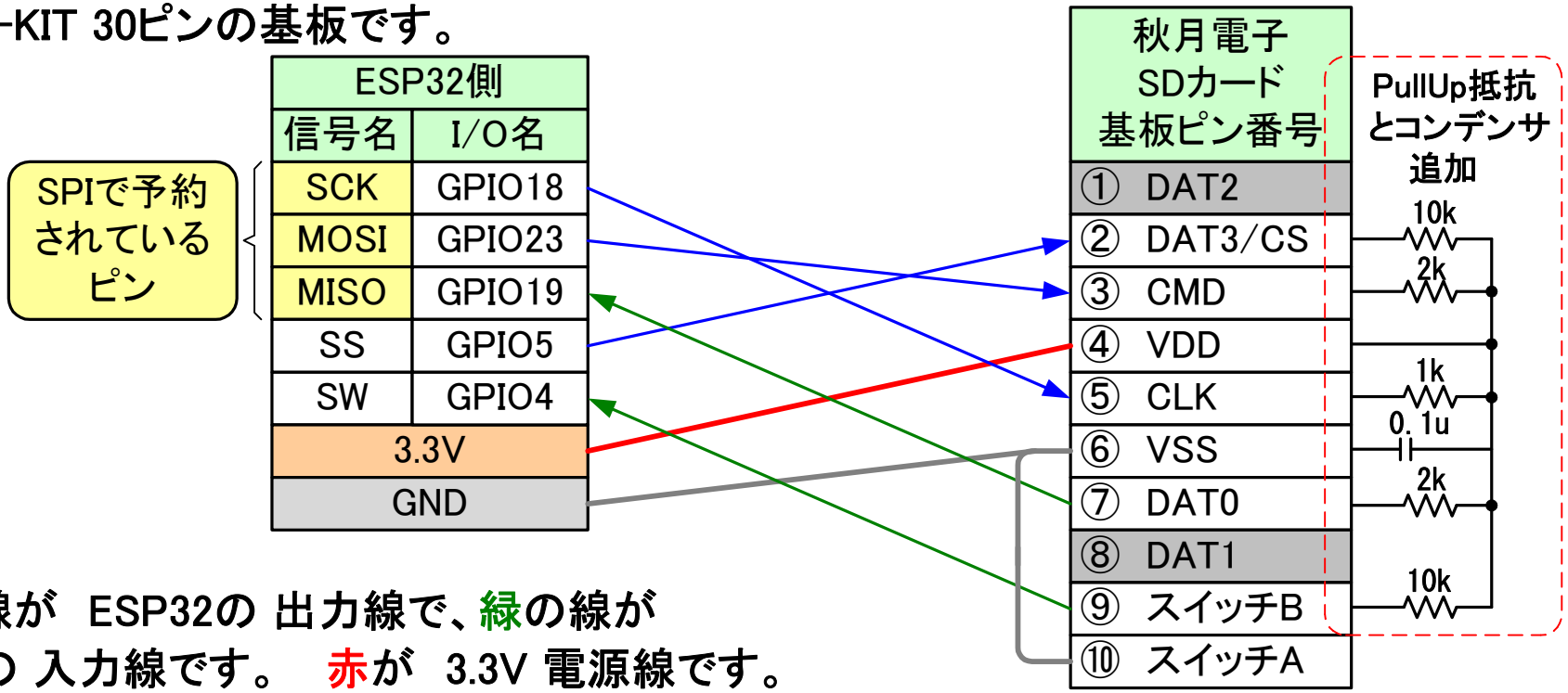
# ESP32と SDカード基板の接続（改）PullUp抵抗の追加

前回の接続図にて、PullUp抵抗の抜けが ありした。
追加しておきます。 使用する ESP32は ESP32-WROOM-32
の DEV-KIT 30ピンの基板です。

0.1uFの積層セラコンを
VDD-VSS間に 入れました。




青の線が ESP32の 出力線で、緑の線が
ESP32の 入力線です。 赤が 3.3V 電源線です。
灰色が グランド線です。
そして、⑥と ⑩を 結線します。

CLKが、最も高速動作するので 1kΩに します。
CMDと DAT0が 次に高速動作するので 2kΩに
します。 他は 10kΩと します。

# SDカードアクセス、セクタの概念

ハードの準備は出来たので 次はソフトです。いきなりアルディーノライブラリで、SDカードをアクセスしてもいいですが、その前に ちょっとSDカードの事、FATファイル管理の事について説明します。

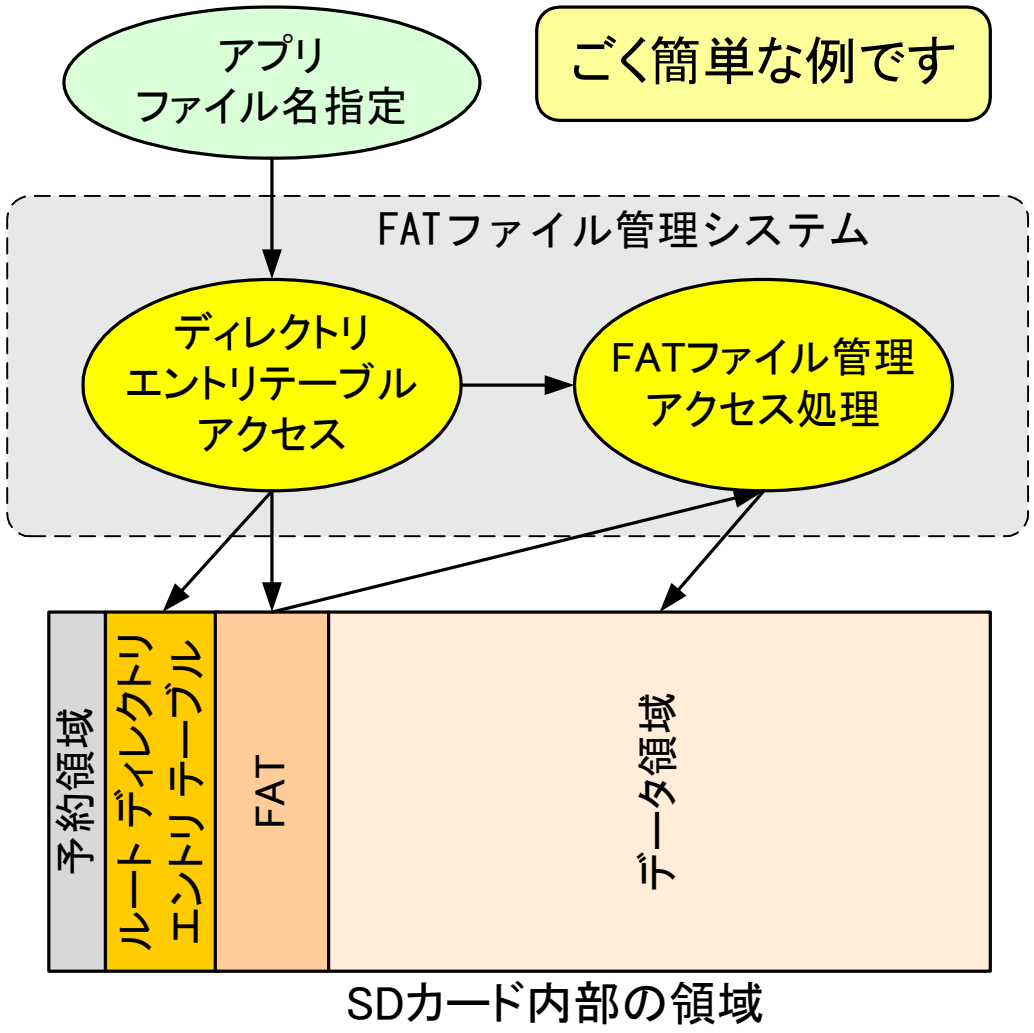
まず、SDカードに限らず、HDDもそうですが、物理的には 1byte単位では アクセス出来ません。 セクタと呼ばれるアクセス単位が ありま す。 SDカードも、HDDも 1セクタは 512byteです。 昔の事で 恐縮ですが

昔のフロッピー 2HDとかは、フォーマットにより、1セクタが 256、512、1024 byteとか 変化します。 但し、1セクタのサイズを大きくすると、1トラックに入るセクタ数が、少なくなります。よって 1トラックに byte数が 最大に なるセクタサイズを 選択されてました。

HDDの様な回転する高速記憶媒体は、ヘッド、トラック、セクタという概念が ありました。ヘッドは 回転する磁性体円盤が、1枚の場合は、ヘッドは 円盤の表、裏にあります。 トラックは 陸上競技のトラックと同様に 同心円状の 円周を指します。 そして一つのトラックを 細かい回転角で分けた円弧状の区画を セクタといいます。 特にトラック間の移動は、ヘッドシークといって、機械的にヘッドをマウントした キャリッジを移動させます。 よってヘッドシークはやや時間がかかります。そして所定のトラックに来て トラック内の目的のセクタを 読み書きできる位置に来るまで、待つ時間を 回転待ち時間といいました。 よって、機械的な待ち時間が存在するので、いかに高速にアクセスするか、ヘッド、トラック、セクタの パラメータ調整がアクセスタイムの向上に重要でした。

SDカードや USBメモリには、ヘッドや トラックという概念は 有りません。 USBメモリは分かりませんが、SDカードは 1セクタのサイズは 512byteです。 そのセクタが 桁の大きい連番で管理されています。 SDカード内のフラッシュメモリ内のセクタの区画は、別の物に例えると、1要素 512byteの データの 巨大な配列と見る事が出来ます。 どの要素のデータを取り出すかは、配列の添え字で指定しますが、それが SDカードのセクタ番号という事です。 で、このセクタの概念が FATファイル管理システムにも密接に関係してきます。

アプリからは、ファイル名を指定して ファイルを アクセスします。 そして書き込んだり、読み出したりするデータのサイズは、任意に指定できます。 それが、最終的に物理層に行くと、512byteのセクタ単位になる という事です。



ディレクトリエントリテーブルは 遥か昔 MS-DOSの時代は ファイル名が ファイル名 8byte と 拡張子 3byteで 32byteの ディレクトリエントリテーブルで、1個のファイルを管理していました。FATのエントリポイントや、ファイルサイズが、格納されています。 長いファイル名に対応する方法は、この32byteのディレクトリエントリテーブルを連結して長いファイル名に対応していました。 サブディレクトリの対応は、データ領域に 新たにディレクトリエントリテーブルを作成するという方法で 対応しています。

このような FATの情報は、昔 90年代 MS-DOSから Windowsへの 移行時期の頃に 書籍が ありました。

Windowsの NTFSとかは、全く構造が異なると思います。

## SDカードアクセスの ライブラリ

ちょっと余談が過ぎましたが、本来の話に入ります。 まず、Arduino IDEの ライブラリで SDカードを アクセスするには、以下の 2つの インクルードファイルを 取り込みます。

```
#include <SPI.h>
#include <SD.h>
```

それと、1ページ目で SS( スレーブセレクト ) 信号を GPIO5 ( 5 ) に 設定してましたので、setup関数内にて

```
  Serial.begin( 115200 );// シリアルオープン
  if( SD.begin( 5 ) )    // SD オープン
      Serial.println("SD Ready.");  // SD OK
  else {
      Serial.println("SD Error.");  // SD NG
      while( 1 );
  }
```

を 行います。 エラーが 起きたら そこで止まるように しておきます。

# ファイルの オープンと クローズ

ファイルの オープンは、 ファイルのオブジェクト を 関数先頭で宣言します。
File f; 左のように宣言します。

今回は データロガーを 想定しているので ファイルを新規作成して データを書き込み続ける使い方を 想定します。

ファイルのオープンは、書き込みモードで オープンします。
f = SD.open( "/test.dat", FILE_WRITE );
で、ファイル名文字列の先頭に / を付けていますが、これが無いとファイルのオープンに失敗します。 で、この場合は、ルートディレクトリ上に ファイルを作成します。 遥か昔は ファイル名 8文字、拡張子 3文字でしたが、今回、18文字のファイル名で ファイルを 作成できる事を確認しました。

FILE_WRITEの 意味ですが、指定されたファイル名が、無い場合はファイルを 新規作成し、先頭に書き込み位置を設定します。 指定されたファイル名が ある場合は、ファイルをオープンして先頭に書き込み位置を設定します。

ファイルの書き込み処理を終了する時は ファイルのクローズ処理を 行います。
f.close(); を 行います。

データの書き込み:
今回は、バイナリコードを含んだ固定長の1レコードの バッファを書き込みます。 この場合 データを 構造体にした方が便利です。Rec という構造体を 書き込む場合は
f.write((uint8_t *)&Rec, sizeof( Rec ));
と、なります。 ちょっと難しそうにみえるので、バラして説明します。

```
f.write((uint8_t *)&Rec, sizeof( Rec ));
```

f.writeの 引数として渡すのは、第一引数はデータのポインタで、第二引数は、データのサイズ( byte単位の値 )です。

第二引数の sizeof() は 演算子で、引数のデータの大きさを返します。 int だったら 4 になります。 で、第一引数の方ですが、Recという構造体変数に & が 付いてます。&は その変数が、配置されるメモリの先頭アドレスを 返す演算子なのです。(uint8_t *)は パソコン上の C言語であれば、(char *) となります。 これらは 一般にキャストと呼ばれます。 データの型を 引数の型に 合わせて渡すための演算子です。

uint8_t は ESP32独自の型なのか、あるいは Arduino環境の 独自の型なのかは 分かりませんが、独自の型と思われます。

パソコンであれば (char *)で OKです。
どちらも、byte単位のデータ型なのに 異なる名前を 付けてある関係で、(uint8_t *)の所に (char *) を 置くとエラーになります。 よって、ESP32の f.write の第一引数には (uint8_t *) のキャストを 付けて下さい。

**シーク関数:** ファイル上の読み出し、書き込み位置を 移動させる関数です。
例) `f.seek( 0 );` 引数は ファイル先頭からの byte単位の 移動量となります。0 で あれば ファイル先頭に、書き込み、読み出し位置を戻す事になります。

**ファイルサイズを得る関数:** byte単位の ファイルサイズを 得る関数 `n = f.size();` が あります。 今回は、使いませんでした。

## ファイルの削除:

SD.remove( "test.dat" ); ファイル名を指定して ファイルを削除する関数です。 削除する
ファイルが、無い場合は、関数値で、False が 戻るだけで、支障はないです。

## 今回、遭遇した障害:

今回、データレコード構造体として、以下の 128byteの構造体の型宣言をしました。

```
typedef struct {
  int   id, cnt;
  char  ttl[24];
  char  tx[96];
} RECORD;
```

そして、構造体変数 Recを 宣言をしました。

```
static  RECORD  Rec;
```

次に setup関数内にて、Recのメンバー変数 txに ASCIIコード 96文字を入れました。

```
  for( i=0; i<96; i++ ) {
    Rec.tx[i] = i + 32;
  }
```

ところが、loop関数内で使ったら、ASCIIコードが Rec.txの中に入って無い。？

これは、コンパイラのオプティマイザにより、setup関数内にて、ASCII文字データを 設定した後、setup関数内では、何も ASCII文字データを利用して無いので 意味のないコーディングという事で、オプティマイザによる最適化が実行されてコーディングが 外された物と思われます。 これを避けるためには、影響を受ける変数を 宣言している箇所で Volatileを 追加する事で 最適化を 抑える事が出来ます。

```
//static  RECORD  Rec;  // 変更前
volatile static RECORD  Rec; // 変更後
```

という事で、今回はファイル名先頭に / を付ける事と、最適化の悪影響を受ける場合は該当する変数宣言に volatile を 付ける事の2点で、後は 順調にプログラム開発が進みました。

今回は、テストプログラムによりテストデータを5本書き出しました。

```
void loop() {
  data_write( "/test_01.dat", 'A', 200 );
  data_write( "/test_02.dat", 'B', 400 );
  data_write( "/test_03.dat", 'C', 600 );
  data_write( "/test_04_ABCDE.dat",
                                  'D', 800 );
  data_write( "/test_05_0123456789.dat",
                                  'E', 1000 );
  while( 1 );
}
```

data_write()関数が 1本、データファイルを書き出す処理です。最初の引数がファイル名で、2番目が 1レコードで 同じ文字を8個書き込んでいます。 3番目の引数が、書き出すレコード数です。 最後は 1000レコード書き出しました。
全て、無事 書き出せました。

# SDカードに出力したデータの Hexダンプ

このダンプリストは　1レコード 256byteで
今回のデータレコードは 1レコード 128byteなので
データ先頭の　2レコードを ダンプ表示してます。

```
FileName = F:¥test_01.dat    Record = 0    2024/09/07
                Id                cnt                ttl
0000    00 00 00 00   C8 00 00 00   41 41 41 41   41 41 41 41   ....ﾈ...AAAAAAAA
0010    00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   ................
0020    20 21 22 23   24 25 26 27   28 29 2A 2B   2C 2D 2E 2F    !"#$%&'()*+,-./
0030    30 31 32 33   34 35 36 37   38 39 3A 3B   3C 3D 3E 3F   0123456789:;<=>?
0040    40 41 42 43   44 45 46 47   48 49 4A 4B   4C 4D 4E 4F   @ABCDEFGHIJKLMNO
0050    50 51 52 53   54 55 56 57   58 59 5A 5B   5C 5D 5E 5F   PQRSTUVWXYZ[¥]^_
0060    60 61 62 63   64 65 66 67   68 69 6A 6B   6C 6D 6E 6F   `abcdefghijklmno
0070    70 71 72 73   74 75 76 77   78 79 7A 7B   7C 7D 7E 7F   pqrstuvwxyz{|}~.
0080    01 00 00 00   00 00 00 00   41 41 41 41   41 41 41 41   ........AAAAAAAA
0090    00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   ................
00A0    20 21 22 23   24 25 26 27   28 29 2A 2B   2C 2D 2E 2F    !"#$%&'()*+,-./
00B0    30 31 32 33   34 35 36 37   38 39 3A 3B   3C 3D 3E 3F   0123456789:;<=>?
00C0    40 41 42 43   44 45 46 47   48 49 4A 4B   4C 4D 4E 4F   @ABCDEFGHIJKLMNO
00D0    50 51 52 53   54 55 56 57   58 59 5A 5B   5C 5D 5E 5F   PQRSTUVWXYZ[¥]^_
00E0    60 61 62 63   64 65 66 67   68 69 6A 6B   6C 6D 6E 6F   `abcdefghijklmno
00F0    70 71 72 73   74 75 76 77   78 79 7A 7B   7C 7D 7E 7F   pqrstuvwxyz{|}~.
```

Idは　0 から始まる レコード番号です。

cntは　書き込んだレコード数です。
因みに　C8hは 200 です。

1レコード 128 byteの内 後ろの 96byteは ASCII文字です。

# 使用開発環境の バージョン等

ESP32-WROOM-32
mode:DIO, clock div:1
load:0x3fff0030,len:1448
load:0x40078000,len:14844
ho 0 tail 12 room 4
load:0x40080400,len:4
load:0x40080404,len:3356
entry 0x4008059c
--------------------------------

Arduino IDE
バージョン：2.3.2
日付：2024-02-20
CLIバージョン：0.35.3
Arduino ESP32 Boards
by Arduino
2.0.13 intalled