グラフィックライブラリ　ＲＸ２３０＿ＬＣＤ　の使い方　Ｖｅｒ１．０　２０２５０６１３

■概要

描画ライブラリはルネサスエレクトロニクス社のマイコンＲＸ２３０と表示器２．８ｉｎｃｈ　２４０×３２０ドットフルカラーＴＦＴ液晶※1 で動作するグラフィック、タッチスイッチライブラリです。ＳＰＩ接続　クロック６．５ＭＨｚ※２　開発環境ＣＳ＋　ｆｏｒ　ＣＣ　です。

※1　ＭＳＰ２８０７（秋月電子殿）、ＫＫＨＭＦ（アマゾン）等のコントロールＩＣ　ＩＬＩ９３４１を使った液晶を想定。

※２　ＩＬＩ９３４１の最高動作クロックは１０ＭＨｚです。

目次ーーーーーーーーーーーーーー

－－

●セットでご購入時の梱包物

ハード、ソフトのセットでご購入いただいた場合の箱の梱包物です。

グラフィックライブラリの使い方（このマニュアルです）

RX230__240×320LCD ボードハードマニュアル

ＣＤ（グラフィックライブラリ、サンプルソフト、
ドキュメント、フォントを作るソフト等）

液晶板+RX230CPU ボード

タッチキー用ペン

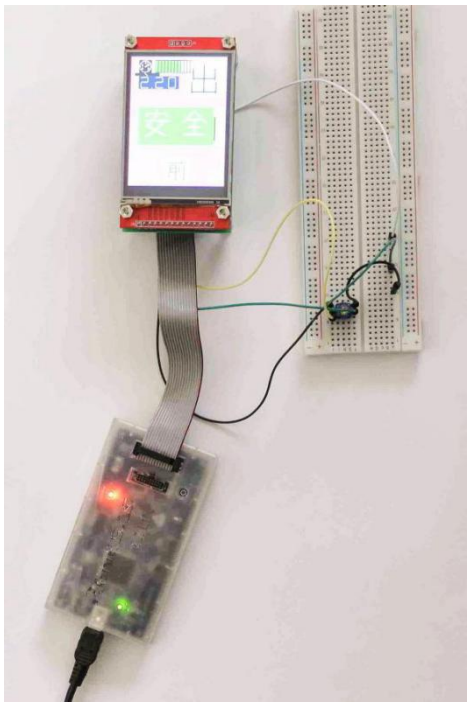
デバック用２５ｍｍスペーサー

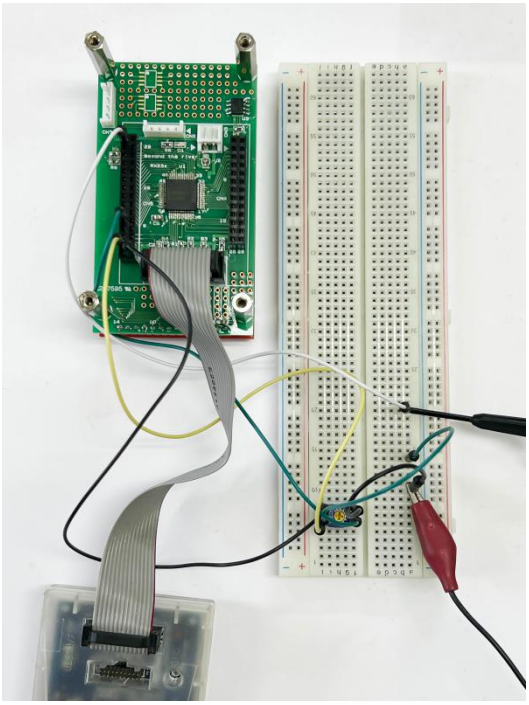
E2lite やコネクタからブレットボード
への線がストレスなく接続できます。

本製品開発には E1,E2lite 等のエミュレータが必要です。
サンプルソフトは E2lite 用に書かれていますが、容易に E1 でも動作するように変更可能です。お問い合わせください。

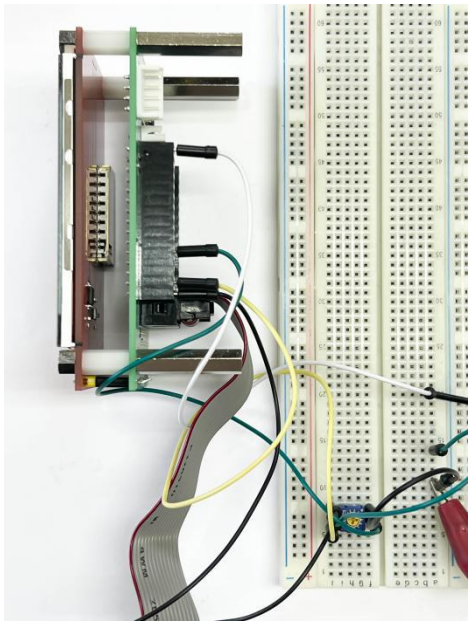
●ハードウエア　マイコン、液晶板製品外観

表面は 2.8inch 液晶が見えます。　　裏に専用の RX230 マイコン基板が搭載されています。





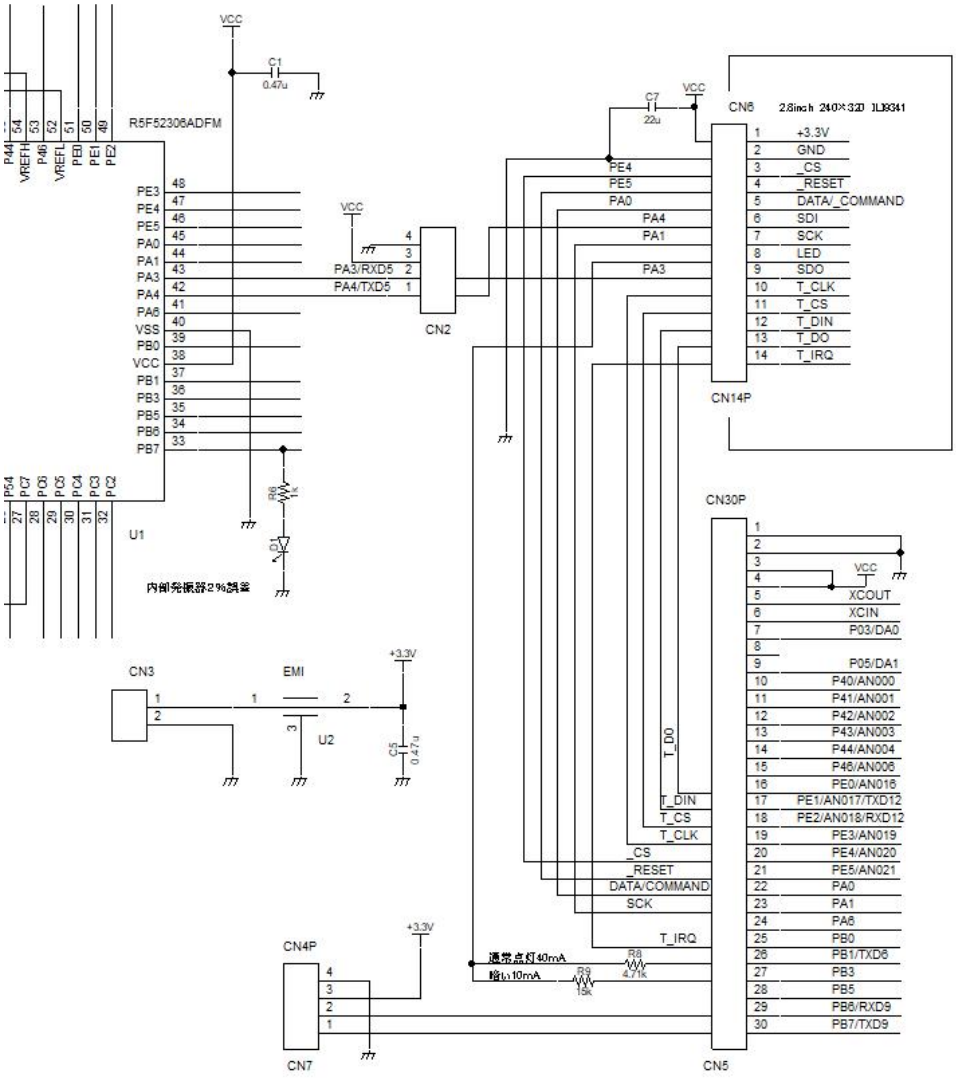
液晶板とマイコン基板は１１ｍｍの間隔でスペーサーで止められています。



CPU ボードのコネクタはメスコネクタが実装済みで、ブレッドボードに簡単に接続できます。デバック時は E2lite からの 3.3V で液晶を駆動出来ます。デバック終了時は電源コネクタに添付ケーブルで 3.3V を印加し使用してください。

●回路図概要

マイコン、液晶は+３．３Ｖ動作です。マイコンは５Ｖでも動作可能ですが、液晶が不可なので、３．３Ｖで使用します。マイコンと液晶の接続はＳＰＩ接続です。ＣＮ２　２番ＲＸＤ５がＳＤＯに接続され、１番ＴＸＤ５がＳＤＩに接続されています。ＳＣＫは６．５ＭＨｚのクロックです（最大１０ＭＨｚ）。液晶側ＣＮ６のＴ＿ｘｘがつくポートはタッチスイッチのインターフェイス信号です。１４番　Ｔ＿ＩＲＱはタッチスイッチが押されるとＬｏｗ＝０Ｖに下がります。Ｔ＿ＤＯからのデータで押された位置のＸ，Ｙポジションを認識します。



●ソフトウエア　ＣＤの詳細

ＣＤには以下のホルダがあります。

Ｆｏｎｔｓはライブラリで使用出来る文字や図形データ集で、詳しい使い方は後述しますが、使いたいフォントをエディタで br_fonts.dat ファイルに移して使います。

サンプルソフトはＣＳ＋　ｆｏｒ　ＣＣ環境で動作し、動きはｙｏｕｔｕｂｅでも確認できます。これをひな形として独自のプログラムを開発することが出来ます。

ドキュメントは回路図、ハードの取扱説明書、ソフト　グラフィックライブラリの使い方があります。

フォントを作るは 16×16 のフォントを作る lfms16.exe と 32×32 のフォントを作る lfms32.exe の 2 つと、いくつかのフォントファイルが入っています。

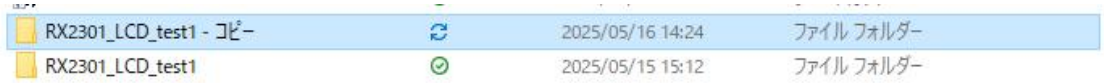
ライブラリはグラフィックライブラリの本体で、新規にプログラムを作る場合、ここからコピーしてください。

●ＣＳ＋　ｆｏｒ　ＣＣの設定

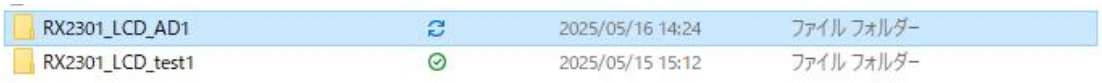
液晶を駆動するにあたり、ポートの設定やタイマー割り込みを設定する必要があります。

新規にプログラムを作る場合の開発環境ＣＳ＋　ｆｏｒ　ＣＣのプロジェクト設定の仕方は、簡易版と初めから設定する方法の２つがあります。初めからの設定は項目が多数あり、ハードのことなので、特に液晶プログラムを開発、動作させるにあたり、必要な知識とも言い難く、後の方に書いておきますので、必要な時に参考にしてみてください。ここではコピペで行う簡易版の方法を示します。

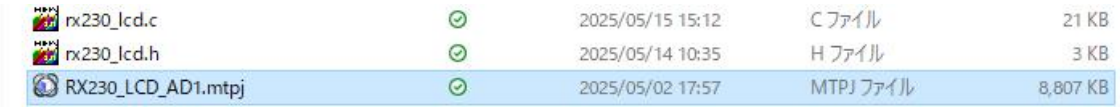
１．既に動作しているホルダをコピーします。以下例ではＲＸ２３０__ＬＣＤ__ｔｅｓｔというホルダをコピーしています。



２．ホルダの名前を付けます。例ではＲＸ２３０__ＬＣＤ__ＡＤ１に変更。



３．ホルダの中の拡張子ｍｔｐｊファイルの名前が前のままなので直します。



以上で新しいプロジェクトが完成しました。ハード的な設定は継承され、ユーザーは液晶表示を制御するプログラムを新たなプロジェクトで書くことが出来ます。

●ライブラリの使い方

ライブラリは

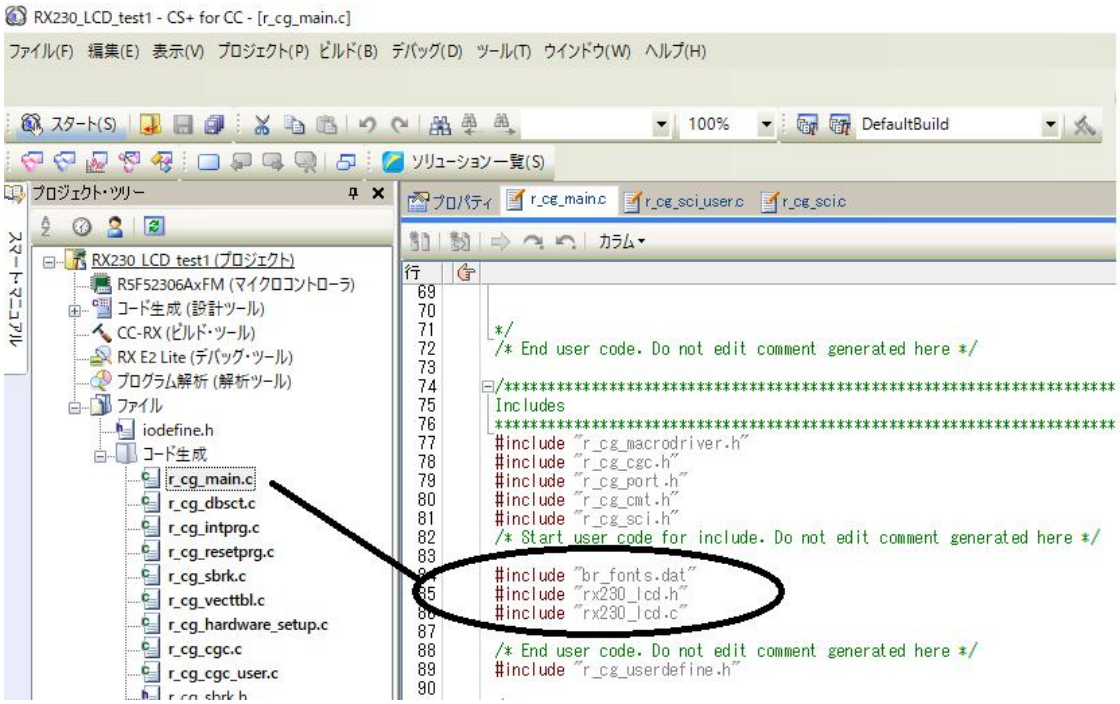
ｂｒ＿ｆｏｎｔｓ．ｄａｔ　　フォントデータファイル

ｒｘ２３０＿ｌｃｄ．ｈ　　ライブラリ本体　ヘッダファイル

ｒｘ２３０＿ｌｃｄ．ｃ　　ライブラリ本体

で構成されます。新規作成の場合、下記例のように　/*　Start、、*/　/*　End、、*/で挟むように書いて下さい。

```
/* Start user code for include. Do not edit comment generated here */

#include "br_fonts.dat"
#include "rx230_lcd.h"
#include "rx230_lcd.c"

/* End user code. Do not edit comment generated here */
```



ｍａｉｎ（）関数の/*　Ｓｔａｒｔ　ｕｓｅｒ　ｃｏｄｅ、、*/の後に

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */



//液晶初期化

                init_lcd();

//バックライト　明るい
//              LCD_DARK_ON;                //LCD バックライトON　ダーク　夜用
                LCD_BRITE_ON;               //LCD バックライトON　明るい　昼用
```

ｉｎｉｔ＿ｌｃｄ（）　液晶の初期化と
ＬＣＤ＿ＢＲＩＴＥ＿ＯＮ；　//ＬＣＤ　バックライトＯＮ　明るい　昼用
//ＬＣＤ＿ＤＡＲＫ＿ＯＮ；　//ＬＣＤ　バックライトＯＮ　ダーク　夜用

ＬＣＤバックライトの明るさを決めて関数を実行することにより液晶画面に文字、漢字、図形、グラフ等々の表示を開始することが出来ます。

●フォントデータの使い方、組込みマイコンの考え方
フォントデータはｂｒ＿ｆｏｎｔｓ．ｄａｔの中にあります。

冒頭、コメントにありますが、表示できる色は０～ｆまで、１６色、例えば黒の場合、０、白は３で表しています。

```
/*
    液晶制御プログラム　2013.2.13

    フォント　16*16　32*32

#define clBlack     0x000000          //0　黒
#define clGray      0x808080          //1　グレー
#define clSilver    0xc0c0c0          //2　銀
#define clWhite     0xffffff          //3　白
#define clFuchsia   0xff00ff          //4　ピンク
#define clPurple    0x800080          //5　紫
#define clMaroon    0x800000          //6　栗
#define clRed       0xff0000          //7　赤
#define clYellow    0xffff00          //8　黄色
#define clLime      0x00ff00          //9　ライム
#define clGreen     0x008000          //a　緑
#define clTeal      0x008080          //b　青緑
#define clBlue      0x0000ff          //c　青
#define clNavy      0x000080          //d　紺
#define clAqua      0x00ffff          //e　アクア
#define clOlive     0x808000          //f　オリーブ

2025.04.21
const ROMに配置　無し　RAMに配置

*/
```

例えば数字の０は以下のような形でunsigned　charデータになっています。背景は３なので白、文字は０で黒と表現されます。

```
//数字０
const unsigned char S0[]={
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
3,3,3,3,0,0,0,0,0,0,3,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
```

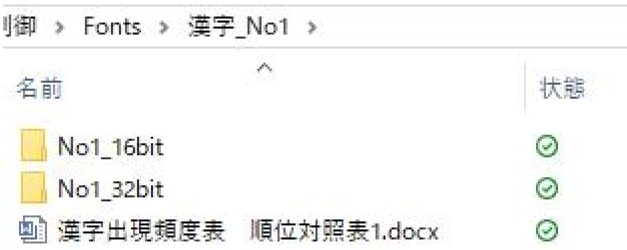
```
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,3,0,0,0,0,0,0,3,3,3,3,3,3,
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3};
```

データは１６×１６、３２×３２ドットの２種類あります。

組込みマイコンでのフォントデータの考え方は、必要なデータをｂｒ＿ｆｏｎｔｓ．ｄａｔファイルに書き加える、不要なものは削除する形になります。理由はフォントデータが例えばＪＩＳ漢字第一、二水準のデータ容量合計は２２０Ｋバイトと言われます。本件のハードウエアに使用されているマイコン　Ｒ５Ｆ２３０６ＡＤＦＭ（ＲＸ２３０）でＲＯＭ容量は２５６Ｋバイトですので、漢字データだけでＲＯＭを使ってしまう可能性があります。外部に漢字ＲＯＭを付けることも考えられますが、コスト高になります。

そこで、このライブラリではデータはある程度、弊社が用意しますので、その取捨選択はユーザーにお任せする形になります。具体例は後述のサンプルプログラムによるデモンストレーションを参照ください。

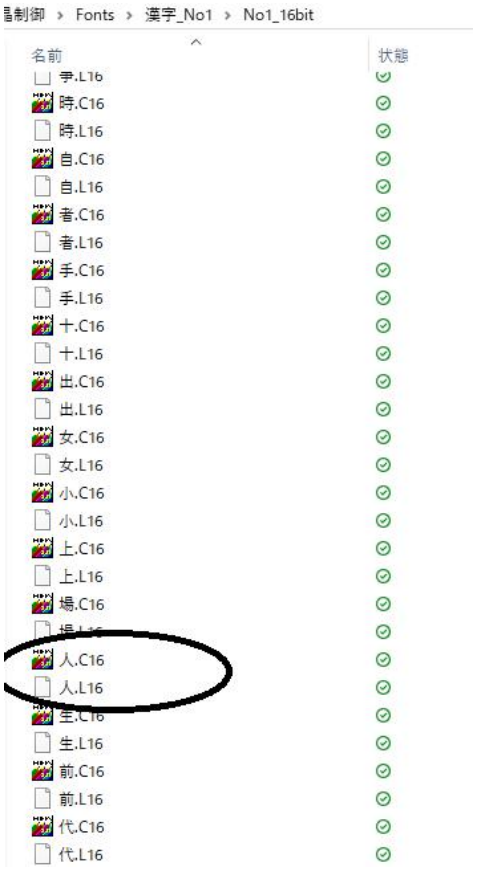
例えば漢字を追加する場合、ライブラリとは別に漢字_No1、No2 というデータがあります。



弊社の漢字フォントは漢字出現頻度表に従って作られています。出現頻度表とは統計的に得られた世の中で使われる漢字の順位です。１つのナンバーに 31 個の 16×16 漢字フォント、32×32 漢字フォントが収納されています。

漢字出現頻度表　順位対照表（Ver.1.2）1

凸　　版（３）順位	漢字	種類	備考
1	人	常用	
2	一	常用	
3	日	常用	
4	大	常用	
5	年	常用	
6	出	常用	
7	本	常用	
8	中	常用	
9	子	常用	
10	見	常用	
11	国	常用	
12	言	常用	
13	上	常用	
14	分	常用	
15	生	常用	
16	手	常用	



例えば　漢字出現頻度表１位は「人」という字ですが、No1_16bit の中に人.C16 と人.L16 と２つのファイルが収められています。人. Ｃ１６がエディタで編集して実際に使うファイルで、人.Ｌ１６は後述するフォントの作り方に出てくるフォント作成ソフトが使うデータ型式です。

人．Ｃ１６をエディタで開けると１６×１６ドットのデータが出てきます。

```
unsigned char 人[]={
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,0,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,0,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,0,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,0,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,0,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,0,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,0,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,0,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,0,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,0,3,0,3,3,3,3,3,3,3,
3,3,3,3,3,0,3,3,3,0,3,3,3,3,3,3,
3,3,3,3,0,3,3,3,3,3,0,3,3,3,3,3,
3,3,3,0,3,3,3,3,3,3,3,0,3,3,3,3,
3,0,0,3,3,3,3,3,3,3,3,3,0,3,3,3,
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3};
```

これをコピーし、ｂｒ＿ｆｏｎｔｓ．ｄａｔファイルに加えて下さい。注意点は２つあります。

１．unsigned char 人[]={　人はそのままではエラーになるので、アルファベットや数字に変えて下さい。例　unsigned char hito[]={

２．

データをＲＡＭに配置する場合はそのまま、ＲＯＭに配置する場合はｃｏｎｓｔを頭に付けて下さい。const unsigned char hito[]={

ＲＸ２３０はＲＡＭ容量は３２ＫＢとＲＯＭ　２５６ＫＢの１／８しかないので、フォントをｂｒ＿ｆｏｎｔｓ＿ｄａｔに登録していくとＲＯＭよりも早くオーバーフローすることになります。詳細は次項をご参照ください。

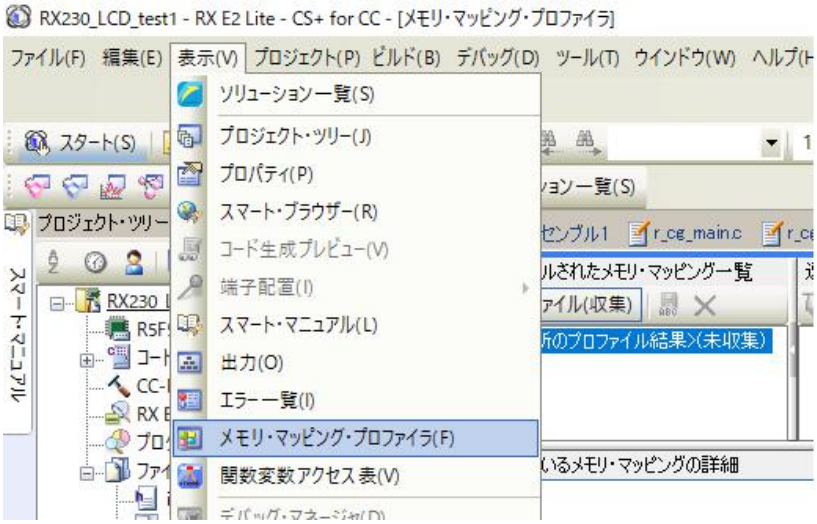
●フォントデータをＲＡＭに配置するかＲＯＭに配置するか

フォントデータは表示したい内容が増えてくると、どんどん増加し、時にＲＡＭやＲＯＭの領域を超えてしまうことがあります。組込みではこの管理が重要になります。

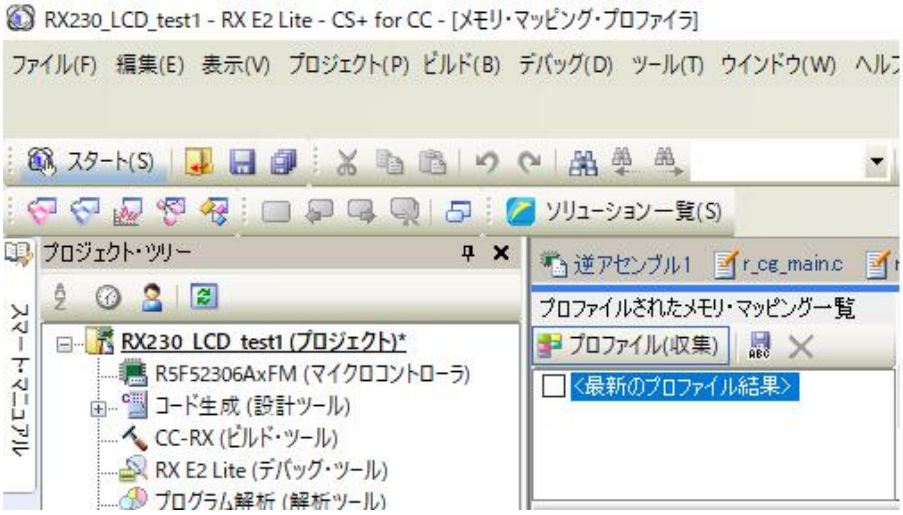
フォントデータの頭に const を書かないとデータはＲＡＭ上に展開され、書くとＲＯＭに展開されます。以下例ではＲＯＭ上に配置されます。

```
//数字０
const unsigned char S0[]={
```

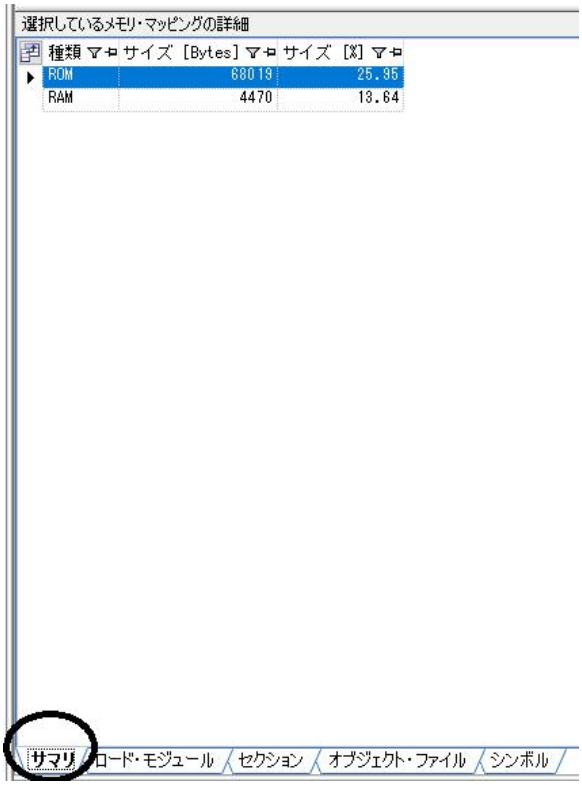
内容を調べるには表示→メモリマッピングプロファイラを開きます。



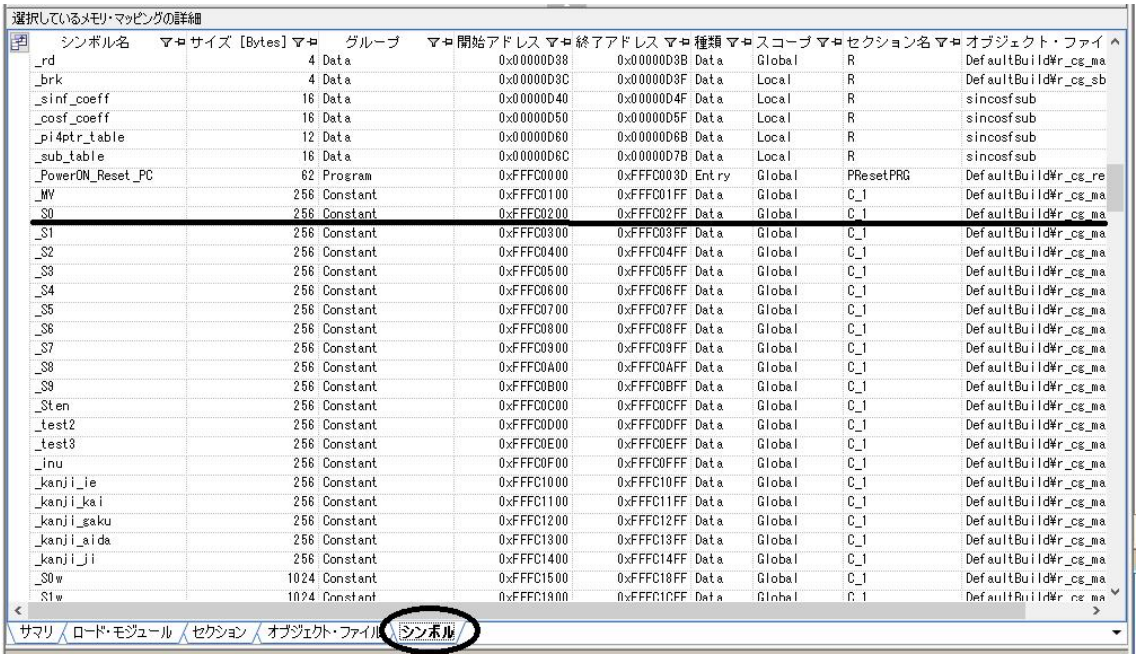
プロファイル収集にして、コンパイルするとデータが収集されます。



サマリで現在のＲＯＭ、ＲＡＭの使用状況が%で表示されます。残り%が少なくなり、スタック領域などが維持できなくなると正常動作しなくなりますので注意が必要です。



シンボルで例えば数字の０（Ｓ０　１６×１６ドットデータ）がサイズ２５６バイトでＲＯＭ領域に配置されていることが確認できます。



■ライブラリ関数

```
void lwait(long time)   //短時間のウエイト

//ILI9341 液晶コントローラ制御関数
void CMD_Send(uint8_t command)  //ILI9341 へのコマンド書き
void Data_Receive(rdata)        //ILI9341 からのデータ受信
void Data_Send(uint16_t tdata,uint16_t rdata)  //ILI9341 へのデータ書き込み、受信
void Data_Send16(uint16_t tdata)    //ILI9341 への 16bit data 書き込み

//画面制御
//RX230_lcd.h
#define LCD_BRITE_ON        //LCD  バックライト  明  オン（昼用）
#define LCD_BRITE_OFF       //LCD  バックライト  明  オフ
#define LCD_DARK_ON         //LCD  バックライト  暗  オン（夜用）
#define LCD_DARK_OFF        //LCD  バックライト  暗  オフ

//RX230_lcd.c
void gbeta(uint16_t color)  //ベタ塗り  240×320 ドット画面を color 色で塗ります。
                            //720msec 程度かかります。

void pset(uint16_t xposi,uint16_t yposi,uint16_t color)  //ピクセルセット
                            //xposi0-239  yposi0-319  の位置に color 色で 1 ドット
                            //描画します。横軸が X、縦軸が Y

uint16_t color_sel(uint8_t color_code)  //カラーコード 0～F に応じて実際に書き込
                            //む RGB データを返します。

void gwrite16(uint16_t xposi,uint16_t yposi,uint8_t *font_data)  //16×16 データ
                            //の書き込み。文字、漢字、図などを描きます。
                            //*font_data はフォントデータの開始アドレス

void gwrite32(uint16_t xposi,uint16_t yposi,uint8_t *font_data) //32×32 データ
                            //の書き込み。文字、漢字、図などを描きます。
                            //*font_data はフォントデータの開始アドレス

void gwrite32_96(uint16_t xposi,uint16_t yposi,uint8_t *font_data)//32×32 データ
                            //を 96×96(3 倍）に拡張して書き込み。文字、漢字、図
                            //などを描きます。
                            //*font_data はフォントデータの開始アドレス

void gwrite32_64(uint16_t xposi,uint16_t yposi,uint8_t *font_data)//32×32 データ
                            //を 64×64(2 倍）に拡張して書き込み。文字、漢字、図
```

```
                        //などを描きます。
                        //*font_data はフォントデータの開始アドレス
void yline(uint16_t xposi1,uint16_t yposi1,uint16_t xposi2,uint16_t yposi2,uint16_t
color)  //直線 Y 縦軸 line
                        //Y 縦軸に高速に直線を引きます。斜め線は引けません
                        //xposi1=xposi2 でないといけません
                        //yposi2 > yposi1 でないといけません
void gline(uint16_t xposi1,uint16_t yposi1,uint16_t xposi2,uint16_t yposi2,uint16_t
color)  //直線 XY  line
                    //XY 横軸に直線を引きます。斜め線は引けません
                    //Y 軸の線は yposi2  > yposi1 でないといけません
                    //X 軸の線は xposi2  >  xposi1  でないといけません
                    //X 軸描画は pset()関数を使用（時間がかかる）

void gbox(uint16_t xposi1,uint16_t yposi1,uint16_t xposi2,uint16_t yposi2,uint16_t
line_color,uint16_t fill_color)//BOX
                    //BOX  四角形を描画します。
                    //xposi1、yposi1 起点  xposi2、yposi2 終点
                    //line_color  線の色  fill_color  box の中の色
                    //xposi2  >  xposi1 でないといけません
                    //yposi2  >  yposi1 でないといけません
                    //描画に 150msec かかります（サイズで異なります）

void gcircle_fill(uint16_t xposi1,uint16_t yposi1,uint16_t radis,uint16_t color)
                    //丸を描画します。
                    //xposi1、yposi1 起点  radis 半径  color 円の中の色
                    //描画に 125msec かかります（サイズで異なります）

void gcircle_outline(uint16_t xposi1,uint16_t yposi1,uint16_t radis,uint16_t color)
                    //丸を描画します。
                    //xposi1、yposi1 起点  radis 半径  color 外周の色
                    //描画に 30msec かかります

void  gcircle_outline2(uint16_t  xposi1,uint16_t  yposi1,uint16_t  radis,uint16_t
color)
                    //丸を描画します。外周を半径を小さくして 3 回書いてます。
                    //（外周強調）
                    //xposi1、yposi1 起点  radis 半径  color 外周の色
                    //描画に 100msec かかります

void gcircle(uint16_t xposi1,uint16_t yposi1,uint16_t radis,uint16_t mode,uint16_t
color)
```

```
                        //丸を描画します。
                        //mode  0  gcircle_outline();
                        //mode  1  gcircle_fill();
                        //mode  2  gcircle_outline2();
                        //xposi1、yposi1 起点  radis 半径  color 外周の色
                        //描画に～100msec かかります

void Dsiplay_ON(void)   //液晶表示 ON
void Dsiplay_OFF(void)  //液晶表示 OFF

void init_lcd(void)     //液晶初期設定、1msec 定周期タイマー ON

//タッチスイッチ
void T_com_write(uint8_t comm)  //タッチスイッチ  コマンドライト

void touch_switch_read(void)    //タッチスイッチ  データ読み込み
                                //TX_data  に  X 軸データ  8bit  が入ります。
                                //TY_data  に  Y 軸データ  8bit  が入ります。
                                //58μsec かかります。1msec 定周期割り込みで呼んで
                                //も 10%以下の消費時間。
//テスト表示

void test_kanji(void)           //16×16  漢字  学間家字  を全画面に書く

void test_zukei(void)
{
        gbox(50,50,200,200,clBlue,clAqua);      //四角
        gcircle(100,100,50,2,clGreen);          //mode 2 円周強調  丸
        gcircle(120,140,25,1,clRed);            //mode 1 内部ベタ塗  丸
        gcircle(220,220,15,0,clBlue);           //mode 0 円周描画  丸
}

void test_zukei2(void)
{
                gwrite32_64(90,10,test_color);
                            //32×32  16 色データを 64×64(2 倍) に拡大して表示
                gwrite32_96(140,100,test);
                            //32×32  データを 96×96(3 倍) に拡大して表示
                gwrite32_96(140,220,test_color);
                            //32×32  16 色データを 96×96(3 倍) に拡大して表示
                gwrite32_64(10,220,test);
                            //32×32  データを 64×64(2 倍) に拡大して表示
```

```
}

void test_kanji3(void)
{
                gwrite32_64(90,10,kanji_mae);
                        //32×32　漢字　前を 64×64(2 倍) に拡大して表示
                gwrite32(200,10,kanji_usiro);
                        //32×32　漢字　後を表示
                gwrite32_96(140,220,kanji_syutu);
                        //32×32　漢字　出を 96×96(3 倍) に拡大して表示
                gwrite32_96(140,100,kanji_riki);
                        //32×32　漢字　力を 96×96(3 倍) に拡大して表示

                gwrite32(10,280,kanji_mae);     //32×32　漢字　前を表示
                gwrite32(42,280,kanji_usiro);   //32×32　漢字　後を表示
                gwrite32(10,240,kanji_syutu);   //32×32　漢字　出を表示
                gwrite32(42,240,kanji_riki);    //32×32　漢字　力を表示

}
void test_char(void)
{
        for(yloop = 0;yloop < 320;yloop+=16)    //画面いっぱいに書きます。
        {
            for(xloop = 0;xloop < 240;xloop+=64)
            {
                gwrite16(xloop,yloop,kao1);     //16×16　人の顔 1
                gwrite16(xloop+16,yloop,inu2);  //16×16　犬の顔 2
                gwrite16(xloop+32,yloop,kao2);  //16×16　人の顔 2
                gwrite16(xloop+48,yloop,inu3);  //16×16　犬の顔 3

            }
        }
}
```

//数字表示関数

```
//16×16、32×32 ドットの数字フォント開始アドレスをセットする
void cv_suuji(uint8_t bangou,uint8_t fonts_size)
```

lcd_puts 関数などで、データが数字ではなく、小数点だった場合に自動的に使われる隣との表示間隔を狭め自然な表示にする関数群。32、64, 96 ドット用があります。

//点専用縦4列書き込み void gwrite32_64_ten_color(uint16_t xposi,uint16_t yposi,uint8_t *font_data,uint16_t color1,uint16_t color2)

//点専用縦4列書き込み
void gwrite32_ten_color(uint16_t xposi,uint16_t yposi,uint8_t *font_data,uint16_t color1,uint16_t color2)

液晶に数字を表示する関数　色情報はフォントデータのまま表示される。フォントサイズ16,32,64ドットを選択できる。
void lcd_puts(uint16_t xposi,uint16_t yposi,uint8_t fonts_size,uint8_t *data)

液晶に数字を表示する関数　フォントデータの文字、背景の色を変えて表示できる。
void lcd_puts_color(uint16_t xposi,uint16_t yposi,uint8_t fonts_size,uint8_t *data,uint16_t color1,uint16_t color2)
//xposi、 yposi、 fontsize(16,32,64)、 *dataの先頭アドレス、color1 黒データの変換色、coloer2　白データの変換色

//各種表示コンポーネント　具体的なコンポーネンツの使い方は■動作　デモンストレーションをご参照下さい。

回転するメーター表示　0-4095入力を0-9の10段階の位置に黄色、赤色の針で示します。サイズは16, 32, 64, 96ドット

//メーター黄色　0-9まで動くメーター
void meter_yellow(uint16_t xposi,uint16_t yposi,uint8_t m_size,uint16_t data)

//メーター赤色　0-9まで動くメーター
void meter_red(uint16_t xposi,uint16_t yposi,uint8_t m_size,uint16_t data)

レベルメーター　入力0-4095を12分割し、10迄は緑、11黄色、12赤で表示します。サイズは32, 64ドット。

//レベルメーター初期化
void init_level_meter(uint16_t xposi,uint16_t yposi,uint8_t size)
//レベルメーター　入力を12分割し、10迄は緑、11黄色、12赤で表示します。
void level_meter(uint16_t xposi,uint16_t yposi,uint8_t size ,uint16_t data)

縦棒、点グラフ描画
//縦棒、点グラフ初期化
void init_Vbar_chart(uint16_t xposi,uint16_t yposi,uint8_t size)

縦棒（X 軸)、点グラフ描画　開始、終了 X 座標指定　描画 64、96 ドット　色は選択 16 色、入力データ 0-4095
void Vbar_chart(uint16_t xposi_start,uint16_t xposi_end,uint16_t yposi,uint8_t size,uint8_t mode,uint16_t color,uint16_t addata)

//タッチスイッチテスト　タッチスイッチを押すたびに画像が変わります。
void touch_switch_test(void)

■動作

●デモンストレーション１　RXD230_LCD_test1

画面ベタ塗り、図形表示、漢字表示、メーター表示、レベルメーター表示、タッチスイッチで表示切替

●動作が見える youtube　URL　https://www.youtube.com/watch?v=0JpwdZ0Hc1E

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */

//液晶初期化
        init_lcd(); ①

//バックライト　明るい
//      LCD_DARK_ON;            //LCD バックライト ON　ダーク　夜用
        LCD_BRITE_ON;②          //LCD バックライト ON　明るい　昼用

        gbeta(clWhite);③        //ベタ塗

        test_kanji();   ④       //漢字表示
        test_zukei();   ⑤       //図形表示
        test_zukei2();  ⑥       //図形 2 表示
        test_kanji3();  ⑦       //漢字 3 表示

        init_lm_flg = 0;  ⑧     //レベルメーター初期化

        while(1)  ⑨
        {
            for(loop5 = 0;loop5 < 10;loop5++)  ⑩
            {
                touch_switch_test();  ⑪             //タッチスイッチテスト

                level_meter(10,80,fsize_32,loop5*455);      ⑫  //レベルメーター　140msec
                meter_yellow(10,150,fsize_64,loop5*455);    ⑬  //メーター黄色
                meter_red(10,10,fsize_32,loop5*455);        ⑭  //メーター赤色
                if(loop5 == 9)                              ⑮
                {
                    int_wait(500);
                }
            }
        }
```

```
    /* End user code. Do not edit comment generated here */
}
```

○プログラム解説

```
//液晶初期化
        init_lcd(); ①
```

①液晶の初期化です。関数はライブラリ RX230_lcd.c の中にあります。RX230 と液晶のやり取りを行う SPI インターフェイスの初期化と定周期タイマー割り込み 1msec の初期化を行っています。

```
//バックライト　明るい
//          LCD_DARK_ON;            //LCD バックライト ON　ダーク　夜用
            LCD_BRITE_ON;②          //LCD バックライト ON　明るい　昼用
```

②液晶のバックライトの明るさは 2 段階に調整出来ます。昼用、夜用を切り替え可能です。

```
            gbeta(clWhite);③        //ベタ塗
```

③電源ＯＮ時にグラフィックＲＡＭを白色で塗っています。

```
            test_kanji();   ④       //漢字表示
```

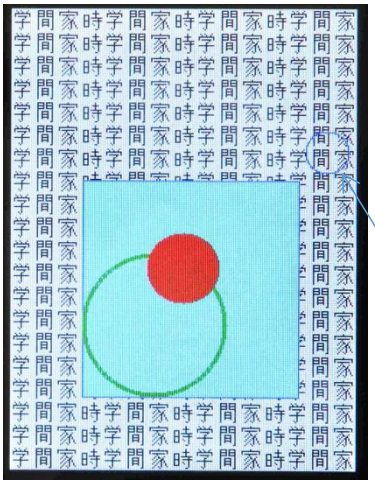
④16×16 の「学問家時」という漢字が画面いっぱいに表示されます。縦 20 文字、横 15 文字、合計 300 文字。

関数の中身を見ると学問家時の漢字 4 文字を横（X）64 ドット、縦（Y）16 ドット毎に書いているのが分かります。

```
void test_kanji(void)       //16×16　学問家字を全画面に書く
{
        for(yloop = 0;yloop < 320;yloop+=16)
        {
            for(xloop = 0;xloop < 240;xloop+=64)
```

```
                {
                    gwrite16(xloop,yloop,kanji_gaku);
                    gwrite16(xloop+16,yloop,kanji_aida);
                    gwrite16(xloop+32,yloop,kanji_ie);
                    gwrite16(xloop+48,yloop,kanji_ji);

                }
            }
}
            test_zukei();      ⑤          //図形表示
```

Y:320



Y:0

X:0 X:240

図形を書いています（以下プログラム） ブルーの円

```
void test_zukei(void)
{
            gbox(50,50,200,200,clBlue,clAqua);          //四角
            gcircle(100,100,50,2,clGreen);              //mode 2 円周強調　丸
            gcircle(120,140,25,1,clRed);            //mode 1 内部ベタ塗　丸
            gcircle(220,220,15,0,clBlue);               //mode 0 円周描画　丸
}
```

初めに

```
gbox(50,50,200,200,clBlue,clAqua);          //四角
```

起点が X:50、Y:50 終点が X：200、Y:200 の外が青、中がアクア（青緑）色の四角形を描く

```
gcircle(100,100,50,2,clGreen);              //mode 2 円周強調　丸
```

起点が X:100、Y:100、半径 50、描画 mode:円周強調、緑色の円を描く

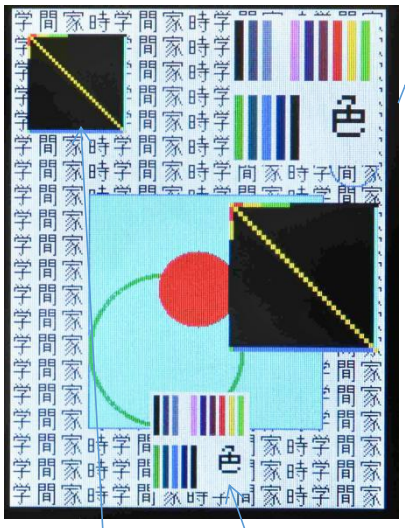
```
gcircle(120,140,25,1,clRed);                //mode 1 内部ベタ塗　丸
```

起点が X:120、Y:140、半径 25、描画 mode：内部赤で塗る

```
gcircle(220,220,15,0,clBlue);               //mode 0 円周描画　丸
```

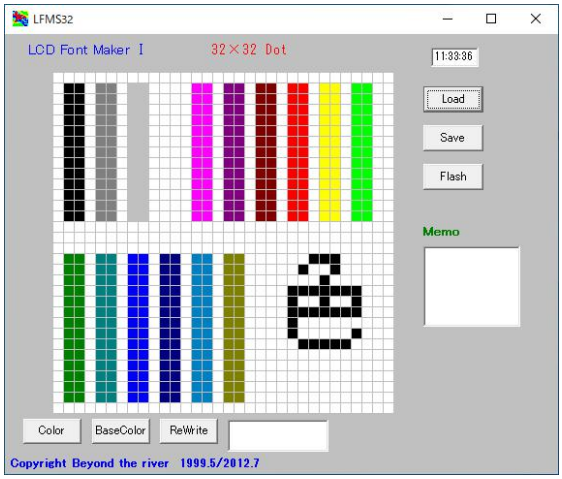
起点が X:220、Y:220、半径 15、描画 mode：通常、青色描画　見えにくいですがブルーの円です。

```
        test_zukei2();     ⑥          //図形 2 表示
```

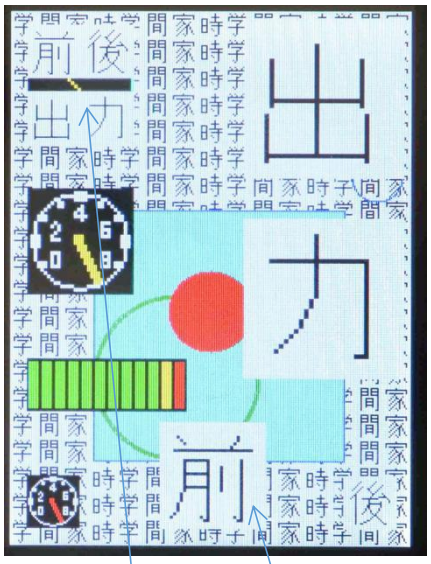


```
void test_zukei2(void)              //
{
                    gwrite32_64(90,10,test_color);     //32×32   16 色データを 64×64(2 倍）に拡大して表示
                    gwrite32_96(140,100,test);         //32×32   データを 96×96(3 倍）に拡大して表示
                    gwrite32_96(140,220,test_color); //32×32   16 色データを 96×96(3 倍）に拡大して表示
          gwrite32_64(10,240,test);          //32×32   データを 64×64(2 倍）に拡大して表示

}
```

gwrite32_64（）関数は 32×32 ドットデータを 2 倍の 64×64 に表示する関数です。
gwrite32_96（）関数は 3 倍の 96×96 ドットデータに拡大し表示します。使い方は後述しますが、フォントメーカーで 32 ドットデータを作成すれば 32, 64, 96 ドット、3 種類の大きさの文字、図形が液晶画面に表示できます。



test_kanji3(); ⑦ //漢字 3 表示



```
void test_kanji3(void)
{
        gwrite32_64(90,10,kanji_mae);    //32×32  漢字  前を 64×64(2 倍）に拡大して表示
        gwrite32(200,10,kanji_usiro);     //32×32  漢字  後を表示
        gwrite32_96(140,220,kanji_syutu);//32×32  漢字  出を 96×96(3 倍）に拡大して表示
        gwrite32_96(140,100,kanji_riki); //32×32  漢字  力を 96×96(3 倍）に拡大して表示

        gwrite32(10,280,kanji_mae);       //32×32  漢字  前を表示
        gwrite32(42,280,kanji_usiro);     //32×32  漢字  後を表示
        gwrite32(10,240,kanji_syutu);     //32×32  漢字  出を表示
        gwrite32(42,240,kanji_riki);      //32×32  漢字  力を表示

}
```

本ライブラリは一般的なライブラリのようにｐｓｅｔ（）、ｌｉｎｅ（）などで表示させる

絵を作ることも可能ですが、固定された形であれば、あらかじめフォントメーカーで 16×16、32×32 ドットの表示したい文字や絵を作り、関数で 16、32、64、96 ドット表示させる方が断然、描画が高速です。

```
init_lm_flg = 0;  ⑧          //レベルメーター初期化
```

⑧はこの後、使うレベルメーターコンポーネントの初期化です。

```
while(1)   ⑨
        {
```

⑨これ以降、無限ループ動作です。

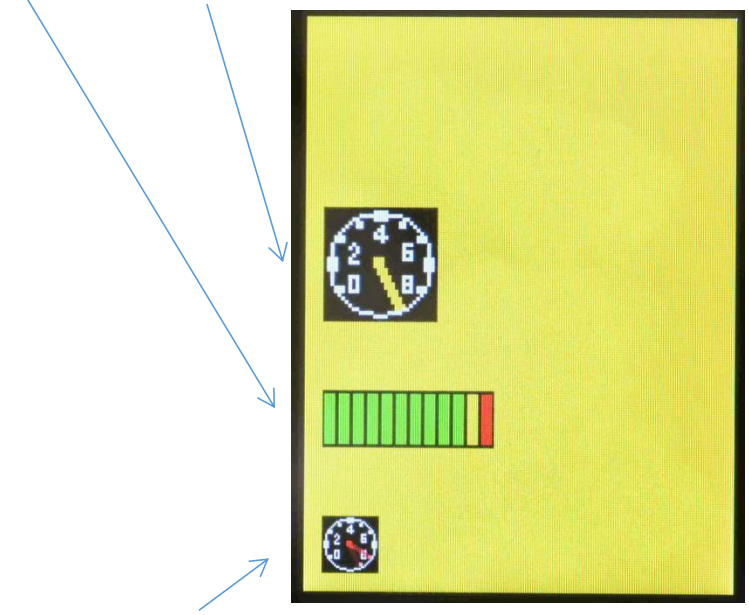
```
for(loop5 = 0;loop5 < 10;loop5++)   ⑩
{
```

Ｉｏｏｐ５は０から９まで変化します。

```
touch_switch_test();   ⑪                    //タッチスイッチテスト
```

液晶表面のタッチスイッチのデータを読み、各種描画を行います。タッチスイッチは押された座標がそれぞれ８ｂｉｔデータ　で変数 TX__ｄａｔａ，ＴＹ__ｄａｔａに格納されます。この関数では、押されるたびに異なる図形を描画していて、押された位置情報は参照していません。

```
level_meter(10,80,fsize_32,loop5*455);          ⑫ //レベルメーター　140msec
meter_yellow(10,150,fsize_64,loop5*455);          ⑬  //メーター黄色
```



```
meter_red(10,10,fsize_32,loop5*455);   ⑭//メーター赤色

if(loop5 == 9)                                ⑮
```

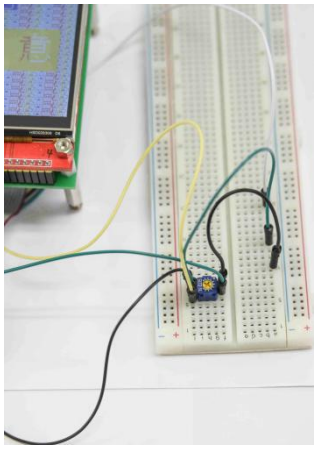
```
{
    int_wait(500);
}
```

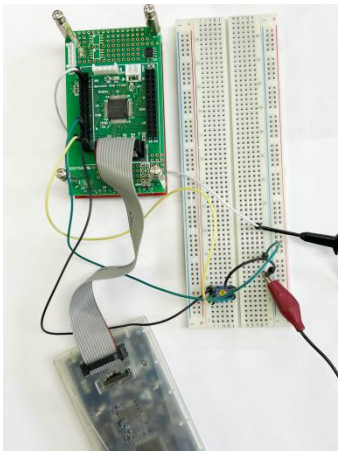
Ｌｏｏｐ５が９の時だけ、見た目の区切りのために 500msec のウエイトを入れて止めています。このウエイト関数は 1msec 定周期割り込みで作成されていて、極めて高精度です。

●デモンストレーション２　RXD230_LCD_AD1
ＡＤコンバータ値を数値、レベルメーター、縦棒グラフ、メーターで表示

ＡＤコンバータの値を数値で表示し、縦棒グラフ表示で時間経過によるデータの変化を示すことが出来ます。

●動作が見える youtube　URL　https://www.youtube.com/watch?v=KV62mYcnDxM






```
//12bit AD スタート
            R_S12AD_Start();   ①
//          LED_ON;                 //時間計測
            gbeta(clWhite);                         //ベタ塗

            init_lm_flg = 0;                        //レベルメーター初期化
            init_vc_flg = 0;   ②                    //縦棒グラフ初期化
```

```
        gwrite16(5,5,S0);       ③                       //縦棒グラフ目盛り付け
        gwrite16(5,75,S3);
        gwrite16(20,75,Sten);
        gwrite16(35,75,S3);
        gwrite16(50,75,V);


        while(1)
        {
            S12AD.ADCSR.BIT.ADST = 1;       ④           //AD 変換開始
            while(S12AD.ADCSR.BIT.ADST)
                ;
            ad0 = S12AD.ADDR0;              ⑤           //AD 変換終了　10μsec
            fdata1 = ad0/(4095/3.3);        ⑥           //演算開始
            sprintf(ad_buff,"%1.2f¥n",fdata1);      ⑦

            if(ad0 < lm_yellow_level)       ⑧
            {
                lcd_puts_color(20,100,fsize_64,ad_buff,clGreen,clWhite);//緑文字　演算開始からここま
で 180msec
            }
            else
            {
                if(ad0 < lm_red_level)      ⑨
                {
                    lcd_puts_color(20,100,fsize_64,ad_buff,clYellow,clWhite);   //黄色文字
                }
                else
                {
                    lcd_puts_color(20,100,fsize_64,ad_buff,clRed,clWhite);      //赤文字
                }

            }

            meter_yellow(10,180,fsize_64,ad0);      ⑩       //meter　46msec
            level_meter(10,250,fsize_64,ad0);       ⑪       //level_meter 140msec
            Vbar_chart(30,222,5,fsize_64,0,clYellow,ad0);   ⑫
                                        //x:開始、x:終了、y:y 座標、サイズ、描画モード、色
                                                //縦棒グラフ　500μsec
                                                //1 ループ　360msec 程度　1 秒間に約 3 回
        }
```

○プログラム解説

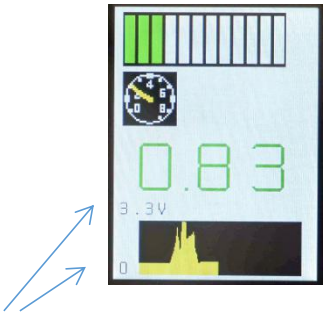
```
//12bit AD スタート
        R_S12AD_Start();   ①
```

①コード生成機能で自動的に作られる AD コンバータ初期設定関数です。

```
//          LED_ON;                 //時間計測
            gbeta(clWhite);                         //ベタ塗

            init_lm_flg = 0;                        //レベルメーター初期化
            init_vc_flg = 0;   ②                    //縦棒グラフ初期化
```

②縦棒グラフの初期設定フラグです。

```
            gwrite16(5,5,S0);       ③                  //縦棒グラフ目盛り付け
            gwrite16(5,75,S3);
            gwrite16(20,75,Sten);
            gwrite16(35,75,S3);
            gwrite16(50,75,V);
```



③縦棒グラフのスケール 0，3.3V を表示しています。

```
            while(1)
            {
                S12AD.ADCSR.BIT.ADST = 1;       ④          //AD 変換開始
                while(S12AD.ADCSR.BIT.ADST)
                    ;
```

④AD 変換を開始し、データが揃うのを待ちます。

```
                ad0 = S12AD.ADDR0;              ⑤          //AD 変換終了   10μsec
```

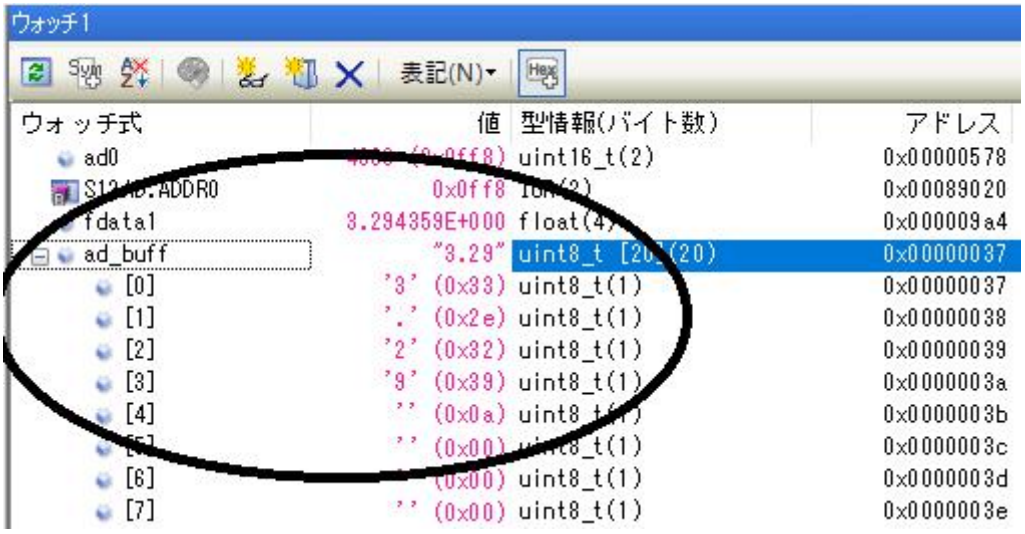
⑤データが揃ったら、変数ａｄ０に入れます。

```
                fdata1 = ad0/(4095/3.3);        ⑥          //演算開始
```

⑥AD コンバータデータは分解能 12bit ですので、0-4095 迄変わります。4095 で 3.3V と表示するために演算を行います。

```
sprintf(ad_buff,"%1.2f¥n",fdata1);        ⑦
```

⑦16 進数データを表示するためにＡＳＣＩＩ文字に変換し、ａｄ＿ｂｕｆｆに整数１桁、小数点以下２桁に変換します。



例えば入力が 3.29V の場合、上記のように１数字毎にａｄ＿ｂｕｆｆ［　］に 3.29 と格納されます。

```
if(ad0 < lm_yellow_level)                ⑧
{
    lcd_puts_color(20,100,fsize_64,ad_buff,clGreen,clWhite);//緑文字　演算開始からここまで 180msec
}
```

⑧レベルメーターは通常、緑表示ですが、lm_yellow_level を越て、lm_red_level 以下の場合、黄色、以上の場合、赤表示になります、ここでは数値表示も同期して色を変えています。現在の設定は RX230_lcd.h の中になります、黄色が 3597/4095×3.3V≒2.9V、赤が 3924/4095×3.3V＝3.16V です。

```
#define lm_yellow_level 3597
#define lm_red_level 3924
```

```
else
{
    if(ad0 < lm_red_level)               ⑨
    {
        lcd_puts_color(20,100,fsize_64,ad_buff,clYellow,clWhite);      //黄色文字
    }
```

⑨lm_yellow_level 以上で、lm_red_level 以下の場合、文字黄色、背景色白にしています。
例 1.90→3.12
lm_red_level 以上で　文字赤、背景色白にしています。　例 3.12→3.30

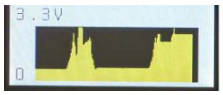
```
else
{
```

```
                lcd_puts_color(20,100,fsize_64,ad_buff,clRed,clWhite);          //赤文字
            }

        }

        meter_yellow(10,180,fsize_64,ad0);      ⑩           //meter   46msec
```

⑩回転するメーターです。ａｄＯの値 0-4095 を 0-9 の針の位置で表示します。

```
        level_meter(10,250,fsize_64,ad0);       ⑪           //level_meter 140msec
```

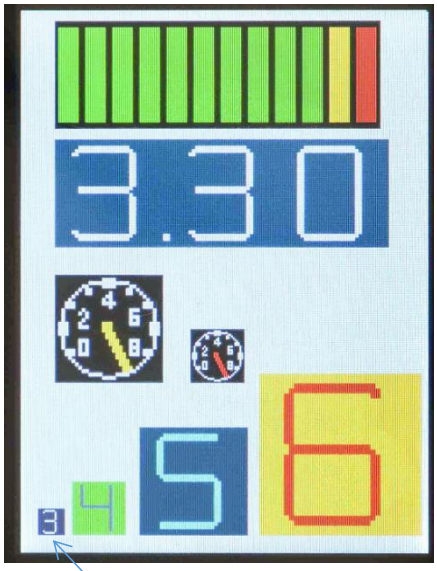
⑪レベルメーターです。ａｄＯの値 0-4095 を 0-11 分割して位置で表示します。



⑫縦棒グラフです。Y 軸高さ 64（入力電圧レベル）、X 軸 64×3=192 ドット（時間軸）の棒グラフを作成します。黄色線で描画します。オシロスコープのように時間軸での電圧変化が目視出来ます。

```
        Vbar_chart(30,222,5,fsize_64,0,clYellow,ad0);   ⑫
                                        //x:開始、x:終了、y:y 座標、サイズ、描画モード、色
                                            //縦棒グラフ   500μsec
                                            //1 ループ   360msec 程度   1 秒間に約 3 回
    }
```

●デモンストレーション３　RXD230_LCD_AD2
ＡＤコンバータ値を数値、レベルメーター、メーターで表示
文字、背景の色を独立して 16 色ずつ設定できる関数を使って文字表示。
●動作が見える youtube　URL　https://www.youtube.com/watch?v=3CuG_DaCR30




```
gwrite16_color(10,10,S3,clWhite,clGray);          ①    //文字表示
gwrite32_color(30,10,S4w,clSilver,clLime);        ②
gwrite32_64_color(70,10,S5w,clAqua,clTeal);       ③
gwrite32_96_color(140,10,S6w,clRed,clYellow);     ④
```

○プログラム解説
既説明の部分は省略します。

```
gwrite16_color(10,10,S3,clWhite,clGray);          ①    //文字表示
```

①１６×１６ドット、数字の３を白色、背景グレーで X:10、Y10:の位置に表示。

```
gwrite32_color(30,10,S4w,clSilver,clLime);        ②
```

②３２×３２ドット、数字の４をシルバー、背景をライムで X:30, Y:10 の位置に表示。

```
gwrite32_64_color(70,10,S5w,clAqua,clTeal);       ③
```

③３２×３２データ、数字の５を６４ドットに拡張し、アクア、背景をテイルで X:70, Y10 の位置に表示

```
gwrite32_96_color(140,10,S6w,clRed,clYellow);     ④
```

④３２×３２のデータ、数字の６を９６ドットに拡張し、赤、背景を黄色で X:140, Y10 の位置に表示

●デモンストレーション４　RXD230_LCD_安全

安全、注意、危険の表示とタッチスイッチによる表示切替。安全、注意、危険を漢字や色で表示でき、より人間に理解でき易い表示になります。

●動作が見えるｙｏｕｔｕｂｅＵＲＬ　https://www.youtube.com/watch?v=D5WdA6a_UUE

●プログラム

```
while(1)
{

    cnt++;if(cnt > 12){cnt = 0;}        ①


    sprintf(ad_buff,"%1.2f¥n",cnt/3.63);    ②
    lcd_puts_color(20,240,fsize_32,ad_buff,clWhite,clNavy); //xposi yposi mode(16,32,64,96) data[]
    level_meter(60,280,fsize_32,cnt*341);          ③     //level_meter 140msec
    meter_yellow(20,280,fsize_32,cnt*341);         ④     //meter    46msec

    if(cnt < 11)       ⑤
    {
        gwrite32_96_color(20,100,an,clWhite,clLime);              //安
        gwrite32_96_color(20+96,100,zen,clWhite,clLime);          //全
    }
    else
    {
        if(cnt < 12)
        {
            for(cnt2 = 0;cnt2 < 5;cnt2++)        ⑥
            {
```

```
                    gwrite32_96_color(20,100,cyuu,clWhite,clYellow); //注
                    gwrite32_96_color(20+96,100,i,clWhite,clYellow); //意
                    int_wait(100);
                    gwrite32_96_color(20,100,cyuu,clRed,clYellow);   //注
                    gwrite32_96_color(20+96,100,i,clRed,clYellow);   //意
                    int_wait(100);

                    touch_switch_test();

                }
            }
            else
            {
                gwrite32_96_color(20,100,ki,clWhite,clRed);          //危      ⑦
                gwrite32_96_color(20+96,100,ken,clWhite,clRed);      //険
                int_wait(2000);

            }

        }

        touch_switch_test();

    }
```

○プログラム解説

```
cnt++;if(cnt > 12){cnt = 0;}      ①
```

①このプログラムでは AD の値でなく、変数ｃｎｔを０から１２まで変えて動作を見ています。

```
        sprintf(ad_buff,"%1.2f¥n",cnt/3.63);   ②
```

②ｃｎｔ＝１２のときに３．３Ｖと表示するためにｃｎｔを 3.63 で割っています。

```
        lcd_puts_color(20,240,fsize_32,ad_buff,clWhite,clNavy); //xposi yposi mode(16,32,64,96) data[]
        level_meter(60,280,fsize_32,cnt*341);        ③    //level_meter 140msec
```

③レベルメーターを表示するためにｃｎｔに 341 を掛けています。12×341＝4092≒4095

```
        meter_yellow(20,280,fsize_32,cnt*341);       ④    //meter   46msec
```

④黄色メーターを表示するためにｃｎｔに 341 を掛けてます。12×341＝4092≒4095

```
    if(cnt < 11)      ⑤
        {
            gwrite32_96_color(20,100,an,clWhite,clLime);          //安
```

```
            gwrite32_96_color(20+96,100,zen,clWhite,clLime);        //全
        }
```

⑤ｃｎｔが 11 以下では「安全」とライム色、背景白で表示します。96×96 ドットです。

```
        else
        {
            if(cnt < 12)
            {
                for(cnt2 = 0;cnt2 < 5;cnt2++)        ⑥
                {
                    gwrite32_96_color(20,100,cyuu,clWhite,clYellow); //注
                    gwrite32_96_color(20+96,100,i,clWhite,clYellow); //意
                    int_wait(100);
                    gwrite32_96_color(20,100,cyuu,clRed,clYellow);    //注
                    gwrite32_96_color(20+96,100,i,clRed,clYellow);    //意
                    int_wait(100);

                    touch_switch_test();

                }
            }
```

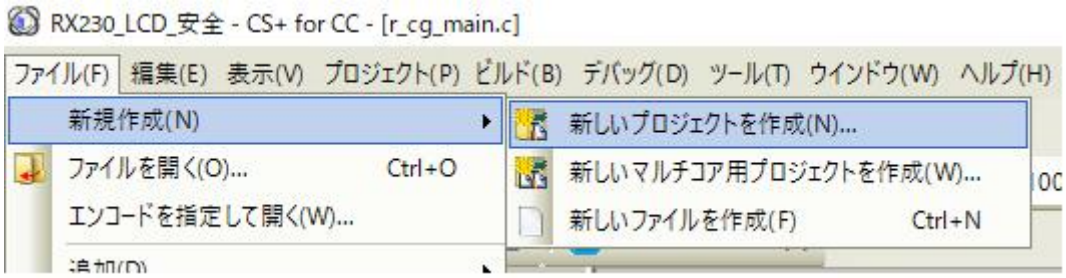
⑥ｃｎｔが 11 以上、12 以下で「注意」が表示されますが、より注意を喚起するために白文字、背景黄色、
赤文字、背景黄色を 100msec の時間を挟んで 5 回繰り返しています。
タッチスイッチデータ読み込んで、タッチがあれば各種表示します。

```
            else
            {
                gwrite32_96_color(20,100,ki,clWhite,clRed);              //危        ⑦
                gwrite32_96_color(20+96,100,ken,clWhite,clRed);          //険
                int_wait(2000);

            }
```

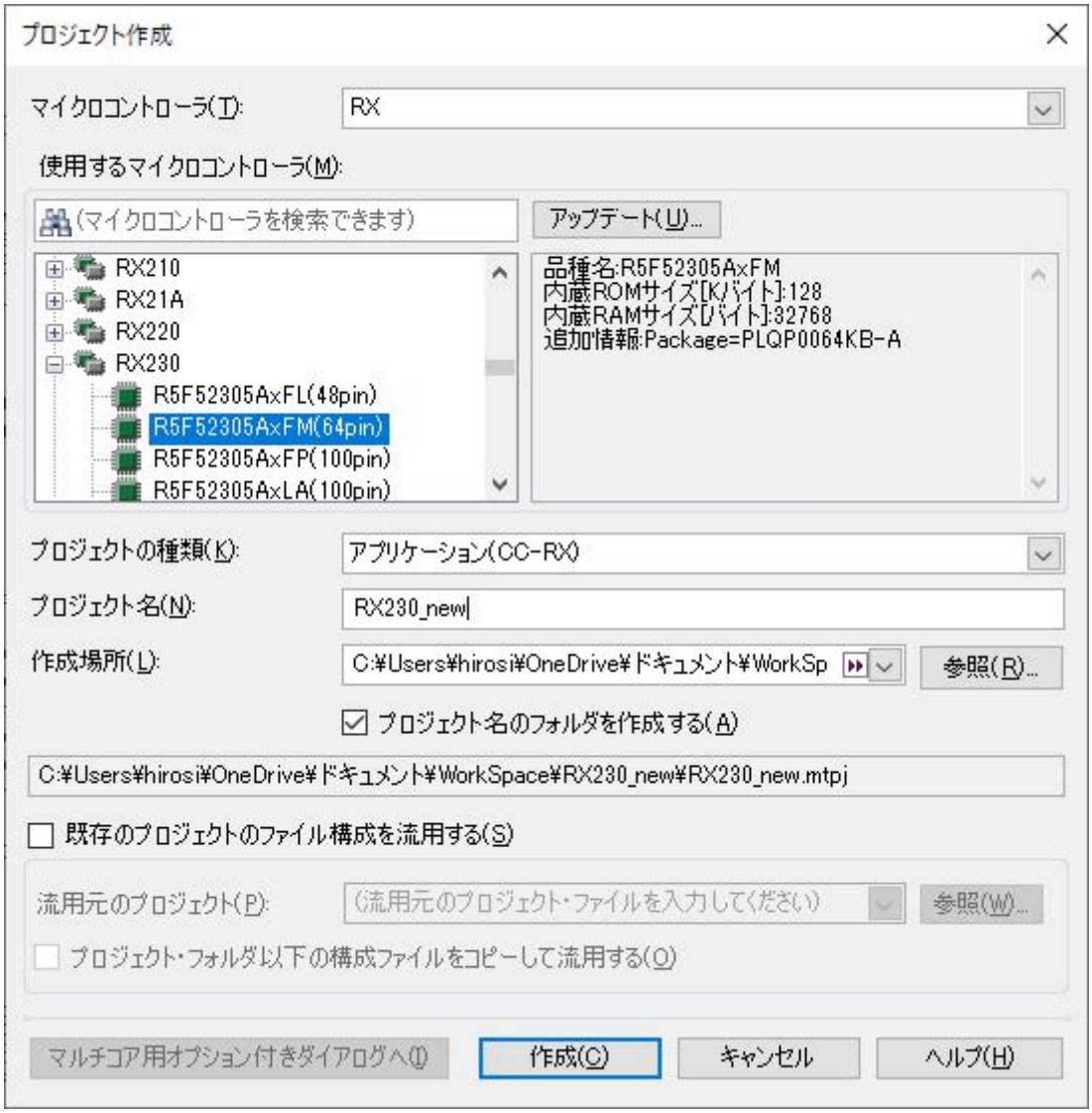
⑦12 で「危険」と表示します。白文字、背景赤、96×96 ドットサイズで 2 秒間表示します。

■備考

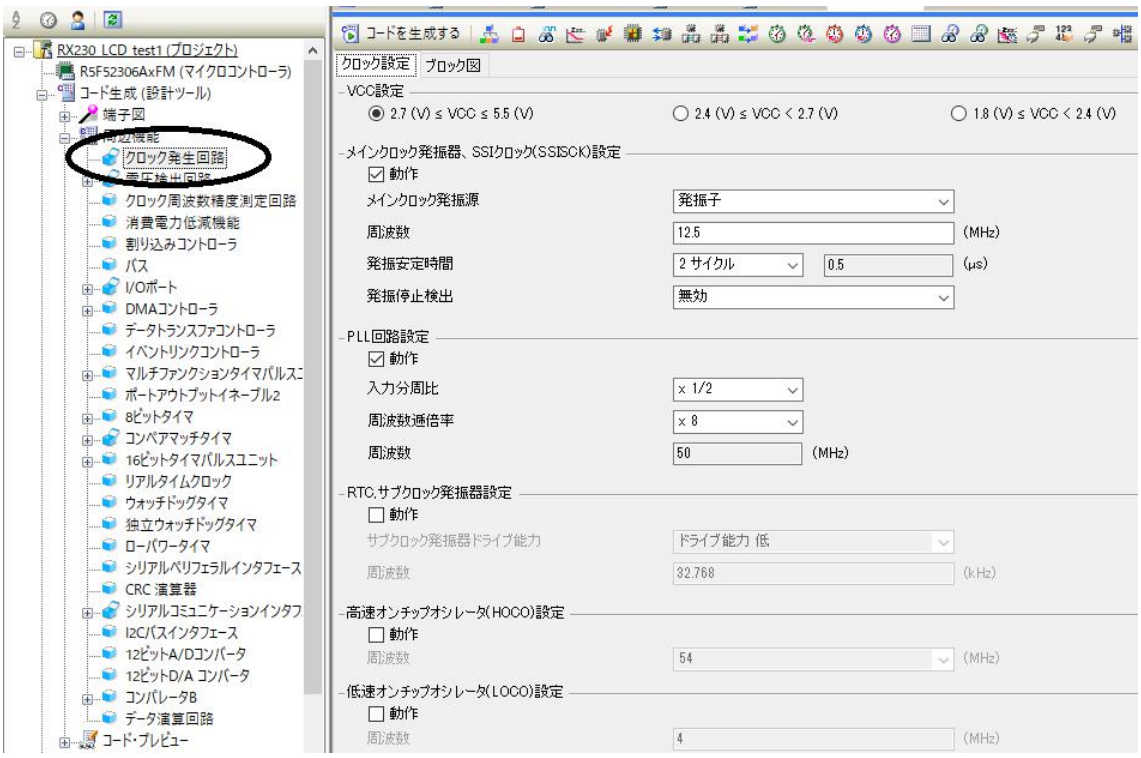
●ＣＳ＋　ｆｏｒ　ＣＣ　初めからプログラムを作る設定

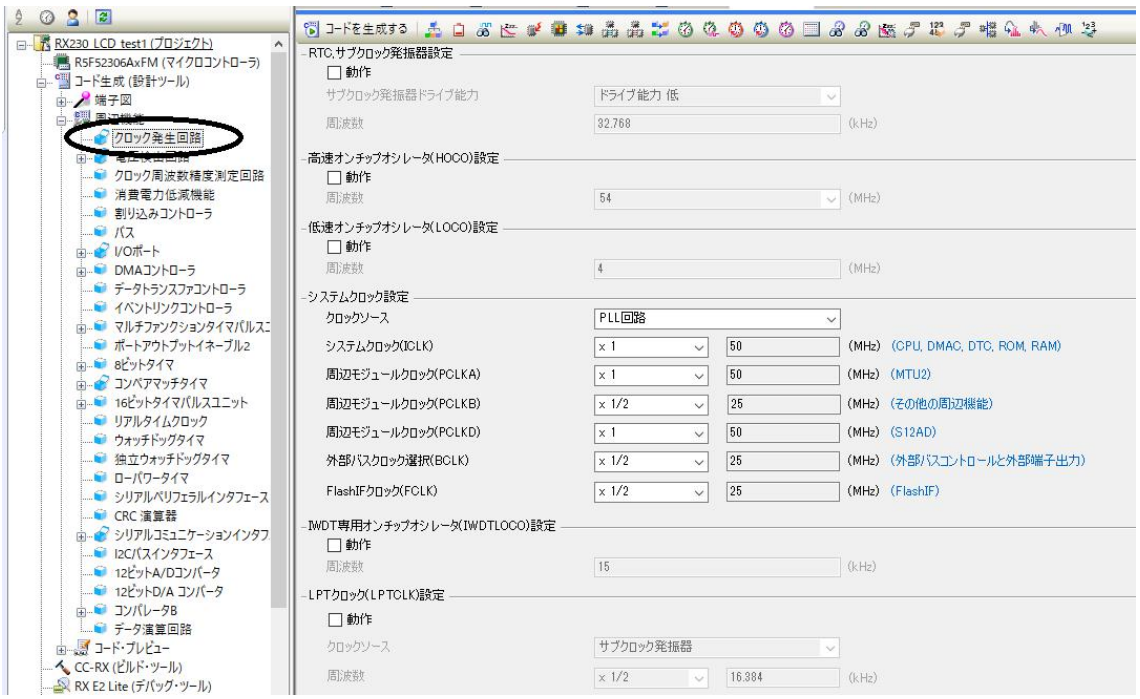


64pin の R5F52305AxF を選択します。ファイル名はここでは RX230_new にしました。

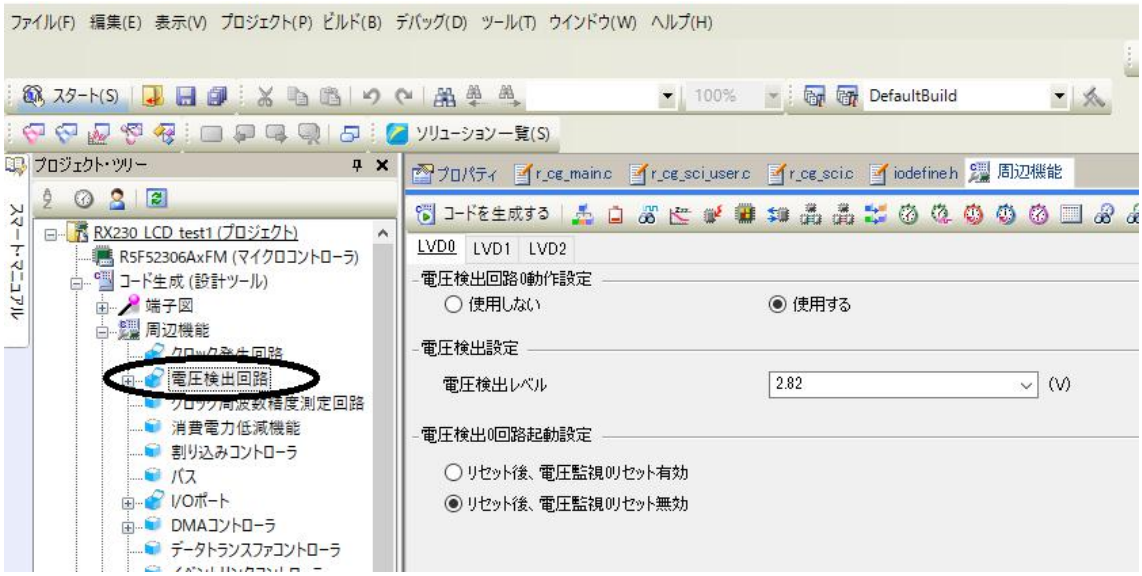


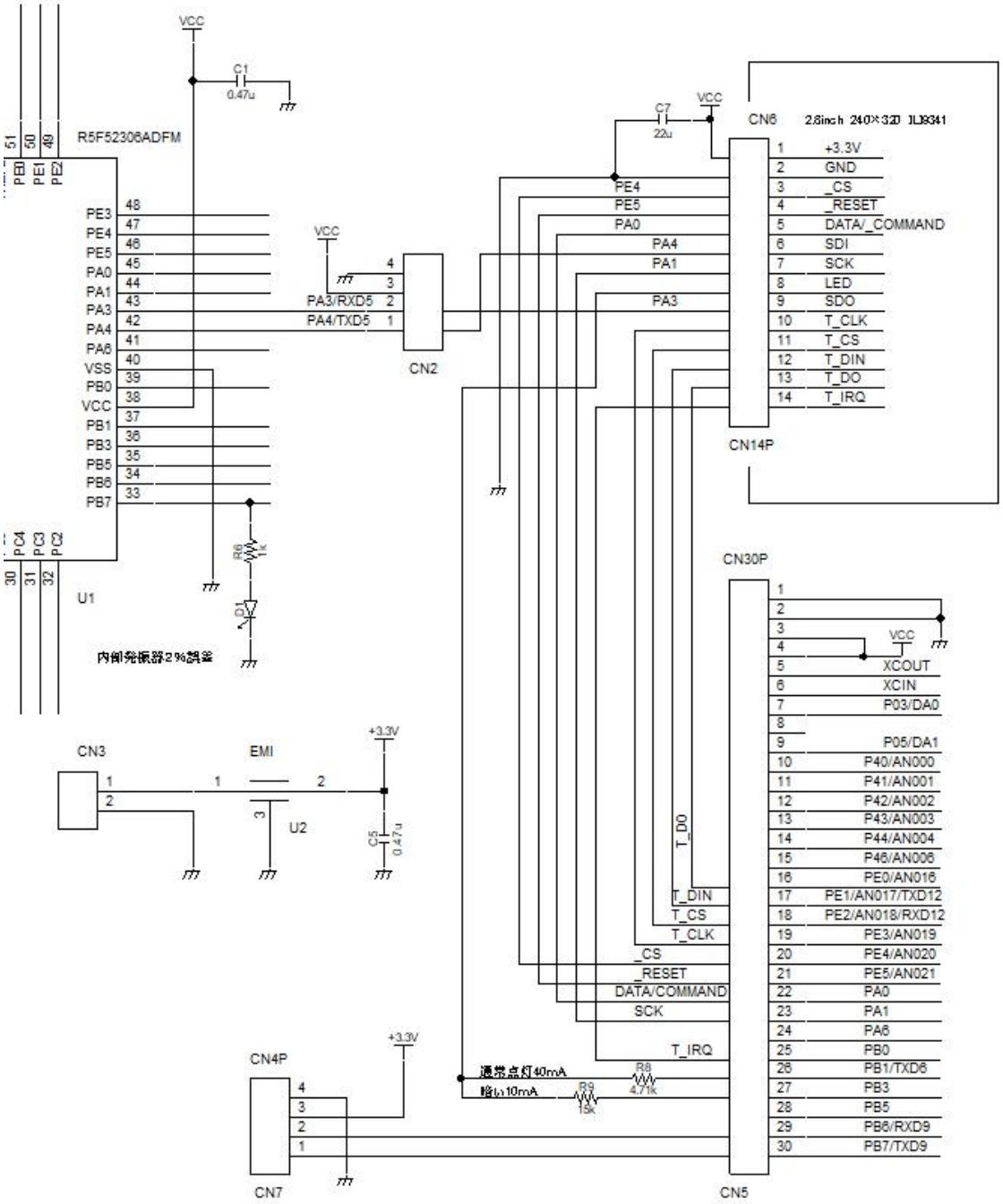
クロックは外部の１２．５ＭＨｚを使います。ＣＰＵは５０ＭＨｚで動かします。





電圧検出回路はリセット電圧を決めます。液晶板が３．３Ｖ動作なので、ＲＸ２３０も３．３Ｖで動かし、３．３Ｖより下の２．８２Ｖにします。





ＲＸ２３０マイコンと液晶表示器の接続はＳＰＩバス、タッチスイッチ、__ＣＳ（チップセレクト）、__ＲＥＳＥＴ、バックライト選択などの線が接続されています。

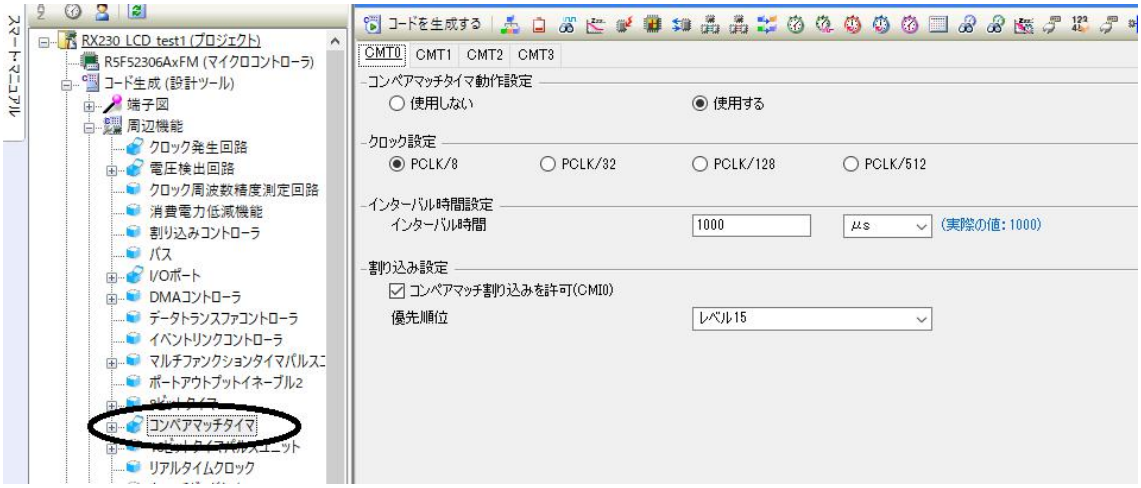
I/O ポート

PA ポートは PA1 が SCL、PA3 が RXD5、PA4 が TXD5 と SPI インターフェイスに使われるため、使用しない設定にする必要があります。

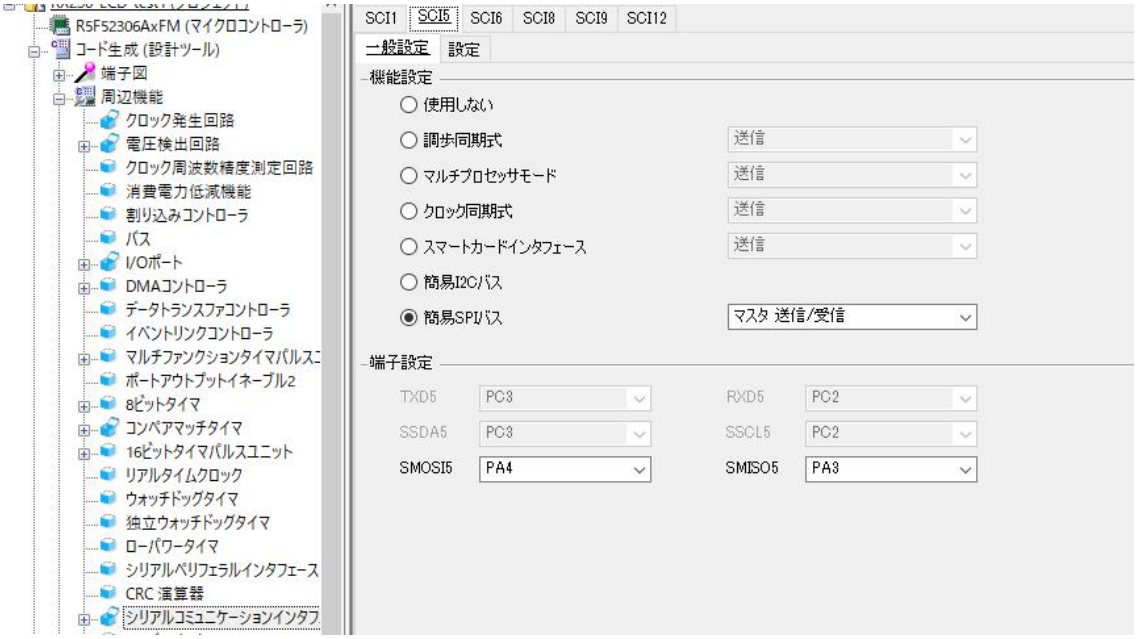
PB ポートは PB0 がタッチスイッチの T__IRQ、PB1 が液晶バックライト：通常点灯（昼用）、PB3 が夜間点灯（夜用）を制御します。PB7 は基板に実装済の LD1 に抵抗を介して接続されていて、各種インジケート、関数時間計測に使用します。

PE ポートはタッチスイッチインターフェイス、液晶の__CS や__RESET に使用します。

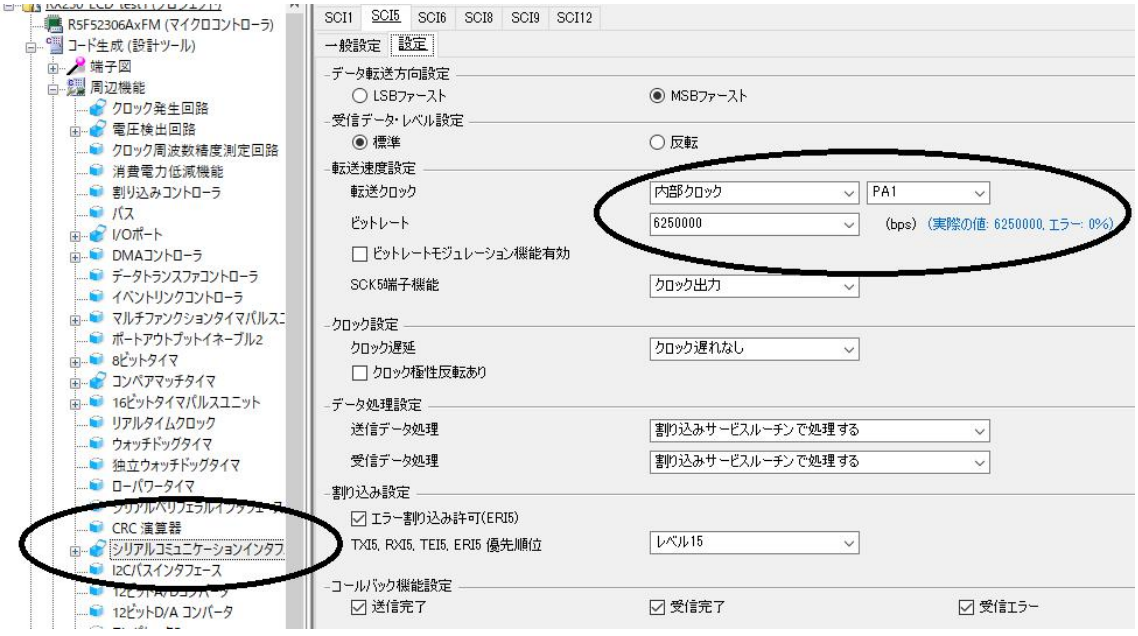
コンペアマッチタイマーで 1msec（1000μsec)の定周期割り込みを動作させます。
int_wait()関数はこの時間を計測しているので、極めて正確（クリスタル精度）です。
AD 変換やタッチキースキャンにも使用出来ます。



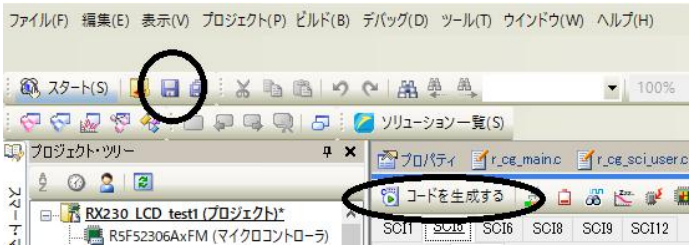
RX230 と ILI9341（液晶コントローラ）の接続は SPI で行われています。SCI5 を簡易 SPI バスモードに設定。



転送クロックを最大の 6.25MHｚにします。



設定をセーブし、コード生成をします。

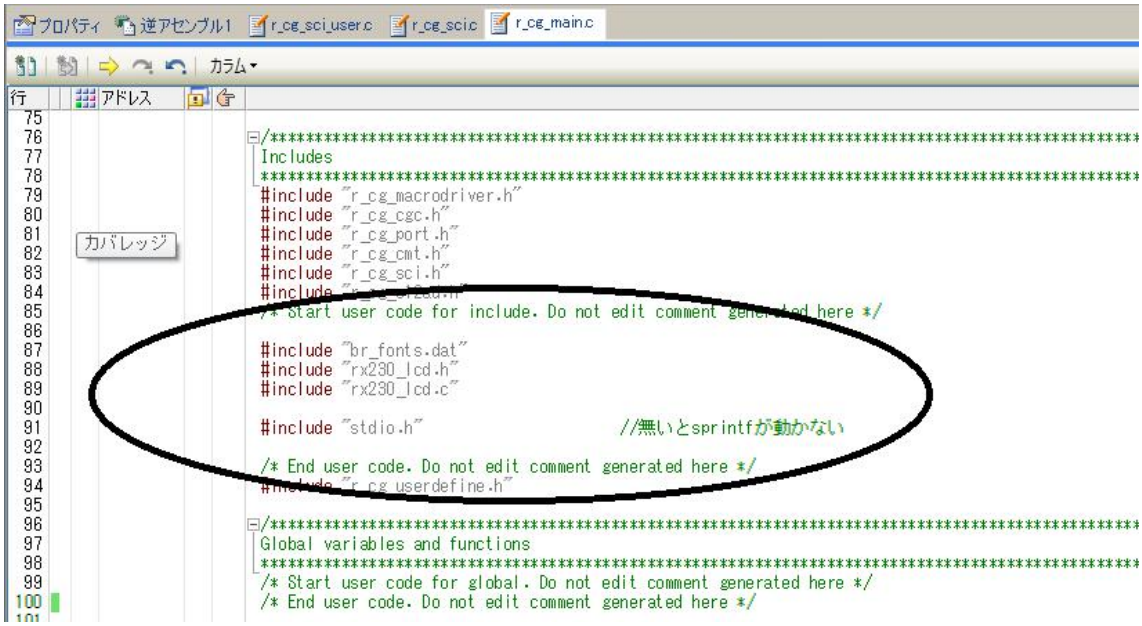


各種ファイルが作成されます。グラフィックライブラリの３つのファイルをホルダにもってきて、ｒ_ｃｇ_ｍａｉｎ．ｃファイルに以下の＃ｉｎｃｌｕｄｅを追加してください。

```
#include "br_fonts.dat"          //フォントファイル
#include "rx230_lcd.h"           //ヘッダファイル
#include "rx230_lcd.c"           //グラフィックライブラリ本体
```

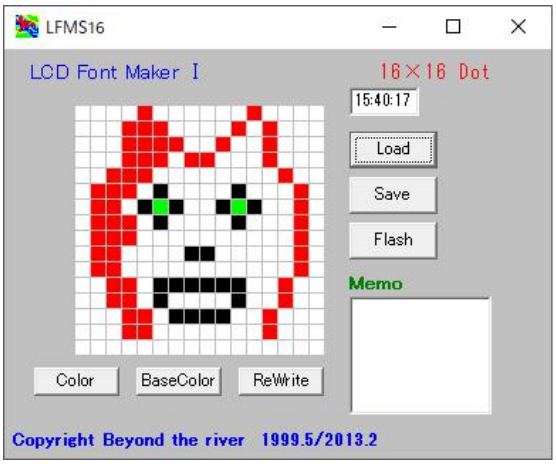
sprintf を使う場合、以下も追加する。

```
#include "stdio.h"               //無いと sprintf が動かない
```



以上、ゼロからプロジェクトを作る場合の説明ですが、前に書いたようにコピペで名前だけ変える方が簡単で落ちが無いと思います。

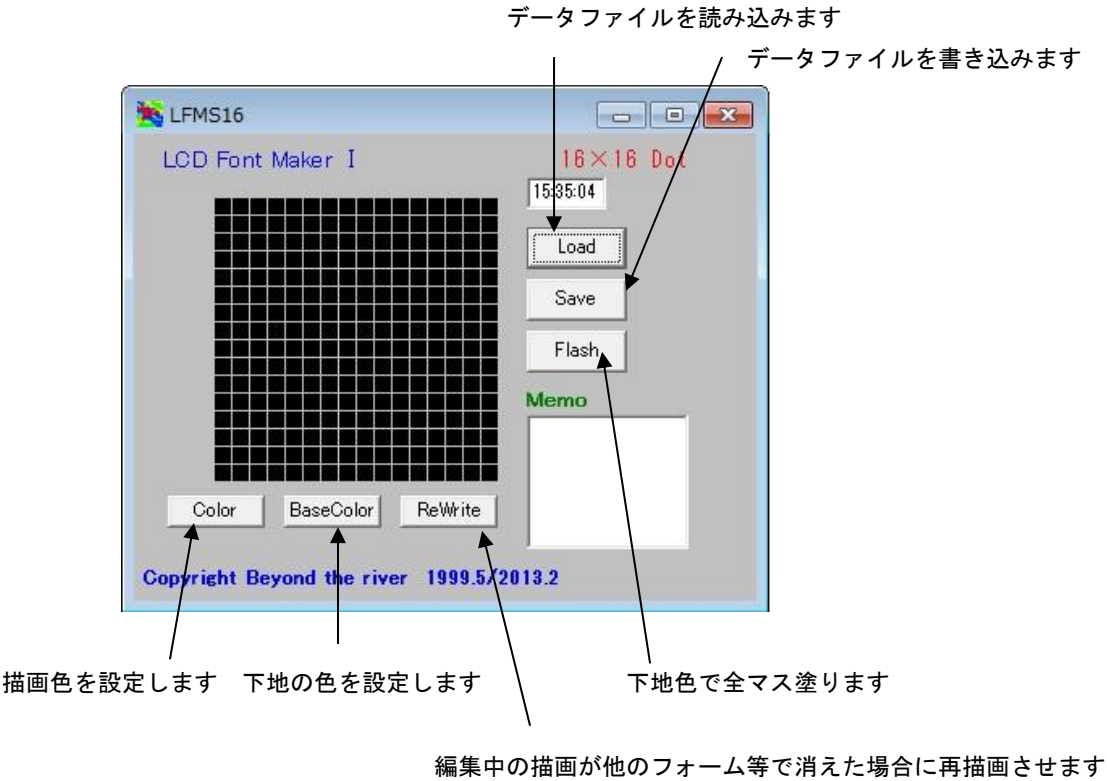
●フォントデータの作り方



○フォントメーカーの使い方

「フォントを作る」ホルダに１６×１６ピクセル用フォントメーカーｌｆｍｓ１６．ｅｘｅと３２×３２ピクセル用フォントメーカーｌｆｍｓ３２が用意されています。これらフォントメーカーは以降で説明する漢字、数字、絵などを液晶画面に表示するためにＣ言語で扱える形のデータを製作するソフトウエアです。

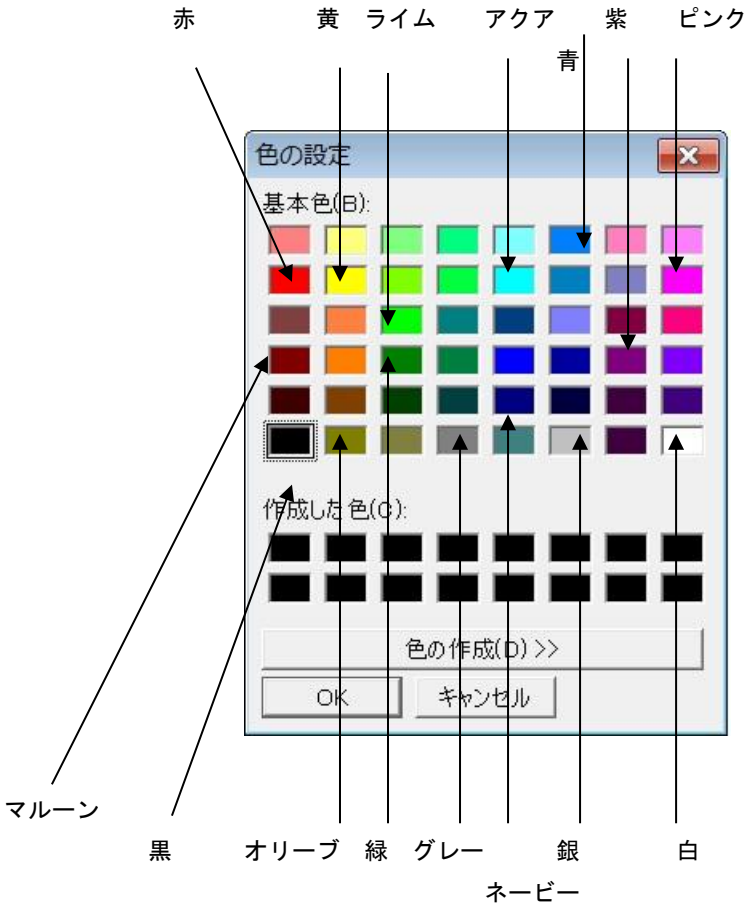
ｌｆｍｓ１６．ｅｘｅで使い方を説明します。ｌｆｍｓ３２も使い方は同じです。

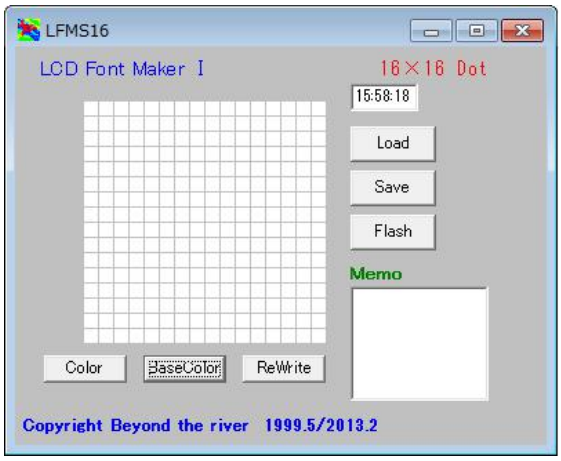


新規にデータを製作する場合、初めに「ＢａｓｅＣｏｌｏｒ」をクリックし、色指定します。このプログラムは１６色の色を指定できます。それ以外の色指定を行うと正常に表示されませんので、ご注意下さい。

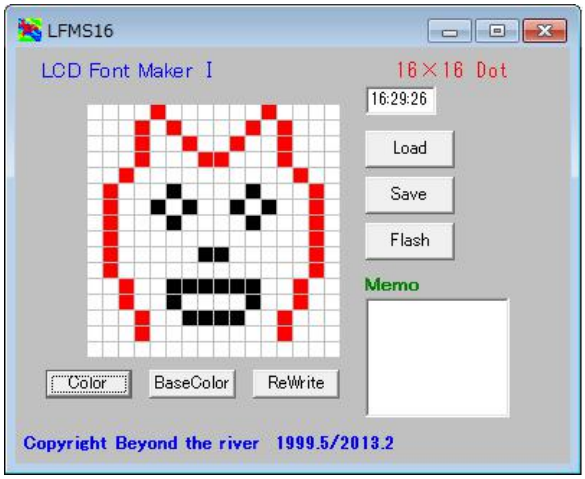
■設定できる１６色

それ以外を選択すると液晶表示が白色になります。

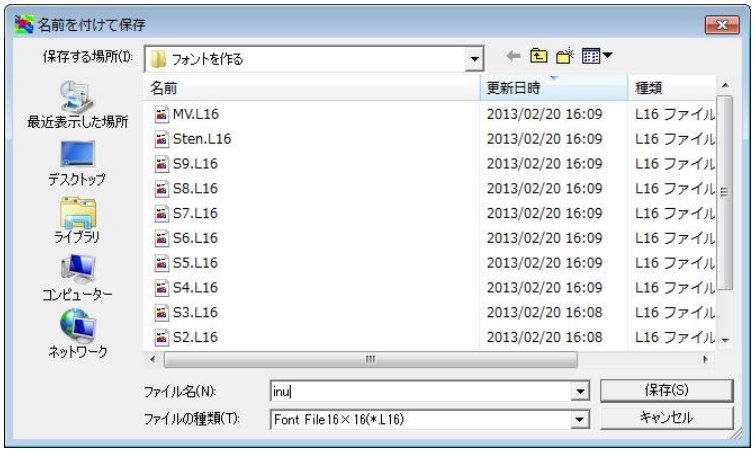




例えば白を選択し、ＯＫをクリックすると１６×１６ピクセルが全て白になります。
次に、描画色を決めます。「Ｃｏｌｏｒ」をクリックし、ＢａｓｅＣｏｌｏｒ同様、設定して下さい。
マウスで変えたい色をクリックします。



描画してＯＫであればＳａｖｅをクリック。拡張子はＬ１６です。付けなくて「保存」できます。
（Ｉｆｍｓ３２はＬ３２）

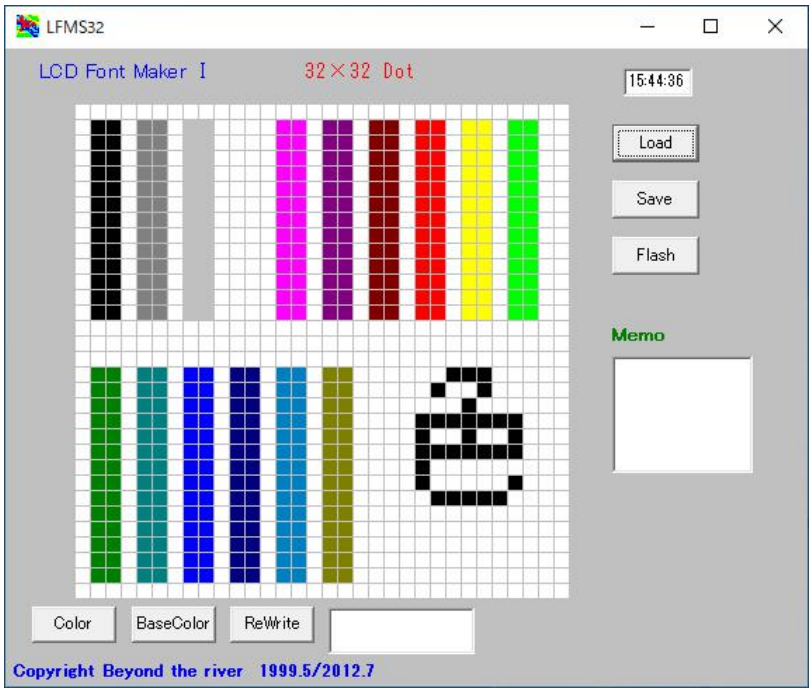


画像データを読み込む場合はＬｏａｄをクリックしてください。

ここでフォントメーカーは２種類のファイルをセーブしました。拡張子Ｌ１６はデータ専用のバイナリーファイル、Ｃ１６という拡張子のファイルはエディタで開くためのアスキーファイルです。

ｉｎｕ．ｃ１６ファイルをメモ帳で開いてみます。ｃｈａｒ型で先ほど製作した情報がセーブされています。このファイルは１６×１６ピクセルの位置と色情報が入っています。このデータをそのままｂｒ＿ｆｏｎｔｓ＿ｄａｔに追加コピペしてください。

下記例は色の見栄えをテストするサンプル　ｔｅｓｔ＿ｃｏｌｏｒです。



20250606

製品のお問い合わせは以下にお願いします。

〒350-1213　埼玉県日高市高萩1141-1
有限会社ビーリバーエレクトロニクス
http://www.beriver.co.jp/
info@beriver.co.jp
TEL:042-985-6982

20250606