボードの設計に合わせて変更。SPI通信のMOSI（TX)信号のポート
30        cfg.pin_dc     = 15; //ボードの設計に合わせて変更。DC信号のポート
31        _bus_instance.config(cfg);
32        _panel_instance.setBus(&_bus_instance);
33      }
34
35      {
36        auto cfg = _panel_instance.config();
37        cfg.pin_cs       = 13;  //ボードの設計に合わせて変更。CS信号のポート
38        cfg.pin_rst      = 14;  //ボードの設計に合わせて変更。RST信号のポート
39        cfg.panel_width  = 240; //液晶の仕様に合わせて変更。ILI9341の場合240
40        cfg.panel_height = 320; //液晶の仕様に合わせて変更。ILI9341の場合240
41        cfg.offset_x     = 0;   //液晶の仕様に合わせて変更。ここでは0
42        cfg.offset_y     = 0;   //液晶の仕様に合わせて変更。ここでは0
43        cfg.invert       = false; //補色モードかどうか。trueだと0xFFFFFFが黒になる
44        cfg.rgb_order    = false;
45        //cfg.offset_rotation = 0;
46        _panel_instance.config(cfg);
47      }
48
49      {
50        auto cfg = _light_instance.config();
51        cfg.pin_bl      = 22;   //ボードの設計に合わせて変更。バックライトLED制御をマイコンから行っていれば書く
52        cfg.pwm_channel = 0;//1;//ボードの設計に合わせて変更。PWM channelは0か1になる。
53        _light_instance.config(cfg);
54        _panel_instance.setLight(&_light_instance);
55      }
56
57      setPanel(&_panel_instance);
58    }
59  };
```

図 13　**LovyanGFX のヘッダファイルの修正（修正部分は日本語コメント）**

修正したヘッダファイルに適当な名前を付けて（例えば LGFX_RP2040_ILI9341.hpp)、プロジェクトと同じフォルダに入れ、メインプログラムからインクルードする。

サンプルプログラムと動作の様子を図 14 に示す。このサンプルプログラムではバックライト LED を制御するようにしているが、バックライト LED 用端子を 3.3V に接続している場合は不要である。またスプライト(Sprite)を定義しているがプログラム中で使用していない。ビットマップ画像の高速な転送を行いたい場合にはスプライト機能を使うとよい。

```
1   #include <SPI.h>
2   #include "LGFX_RP2040_ILI9341.hpp"  // 自分で作成したヘッダファイルをインクインクルード
3
4   static LGFX lcd;                 // LGFXのインスタンスを作成。
5   static LGFX_Sprite sprite(&lcd); // スプライトを使う場合はLGFX_Spriteのインスタンスを作成。
6   const int BACKLIGHT=22; //ここではバックライトの制御も行っている。バックライト常時ONの場合は不要。
7
8   void setup() {
9     pinMode(BACKLIGHT, OUTPUT);
10    delay(1);
11
12    Serial.begin(9600);
13    lcd.init();  // 最初に初期化関数を呼び出します。
14    lcd.setRotation(1);// 回転方向を 0～3 の4方向から設定します。(4～7を使用すると上下反転になります。)
15    lcd.setBrightness(128); //バックライトを制御する場合。バックライト常時ONの場合は不要。
16    lcd.setColorDepth(16);  // RGB565の16ビットに設定
17    lcd.fillScreen(lcd.color565(0,0,0));  // 画面全体の塗り潰し。
18  }
19
20  void loop() {
21    int count=0, i;
22    lcd.fillRect(0, 0,100,100, lcd.color565(255,0,0));  // 矩形の塗り
23    lcd.fillCircle    (200,200, 50, lcd.color565(0,255,0));  // 円の塗り
24    lcd.fillTriangle  ( 200, 0, 200, 100, 100, 0,  lcd.color565(0,0, 255)); // ３点間の三角形の塗り
25
26    delay(20);
27  }
```

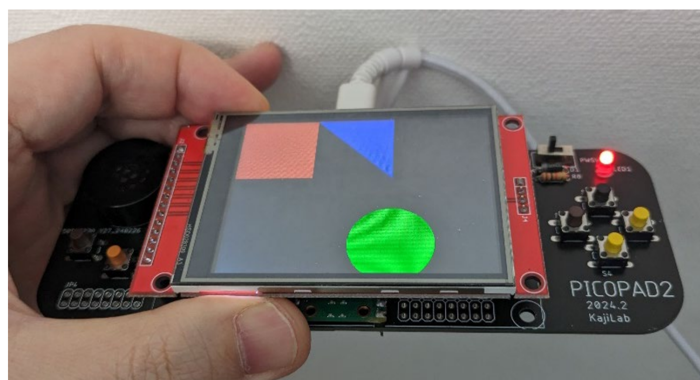


図 14　LovyanGFX のサンプルプログラムと動作の様子

6　音出力

PWM 出力を利用して圧電ブザーで音声出力を行う。 圧電ブザーは電磁式の一般的なスピーカと比べて音質は悪いがマイコンのデジタル IO 端子に直結しても音を出すことができるため、簡易的な方法として良く用いられる。

一般に圧電式ブザーにはアクティブなもの（共振回路が入っており直流電圧を加えれば音が出るもの。周波数の変更は難しい）とパッシブなもの（交流電圧を加えることで音が出るもの）がある。ここではパッシブなものを用いる。

- 圧電サウンダ PKM13EPYH4000-A0 https://akizukidenshi.com/catalog/g/g104118/

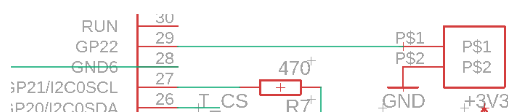


図 15　圧電ブザーの回路

Pico は通常のアナログ出力を持っておらず、代わりに PWM（パルス幅変調）によって疑似的なアナログ出力を可能としている(analogWrite 関数)。PWM 周波数は analogWriteFreq 関数で変更できるため、これを鳴らしたい音の周波数にすることで一定の周波数のビープ音を発生させる。例えばプログラムの一部は下記のようになる。

```
～ファイル冒頭の定義部分～
const int buzzerPin = 22; //GPIO22 をブザーに接続している場合。
#define DOREMI_NUM 8
const int DoReMi[DOREMI_NUM]={523,587,659,698,784,880,988,1047};
～setup 関数の一部～
  pinMode(buzzerPin, OUTPUT);
  analogWriteResolution(12); //Analog range set to 0-4095
  for (int i=0;i<DOREMI_NUM;i++) {
    analogWriteFreq(DoReMi[i]); //PWM set to certain frequency
    analogWrite(buzzerPin, 4095/2); // PWM 信号をブザーに送る
    delay(500); // 0.5 秒待つ
  }
  analogWrite(buzzerPin, 0); // 音を止める
```

`analogWrite(buzzerPin, 4095/2);` という部分に注意する。これは最大出力電圧の半分の出力という意味だが、PWM による波形表現を考えると、決められた周期のうちの

半分の時間１を出力し、残りの半分の時間０を出力するということになる。**つまりこの時音としては最大になる。**

[課題11]　　PWM について自習する。その後、上記を参考にして音を鳴らしてみる。

6.1 より綺麗な音楽を鳴らす（アドバンスト）

これまでに試した方法では、矩形波によるビープ音しか対応できず、同時に発生する音も１音である。これでも効果音として十分だが、BGM を鳴らしたいような場合には力不足になる。

音楽の再生を行っていくには、（１）一定周波数の矩形波ではなく「波形」を再生する、（２）音楽ファイルを用意する、というステップが必要になる。さらに音楽の再生は負荷が高くゲーム本体のプログラムに影響するため、（３）CPU コアを分けることが行われる。さらに圧電素子自体が狭い周波数帯域でしか駆動できないため（４）電磁式のスピーカを駆動する、こともできる。（４）までできれば一種の音楽プレイヤーと言ってよいものになる。いずれも概要のみ示すので必要であれば相談のこと。

6.1.1　波形の再生

図 16 のように analogWrite 関数を使って DA 出力を行う。これまでと原理的に違うところはないが、PWM の更新周波数を大幅に上げ、60kHz としている。またこの場合、マイコンの制約からこれまでのように 12bit(0-4095)での値の指定はできなくなり、8bit(0-255)での指定となる。矩形波の集まりであることには変わりがないが、周波数が非常に高いためにアナログ値の電圧を与えたのと同じ効果になる。

このプログラムは PWM によって 100Hz の正弦波を表現しようとしており、**正確な周波数を実現するために micros 関数を使っている。** micros 関数はプログラム起動時から現在までの時間をマイクロ秒単位で返す関数であり、これを 1/1000000 すれば現在の時刻が秒単位で得られる。例えば f[Hz]の周波数の音を出したい場合、現在の時刻が t[s]であれば、sin(2πft)を出力すれば良い。

この例では出力ピンを GP21 としているが実際の状況に合わせる。

```
const int analogOutPin = 21; //GP21
unsigned long t;
int outputValue = 128;        // value output to the PWM

void setup() {
  analogWriteFreq(60000); //PWM set to 60kHz
}

void loop() {
  t = micros();
  outputValue = (int)(128.0 * sin(2.0 * PI * 100.0 * (float)t/1000000.0)+127.0);
  analogWrite(analogOutPin, outputValue);
}
```

図 16　analogWrite 関数による PWM 出力

6.1.2　ファイルからの読み込み

正弦波などの波形を作るのではなく一般的な音楽を BGM として鳴らしたい場合、データをファイルの形で用意する必要がある。この最も簡単な方法はヘッダファイルに C 言語の配列の形で記述することである。

このためには Audacity を使うとよい。適当な音源ファイルを読み込み、モノラル音源化し、長さを短くし（長くても 30 秒程度）、8kHz、8bit 非負 PCM データとして書きだす。そのデータをテキスト化して C 言語の配列とする。この流れは例えば下記のページを参照すればよい。

- Arduino で STEM 教育 応用編：WAV 音源をデータ変換して圧電スピーカから出力する https://stemship.com/arduino-wav2code/

6.1.3　CPU コアを分ける

安定した周期で BGM を流し続けながら、画像描画やユーザの入力を行いたい。そのためには PICO マイコンが二つの CPU コアを持っていることを利用し、音再生を一つのコアに割り当てればよい。これは setup 関数と loop 関数の代わりに setup1 関数と loop1 関数を用いればよい。例えば loop1 の中身は次のようになる。

```
void loop1() {
  int soundArrayLength = sizeof(BGMData) /sizeof(BGMData[0]);
  unsigned char uc;
  for (int i = 0; i < soundArrayLength; i++) {
    uc = pgm_read_byte_near(&SuperMarioBGM[i]);
    analogWriteResolution(8); //Analog range set to 0-255
    analogWrite(buzzerPin, uc); // PWM 信号をブザーに送る
    delayMicroseconds(125);
  }
```

```
}
```

6.1.4 DA 変換用の IC とオーディオアンプを用いてスピーカを駆動する

圧電スピーカを用いた音の再生は、駆動可能な周波数領域が狭いことなどからどうしても高品位な音とはならない。このため、もしより高品位な音が必要であればオーディオスピーカないしイヤフォンを使う必要がある。Pico マイコンの疑似的なアナログ出力ではなくDA 変換器を用いたアナログ出力、オペアンプ等による増幅、小型のオーディオスピーカによる出力、を行う。部品としては例えば下記のようなものを使うことができる。これらを利用したい場合は相談すること。

- 12bit DA コンバータ MCP4921-E/P https://akizukidenshi.com/catalog/g/g118091/
- オーディオアンプ IC M2073 https://akizukidenshi.com/catalog/g/g117431/
- 基板取付用スピーカーユニット 8Ω0.08W UGSM23A
  https://akizukidenshi.com/catalog/g/g109797/

6.1.5 SD カードによる読み込みを可能とする

PICO（RP2040）はメモリ領域がそれほど大きくなく、音データを前述の方法で扱う場合 30 秒程度が限界となる。これに対処する必要がある場合、SD カードにデータを保存しておくことが考えられる。

今回使用している液晶ディスプレイは SD カードリーダを備えている（回路的には独立している）のでこれを利用すればよい。SPI 通信のポートを変更できるライブラリとして RP2040_SD を勧める。またこの SD カードリーダは MISO 端子をプルアップしないとうまく読み出すことができない。

- RP2040_SD https://github.com/khoih-prog/RP2040_SD/tree/main/examples
- プルアップについて https://yokahiyori.com/esp32-devkitc_ili9341-lcd_tft_espi/

7　振動出力

振動モータを動かす。トランジスタ(2SC1815)を用いる（図 17）。マイコンの適当なデジタル出力ピンに接続する。振動モータは直流によって振動を発生させることが出来るので、デジタル出力ピンを HIGH にすれば駆動できる。

ここではブラシレス振動モータ LBV10B-009 を用いる。この振動モータは極性があることに注意。

- LBV10B-009 https://akizukidenshi.com/catalog/g/g106787/

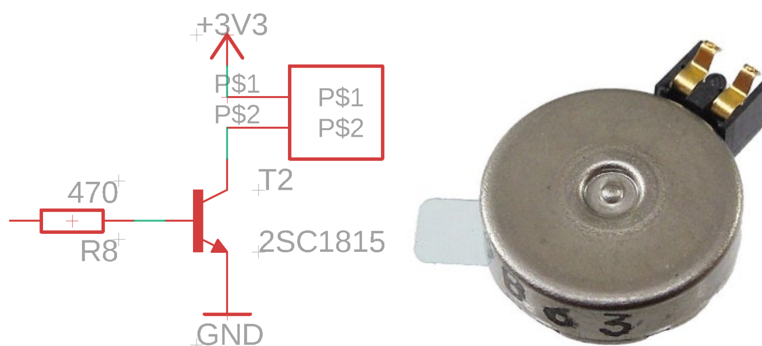


図 17　（左）トランジスタによる振動モータ駆動（右）ブラシレス振動モータ

8　モバイル化

このあたりのプロセスは本資料の範囲を超えるので個々に web 検索で自習する。情報工学工房の進捗によっては後日追記するかもしれない。

8.1 基板を外注する

回路が決定したら、KiCAD を用いて基板を設計する。

その後、JLCPCB（中国のメジャーな基板製造会社の一つ）に発注する。発注の際はデータの確認等を教員または TA と行う。

8.2 外観を整える

回路が届いたら、3D プリンタでゲーム機の外観を整える。

ここでは TinkerCAD を一応推奨するが何でもよい。研究室の 3D プリンタでプリントアウトする。プリントアウトの前にデータの確認等を教員または TA と行う。

8.3 バッテリー駆動（アドバンスト）

バッテリーを使用してモバイル化する。以下の方法が考えられる。

- リチウムイオンポリマー電池を用いる方法
- 電池（単四電池等）
- USB 接続のモバイルバッテリーを用いる方法

特にこだわりが無ければ USB 接続のモバイルバッテリーを PICO に接続すれば十分である。モバイルバッテリーは多くの場合長時間の使用で自動的に電源が切れることに注意。

また電池を用いる方法も問題ない。PICO マイコンは比較的低い電圧（1.8V 以上）でも内部に電圧変換機（DCDC コンバータ）を積んでおり、3.3V に昇圧されて動作する。つまり単四電池２本で動かしてよく、大きさが十分小さいといえる。ただし USB ケーブルを PICO マイコンに接続した際にこの電池に対して充電してしまう可能性があるためダイオードによってこれを防止する必要がある。

リチウムイオンポリマー電池を用いる方法は最も小型、スマートにできる。しかし充電を一歩誤ると膨張、火災の危険があり、工房の範囲では勧めない。

9　その他

ここではポータブルゲーム機としてはすぐには使用しないと思われるいくつかの拡張を紹介する。

9.1 RC サーボモータ駆動（PWM）

PWM 出力を利用して RC（ラジコン）サーボモータを動かす。

図 18 に示すように RC サーボモータを接続する。3 ピンのピンヘッダを用いる。

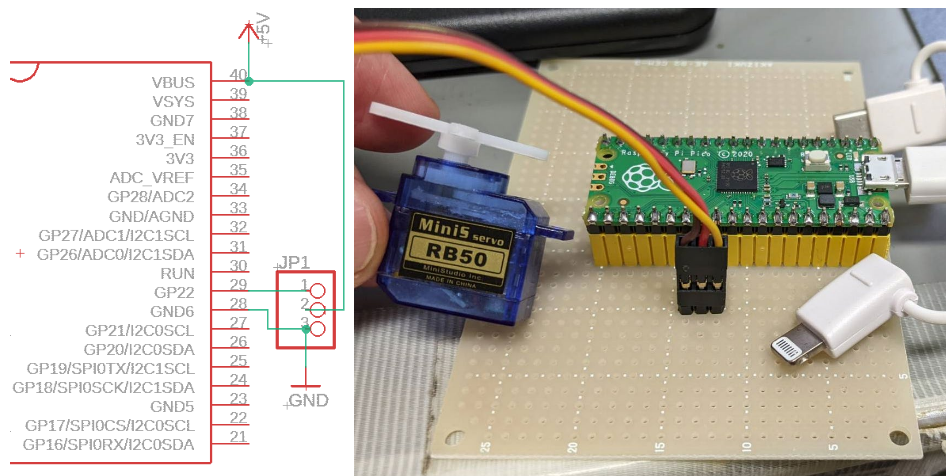


図 18 RC サーボ回路

RC サーボライブラリを使う。

https://docs.arduino.cc/learn/electronics/servo-motors

https://www.arduino.cc/reference/en/libraries/servo/

こちらのページの”SWEEP”というサンプルを使えば動かすことができる。ピン番号は適切に変更する。

9.2 無線通信

Raspberry Pi Pico W は無線通信を手軽に行うことができる。

https://sozorablog.com/raspberry-pi-pico-w-review/

これをうまく使うと、ポータブルゲーム機同士で対戦ゲームを作ることができるかもしれない。