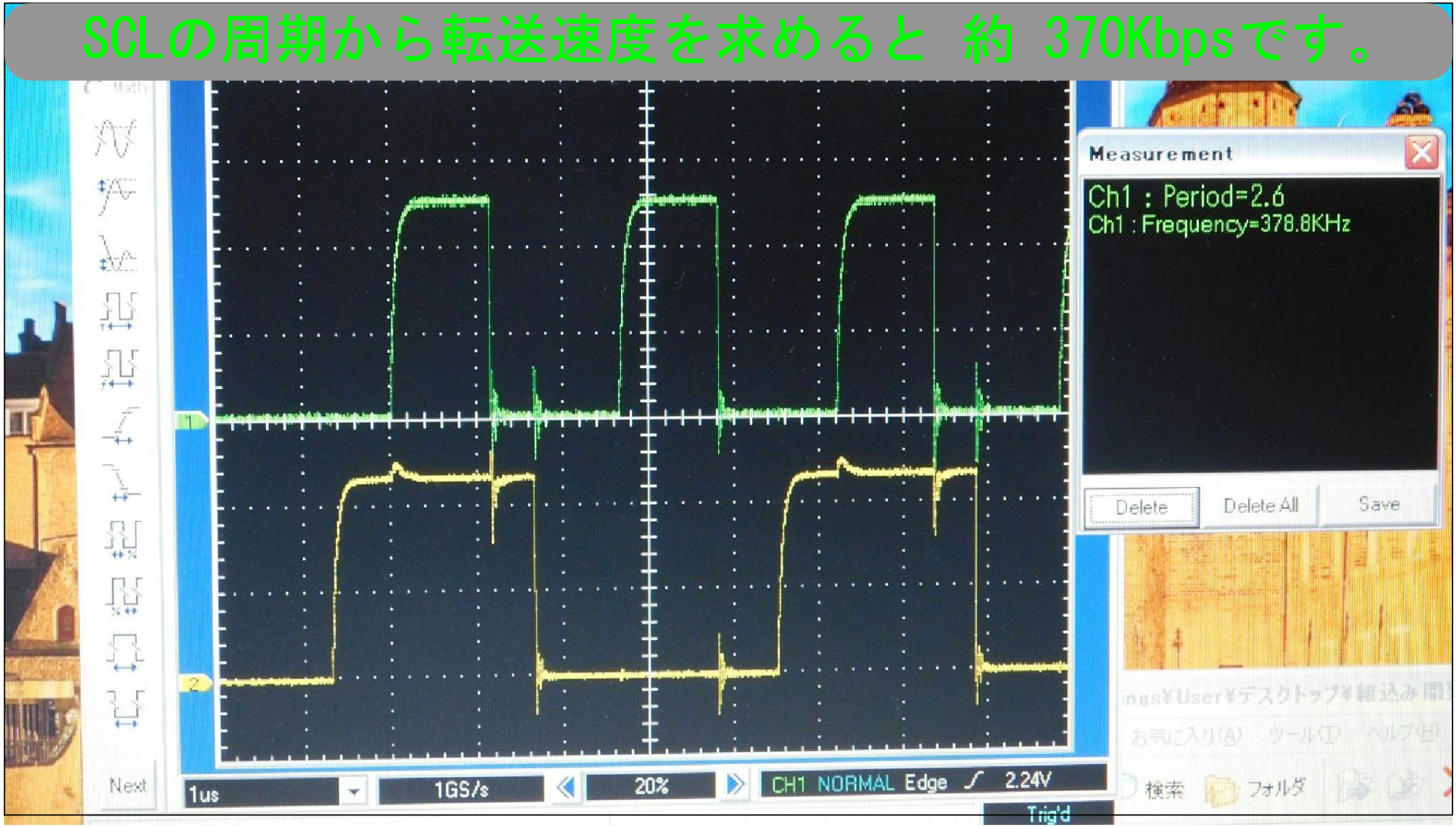
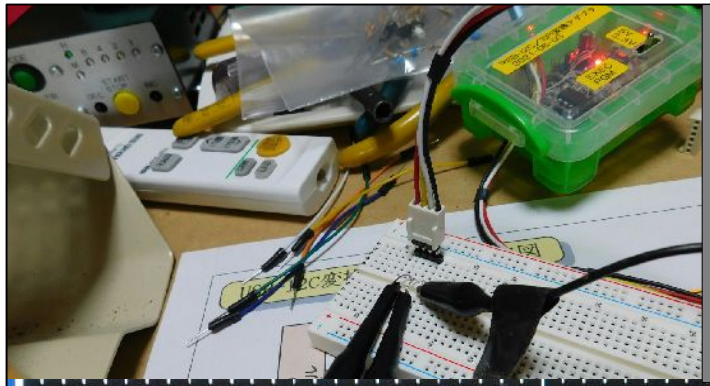
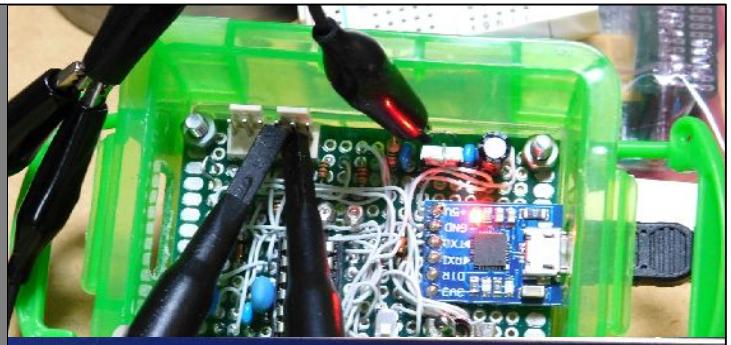

SCL、SDAの信号測定を行います。
ソフトが完成しないと信号が出ないので
ソフト完成後の測定になります。
I2C信号引き出し用の 40cmのケーブル末端の
コネクタ部分の信号を測定します。

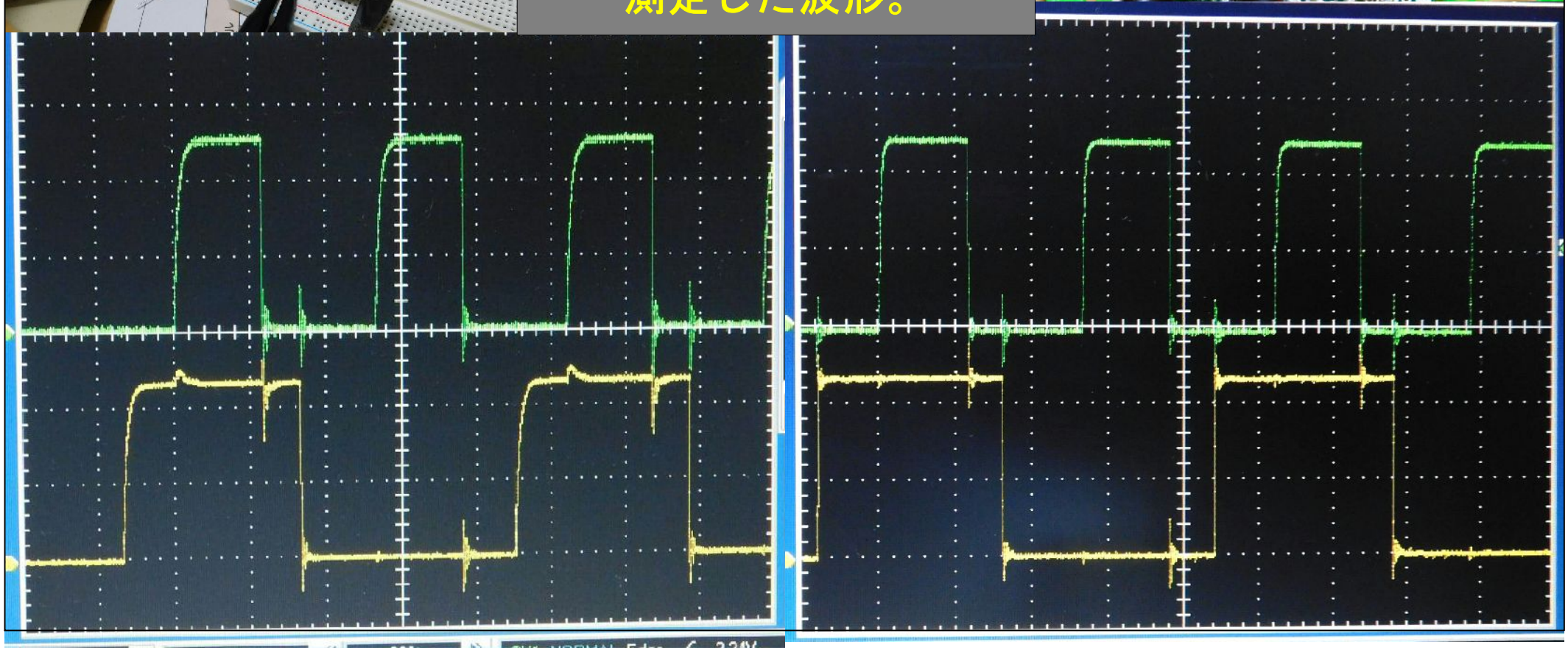
SCLの周期から転送速度を求めると 約 370Kbpsです。



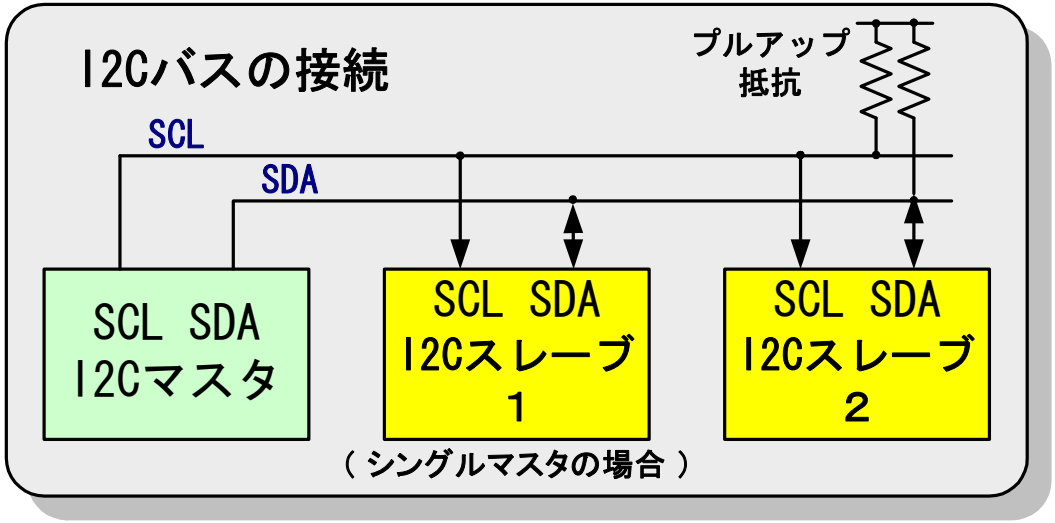


左が40cmのケーブルを
通して測定した波形。
右がアダプタ基板から
信号を取り出して
測定した波形。





I2Cインタフェースについて



I2Cバスは、SCL（シリアルクロック信号）とSDA（シリアルデータ信号）の、２本の信号線で構成されています。

SCL信号は、マスタから出力されるクロック信号で、スレーブは、この信号に合わせたタイミングで、データの送受信を行います。

SDA信号は、双方向の信号線で、その時の状況に応じて SDAの信号の方向が変わります。 マスタが 特定のスレーブにデータを、書き込む時は、Write信号として マスタからスレーブに、データが送られます。

マスタが 特定のスレーブからデータを読み出す時は、Read信号として スレーブからマスタへ、データが送られます。

今回は、USB-I2C変換アダプタ （ R8C/M110Aマイコン ）が I2Cの マスタになります。 今回、プログラムで I2Cの通信機能を作成するので、I2Cのプロトコルを しっかり理解する必要があります。

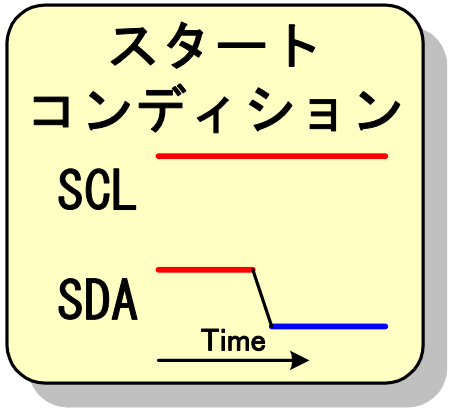
今回は、I2Cのサブセットとして、I2Cアドレスは 7bit固定、通信速度は 400[Kbps]以下の シングルマスタという基本的な構成を 実現します。

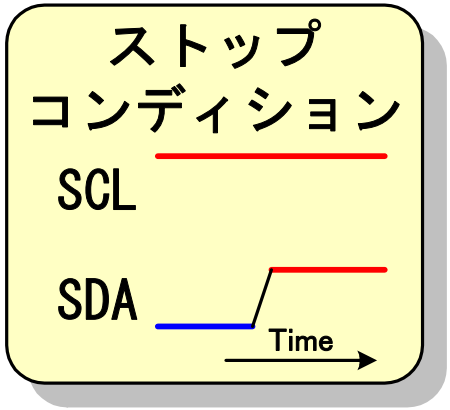
I2C通信シーケンス(1)

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。

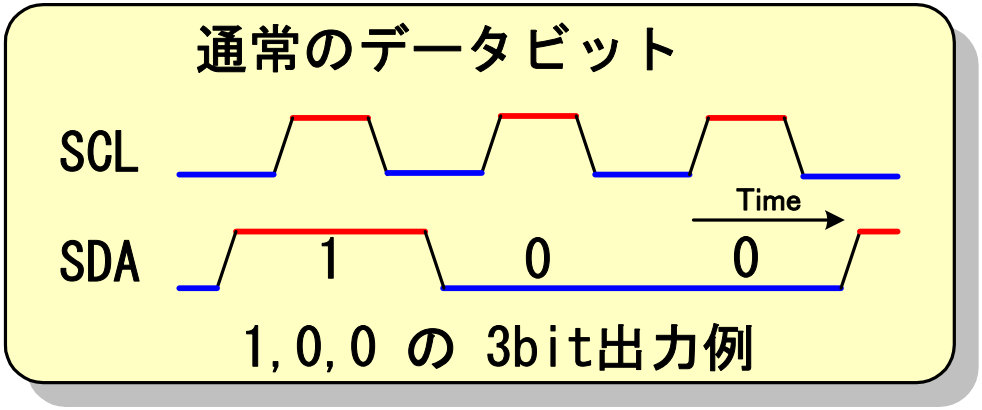
[1] スタートコンディション：
今から通信シーケンスを開始する事をマスタが、スレーブに通知するための信号です。 SCLが、Hiの期間中に SDAを Hiから Lowに変化させます。

[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。

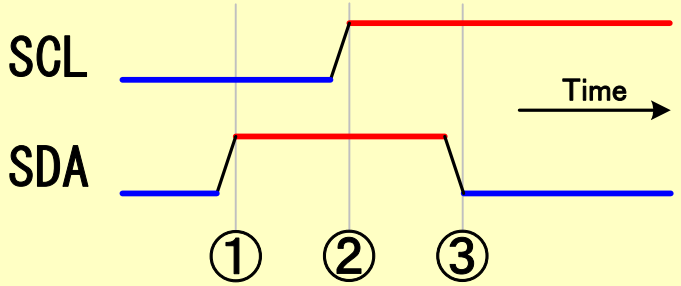


I2C通信シーケンス(2)

[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

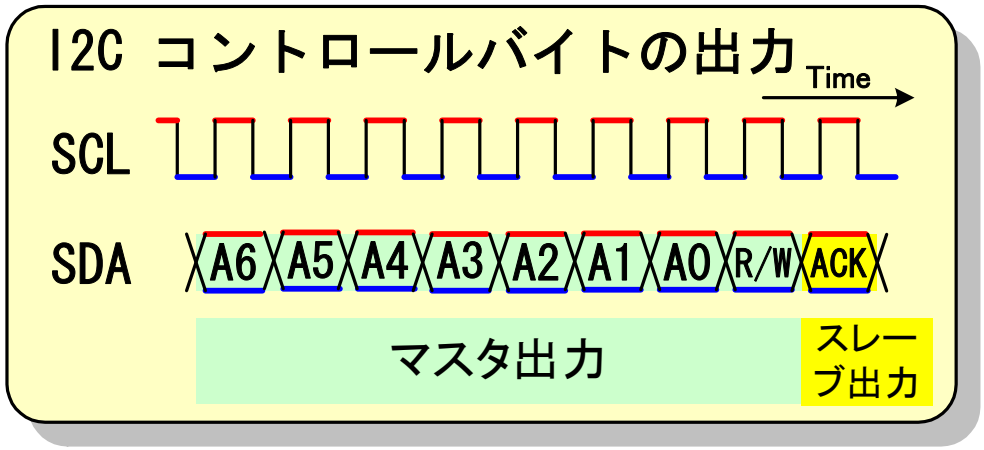


最近は、殆どのマイコンに、データ用フラッシュROMが入っている事もあり外付けで 8pinのシリアルEEPROMを使う事が、少なくなってきました。
　これにより、リピートスタートコンディションを使う機会も減ったように思います。

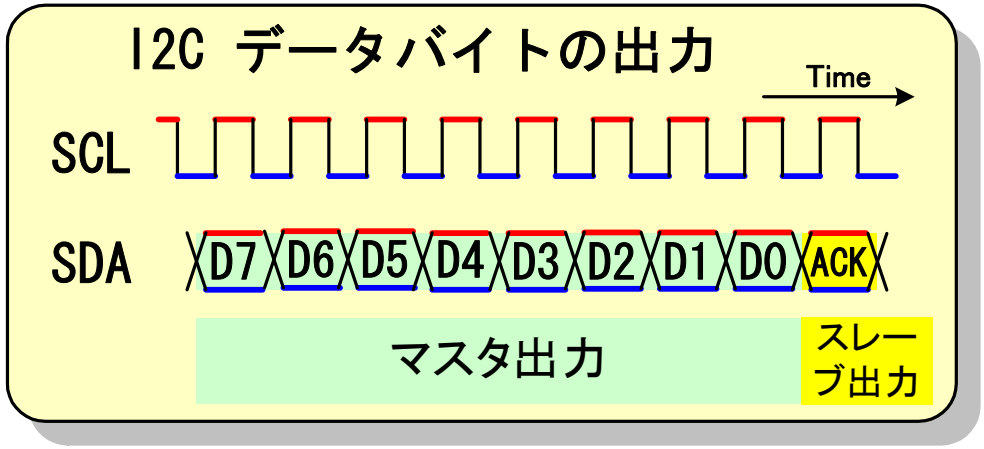
I2C通信シーケンス(3)

[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。

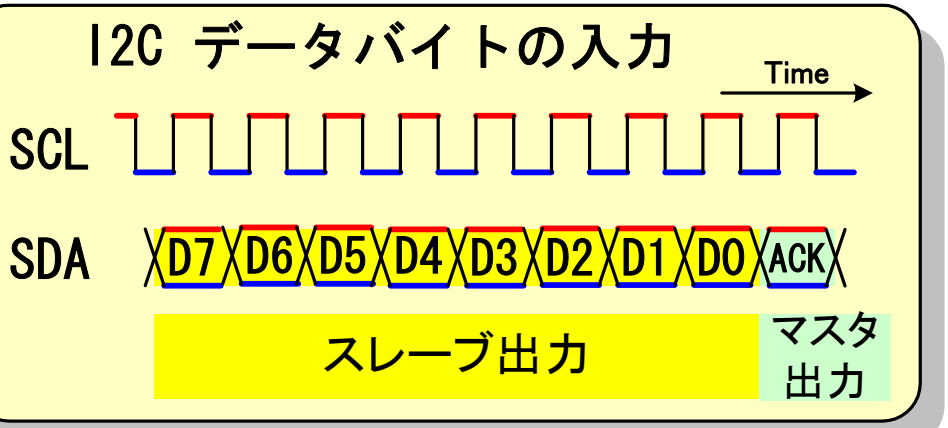


[5] データバイト出力（Write）：
内容(データ)が異なるだけで、コントロールバイト出力と同じです。



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通信のソフト開発

[8] I2C通信の基本的なシーケンスの説明を行いましたので、今回のUSB-I2Cアダプタのソフト開発に関して概略を説明します。

最初は マイコン側だけのソフトを作成してパソコン側は フリーのターミナルソフトを 使えないかと考えてました。
マイコン側で I2Cルーチンのデバッグを行って行くうちにI2Cデバイスの初期化に 手間がかかる事が見えてきました。 これらを都度ターミナルソフトで打ち込むのは大変だ。と、思いました。

という事で方針転換をして、今回のUSB-I2Cアダプタ用の PC側ユーティリティソフトを作る事にしました。
パソコン側の開発環境を使うのは、久々であった事もありソフト開発に 時間がかかってしまいました。 最初のパソコンとマイコン間の伝送手順の取り決め、転送するコマンド群のスクリプトファイルの仕様等を決めます。 終わり方PC側シリアル受信処理で バグ取りで手間取りました。

開発環境は、マイコンが ルネサスのHEWで、パソコンが 旧バージョンのDelphiです。 C+Asm と ObjectPascalの 混在環境で プログラム作成をしました。

[9] パソコン側と、マイコン側の役割分担:
今回マイコン側では

① USBシリアルで送られてきた 先頭 `W'の Writeコマンドを
I2Cデバイスに書き込む。 その後、PCに ACKを 返送する。

② USBシリアルで送られてきた 先頭 'R'の Readコマンドは
I2Cデバイスから 指定されたByte数、データを読み込み USBシリアルで
ＰＣに送る。

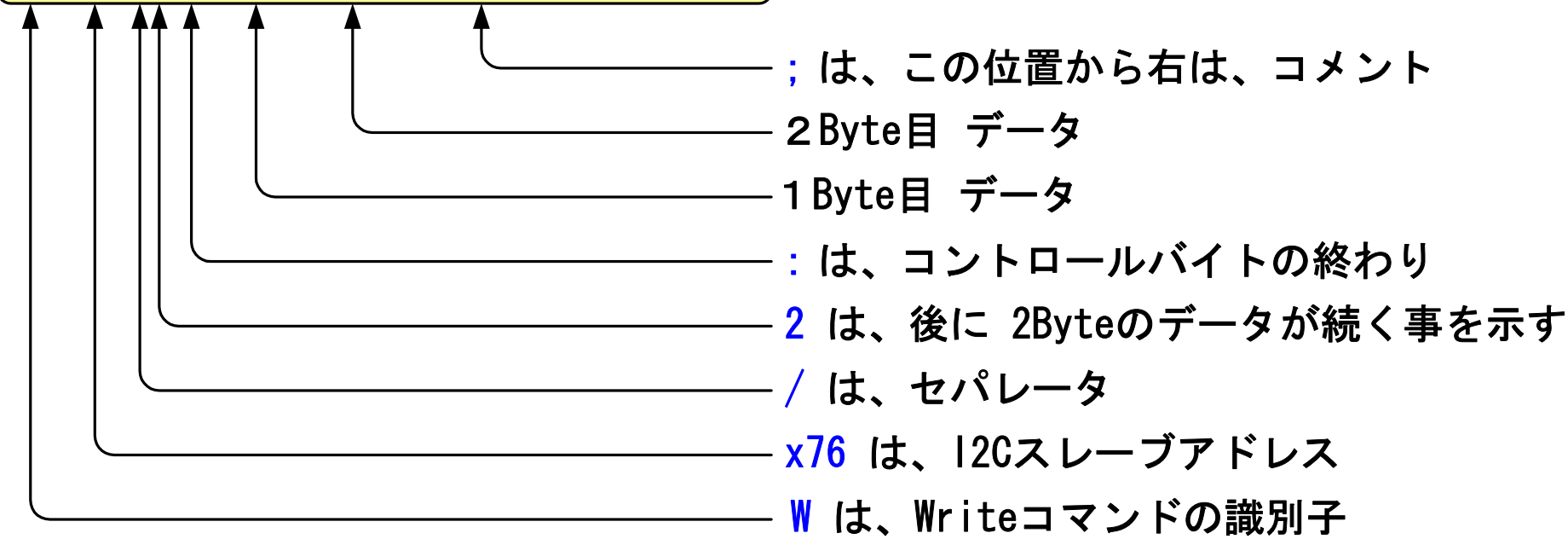
③ USBシリアルで送られてきた 先頭 'D'の Delayコマンドは、マイコン側で
指定されたミリ秒単位の時間が 経過したら PCに ACKを 返送する。

基本この ３つの機能を受け持ちます。

[10] Wコマンドの例：

```
Wx76/2: xF2 x01      ; 初期化
```

PC側ユーティリティ１行のスクリプト



; は、この位置から右は、コメント

２Byte目 データ

１Byte目 データ

: は、コントロールバイトの終わり

2 は、後に 2Byteのデータが続く事を示す

/ は、セパレータ

x76 は、I2Cスレーブアドレス

W は、Writeコマンドの識別子

マイコンに送るバイナリコマンドは

57h	76h	02h	F2h	01h

左の５Byteになる。（ 先頭の 57hは 'W' の ASCIIコード ）

コマンド完了で、ＰＣに ACK 06h を返送する。

[11] Rコマンドの例：

```
Rx76/4:        ; データ読み出し
```

PC側ユーティリティ１行のスクリプト

; は、この位置から右は、コメント

: は、コントロールバイトの終わり

4 は、I2Cスレーブから、データを読出すByte数

/ は、セパレータ

x76 は、I2Cスレーブアドレス

R は、Readコマンドの識別子

マイコンに送るバイナリコマンドは

52h	76h	04h

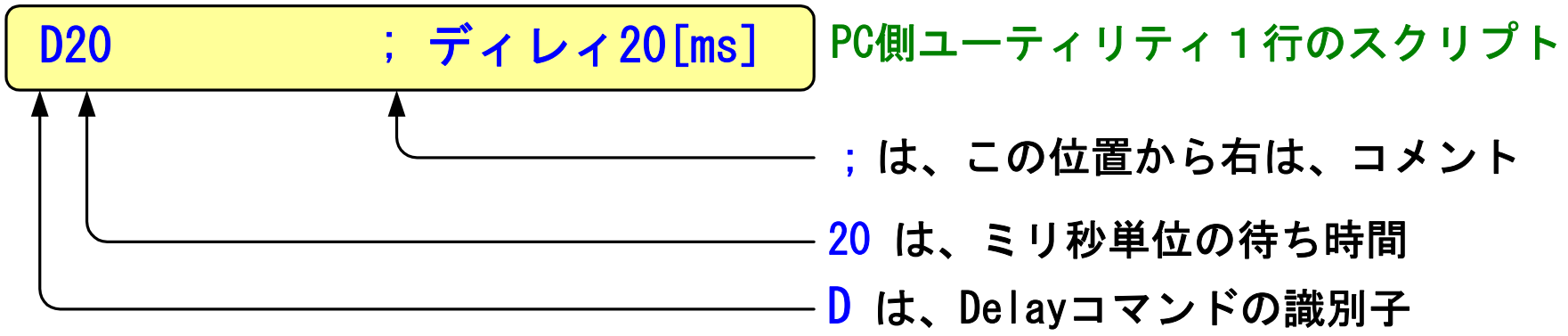
左の３Byteになる。（ 先頭の 52hは 'R' の ASCIIコード ）

その後、I2Cスレーブから、4Byteデータを読み出す。

01h	02h	03h	04h

読み出したデータを USBシリアルで、ＰＣに送信する。
（ Readの場合は、ACK は 送りません。）

[12] Dコマンドの例：



マイコンに送るバイナリコマンドは

44h	14h

左の２Byteになる。（ 先頭の 44hは 'D' の ASCIIコード ）

指定遅延時間 経過後、ＰＣに ACK 06h を返送する。

16×2 OLED表示器のコマンド

```
;[1]:初期化
D100
Wx3C/2: x00 x01     ; コマンド、画面消去
D20                 ; 20ms 待ち
Wx3C/2: x00 x02     ; コマンド、HOME位置へ
D2                  ; 2ms 待ち
Wx3C/2: x00 x0F     ; コマンド、表示開始
D2                  ; 2ms 待ち
Wx3C/2: x00 x01     ; コマンド、画面消去
D20                 ; 20ms 待ち
```

```
;[2]:１行目文字列
Wx3C/2: x00 x02     ; コマンド、HOME位置へ
D10                 ; 10ms 待ち
Wx3C/2: x40 'M'     ; データ、１文字表示
D1                  ; 1ms 待ち
Wx3C/2: x40 'I'
D1
Wx3C/2: x40 'C'
D1
Wx3C/2: x40 'H'
D1
Wx3C/2: x40 'I'
D1
Wx3C/2: x40 'K'
D1
Wx3C/2: x40 'U'
D1
Wx3C/2: x40 'S'
D1
Wx3C/2: x40 'A'
D1
```

```
;[3]:２行目先頭ロケート
Wx3C/2: x00 xA0     ; ２行目先頭ロケート
```

```
;[5]:輝度設定
Wx3C/2: x00 x2A
D10
Wx3C/2: x00 x79
D10
Wx3C/2: x00 x81
D10
Wx3C/2: x00 xFF
D10
Wx3C/2: x00 x78
D10
Wx3C/2: x00 x28
D10
```

```
;[4]:２行目日付
Wx3C/2: x40 '2'
D1
Wx3C/2: x40 '0'
D1
Wx3C/2: x40 '2'
D1
Wx3C/2: x40 '1'
D1
Wx3C/2: x40 '-'
D1
Wx3C/2: x40 '0'
D1
Wx3C/2: x40 '6'
D1
Wx3C/2: x40 '-'
D1
Wx3C/2: x40 '1'
D1
Wx3C/2: x40 '3'
D1
```

センサBME280のコマンド

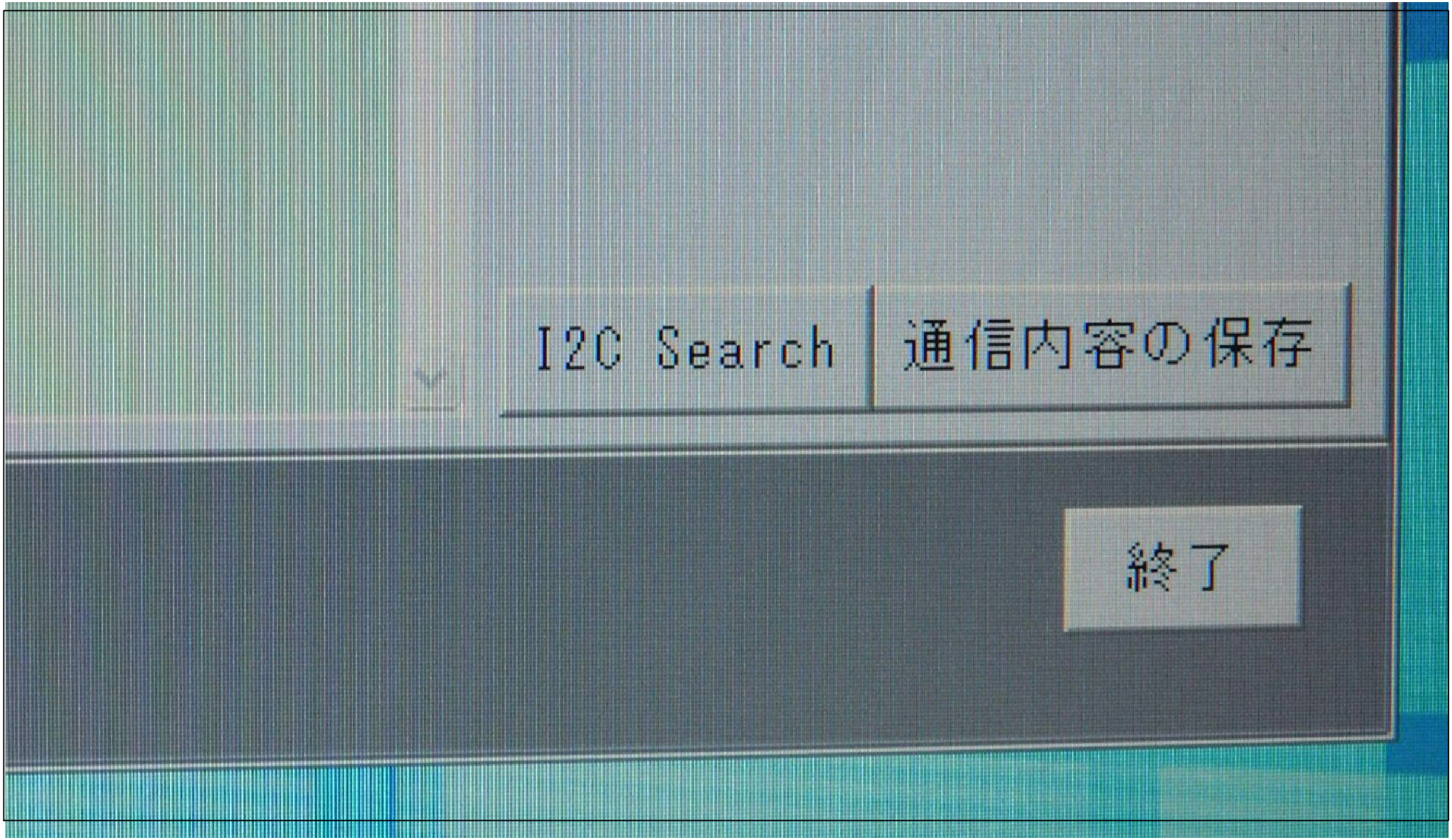
```
;[1]:初期化
Wx76/2: xF2 x01      ; 初期化(1)
D20
Wx76/2: xF4 x27      ; 初期化(2)
D20
Wx76/2: xF5 xA0      ; 初期化(3)
D20
```

```
;[3]:測定データ取出し
Wx76/1: xF7      ; 測定データを送る指定
D20
Rx76/8:            測定データ 8byte Read
D20
```

```
;[2]:トリムデータ読出し
Wx76/1: x88      ; ？トリムデータ先頭
D20
Rx76/24:         ; トリムデータ 24byte Read
D20

Wx76/1: xA1      ; ？次の 1byteトリム指定
D20
Rx76/1:          ; トリムデータ 1byte Read
D20

Wx76/1: xE1      ; ？最後の 7byteトリム指定
D20
Rx76/7:          ; トリムデータ 7byte Read
D20
```


I2C Search
通信内容の保存
終了

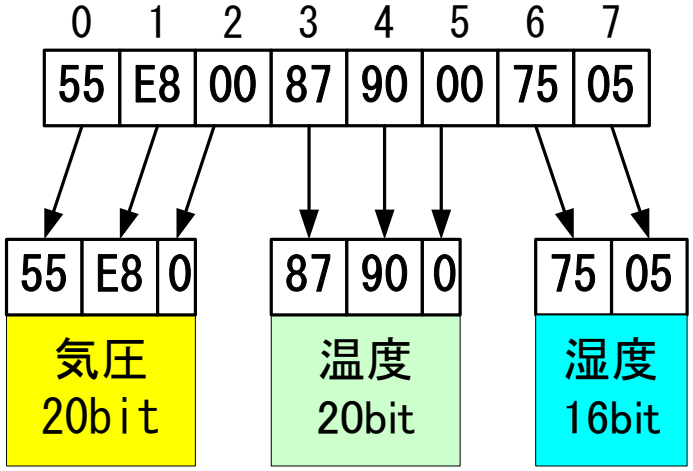
BME280(温度、湿度、気圧センサ)の読み出しデータ

BME280の読み出しデータは、通常の測定データ以外に、トリムデータと呼ばれる物があります。
トリムデータは、計:32byteのデータです。　測定データは、8byteです。
読み出した、測定データを、トリムデータを 系数にしたキャリブレーション演算をしないと、本来の温度、湿度、気圧の値にならないようです。 それと電源ONから、最初の数十秒は、測定値が安定しないとの事です。　では、実際のセンサから読み取ったデータのサンプルを示します。
連続して測定データを読み出す時は、1秒以上の間隔を開けて取り出して下さい。

```
* BME280 TrimData Dump.
A7, 70, 50, 69, 32, 00, 1F, 8D, 4B, D7, D0, 0B, 1A, 1E, BB, FF,
F9, FF, AC, 26, 0A, D8, BD, 10, 4B, 60, 01, 00, 15, 29, 03, 1E,
* BME280 Muaseres Data Dump.
55, E8, 00, 87, 90, 00, 75, 05,
* BME280 Muaseres Data Dump.
55, EC, 00, 87, 93, 00, 75, 0F,
* BME280 Muaseres Data Dump.
55, EB, 00, 87, 91, 00, 75, 3F,
```

BME280 8byteの測定データの内訳

測定データサンプル／先頭から 3byteが気圧、次の3byteが温度、最後の2byteが湿度です。
2byte目と5byte目のデータは、上位4bitを採用。



ちなみに、今回の BME280では、2byte目と5byte目のデータは 常時 00 です。　将来的に分解能の高いセンサデバイスが出来た時を想定して 20bitにしているのかもしれません。

BME280 32byteの トリムデータに関して

32Byteのトリムデータですが、

① 先頭の6Byteが、
2Byte整数 3個の温度係数

② 次の18Byteが、
2Byte整数 9個の気圧系数

③ 次の 8Byteが、
6個の湿度系数

になっています。 これらの係数を用い温度、湿度、気圧のキャリブレーション計算を行う事になってます。 ここでは演算処理の説明は 省略します。

スイッチサイエンスというサイトで今回の BME280の資料を見つけました。
（ 興味のある方はそちらを参照して下さい ）


マイコン
マイコンの1線 2線 3線
インターフェース活用入門
Introduction to microcomputer interface use to 1-Wire, SPI, I²C
中尾 司 著
PICとH8で具体的な
1-Wire, SPI, I²Cプログラミングを行う
CQ出版社


USB-I2C／SPI変換アダプタ
2021-06-03
5V
3.3V
EXEC
PGM
ミチクサショクニン - TAKE