前回の基板　マイコンによる検査

前回、時間が押して出来ませんでしたが、H8/3069Fマイコンから、I2Cの SCL、SDA そして追加の　LD、IRQ信号、及び ブザーの 計 5 本の 信号線から、簡易パルス出力を 行い信号が　正常にだせるかの確認を 行います。

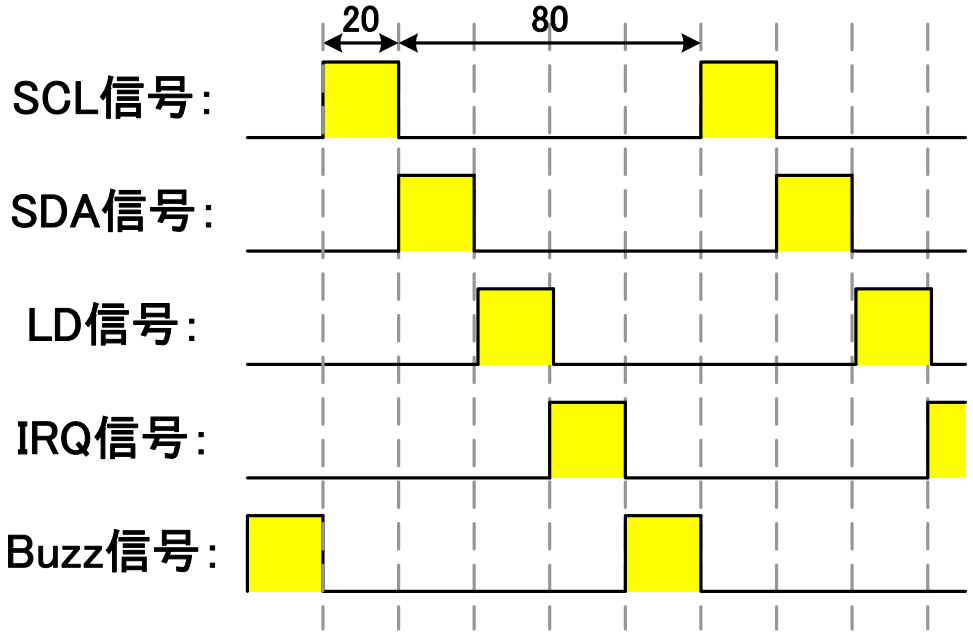
まずは、5本の信号線のポート番号を確認します。

分類	信号名	Direction	ポートNo	機能	ピン番号
I2C	SCL	出力	Port6/b0	I/Oポート	CN1/1
	SDA	入出力	Port6/b1	I/Oポート	CN1/2
追加	LD	出力	Port6/b2	I/Oポート	CN1/3
	IRQ	入力	Port9/b4	IRQ4	CN2/3
	Buzz	出力	PortA/b4	TIOCA1	CN1/29

信号名　SCL、SDA、LD は　I/Oポートに 設定して使用します。 その中で　SDAは　双方向のデータのやり取りを 行います。 IRQは　H8マイコンの　IRQ4 割り込み受付け信号として設定します。 Buzzは　圧電ブザー出力ですが、1KHzから　4KHzぐらいの周波数で　ブザーを鳴らします。 この関係で　H8の TCU 16bitタイマーのチャネル1を 使用する予定です。

しかし、今回の信号線ハードが 正常に 機能するか調べる用途では、マイコン側にて　全てを 出力ポートに設定して、全ポートに　異なる位相の パルス出力を 行い 互いのポートで 干渉する事が無いか　確認を行います。

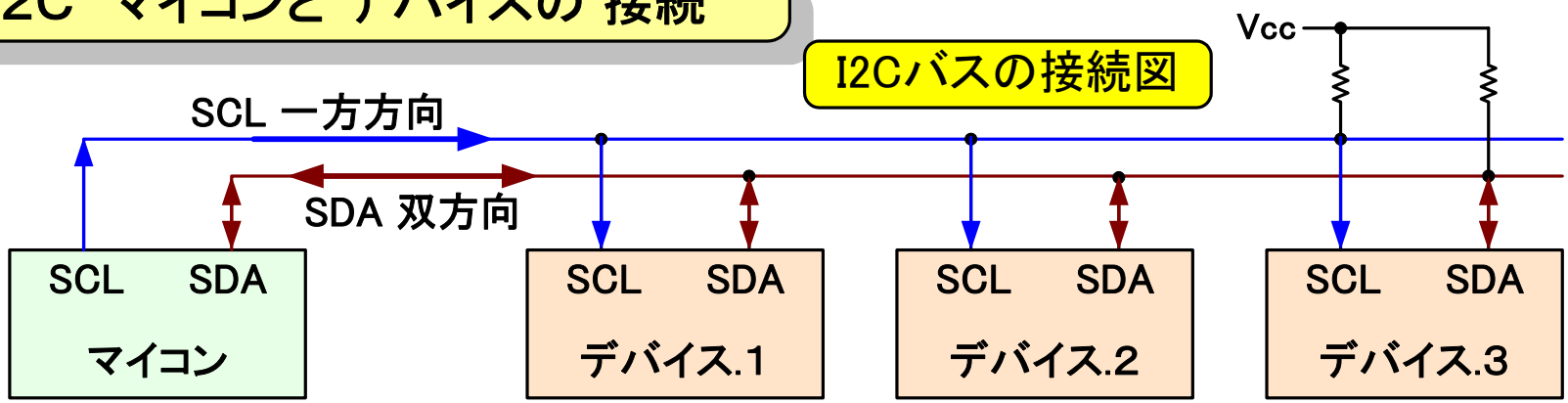
分類	信号名	Direction	ポートNo	ピン番号
I2C	SCL	出力	Port6/b0	CN1/1
	SDA	入出力	Port6/b1	CN1/2
追加	LD	出力	Port6/b2	CN1/3
	IRQ	入力	Port9/b4	CN2/3
	Buzz	出力	PortA/b4	CN1/29



マイコン側から出力する波形は SCL端子の信号を 基準とし、他の信号波形の位相がどの程度遅れているかで 確認します。

右上の信号波形を H8マイコンから 出力します。 各信号のパルス幅は、デューティ 20 です。 各信号の位相差は SCL、SDA、LD、IRQ、Buzzの順で デューティ 20の 幅で 遅らせています。

I2C マイコンと デバイスの 接続



I2Cは SCL:シリアルクロック信号、SDA: 双方向データ信号の 2本の信号線で構成されます。 標準的な転送速度は 400Kbpsです。 デバイスによっては、更に早い物もあります。 また、Read/Writeコマンドに 7bitのアドレスも付くので、2線に アドレスの異なる複数のデバイスを接続する事が出来ます。

プルアップ抵抗は必要ですが、デバイスに内蔵されている場合も ありま す。 その場合は、デバイスの複数接続を考慮して やや高めの抵抗値にしてあります。

I ２ C	
長所	① 信号線2本＋GNDでデバイスと通信可能。 ② 複数デバイスを接続する事が可能。 複数デバイスを接続しても、信号線は2本のままで、OK。 但し 各デバイスのアドレスは、重複させてはならない。
短所	① SPI と比べると データ転送速度が遅い。

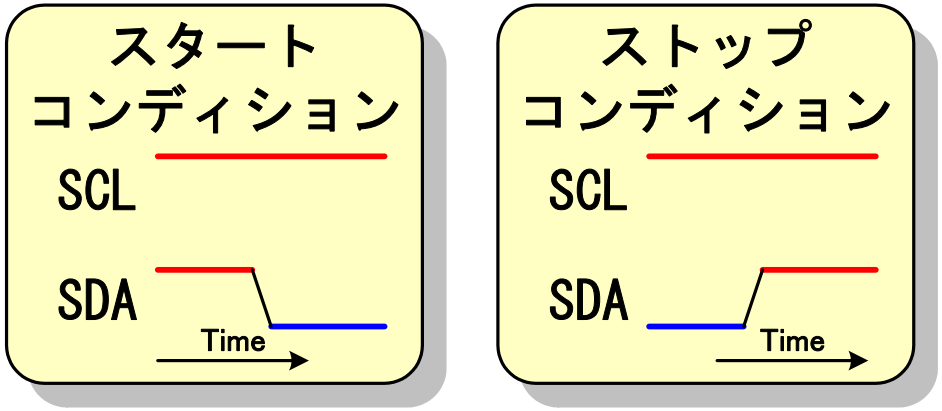
という事で、データ転送速度が あまり問題にならなければ、I2Cは 気軽に使えると思います。

I2C通信シーケンス(1)

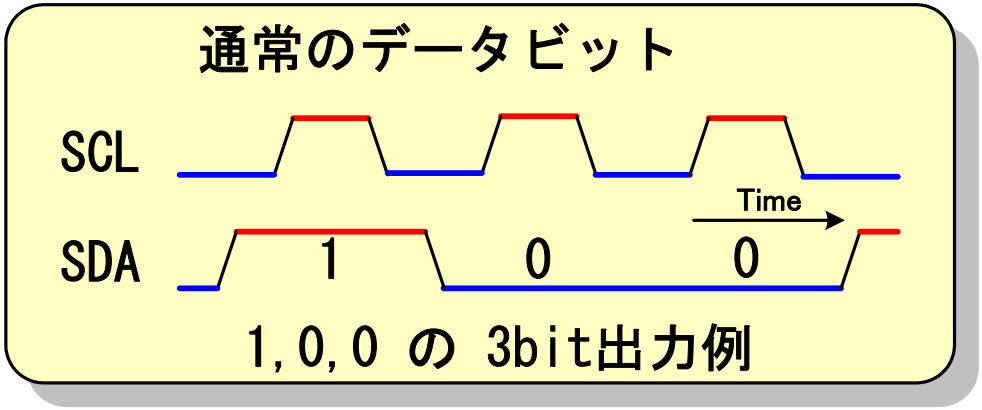
I2C通信は、SCLと SDAの２本の信号線を用います。待機中 SCLと SDAは、両方とも Hiレベルです。

[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。

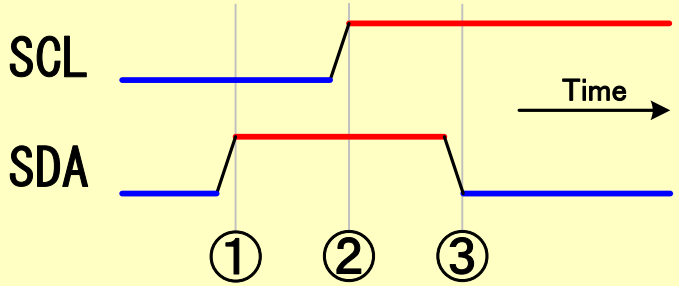


I2C通信シーケンス(2)

[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



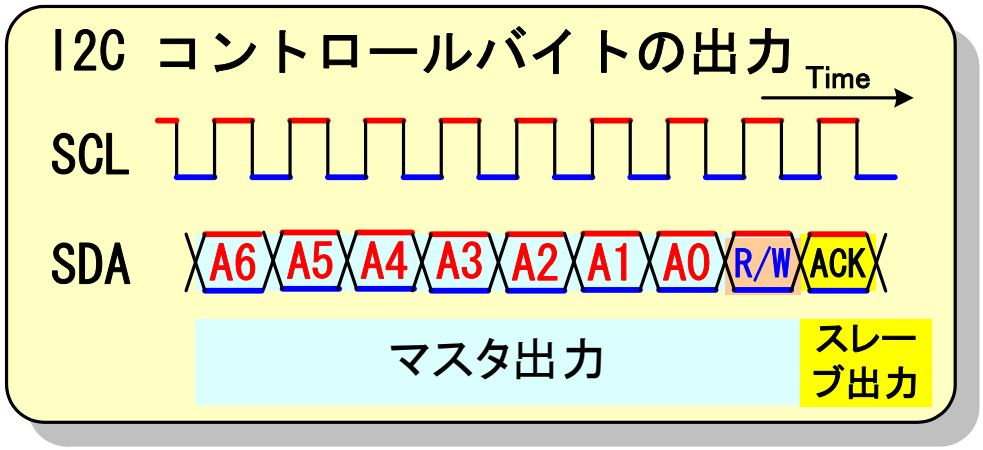
最近は、殆どのマイコンに、データ用フラッシュROMが入っている事もあり
外付けで 8pinのシリアルEEPROMを使う事が、少なくなってきました。
　これにより、リピートスタートコンディションを使う機会も減ったように思います。

　しかし、まだリピートスタートコンディションが 必要なデバイスが、一部存在します。 殆どの場合、コマンドWriteから、データ Readに 連続して切り替える用途で 使われます。

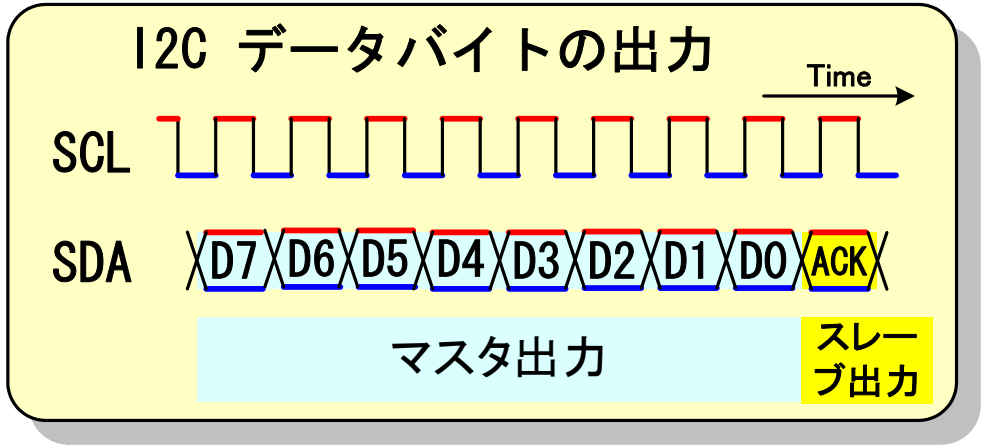
I2C通信シーケンス(3)

[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。

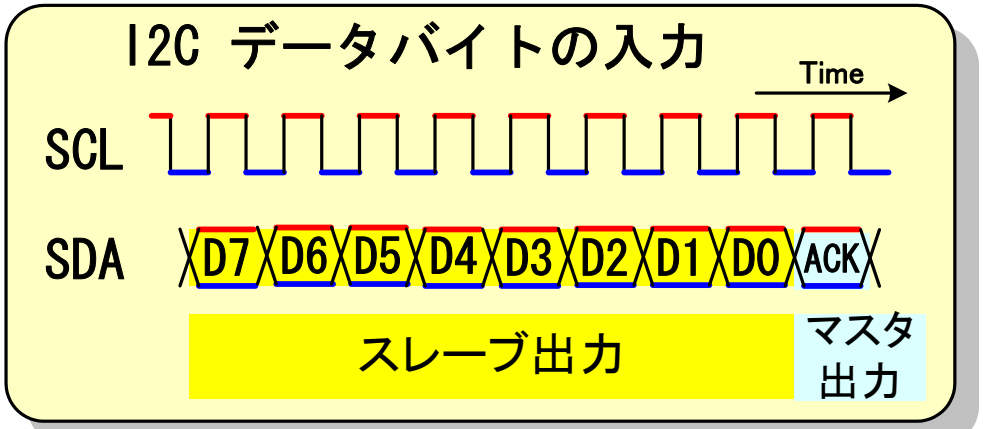


[5] データバイト出力（Write）：
内容(データ)が異なるだけで、コントロールバイト出力と同じです。



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。

今回の I2Cプログラムの 構成

最初、I2Cアクセスプログラムのローレイヤー部分は、アセンブラで作ろうと思っていましたが開発途中 アセンブラで ある機能(マクロ宣言内で 使えるローカルラベル)が サポートしてなかった事、その他 細々とした事が有り、R8Cのアセンブラソースの移植は諦めました。

アセンブラで プログラムを作成する場合は、R8Cの方が 作りやすい要素が、ありました。
HEWの H8マイコンのアセンブラは、やや古い仕様で ちょっと扱い難い感じを受けました。

私が、マクロを多用してプログラムを作成していたので、そうなってしまったという事もあります。 仕方ないので ローレイヤのプログラムを新規に C言語で 作成し直しました。

仕様的に どのような物を作ればいいのかは把握していたので、半日ほどで出来ました。
ファイルの構成は、以下の3本に なります。

① i2c_packet.h (I2C関数のプロトタイプ宣言)

② i2c_packet.c (I2C 上位層 関数の実装)

③ H8_i2c_tcc.c (I2C 下位層 関数の実装)

①と ②は 過去に作成したR8C百円マイコンの ソースを ほぼ そのまま使用しています。
③が、今回 H8に対応するために 新たに作ったI2Cローレイヤーの ソースです。

どのような関数が あるのか、①の ヘッダーファイル中身を お見せします。 ②、③は ソースの記述量が やや多いので 説明は省略します。

i2c_packet.h

```
//  I2C Port( ハードに近いレイヤ関数 )  関数プロトタイプ宣言
// ------------------------------------------------------------
void  init_i2c_port( void );              // ポート初期設定
void  i2C_start_cond( void );             // スタートコンディション
void  i2c_stop_cond( void );              // ストップコンディション
void  i2c_rep_start_cond( void );         // リピートスタートコンディション
_UBYTE      i2c_rd_adr7( _UBYTE adr );    // I2C/1byte読み込み adr = 7bitアドレス
_UBYTE      i2c_wr_adr7( _UBYTE adr );    // I2C/1byte書き出し adr = 7bitアドレス
_UBYTE      i2c_send_byte( _UBYTE dt );   // 1 byte I2C 送信 ( 関数値は、相手からの ACK/NAK )
_UBYTE      i2c_recv_byte( void );        // 1 byte I2C 受信 ( 継続 受信時 )
_UBYTE      i2c_recv_byte_final( void );  // 1 byte I2C 受信 ( 最終byte 受信時 )
//  ( I2C_Packet.c ) 上位レイヤ関数   関数プロトタイプ宣言
// ------------------------------------------------------------------------
_UBYTE  i2c_check_slave( _UBYTE adr );                    // 指定アドレスの スレーブ有無の確認
void  i2c_write_7b( _UBYTE adr, _UBYTE* buf, short len );      // 指定Slaveへ、データ書き込み
void  i2c_write_7bfix( _UBYTE adr, _UBYTE fdt, _UBYTE ptn, short len ); // 固定Byte付き データ書き込み
void  i2c_write_7bfv( _UBYTE adr, _UBYTE fdt, _UBYTE *ptr, short len ); //固定Byteとポインタでの書込み
void  i2c_read_7b( _UBYTE adr, _UBYTE* buf, short len ); // 指定Slaveからデータ読み込み
void  i2c_write_rep_7b( _UBYTE adr, _UBYTE* buf, short len ); // RS前半 Write
void  i2c_rep_read_7b( _UBYTE adr, _UBYTE* buf, short len );  // RS後半 Read
```

テストプログラム１

```
// 初期化処理は　省略しています
// ----------------------------
   while( 1 )
   {
      for( i=0; i<256; i++ )
      {
①        i2c_start_cond();     // I2C スタートコンディション
②        i2c_send_byte( i );   // I2C busへ　1byte 送信
③        i2c_stop_cond();      // I2C ストップコンディション
④        wait_ms( 500 );       // 0.5秒 待ち
      }
   }
```

ホワイルの無限ループ内に 0 から 255回の forループがあります。 そのループ内に

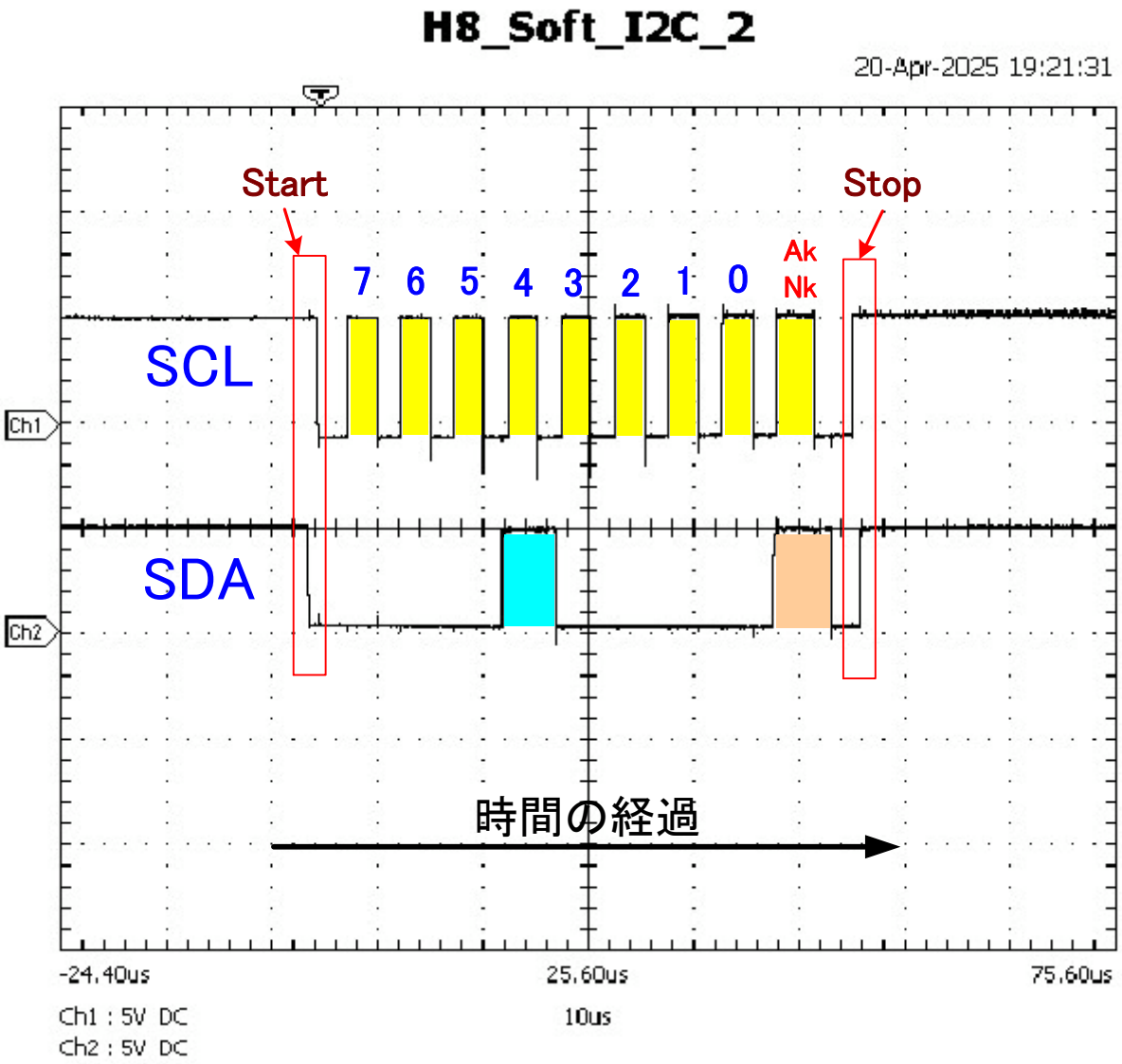
① スタートコンディションを出しています。

② I2Cバスに iの値を データとして 1byte送信します。

③ ストップコンディションを出しています。

④ 0.5秒の時間待ちを入れます。

上記テストプログラム1を 実行して、SCL信号、SDA信号を USBオシロで観測した波形の動画を お見せします。 その前に オシロ波形の表示に 関わる各部分が 何に相当するのかを説明します。



左のグラフの見方ですが、上が SCL 下が SDA です。 左の 縦長の赤四角で囲った部分が、スタートコンディションです。 このグラフでは 細かくて微妙ですが、SCLが Hiの状態で SDAを Hi から Lowに 降ろしてます。 この動作が スタートコンディションになります。

次に 8bitの データbitが 並びます。 データbit は b7 から b0 の順で 並びます。 その、後 赤で Ak Nk と書いてますが、ACK NAK の ビットです。

上側の SCKは Hi側が データを取り込むタイミングを 決める信号になります。 その関係で SDAに比べ 細みのパルスになってます。 SDAは データの信号なので、SCLのパルスに比べ、パルス幅が 広めになっています。 最後に ストップコンディションが あります。 SCLが Hiの期間中に SDAが Low から Hi に 変ります。

I2Cデバイスの　サーチ

```
// 初期化処理は　省略しています
// ----------------------------
  for( i=0; i<128; i++ )  // デバイスアドレス値のループ
  {
     c = i2c_check_slave( i );  // スレーブ応答の確認
     if( c == 1 )  // Cが 1なら デバイス有り
        sci_prin_byte_hex2_1( i ); // 16進2桁 表示
     else
        sci_prin_1( ".." );  // 無い時は ".."
     sci_prin_1( "  " );      // 表示の体裁調整
     if( (i % 16 ) == 15 )  sci_put_crlf_1();
  }
```

　I2C デバイスの　アドレス確認のサーチ処理です。
デバイスが あれば　そのアドレスを　16進 2桁で　表示します。 サーチ範囲は　0 から　127の範囲です。
　複数接続されていれば、その個数分　アドレスが表示されます。 ごくまれに　一つのデバイスで　2つのアドレスを 占有するデバイスも あります。