1番　RTC RX8900の　I2C アクセス

久々の　RTC RX8900のアクセスですが
かなり忘れているので、過去に作成した
「044 秋月電子 3種類の i2C接続 RTCレビュー」動画を　見てみました。

やや高機能な RTCで、設定用のレジスタや、通信電文も何種類かあり、044の動画で、細かく説明してあったので、
044の動画ファイルから、RX8900の 設定用レジスタと、通信電文の説明部分を
切り出す事にしました。

次のページから　13ページほど
RX8900の 説明が 続きます。

その前に　RX8900の特長を
右に 示します。

RX8900 特長

- 32.768 kHz 温度補償発振器(DTCXO)を搭載
  高精度
- I2C-Bus シリアル･インターフェース
- 曜,日,時,分のアラーム割り込み機能
- 定周期タイマー割り込み機能
- 時刻更新割り込み機能 (毎秒･毎分)
- 電源切り替え機能
- OE 機能付き 32.768 kHz 出力 (FOE, FOUT 端子)
- 自動うるう年補正機能 (2000 ～ 2099 年まで対応)
- 2.5 V ～ 5.5 V の幅広いインターフェース電圧範囲
- 2.0 V ～ 5.5 V の幅広い温度補償電圧範囲
- 低消費電流 0.70 μA / 3 V (Typ.)
- 1.6 V ～ 5.5 V の幅広い計時(保持)電圧範囲

RX8900アクセスにて 不具合発生

前回、スレーブアドレスのサーチがうまく行ったので、ほぼ、I2Cの動作は動くのではないかと思っていましたが、そう 甘く なかったです。

最初にテストしたのが、RX8900で ややアクセスが 面倒なデバイスだったのも 失敗でした。
最初に I2Cのテストを行うのであれば、16桁2行の LCD等を使用した方が いいです。 表示デバイスは I2Cで転送した文字列データを 表示してくれるので、データが 正常に 転送できているかの確認が やりやすいです。
下は、有機ELの SO1602A 緑表示 16桁 2行の デバイスです。

で、不具合発生とはなにかというと、データ転送が正常に出来なくて デバイスが 動かない、及びマイコン側が 不可解な動きを しました。

で、その前に、I2Cの SCL、SDAのバスの仕様ですが、2KΩ前後の抵抗で プルアップされてますよね。 で、SCLは マスターから出力される一方通行の信号です。 それに対し、SDAは データを送る信号で 書き込み、読み出しの 双方向になります。I2Cで 時として問題になるのは、マスター側と デバイス側の 信号を 出力から入力、入力から出力に切り替える タイミングなのです。

で、マスター側と デバイス側で 入出力の切り替えタイミングが 僅かにずれると 場合によってマスター側と スレーブ側とで、一瞬 両方出力状態になり、且つ マスター側が Hiで スレーブ側が Lowの場合、出力信号同士で 衝突し、一瞬ですが 大きな電流が流れます。

出力信号の衝突で、一瞬　大きな電流が流れると、そのショックで　正常な通信が出来なくなります。

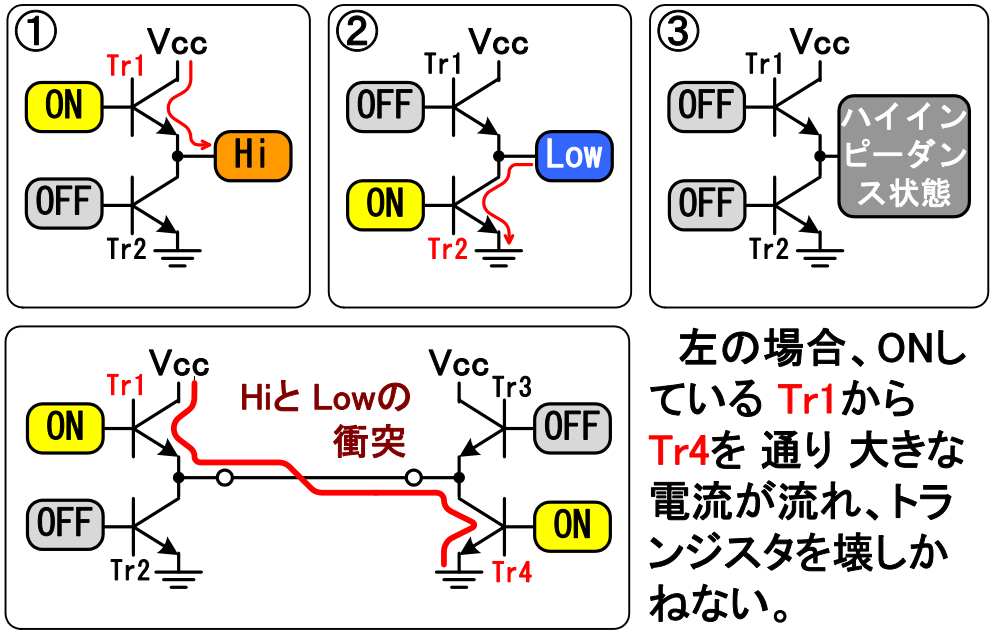
よって　I2Cの 規格に SDA信号の方向切り替え時に 瞬時 衝突が発生しても、支障が無いように　信号出力は　オープンコレクタで　行う事と決められています。

基本、オープンコレクタ（ 今は、MOS全盛なので　オープンドレインですけど ）は、信号を Low に落とすドライブ力は　強力ですが、Hiに　引き上げる能力は持ちません。 よって、プルアップ抵抗によって　弱く Hiレベルに　引き上げられます。 これにより　信号衝突の問題を回避しているという事です。

但し、このやり方では、信号が 緩やかに立ち上がるので、転送速度は　あまり速くは　出来ません。

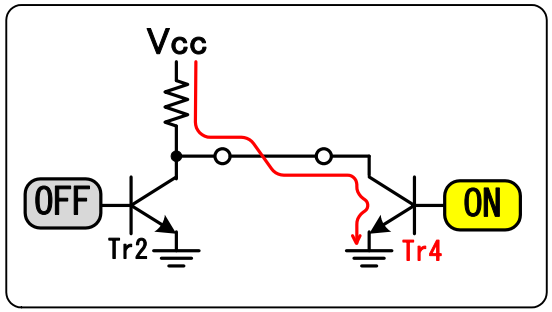
という事で　400Kbpsという標準速後が　規定されているのではと思います。

それに対し　マイコンの入出力ポートは　基本トライステートです。 3ステートとも　呼びますが　入出力端子が　①　Hiの状態、②　Lowの状態、③　ハイインピーダンス状態（ 信号入力時　使用する ）の　3つの状態を 持ちます。



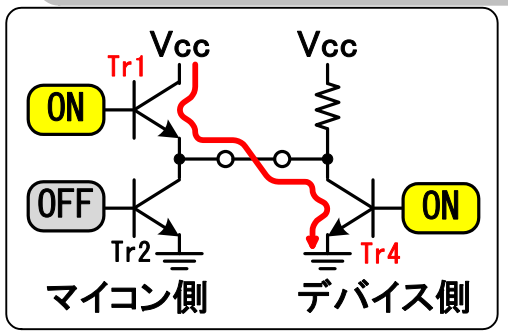
左の場合、ONしている Tr1から Tr4を 通り 大きな電流が流れ、トランジスタを壊しかねない。

オープンコレクタ同士の Hiと Lowの衝突



上に引っ張り上げるトランジスタが無いため プルアップ抵抗の制限を受けて電流が流れる。
問題無しです。

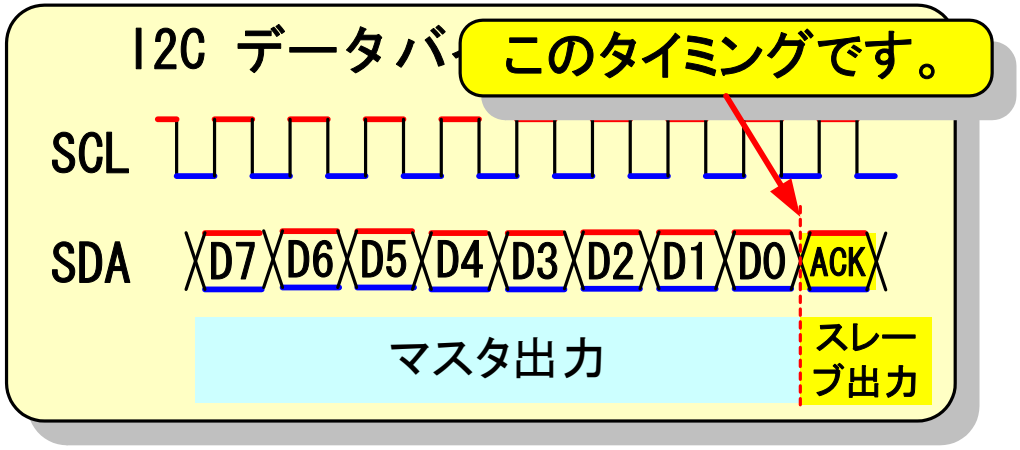
トライステートと オープンコレクタの 衝突



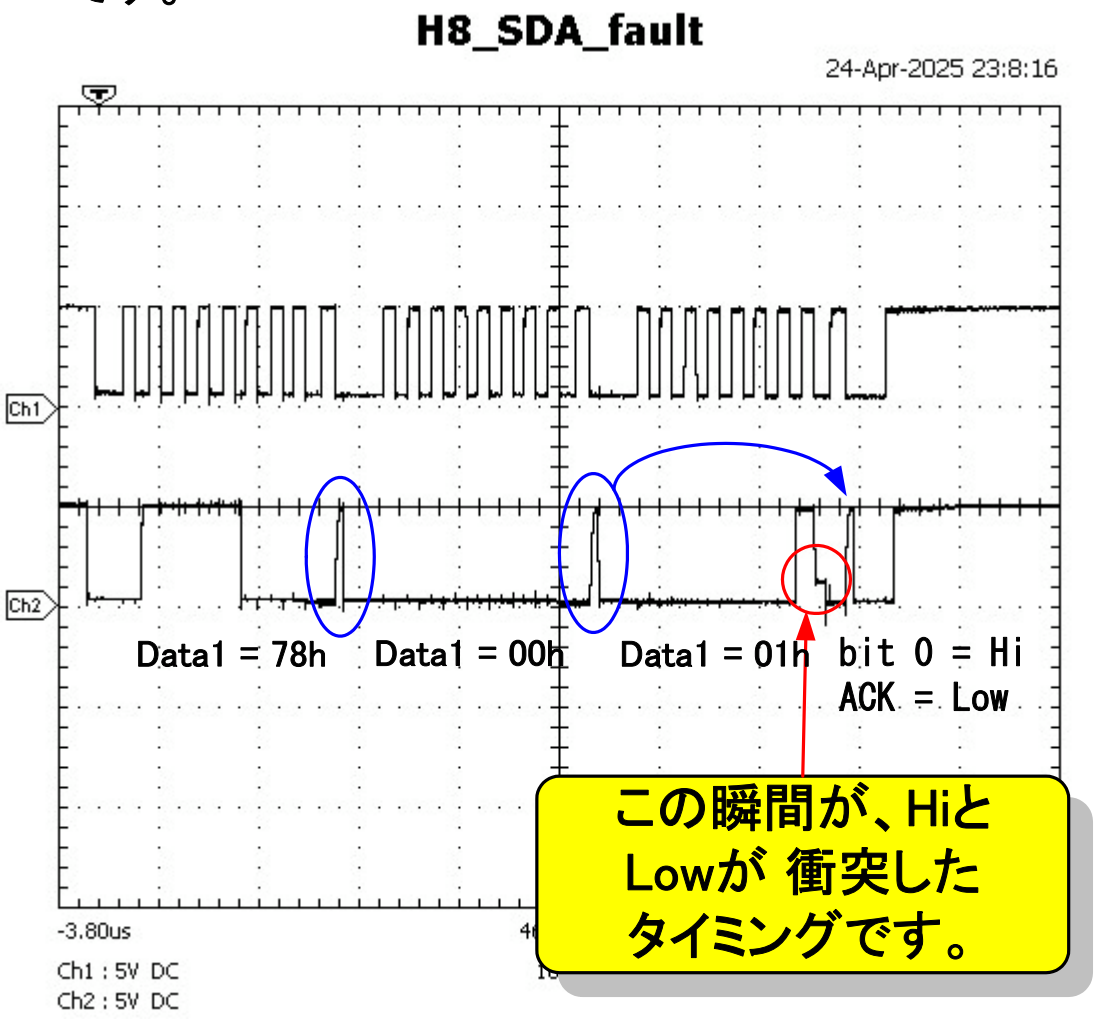
左の場合、左側のトライステートの 上側のトランジスタが、ONして、右側のオープンコレクタのトランジスタが ONした場合に 限って 大きな電流が流れ、トラブルが 発生します。

通常 I2Cデバイス側の出力は、オープンコレクタですが、マイコン側の I/Oポートは トライステート出力です。

よって、SDAの入出力方向の切り替え時、マイコン側が Hiレベルを出力していて、デバイス側が Lowレベルを出力しようとしていると、一瞬信号が 衝突します。 I2Cの場合 8bitのデータを出力した後、デバイス側が ACK（Lowレベル）を 返します。で、直前のデータの bit0が 1 の場合 Hiレベルになります。 この bit0 出力から出力方向を 切り替えてデバイスが ACKを出す時に 一瞬ショートする危険があります。 bit0 が 0 の場合は 一瞬のショートは 発生しません。



実際、ショートした部分を記録したオシロ波形です。

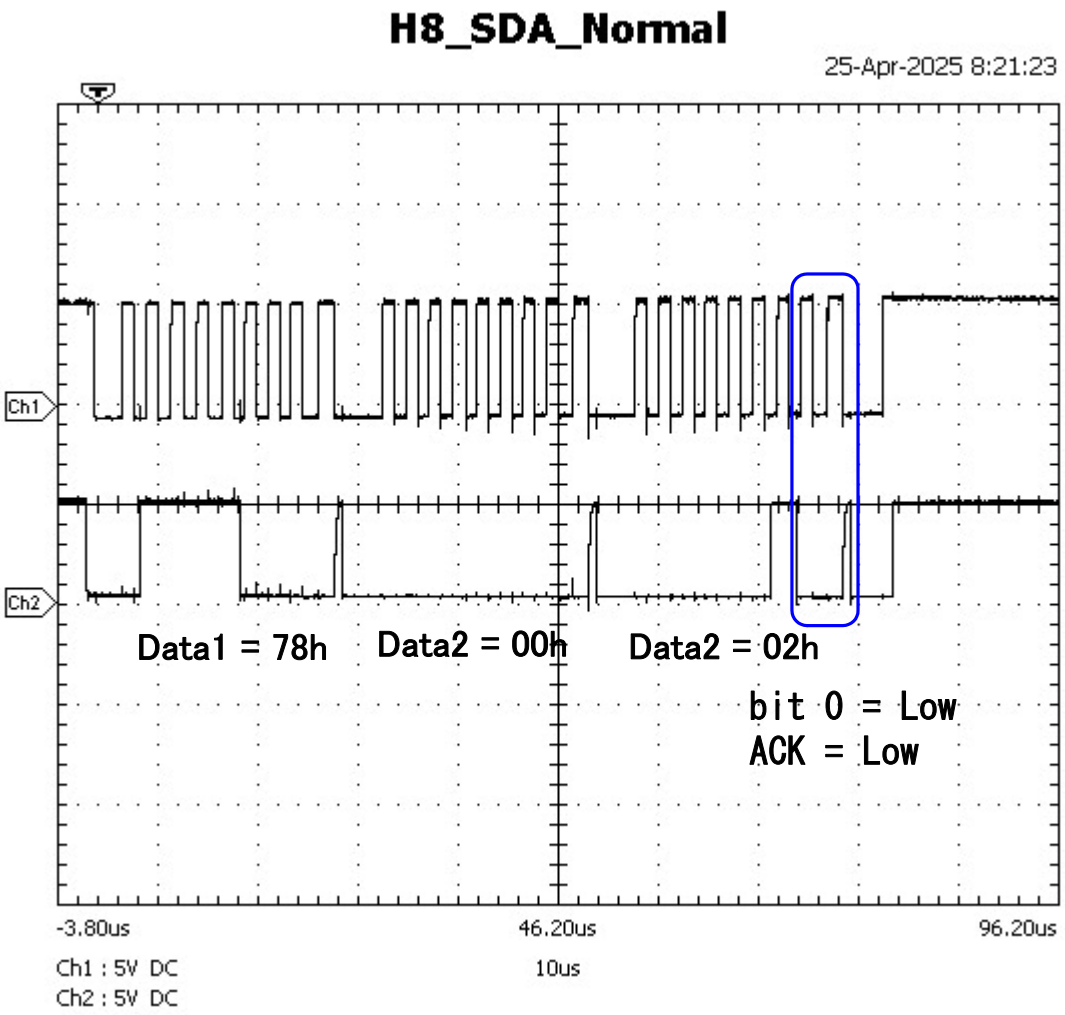


中途半端なレベルの段差が ありますが、これが Hiと Lowの信号が衝突した瞬間です。
トライステート出力は どちらかというと Lowの方が やや強いです。 その関係で Low寄りの段差になっている物と思われます。

一瞬ショートした程度では 出力トランジスタは、壊れる事は あまり無いと思われますが、今回の場合、ショックで マイコンのプログラムが 暴走したように なりました。 最初、原因が分からず悩みました。

青の縦長楕円、及び青の円弧の線の矢印で示している部分は ACKの 出終わった後、デバイスも マイコン側も SDAバスを 一瞬 オープン状態にしているので、プルアップ抵抗で ゆっくり Hi に 引き上げられています。
これは、全く問題ありません。

この　SDA波形は b0=0 で 最下位ビットが Lowで 問題ありません。



で　H8マイコンの I/Oポート出力が　トライステート出力 というか　殆どのマイコンが トライステート出力です。

で、どのように対応したかというと、SDAを　Low に　する場合は、通常通り　SDAポートに 0 を 出力します。 で、SDAポートを　Hiにする場合は、SDAポートに 1 を　出力しません。 代わりに データディレクションレジスタを操作して　SDA ポートを　入力に 切り替えます。 こうする事により、SDAポートは　ハイインピーダンス状態となり プルアップ抵抗で　弱く Hi に引き上げられます。 等価的に　オープンコレクタと　同様の状態に なったといえます。 これにより、マイコンの不可解な暴走現象は　収まりました。
マイコンによっては、I2Cの 内蔵周辺回路を 持っている物もあります。 それらの I2C出力は オープンコレクタ出力に してあると思います。

過去に、R8Cの百円マイコンで ソフトによるI2Cを 実現しましたが、その時は トラブル無くすんなり出来ました。 実は、R8Cの百円マイコンは、I/Oポートの設定に オープンコレクタ出力にする設定が あったのです。
I/Oポートを オープンコレクタ出力にする設定は、珍しいですね。 そのお陰で R8Cの時は楽に出来ました。

で、H8マイコンの I2Cですが 動き始めたのですが、時々データが 化ける時が あるのでちょっとデータ転送時の タイミング調整を 行いました。 若干遅くなりましたが、それでも388Kbpsなので 良しとします。

今のところ、RTC RX8900と OLED表示器のSO1602 を 接続して RTCから読み出した時刻を OLEDに表示して 問題なく動いています。

OLED SO1602Aの アクセスのやり方

RTCよりは 簡単です。 3つの関数を 用意しました。
① 初期化処理 so1602_init 関数と
② 1行目の文字列表示 so1602_prin1 関数と
③ 2行目の文字列表示 so1602_prin2 関数の3本です。
あと、用途が限定的ですが、RTC RX8900からの データを 渡して 時刻表示を行う処理 so1602_disp_time を 入れてます。
ソースは、SO1602A.c と SO1602A.h の 2本です。

ソース表示は横長になるので、次のページに示します。

SO1602A.h

```
#define    SA_SO1602  0x3C      // SO1602A デバイスアドレス

//  関数プロトタイプ宣言
// ------------------------------------------------------
void  so1602_disp_time( _UBYTE dt[] );   // 時刻表示
void  so1602_init( void );       // SO1602 初期化
void  so1602_prin1( char *p );  // SO1602 1行目 文字列出力
void  so1602_prin2( char *p );  // SO1602 2行目 文字列出力
```

SO1602A.c (1/5)

```
//**  SO1602Aへ コマンド、データ出力を 行う サブ関数
//**  SA_SO1602 は  OLED表示器の I2Cスレーブアドレス ( 3Ch )
//**  c1 :  コマンドバイト、 d1 :  データバイト、 tm :  待ち時間 ( ミリ秒単位 )
// ------------------------------------------------------------------------
static  void  so1602_put( _UBYTE c1, _UBYTE d2, _UBYTE tm )
{
     i2c_put_one( SA_SO1602, c1, d2 );    // コマンド 1byte、データ 1byte 出力
     wait_ms( tm );        // ミリ秒単位の 時間待ち
}
```

左は、SO1602処理のヘッダーファイルです。 SO1602の デバイスアドレスは 3Ch です。

左下の 関数は 内部関数の so1602_put です。 SO1602Aに コマンド byteと データ byte を転送します。 このタイプの表示器は コマンドを 一つ送ると、その後に 1～20msの 時間待ちを行う必要があります。よって 時間待ちの処理も 内部関数に入れ込みました。

最後の引数 tm が 時間待ちの値です。

SO1602A.c (2/5)

```
//**  SO1602 初期化  **
void  so1602_init( void )
{
      so1602_put( 0x00, 0x01, 20 );      //  全領域  表示消去
      so1602_put( 0x00, 0x02, 2 );       //  左上にカーソル移動
      so1602_put( 0x00, 0x0F, 2 );       //  表示 開始
      so1602_put( 0x00, 0x01, 20 );      //  全領域  表示消去
}
```

SO1602A.c (3/5)

```
//**  1行目  文字列出力  **
void  so1602_prin1( char *p )
{
      char  c;

      so1602_put( 0x00, 0x02, 10 );      //  1行目 左端に
      c = *p;     p++;                   //  カーソル移動
      while( c != NULL )  {
            so1602_put( 0x40, c, 1 );    //  DDRAMに 1文字
            c = *p;     p++;             //  文字コード書込み
      }
}
```

左は SO1602Aの初期化処理です。
それぞれの行に コメントを付けているので、それぞれのコマンドの機能は分かると思います。

SO1602Aに送る コマンドコードは 00h と 40h が あります。
00h は 制御コマンドです。
40h は 文字コード 表示コマンドです。 ASCIIコードの 41h を データとして送ると "A" が 表示されます。

尚、左のソースを見て 分かると思いますが、1回の送信電文では、1byteのコマンドコードと 1byteの 文字コードを送る仕様に なっているので文字数分 文字コードを送信するコマンド電文を 送る事になります。

SO1602A.c (4/5)

```
//**  2行目  文字列出力  **
void  so1602_prin2( char *p )
{
      char  c;

      so1602_put( 0x00, 0xA0, 10 );      //  2行目左端に
      c = *p;     p++;                   //  カーソル移動
      while( c != NULL )  {
            so1602_put( 0x40, c, 1 );    //  DDRAMに 1文字
            c = *p;     p++;             //  文字コード書込み
      }
}
```

2行目に 表示させる文字列出力関数はカーソル移動の コマンド出力関数が異なるだけで、後は 1行目出力関数と 同じです。

あと、時刻表示関数ですが、やや長いので 次のページに 表示します。

SO1602A.c (5/5)

```
//**  2行目  時刻表示出力
//**  char  dt[] : RTC 時刻 data 配列
// ----------------------------------------
void  so1602_disp_time( _UBYTE *dt )
{
      char  buf[18], tx[4];

      buf[0] = NULL;          // buf内に 時刻表示文字列生成
      byte_bcd2( tx, dt[5] );      // 月の項目 -> BCD 2桁変換
      str_cat( buf, tx );          str_cat( buf, "-" );
      byte_bcd2( tx, dt[4] );      // 日の項目 -> BCD 2桁変換
      str_cat( buf, tx );          str_cat( buf, "/" );
      byte_bcd2( tx, dt[2] );      // 時の項目 -> BCD 2桁変換
      str_cat( buf, tx );          str_cat( buf, ":" );
      byte_bcd2( tx, dt[1] );      // 時の項目 -> BCD 2桁変換
      str_cat( buf, tx );          str_cat( buf, ":" );
      byte_bcd2( tx, dt[0] );      // 時の項目 -> BCD 2桁変換
      str_cat( buf, tx );
      so1602_prin2( buf );   // SO1602 2行目に 時刻文字列 表示
}
```

SO1602Aの 2行目に RTC RX8900 の 時刻情報を 表示させる文字列出力関数です。

コメントを見ると分かると思いますが月のBCD 2文字＋日のBCD 2文字＋時のBCD 2文字＋分のBCD 2文字＋秒のBCD 2文字を 順次連結してます。
そして、組み立てた文字列を 表示器に 転送しています。

次は、RTCに設定した時刻を RTCの秒更新 IRQ信号を 読み込み 1秒毎に 時刻を読み出して、テラタームの画面と SO1602A表示器に 時刻を表示するデモを お見せします。