まずは自由自在に画面表示を行おう

液晶搭載マイコン・モジュール活用テクニック

すでに店頭に並んでいる書籍『すぐに使える！液晶搭載マイコン・モジュール』，お手にとっていただけたでしょうか．本稿では，液晶搭載マイコン・モジュールを使いこなすためのさまざまなテクニックを紹介します．
（編集部）

液晶搭載マイコン・モジュール，使いこなしていただけているでしょうか．店頭で『すぐに使える！液晶搭載マイコン・モジュール』（以下，本書）を購入し，パソコンにUSBケーブルを接続し，LEDが点滅することを確認できたでしょうか．はんだ付け不要で，「とりあえず」動作するというコンセプトで，この液晶搭載マイコン・モジュールは作られています．そしてWebサイトで配布されている開発ツールとサンプル・プログラムをダウンロードし，LCD表示やスイッチ入力，最後のブロック崩しのサンプルまで自力でコンパイル＆ダウンロードできたでしょうか．ここまではCQ出版社および筆者の動作確認がとれているものなので，誰でも動作できるようになっています．

そこからさらに一歩進んで，オリジナルのプログラムを書いているでしょうか．せっかくの液晶搭載マイコンですので，自分の好きな絵を描いてみたいと思いませんか？今回はその方法について解説を行います．本書 第5章に掲載されている液晶表示のサンプル・プログラムを改造して自分で作成した任意のデータを表示できるようにします．

1. 搭載しているマイコンの特徴

本書にはセイコーエプソン社のS1C17702マイコンと，解像度72ドット×32ドットの液晶画面（LCD）が搭載された基板が付属しています．同マイコンはセイコーエプソン社オリジナルのCPUコアであるC17マイコン・アーキテクチャを採用しています．また同マイコンはCPUコアだけでなくLCDコントローラを内蔵しているのが特徴です．そのため，CPUから出力されている信号線を液晶に直結するだけで画面表示ができるという特徴があります．

今回はこの液晶画面の表示を行ってみましょう．

2. 下準備：プロジェクトのコピー

● 既存のプロジェクトをコピーして使う

液晶表示プログラムを0から作るのは大変なので，すでに動いているプログラムを元に作るのが得策です．プログラミングでは，このような「差分開発」によって工期を短縮することができます．

幸い，今回はLCDの表示を行うLCD_Testがサンプル・プログラムとして公開されているので，これを元に自分の好きな画面を表示してみましょう．

● LCD_Testの動作確認

まずはWebページで公開されている，第5章のサンプル・プログラムLCD_Testを`C:¥EPSON¥C17WBIF¥eclipse¥workspace`に展開し，ビルドします．zipファイルを展開し，File→Importでインポートします．その後，Project→Build Projectでビルドした後，Run→Externel Tools→Externel Tools Configurationでマイコン基板へ書き込みます．このあたりの手順は本書で解説されている通りです．

● Eclipseでのプロジェクトのコピー方法

ここからは自分用のプロジェクトLCD_Test2を作ります．元のLCD_Testは残しておきたいので，プロジェクトをコピーして名前を変更します．

そこで，ディレクトリLCD_TestをLCD_Test2という名前でいったんデスクトップに丸ごとコピーしてFile→Importします．しかし，「Some projects were hidden because they exist in the workspace directory」という警告が発生し，インポートできません．どうやらディレクトリ名を変えただけではLCD_Testと同じプロジェクトと認識されてインポートできないようです．

そこでテキスト・エディタでファイルを眺めると，ファイル.projectでプロジェクト名を指定している「<projectDescription> <name>LCD_Test</name>」という記述があります．これをLCD_Test2に書き換えたところ，無事にインポートできました．この状態でビルドは行えます．

次にRun→External Tools→External Tools Configurationとしたところ，External Tools Configurationsウィンドウには C17 Debbuger Launch for LCD_Testしか出てきません．今回コピーしたLCD_Test2がありません．同じようにディレクトリを眺めて，拡張子.launchのファイルC17 Debbuger Launch for LCD_Test.launchをC17 Debbuger Launch for LCD_Test2.launchにリネームするとConfigurationsウィンドウに出てきました．

さらにこのファイルの中にワークスペースの場所を指定するworkspace_loc:/LCD_Testという部分があったので，これもLCD_Test2に変更します．この作業をしないとデバッグ時にコピー前のファイルが参照されてハマります．

以上の変更で，無事新しいプロジェクトLCD_Test2のビルド&実行ができました．

ここまでの作業をまとめると，Eclipseプロジェクトのコピーは，以下の手順で行います．

1）.projectの編集

2）＜プロジェクト名＞.launchを＜新プロジェクト名＞.launchにリネーム

3）＜新プロジェクト名＞.launchのworkspace_loc:/を編集

3. 文字を書いてみる

● main.cを改変する

LCD_Testに含まれるmain.cがメイン・ルーチンです．initLcdPower()やinitLcd()，lcdOn()などの関数を呼んで初期化を行っています．初期化が終わった後にdrawLine()などで描画を行っているようです．ソース・ウィンドウに表示されているmain.cの行番号の左で右クリックすると，メニューの中に「Toggle Break Point」という選択肢があります．これを選択すると青色の丸が付き，ここでプログラムを止められます（**図1**）．実際にプログラムがどこまで実行されたか確認するには，このようにブレークポイントを設定して試していくのが一番です．

ソースを見ると，forループで時間待ちをしているようです．reverseLcdBW();の後のforループあたりにプログラムを追加してみましょう．

● VRAMの構造

S1C17702の液晶画面は表示用ビデオRAM（以下VRAM）を持っており，ここに値を書き込めば描画が行えます．VRAMの構造は参考文献(1)の「22.5 表示メモリ」の「図22.5.1 表示メモリマップ」に記載されています．これを簡略化したものを**図2**に示します．なお，今回の基板では1/32デューティを選択しています．

縦8ドット×4行＝32ドットで，横は72バイトです．これによりカタログ・スペックどおり72ドット×32バイトになります．

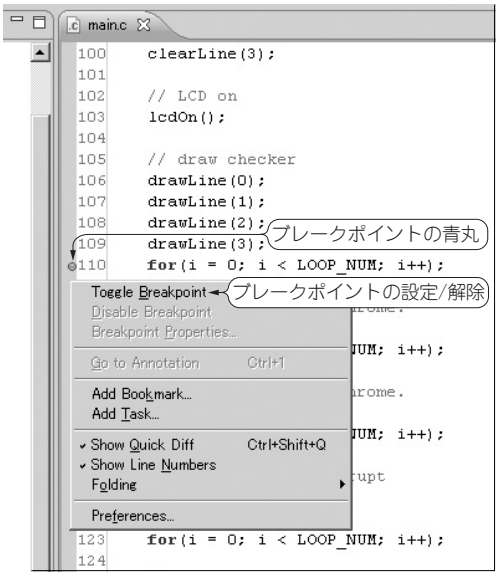


図1　ブレーク・ポイントの設定

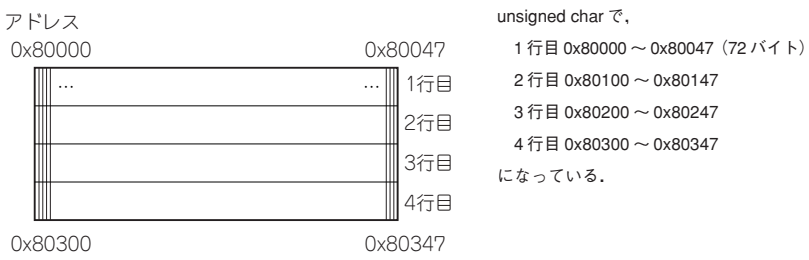


図2　VRAMの構造

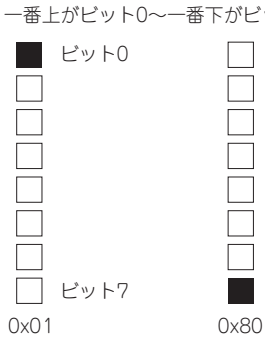
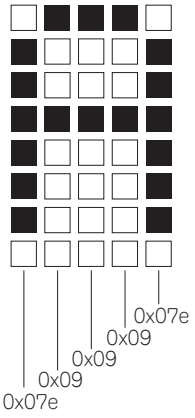
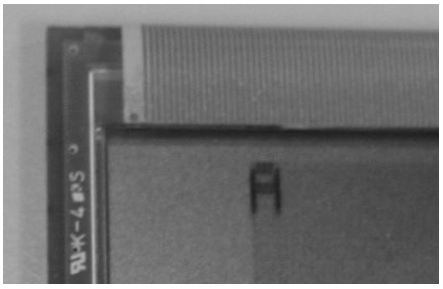


図3　書き込むデータの縦方向



◀ 図4
文字「A」のデータ



▶ 写真1
文字「A」を表示

縦方向は**図3**のようになっています．一番上のドットだけ黒くしたい場合には16進数で0x01を，一番下のドットだけ黒くしたい場合には0x80を書き込みます．すべて黒くしたい場合は0xffです．たとえば，文字「A」の形で左上に表示したい場合は**図4**のようなデータになります．これをアドレス0x80000から順に書き込むと**写真1**のように表示されます．このプログラムを**リスト2**に示します．**リスト1**で示した「プログラムを追加する個所」に**リスト2**を挿入すると，**写真1**のような表示が行われます．

● 応用例と注意

アドレスと書き込むデータがわかれば，後は好きな絵を描くだけです．

一つだけ注意する点は，右端のアドレスと次の行の左端がつながっていないことです．たとえば，1行目の右端は0x80047ですが，次の行の左端は0x80100であることに注意してください．

0から71までの値を書き込むプログラムを**リスト3**に示します．この実行結果は**写真2**のようになります．

4. 点を打つプログラム

● アドレスの計算

画面へのパターン表示ができたので，次は任意の座標に点を打つ関数を作成しましょう．

`int pset(x,y);` でx=0～71，y=0～31の範囲で点を打ちます．点を打つといっても，結局はアドレスを計算してそこにデータを書き込むだけです．

アドレスの計算式ですが，座標(0, 0)のアドレスが0x80000で，xが1増えるごとにアドレスも1増えます．面

リスト1　LCD_Testに含まれるmain.cリスト1

```
/*
 * Main function.
 *
 * return: int Normality 0, Abnormality 1
 * history: 2007/08/17  start
 */
int main(void) {
    long i; // loop counter
    long sTimmer;

    psTimmer = &onesec;

    debugModeMisc(MISC_01DBG_STOP); //misc stop in debug mode
    controlPsc(PSC_PRUN_RUN);       //prescaler run
    debugModePsc(PSC_PRUND_STOP);   //prescaler stop in debug mode
    setClockGear(CLG_CCLKGR_1);     //set clock gear *defalt=0x0
    controlClg(CLG_PCLKEN_ENA);     //run clg
    controlItc(ITC_CTL_ENA);        //itc enable

    // initializes LCD
    // power booster off, regulator on, heaby protection off.
    initLcdPower(LCD_PWR_BOOSTER_OFF, LCD_HEAVY_PROTECT_OFF);

    // clock is OSC3
    startLcdClk(LCD_CLK_SELECT_OSC3, LCD_CLK_DIV_32);

    // duty is 1/32, contrast is LV.14.
    initLcd(LCD_DUTY_32, LCD_CONTRAST_LV_14);
    reverseLcdCom();
    reverseLcdSeg();

    // initializes LCD interrupt
    initLcdInt(0x1);            // interrupt level is 1

    // clear LCD
    clearLine(0);
    clearLine(1);
    clearLine(2);
    clearLine(3);

    // LCD on
    lcdOn();

    // draw checker
    drawLine(0);
    drawLine(1);
    drawLine(2);
    drawLine(3);
    for(i = 0; i < LOOP_NUM; i++);

    // reverse LCD monochrome.
    reverseLcdBW();
    for(i = 0; i < LOOP_NUM; i++);

    // reverse LCD monochrome.
    reverseLcdBW();
    for(i = 0; i < LOOP_NUM; i++);
                                    ←このあたりに新しい描画を追加する
～略～
```

リスト2　文字「A」を書き込む

```
    clearLine(0); //画面消去
    clearLine(1);
    clearLine(2);
    clearLine(3);

    //文字Aのビット・パターン
    *(unsigned char *)0x80000 = 0x7e;
    *(unsigned char *)0x80001 = 0x09;
    *(unsigned char *)0x80002 = 0x09;
    *(unsigned char *)0x80003 = 0x09;
    *(unsigned char *)0x80004 = 0x7e;

    for(i = 0; i < LOOP_NUM; i++);
```

▶ リスト3
0から71までの値を書き込むプログラム

```
    unsigned char *p;

～略～

    clearLine(0);
    clearLine(1);
    clearLine(2);
    clearLine(3);

    p = (unsigned char *)0x80000;
    for (i = 0; i < 72; i++) {
        *p++ = i;
    }
    p = (unsigned char *)0x80100;
    for (i = 0; i < 72; i++) {
        *p++ = 256-i;
    }
    p = (unsigned char *)0x80200;
    for (i = 0; i < 72; i++) {
        *p++ = i;
    }
    p = (unsigned char *)0x80300;
    for (i = 0; i < 72; i++) {
        *p++ = 256-i;
    }

    for(i = 0; i < LOOP_NUM; i++);
```

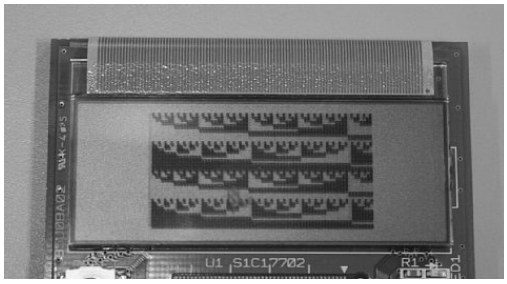


写真2　リスト3の実行結果

▶ リスト4
pset()関数

```
int pset(int x, int y){
   const unsigned char *vram=0x80000;
   unsigned char *p;
   unsigned char data=1;

   //範囲チェック
   if ((x>72)||(y>32)){
      return (-1); //異常終了
   }

   p = vram+(y/8)*0x100+x; //アドレスの計算
   data <<=(y % 8);        // 書き込むデータ
   *p = data;

   return (0); //正常終了
}
```

倒なのがyの計算で，yが8増えるごとに0x100増えます．そこで計算式は，ポインタ変数を(unsigned char *)pとして，

```
p = vram+(y/8)*0x100+x;
```

になります．ここに書き込むデータは，y座標＝0，8，16，24のときは0x01，y座標＝1，9，17，25のときは0x02となります(**図5**)．

ここではシフト演算子を使って，

```
unsigned char data=1;
data <<=(y % 8);      // 書き込むデータ
```

となります．ということで関数は**リスト4**のようになります．

入力値xとyの範囲チェックもちゃんと行ってます．これは意外と重要です．**リスト5**のようにこれを呼び出すと**写真3**のように表示されます．

これで完成…と思ったのですが，実はバグがあります．

```
pset(0,0);
pset(0,2);
```

のように，同じアドレスへデータを書き込むと，前のドットが消えてしまいます(**写真4**)．最初に書き込んだデータ0x01が0x04で上書きされ，消えてしまうためです．これを回避するためには，データの論理和(OR)をとって書き込みます．この場合，最終的に書き込みたいデータは0x05です．そこで，前に書き込んだデータとのORをとるように変更します．**リスト6**が変更後です．

リスト5
pset()関数の呼び出し

```
for (i=0; i<32; i++){
   pset(i,i);
}
```

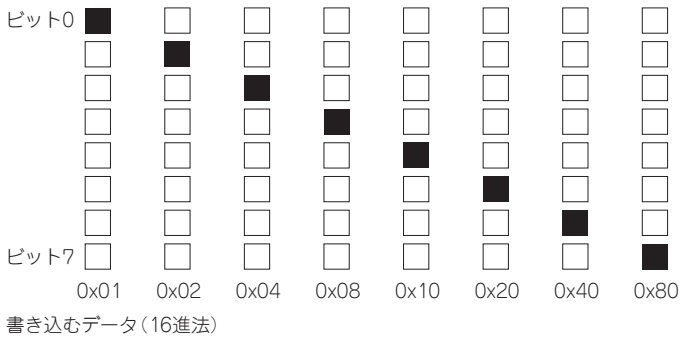


図5　ドットごとに書き込むデータ

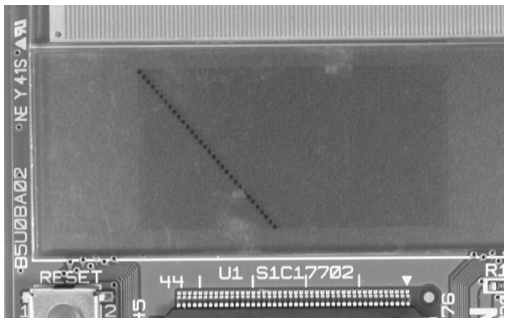


写真3　pset()関数の表示結果

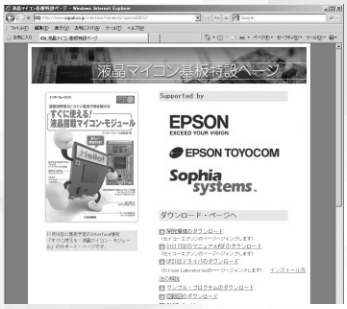
「Interface」液晶搭載マイコン基板アプリケーション制作コンテストのお知らせ

Interface 編集部では，書籍『すぐに使える！液晶搭マイコン・モジュール』に付属したマイコン基板を利用する「アプリケーション制作コンテスト」を開催する予定です．

本基板には，液晶はもちろんのこと，スイッチやGPIO，タイマなどが搭載されています．これらを活用した制御アプリケーションや表示装置，ゲームなどさまざまな応用ができるでしょう．

この機会に実用的な作品，楽しい作品，ユニークな作品，教育的な作品など，自慢の製作事例を持ち込んで，本コンテストに参加してみませんか？

入賞者には，すてきな商品を差し上げる予定です．また，入賞作品については，その製作レポートを本誌に掲載します．アプリケーション制作コンテストの詳細は，次号（2010年4月号）の誌面および本誌Webページ（**図A**）で発表します．



図A　『すぐに使える！液晶搭載マイコン・モジュール』サポート・ページ
http://www.cqpub.co.jp/interface/contents/special0912/

リスト6　pset()関数（OR版）

```
int pset(int x, int y){
   const unsigned char *vram=0x80000;
   unsigned char *p;
   unsigned char data=1;

   //範囲チェック
   if ((x>72)||(y>32)){
      return (-1); //異常終了
   }

   p = vram+(y/8)*0x100+x; //アドレスの計算
   data <<=(y % 8); // 書き込むデータ
   *p |= data;                    ← 変更した

   return (0); //正常終了
}
```

これで**写真5**のように正常に表示されるようになりました．このようにVRAMへ直接書き込むプログラムでは，前に描いたものを消さないために「重ね合わせ」の問題が発生します．今回はモノクロ液晶なので単純にORで重ね合わせができましたが，これがフル・カラーになると，1ドットずつ確認していかないといけないので大変です．そのため，重ね合わせ機能をハードウェアで実装したグラフィックス・チップが世の中にたくさんあります．

＊　＊　＊

ここまで理解できれば，もう好きな画像を描けるはずです．Let's Try！

最近のパソコンはVRAMへ直接アクセスすることができず，どのような機構で画面に絵が描かれているのかを理解するのは難しいと思います．しかし今回の液晶搭載マイコン・モジュールのような素朴なシステムであれば，メモリへのアクセスが画面への描画と直結していることが体で理解できると思います．このような環境は貴重なので，ぜひ，自分の手を動かして試してみてください．

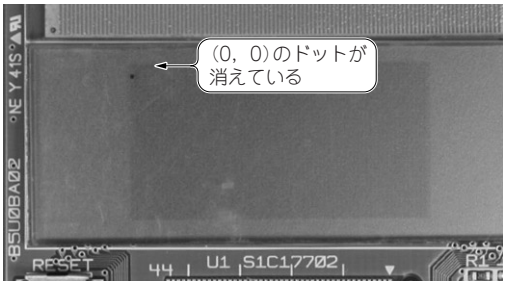


写真4　pset()関数のテスト（ORなし版）

写真5　pset()関数のテスト（ORあり版）

参考文献

(1) S1C17702テクニカルマニュアル，セイコーエプソン．