ボイスチェンジャーの制作

1.製作目的

私は音楽を聴く事が趣味の一つにあるが、近年の音楽では楽器の音色にとどまらず声も音色を変えるのが一般化している。その中で音声を変えること自体に興味がわき、ボイスチェンジャーの制作を通して仕組みを理解するのが目的である。

2.理論。

2.1　音

音の正体は空気の振動であるので、空気の振動のデータを取りそのデータを完璧に空気の振動に変えることのできるならば、理論上完璧に同じ音を出すことも可能である。ボイスチェンジャーはデータを変換させるものだということも推測できる。

3.使用機器名

- ボイスチェンジャーキット(マイコンキットドットコム)　MK-137B　１つ
- 耐熱電子ワイヤー　AWG２４　２本
- ダイナミックスピーカー　１つ
- 電池ボックス　１つ
- アルカリ乾電池９ｖ　１つ

4.製作手順

4.1 ボイスチェンジャーキットのはんだづけ

初めに抵抗の種類の区別から取り掛かる。抵抗の色の種類、順番によって抵抗値が判別できるので慎重にチェックをした。　また、コンデンサーなどの正負も確認をしてから、はんだ付けに取り掛かった。(図１　回路図　一部)

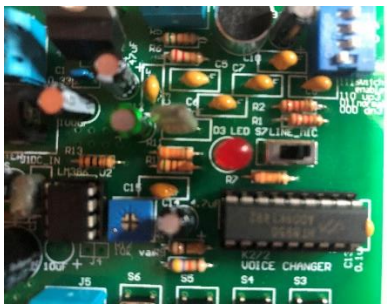


図１　回路図　一部

またコンデンサーマイクは利便性をよくするため、電子ワイヤーで繋げて制作することにした。

5.結果

まず初めに制作したキットでは雑音が多くなってしまいコンデンサーマイクを回路に直接つなげることにした。　（図２　コンデンサーマイク）

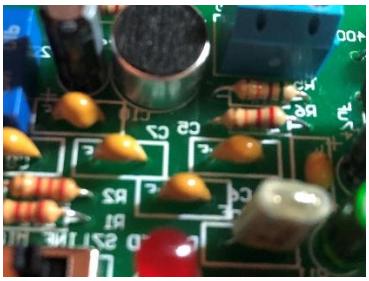


図２　コンデンサーマイク

コンデンサーマイクを直接繋げたので、雑音は少なくなった。

また、制作したキットを使って音の周波数を変えることができた。例えば女の人の声やロボットの合成音声などに変えることができた。しかしながら音色を変換することはできなかった。

6.考察

以上の制作を通して簡易的に周波数をかえて声の高低を変え違う音に変えることはできた。このことから音の高さを変えることは容易であることがわかる。また音の大きさの変換ボリュームを変えることによっても可能であることがわかる。しかしながら音色を変化させることはできなかった。

音は空気の振動なので、波の形も関係する。今回は周波数を変化させることと音の大きさを変えることにより、音の高低と音の大きさを変えることができた。以上のことから音色は波の形と関係があることが推測できる。

7.結論

今回の電子工作を通して音の要素とそれを変換する要素が確認でき、音色を変換する方法も学んでいきたいと思います。また最近の技術のデミックスでは楽器の音が合わさったものから楽器の音を抜き出す技術も作られているのでそれも勉強していきます。

参考文献

[1]マイコンキットドットコム　MK-137B　https://www.mycomkits.com/SHOP/MK-137B.html
閲覧日　２０２２年１１月１４日
[2]スガナミ楽器　音の三要素（音の大きさ、音程、音色）について知ろう！
https://www.suganami.com/shop/sound/column/so0002　閲覧日　２０２２年１１月１５日
[3]宮崎　純　宇宙一わかりやすい高校物理　力学・波動　２６７～２９３ページ　株式会社四国写研

ラズパイ監視カメラ

1.制作目的

初めての電子工作ということで簡単に作れてかつ日常生活でもしっかり役にたってくれるものを作りたいと思ったのがきっかけです。自分は以前からただ漠然とスマートホームにあこがれを持っていたので、それを実現するための第一歩ということで自分の部屋のセキュリティ向上を目的に制作に取り掛かりました。

2.理論

2.1 Raspberry Pi

Raspberry Pi は英ラズベリーパイ財団という組織がマイコンボードのことです。身の回りにある普通のパソコンと同じように OS を搭載しており、それを利用できる機能を持ちながらも 1 万円以下という低価格で購入できるということで非常に人気のコンピューターです。

2.2 Python

プログラミング言語の一種で、アプリケーションの開発、人工知能、データ解析など様々な用途に使われます。今回はこの Python を用いて実際にコードを書いていこうと思います。

3.使用機器名

・Raspberry Pi　4（以下ラズパイ）　1つ
・Raspberry Pi　Camera　Module　1つ
・メス-メスのジャンパー線　1つ（使えれば何でも可）
・HDMI ケーブル　1つ
・AC アダプター　　1つ（Type-C　3.0A 以上）
・焦電型赤外線モジュール　1つ

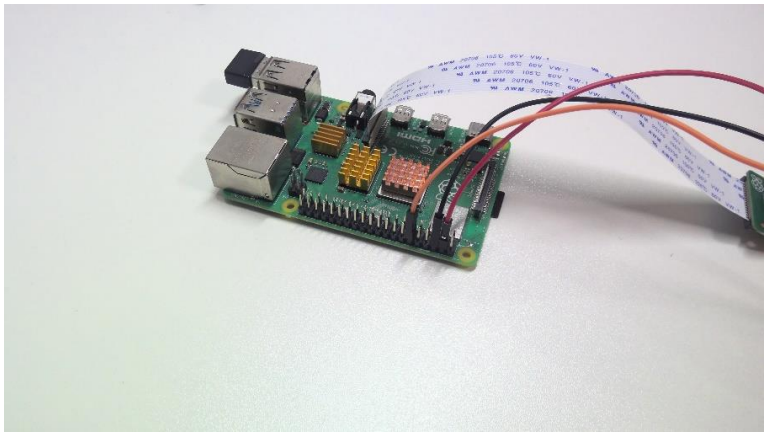
4.制作手順

4.1 カメラモジュールの取り付け

カメラモジュールをラズパイ本体と接続していきます。ラズパイの HDMI ポートの右側にある帯状の端子が差し込めるか所にカメラモジュールを差し込みます。

図 1　カメラモジュールをつけた直後

4.2　赤外線モジュールの取り付け



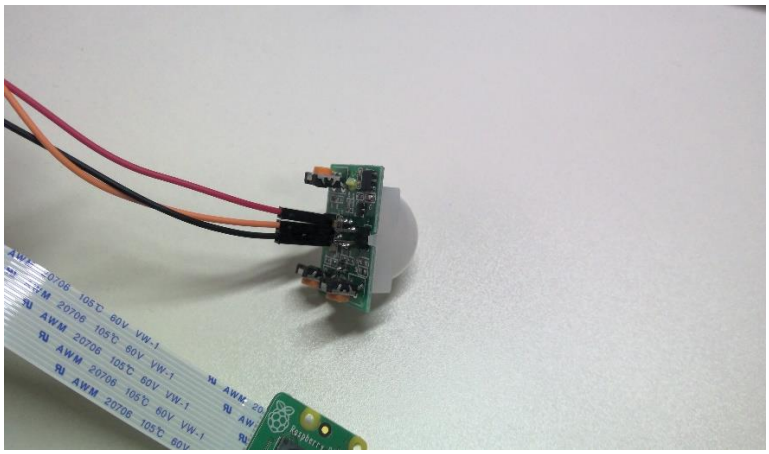


図 2，3　赤外線モジュールをつけた直後

ラズパイの GPIO ピンの 5V 電源に赤いジャンパー線を、１８番ピンにオレンジのジャンパー線を、グランドピンに黒いジャンパー線を接続します。

4.3　カメラモジュールを有効にする

写真のように接続しただけではまだラズパイ側には使用できる機器として認識されていないので、ラズパイの OS を起動してラズパイ本体側から設定しないといけません。まず起動したらターミナルを開いて、”sudo raspi-config"とコマンドを入力 Enter を押します。すると設定画面が表示されるのでそこで Interface Options ➡ Legacy Camera Enable/disable legacy camera support と進み、最後に Enter を押せば設定完了です。

4.4　カメラを使うために必要なライブラリのインストール

“sudo python -m pip install –upgrade”と入力し pip を最新のバージョンにします。次に”sudo pip3 install opencv-python==4.5.48 “と入力します。これは OpenCV をバージョン指定でインストールしています。そして”pip install -U numpy”と入力し numpy というライブラリを最新のものにします。更に”sudo apt update”と入力してパッケージリストを最新にして最後に”sudo apt install libatlas3-base”と入力して「libatlas3-base」パッケージをインストールすれば完了です。尚、写真を載せるとかなりの枚数になってしまうのでここでは割愛させていただきます。

4.5　実際にコードを書き込んでいく

冒頭でも言った通り、今回は Python を使って書いていこうと思います。一行ごとに解説すると日が暮れてしまうので実際のコードと大まかな流れを説明します。

以下、ソースコード⇩

```
import RPi.GPIO as GPIO
import cv2
import time
import datetime
import requests
import subprocess
```

```
#LINE にメッセージを送信するための関数
def send_message(Discovery_time):
    url = "https://notify-api.line.me/api/notify"
    token = "発行されたトークンをここに入力する"
    headers = {"Authorization" : "Bearer "+ token}
    files = {'imageFile': open("image.jpg", "rb")}
    message =  (Discovery_time,"An illegal intruder is invading！！（※任
意の文字を入力）")
    payload = {"message" :  message}
    r = requests.post(url, headers = headers, params=payload, files=files)
    #subprocess を使うための準備
    print(r.status_code)

#センサーを使うための準備
GPIO_PIN = 18

GPIO.setmode(GPIO.BCM)
GPIO.setup(GPIO_PIN,GPIO.IN)

while True:
    if(GPIO.input(GPIO_PIN) == GPIO.HIGH):
        #センサー検出時の処理
        print("1")
        #ラズパイに音声をしゃべらせる
        Subprocess.call("~~~~.mp3", shell=True)
        #検出時間の取得
        dt_now = datetime.datetime.now()
        Discovery_time = dt_now.strftime('%Y_%m_%d_%H_%M_%S')
        print(Discovery_time)

        #撮影した画像を保存する
        cap = cv2.VideoCapture(0)
        ret, frame = cap.read()
        cv2.imwrite("image.jpg", frame)
        cap.release()
```

```
            #LINE メッセージ送信
            send_message(Discovery_time)

            #0.1 秒待機（0.1 秒間隔で対象物を撮影）
            time.sleep( 0.1 )

        else:
            #センサー未検出時の処理
            print("0")
            time.sleep( 1 )

    GPIO.cleanup()
```

今回作ったラズパイ監視カメラのシステム全体の流れとしては

① 赤外線モジュールが人の動きを検知する
② 予め録音しておいた音声をラズパイ（につないだスピーカー）に喋らせて侵入者に警告する
③ カメラモジュールが一定間隔で対象物を撮影
④ 写真を保存しその画像データを指定した LINE アカウントに決まったメッセージとともに送信する

といった感じになっています。

メッセージを送信するには、”LINE Notify”というサービスを利用します。このサービスを使うには自身のアカウントでトークンを取得する必要があります。

人の動きを検知した瞬間音声によってラズパイが警告をしてくれるのですが、それには予め音声ファイル（.mp３）を用意しなければいけません。自分の声を録音しても良いのですが、それだと反応する度に自分の声が部屋に響き渡ることになり若干恥ずかしいので入力した文字列を様々な種類の声で読み上げてくれるサイト（音読さん https://ondoku3.com/ja/）があるのでそれを活用しました。

最後に気を付けることは、ラズパイで撮影した画像は撮影する度にその画像ファイルが次々と上書きされてしまうようになっているということです。これだと過去に撮影した写真が見れなくて困ります。そこで、ファイル名を秒単位の現在時刻にすることによって別のファイルと認識させることができデータを蓄積させることができます。Python なら簡単に現在時刻を取得できるのでこの辺りは非常に楽です。

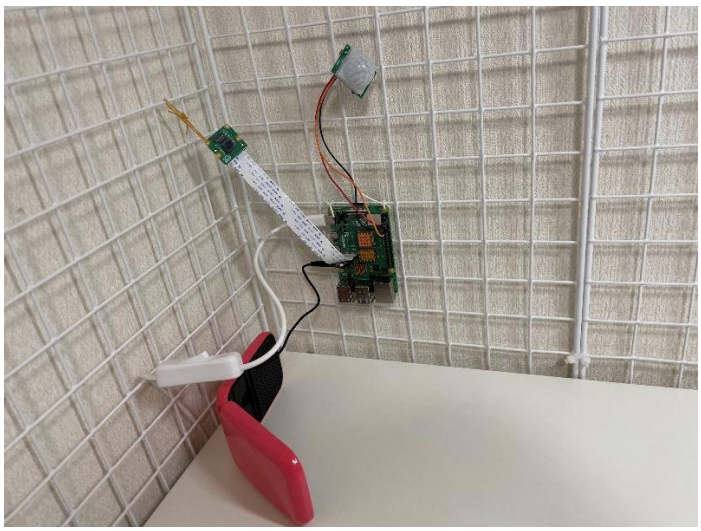


図４　実際に取り付けた様子

写真のように輪ゴムでワイヤーネットに括り付けて使用しています。

５.結果

自室に設置し稼働させてみると、実際に写真がラインに送られてくるようになりました。ただ、送信間隔がかなり速いので一度誰かが侵入しカメラに認識され続けている限り通知が鳴りやみませんでした（笑）

６．考察

あくまで今回使用したセンサーは熱源の”動き“に反応するので、カメラの前で何かしらアクションを起こさないと撮影がされないということが分かりました。ラズパイカメラはどうしても動画の撮影、送信ができないので写真を連続で送信することになったがもし可能だったとしたら通知が少なくて済み使い勝手がいいと思いました。ただメモリの容量がそれなりに必要になってくるから今の仕様でもいいのかなとも思いました。

７．結論

今回の電子工作によって確実に自分の部屋のセキュリティが向上したと言えると思います。これをきっかけに少しずつでも自分の思い描くスマートホームに近づくことができたらいいと思いました。

８．参考文献

みんなの Raspberry Pi 入門　第４版
ラズパイマガジン 2021 年夏号

RFID 式自動ドアロック

1.　制作目的

初めてラズパイを使って電子工作し監視カメラという日常で実際に役に立つものを作れたのが楽しかったのでまたもう一つ作品を作ってみようと思ったことがきっかけです。

2.　理論

2.1　RFID カードリーダー

RFID とは主に情報が書き込まれた IC タグと電波などでワイヤレス通信し情報の読み取りや書き込みをするシステムのことで、RFID カードリーダーとはその名の通りそれをするためのカードリーダーのことです。

2.2　サーボモーター

指示を出した通りに位置、速度、回転力などを正確に実現するサーボ機構に使用されるモーターのことです。

3.　使用機器名

・Raspberry Pi Pico(以下ラズパイピコ)　1つ
・ピンヘッダ　1つ
・TowerPro　サーボ　MG996R　1つ
・RFID　RC522　モジュール　1つ
・片面ガラス ユニバーサル基板　1つ
・スイッチング AC アダプター　5V/2A　1つ
・ブレッドボード用 DC ジャック DIP 化キット　1つ
・TUOFENG 22 awg Solid Wire（単線）　1つ
・USB ケーブル（Micro B）1つ

4.　制作手順

4.1　ラズパイピコにピンヘッダを取り付ける

ラズパイピコをブレッドボード上に固定したいのでそのためにまずピンヘッダとラズパイピコをはんだ付けします。こうすることによってラズパイピコの GPIO 端子とその他の配線がしやすくなることに加え、後でユニバーサル基板に回路を移植するときにもスムーズにいきます。(この時点ではまだ試作段階です)

4.2　サーボモーターを取り付ける

サーボモーターからは 3 本の端子が出ていて、黒を電源のマイナス側（グランド）に、オレンジを電源のプラス側に繋ぎ、残った黄色い線はラズパイピコと直接通信する制御線として使います。今回は写真のようにして GPIO16 番に接続しています。（この時点ではまだ試作段階です）

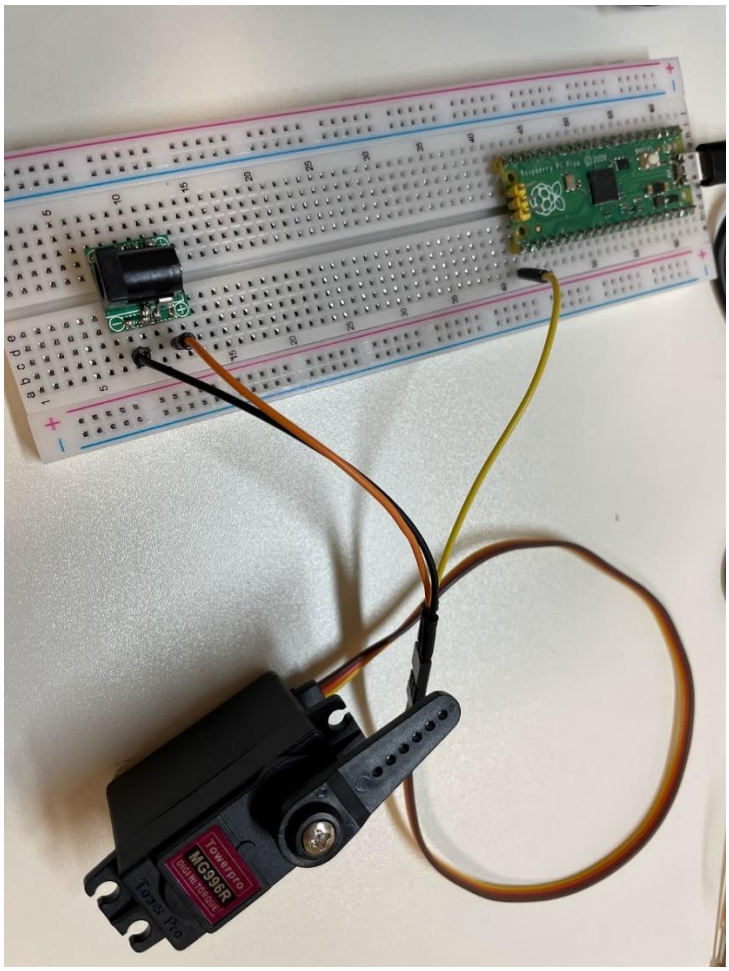


図 1　サーボモーターの実際の配線

4.3　RFID モジュールの取り付け

RFID モジュールはラズパイピコと通信するために 6 本の配線を施さなければいけません。具体的には以下のように接続します。

（RFID 側）VCC→→3.3V（ラズパイピコ側）

（RFID 側）RST→→GPIO 0 番（ラズパイピコ側）

（RFID 側）GND→→GND（ブレッドボードの GND）
（RFID 側）MISO→→GPIO ４番（ラズパイピコ側）
（RFID 側）MOSI→→GPIO ３番（ラズパイピコ側）
（RFID 側）SCK→→GPIO ２番（ラズパイピコ側）
（RFID 側）SDA→→GPIO １番（ラズパイピコ側）

尚、今回は RFID にある IRQ という名前の端子は使いません。
（この時点ではまだ試作段階です）

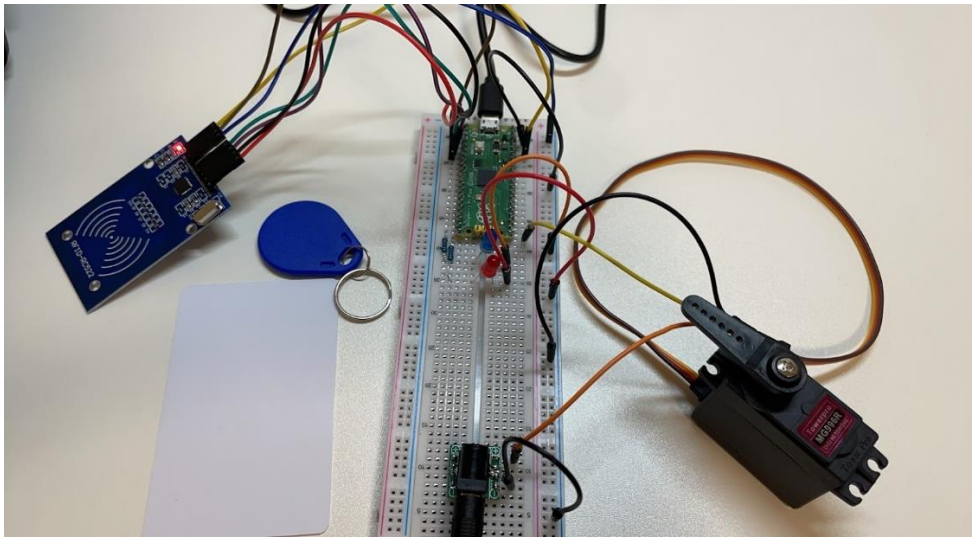


図２　サーボモーターと RFID モジュールを繋げた様子

写真では LED がついていますがこれはドアの鍵が開いたとき、閉まったときにそれぞれ別々の色の LED を光らせようとしていたのですが、最終的にこの機能は実装されることはありませんでした（泣）

4.4　回路がちゃんと動くかテストする

RFID モジュールにある特定のカードをかざすことによってサーボモーターが任意の角度まで回転し、また別の特定のカードをかざすとサーボモーターが元の位置に戻るという動きを作ったのですが、これが実際にブレッドボード上で思い通りに動くのかをしっかりと検証します。（ちゃんと動きました）そしてこの回路をそのままユニバーサル基板に移植していくという流れになります。

4.5　サーボモーターへの電源供給を考える

サーボモーターにはいろいろな種類があって、それぞれ力の強さ（回転力）、また

それに伴って必要になる電力などが異なってきます。今回はドアロックということでドアノブを物理的にサーボモーターに固定してもらう必要があるのですがそうするとそれなりにパワーが必要になってきます。(ちなみに使ったサーボモーターは 4.8V のとき 9.4kgf ㎝、6.0V のとき 11kgf ㎝です。) しかし、ラズパイピコから出力できるのは 3.3V までなので電力不足です。そこで色々調べたところ、それを回避するために 2 通りの方法があることがわかりました。1 つ目は何かしらの 5 V の電源を別にとり、トランジスタを使ってラズパイピコからの弱い電気信号を増幅させるという方法です。2 つ目は何かしらの 5V 電源を同じようにとり、専用のキットを用いて電力を供給するという方法です。前者は回路を組むことも難易度が高く、また増幅率なども計算しないといけないということからまだ経験が浅い自分にはできないと思ったのでおとなしく簡単な後者のやり方でやることにしました。

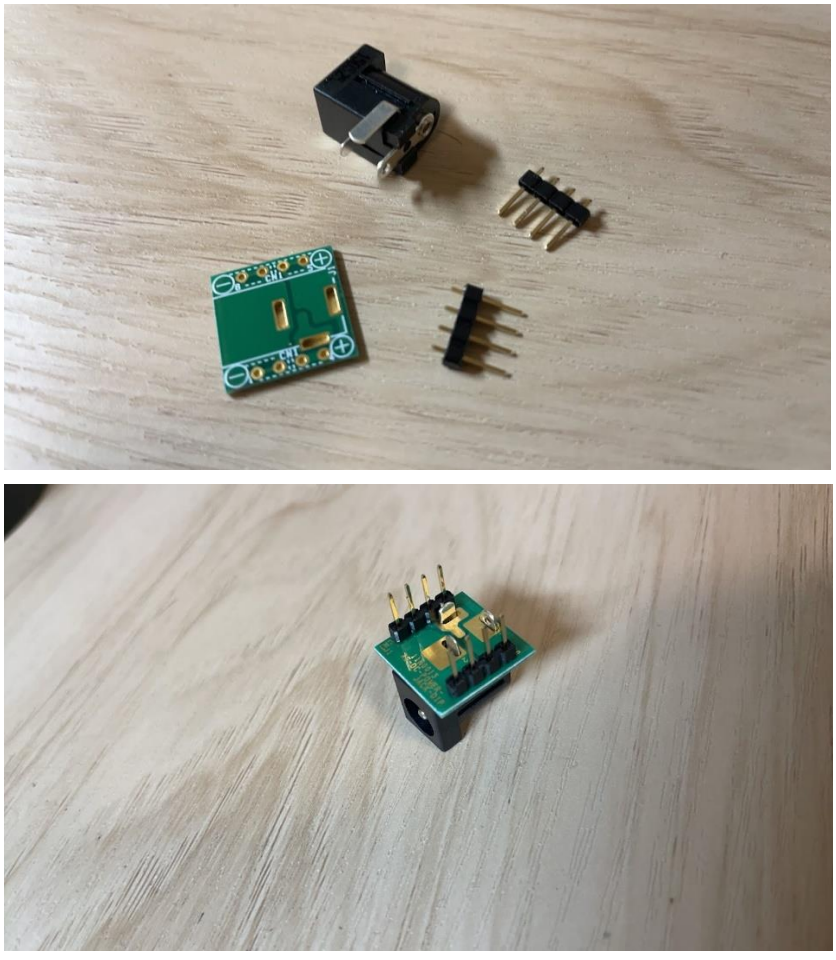


図 3，4　DC ジャック DIP 化キットの組み立てる前と後の様子

このキットを使えば、何も経由せずにコンセントから直接電力を取り出すことができるので便利です。そして最初に言った AC アダプタはちょうどこの端子に合うの

で安心して使うことができます。

4.6　ユニバーサル基板に移植する

回路の移植自体は特に問題なくできますが、今回大変なのは基板から RFID モジュール、そしてサーボモーターへの配線がかなり長いということです。装置全体のイメージは以下のような感じです。

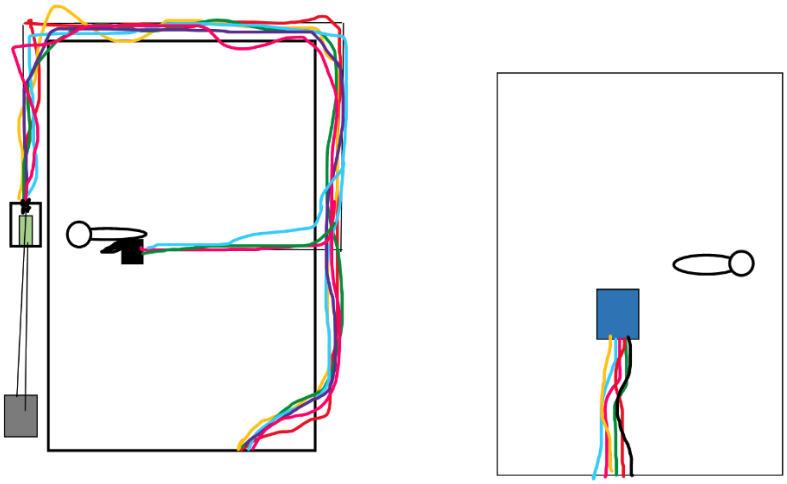


図 5，6　ドアの表側と裏側の様子

緑のやつがラズパイピコ、黒いやつがサーボモーター、グレーのやつがコンセント、青いやつが RFID モジュールです。このように、ドアの周りをほとんど一周するくらいの長さのジャンパー線が必要です。さらに、RFID モジュールに関しては 6 本も必要なのでなおさらです。そこで TUOFENG 22 awg Solid Wire（単線)を買って使うことにしました。本来このようなタイプのジャンパー線は被覆をワイヤーストリッパー等ではがす必要があるのですが値段が高いのでニッパーを使ってうまいことはがしました。

4.7　ジャンパー線と RFID モジュールのはんだ付け

正直ここが一番難しかったです。はんだ付けと聞いて思い浮かべるような一般的な状況とは違って、写真のように導線と導線をはんだ付けすることになったので上手に線と線を固定しながらちょうどいい量のはんだをつけるのがかなり大変でした。

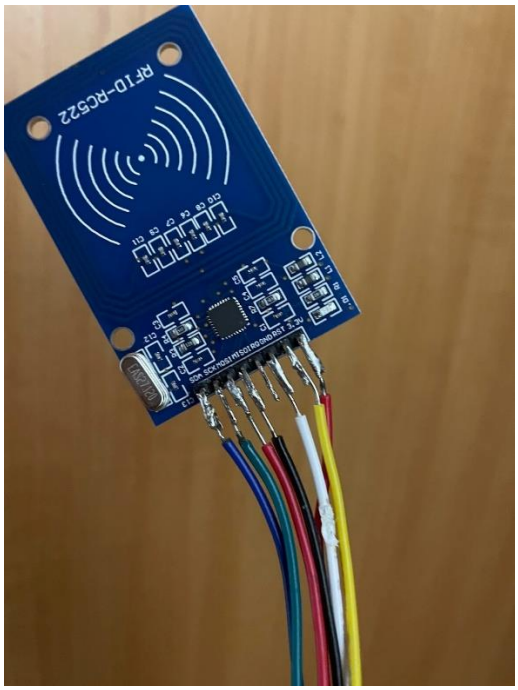


図７　地獄のはんだ付け

4.8　制御部分の制作

　　ユニバーサル基板上にはラズパイピコと DC ジャックとその他ジャンパー線をはんだ付けします。基盤は画鋲で壁に貼り付けています。

図８　基盤の様子

4.9　実際にコードを書き込んでいく

今回は Python でプログラミングしていきます。
以下ソースコード↓↓

```
from machine import PWM,Pin

from mfrc522 import MFRC522

import utime

 #GPIO 番号の設定

reader = MFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=1,rst=0)



print("RFID TAG を近づけてください")

print("")

#サーボモーターのデューティー比を計算し変数に入れる

mgservo = PWM(Pin(16))

mgservo.freq(50)

max_duty = 65535

angle_0 = 0.0725

angle_90 = 0.12
```

```
angle_90n = 0.025



while True:
    reader.init()
    (stat, tag_type) = reader.request(reader.REQIDL)
     if stat == reader.OK:
         (stat, uid) = reader.SelectTagSN()
      if stat == reader.OK:
           card = int.from_bytes(bytes(uid),"little",False)
  #598632465 番のカードが認識された時の処理
       if card == 598632465:
            print("Card ID: "+ str(card)+"PASS")
             utime.sleep(1)
              mgservo.duty_u16(int(max_duty*angle_0))
  #563350640 番のカードが認識された時の処理
       elif card == 563350640:
              print("Card ID: "+ str(card)+"PASS")
              utime.sleep(1)
```

```
            mgservo.duty_u16(int(max_duty*angle_90n))
#登録されたカードではなかったときの処理
    else:
        print("Card ID: "+ str(card)+"FAILED")
```

最初の import のところの mfrc522 というライブラリは下の GitHub のリンクより入手します。
（https://github.com/danjperron/micropython-mfrc522/blob/master/mfrc522.py）

また、コードに print とあり PASS や FAILED と書いてありますが、これは Thonny という統合開発環境上でコードを書いているときにターミナル上に表示させるためのものなので今回の動作に関しては関係ないです。

5. 結果

ドアの外側から片方のカードを RFID モジュールにかざすとサーボモーターが回転しドアがロックされました。そしてもう一方のカードをかざすとまたサーボモーターが回転してドアのロックが解除されました。

図 9　サーボモーターがドアをロックする仕組み

とても汚いかつ貧弱ですが、これから家族にもっとがっつり固定していいか相談し

ようと思います（笑）

6. 考察

今回一番思ったことは配線関係がとにかく面倒くさかったということです。プログラミングよりもどちらかというとそれぞれの部品についてや簡単ではありますが回路の組み方などについて勉強になりました。次回以降は今回使わなかったトランジスタを用いて作ってみようと思いました。そして今回の工作ではあくまでドアの外からのロック、そしてその解除が目的だったのですが今度は内側からもそれを操作できる機能を追加してみたりしたいです。

7. 結論

以前作った監視カメラに引き続き、今回の工作でさらに自分の部屋のセキュリティーが向上したと思います。これからも更に理想のスマートホームに近づけるように頑張りたいです。

8. 参考文献

みんなの Raspberry Pi 入門　第 4 版
ラズパイマガジン　2021 年夏号
RC522 RFID Reader Module with Raspberry Pi Pico ⇩⇩
(https://microcontrollerslab.com/raspberry-pi-pico-rfid-rc522-micropython/)

Arduino Uno ・ NanoEvery による I2C 通信について

1　はじめに

Arduino でセンサなどを利用する際のデジタル通信方式として I^2C がある。これは、2 本と少ない通信線で複数の機器を接続することができる。自分は最近そういったセンサである BME280 を使うことがあり、I^2C について知る必要があったので調べたことをまとめた。まず I^2C について調べ、Arduino の Wire ライブラリによる通信を行った。次に Wire ライブラリを使わないプログラムを書けるか試したが失敗した。また、BME280 との通信は時間がなかったので出来なかった。次回やるかもしれない。

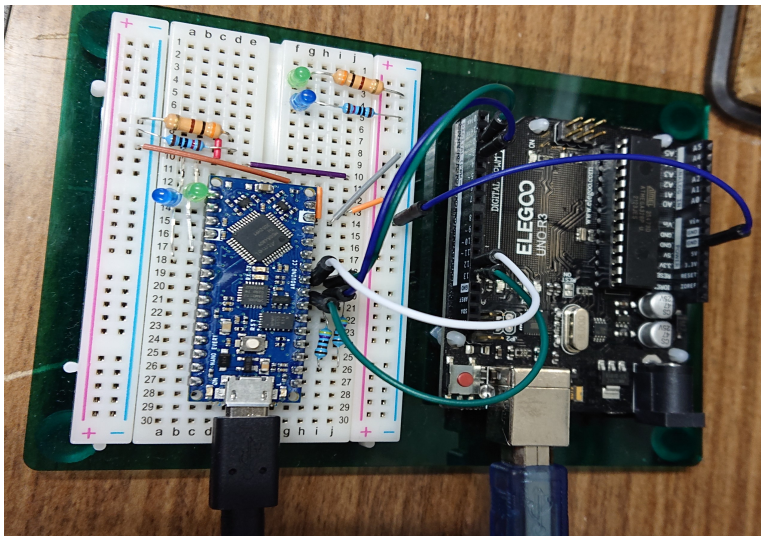


図 1: 使用した Arduino

2　I^2C について

I^2C 通信は一つのマスタと複数個のスレーブデバイスによって行われる。マスタ・スレーブ間はクロック線 (SCL) とデータ線 (SDA) の 2 本で通信を行う。2 本はそれぞれプルアップされていて、レベルを LOW に下げることでビットを切り替える。電圧は機器による。クロック信号は常にマスタから出力され、通信のタイミングを決める。

通信の流れは、まずマスタによる開始条件という操作があり、次に送信先のスレーブのアドレスが送られる。スレーブはそれぞれアドレスを受信し、自分のアドレスと一致したらアクノレッジと言われる受信確認のビットを送信する。一致しなかったスレーブはそれ以降通信を無視する。その後マスタからデータビットが送信される。

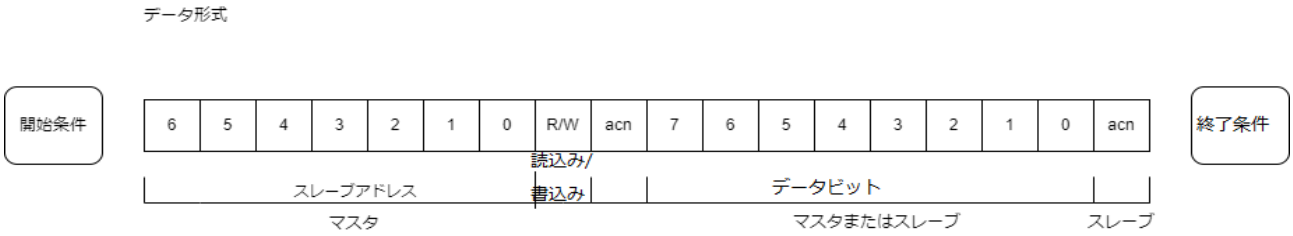


図 2: データ形式

3　Wire ライブラリの利用

ここでは、Wire ライブラリについて説明し、実際に ArduinoUno,NanoEvery 間で簡単な通信を行う。

3.1　Wire ライブラリの関数

Wire ライブラリの関数で今回使用するもの・重要だと思ったものについてのメモ書き

引数・戻り値は今回使わなければ書かない。（ないとは限らない）時間がないので一部省略　正確でもないのでリファレンス見てください

Wire.begin()

[引数:int] 指定しなければマスタとして開始　スレーブの場合はアドレス（7 ビット、一部使用不可？）

[戻り値:]

Wire での I^2C 通信を開始する関数。これでクロックが始まるかはわからない（確認していない）。終了する関数はない？

Wire.setClock()

[引数:int] 通信速度　標準:100kHz, 高速:400kHz, 一部のマイコンで低速 10kHz(らしい) それ以外も指定できるが変化しない？(もしくは計測自体できてなかった)

[戻り値: なし？]

I^2C 通信の速度を設定する関数　クロックはマスタが管理するのでそこで指定する

Wire.beginTransmission()

[引数:int] アドレス

[戻り値:]

マスタがスレーブに対して送信を始める際に呼ぶ。この時点では開始条件は送られてなさそう

Wire.write()

Wire.endTransmission()

stop の false はたぶん反復開始

Wire.requestFrom()

Wire.available()

3.2　通信の構成

Uno と Every をそれぞれマスタ、スレーブとして Every 内のデータを Uno が読むようにした。センサから値を読み込む場合を考えて、Every には 2 つのレジスタがあるとし、それぞれアドレスで指定するようにした。実際にこの書き方で動作するかは分からない。また、実際の I^2C 通信では反復開始という方法で、レジスタアドレスを書き込んだ直後に読出しの通信を始めるらしい。

図のように配線した。SDA,SCL どうしを接続し、グラウンドは共通、それぞれ 4.9k Ωでプルアップした。どちらも 5V で動作するのでそのまま接続したが、例えば BME280 などのセンサは電圧が違う場合があるので電圧レベル変換 IC を利用する必要がある。今回は BME280 は時間がないのでやらなかった。

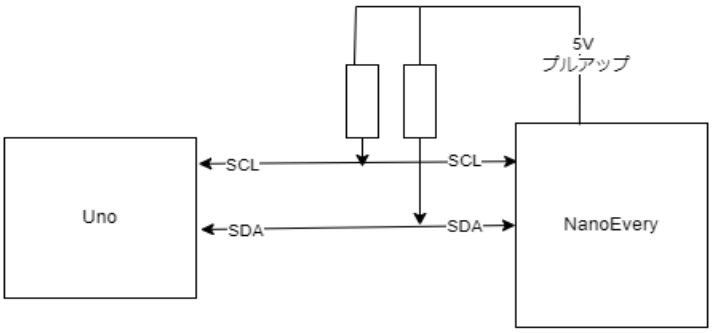


図 3: 配線

3.3 プログラム

- Uno 側

```
--------------------------------------------------------------------------------------------
#include <Wire.h>
/*
PC から 1 文字 (1 バイト) の命令をシリアル通信で受け取り、Nano にデータを要求する
*/

const uint8_t NADR = 0x04;
//スレーブアドレスとする

const uint8_t MADR1 = 0x20;
const uint8_t MADR2 = 0x21;

void setup(){
    Serial.begin(9600);//pc との通信
    Wire.begin();//マスタとして開始
}

void loop(){
    char cmd = SerChk();
    if(cmd == 'r'){//指定したスレーブアドレスにバイトを要求する
        Wire.requestFrom(NADR,1);//アドレス、バイト数
        /*
        *この関数には 3 つ目のパラメータ"stop"を付加させられる。
        true(デフォルト) では stop メッセージを送信することで I2C バスを開放するが、
        false にすると通信を継続しほかのマスタのリクエストを防げる
        (http://www.musashinodenpa.com/arduino/ref/index.php?f=1&pos=363)
        今回は 2 台しかないので利用しない
        */
    }

    if(cmd == 't'){//指定したスレーブアドレスのレジスタアドレスからデータを受信する
    //実際のセンサなどでは複数個のレジスタがデータを持つことがある
        Wire.beginTransmission(NADR);
        Wire.write(MADR1);
        /**
        write() の時点ではバイトを送信キューに入れるだけで、実際に送信するのは endTransmission()
        **/
        Wire.endTransmission(false);
        /**
```

```
        送信終了
        これも requestFrom() と同様"stop"パラメータを付加させられる。(デフォルトは true)
        **/
        Wire.requestFrom(NADR,1);
        /*
         * データを要求　マスタからのリクエストによりスレーブはデータを送る
         */
    }
    if(cmd == 's'){
      Wire.beginTransmission(NADR);
      Wire.write(MADR2);
      Wire.endTransmission();
      Wire.requestFrom(NADR,1);
    }

    //データを受け取っているか確認
    while(Wire.available()){
        uint8_t c = Wire.read();
        Serial.println(c);
    }
}

/*
PC から 1 文字のコマンドを読み込む
*/
char SerChk(){
    char cmd = 0;
    if(Serial.available()){
        cmd = Serial.read();
        return cmd;
    }
    return '0';
}
```

- NanoEvery 側

```
#include <Wire.h>
/*
  このスレーブデバイス (アドレス:NADR) は 2 つのメモリ (アドレス:NAD1,NAD2) と
  それぞれ 1 バイトのデータを持つとする。
  マスタはメモリアドレスを送信し、スレーブはそれに対応するデータを送る。
*/

const uint8_t NADR = 0x04;//4(int) //スレーブアドレス
const uint8_t MADR1 = 0x20;//32(int)  //レジスタアドレス 1
const uint8_t MADR2 = 0x21;//33(int)  //2
const uint8_t DATA0 = 0xFF;//255(int) レジスタアドレスを指定しない場合
const uint8_t DATA1 = 0x37;//55(int)  DATA を MADR1 の持つデータの 1 バイトとする
const uint8_t DATA2 = 0xa4;//164(int) MADR2
int radr;

void setup() {
  Wire.begin(NADR);//スレーブアドレス 4(int) で開始
  Wire.onRequest(requestEvent);//データを要求されたら呼ぶ関数を登録する
  Wire.onReceive(receivedEvent);
}
```

```
void loop() {
}
void receivedEvent(int bnum){
  radr = Wire.read();
}

void requestEvent(int bnum) { //int 型の引数 (受け取ったバイト数) を一つとる
  //スレーブから送る場合は Wire.beginTransmission() は不要
  //https://docs.arduino.cc/learn/communication/wire

  switch (radr) {
    case MADR1:
      Wire.write(DATA1);
      break;
    case MADR2:
      Wire.write(DATA2);
      break;
    default:
      Wire.write(DATA0);
      break;
  }
  radr = 0;
}
```

- 結果

```
>> r,t,s
255
55
164
```

4 I²C スレーブ機能の作成

Arduino を Wire ライブラリを使わないプログラムで I²C スレーブとして動作させられるか試した。NanoEvery をマスタとして 1 バイト読む命令を一定時間ごとに送信し、Uno をスレーブとしてデータを送ることを目標としたが、結果は、アドレスをうまく読み込めず失敗した。

4.1 クロックの確認

まず、Arduino が digitalRead() でクロック信号 (SCL) を読み込めることを図の配線とプログラムで確認した。(この時と次のアドレス受信 1 では digitalRead() の繰り返しで信号を記録していたが、割り込み機能を利用したほうが簡単で安定していると思う。アドレス受信 2 では割り込みを利用した。思っていたより面倒ではないのでこちらでやっていればよかった)

Arduino の資料によれば I2C の通信速度は 100000Hz(10kHz) が標準らしい。Uno と Every で低速モードは利用できなそうだったので標準モードとする。1 秒間クロック信号が切り替わる回数を数えて（割り込みの LISING でよかった）出力させた。数回実行させて、どれも同じような数値になった。10kHz に対して近い値になっているが、正確に測れているか分からない。

プログラム：

- Uno 側

```
#include <Wire.h>

const unsigned long STOP = 10000;//us
unsigned long STARTTIME;
unsigned long count = 0;

unsigned long STOPTIME = 0;
unsigned long DTIME = 0;
unsigned long DCOUNT = 0;

bool isH = true;
bool isHtemp = true;

bool isEnd = false;

void setup(){
  Serial.begin(115200);
  Serial.println("SStart");
  Wire.begin(0x04);
  pinMode(12,INPUT_PULLUP);
  STARTTIME = micros();
}

void loop(){
  isHtemp = digitalRead(12);
  if(isHtemp != isH){
    count++;
    isH = isHtemp;
  }
  if(micros() - STARTTIME > 1000000){
    STOPTIME = micros();
    DTIME = STOPTIME - STARTTIME;
    DCOUNT = count;
    digitalWrite(4,HIGH);
    if(count > 100) digitalWrite(5,HIGH);

    if(isEnd == false){
      Serial.print("us:");
      Serial.println(DTIME);
      Serial.println(DCOUNT);
    }
    isEnd = true;

  }
}
```

- NanoEvery 側

```
#include <Wire.h>
void setup() {
  Wire.begin();
  Wire.setClock(100000);
```

```
  Wire.setTimeout(0);

  Wire.beginTransmission(0x04);
  //Wire.write('a');
  Wire.endTransmission(false);
}

void loop() {
}
```

---

item 実行結果

```
micros() - STARTTIME > 1000000
Serial.begin(115200);
Wire.setClock(100000);

SStart
us:1000016
31156
SStart
us:1000008
31360
SStart
us:1000008
31272
```

4.2 アドレス受信 1(digitalRead())

クロックのカウントは出来ていたとして、マスタから送信されたアドレスデータを読み込めるか試した。while() と digitalRead() でクロックが HIGH になった時の SDA の状態を記録するプログラムを書いたが、正しいアドレスは出力されなかった。クロックの幅に対して読み込みに時間がかかったと考え、次ではより速そうな割り込み機能を使ってみた。

プログラム：データの送信まで書いていて長いので省略

4.3 アドレス受信 2(割り込み)

Arduino の割り込み機能により、開始条件と SCL が HIGH になったタイミングの SDA の状態を 8 ビット分記録した。アクノレッジ以下は考えなかった。結果は、正しいアドレスである 19 も出力されたが、関係ない値も出力された。同じ値が繰り返し現れるので、読み込みのタイミングがずれているのかもしれない。この状態では正常にデータを送信できなそうであり、また時間がなかったので今回のレポートではここまでとする。

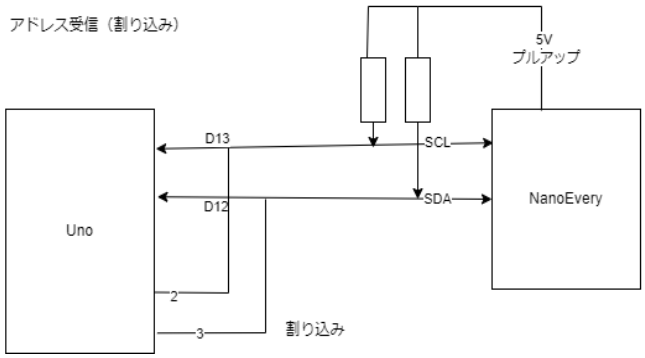


図 4: 配線 (一部)

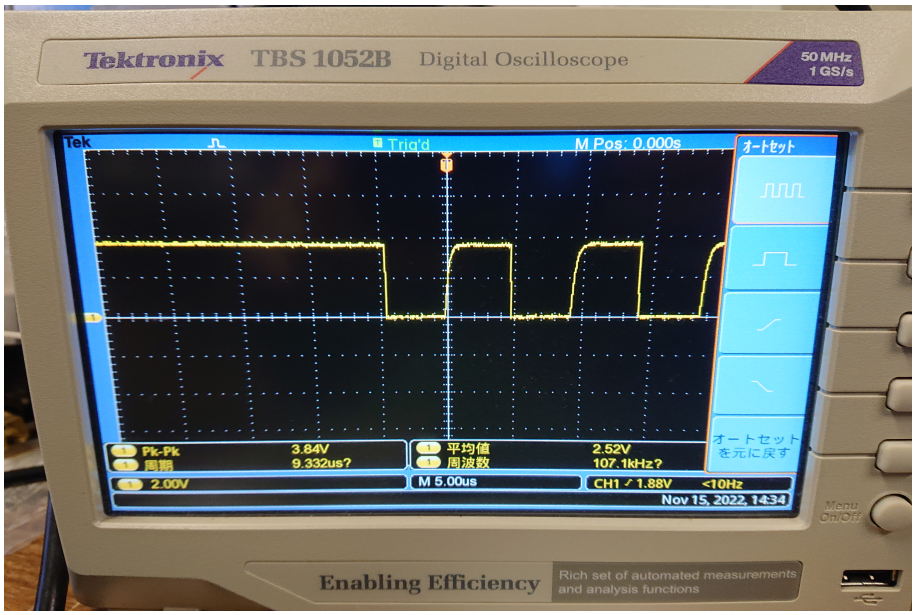


図 5: クロック波形

プログラム：

- Uno 側

```
#include <Wire.h>

const int SCLpin = 12;//クロック 常に INPUT
const int SDApin = 13;

int rcvdAdrs = 0;
volatile int adrs[8] = {0, 0, 0, 0, 0, 0, 0, 0};
volatile int nowAdrsByte = 0;

bool isstartcnd = false;

void setup() {
  pinMode(SCLpin, INPUT_PULLUP);
  pinMode(SDApin, INPUT_PULLUP);
  pinMode(2,INPUT);
```

```
  pinMode(3,INPUT);
  attachInterrupt(0, readbit, RISING);
  attachInterrupt(1, readcond, FALLING);
  Serial.begin(9600);
  Serial.println("ustart");

}

void loop() {
  //開始条件判定
  if (isstartcnd) {
    while (nowAdrsByte < 8) {
    }
    detachInterrupt(0);

    for (int i = 0; i < 7; i++) {
      if (adrs[i] != 0) adrs[i] = 1;
    }
    rcvdAdrs = adrs[0] * 64 + adrs[1] * 32 + adrs[2] * 16
     + adrs[3] * 8 + adrs[4] * 4 + adrs[5] * 2 + adrs[6];
    if (rcvdAdrs != 0) {
      //digitalWrite(5, HIGH); //LED 表示
      Serial.println(rcvdAdrs);
      //detachInterrupt(1);
      isstartcnd = false;
      attachInterrupt(0, readbit, RISING);
      nowAdrsByte = 0;

    }
  }

  //通信終了
}

void readbit() {
  if (isstartcnd) {
    adrs[nowAdrsByte] = digitalRead(SDApin);
    nowAdrsByte++;
  }
}

void readcond() {
  if (digitalRead(SCLpin)) {
    isstartcnd = true;
  }
}
```

- NanoEvery 側

```
#include <Wire.h>
int res = 0;
unsigned long starttime = 0;
unsigned long ntime = 0;
void setup() {
  //Serial.begin(9600);
  //Serial.println("start");
  Wire.begin();
```

```
  Wire.setClock(100000);
  //Wire.setTimeout(500000);
  delay(1000);
  starttime = millis();

  //Wire.beginTransmission(0xA4);//164
  //res = Wire.requestFrom(64, 1);
  //Wire.write('a');
  //Wire.endTransmission(false);
  res = Wire.requestFrom(19, 1);
}

void loop() {

  //Serial.println(res);
  //delay(1000);
  ntime = millis();

  if(ntime - starttime > 1000){
    starttime = ntime;
  Wire.requestFrom(19, 1);
  }
}
```

- 実行結果

```
ustart
19
7
28
39
19
19
39
```

参考文献

[1] Arduino Reference - Arduino Reference

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

公式の言語リファレンス

[2] Wire - Arduino Reference

https://www.arduino.cc/reference/en/language/functions/communication/wire/

[3] Arduino 日本語リファレンス

http://www.musashinodenpa.com/arduino/ref/index.php

公式リファレンスの日本語版

[4] Connecting Two Nano Every Boards Through I2C

https://docs.arduino.cc/tutorials/nano-every/i2c

NanoEvery2 台の間での I2C 通信についての解説

[5] A Guide to Arduino & the I2C Protocol (Two Wire)
https://docs.arduino.cc/learn/communication/wire
Wire ライブラリについてのいろいろな例

[6] I^2C バス仕様およびユーザーマニュアル

[7] I2C プライマ：I2C とは（パート 1） — アナログ・デバイセズ
https://www.analog.com/jp/technical-articles/i2c-primer-what-is-i2c-part-1.html

[8] I2C プライマ、SMBus、PMBus の仕様を学ぶ — アナログ・デバイセズ
https://www.analog.com/jp/analog-dialogue/articles/
i2c-communication-protocol-understanding-i2c-primer-pmbus-and-smbus.html

電流帰還アンプ

1.製作目的

電子工作に触れ、はんだ付けの方法や工作過程を理解するため。また、電流帰還アンプの仕組みを理解するため。

2.理論

一般的な電圧駆動アンプでスピーカーを駆動させると、入力に対して電圧が増幅されるように出力される。この時にスピーカーのインピーダンスの上昇によって出力電力が変動し、スピーカーからの音量にムラが生じてしまう。つまり、低音不足または高音不足という形で現れる。しかし、今回制作する電流帰還アンプはインピーダンス変動に左右されないため、すべての周波数で入力信号に対し適正な出力電力を出すことができる。そのため、低音や高音域が忠実に再現され、スピーカー本来の性能を発揮させることができる。

〈仕様〉
・入力端子：RCA（L/R）
・出力端子：ターミナル
・出力電力：0.5W+0.5W（4Ω~16Ω）
・電源　　：DC15V

3.使用機器名

・抵抗（茶緑茶金）/150Ω　　2つ
・抵抗（緑茶赤金）/5.1kΩ　　2つ
・抵抗（茶黒金金）/1Ω　　2つ
・抵抗（黄紫黒金）/47Ω　　2つ
・抵抗（橙黒金金）/3Ω/1W　　2つ
・抵抗（茶青橙銀）/16kΩ　　1つ
・抵抗（茶緑赤金）/1.5kΩ　　1つ
・抵抗（茶黒黄金）/100kΩ　　1つ
・抵抗（灰赤金金）/8.2Ω/1W　　2つ
・抵抗（茶黒橙金）/10kΩ　　2つ
・抵抗（茶赤赤金）/1.2kΩ　　1つ

・電解コンデンサ /1μF　　1つ
・電解コンデンサ /100μF　　1つ
・電解コンデンサ /470μF　　4つ
・電解コンデンサ /1000μF　　2つ
・フィルムコンデンサ /0.22μF　　2つ
・コイル /1mH0.6A　　2つ
・リレー　　1つ
・ダイオード　　1つ
・トランジスタ　　3つ
・三端子レギュレータ /7812TV　　1つ
・パワーアンプ /NJM2073　　1つ
・RCA ジャック（赤・白）　　各1つ
・DC ジャック　　1つ
・青 LED（角型）　　1つ
・スイッチ付きボリューム　　1つ
・ボリュームつまみ　　1つ
・金属ターミナル　　4セット
・IC ソケット　　1つ
・スペーサー（金属）　　4つ
・ナベネジ　　4つ

図1　使用した電子部品

4.製作手順

4.1　抵抗のはんだ付けを行った。
4.2　ダイオード、IC ソケット、トランジスタのはんだ付けを行った。
4.3　LED ライト、リレーを取り付けた。
4.4　コンデンサ、レギュレーター、コイルを取り付けた。
4.5　DC、RCA ジャックを取り付けた。
4.6　スピーカーターミナルを取り付けた。
4.7　スピーカーターミナルのタブと基盤を配線した。
4.8　ボリュームとパワーアンプを取り付けた。
4.9　ボリュームつまみとスタビライザを取り付けた。

図２　組み立てた電流帰還アンプ

5.結果

電源を入れると LED ライトが光り、通電していることが確認できた。

図 3　LED ライトが点灯し、通電している様子

6.考察

LED ライトが点灯したことでアンプを使うことができると分かった。しかし、今回は接続先のスピーカーを用意することができなかったため、使用感が分からないという結果になってしまった。

LED ライトの光がかなり強かったと感じたため、もう少し優しい光のものに変えることで使い勝手がよくなると考える。

7.結論

はんだ付け、電流帰還アンプの仕組みを理解できた。

参考文献

〔1〕　Bit Trade One　「電流帰還アンプ」https://bit-trade-one.co.jp/product/assemblydisk/ad00026/　閲覧日：2022 年 10 月 19 日