ラズパイに RTCを接続する

ラズパイは、名刺サイズの小さなマイコン基板で右の画像は Raspberry Pi 3B です。

CPUは ARM Cortex-A53(　4コア、1.2GHz　)で メモリ 1GByteです。で、Raspbian OSという Linuxを マイクロSDカードに書き込んで起動する事が出来ます。

入出力には、USB 2.0ポート4個、HDMI映像出力、ネットワーク 10/100Mbps イーサネット、カメラコネクタ、GPIOという汎用I/Oポート付きです。

　このラズパイの事を初めて知った時は、パソコン並みの機能が、この小さな基板で実現出来る事に驚きました。 で、今回は GPIOの端子の一部を I2Cインタフェースとして設定して I2Cの RTCを接続して時刻の読み書きが出来るようにします。

OSを 再インストール

私の ラズパイ 3Bは、3年前の Raspbian OS なので、最新版のOSを入れ直す事にしました。

で、ダウンロードサイトを探したら Raspberry Pi Imager なるソフトがあって、何と Raspbian OS を ダウンロードして、そのまま SDカードに Raspbian OS を書き込んでくれる便利なソフトでした。

過去やった時は、SDカードを 特殊なフォーマットで初期化したり、やや面倒でしたが、これは楽です。 OSを 書き込んだ SDカードを、ラズパイに挿入して 電源ONすれば、あっさり Raspbian OS が 起動しました。

上の画像は、Raspberry Pi Imager の フォーム画像です。 OS と ストレージ(SDカード)を、選択して 書き込みボタンをクリックするだけです。
Raspberry Pi Imager 現在のバージョンは v1.7.2 です。(2022-05-11 現在)

Raspbian OS上に、開発環境を構築

最初の起動時の初期設定が終わったら、画面左上の 黒いアイコン LXTerminal を クリックします。 タ―ミナルの ウィンドウが開いたら順次、コマンドを入力して行きますが、その前に、イーサネットコネクタに LANケーブルを差し込んで下さい。 まずは、OSインストール直後の、アップデート確認です。

① sudo apt update

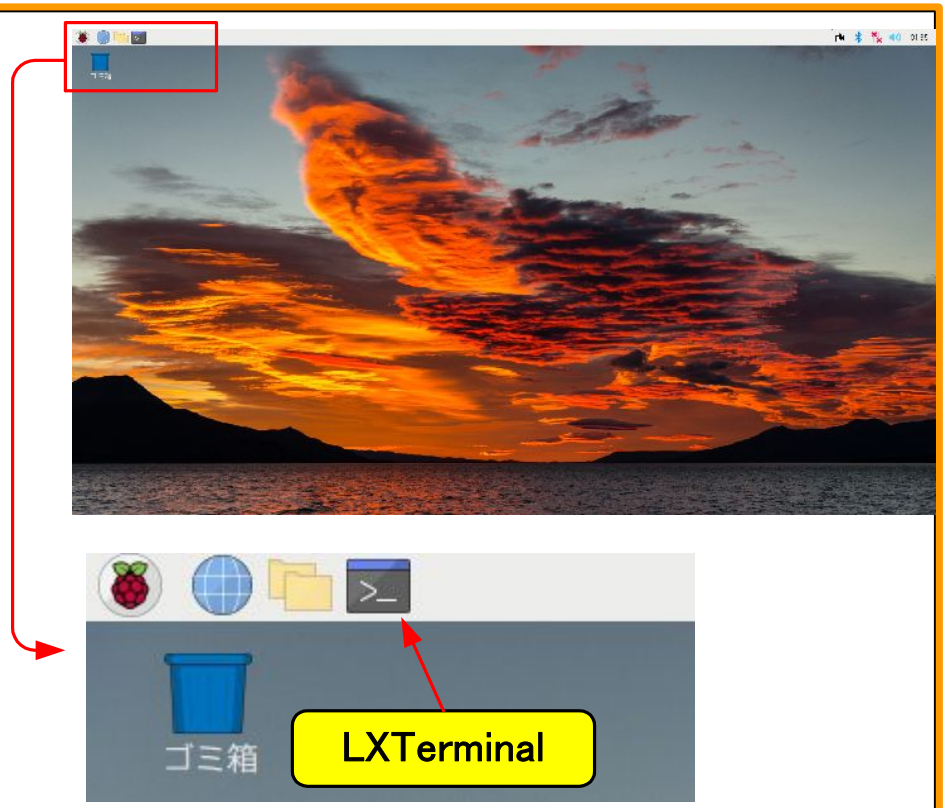
② sudo apt upgrade -y

I2Cを使うために wiringPi をインストールします。 まず、git の インストール

③ sudo apt-get install git-core

次に、分かりやすい場所にディレクトリを作成します。

④ mkdir wiringPi-source



作成したディレクトリに、移動します。

⑤ cd wiringPi-source

次のコマンドは、やや長いので次のページに記述します。

gitを 使い wiringPi を ダウンロードします。

⑥ sudo git clone https://github.com/wiringPi/wiringPi.git

ダウンロードした wiringPiのディレクトリに移ります。

⑦ cd wiringPi

wiringPi ディレクトリ内で、build ファイルを実行します。

⑧ sudo ./build

wiringPi ディレクトリの内容を確認します。

⑨ ls

ファイル、サブディレクトリの一覧が表示されます。

```
take@raspberrypi:/wiringPi-source/WiringPi $ ls
COPYING.LESSER  README.md  debian           examples    pins       wiringPi
INSTALL         VERSION    debian-template  gpio        update     wiringPiD
People          build      devLib           newVersion  version.h
take@raspberrypi:/wiringPi-source/WiringPi $
```

wiringPi と wiringPiD というフォルダが囲ってありますが、

元の資料通りになる事は確認しましたが、正直 何を意味するのかが

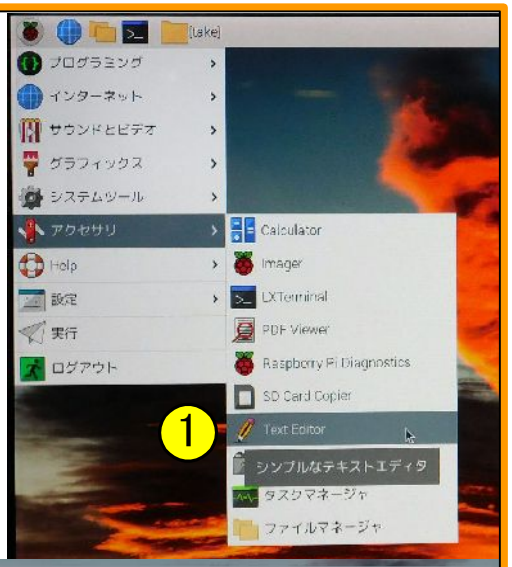
いまいち分かりません。 以上で必要なライブラリは、

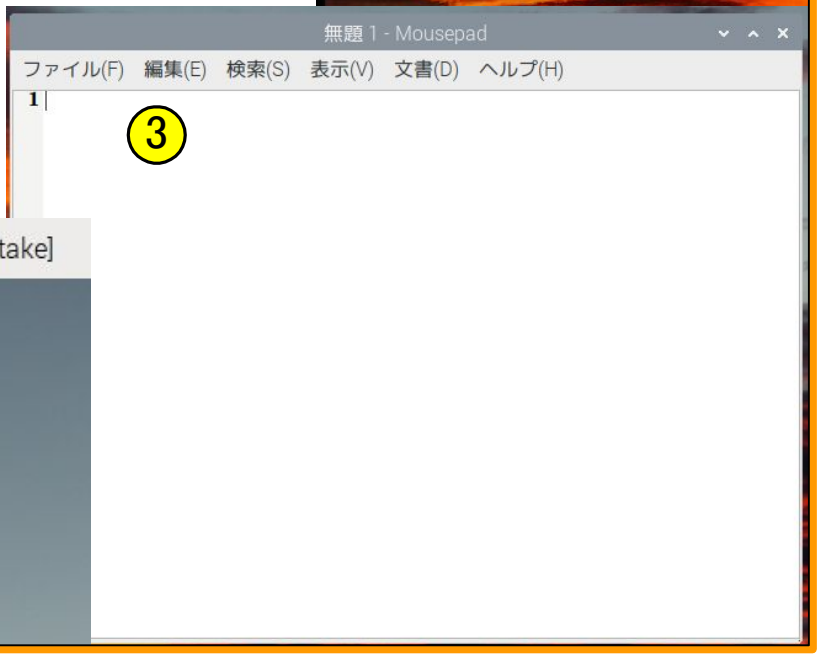
ラズベリーパイに用意されました。 と 資料には 書いてあります。

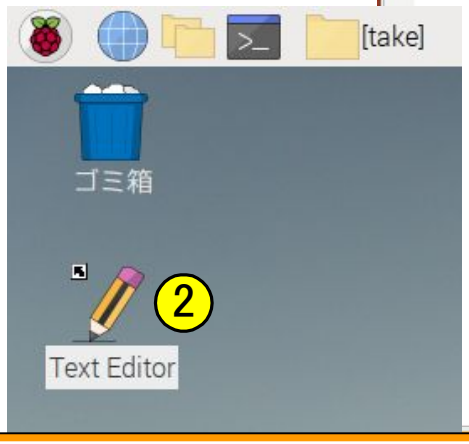
ワークディレクトリと テキストエディタ

Raspbian OSの場合は、gcc は、標準で入っているようで、すぐ使う事は出来ました。 まず、自分の ホームディレクトリ内に 作業用のディレクトリを作成し、その中でプログラムの開発作業をするといいです。
例えば、ディレクトリ名 work であれば、
`mkdir work` で、作成して下さい。
`cd work` で、中に入って作業を して下さい。

ソースファイルを作成する テキストエディタは、左上のイチゴアイコンをクリックしてアクセサリ内の Text Editor を使いました。①
アクセサリ内の Text Editor を 右クリックすると 更にメニューが出てきて デスクトップに追加を クリックすると、デスクトップ上に Text Editorの アイコンが、貼り付けられます。②
ダブルクリックすると すぐ
Text Editor が起動します。③







gccの コマンド実行

ワークディレクトリ内にて、テスト用の C ソースファイル(hello.c)を作成します。

```
#include <stdio.h>

int main( int argc, char *argv[] )
{
    printf( "Hello World!/n" );
    return 0;
}
```

hello.c を ワークディレクトリ内に保存したら、端末ウィンドウ内にて gcc コマンドを実行します。
(gccコマンドは コンパイル、リンクを 行います。)

```
gcc hello.c -o hello
```

次に、hello を実行します。

```
./hello
```

(最初に ./ を付けます。)

gccには、細かいオプション指定が、ありますが
ここでは、説明は省略します。

ビルド用シェルスクリプトの作成

デバッグ作業は、何回もソース修正、コンパイル、リンク、実行を繰り返す作業です。

特にビルド時のコマンドは、同じ文字列を入力する事になるので、シェルスクリプトにした方が楽です。
シェルスクリプトは、複数のコマンドを 自動実行する機能です。 ここでは cl.sh という 名前で 以下の3行を入力した テキストファイルを 作成します。

```
#!/bin/bash
echo "* ビルド : $1.c"
gcc $1.c -o $1 && ./$1
```

hello.c を ビルド、実行する場合は端末ウィンドウで

```
bash cl.sh hello
```

と 入力します。

```
* ビルド : hello.c      この行は表示されます。
gcc hello.c -o hello   この行は 非表示です。
./hello                この行も 非表示です。
Hello World!           Hello の実行で 表示されます。
```

シェルスクリプトを 簡単に説明

前ページの cl.sh の 中身で 分かる範囲で説明します。

```
#!/bin/bash
echo "* ビルド： $1.c"
gcc $1.c -o $1 && ./$1
```

1行目の #!/bin/bash は、シェルスクリプトを作る時は、毎回のお約束と思って下さい。
echo "* ビルド: $1.c" は、ダブルコートで囲んだ文字列を、表示するコマンドです。 この中で $1 が、ありますが、これは cl.sh の コマンドパラメータです。
例を示すと bash cl.sh hello 文字列内の hello の 語が、$1 の所に 展開されます。

```
echo "* ビルド： $1.c"
gcc $1.c -o $1 && ./$1
```

は、以下に置き換えられるという事です。

```
echo "* ビルド： hello.c"
gcc hello.c -o hello && ./hello
```

最後に、&& は、何かというと、&& の 左側の式が正常終了したら、&& の 右側の ./hello が、実行されます。 gcc の 処理が エラー等で 異常終了したら ./hello は、実行されません。 cl.sh は 一番基本的な シェルスクリプトです。 今回は、もう一つシェルスクリプト(clw.sh)を 作成します。

```
#!/bin/bash
echo "* ビルド： $1.c"
gcc $1.c -o $1 -lwiringPi && ./$1
```

-l は、リンクするライブラリファイルを指定するオプションです。 wiringPi は、今回ダウンロードしてビルドした ラズパイの GPIO端子をアクセスするための ライブラリを リンクするという指定です。

この、ライブラリ指定をしておかないと、GPIOをアクセスする関数を、使用出来ません。
I2C関数も入っています。

wiringPi関数をつかって LEDを点滅するプログラムを作る

では、今回 LEDを周期的に点滅させるプログラムの ソースを 右に示します。 この ソースプログラムを test_led.c という名前で 保存します。
そして、ビルドのコマンドは
bash clw.sh test_led と入力します。
これで、エラーが、出なければプログラムは、生成されて 実行しているはずです。
但し、10秒ほど、だんまりになります。
GPIO端子に LEDを付けていなければ、動作確認が出来ません。 という事で GPIOに接続できる形で 抵抗 1KΩと LEDを 用意する必要があります。
右の 4ピン 2列の ピン フレームに 1KΩ抵抗と LEDを 付けました。

```
#include <wiringPi.h>
#define  GPIO4  4

int  main( void )
{
    int     i;

    if( wiringPiSetupGpio() == -1 )  return  1;

    pinMode( GPIO4, OUTPUT );
    for( i=0; i<10; i++ )
    {
      digitalWrite( GPIO4, 0 );
      delay( 100 );
      digitalWrite( GPIO4, 1 );
      delay( 900 );
    }
    digitalWrite( GPIO4, 1 );

    return  0;
}
```

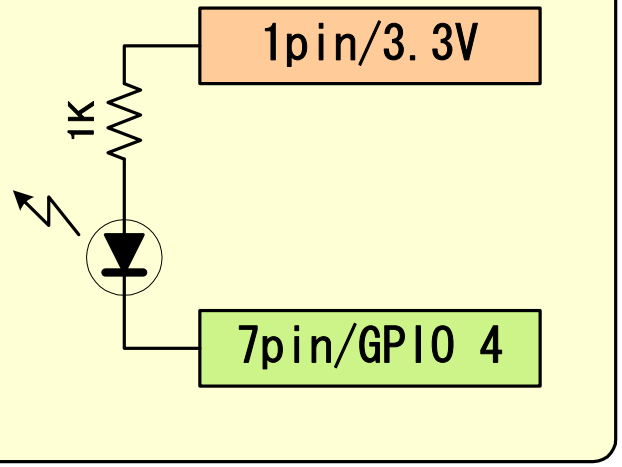
ラズパイの GPIO端子について

右に、ラズパイのGPIO端子ピン番号、信号名一覧表を示します。

1KΩ抵抗と、LEDは、1ピン／3.3Vと 3ピン／GPIO 4 の間に 直列接続しました。 下図を 参照の事。

ちなみに、 I2Cの端子は、3ピン／GPIO 2 = SDA で、5ピン／GPIO 3 = SCL です。

LED点滅動作確認用ハード回路図



	ピン番号		
3.3V	1	2	5V
GPIO 2	3	4	5V
GPIO 3	5	6	GND
GPIO 4	7	8	GPIO 14
GND	9	10	GPIO 15
GPIO 17	11	12	GPIO 18
GPIO 27	13	14	GND
GPIO 22	15	16	GPIO 23
3.3V	17	18	GPIO 24
GPIO 10	19	20	GND
GPIO 9	21	22	GPIO 25
GPIO 11	23	24	GPIO 8
GND	25	26	GPIO 7
ID_SD	27	28	ID_SC
GPIO 5	29	30	GND
GPIO 6	31	32	GPIO 12
GPIO 13	33	34	GND
GPIO 19	35	36	GPIO 16
GPIO 26	37	38	GPIO 20
GND	39	40	GPIO 21

ラズパイの CPU冷却について

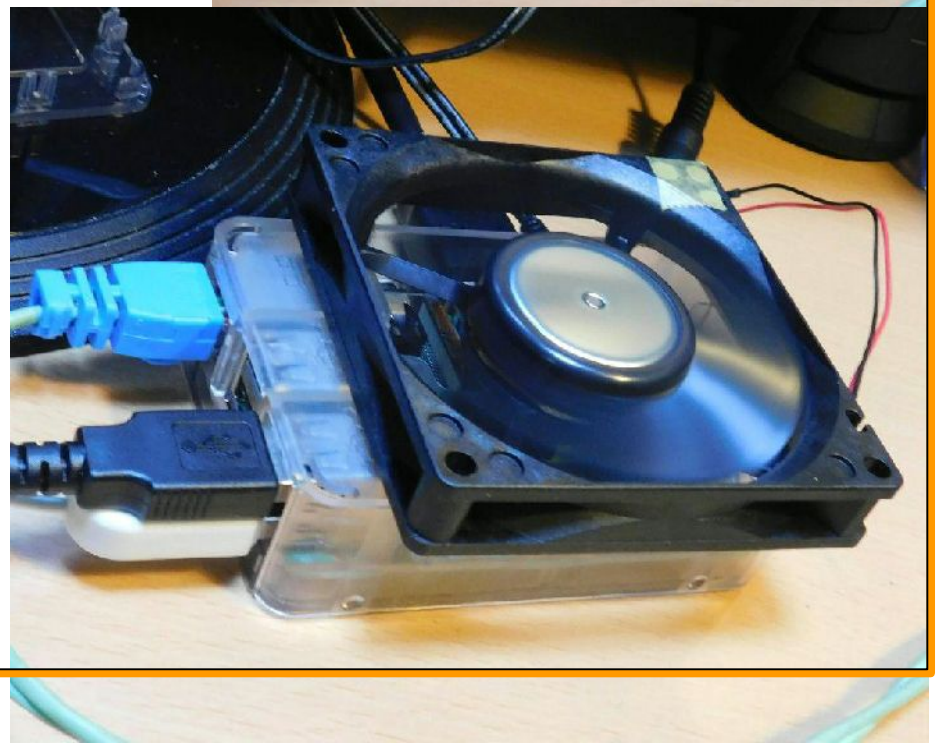
ちょっと、横道にそれますが、

ラズパイ4は、CPU冷却は、必須のようですが、
私の ラズパイ 3Bも 長時間使っていると CPUが 結構熱を持ってきます。 やはり冷やした方が、寿命の面でいいのではないかと思い 空冷ファンを付ける事にしました。さしあたり、手元にあった 80mm角の 薄型FANをラズパイの透明ケースの上に蓋を外して載せてます。

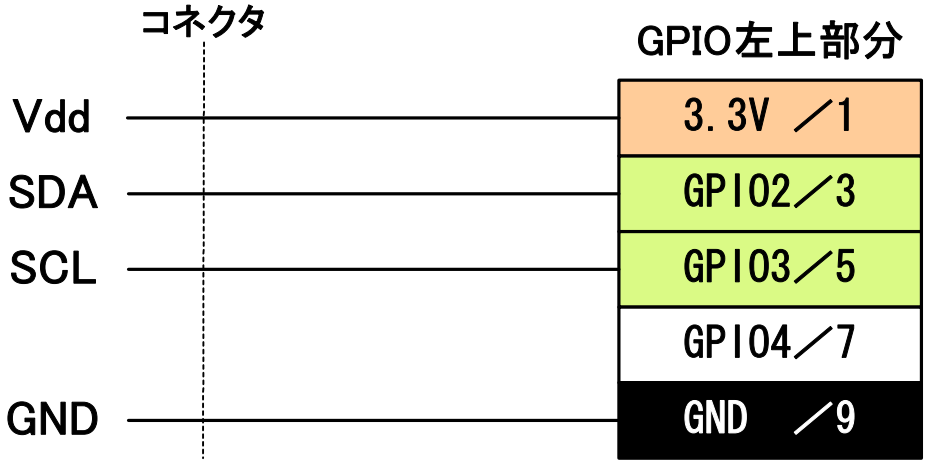
12VのFANを 6Vの小型ACアダプタが あったので、6Vで 回転させています。 通常より回転は遅いですが、その分 静かに回っています。

40mm角サイズのFANの方が、ラズパイの透明ケースの蓋に、スマートに付けられそうです。

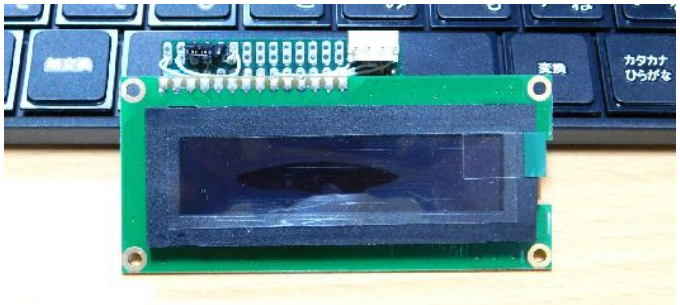
次は、LED点滅プログラムの ビルドと、実行時のLED点滅動作の動画を ご覧ください。



I2Cデバイスを接続するケーブル作成

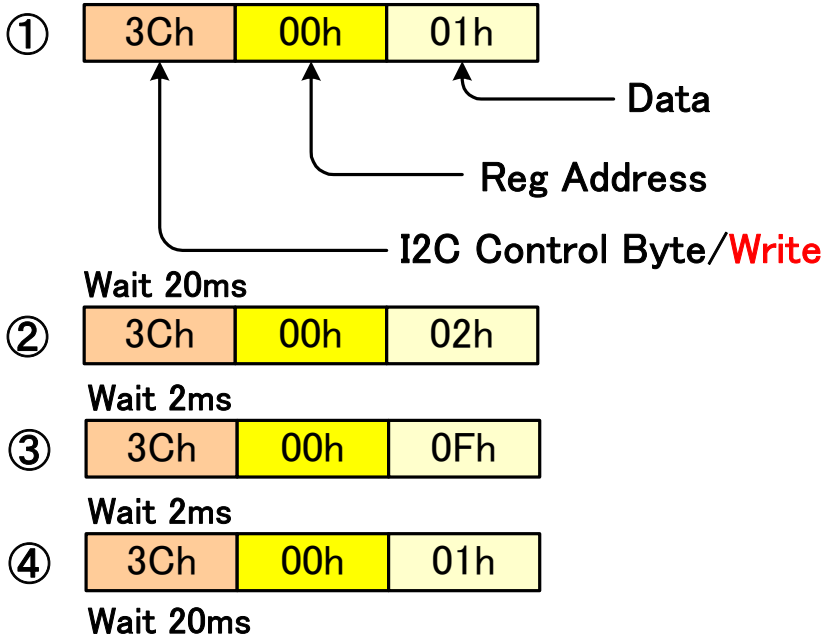


上記、ケーブルを用意して、最初は、アクセスが簡単で 動作確認しやすい 小型OLEDの キャラクタディスプレィで、I2Cアクセスを 試してみようと思います。 下の画像の 物です。



画像の物は、緑色表示の 16文字×2行の OLED キャラクタディスプレィです。 I2Cアドレスは 3Chです。このディスプレィのコマンドを、簡易 I2C電文形式で示します。

初期化:



初期化は、以上です。

(コマンド間に Wait を入れて下さい。)

1行目　文字列表示:

①	3Ch	00h	02h
	Wait 10ms		
②	3Ch	40h	'A'
	Wait 1ms		
③	3Ch	40h	'c'
	Wait 1ms		
④	3Ch	40h	'c'
	Wait 1ms		
⑤	3Ch	40h	'e'
	Wait 1ms		
⑥	3Ch	40h	's'
	Wait 1ms		
⑦	3Ch	40h	's'
	Wait 1ms		
⑧	3Ch	40h	'-'
	Wait 1ms		
⑨	3Ch	40h	'T'
	Wait 1ms		
⑩	3Ch	40h	'e'
	Wait 1ms		
⑪	3Ch	40h	's'
	Wait 1ms		
⑫	3Ch	40h	't'
	Wait 1ms		
⑬	3Ch	40h	'.'
	Wait 1ms		

1行目 文字列表示は、以上です。
表示器に Access-Test. と
表示されれば、OKです。
今回の OLED表示器の出力テスト
のソースを、次に、説明します。

wiringPiI2C 関数を使用したプログラム

今回のプログラム先頭の１/３です。
先頭で取り込んでいる
#include <wiringPi.h>
#include <wiringPiI2C.h>
の 2行が wiringPiI2C関数を 呼び出す上で必要になります。
OLED_I2C_Adr が、OLED表示器の I2Cアドレスです。
OLED_Cmd が、OLED表示器のコマンド書き込み
レジスタアドレスです。
OLED_Data が、OLED表示器のデータ書き込み
レジスタアドレスです。

メイン関数では、init_oled()；初期化処理と、
print_oled_1()； テスト文字列表示処理の、
２つの関数を 呼び出しています。

```
#include <stdio.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>

#define  OLED_I2C_Adr   0x3c
#define  OLED_Cmd       0x00
#define  OLED_Data      0x40

int    init_oled( void );
void   print_oled_1( int fd );

int  main( void )
{
     int  fd;

     fd = init_oled();
     print_oled_1( fd  );
}
```

今回のプログラム残り、2/3 と 3/3 です。
左が 初期化処理、右が テスト文字列表示処理です。やはり、I2C処理のプログラムは、長くなりますね。

```
int  init_oled( void )
{
    int  fd;

    fd = wiringPiI2CSetup( OLED_I2C_Adr );

    wiringPiI2CWriteReg8( fd, OLED_Cmd, 0x01 );
    delay( 20 );
    wiringPiI2CWriteReg8( fd, OLED_Cmd, 0x02 );
    delay( 2 );
    wiringPiI2CWriteReg8( fd, OLED_Cmd, 0x0F );
    delay( 2 );
    wiringPiI2CWriteReg8( fd, OLED_Cmd, 0x01 );
    delay( 20 );

    return  fd;
}
```

```
void  print_oled_1( int fd )
{
      wiringPiI2CWriteReg8( fd, OLED_Cmd, 0x02 );
      delay( 20 );

      wiringPiI2CWriteReg8( fd, OLED_Data, 'A' );
      delay( 1 );
      wiringPiI2CWriteReg8( fd, OLED_Data, 'c' );
      delay( 1 );
      wiringPiI2CWriteReg8( fd, OLED_Data, 'c' );
      delay( 1 );
      wiringPiI2CWriteReg8( fd, OLED_Data, 'e' );
      delay( 1 );
      wiringPiI2CWriteReg8( fd, OLED_Data, 's' );
      delay( 1 );
      wiringPiI2CWriteReg8( fd, OLED_Data, 's' );
      delay( 1 );
      wiringPiI2CWriteReg8( fd, OLED_Data, '-' );
      delay( 1 );
      wiringPiI2CWriteReg8( fd, OLED_Data, 'T' );
      delay( 1 );
      wiringPiI2CWriteReg8( fd, OLED_Data, 'e' );
      delay( 1 );
      wiringPiI2CWriteReg8( fd, OLED_Data, 's' );
      delay( 1 );
      wiringPiI2CWriteReg8( fd, OLED_Data, 't' );
      delay( 1 );
}
```

wiringPiI2C関数では RX8900をアクセス出来ない

予想してましたが、やはり wiringPiI2Cのライブラリでは、セイコーエプソンの RTC／RX8900は アクセス出来ない事が判明しました。

何故かというと wiringPiI2Cの関数は、シンプルな仕様で初心者の方には使いやすいと思います。 但し I2Cの電文が BYTE単位 WORD単位でしか出せない仕様なのと 致命的なのは、リピートスタートコンディションを使った電文を出せない事です。

RX8900をアクセスするためには、リピートスタートコンディションが必要なのです。

という事で、ラズパイには 他にも I2Cも含むGPIOの ライブラリが いくつかあるらしいので調べてみます。

ネットで調べてみると ラズパイ用の I2Cは 何種類かあるようで どれがいいのか 及びライブラリ関数などの使い方も よく分かりません。

ネット上にＵｐされてる資料は、断片的な解説が多いので、多数の資料を見て 役に立ちそうなのを複数 印刷しました。複数の資料を見比べながら判断したのは、piGPIO というライブラリでした。

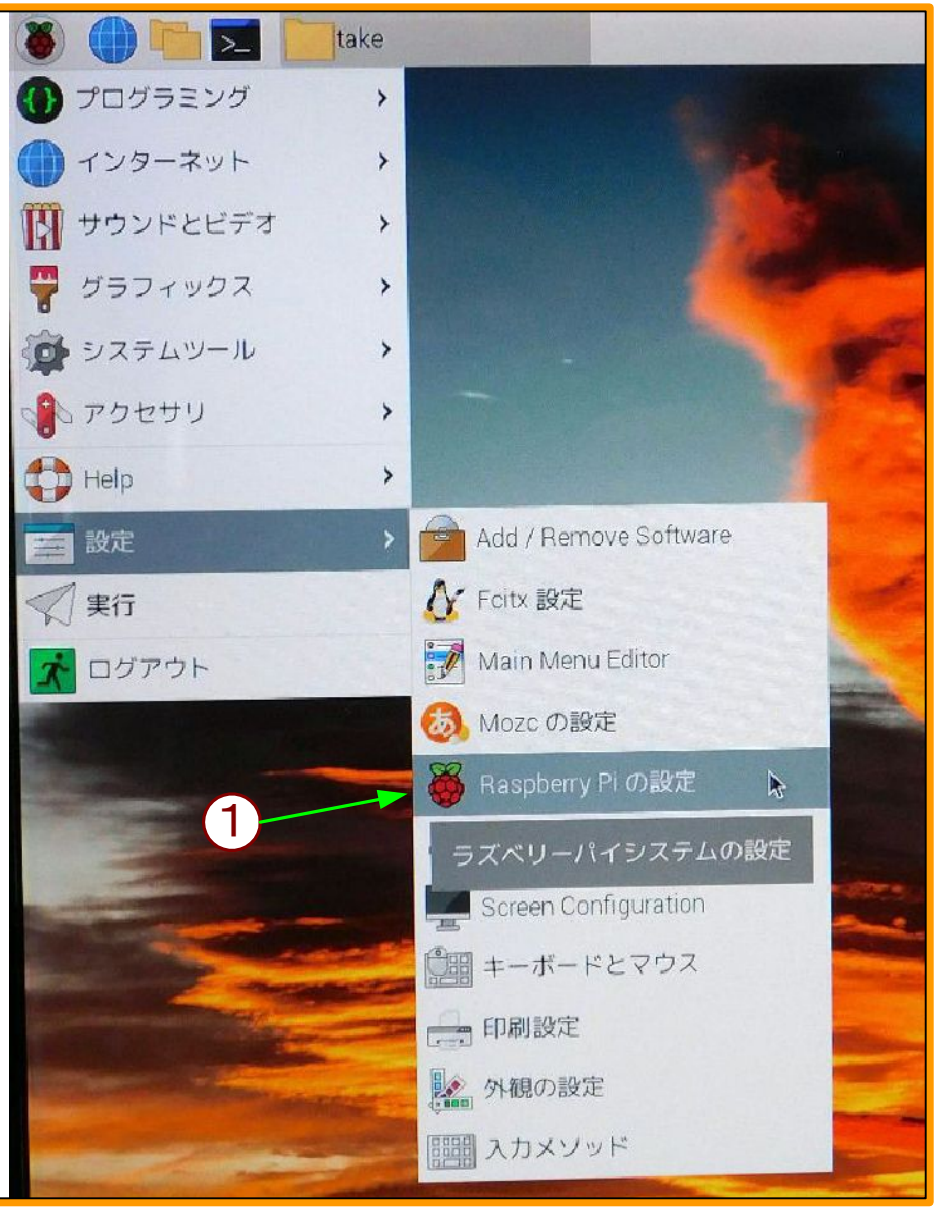
wiringPiのライブラリは スタティックライブラリで リンク時に １本の実行ファイルとして生成されます。そして多分 wiringPiのライブラリは、ラズパイの GPIO端子の I/Oポートを直接アクセスしているのではと思われます。

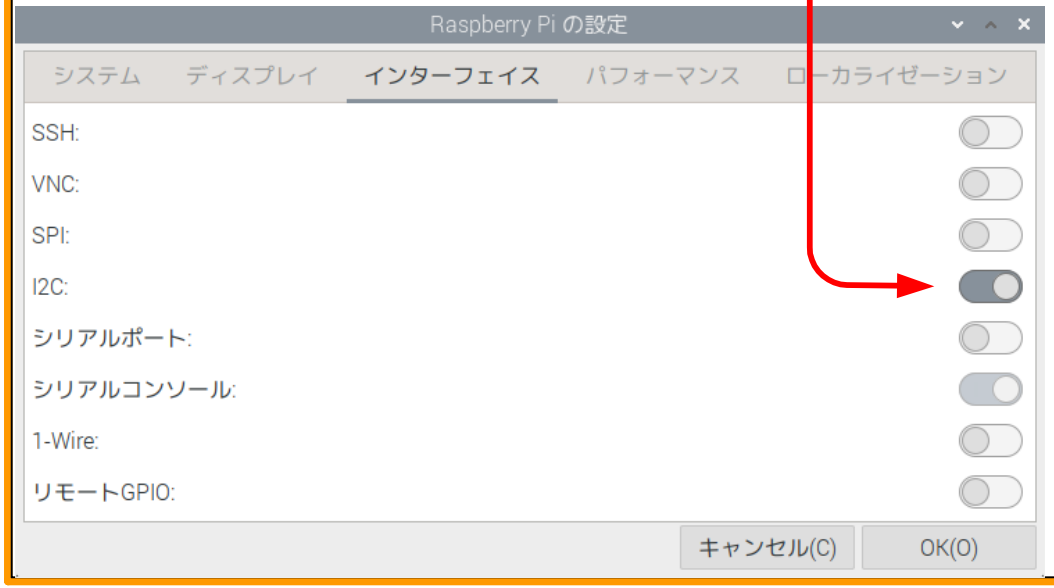
それに対し piGPIO本体は システムに登録された DLLではないかと思われます。 そのせいか、実行時に管理者権限が、要求されます。 コマンド入力時 sudo を 付けて実行すれば実行できます。Raspbian側の I2C設定も有効にする必要が あります。

piGPIOの 初期設定

LXターミナルで 以下のコマンドを入力します。
sudo apt-get update
sudo apt-get install pigpio
参考資料に書いてあったこの２行を実行したら
すでに最新バージョンです。というようなメッセージが出てきました。　pigpioは、Raspbianに最初から入っていたのかもしれません。

　次は、Raspbian上で I2Cのインタフェースを有効に 設定して再起動する必要があります。
左上イチゴアイコンをクリックして、設定 -->
Raspberry Pi の設定 をクリックします。①
Raspberry Piの設定ウィンドウが表示されます。
インターフェイスのタブをクリックします。
左の I2Cに対応する右の スイッチをクリック ②
して ON状態にします。
そして 再起動を行って下さい。





②

gccで pigpioを使う

gccで、pigpioを使用する上で、必要となる コーディングを示します。

ソースファイル先頭で、

```
#include <pigpio.h>
#include <wiringPi.h>
```

<- は delay関数などのわずかな時間待ちを入れる場合は、一緒にインクルードして下さい。

初期化処理：

```
int  sts = gpioInitialise();
```

を呼ぶ、sts が マイナスであればエラーで中断。

次に

```
int  fd = i2cOpen( 1, I2C_Address, 0 );
```

を呼ぶ、fd は、ハンドル値となる。

終了処理：

```
    i2cClose( fd );
    gpioTerminate();
```

pigpioで I2Cスレーブをアクセス

まずは、wiringPiと同様の レジスタ指定の
１Byte書き込み、読み出し関数を示します。

Byte書き込み：

int　fd： ハンドル値(I2Cアドレスを保持)
Byte reg_adr： I2Cデバイスのレジアドレス
Byte data： 書き込むデータ
int　i2cWriteByteData(fd, reg_adr, data);

Byte読み出し：

int　fd： ハンドル値(I2Cアドレスを保持)
Byte reg_adr： I2Cデバイスのレジアドレス
関数値： 読み出しデータ (多分 下位Byte)
int　i2cReadByteData(fd, reg_adr);

Block書き込み：

int　fd： ハンドル値(I2Cアドレスを保持)
Byte reg_adr： I2Cデバイスのレジアドレス
char *buf： データバッファ
int　 cnt： 書き込みデータ長
int　i2cWriteBlockData(fd, reg_adr, buf, cnt);

Block読み出し：

int　fd： ハンドル値(I2Cアドレスを保持)
Byte reg_adr： I2Cデバイスのレジアドレス
char *buf： データバッファ
int　i2cReadBlockData(fd, reg_adr, buf);

上記 ブロック読み出し関数は　読み出しByte数を引数で渡していません。 ハード的な I2Cの伝送仕様では、読み出し時、マスタは、最終 Byteを受け取ったら、データ読み出しを中断する事をスレーブに伝えるため、Nak を返す事になってます。つまり、受信文字数を 引数として渡して無いと Nakを返すタイミングが分からないはずです。？ この関数 仕様的におかしい。と思いますけど...。

もう１種類 Blockアクセス関数が ありました：

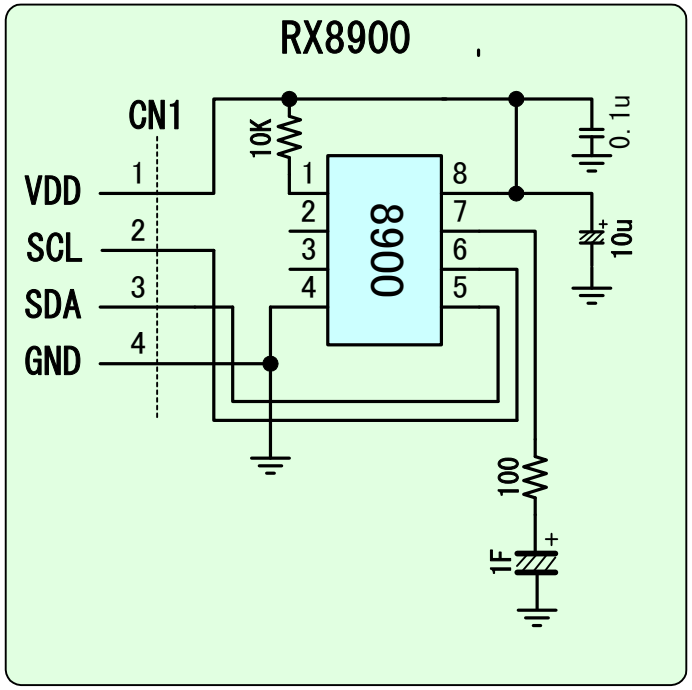
int　fd：　ハンドル値（ I2Cアドレスを保持 ）
Byte reg_adr：　I2Cデバイスのレジアドレス
char *buf：　データバッファ
int　 cnt：　書き込み or 読み出し データ長

```
int  i2cWriteI2CBlockData( fd, reg_adr, buf, cnt );
int  i2cReadI2CBlockData( fd, reg_adr, buf, cnt );
```

　この関数は 引数仕様的には まともな気がします。
それと、この引数では、reg_adrを デバイスに書き込んで読み出し位置を確定して、それからブロックデータを、読み出すものと思われます。　これを、１本の I2C電文でやろうとすれば、リピートスタートコンディションを使わないと実現できません。　　電文を １Byteの書き込みと複数Byteの読み出しの ２本に分けてあると
RX8900のアクセスは アウト　に なります。
　１本の I2C電文で リピートスタートコンディションを使用している事を期待したいのですが、関数の中身は、どうなっているのか 分かりません。
分からないので やるだけやってみます。

秋月電子 RX8900モジュール ベース基板回路図

前回の動画 最終ページの RX8900の基板回路図で、コネクタ部分を今回の仕様に合わせて変更しました。



追記：　10KΩは 付けずに 1pinは オープンにして下さい。FOUTを 出さない設定です。

RX8900アクセスのプログラム

RX8900の 設定のための電文仕様は、前回の動画にて説明していますので、そちらを参照して下さい。　用意する機能としては

① RTCの初期化
② RTC時刻の読み出し
③ RTC時刻の書き込み
④ VLFフラグの確認、初期化機能

FOUT出力の機能、１秒割込み機能は 今回は付けません。

④の VLFフラグは ICの電圧が VLOW電圧を下回った時、または 水晶発振器が 約 10ms 停止した時で、VLFが ONの時は、自動的に RTCの初期化を行います。

このような仕様で、プログラムを作成します。

ラズパイにて、RX8900を 以下の pigpio関数を使用して gccにて プログラムを作成し アクセステストを行いました。 正常にアクセス出来ました。

```
int  i2cWriteI2CBlockData( fd, reg_adr, buf, cnt );
int  i2cReadI2CBlockData( fd, reg_adr, buf, cnt );
```

左に書いている４つの機能のサブプログラムも出来ました。　という事で、RTC/RX8900の 時刻の書き込み、読み出しが 出来ます。 但し、ラズパイ上で 一つのアプリケーションとして独立して、RTCの時刻の読み出し 書き込みが出来るだけだと意味がないです。 システム起動時に Raspbianで管理している ローカルタイムに RTCの時刻を自動的に設定出来ないと、ラズパイのシステムとして意味がないです。 という事で システム起動時に RTCの時刻を ラズパイのシステムに設定出来るようにしてみます。

Linuxシステム時刻の設定

Linuxのコマンドで、時刻を表示、設定する date というコマンドがあります。
時刻を表示する時：
date
2022年 5月 18日 水曜日 23:29:29 JST　のように表示されます。
時刻を設定する時：
sudo date -s "23:49:00 05/18/2022"　入力で
2022年 5月18日 23時49分が 設定出来ます。

この dateコマンドの パラメータ部分に gccのプログラムで生成した 現在時刻の文字列を 流しこめないかと考えたのです。 コマンド間で、I/O リダイレクションとか パイプとかの機能があるので、何とかなるのではと思っていましたが、コマンドパラメータに 文字列を 流し込む手段は、見つけられませんでした。ここで、行き詰まっていましたが、別の手段を思いつきました。

gccの、文字列操作関数の種類、機能を調べている時、たまたま system(char *cmd); 関数が、目に留まりました。 一行のコマンドライン文字列を与えて実行できる関数です。　であれば、
sudo date -s "hh:mm:ss MM/DD/YYYY" の文字列をgccで作成して、system関数の 引数として渡せばローカル時刻が 設定出来るのではと思い、やってみたら出来ました。

あとは、システム起動時に このプログラムを呼び出す仕組みを作れば、ＯＫとなります。

この仕組みを実現するには、/etc/rc.local 設定ファイルを 編集すれば出来ます。rc.local は起動時に実行されるスクリプトファイルです。 そのファイルの最後に exit 0 がありますので、その前の行に

```
cd /home/take/take   今回のプログラムの格納場所
./rtc_8900 -sst      今回のプログラム名
```

の２行を 挿入しました。これでシステム起動時にRTCの時刻を設定する事が出来ました。

細かい話ですが、/etc/rc.local ですが、GUIのテキストエディタでは、ファイル設定が、読み出し専用になっているので、変更できません。 CUI環境で使う vi エディタで、管理者権限で 編集する必要が あります。

```
sudo vi /etc/rc.local
```

vi エディタは、使い方を ネットで調べてから使用して下さい。 コマンドで動かすエディタなので、最初は悩みます。

最後に 保存する時は、:wq! で、保存終了して下さい。

ちなみに rtc_8900 のソースは rtc_8900.c １本です。 250行ぐらいのプログラムなので、大した事はないと思います。今回も ラズパイ上の ワークディレクトリ内のファイル一式を、ダウンロード出来るように しておきます。

rtc_8900プログラムの使い方

今回作成した RTC/RX8900をアクセスする
プログラム名は、rtc_8900 です。
rtc_8900 は 管理者権限が無いと 動きません。

ここでは、パラメータの与え方で どのような
機能があるかを 説明します。

① パラメータ無しは、RTC時刻の表示です。

```
sudo ./rtc_8900
YY-MM-DD/hh:mm:ss
```

RTC時刻を 上記 Formatにて表示します。

② -rcw パラメータは、RTC時刻の設定です。
時刻は、年 月 日 時 分 秒の順で 各項目
２文字で、1 の場合は 必ず 01 と入力して
下さい。

```
sudo ./rtc_8900 -rcw YY-MM-DD/hh:mm:ss
例） sudo ./rtc_8900 -rcw 22-05-20/12:30:00
```

③ RTC時刻を、Raspbian OSに 設定する場合。
-sst の パラメータを使用します。

```
sudo ./rtc_8900 -sst
```

起動時 /etc/rc.localにて自動実行する場合
画面上は、何も表示されませんが、
ターミナル窓で、上記コマンドを入力すると
date コマンドで表示される Formatで、設定
された時刻が表示されます。

例）2022年　5月 20日 金曜日 11:17:45 JST