I2C通信機能を ソフトで実現、I/Oポート設定

P3 0D	b7	
	b6	
	b5	CN2/ 28 : NMI
	b4	
	b3	
	b2	CN2/27 : I2C
	b1	CN2/26 : I2C
	b0	CN2/25 : RxD1

左は、今回 I2Cで 使用する予定の、ポート3の I/Oポート表です。
この中の b1と b2 を使用します。

このポート3を 使用する上での注意点ですが NMIの入力端子でもある、P35は、絶対に出力ポートにしては、ならない。という事です。
出力ポートとして機能しない。 というだけではなく、誤動作の原因にもなりかねません。
もう一つが、RxD1 として使用される P30 です。 この端子は 周辺回路 SCI1 に 占有されてますので、出力ポートに設定しても I/O ポートとしてアクセスできません。 RxD1 は、データ受信側なので、入力に設定します。

という事で、P35と P30 は、入力に設定します。
I2Cで使用する、P31(SCL) と P32(SDA) は出力に設定します。 P31 は、ずっと出力ポートとなります。 P32は、Write時の ACK読出し時、データRead時に、一時的に 入力に なります。

初期化時の設定:

```
PORT3.PODR.BYTE = 0x06; // Data 初期値
PORT3.PDR.BYTE  = 0x06; // 入出力方向 初期値
  // P35は 1 にしては いけない ハングする。?
PORT3.ODR0.BYTE = 0x14;
      // b1 と b2 を オープンドレインに設定
```

データ読み取り時の設定:

```
PORT3.PDR.BYTE  = 0x02; // 入出力設定 Read時
```

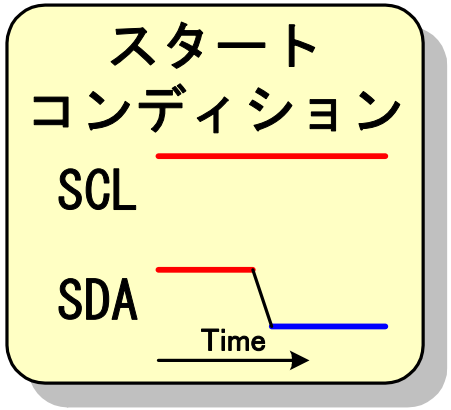
に、なります。 次に、ソフトで制御するので、I2C通信シーケンスをしっかり理解する必要があります。
過去の動画に「024 USB-I2C変換アダプタを作る、続編」に I2C通信シーケンスの いい資料が ありましたので流用します。 私も 細かい事は 忘れていました。

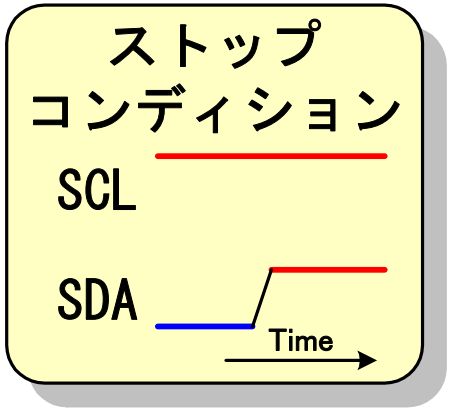
I2C通信シーケンス(1)

I2C通信は、SCLと SDAの２本の信号線を用います。待機中 SCLと SDAは、両方とも Hiレベルです。

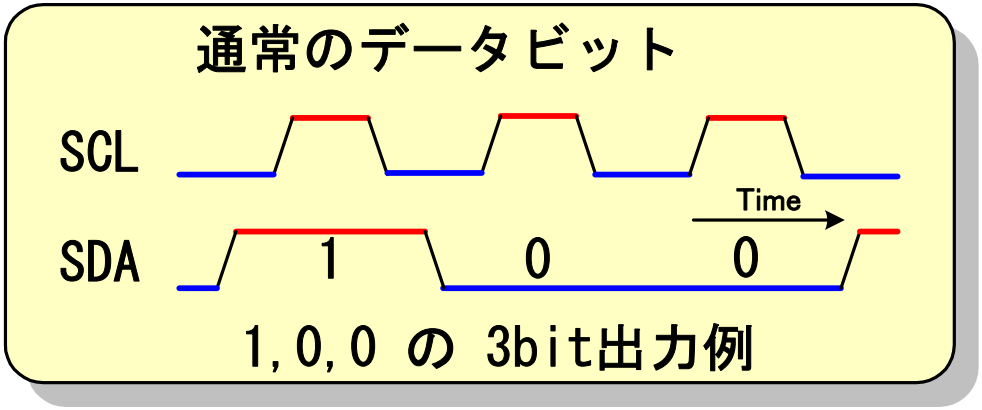
[1] スタートコンディション：
今から通信シーケンスを開始する事をマスタが、スレーブに通知するための信号です。 SCLが、Hiの期間中に SDAを Hiから Lowに変化させます。

[2] ストップコンディション：
マスタが、スレーブに対し通信を終了させる時に出します。 SCLが、Hiの期間中に SDAを Lowから Hiに変化させます。





通常のデータビットでは、SCLが Lowの期間中に、SDAを変化させるので、データビットと、スタート／ストップ コンディションは、区別出来ます。

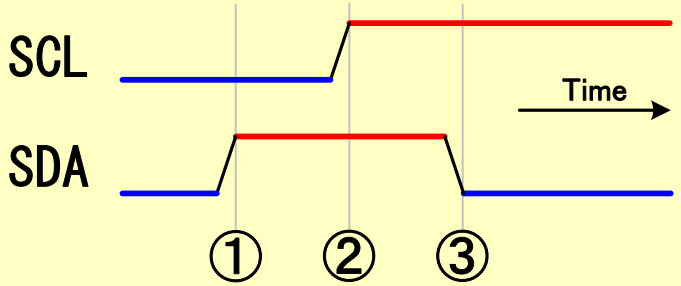


I2C通信シーケンス(2)

[3] リピートスタートコンディション：
　8ピンの EEPROMをアクセスする際にリピートスタートコンディションを発行する場合があります。

① SCLが、Lowの期間に一旦、SDAをHiにします。
② SCLを Hiにします。
③ SDAを Lowにします。

リピートスタートコンディション

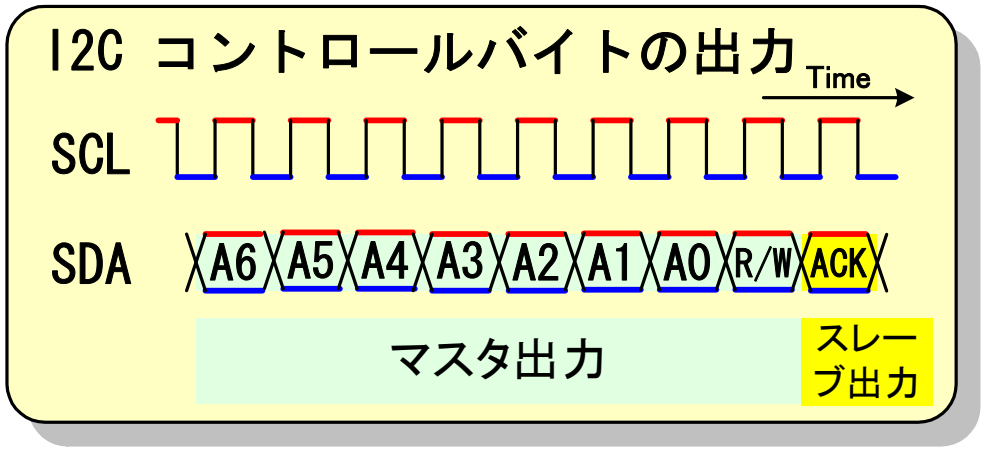


最近は、殆どのマイコンに、データ用フラッシュROMが入っている事もあり外付けで 8pinのシリアルEEPROMを使う事が、少なくなってきました。
　これにより、リピートスタートコンディションを使う機会も減ったように思います。

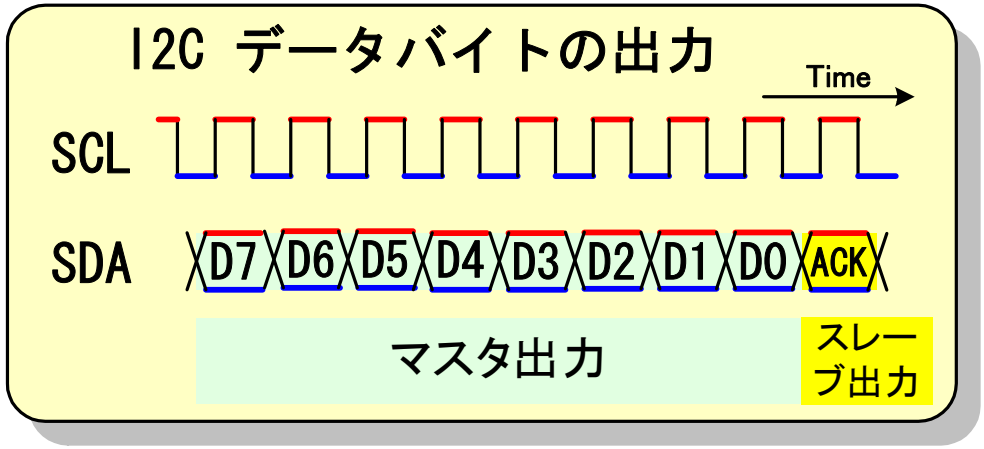
I2C通信シーケンス(3)

[4] I2Cコントロールバイト：
スタートコンディション直後、最初に出力するバイトデータが、コントロールバイトです。今回は、7bitアドレスで説明します。 10bitアドレスも規格上はありますが、私は使った事が無いです。

① 一旦 SCLをLowに降ろします。
② スレーブのI2CアドレスのA6 ～ A0の 7bitを 順次 bit単位でスレーブに書き込みます。
③ 次にデータを書込む際は、Write(SDA=Low)、読出す際は、Read(SDA=Hi)を、1bit 出力します。スレーブからの ACK/NAK(1bit)を受け取ります。

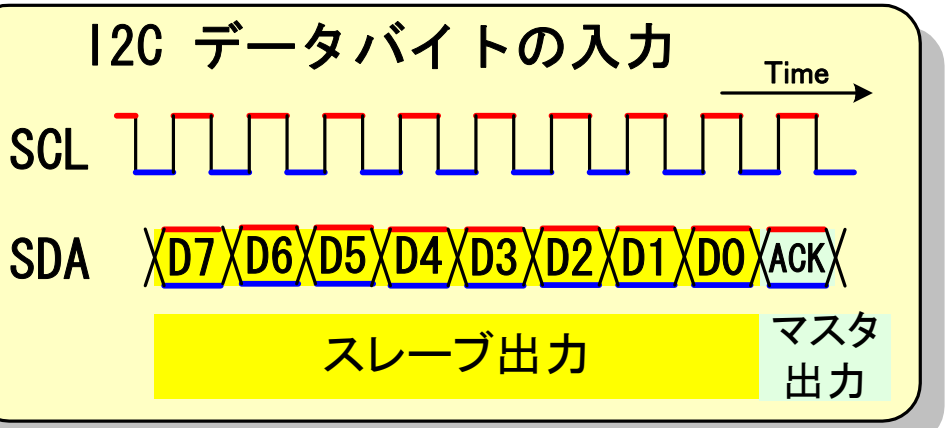


[5] データバイト出力（Write）：
内容(データ)が異なるだけで、コントロールバイト出力と同じです。



I2C通信シーケンス(4)

[6] データバイト入力（Read）：
SDAの出力元が、入れ替わるだけで
シーケンスは、同じです。



[7] 一連の電文シーケンス例：
I2Cスレーブアドレス 3Ch に、
40h、41hのデータ2byteを 書き込む
例です。

① スタートコンディションを実行。
② 7bitAddress = 3CHでコントロール
バイト(Write)を、出力します。
③ データ40hを データバイトとして
出力します。
④ データ41hを データバイトとして
出力します。
⑤ ストップコンディションを実行。

ACK／NAKに関して：
通常、通信制御コードの ACK、NAKは、肯定応答、否定応答の意味で、送り元が、受信側からNAKを受け取った場合は、再送信等のエラーリカバリ処理を行います。が、I2Cはどちらかというと、転送する最終バイト識別の意味合いで用います。

I2C通信機能を ソフトで実現、I/Oポート管理

I2Cの信号は、SCLと SDAの ２本だけです。であれば、 SCLと SDAの 瞬時の組み合わせは、４パターンだけです。

SDA(P32)	SCL(P31)	P3に出力するデータ
0	0	00h
0	1	02h
1	0	04h
1	1	06h

よって、00h、02h、04h、06h を 差し替えながら、ポート3に出力する事で、I2Cのシーケンスが作れます。

ビットデータの読み出し時は、SDA出力はオープンドレインなので、1にしておいた方がいいと思います。

であれば 04h、06h を 使う事になります。

書き込み時と 読み出し時の SDAの入出力切り替えは、以下のように ディレクション制御を 行います。

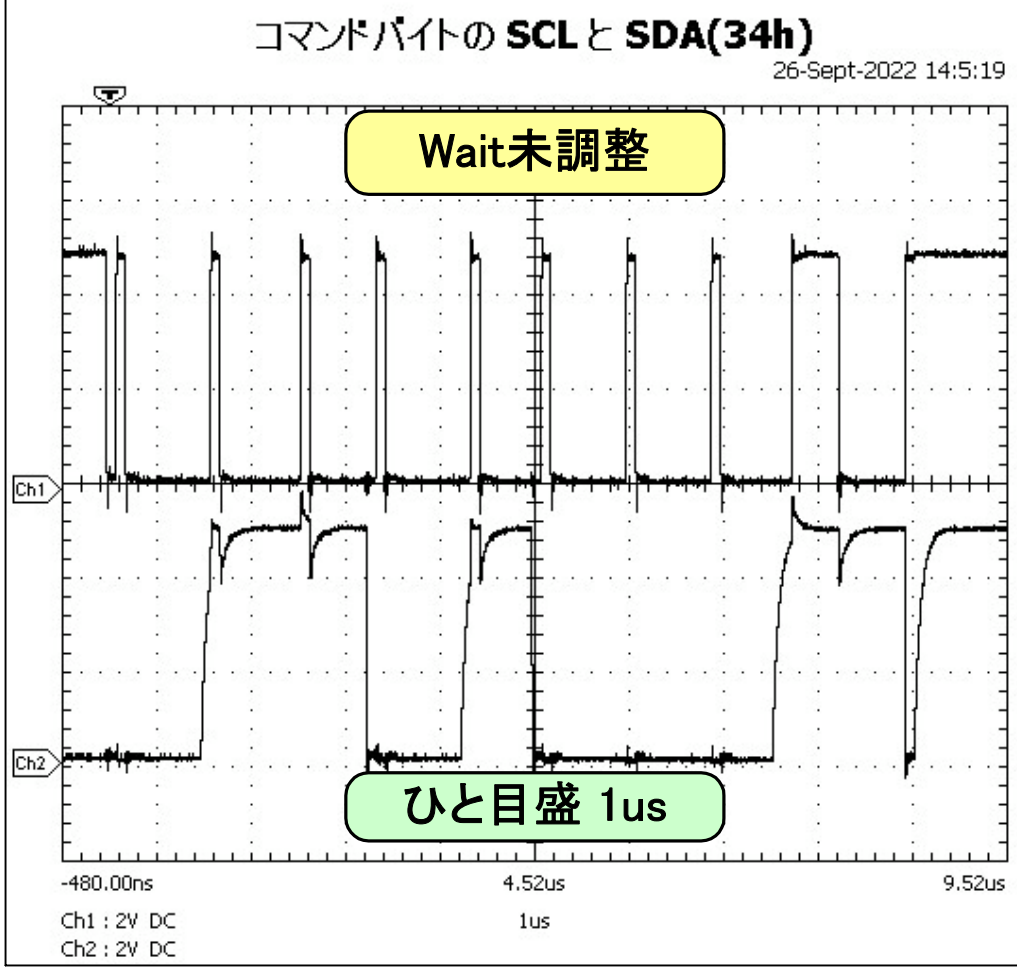
```
PORT3.PDR.BYTE = 0x06; // 書き込み切り替え
                       // SDA=1 、SCL=1
PORT3.PDR.BYTE = 0x02; // 読み出し切り替え
                       // SDA=0 、SCL=1
```

左の ４つの 出力データパターンと 上記 ２つの ディレクション制御を 組み合わせれば、I2C 通信シーケンスは 実現出来ます。

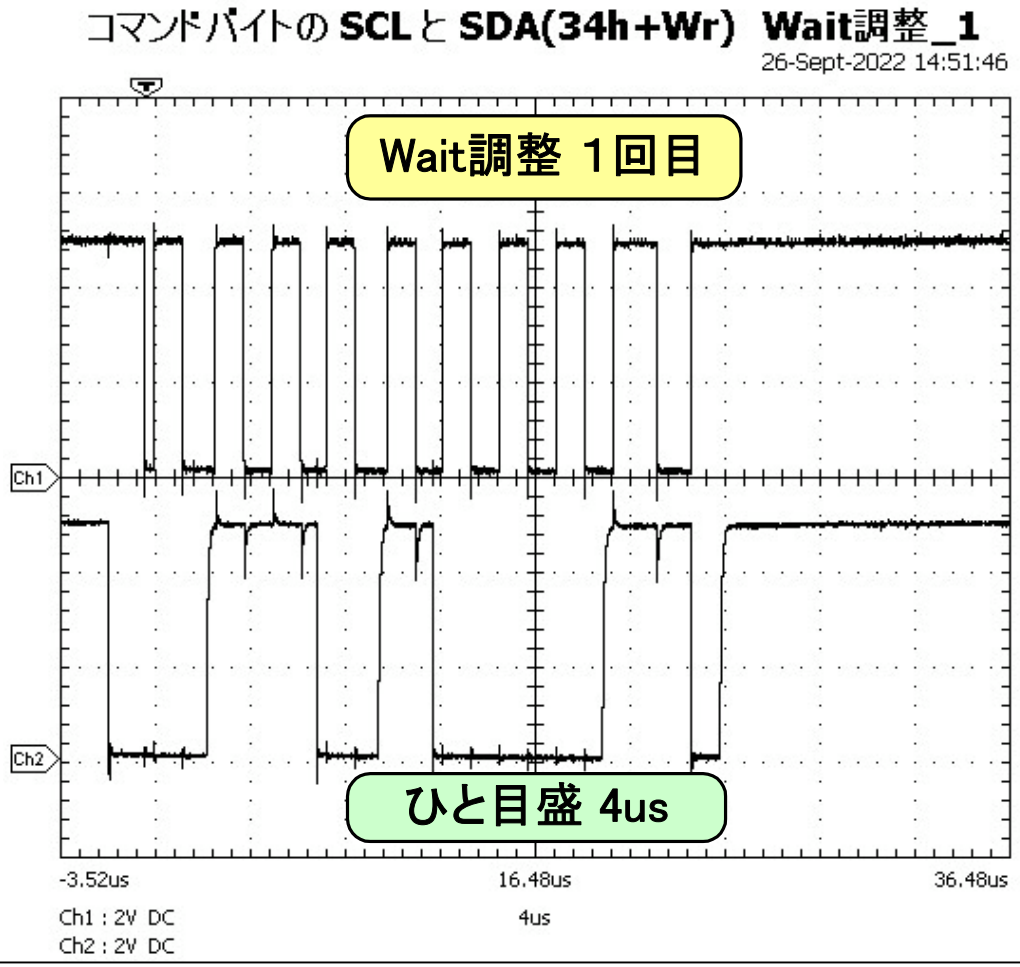
ここまで、約束事を決めれば、あとはプログラムを作成した方が 早いと思います。

RX220で プログラムを作成した場合、400Kbpsよりも 早く動く事も考えられます。 その場合僅かな 空ループを入れて、オシロを見ながらタイミング調整を 行う必要があります。

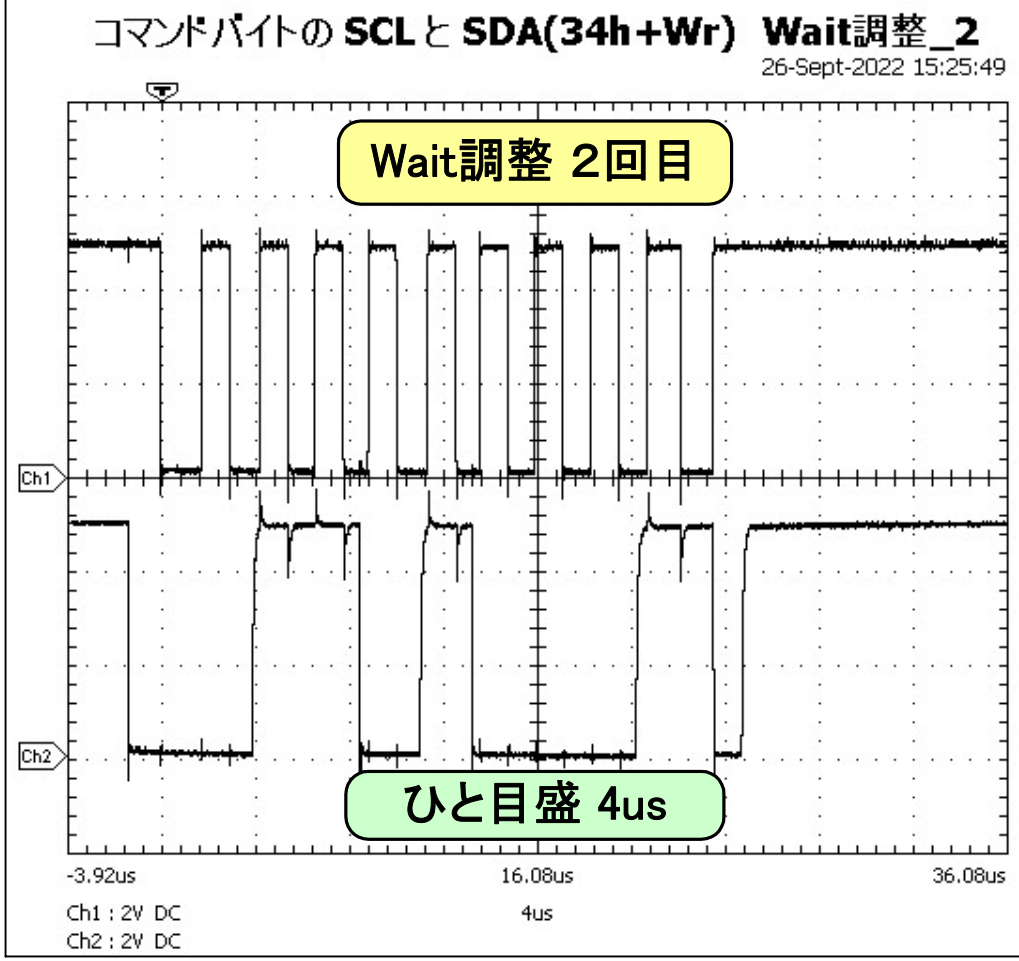
SCLの1ビット周期が 1usぐらいで、
パルス幅も狭い。 という事で 早過ぎる。



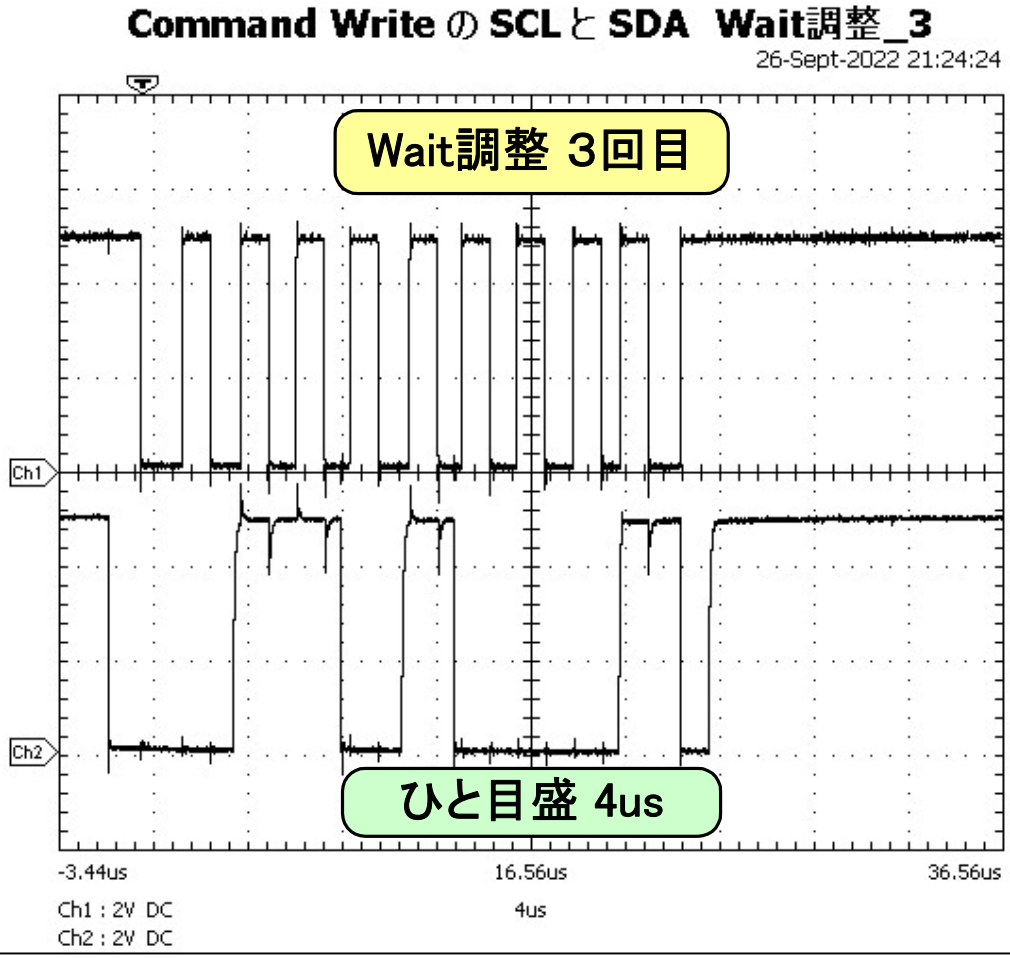
SCLの1ビット周期が 2.5usぐらいに調整
パルス幅を 1.2usぐらいに調整。約 400Kbps



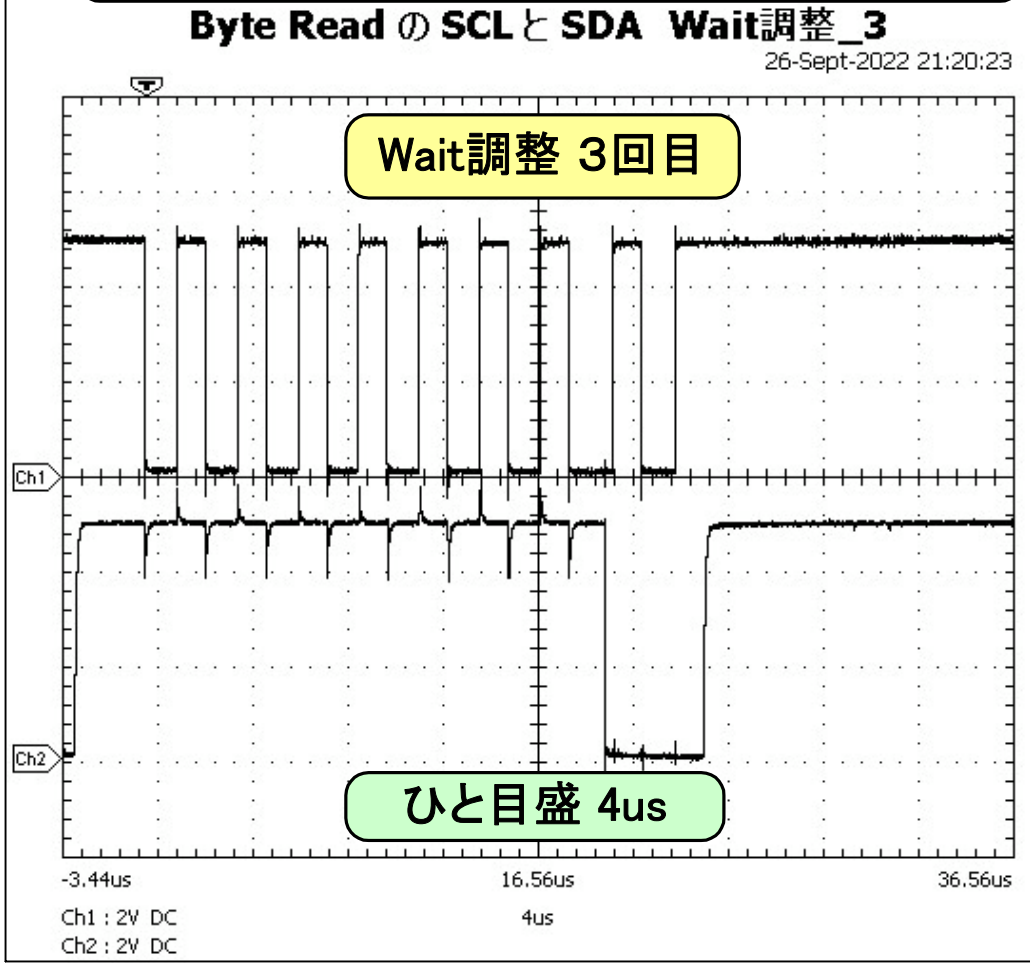
局部的に狭いパルス幅等の微調整1



局部的に狭いパルス幅等の微調整2



この波形は、Read時の波形です。
SCLパルスが Hiのタイミングで
データを、読み込みます。



C言語ソースで、具体的な Wait調整は、

```
// マイクロ秒レベルの時間待ち処理
#define  WAIT_0  for( jk=0; jk<2; jk++ )
#define  WAIT_1  for( jk=0; jk<6; jk++ )
#define  WAIT_4  for( jk=0; jk<4; jk++ )
#define  WAIT_5  for( jk=0; jk<3; jk++ )
```

上記、for文のマクロ定義を行い、以下の様に使用しました。

```
void i2c_start_cond( void )
{
     Uchar  jk;

  TP_D1_C1;  // SCL = H , SDA = H
  WAIT_1;    // μs オーダーの  時間待
  TP_D0_C1;  // SCL = H , SDA = L
}
```

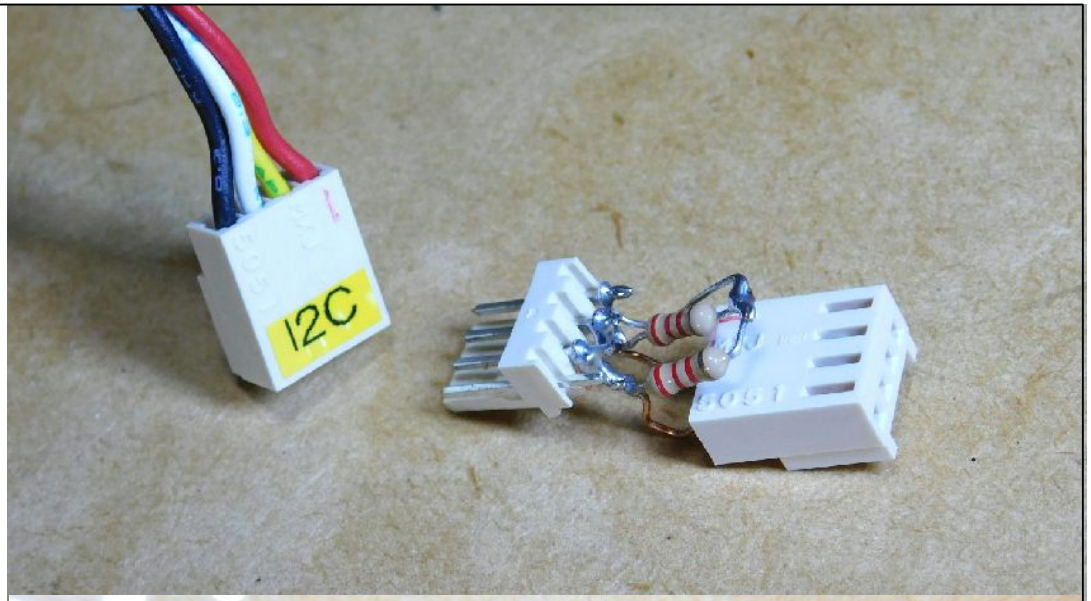
TP_D1_C1等のマクロは ソースを確認して下さい。
やはり、for文の空ループでは、待ち時間の分解能が荒いので、出来れば アセンブラで NOP命令を並べて調整したいです。

順番が、前後しますが

右上の画像は、ポカミスで、I2Cバスの SCL、SDA に、プルアップ抵抗を入れ忘れてました。

後で、気付いたので 基板上に プルアップ抵抗を、入れるスペースも無かったので、中継のコネクタを作り、そこにプルアップ抵抗を入れました。 Vccと GNDの配線も通していたので、プルアップ抵抗が付けられたという事です。

見た目が、みっともないですね。



右下の画像は、上側の ICが、SPIインタフェースの MCP23S17です。 下側の ICが、I2Cインタフェースの MCP23017 です。

どうでもいい事ですが、MCP23S17の文字が薄くなっているのは、MCP23S17を、先に使いだした関係で、その分 触ってるからではないかと思います。