

## 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-----

#### ■動作環境

- セットでご購入時の梱包物
- ハードウェア マイコン、液晶板製品外観
- 寸法図
- 回路図概要
- ソフトウェア CDの詳細
- CS+ for CCの設定
- ライブラリの使い方
- フォントデータの使い方、組み込みマイコンの考え方
- フォントデータをRAMに配置するかROMに配置するか

#### ■ライブラリ関数

##### ■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

- 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

- 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、

- 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。

##### ■備考

- CS+ for CC 初めからプログラムを作る設定
- フォントデータの作り方

-----

## ●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 2.5mm スペーサー

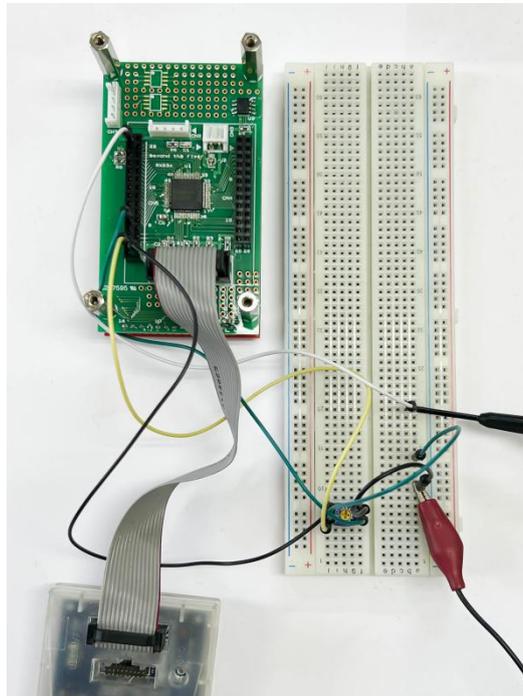
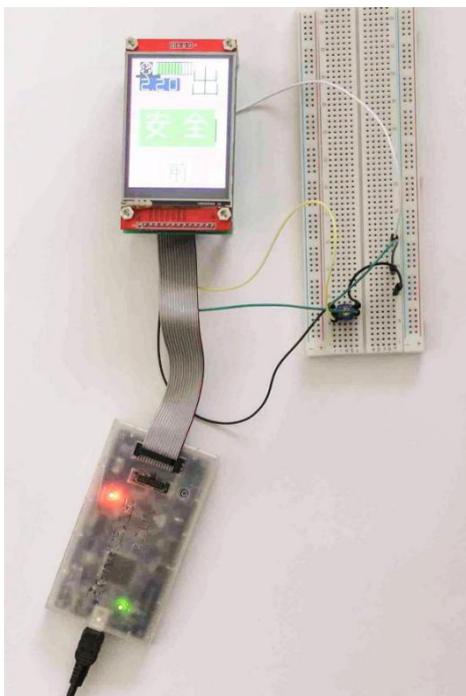
E2lite やコネクタからブレッドボードへの線がストレスなく接続できます。

本製品開発には E1, E2lite 等のエミュレータが必要です。

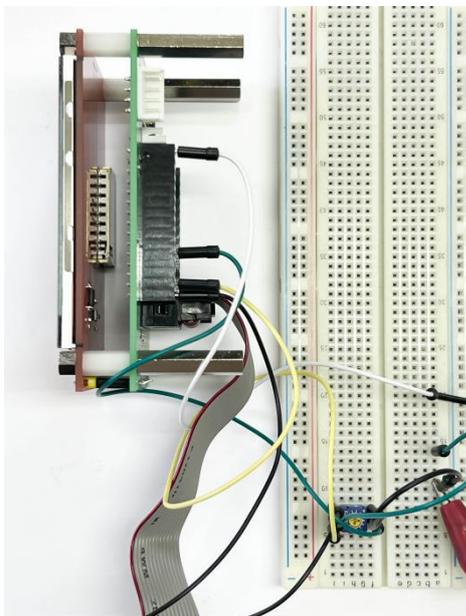
サンプルソフトは E2lite 用に書かれていますが、容易に E1 でも動作するように変更可能です。お問い合わせください。

●ハードウェア 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。



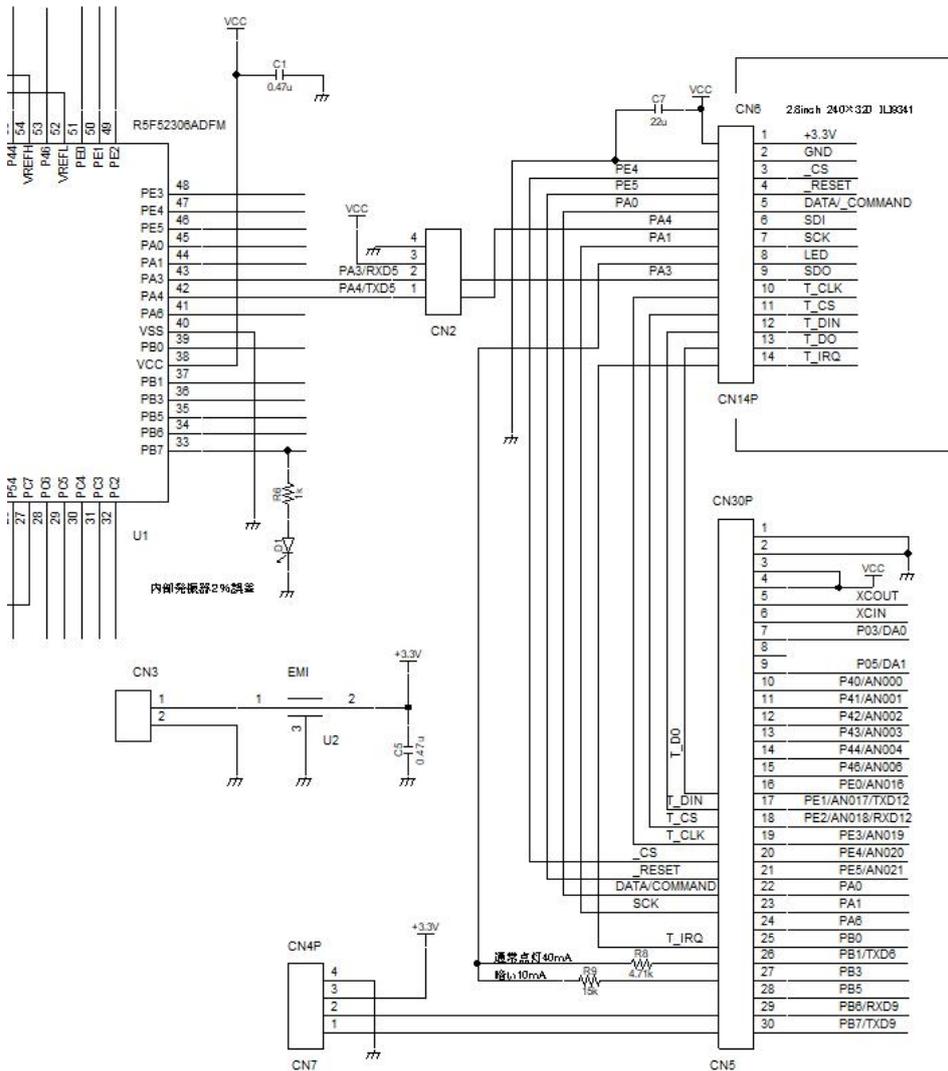
液晶板とマイコン基板は 1.1mm の間隔でスペーサーで止められています。



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<sub>xx</sub>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。



## ●ソフトウェア CDの詳細

CDには以下のホルダがあります。

ドキュメント > RX230\_240×320液晶制御 > RX230\_240\_320LCD\_CD出荷用データ >

名前	状態	更新日時
Fonts_20250606	⊙	2025/06/10 17:11
サンプルソフト	⊙	2025/06/10 17:10
ドキュメント	⊙	2025/06/10 17:01
フォントを作る	⊙	2025/06/10 16:53
ライブラリ_20250606	⊙	2025/06/10 17:23

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。

C D出荷用データ > RX230\_240\_320LCD\_CD出荷用データ > Fonts >

名前	状態	更新日時
アルファベット小文字 16×16	↻	2025/06/12 14:22
アルファベット大文字 16×16	↻	2025/06/12 14:22
アルファベット大文字 32×32	↻	2025/06/12 14:22
ひらがな 16×16	↻	2025/06/12 14:22
フォントを作る	↻	2025/06/12 14:22
バタ	↻	2025/06/12 14:22
メーター	↻	2025/06/12 14:22
レベルメーター	↻	2025/06/12 14:22
漢字_危険注意安全	↻	2025/06/12 14:22
漢字出現1-51	↻	2025/06/12 14:22
漢字出現52-102	↻	2025/06/12 14:22
数字16×16	↻	2025/06/12 14:22
数字32×32	↻	2025/06/12 14:22

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。

C D出荷用データ > RX230\_240\_320LCD\_CD出荷用データ > サンプルソフト >

名前	状態	更新日時
RX230_LCD_AD1	⊙	2025/06/11 17:31
RX230_LCD_AD2	⊙	2025/06/11 17:31
RX230_LCD_test1	⊙	2025/06/11 17:31
RX230_LCD_安全	⊙	2025/06/11 17:31

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。

› C D出荷用データ › RX230\_240\_320LCD\_CD出荷用データ › ドキュメント

名前	状態	更新日時
 RX230_240×320LCD_回路図.pdf	⊙	2025/06/10 16:59
 RX230_240×320LCD取扱説明書.pdf	⊙	2025/06/10 17:01
 グラフィックライブラリRX230_LCDの使い方2025...	⊙	2025/06/10 16:55

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。

C D出荷用データ › RX230\_240\_320LCD\_CD出荷用データ › フォントを作る

名前	状態	更新日時
 lfms32.exe	⊙	2013/02/26 15:57
 lfms16.exe	⊙	2013/02/26 15:54
 力.L32	⊙	2013/02/21 14:44
 田.L32	⊙	2013/02/20 11:57
 前.L32	⊙	2013/02/18 17:42
 出.L32	⊙	2013/02/21 14:46
 後.L32	⊙	2013/02/18 17:43
 yellow_panel.L32	⊙	2013/02/21 14:15
 test_color_full.L32	⊙	2013/02/20 18:54
 test_color.L32	⊙	2013/02/20 11:31

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。

C D出荷用データ › RX230\_240\_320LCD\_CD出荷用データ › ライブラリ\_20250606

名前	状態	更新日時
 br_fonts.dat	⊙	2025/05/30 16:22
 rx230_lcd.c	⊙	2025/06/06 14:34
 rx230_lcd.h	⊙	2025/06/06 14:34

## ●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、必要な時に参考に見てください。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。

 RX2301_LCD_test1 - コピー		2025/05/16 14:24	ファイルフォルダー
 RX2301_LCD_test1		2025/05/15 15:12	ファイルフォルダー

2. ホルダの名前を付けます。例ではRX230\_LCD\_AD1に変更。

 RX2301_LCD_AD1		2025/05/16 14:24	ファイルフォルダー
 RX2301_LCD_test1		2025/05/15 15:12	ファイルフォルダー

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。

 rx230_lcd.c		2025/05/15 15:12	Cファイル	21 KB
 rx230_lcd.h		2025/05/14 10:35	Hファイル	3 KB
 RX230_LCD_AD1.mtpj		2025/05/02 17:57	MTPJファイル	8,807 KB

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できます。

## ●ライブラリの使い方

ライブラリは

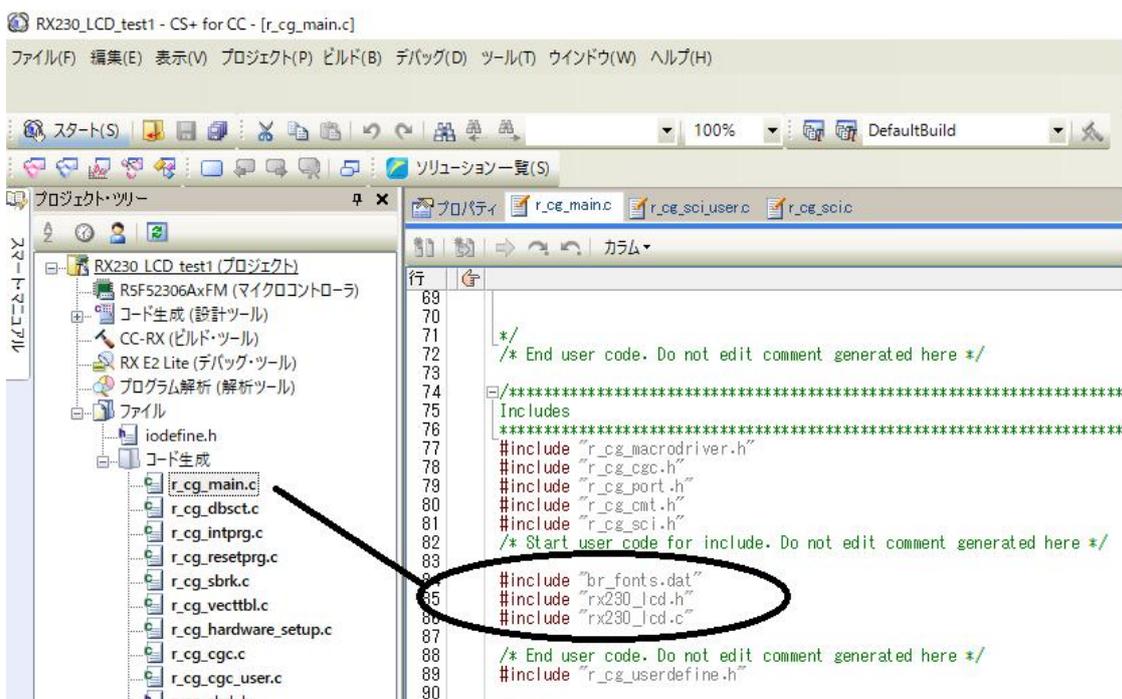
```
br_fonts.dat    フォントデータファイル
rx230_lcd.h     ライブラリ本体 ヘッダファイル
rx230_lcd.c     ライブラリ本体
```

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

```
123 | void main(void)
124 | {
125 |     R_MAIN_UserInit();
126 |     /* Start user code. Do not edit comment generated here */
127 |
128 |
129 |     //液晶初期化
130 |
131 |
132 |     init_lcd();
133 |
134 |     //バックライト 明るい
135 |     LCD_DARK_ON; //LCD バックライトON ダーク 夜用
136 |     LCD_BRIT_E_ON; //LCD バックライトON 明るい 昼用
137 |
138 | }
```

```
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できます。

## ●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      0xffffffff      //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};

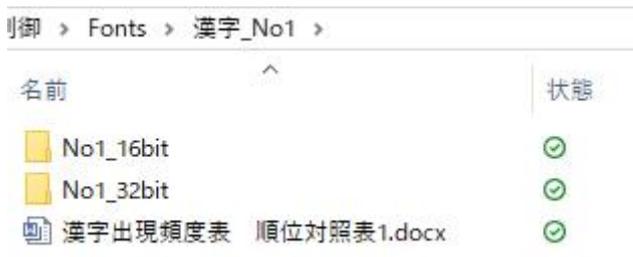
```

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。本件のハードウェ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	

制御 > Fonts > 漢字\_No1 > No1\_16bit

名前	状態
争.L16	✓
時.C16	✓
時.L16	✓
自.C16	✓
自.L16	✓
者.C16	✓
者.L16	✓
手.C16	✓
手.L16	✓
十.C16	✓
十.L16	✓
出.C16	✓
出.L16	✓
女.C16	✓
女.L16	✓
小.C16	✓
小.L16	✓
上.C16	✓
上.L16	✓
場.C16	✓
場.L16	✓
人.C16	✓
人.L16	✓
生.C16	✓
生.L16	✓
前.C16	✓
前.L16	✓
代.C16	✓
代.L16	✓

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形式です。

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,
3,3,3,3,3,3,3,0,3,0,3,3,3,3,3,3,3,
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,
3,0,0,3,3,3,3,3,3,3,3,0,3,3,3,
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。

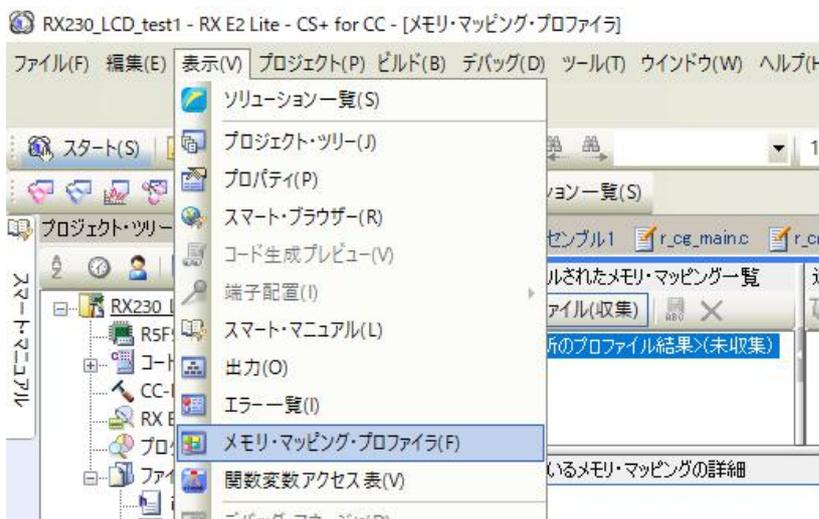
## ●フォントデータをRAMに配置するかROMに配置するか

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。

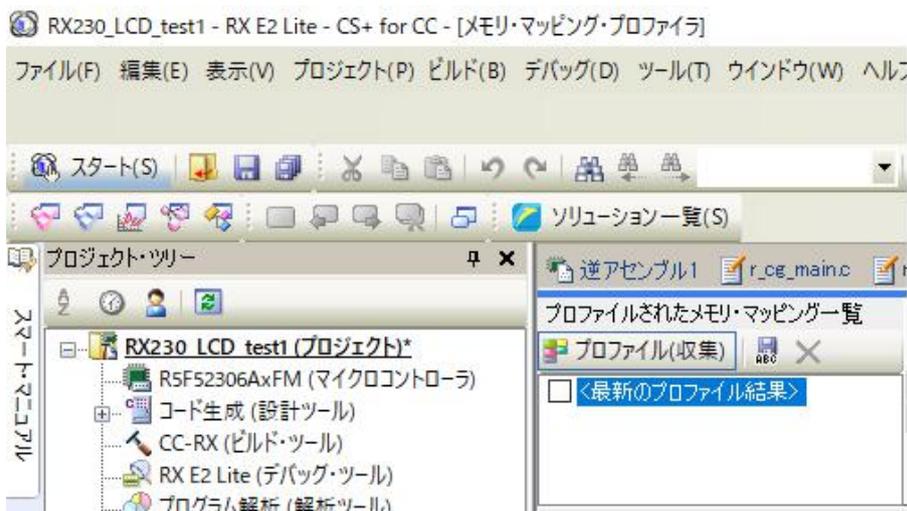
フォントデータの頭に const を書かないとデータはRAM上に展開され、書くとROMに展開されます。以下例ではROM上に配置されます。

```
//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。

選択しているメモリ・マッピングの詳細

種類	マロサイズ [Bytes]	マロサイズ [%]
ROM	68019	25.95
RAM	4470	13.64

サマリ / ロード・モジュール / セクション / オブジェクト・ファイル / シンボル

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。

選択しているメモリ・マッピングの詳細

シンボル名	マロサイズ [Bytes]	グループ	マロ開始アドレス	マロ終了アドレス	マロ種類	マロスコープ	マロセクション名	マロオブジェクト・ファイル
_rd	4	Data	0x0000038	0x000003B	Data	Global	R	DefaultBuild#r_cg_ma
_brk	4	Data	0x000003C	0x000003F	Data	Local	R	DefaultBuild#r_cg_sb
_sinf_coeff	16	Data	0x0000040	0x000004F	Data	Local	R	sincosfsub
_cosf_coeff	16	Data	0x0000050	0x000005F	Data	Local	R	sincosfsub
_pi4ptr_table	12	Data	0x0000060	0x000006B	Data	Local	R	sincosfsub
_sub_table	16	Data	0x000006C	0x000007B	Data	Local	R	sincosfsub
_PowerON_Reset_PC	62	Program	0xFFFF000	0xFFFF03D	Entry	Global	PRestPRG	DefaultBuild#r_cg_re
_MV	256	Constant	0xFFFF0100	0xFFFF01FF	Data	Global	C_1	DefaultBuild#r_cg_ma
S0	256	Constant	0xFFFF0200	0xFFFF02FF	Data	Global	C_1	DefaultBuild#r_cg_ma
S1	256	Constant	0xFFFF0300	0xFFFF03FF	Data	Global	C_1	DefaultBuild#r_cg_ma
S2	256	Constant	0xFFFF0400	0xFFFF04FF	Data	Global	C_1	DefaultBuild#r_cg_ma
S3	256	Constant	0xFFFF0500	0xFFFF05FF	Data	Global	C_1	DefaultBuild#r_cg_ma
S4	256	Constant	0xFFFF0600	0xFFFF06FF	Data	Global	C_1	DefaultBuild#r_cg_ma
S5	256	Constant	0xFFFF0700	0xFFFF07FF	Data	Global	C_1	DefaultBuild#r_cg_ma
S6	256	Constant	0xFFFF0800	0xFFFF08FF	Data	Global	C_1	DefaultBuild#r_cg_ma
S7	256	Constant	0xFFFF0900	0xFFFF09FF	Data	Global	C_1	DefaultBuild#r_cg_ma
S8	256	Constant	0xFFFF0A00	0xFFFF0AFF	Data	Global	C_1	DefaultBuild#r_cg_ma
S9	256	Constant	0xFFFF0B00	0xFFFF0BFF	Data	Global	C_1	DefaultBuild#r_cg_ma
_Sten	256	Constant	0xFFFF0C00	0xFFFF0CFF	Data	Global	C_1	DefaultBuild#r_cg_ma
_test2	256	Constant	0xFFFF0D00	0xFFFF0DFF	Data	Global	C_1	DefaultBuild#r_cg_ma
_test3	256	Constant	0xFFFF0E00	0xFFFF0EFF	Data	Global	C_1	DefaultBuild#r_cg_ma
_inu	256	Constant	0xFFFF0F00	0xFFFF0FFF	Data	Global	C_1	DefaultBuild#r_cg_ma
_kanji_ie	256	Constant	0xFFFF1000	0xFFFF10FF	Data	Global	C_1	DefaultBuild#r_cg_ma
_kanji_ka	256	Constant	0xFFFF1100	0xFFFF11FF	Data	Global	C_1	DefaultBuild#r_cg_ma
_kanji_gaku	256	Constant	0xFFFF1200	0xFFFF12FF	Data	Global	C_1	DefaultBuild#r_cg_ma
_kanji_aida	256	Constant	0xFFFF1300	0xFFFF13FF	Data	Global	C_1	DefaultBuild#r_cg_ma
_kanji_ji	256	Constant	0xFFFF1400	0xFFFF14FF	Data	Global	C_1	DefaultBuild#r_cg_ma
_S0w	1024	Constant	0xFFFF1500	0xFFFF18FF	Data	Global	C_1	DefaultBuild#r_cg_ma
S1w	1024	Constant	0xFFFF1900	0xFFFF1CFF	Data	Global	C_1	DefaultBuild#r_cg_ma

サマリ / ロード・モジュール / セクション / オブジェクト・ファイル / シンボル

## ■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_BRITTE_ON //LCD バックライト 明 オン (昼用)
#define LCD_BRITTE_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 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 radius 半径 color 外周の色
        //描画に~100msec かかります

void Dsisplay_ON(void) //液晶表示 ON
void Dsisplay_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 学問家字 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、color2 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トの使い方は■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 //ベタ塗り

    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,fsz_32,loop5*455); ⑫ //レベルメーター 140msec
            meter_yellow(10,150,fsz_64,loop5*455); ⑬ //メーター黄色
            meter_red(10,10,fsz_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 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)

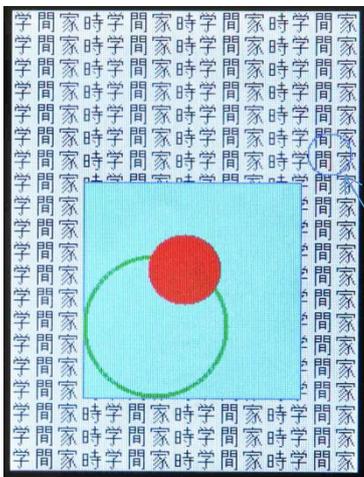
```

```

    {
        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

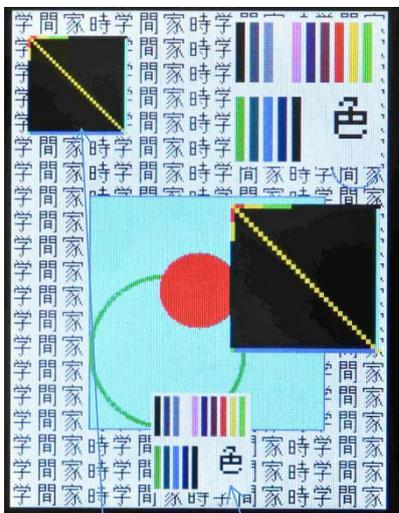
```
gcircle(120,140,25,1,clRed);         //mode 1 内部ベタ塗 丸
```

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
```

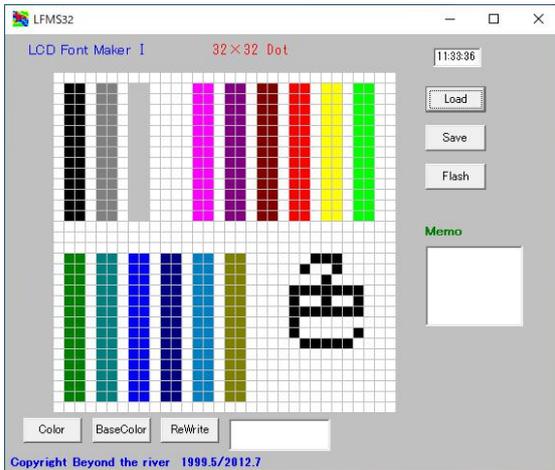
起点が X:220、Y:220、半径 15、描画 mode:通常、青色描画 見えにくいですがブルーの円です。

test\_zukei2(); ⑥ //図形 2 表示

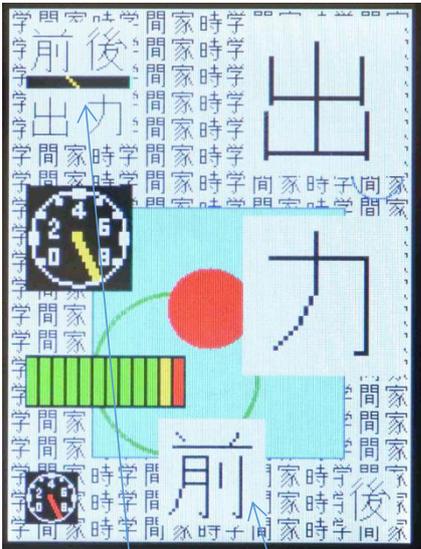


```
void test_zukei2(void) //
{
    gwrite32_64(90,10,test_color); //32×32 16色データを64×64(2倍)に拡大して表示
    gwrite32_96(140,100,test); //32×32 データを96×96(3倍)に拡大して表示
    gwrite32_96(140,220,test_color); //32×32 16色データを96×96(3倍)に拡大して表示
    gwrite32_64(10,240,test); //32×32 データを64×64(2倍)に拡大して表示
}
```

`gwrite32_64()` 関数は  $32 \times 32$  ドットデータを 2 倍の  $64 \times 64$  に表示する関数です。  
`gwrite32_96()` 関数は 3 倍の  $96 \times 96$  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
```

```
{
```

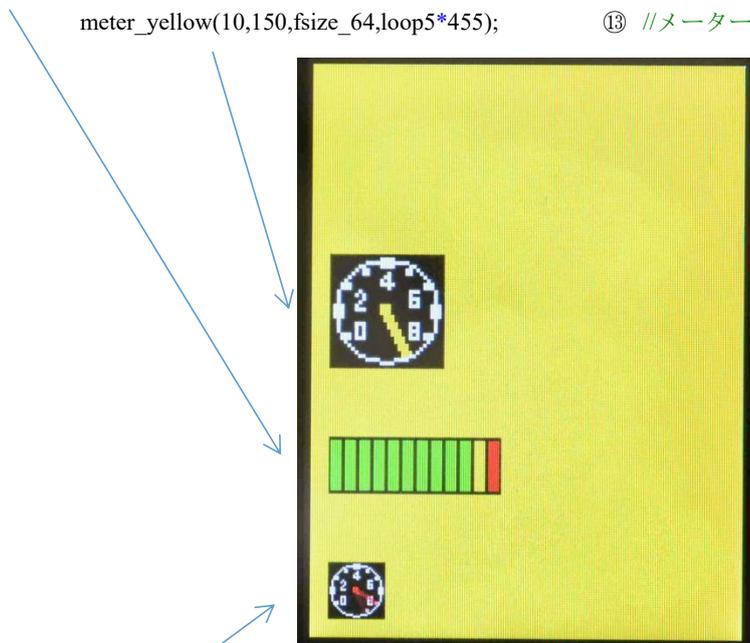
loop5は0から9まで変化します。

```
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,fszize_32,loop5*455); ⑫ //レベルメーター 140msec
```

```
meter_yellow(10,150,fszize_64,loop5*455); ⑬ //メーター黄色
```



```
meter_red(10,10,fszize_32,loop5*455); ⑭ //メーター赤色
```

```
if(loop5 == 9) ⑮
```

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

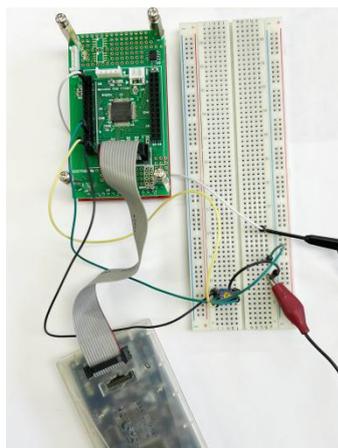
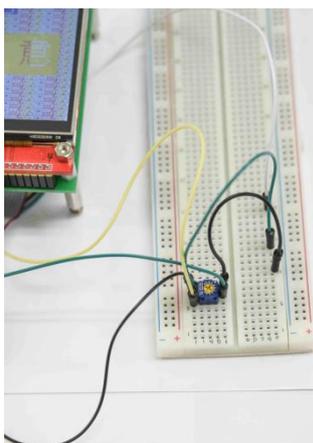
loop 5が9の時だけ、見た目の区切りのために 500msec のウェイトを入れて止めています。このウェイト関数は 1msec 定周期割り込みで作成されていて、極めて高精度です。

## ●デモンストレーション2 RXD230\_LCD\_AD1

A/D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

A/D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できます。

●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\r\n",fdata1);    ⑦

    if(ad0 < lm_yellow_level)    ⑧
    {
        lcd_puts_color(20,100,fszize_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,fszize_64,ad_buff,clYellow,clWhite); //黄色文字
        }
        else
        {
            lcd_puts_color(20,100,fszize_64,ad_buff,clRed,clWhite); //赤文字
        }
    }

    meter_yellow(10,180,fszize_64,ad0);    ⑩           //meter 46msec
    level_meter(10,250,fszize_64,ad0);    ⑪           //level_meter 140msec
    Vbar_chart(30,222,5,fszize_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
```

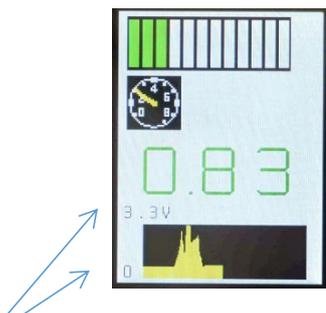
①コード生成機能で自動的に作られる AD コンバータ初期設定関数です。

```
// 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
```

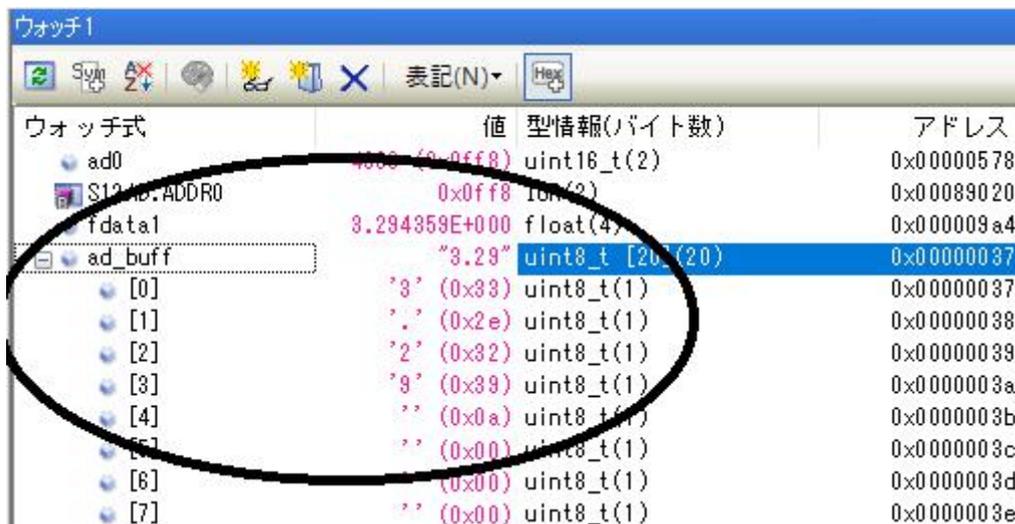
⑤データが揃ったら、変数 a d 0 に入れます。

```
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\r\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を越えて、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 $3597/4095 \times 3.3V \approx 2.9V$ 、赤が $3924/4095 \times 3.3V = 3.16V$ 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。a d 0 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。a d 0 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  $64 \times 3 = 192$  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
}

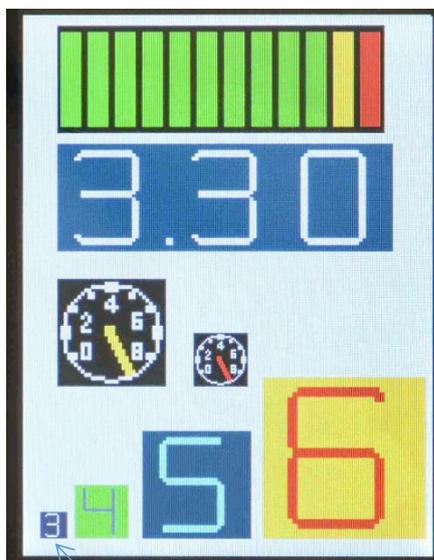
```

### ●デモンストレーション3 RXD230\_LCD\_AD2

A/D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](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
```

① 16 × 16 ドット、数字の3を白色、背景グレーで X:10, Y:10 の位置に表示。

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

② 32 × 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
```

③ 32 × 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, Y:10 の位置に表示

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

④ 32 × 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, Y:10 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 y o u t u b e U R L [https://www.youtube.com/watch?v=D5WdA6a\\_UUE](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,fszize_32,ad_buff,clWhite,clNavy);//xposi yposi mode(16,32,64,96) data[]
    level_meter(60,280,fszize_32,cnt*341);    ③ //level_meter 140msec
    meter_yellow(20,280,fszize_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。

`printf(ad_buff,"%1.2f\n",cnt/3.63);` ②

② `cnt = 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,fszize_32,ad_buff,clWhite,clNavy); //xposi yposi mode(16,32,64,96) data[]`  
`level_meter(60,280,fszize_32,cnt*341);` ③ //level\_meter 140msec

③レベルメータを表示するために `cnt` に341を掛けています。  $12 \times 341 = 4092 \div 4095$

`meter_yellow(20,280,fszize_32,cnt*341);` ④ //meter 46msec

④黄色メータを表示するために `cnt` に341を掛けてます。  $12 \times 341 = 4092 \div 4095$

`if(cnt < 11)` ⑤

```

{
    gwrite32_96_color(20,100,an,clWhite,clLime); //安
}

```

```

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

```

⑤ `cnt` 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();

        }
    }
}

```

⑥ `cnt` 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);

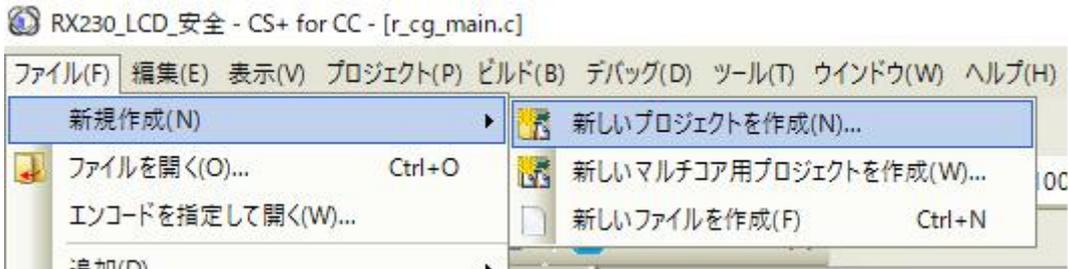
}

```

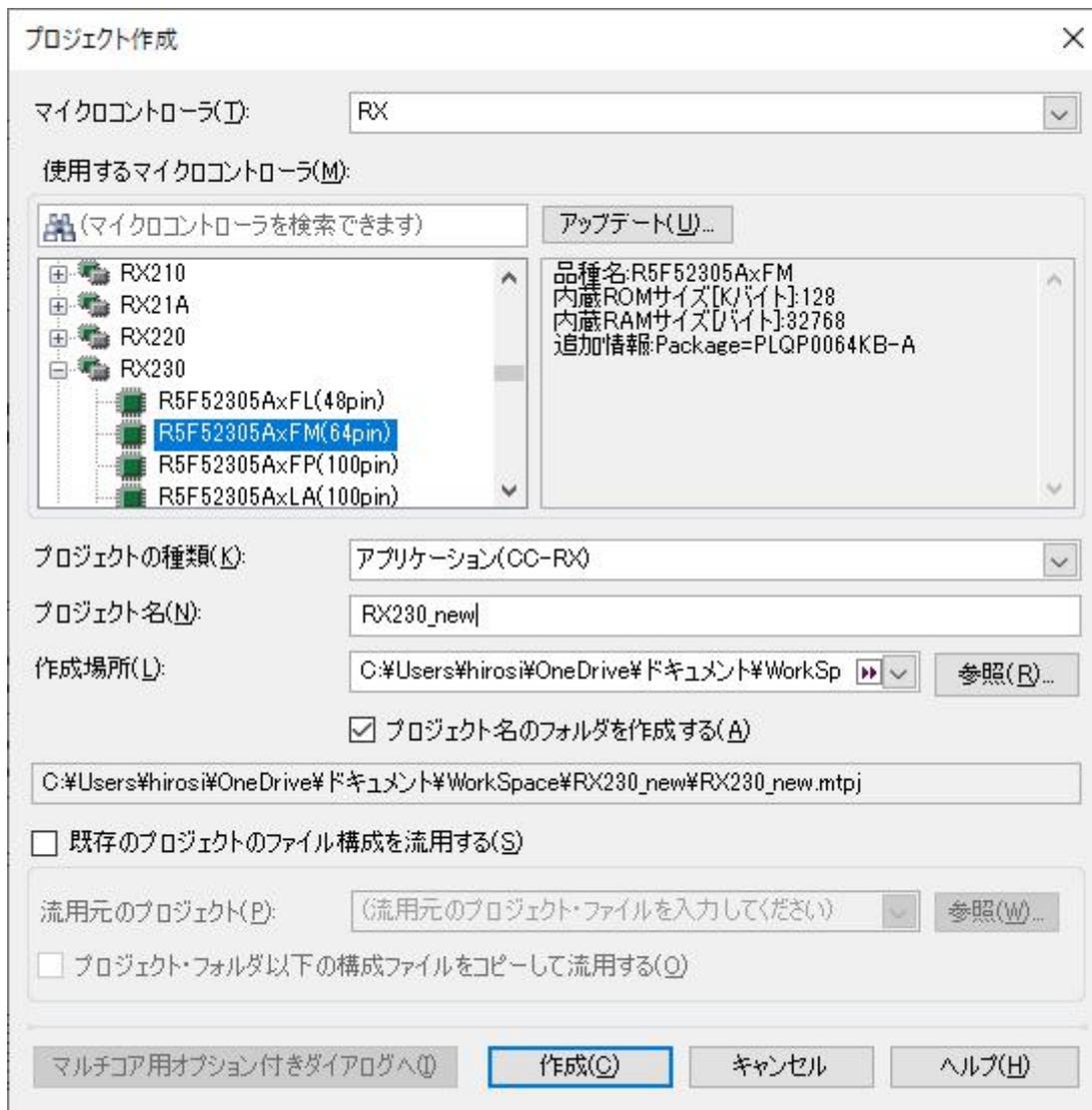
⑦ 12 で「危険」と表示します。白文字、背景赤、96×96 ドットサイズで 2 秒間表示します。

■備考

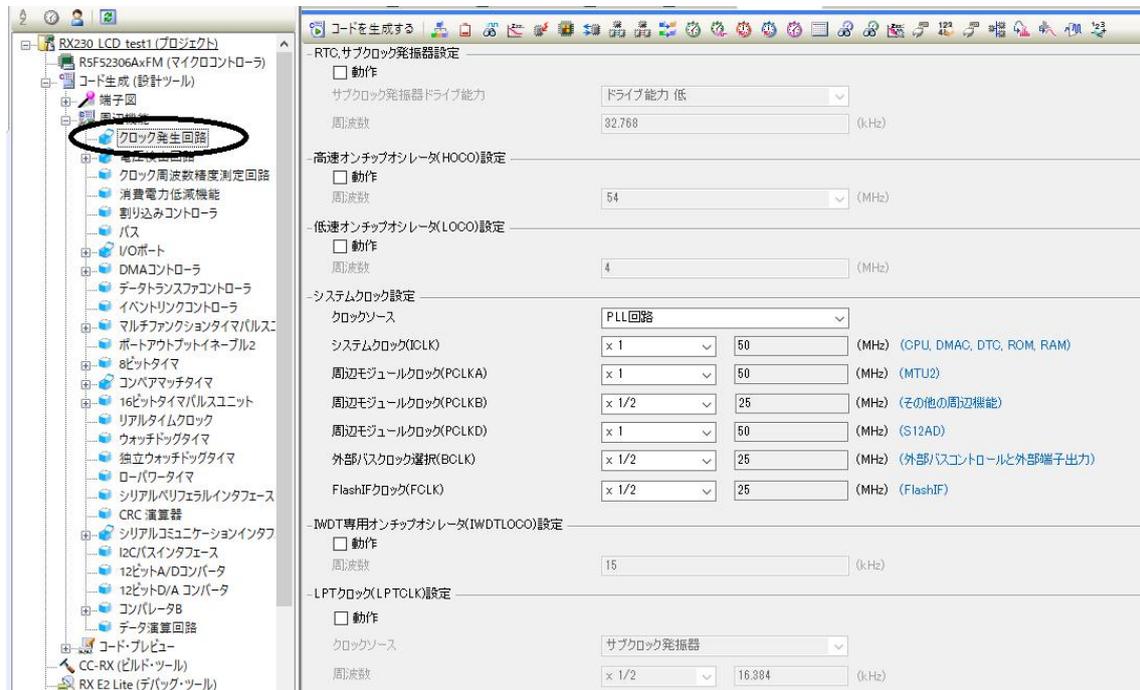
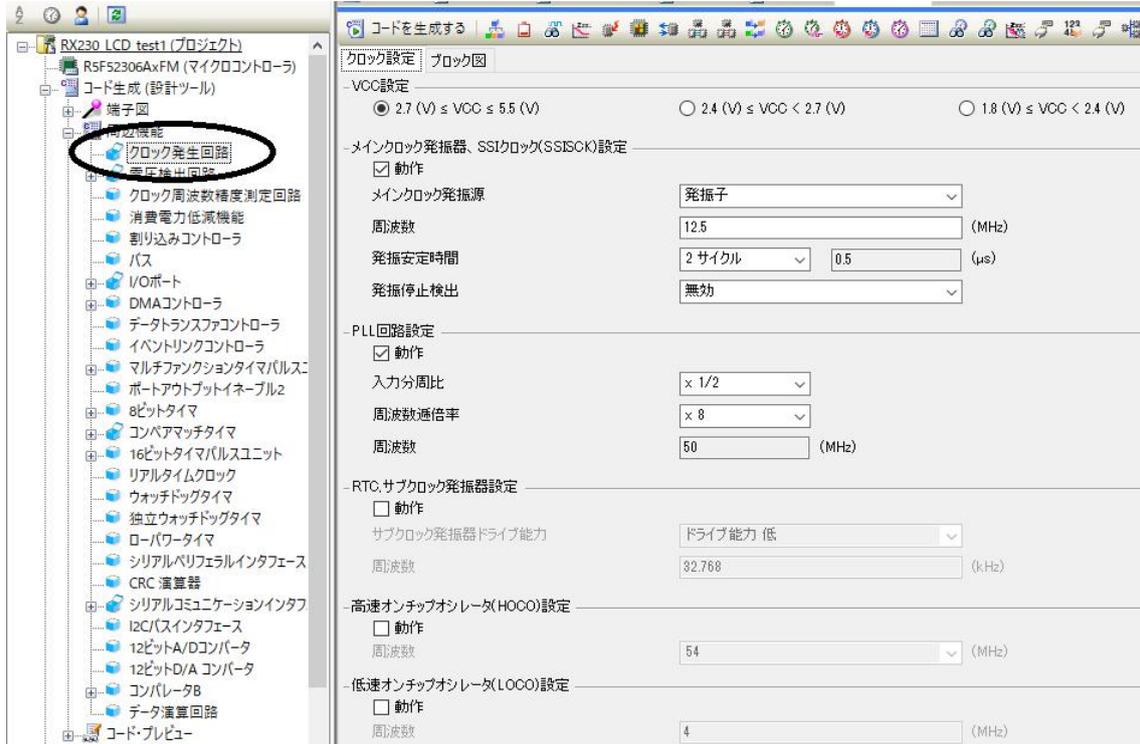
●CS+ for CC 初めからプログラムを作る設定



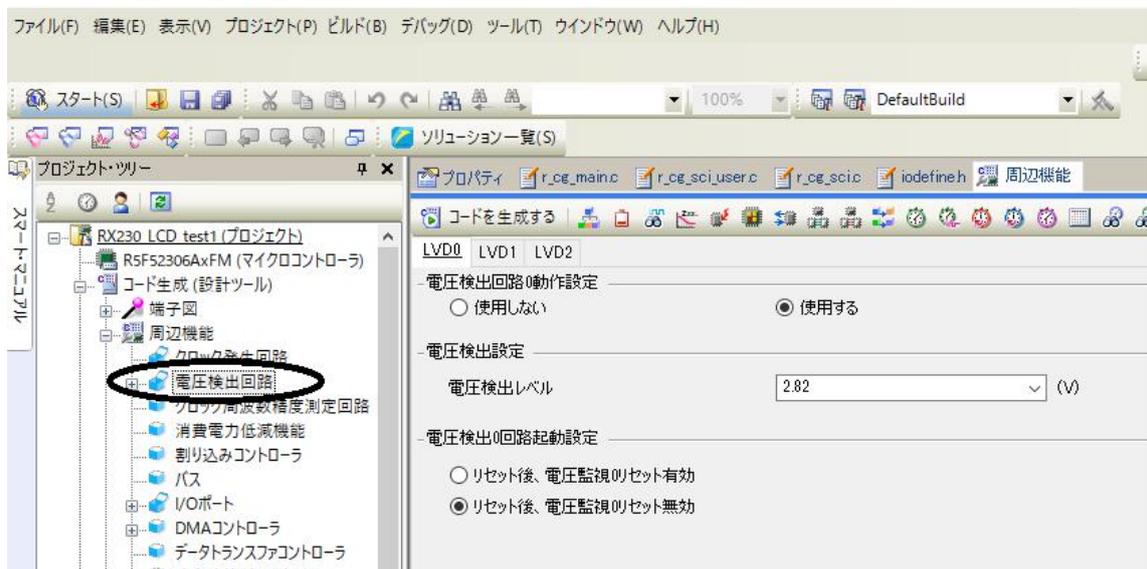
64pin の R5F52305AxF を選択します。ファイル名はここでは RX230\_new にしました。



クロックは外部の 12.5 MHz 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。





## 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。



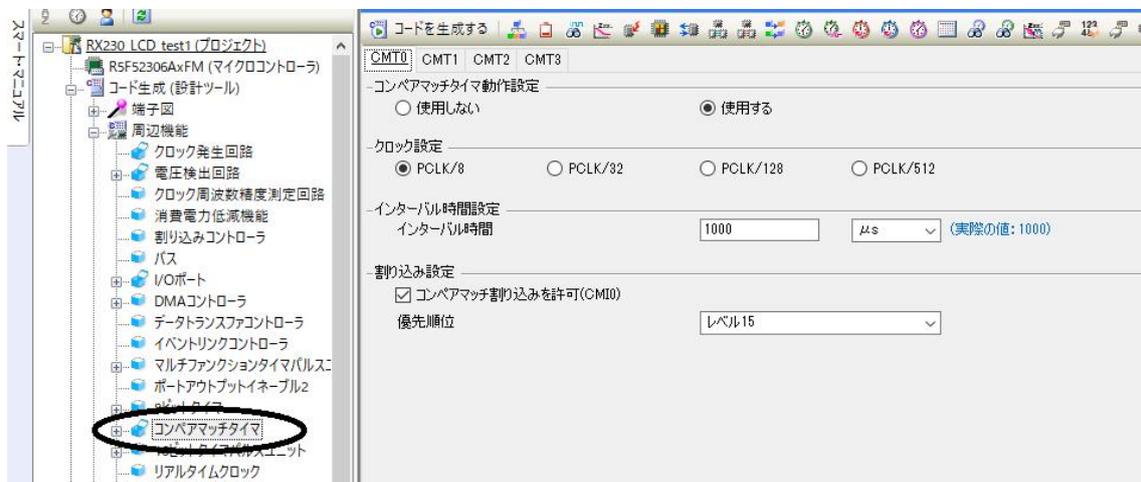
PB ポートは PB0 がタッチスイッチの T\_IRQ、PB1 が液晶バックライト：通常点灯（昼用）、PB3 が夜間点灯（夜用）を制御します。PB7 は基板に実装済の LD1 に抵抗を介して接続されていて、各種インジケート、関数時間計測に使用します。



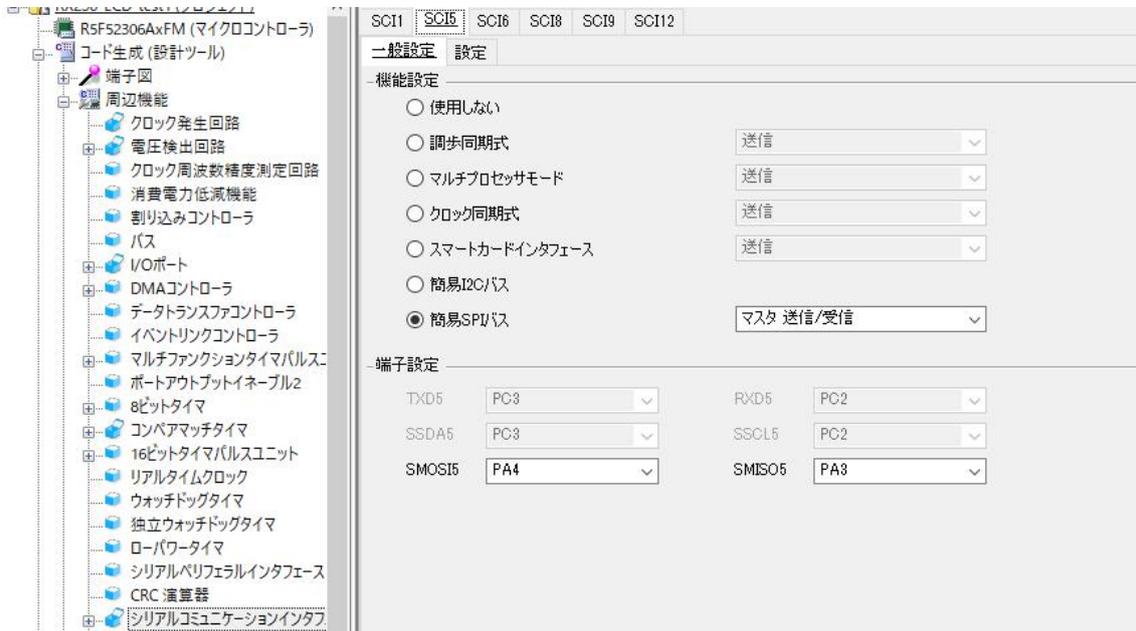
PE ポートはタッチスイッチインターフェイス、液晶の \_CS や \_RESET に使用します。



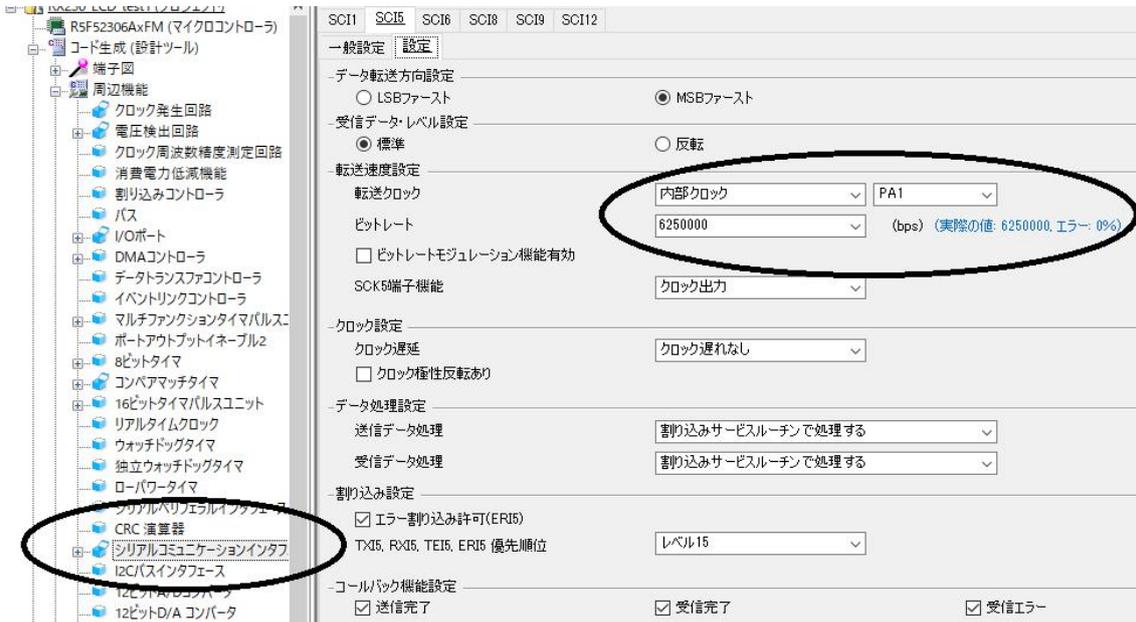
コンペアマッチタイマーで 1msec (1000  $\mu$ sec) の定周期割り込みを動作させます。  
int\_wait() 関数はこの時間を計測しているので、極めて正確 (クリスタル精度) です。  
AD 変換やタッチキースキャンにも使用出来ます。



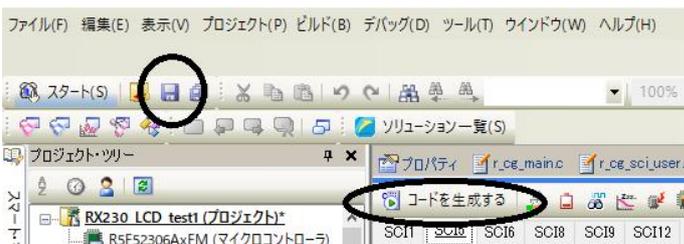
RX230 と ILI9341（液晶コントローラ）の接続は SPI で行われています。SCI5 を簡易 SPI モードに設定。



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。

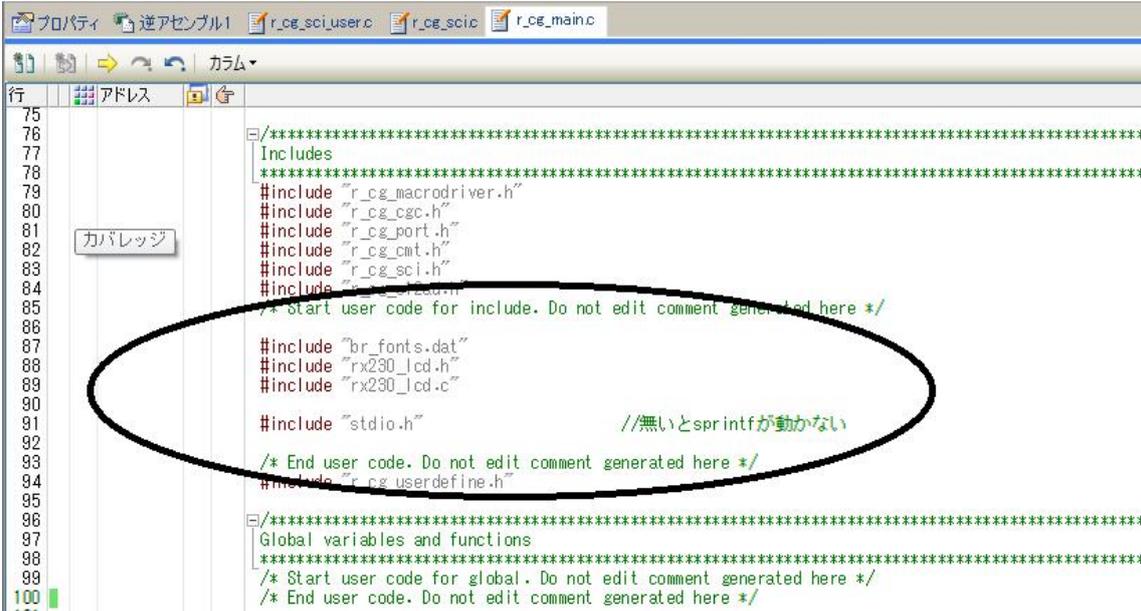


各種ファイルが作成されます。グラフィックライブラリの3つのファイルをホルダにもってきて、`r_cg_main.c`ファイルに以下の`#include`を追加してください。

```
#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
```



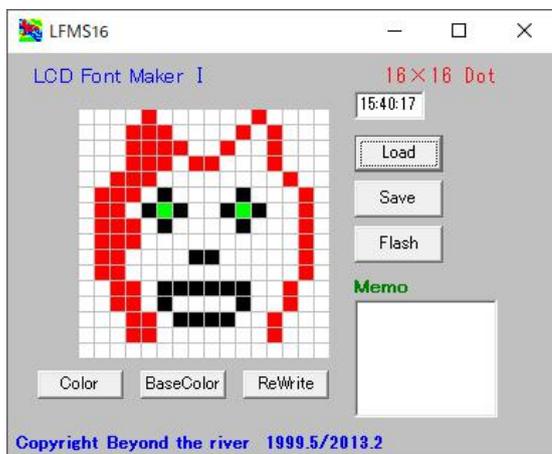
The screenshot shows a code editor window with the file `r_cg_main.c` open. The code is as follows:

```
75
76
77 #include "br_fonts.dat" //フォントファイル
78 #include "rx230_lcd.h" //ヘッダファイル
79 #include "rx230_lcd.c" //グラフィックライブラリ本体
80
81 #include "stdio.h" //無いと sprintf が動かない
82
83
84
85 /* Start user code for include. Do not edit comment generated here */
86
87 #include "br_fonts.dat"
88 #include "rx230_lcd.h"
89 #include "rx230_lcd.c"
90
91 #include "stdio.h" //無いと sprintf が動かない
92
93 /* End user code. Do not edit comment generated here */
94 #include "r_cg_userdefine.h"
95
96
97 Global variables and functions
98
99 /* Start user code for global. Do not edit comment generated here */
100 /* End user code. Do not edit comment generated here */
```

A black oval highlights the lines 87-91, which contain the added include statements. A mouse cursor is visible over the text "カバレッジ" on line 81.

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。

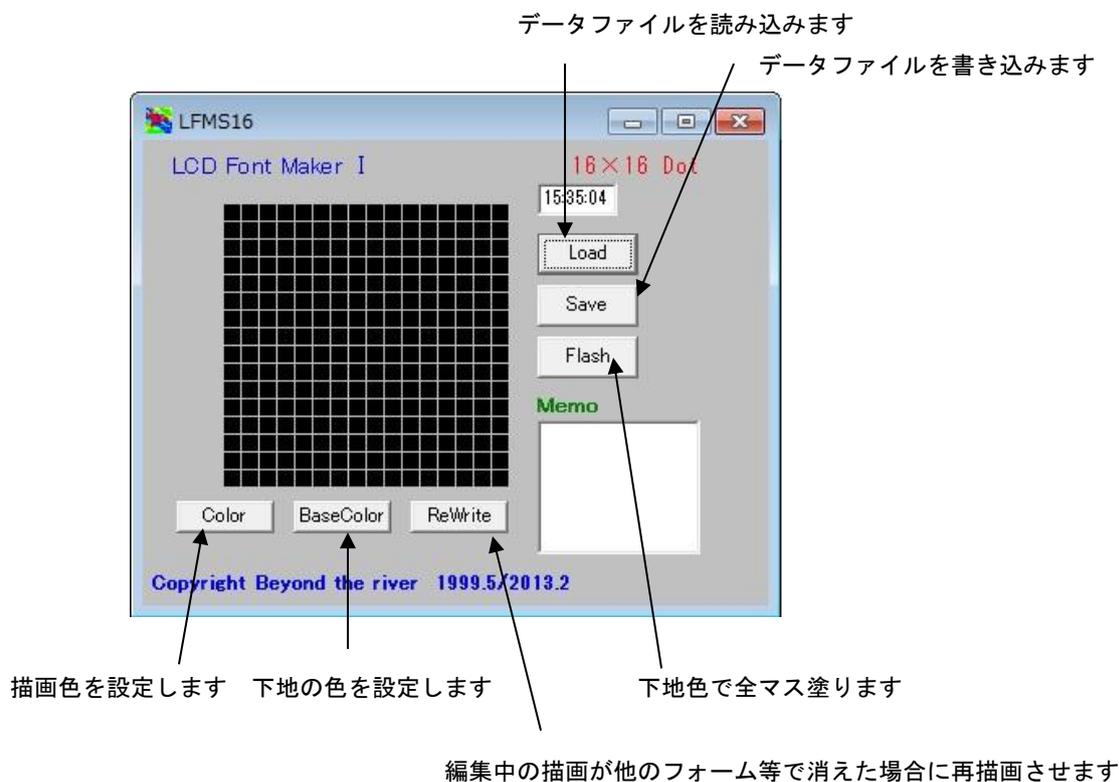
## ● フォントデータの作り方



## ○ 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ェアです。

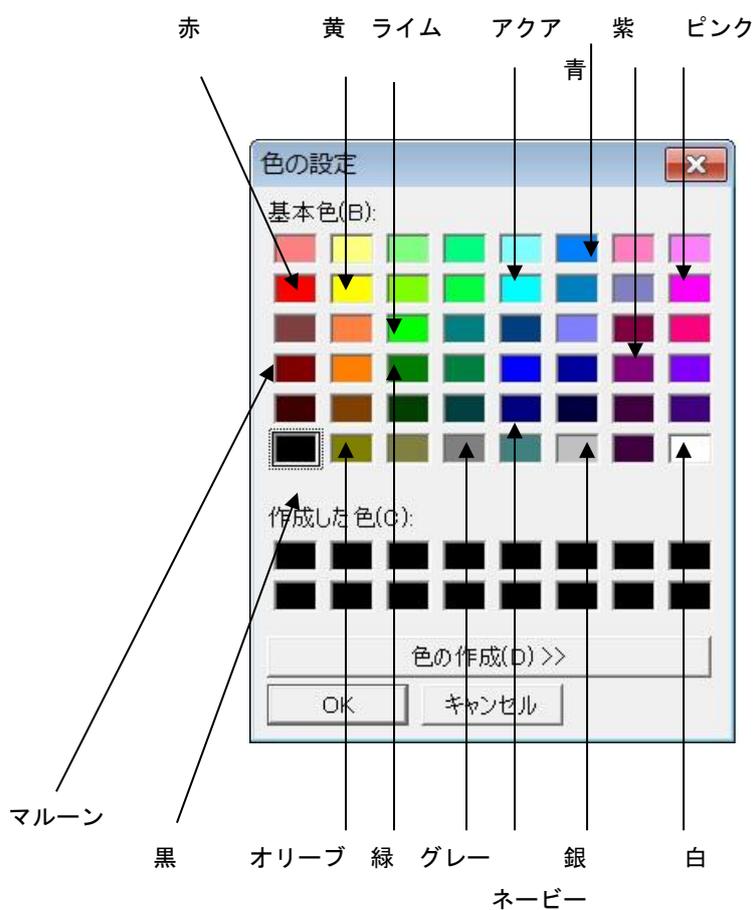
lfms16.exeで使い方を説明します。lfms32も使い方は同じです。

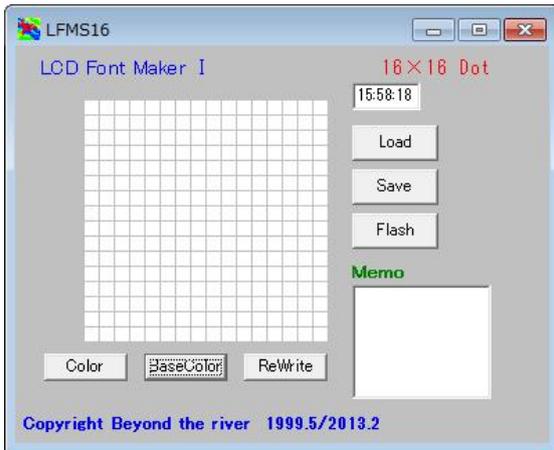


新規にデータを製作する場合、初めに「BaseColor」をクリックし、色指定します。このプログラムは16色の色を指定できます。それ以外の色指定を行うと正常に表示されませんので、ご注意下さい。

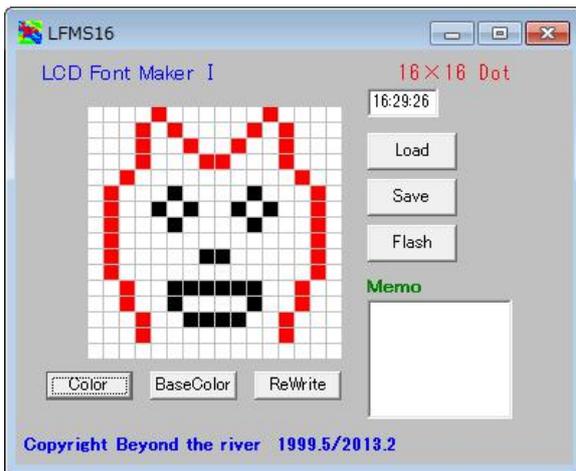
### ■設定できる16色

それ以外を選択すると液晶表示が白色になります。



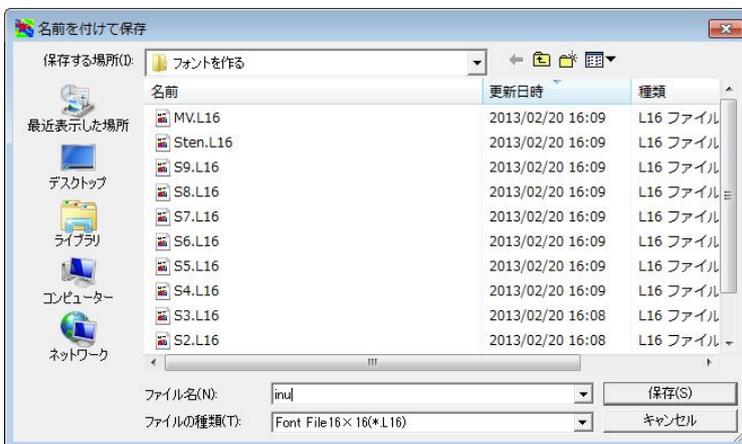


例えば白を選択し、OKをクリックすると16×16ピクセルが全て白になります。  
次に、描画色を決めます。「Color」をクリックし、BaseColor同様、設定して下さい。  
マウスで変えたい色をクリックします。



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。

(lfms32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。

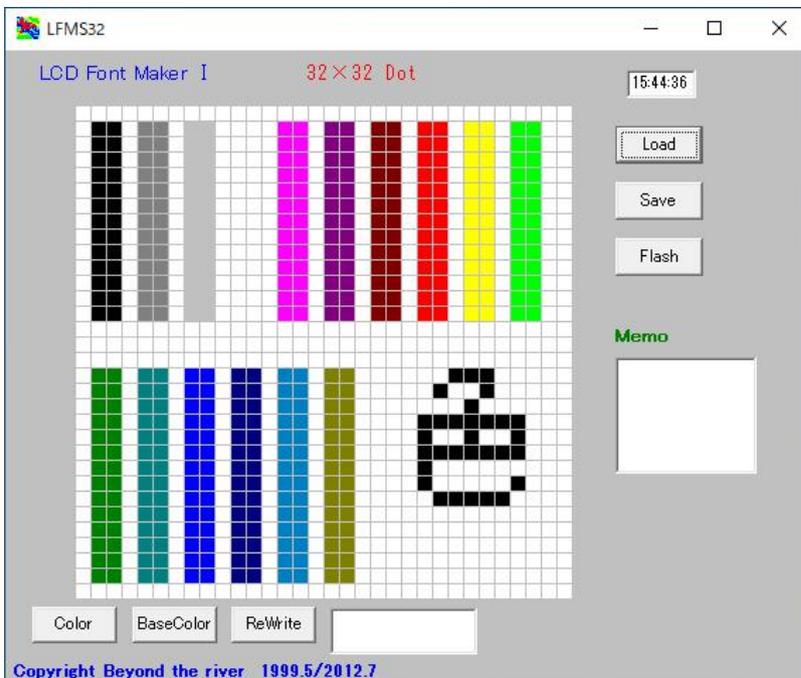
```

inu.C16 - メモ帳
ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)
unsigned char inu[]={
3,3,3,3,7,3,3,3,3,3,7,3,3,3,
3,3,3,7,3,7,3,3,3,3,7,3,7,3,3,3,
3,3,3,7,3,3,7,3,3,7,3,3,7,3,3,3,
3,3,3,7,3,3,3,7,7,3,3,3,7,3,3,3,
3,3,7,3,3,3,3,3,3,3,3,3,3,7,3,3,
3,7,3,3,3,0,3,3,3,3,0,3,3,3,7,3,
3,7,3,3,0,3,0,3,3,0,3,0,3,3,7,3,
3,7,3,3,0,3,3,3,3,0,3,3,3,7,3,
3,7,3,3,3,3,3,3,3,3,3,3,3,7,3,
3,7,3,3,3,3,0,0,3,3,3,3,3,7,3,
3,7,3,3,3,3,3,3,3,3,3,3,3,7,3,
3,3,7,3,3,0,0,0,0,0,0,3,3,7,3,3,
3,3,7,3,3,0,3,3,3,0,3,3,7,3,3,
3,3,3,7,3,3,0,0,0,0,3,3,7,3,3,3,
3,3,3,7,3,3,3,3,3,3,3,7,3,3,3,
3,3,3,3,3,3,3,3,3,3,3,3,3,3};

```

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に追加コピーしてください。

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

C o p y r i g h t   b e y o n d   t h e   r i v e r   I n c .

20250606