# RX ファミリ

## emWin ライブラリとシリアル LCD を使用した GUI サンプルプログラム

### 要旨

本アプリケーションノートでは、Segger 社のグラフィックライブラリ emWin とシリアルインタフェースの TFT-LCD を使用した GUI アプリケーションの開発方法について説明します。

また、本サンプルプログラムは emWin が提供する基本的な機能（ウィンドウマネージャ、ウィジェット、メモリデバイス、アニメーションなど）を用いて作成しており、これらの機能についても解説しています。

なお、本サンプルプログラムは GUI デザインツールの AppWizard は使用しておりません。

注意

1. 本サンプルプログラムを動作させるためには別途シリアルインタフェース(SPI)に対応した TFT-LCD と配線の部品が必要です。
2. 本サンプルプログラムはリンクサイズが 128K バイトを超えるため、無償評価版の試用期間が終了している状態ではビルドを完了することが出来ません。コンパイラ製品をご購入していただいている場合には、同梱されているライセンスキーをライセンスマネージャで登録してください。

### 動作確認デバイス

RX671 グループ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様に合わせて変更し、十分評価してください。

# 目次

# 1. 概要

本サンプルプログラムでは、以下の画面のデモを用意しています。LCD はタッチパネル入力に対応しており、表示されたボタンをタッチしてサンプルプログラムを制御します。また、使用ボードに実装されているタッチキーでデモ画面の遷移が可能です。

- ホーム画面
- RX ロゴ表示
- 空調制御
- 画像表示
- フォント表示

## 1.1 ホーム画面

本サンプルプログラム実行時に表示される最初の画面です。他 4 つの画面に遷移することができます。使用している emWin の主な機能はウィンドウマネージャとダイアログ、ボタンウィジェットです。

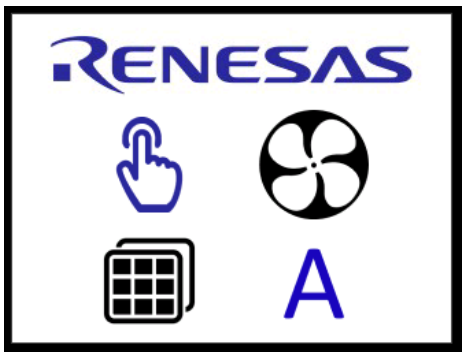



図 1.1　ホーム画面

## 1.2 RX ロゴ表示

画面中央に表示された RX ロゴをドラッグ操作によって画面内を自由に動かすことができます。ドラッグ操作を止めると RX ロゴが画面中央に戻ります。使用している emWin の主な機能はウィンドウマネージャとボタンウィジェット、アニメーションです。

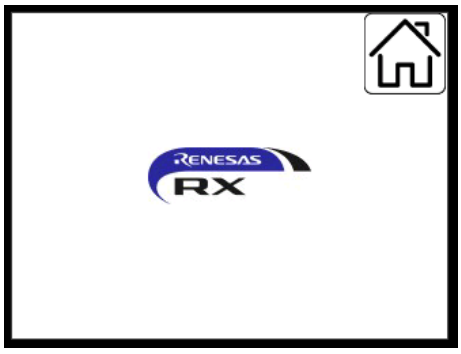



図 1.2　RX ロゴ表示

## 1.3　空調制御

温度、風量を操作するデモです。「Temp」または「Air Flow」のどちらかが選択された状態で左右の矢印をタッチすると、それぞれの設定値を変更できます。設定値に合わせて温度メータや風量メータの状態が変化します。使用している emWin の主な機能はウィンドウマネージャとダイアログ、ボタンウィジェット、タイマ、メモリデバイスです。

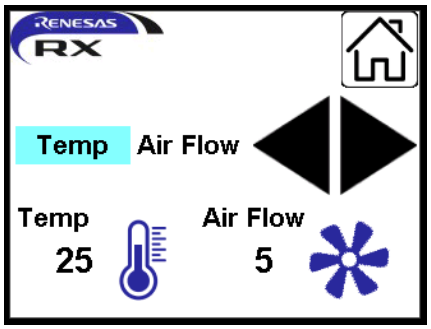



図 1.3　空調制御

## 1.4　画像表示

画像を全画面表示します。サムネイル画像をタッチすると、その画像に切り替わります。使用している emWin の主な機能はウィンドウマネージャとアイコンビューウィジェットです。

図 1.4　画像表示

## 1.5　フォント表示

emWin ライブラリ内に同梱されているいくつかのフォントを表示します。使用している emWin の主な機能はウィンドウマネージャとフォント表示です。


図 1.5　フォント表示

## 1.6　画面の遷移

使用ボードのタッチキーをタッチすると下記のように画面を遷移できます。

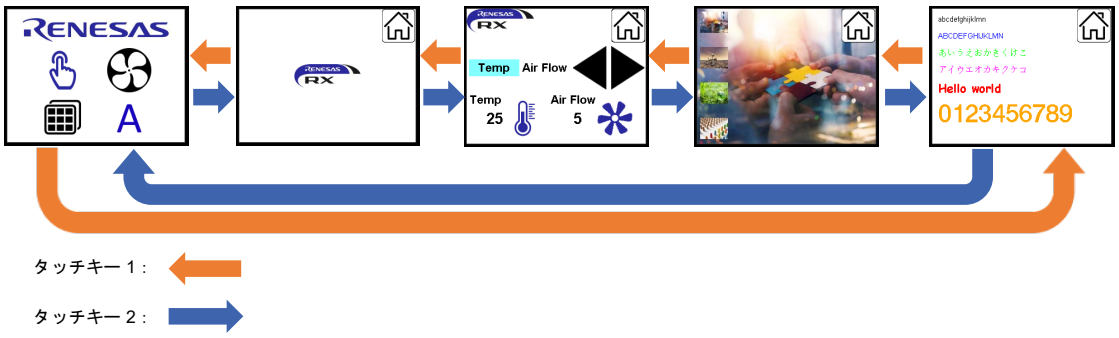



図 1.6　画面の遷移

# 2.　emWin の主な機能について

## 2.1　ディスプレイドライバ（Flex Color ドライバ）とカラーモード

emWin には様々な種類の LCD コントローラに対応したディスプレイドライバと様々な色を表現するためのカラーモードが用意されています。

本サンプルプログラムは、接続する LCD コントローラ（ILI9341）に対応した「Flex Color」と呼ばれるディスプレイドライバ、および「GUICC_565」と定義されている 16bit RGB のカラーモードを使用しています。

https://www.segger.com/doc/UM03001_emWin.html#GUIDRV_FlexColor

https://www.segger.com/doc/UM03001_emWin.html#Colors

## 2.2　メモリ管理

emWin は軽量なグラフィックライブラリですが、使用する画像やデザイン次第でメモリ使用量が大きく変動するため、メモリの管理が大変重要です。特に RAM はワーク領域としてキャッシュやメモリデバイスといった機能を使用するため、十分な検討が必要です。

emWin で使用するワーク領域は emWin FIT モジュールのコンフィグレーション項目“EMWIN_GUI_NUM_BYTES”で指定します。ユーザアプリケーションの開発の際には余裕を持たせて設定することを推奨します。

## 2.3　画面の更新速度について

シリアルインタフェースの LCD を使用する場合、画面のデザインや通信速度によって画面の更新速度に大きく影響します。特にウィンドウやウィジェットを多用したり、データの通信速度が遅かったりすると画面全体の更新パフォーマンスが低下します。その結果、ちらつきが発生したり、画面の更新が視覚的に見える場合があります。emWin では、それらを抑えるために以下の機能が使用できます。また、ハードウェアの観点から 7.画面の更新速度に関する補足も合わせてご参照ください。

### 2.3.1　キャッシュ

通常の画面の更新はウィンドウやウィジェットごとに実行されており、LCD にデータが何度も繰り返し送信されます。キャッシュを使用すると、画面の更新データをいったんキャッシュ内にすべて書き出し、その後にまとめて LCD に送信することで LCD へのアクセスを抑え、画面の更新速度を上げることができます。

ただし、キャッシュには 1 画面分のメモリが必要となるため、十分な RAM 容量が必要です。

**キャッシュに必要な RAM 容量：ディスプレイの幅(px)×高さ(px)×ピクセルあたりのバイト数(bpp)**

キャッシュを使用するには、emWin FIT モジュールのコンフィグレーション設定で“EMWIN_GUI_USE_CACHE”を“1”に設定し、“EMWIN_GUI_NUM_BYTES”にキャッシュに必要なサイズを加算します。

本サンプルプログラムではキャッシュを有効にしています。キャッシュに必要な RAM 容量は 240px×320px×2bpp = 150KB となり、RX671 の内蔵 RAM（384KB）内に収めることが可能です。

### 2.3.2　メモリデバイス

メモリデバイスは以下のような使い方のできるバッファです。

1. 外部デバイスから読み出した画像データのバッファ
2. 画像の回転や色、α値の変更、ぼかしやブレンドなどの操作で使うバッファ
3. 描画に時間がかかる画面のバッファ(キャッシュと同等の使い方)

メモリデバイスは可変的に確保できるバッファであり、いくつものメモリデバイスが同時に有効な場合、有効なメモリデバイスの数だけ RAM 容量が必要です。それらは emWin FIT モジュールのコンフィグレーション項目“EMWIN_GUI_NUM_BYTES”から確保されます。

また、メモリデバイスを生成する API 関数はいくつか種類がありますが、用途に応じて適切な API を選択する必要があります。例えば、回転処理を行うためには、GUI_MEMDEV_CreateFixed32 関数を使用する必要があります。

さらに、キャッシュを使用できるほどのメモリ容量がない製品は、代わりにメモリデバイスを使用することで画面の更新速度を上げることができます。画面を複数に分割して確保可能なサイズのメモリデバイスに書き込み、まとまったデータを送ることでキャッシュを使用しない場合よりもパフォーマンスは上がります。メモリデバイスを画面に適用するには 2.5.ウィンドウ を参照してください。

https://www.segger.com/doc/UM03001_emWin.html#Memory_Devices

本サンプルプログラムでは、1.3 空調制御 の画面でメモリデバイスを使用しています。

## 2.4　ウィンドウマネージャ

emWin のウィンドウマネージャは画面をウィンドウと呼ぶ単位で管理するための機能です。

画面のコンテンツを複数のパーツ（ウィンドウ）に分割して管理しており、互いのパーツを独立して描画することでパフォーマンスを向上させています。

作成したウィンドウは、作成時の処理(WM_CREATE)や描画処理(WM_PAINT)などのイベントに対応するコールバック関数を用意して動作を管理します。

https://www.segger.com/doc/UM03001_emWin.html#The_Window_Manager_WM

各ウィンドウの描画やタッチパネル入力などのイベントを実行させるためには、メインループ内で GUI_Exec 関数の実行が必要です。GUI_Exec 関数が実行されることで、各ウィンドウのコールバック関数が実行されます。

GUI_Exec 関数の代わりに GUI_Delay 関数を使用することができます。GUI_Delay 関数の引数(ms 単位)で指定した時間待機します。待機中は GUI_Delay 関数内部で GUI_Exec 関数が最低 1 回呼び出されます。また、GUI_Delay 関数の引数で指定する待ち時間は最短であり、GUI_Exec 関数によるコールバック関数の処理負荷が高いときは指定どおりの時間にならない場合があります。そのため、待機時間に余裕がある処理の待機関数として使用してください。

## 2.5　ウィンドウ

ウィンドウには下記の種類とステータスがあります。

<種類>

- 子ウィンドウ/孫ウィンドウ

  親と呼ばれる別のウィンドウに対して相対的に定義されるウィンドウです。 親ウィンドウが移動すると、その子ウィンドウもそれに応じて移動します。 子ウィンドウは常に親ウィンドウ内に含まれます。子ウィンドウ内で新たに子ウィンドウ(孫ウィンドウ)を作成することもできます。

- 親ウィンドウ

  子ウィンドウの親にあたるウィンドウです。

- デスクトップウィンドウ

  画面全体をカバーする最下部のウィンドウです。他のウィンドウが定義されていないときにデフォルト（アクティブな）ウィンドウになります。そのため、すべてのウィンドウはこのデスクトップウィンドウの子孫となります。複数のレイヤが定義された場合、レイヤごとにデスクトップウィンドウが生成されます。

<ステータス>

- アクティブウィンドウ(カレントウィンドウ)

  現在描画操作に使用されているウィンドウです。最上位のウィンドウとはかぎりません。

- ショウウィンドウ/ハイドウィンドウ

  表示/非表示のウィンドウです。ウィンドウ作成時に表示/非表示を指定できたり、API 関数を使って表示/非表示を切り替えることができます。

基本的なウィンドウは以下の API 関数で作成できます。

```
WM_HWIN WM_CreateWindow(int x0,
                        int y0,
                        int width,
                        int height,
                        U32 Style,
                        WM_CALLBACK * cb,
                        int NumExtraBytes);

WM_HWIN WM_CreateWindowAsChild(int x0,
                               int y0,
                               int width,
                               int height,
                               WM_HWIN hParent,
                               U32 Style,
                               WM_CALLBACK * cb,
                               int NumExtraBytes);
```

- x0, y0
  ウィンドウの位置です。子ウィンドウのときは親ウィンドウに対しての相対位置です。

- width, height
  ウィンドウのサイズです。

- hParent, (WM_CreateWindowAsChild 関数)
  親ウィンドウを指定します。NULL を指定した場合、デスクトップウィンドウが親ウィンドウとなります。

- Style
  各フラグ(”WM_CF_xxxx”)を指定します。フラグは OR で複数指定可能です。WM_CF_SHOW を含めるとウィンドウが始めから表示されます。また、WM_CF_MEMDEV を指定すると、ウィンドウの描画にメモリデバイスが使用されます。フラグの詳細は下記 URL をご参照ください。
  https://www.segger.com/doc/UM03001_emWin.html#Window_create_flags

- cb
  ウィンドウの動作を定義するコールバック関数へのポインタです。 このコールバック関数内で、WM_PAINT などのイベント(メッセージ)に応じて処理を記述します。
  メッセージの詳細は下記 URL をご参照ください。
  https://www.segger.com/doc/UM03001_emWin.html#List_of_messages

  以下は、ウィンドウ領域を赤色で塗りつぶすコールバック関数の例です。

```
static void cbBk(WM_MESSAGE * pMsg)
{
  switch (pMsg->MsgId) {
    case WM_PAINT:
        GUI_SetBkColor(GUI_RED);
        GUI_Clear();
    break;
    default:
        WM_DefaultProc(pMsg);
    break;
    }
}
```

  コールバック関数の引数である pMsg には様々な情報が含まれています。
  例えば、pMsg->MsgId は上記例のように各イベントメッセージが格納され、pMsg->hWin はコールバック関数が属するウィンドウのプロパティを参照できます。
  https://www.segger.com/doc/UM03001_emWin.html#WM_MESSAGE

- NumExtraBytes
  ユーザデータ用に確保するバイト数を指定します。詳細は WM_SetUserData 関数の説明を参照してください。

## 2.6　ダイアログ

ダイアログは、ウィンドウやウィンドウ内のパーツ（ウィジェット）の配置を構造体で定義することでまとめて管理できる機能です。ウィンドウ内への複数ウィジェットの配置や、構成が変わらない場合に有用です。

基本的なダイアログは以下の API 関数で作成できます。

```
WM_HWIN GUI_CreateDialogBox(  const GUI_WIDGET_CREATE_INFO * paWidget,
                              int NumWidgets,
                              WM_CALLBACK * cb,
                              WM_HWIN hParent,
                              int x0,
                              int y0);
```

- paWidget
  構造体変数へのポインタを指定します。GUI_WIDGET_CREATE_INFO 型の変数を定義し、ウィンドウを含む各ウィジェットを設定します。
  https://www.segger.com/doc/UM03001_emWin.html#Resource_table

```
static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
  { FRAMEWIN_CreateIndirect, "Dialog", 0, 10, 10, 180, 230, 0, 0},
  { BUTTON_CreateIndirect, "OK", GUI_ID_OK, 100, 5, 60, 20, 0, 0},
  { EDIT_CreateIndirect, NULL, GUI_ID_EDIT0, 60, 55, 100, 15, 0, 50}
};
```

- NumWidgets
  ダイアログに含まれるウィジェットの総数です。直接数値を設定してもよいですが、GUI_COUNTOF(“構造体変数名”)と記述することで構造体変数のサイズに応じて動的に対応できます。

- cb
  ダイアログ内の動作を定義するコールバック関数へのポインタです。通常のウィンドウとほとんど同じですが、初期化のメッセージと処理内容が異なります。

```
static void _cbCallback(WM_MESSAGE * pMsg) {
  WM_HWIN hItem;
  WM_HWIN hWin;
  hWin = pMsg->hWin;
  switch (pMsg->MsgId) {
  case WM_INIT_DIALOG:
    hItem = WM_GetDialogItem(hWin, GUI_ID_EDIT0);
    EDIT_SetText(hItem, "EDIT widget 0");
      :
      :
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}
```

ダイアログでは、WM_INIT_DIALOG というメッセージが初期化時に一度だけ実行されます。この中でダイアログ内のウィジェットに対して設定します。

- hParent
  親ウィンドウを指定します。NULL を指定した場合、デスクトップ ウィンドウが親ウィンドウとなります。

- x0, y0
  ダイアログの位置です。親ウィンドウに対しての相対位置です。

emWin に付属の GUI Builder でデザインした画面は、このダイアログで構成されています。

ダイアログの詳細は下記 URL をご参照ください。

https://www.segger.com/doc/UM03001_emWin.html#Dialogs

本サンプルプログラムでは、1.1 ホーム画面や 1.3 空調制御 の画面でダイアログを使用しています。

## 2.7　イメージ

emWin では PNG、JPG、GIF などの一般的な画像形式に対応しています。

https://www.segger.com/doc/UM03001_emWin.html#Displaying_bitmap_files

本サンプルプログラムで使用しているすべての画像は、emWin に付属の Bitmap Converter を使用して C ソースに変換しています。このツールを使用すると、画像を emWin 固有の形式（ビットマップ）に変換できます。システム内部でのみ使用する画像はあらかじめ emWin ビットマップに変換することで処理時間を短縮できます。

https://www.segger.com/doc/UM03001_emWin.html#Bitmap_Converter

本サンプルプログラムで使用する画像は下記形式で出力しています。この出力形式は 2.1 ディスプレイドライバ（Flex Color ドライバ）とカラーモード で説明したカラーモード(GUICC_565)に合わせると効率的に処理されます。

- High color (565)　… 16bit BGR565 画像
- High color with Alpha (565)　… 16bit BGR565 画像(透過度付き)

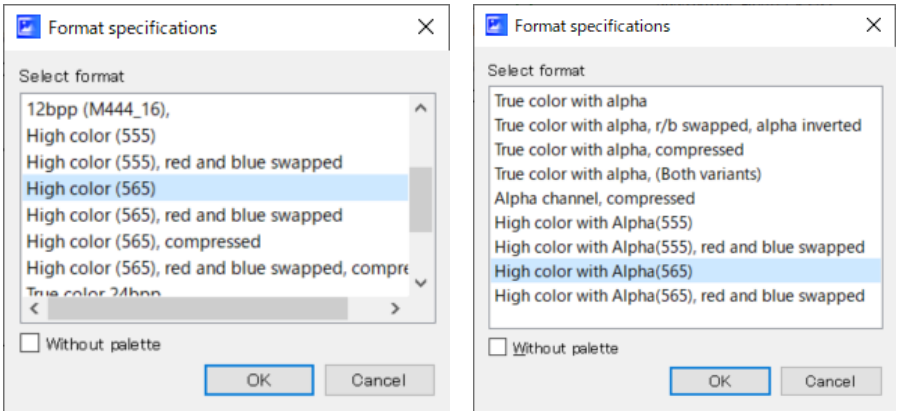



図 2.1　画像フォーマット

## 2.8　フォント

emWin では多種多様なフォントを使用する仕組みが提供されています。ライブラリに付属しているフォントや、外部メモリから SIS や CBF 形式のフォントを取り込んで使用することも可能です。

https://www.segger.com/doc/UM03001_emWin.html#Fonts

また、emWin に付属の Font Converter を使用すると、Windows PC にインストールされているフォントを emWin 形式のフォントに変換できます。 アンチエイリアスの有無の指定や、アプリケーションで使用する文字だけを指定することもできます。ただし、フォントのライセンスは含まれておりませんので、ユーザがライセンスを取得する必要があります。

https://www.segger.com/doc/UM03001_emWin.html#Font_Converter

日本語フォントを使用する際、emWin では SJIS エンコードなどにも対応しておりますが、極力 UTF-8 をご使用いただくことを推奨いたします。emWIn 付属の U2C というツールで UTF-8 のテキストデータを文字コードに変換することができます。日本語フォント以外にも同様に使用できます。なお、UTF-8 のテキストファイルを保存する際、「BOM (Byte Order Mark)」付きで保存してください。

https://www.segger.com/doc/UM03001_emWin.html#Using_U2C_dot_exe_to_convert_UTF_8_text_into_C_code

本サンプルプログラムでは emWin に付属しているフォントのみ使用しています。

## 2.9　アニメーション

emWin によるアニメーションは特定の時間を 32768 分割して 0 から 32767 までインクリメントしていくメカニズムです。このメカニズムを使用して、オブジェクトの座標やカラーコード、その他の値をアニメーション化できます。

https://www.segger.com/doc/UM03001_emWin.html#Animations

以下の関数を呼び出すことでアニメーションを作成できます。

```
GUI_ANIM_HANDLE GUI_ANIM_Create( GUI_TIMER_TIME Period,
                                 unsigned MinTimePerSlice,
                                 void * pVoid,
                                 void (* pfSlice)(int, void *));
```

- Period
  アニメーションの実行期間(ms)です。

- MinTimePerSlice
  アニメーション処理(コールバック関数)の実行周期(ms)です。

- pVoid
  ユーザ定義データへのポインタです。

- pfSlice
  コールバック関数へのポインタです。NULL を設定できます。この関数はアニメーションアイテムのコールバック関数が実行される前後で実行されます。複数のアニメーションアイテムが同時に有効な場合、最初のアニメーションアイテムの前と最後のアニメーションアイテムの後に実行されます。このとき、第一引数には GUI_ANIM_START(“0”)と GUI_ANIM_END(“2”)がそれぞれ渡されます。

アニメーションを作成後、ユーザはアニメーションアイテムを追加する必要があります。

```
int GUI_ANIM_AddItem(GUI_ANIM_HANDLE hAnim,
                     GUI_TIMER_TIME ts,
                     GUI_TIMER_TIME te,
                     GUI_ANIM_GETPOS_FUNC pfGetPos,
                     void * pVoid,
                     GUI_ANIMATION_FUNC * pfAnim);
```

- hAnim
  アニメーションのハンドルです。

- ts, te
  アニメーションの実行期間内におけるこのアイテムの開始時間と終了時間です。

- pVoid
  ユーザ定義データへのポインタです。

- pfAnim
  アニメーションアイテムのコールバック関数へのポインタです。 このコールバック関数はこのアイテムの開始時間から終了時間において、GUI_ANIM_Creat 関数の MinTimePerSlice で設定した周期と一致したときに実行されます。

アニメーションアイテムを設定後、GUI_ANIM_StartEx 関数を実行すると、アニメーションが開始されます。

```
void GUI_ANIM_StartEx(  GUI_ANIM_HANDLE hAnim,
                        int NumLoops,
                        void (* pfOnDelete)(void * pVoid));
```

- hAnim
  アニメーションのハンドルです。

- NumLoops
  アニメーションを実行する回数を指定します。 “-1” を設定すると無期限に実行されます。

- pfOnDelete
  アニメーションが削除されるときに呼び出されるコールバック関数へのポインタです。 NULL を設定できます。

本サンプルプログラムでは 1.2 RX ロゴ表示 にてアニメーションを使用しています。

## 2.10 注意事項

emWin のアンチエイリアシングや半透明ビットマップを使った機能を使用するためには、重ね合わせるピクセルの背面と前面の色情報を取得する必要があるため、キャッシュまたはメモリデバイスを使用する必要があります。

## 2.11 その他の情報

その他の emWin に関する情報は Segger 社の下記 URL をご参照ください。

- https://wiki.segger.com/emWin
- https://wiki.segger.com/emWin_Examples

## 3. 動作確認条件

本サンプルプログラムは、下記の条件で動作を確認しています。

表 3.1　動作確認条件

| 項目 | 内容 |
|---|---|
| 使用マイコン | R5F5671EHDFB (RX671 グループ) |
| 動作周波数 | • メインクロック: 24MHz<br>• PLL:240MHz (メインクロック 1 分周  10 逓倍)<br>• システムクロック(ICLK): 120MHz (PLL 2 分周)<br>• 周辺モジュールクロック A(PCLKA): 120MHz (PLL 2 分周)<br>• 周辺モジュールクロック B(PCLKB): 60MHz (PLL 4 分周)<br>• 周辺モジュールクロック C(PCLKD): 60MHz (PLL 4 分周)<br>• 周辺モジュールクロック D(PCLKD): 60MHz (PLL 4 分周)<br>• FlashIF クロック(FCLK): 60MHz (PLL 4 分周) |
| 動作電圧 | 3.3V |
| 統合開発環境 | ルネサスエレクトロニクス製<br>e² studio Version 2023-04 |
| C コンパイラ | ルネサスエレクトロニクス製<br>C/C++ Compiler Package for RX Family V.3.05.00 |
| | コンパイルオプション<br>-lang = c99 |
| iodefine.h のバージョン | V1.00 |
| エンディアン | リトルエンディアン |
| 動作モード | シングルチップモード |
| プロセッサモード | スーパバイザモード |
| サンプルプログラムのバージョン | Version 1.00 |
| エミュレータ | オンボードエミュレータ(E2 Lite エミュレータ) |
| 使用ボード | EK-RX671 (製品型名: RTK5EK6710S00001BE) |
| 使用 LCD | MSP2807 (製造メーカー: Kuongshun Electronic Limited) |

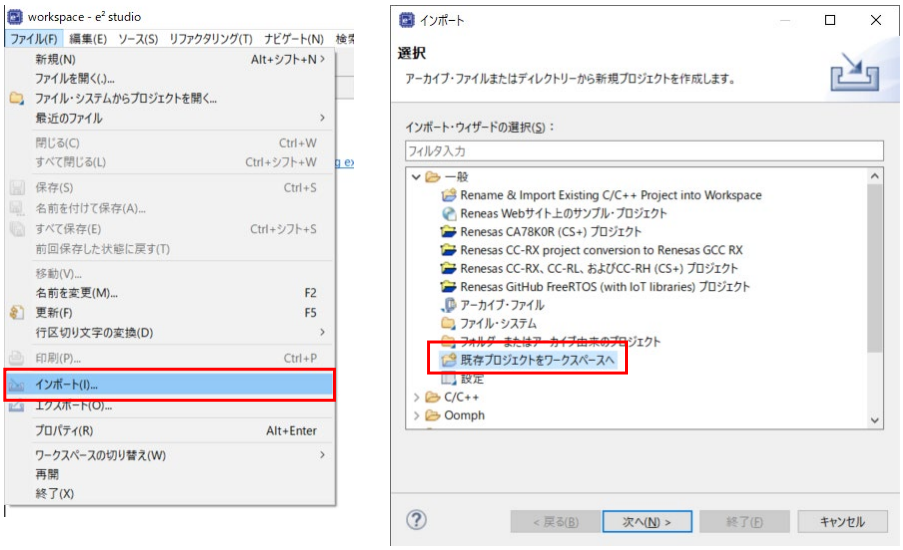
# 4.　サンプルプロジェクトの実行方法

## 4.1　ハードウェアの準備

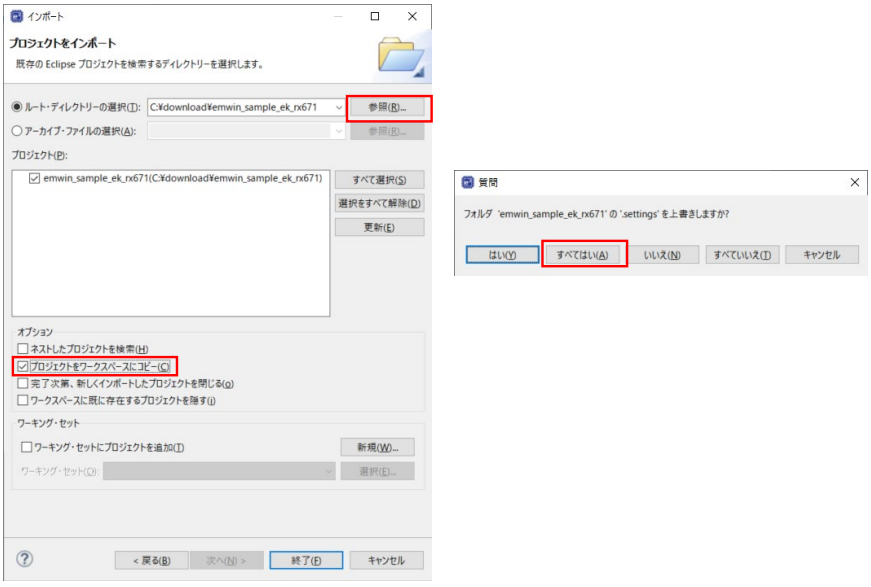
本サンプルコードを動かすためには EK-RX671 のジャンパ設定および LCD(MSP2807)との接続が必要です。接続に必要な部品を用意してください。ジャンパ設定と接続の方法は 5 ハードウェアの説明を参照してください。

## 4.2　プロジェクトのインポート

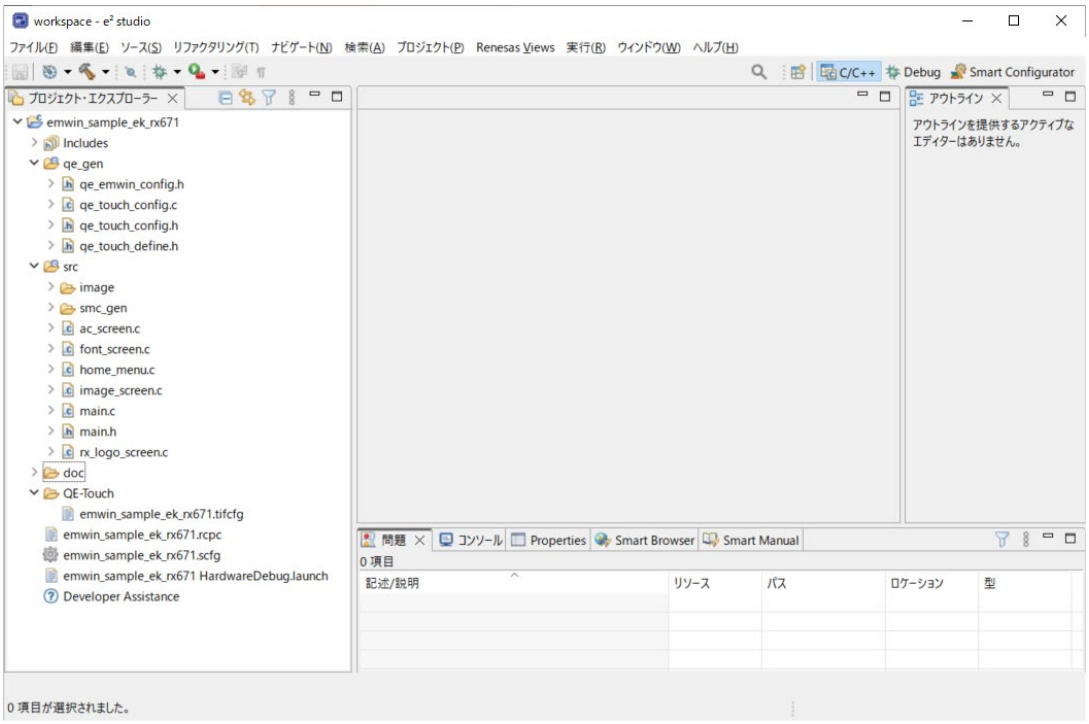
「ファイル(F)」→ 「インポート(I)」を選択すると「インポート 選択」ウィンドウが表示されます。「一般」から「既存のプロジェクトをワークスペースへ」選択し、「次へ(N)」をクリックします。



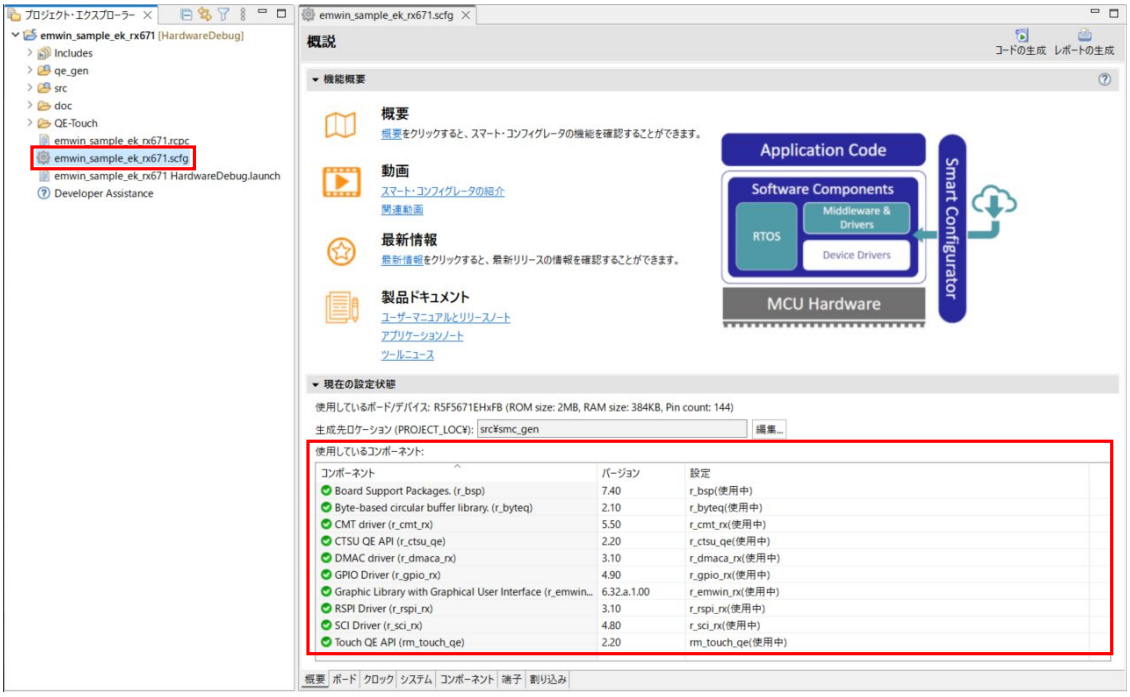

「プロジェクトをインポート」ウィンドウが表示されるので、「ルート・ディレクトリーの選択」の「参照」をクリックし、ダウンロードしたサンプルプロジェクトのフォルダを選択します。プロジェクトが認識されたら、「プロジェクトをワークスペースにコピー」にチェックを入れ、「終了(E)」をクリックします。「質問」のウィンドウが表示される場合、「すべてはい(A)」をクリックしてください。



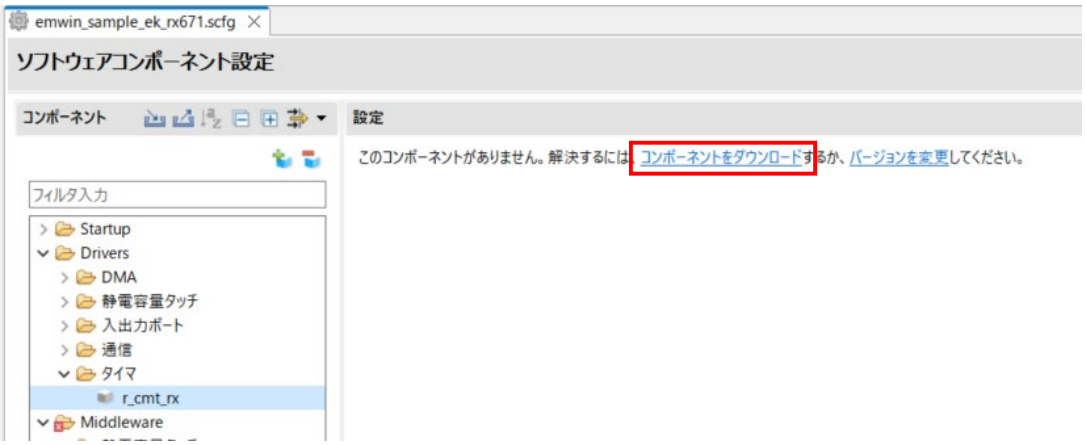
インポートが完了すると、プロジェクト・エクスプローラにサンプルプロジェクトが表示されます。




Smart Configurator を起動します。「概要」タブで「使用しているコンポーネント」を確認し、対象のバージョンがお使いの PC 環境にダウンロードされていることを確認してください。

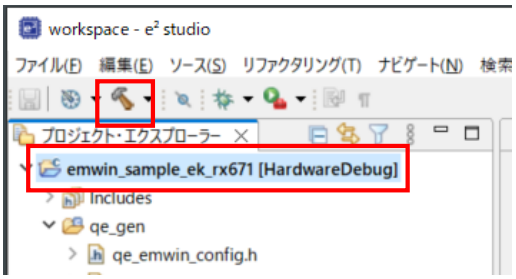


ダウンロードされていない場合、「コンポーネント」タブから対象モジュールをクリックすると対象バージョンをダウンロードすることができます。

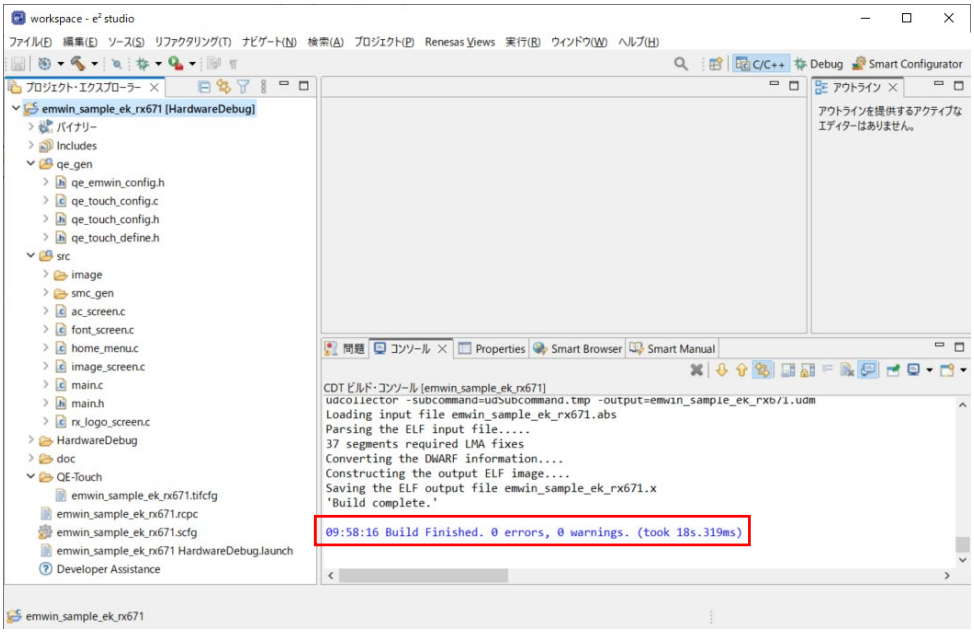



## 4.3　プロジェクトのビルド

ビルド対象のプロジェクトを選択し、ビルドボタンをクリックします。



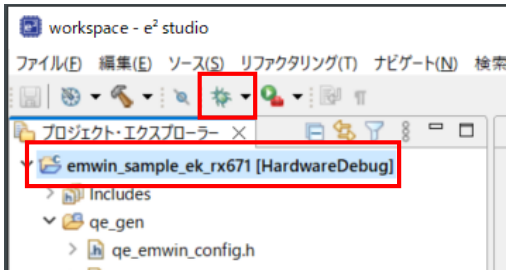

ビルドが完了することを確認してください。



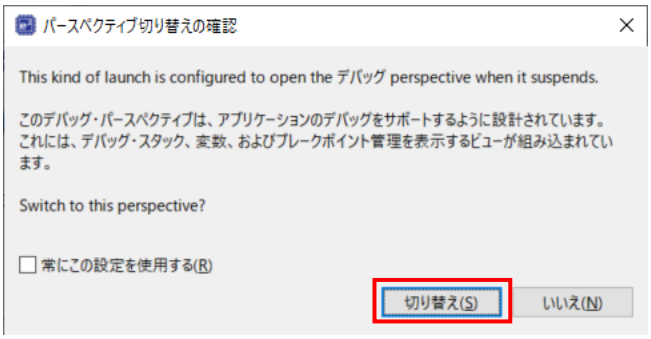
ビルドに失敗した場合、3.動作確認条件に記載のコンパイラがインストールされていない、または無償評価版の試用期間が終了している可能性があります。プロジェクトのプロパティからツールチェーンの設定を確認し、使用可能なコンパイラを設定してください。

## 4.4　デバッグ接続とプログラムの実行

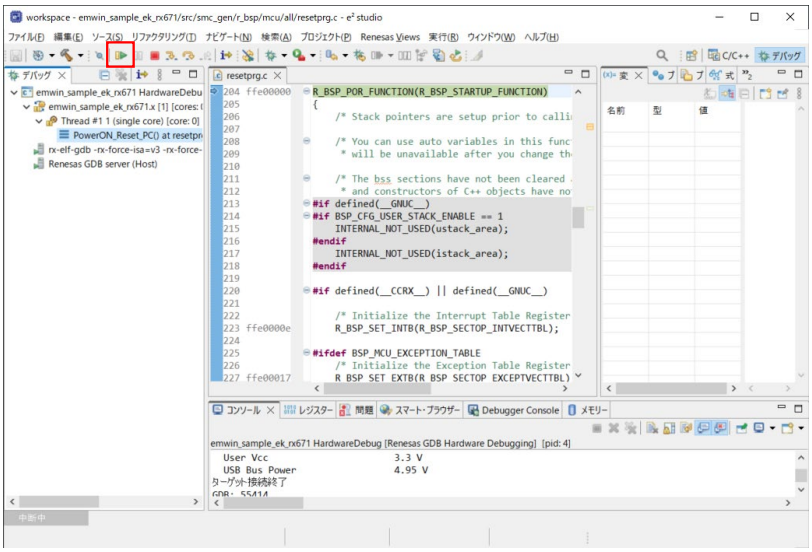
デバッグ対象のプロジェクトを選択し、デバッグボタンをクリックします。



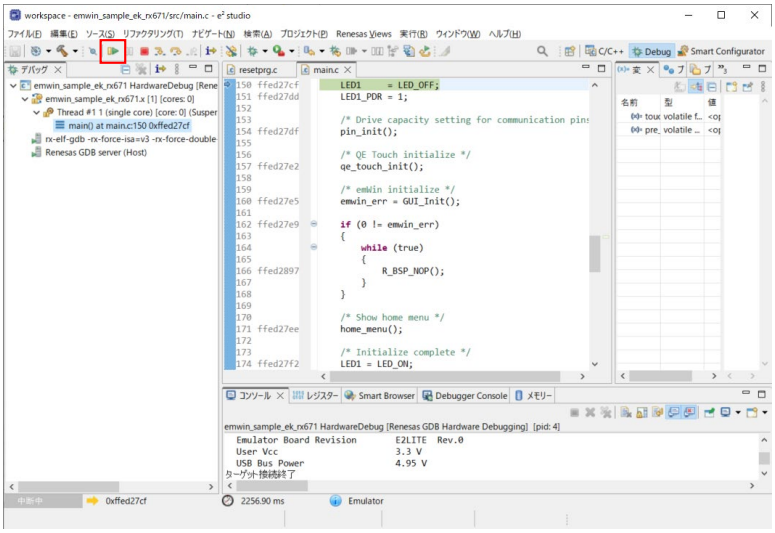

デバッグ接続を開始してしばらくすると、「パースペクティブ切り替えの確認」ウィンドウが表示されるので、「切り替え(S)」ボタンをクリックします。




デバッグパースペクティブに切り替わるので、「再開(M)」ボタンをクリックします。



再度「再開(M)」ボタンをクリックしてプログラムを実行します。



# 5. ハードウェアの説明

## 5.1 ハードウェア構成

図 5.1 にハードウェアの構成を示します。

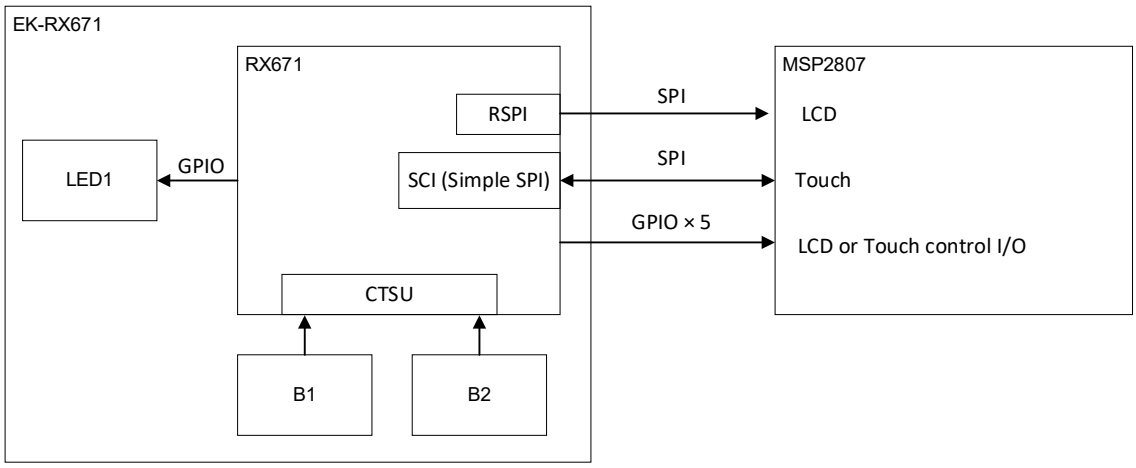



図 5.1　ハードウェアの構成

また、サンプルプログラムを動作させるためには表 5.1 に示す EK-RX671 のジャンパ設定が必要です。

表 5.1　ジャンパ設定

| ジャンパ | 設定 | 説明 |
|---|---|---|
| J30 | オープン | P16 を使用する |

表 5.2　使用 LCD

| LCD 型番 | 解像度 | LCD コントローラ | タッチパネル コントローラ | 動作電圧 | 備考 |
|---|---|---|---|---|---|
| MSP2807 (Kuongshun Electronic Limited) | 240×320 | ILI9341 (ILITEK) | XPT2046 (Xptek) | 3.3V | 本製品と同等品であればサンプルプログラムを動作させることが可能です。 |

## 5.2　LCD との接続

表 5.3 に EK-RX671 と LCD との接続を示します。本表に従って LCD と接続してください。

表 5.3　EK-RX671 と LCD との接続

| EK-RX671 | | emWin 制御端子 | | LCD (MSP2807) | | |
|---|---|---|---|---|---|---|
| ピン番号 | ラベル | | | ピン番号 | ラベル | 説明 |
| J2-1 | 3.3V | - | | 1 | VCC | 3.3V |
| J2-35 | GND | - | | 2 | GND | GND |
| J2-29 | PC3 | EMWIN_LCD_CS_PIN | | 3 | CS | LCD の CS 信号 |
| J2-23 | P50 | EMWIN_DISP_SIGNAL_PIN | | 4 | RESET | LCD のリセット信号 |
| J2-22 | P51 | EMWIN_DATA_CMD_PIN | | 5 | DC/RS | LCD のデータ/コマンド信号 |
| J2-26 | PC6 | MOSIA | | 6 | SDI(MOSI) | LCD のデータ入力 |
| J2-27 | PC5 | RSPCKA | | 7 | SCK | LCD のクロック信号 |
| J2-21 | P52 | EMWIN_BACKLIGHT_PIN | ⇔ | 8 | LED | バックライト制御信号 |
| J2-25 | PC7 | MISOA(未使用) | | 9 | SDO(MISO) | 未使用 |
| J2-11 | P17 | SCK1 | | 10 | T_CLK | タッチパネルのクロック |
| J2-14 | P14 | EMWIN_TOUCH_CS_PIN | | 11 | T_CS | タッチパネルの CS 信号 |
| J2-12 | P16 | SMOSI1 | | 12 | T_DIN | タッチパネルのデータ入力 |
| J2-13 | P15 | SMISO1 | | 13 | T_DO | タッチパネルのデータ出力 |
| - | | | | 14 | T_IRQ | 未使用 |

## 5.3　使用端子と機能

表 5.4 に使用端子と機能を示します。

表 5.4　使用端子と機能

| デバイス | 端子名 | 入出力 | 内容 |
|---|---|---|---|
| LCD (MSP2807) | PC6/MOSIA | 出力 | データ出力 |
| | PC7/MISOA | 入力 | データ入力(未使用)<br>プルアップまたはプルダウンしてください。 |
| | PC5/RSPCKA | 出力 | クロック出力 |
| | PC3 | 出力 | CS 信号出力 |
| | P50 | 出力 | LCD リセット信号出力 |
| | P51 | 出力 | データ/コマンド信号出力 |
| | P52 | 出力 | バックライト制御信号出力 |
| タッチパネル (MSP2807) | P17/SCK1 | 出力 | クロック出力 |
| | P16/SMOSI1 | 出力 | データ出力 |
| | P15/SMISO1 | 入力 | データ入力 |
| | P14 | 出力 | CS 信号出力 |
| タッチセンサ (EK-RX671) | PC4/TSCAP | 入力 | LPF 接続用端子 |
| | P33/TS1 | 入力 | タッチ端子 1 |
| | P24/TS5 | 入力 | タッチ端子 5 |
| LED | P56 | 出力 | LED 点灯 |

# 6.　ソフトウェアの説明

## 6.1　ソフトウェア構成

図 6.1 にソフトウェアの構成を示します。emWin FIT モジュールは emWin の本体となるライブラリと、RX MCU とライブラリを繋ぐインタフェースで構成されており、アプリケーションは emWin の API のみで画面を制御することができます。

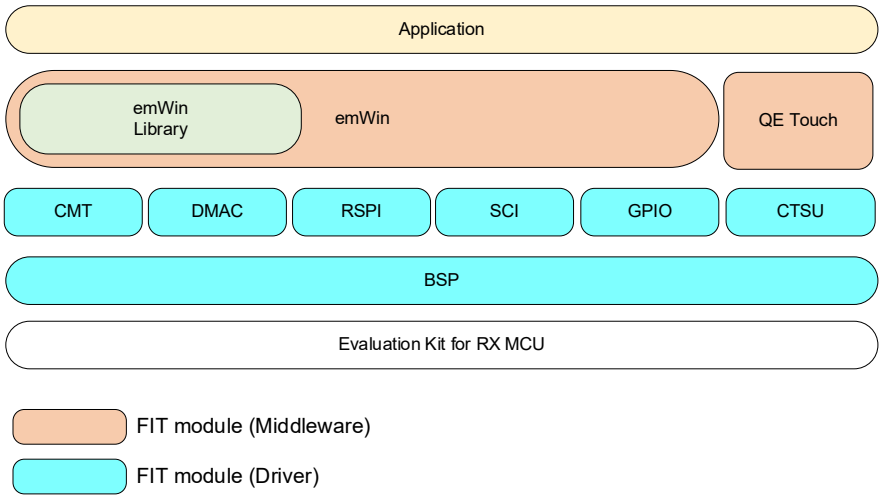



図 6.1　ソフトウェアの構成

## 6.2　使用 FIT モジュール

本サンプルプログラムで使用する FIT モジュールを表 6.1 に示します。

表 6.1 使用 FIT モジュール一覧

| モジュール | ドキュメントタイトル | ドキュメント番号 |
|---|---|---|
| BSP | RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology | R01AN1685 |
| emWin | RX ファミリ emWin v6.32 モジュール Firmware Integration Technology | R01AN6771 |
| CMT | RX ファミリ CMT モジュール Firmware Integration Technology | R01AN1856 |
| GPIO | RX ファミリ GPIO モジュール Firmware Integration Technology | R01AN1721 |
| RSPI | RX ファミリ RSPI モジュール Firmware Integration Technology | R01AN1827 |
| SCI | RX ファミリ SCI モジュール Firmware Integration Technology | R01AN1815 |
| DMAC | RX ファミリ DMAC モジュール Firmware Integration Technology | R01AN2063 |
| CTSU | RX ファミリ QE CTSU モジュール Firmware Integration Technology | R01AN4469 |
| QE Touch | RX ファミリ QE Touch モジュール　Firmware Integration Technology | R01AN4470 |
| BYTEQ | RX ファミリ バイト型キューバッファ（BYTEQ）モジュール Firmware Integration Technology | R01AN1683 |

## 6.3　プロジェクトの構成

図 6.2 にプロジェクトの構成を示します。

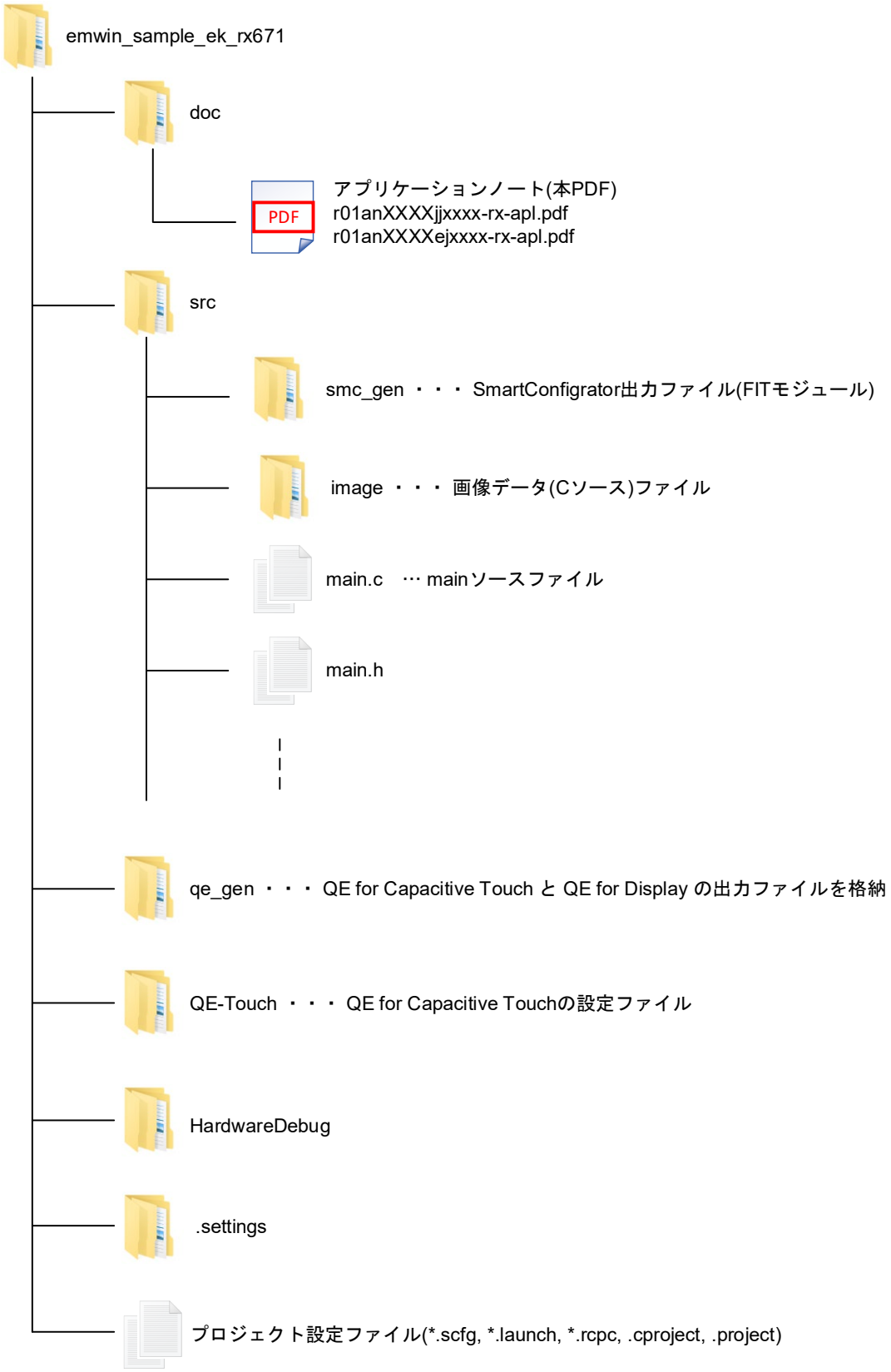



図 6.2　プロジェクトの構成

## 6.4　ファイル構成

本サンプルプログラムで使用するファイルを以下に示します。なお、Smart Configurator や QE から生成されるファイルは除きます。

表 6.2　サンプルプログラムで使用するファイル

| ファイル名 | | 概要 | 備考 |
|---|---|---|---|
| src フォルダ | | | |
| | main.c | メイン処理 | |
| | main.h | 共通ヘッダ | |
| | home_menu.c | ホーム画面の処理 | |
| | rx_logo_screen.c | RX ロゴ画面の処理 | |
| | ac_ screen.c | 空調制御画面の処理 | |
| | image_ screen.c | 画像表示画面の処理 | |
| | font_screen.c | フォント表示画面の処理 | |
| src¥image フォルダ | | | |
| | renesas_logo.c | ルネサスロゴ | Bitmap Converter で生成 |
| | icon1.c | ホーム画面のアイコン 1 | |
| | icon2.c | ホーム画面のアイコン 2 | |
| | icon3.c | ホーム画面のアイコン 3 | |
| | icon4.c | ホーム画面のアイコン 4 | |
| | rx_logo.c | RX ロゴ | |
| | home_btn.c | ホームボタン | |
| | image1.c | 画像 1 | |
| | image1_thumb.c | 画像 1 のサムネイル | |
| | image2.c | 画像 2 | |
| | image2_thumb.c | 画像 2 のサムネイル | |
| | image3.c | 画像 3 | |
| | image3_thumb.c | 画像 3 のサムネイル | |
| | image4.c | 画像 4 | |
| | image4_thumb.c | 画像 4 のサムネイル | |
| | left_btn_down.c | 左向きボタン(押下時) | |
| | left_btn_up.c | 左向きボタン(非押下時) | |
| | right_btn_down.c | 右向きボタン(押下時) | |
| | right_btn_up.c | 右向きボタン(非押下時) | |
| | mode_btn_off.c | モード選択（非選択時） | |
| | mode_btn_on.c | モード選択（選択時） | |
| | temp_meter.c | 温度メータ画像 | |
| | windmill.c | 風量メータ画像 | |

## 6.5　詳細

本節では各処理の詳細を説明します。本サンプルプログラムの関数や変数を用いて説明しているため、サンプルプログラムを参照しながらご確認ください。

### 6.5.1　メイン処理(main.c)

#### 6.5.1.1　仕様

emWin およびタッチキーの初期化後、メインループで GUI の更新とタッチキーのスキャンを行います。

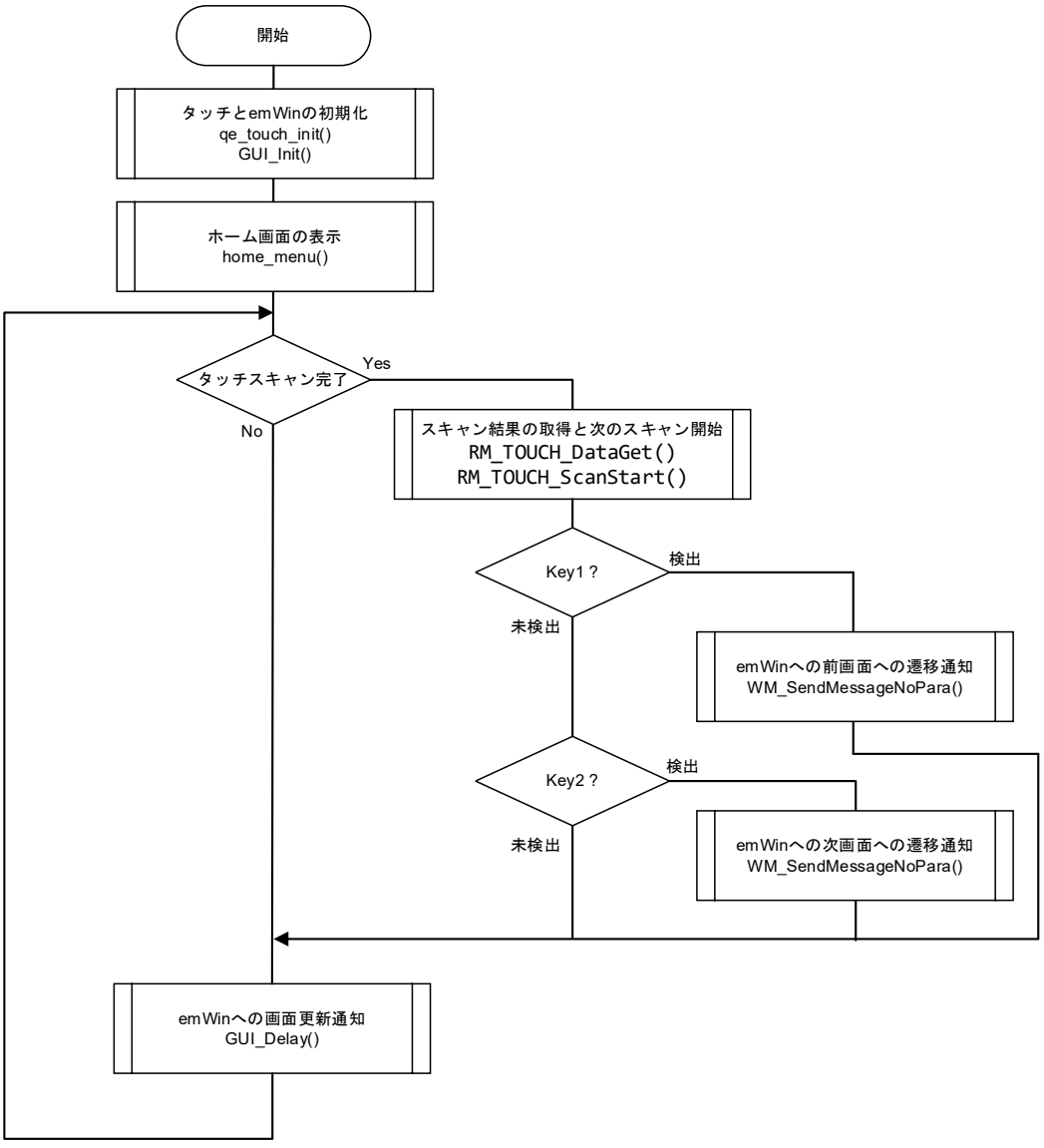



図 6.3　メイン処理の概要フロー

#### 6.5.1.2　変数

グローバル変数の一覧を示します。

表 6.3　グローバル変数

| 型 | 変数名 | 内容 |
|---|---|---|
| WM_HWIN | g_window | 表示中のアクティブウィンドウのハンドル |

#### 6.5.1.3　定数

本サンプルプログラムにおいて共通で使用する定数の一覧を示します。

表 6.4　定数（サンプルプログラム共通 ：main.h）

| 定数名 | 設定値 | 内容 |
|---|---|---|
| LCD_SIZE_X | 320 | 画面の横幅 |
| LCD_SIZE_Y | 240 | 画面の高さ |
| HOME_BTN_POS_X | 259 | ホームボタンの X 座標 |
| HOME_BTN_POS_Y | 0 | ホームボタンの Y 座標 |
| MY_NEXT_SCREEN | WM_USER + 0 | ユーザ定義のメッセージ：次のスクリーン |
| MY_BACK_SCREEN | WM_USER + 1 | ユーザ定義のメッセージ：前のスクリーン |
| BT1_ON | 1 | タッチキー1 の検出判定値 |
| BT2_ON | 2 | タッチキー2 の検出判定値 |
| LED_ON | 1 | LED の点灯 |
| LED_OFF | 0 | LED の消灯 |
| LED1 | PORT5.PODR.BIT.B6 | LED1(P56)の PODR レジスタのビット |
| LED1_PDR | PORT5.PDR.BIT.B6 | LED1(P56)の PDR レジスタのビット |

#### 6.5.1.4　関数

メイン処理の関数の一覧を示します。

表 6.5　関数(main.c)

| 関数名 | 概要 |
|---|---|
| main | メイン処理 |
| pin_init | 通信端子の駆動能力設定処理 |
| qe_touch_init | タッチキーの初期化と初回のスキャン処理 |

#### 6.5.1.5　関数仕様

メイン処理の関数仕様を示します。

| main | |
|---|---|
| 概　要 | メイン処理 |
| ヘッダ | なし |
| 宣　言 | void main(void) |
| 説　明 | emWin およびタッチキーの初期化を行います。その後、メインループで GUI の更新とタッチキーのスキャンを行います。 |
| 引　数 | なし |
| リターン値 | なし |

| pin_init | |
|---|---|
| 概　要 | 通信端子の駆動能力設定処理 |
| ヘッダ | なし |
| 宣　言 | void pin_init(void) |
| 説　明 | LCD との通信で使用する端子の駆動能力を設定します。 |
| 引　数 | なし |
| リターン値 | なし |

| qe_touch_init | |
|---|---|
| 概　要 | タッチキーの初期化と初回のスキャン処理 |
| ヘッダ | なし |
| 宣　言 | void qe_touch_init(void) |
| 説　明 | タッチキーの端子設定およびタッチキーの初期化（RM_TOUCH_Open 関数の実行）後、初回のスキャンを実行します。 |
| 引　数 | なし |
| リターン値 | なし |

### 6.5.2　ホーム画面処理(home_menu.c)

#### 6.5.2.1　仕様

本画面は図 6.4 で示すパーツで構成されています。

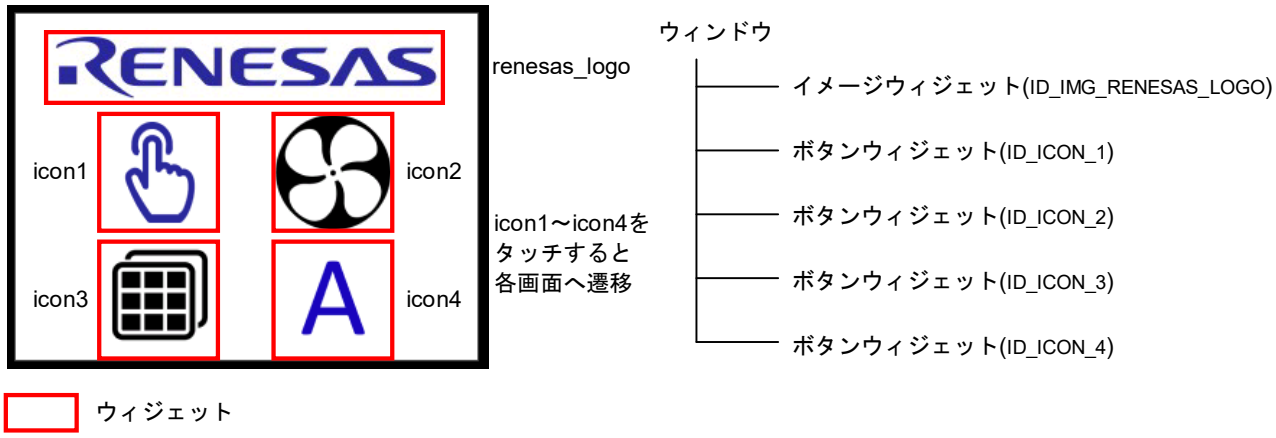



図 6.4　ホーム画面の構成

ホーム画面はダイアログを使用して構成しています。ダイアログを使用すると、構造体変数 g_dialog_home_menu を用いてウィンドウやウィジェットの情報をまとめて設定できます。ダイアログ内の動作はコールバック関数 cb_home_menu で管理されており、メッセージ p_msg->MsgId に応じた処理を実装しています。

表 6.6　メッセージに応じた処理

| メッセージ | 内容 |
|---|---|
| WM_INIT_DIALOG | ダイアログ生成時に 1 度だけ実行されます。<br>ダイアログに配置する各ウィジェットを初期化（背景の設定やボタンの画像の設定）します。 |
| WM_NOTIFY_PARENT | 各ウィジェットでイベントが発生した場合に実行されます。<br>WM_GetId(p_msg->hWinSrc)でどのボタンからのイベント通知なのかを判断し、ボタンに応じて次の画面へ遷移します。<br>遷移前には WM_DeleteWindow(p_msg->hWin)でホーム画面のダイアログを削除します。 |
| MY_NEXT_SCREEN | ユーザ定義のメッセージです。<br>メインループからの WM_SendMessageNoPara(g_window, MY_NEXT_SCREEN)を受けて実行されます。 |
| MY_BACK_SCREEN | ユーザ定義のメッセージです。<br>メインループからの WM_SendMessageNoPara(g_window, MY_BACK_SCREEN)を受けて実行されます。 |

#### 6.5.2.2　変数

ホーム画面処理のグローバル変数の一覧を示します。

表 6.7　グローバル変数

| 型 | 変数名 | 内容 |
|---|---|---|
| extern GUI_CONST_STORAGE GUI_BITMAP | bmrenesas_logo | renesas_logo のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmicon1 | icon1 のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmicon2 | icon2 のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmicon3 | icon3 のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmicon4 | icon4 のデータ変数 |
| extern WM_HWIN | g_window | 表示中のアクティブウィンドウのハンドル |
| static const GUI_WIDGET_CREATE_INFO | g_dialog_home_menu[] | ダイアログ上で使用するウィジェットの管理情報 |

#### 6.5.2.3　定数

ホーム画面処理の定数の一覧を示します。

表 6.8　定数

| 定数名 | 設定値 | 内容 |
|---|---|---|
| ID_WINDOW_HOME | GUI_ID_USER + 0 | ウィンドウの管理 ID |
| ID_IMG_RENESAS_LOGO | GUI_ID_IMAGE0 | イメージウィジェット(renesas_logo)の管理 ID |
| ID_ICON_1 | GUI_ID_BUTTON1 | ボタンウィジェット(icon1)の管理 ID |
| ID_ICON_2 | GUI_ID_BUTTON2 | ボタンウィジェット(icon2)の管理 ID |
| ID_ICON_3 | GUI_ID_BUTTON3 | ボタンウィジェット(icon3)の管理 ID |
| ID_ICON_4 | GUI_ID_BUTTON4 | ボタンウィジェット(icon4)の管理 ID |

#### 6.5.2.4　関数

ホーム画面処理の関数の一覧を示します。

表 6.9　関数

| 関数名 | 概要 |
|---|---|
| cb_home_menu | ホーム画面のコールバック関数 |
| home_menu | ホーム画面のダイアログ生成 |

#### 6.5.2.5 関数仕様

ホーム画面処理の関数仕様を示します。

| cb_home_menu | |
|---|---|
| 概　要 | ホーム画面のコールバック関数 |
| ヘッダ | なし |
| 宣　言 | static void cb_home_menu(WM_MESSAGE * p_msg); |
| 説　明 | ホーム画面の制御を行います。 |
| 引　数 | WM_MESSAGE * p_msg　ダイアログ(ウィンドウ)のメッセージポインタ |
| リターン値 | なし |

| home_menu | |
|---|---|
| 概　要 | ホーム画面のダイアログ生成 |
| ヘッダ | main.h |
| 宣　言 | void home_menu(void); |
| 説　明 | GUI_CreateDialogBox 関数を実行してホーム画面を生成します。 |
| 引　数 | なし |
| リターン値 | なし |

### 6.5.3　RX ロゴ表示処理(rx_logo_screen.c)

#### 6.5.3.1　仕様

本画面は図 6.5 で示すパーツで構成されています。

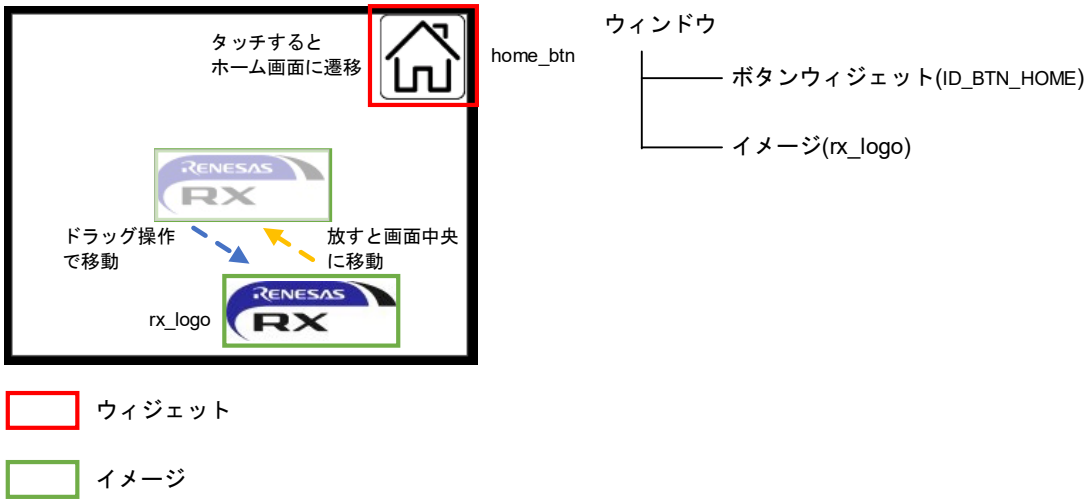



図 6.5　RX ロゴ表示画面の構成

ウィンドウ内の動作はコールバック関数 cb_rx_logo_screen で管理されており、メッセージ p_msg->MsgId に応じた処理を実装しています。

表 6.10　メッセージに応じた処理

| メッセージ | 内容 |
|---|---|
| WM_CREATE | ウィンドウ生成時に 1 度だけ実行されます。<br>ウィンドウ内に配置するボタンウィジェットの初期化と RX ロゴの初期座標を設定します。 |
| WM_PAINT | 画面をクリアして RX ロゴを描画します。 |
| WM_TOUCH | タッチ操作している間実行されます。<br>RX ロゴがタッチされると、WM_SetCapture 関数を実行してドラッグ操作中のタッチイベントを占有します。ドラッグ操作中は、現在のタッチ座標を取得して RX ロゴの座標を更新し、WM_InvalidateWindow 関数で画面の更新を指示します。<br>RX ロゴをはなすと WM_ReleaseCapture 関数でタッチイベントの占有を解除し、アニメーション処理を実行します。 |
| WM_NOTIFY_PARENT | ウィジェットでイベントが発生した場合に実行されます。<br>WM_GetId(p_msg->hWinSrc)でホームボタン(home_btn)からのイベント通知かを判断し、ホーム画面へ遷移します。<br>遷移前には WM_DeleteWindow(p_msg->hWin)で RX ロゴ画面のダイアログを削除します。 |
| MY_NEXT_SCREEN | ユーザ定義のメッセージです。<br>メインループからの WM_SendMessageNoPara(g_window, MY_NEXT_SCREEN)を受けて実行されます。 |
| MY_BACK_SCREEN | ユーザ定義のメッセージです。<br>メインループからの WM_SendMessageNoPara(g_window, MY_BACK_SCREEN)を受けて実行されます。 |

ドラッグ操作後に RX ロゴが画面中央へ戻る処理でアニメーション機能を使用しています。

anim_setup 関数でアニメーションの初期化と開始をしています。アニメーションが開始すると、GUI_ANIM_Create 関数で設定した ANIM_SLICE 時間（100ms）ごとに cb_anim_func 関数が実行されます。cb_anim_func 関数は RX ロゴの座標情報を更新し、WM_InvalidateWindow 関数で画面の更新を指示します。

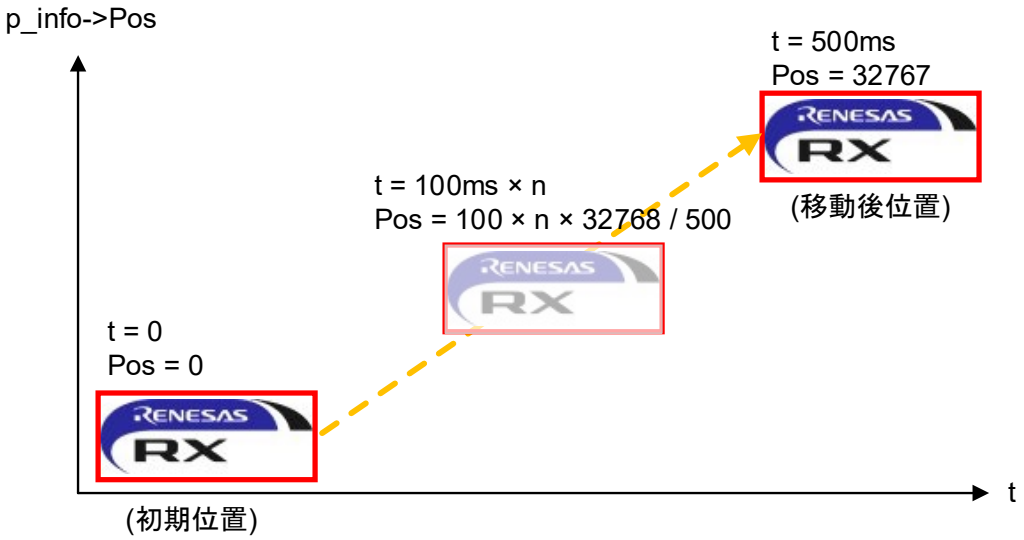



図 6.6　RX ロゴのアニメーション概要

### 6.5.3.2　変数

RX ロゴ表示処理のグローバル変数の一覧を示します。

表 6.11　グローバル変数

| 型 | 変数名 | 内容 |
|---|---|---|
| extern GUI_CONST_STORAGE GUI_BITMAP | bmhome_btn | home_btn のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmrx_logo | rx_logo のデータ変数 |
| WM_HWIN | g_window | 表示中のアクティブウィンドウのハンドル |
| static GUI_POINT | g_logo_pos | rx_logo の座標 |

### 6.5.3.3　定数

RX ロゴ表示処理の定数の一覧を示します。

表 6.12　定数

| 定数名 | 設定値 | 内容 |
|---|---|---|
| ANIM_PERIOD | 500 | アニメーションの実行時間(ms) |
| ANIM_SLICE | 100 | アニメーション処理の実行周期(ms) |
| BM_RX_LOGO | &bmrx_logo | rx_logo のデータ変数へのポインタ |
| ID_BTN_HOME | GUI_ID_BUTTON0 | ボタンウィジェット(home_btn)の管理 ID |

### 6.5.3.4　関数

RX ロゴ表示処理の関数の一覧を示します。

表 6.13　関数

| 関数名 | 概要 |
|---|---|
| cb_anim_func | アニメーション処理の実行 |
| anim_setup | アニメーションの初期化と開始 |
| cb_rx_logo_screen | RX ロゴ画面のコールバック関数 |
| rx_logo_screen | RX ロゴ画面のウィンドウ生成 |

### 6.5.3.5　関数仕様

RX ロゴ表示処理の関数仕様を示します。

| cb_anim_func | |
|---|---|
| 概　要 | アニメーション処理の実行 |
| ヘッダ | なし |
| 宣　言 | static void cb_anim_func(GUI_ANIM_INFO * p_info, void * p_void); |
| 説　明 | GUI_ANIM_AddItem 関数で登録されたアニメーションアイテムのコールバック関数で、定期的に実行されます。RX ロゴの座標を更新し、WM_InvalidateWindow 関数で画面の更新を指示します。 |
| 引　数 | GUI_ANIM_INFO * p_info　アニメーション情報<br>void * p_void　ユーザデータのポインタ |
| リターン値 | なし |

| anim_setup | |
|---|---|
| 概　要 | アニメーションの初期化と開始 |
| ヘッダ | なし |
| 宣　言 | static void anim_setup(WM_HWIN h_win); |
| 説　明 | RX ロゴの現在の座標を anim_data 変数に格納して、GUI_ANIM_Create 関数や GUI_ANIM_AddItem 関数でアニメーション処理の初期化を行います。<br>初期化の際に引数で渡した anim_data 変数は、コールバック関数 cb_anim_func 関数で参照することができます。 |
| 引　数 | WM_HWIN h_win　ウィンドウのハンドル |
| リターン値 | なし |

| cb_rx_logo_screen | |
|---|---|
| 概　要 | RX ロゴ画面のコールバック関数 |
| ヘッダ | なし |
| 宣　言 | static void cb_rx_logo_screen(WM_MESSAGE * p_msg); |
| 説　明 | RX ロゴ画面の制御を行います。 |
| 引　数 | WM_MESSAGE * p_msg　ウィンドウのハンドル |
| リターン値 | なし |

| rx_logo_screen | |
|---|---|
| 概　要 | RX ロゴ画面のウィンドウ生成 |
| ヘッダ | main.h |
| 宣　言 | void rx_logo_screen(void); |
| 説　明 | WM_CreateWindow 関数を実行して RX ロゴ画面を生成します。 |
| 引　数 | なし |
| リターン値 | なし |

### 6.5.4　空調制御処理(ac_screen.c)

#### 6.5.4.1　仕様

本画面は図 6.7 で示すパーツで構成されています。

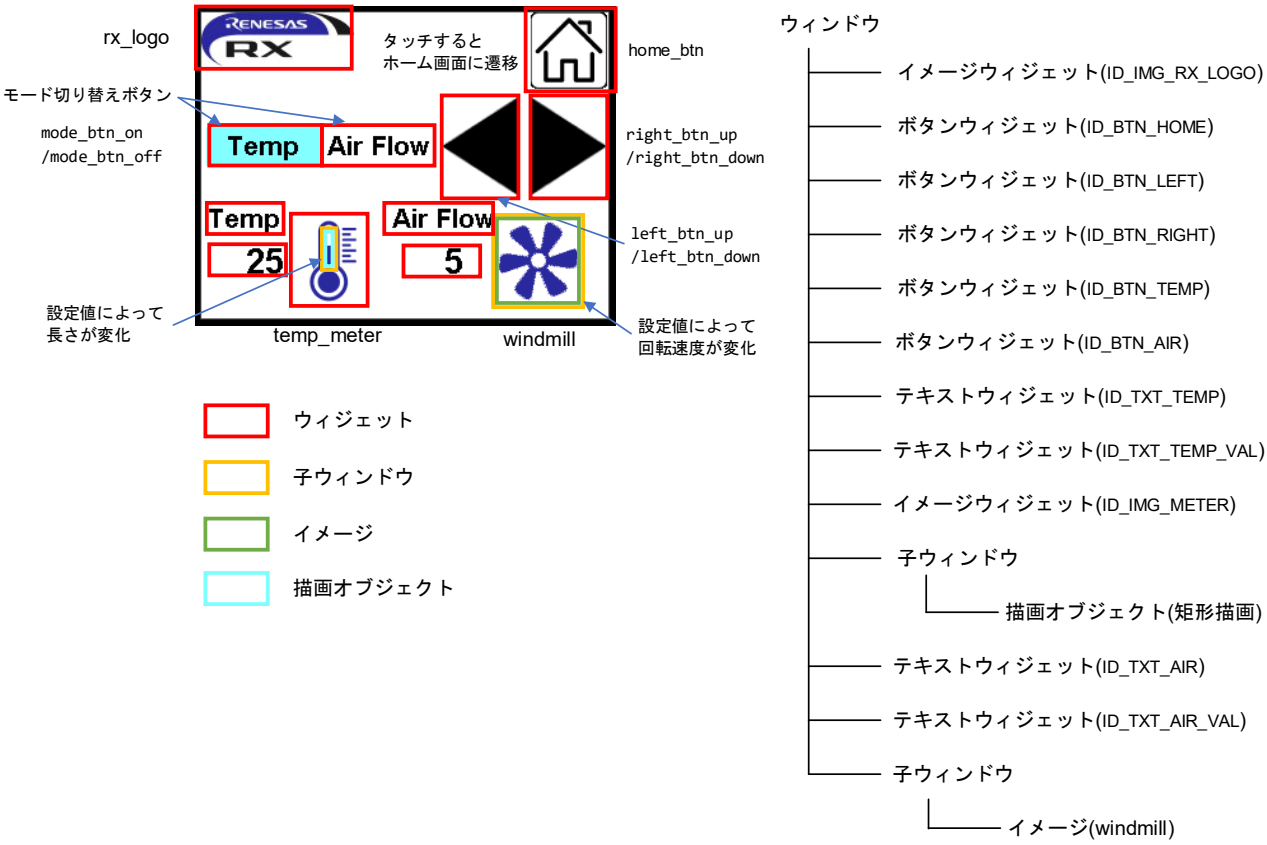



図 6.7　空調制御画面の構成

ウィンドウ内の動作はコールバック関数 cb_ac_screen で管理されており、メッセージ p_msg->MsgId に応じた処理を実装しています。

表 6.14　メッセージに応じた処理(cb_ac_screen)

| メッセージ | 内容 |
|---|---|
| WM_INIT_DIALOG | ウィンドウ生成時に 1 度だけ実行されます。<br>ウィンドウ内に配置するウィジェットの初期化と子ウィンドウの生成を行います。 |
| WM_NOTIFY_PARENT | ウィジェットでイベントが発生した場合に実行されます。<br>WM_GetId(p_msg->hWinSrc)でどのボタンからのイベント通知かを判断し、各処理を行います。 |
| WM_DELETE | 温度メータ、風量メータ用の子ウィンドウを削除します。 |
| MY_NEXT_SCREEN | ユーザ定義のメッセージです。<br>メインループからの WM_SendMessageNoPara(g_window, MY_NEXT_SCREEN)を受けて実行されます。 |
| MY_BACK_SCREEN | ユーザ定義のメッセージです。<br>メインループからの WM_SendMessageNoPara(g_window, MY_BACK_SCREEN)を受けて実行されます。 |

本サンプルプログラムでは、温度メータ用、風量メータ用の二つの子ウィンドウを生成しています。各ウィンドウのコールバック関数は cb_win_temp 関数と cb_win_flow 関数です。

表 6.15　メッセージに応じた処理(cb_win_temp)

| メッセージ | 内容 |
|---|---|
| WM_PAINT | WM_GetUserData 関数にて親ウィンドウから温度の設定値を取得し、四角形の描画サイズを算出します。<br>そして、GUI_FillRectEx 関数で四角形を描画します。 |

表 6.16　メッセージに応じた処理(cb_win_flow)

| メッセージ | 内容 |
|---|---|
| WM_CREATE | ウィンドウ生成時に 1 度だけ実行されます。<br>回転処理のために 2 つのメモリデバイスとタイマを生成します。GUI_MEMDEV_CreateFixed32 関数と WM_CreateTimer 関数を使用します。<br>生成するメモリデバイスとタイマの用途は下記のとおりです。<br><br>メモリデバイス(h_mem[0])：回転前の画像を配置<br>メモリデバイス(h_mem[1])：回転処理後の画像を配置<br>タイマ：ウィンドウ描画周期（50ms） |
| WM_PAINT | ウィンドウをクリアし、GUI_MEMDEV_WriteAt 関数でメモリデバイス(1)の画像をこのウィンドウに描画します。 |
| WM_TIMER | 指定時間後に実行されます。<br>WM_GetUserData 関数にて親ウィンドウから風量の設定値を取得し、画像の回転角度を算出します。<br>そして、GUI_MEMDEV_Rotate でメモリデバイス(1)に回転処理した画像を描画します。<br>その後、WM_InvalidateWindow 関数で画面の更新を指示し、WM_RestartTimer 関数でタイマを再開します。 |
| WM_DELETE | ウィンドウ削除時に実行されます。<br>メモリデバイスを削除します。 |

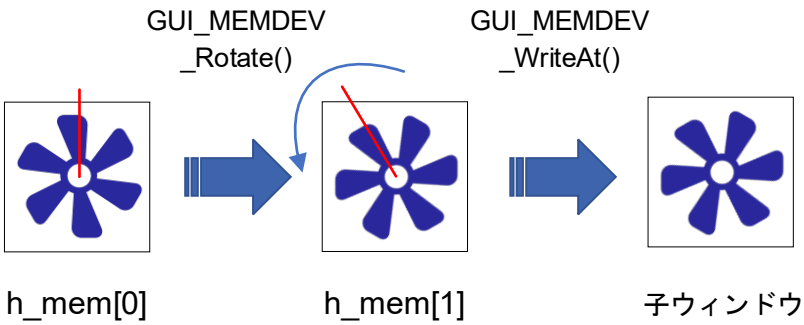



図 6.8　メモリデバイスを使った回転処理

#### 6.5.4.2　変数

空調制御処理のグローバル変数の一覧を示します。

表 6.17　グローバル変数

| 型 | 変数名 | 内容 |
|---|---|---|
| extern GUI_CONST_STORAGE GUI_BITMAP | bmhome_btn | home_btn のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmrx_logo | rx_logo のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmleft_btn_up | left_btn_up のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmleft_btn_down | left_btn_down のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmright_btn_up | right_btn_up のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmright_btn_down | right_btn_down のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmwindmill | windmill のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmtemp_meter | temp_meter のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmmode_btn_on | mode_btn_on のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmmode_btn_off | mode_btn_off のデータ変数 |
| WM_HWIN | g_window | 表示中のアクティブウィンドウのハンドル |
| static const GUI_WIDGET_CREATE_INFO | g_dialog_ac_screen [] | ダイアログ上で使用するウィジェットの管理情報 |

#### 6.5.4.3　定数

空調制御処理の定数の一覧を示します。

表 6.18　定数

| 定数名 | 設定値 | 内容 |
|---|---|---|
| TIMER_PERIOD | 50 | 風量メータのウィンドウ更新の周期(ms) |
| WINDMILL_POS_X | 230 | 風量メータのウィンドウの X 座標 |
| WINDMILL_POS_Y | 160 | 風量メータのウィンドウの Y 座標 |
| SET_VAL_MIN | 0 | 温度、風量の最小設定値 |
| TEMP_VAL_MAX | 40 | 温度の最大設定値 |
| AIR_VAL_MAX | 20 | 風量の最大設定値 |
| TEMP_METER_WIDTH | 6 | メータのウィンドウの幅 |
| TEMP_METER_HEIGHT | 33 | メータのウィンドウの高さ |
| TEMP_METER_COLOR | 0xFF9D282A | メータのカラーコード |
| ID_WINDOW_AIR_CONDITIONING | GUI_ID_USER + 0x00 | ウィンドウの管理 ID |
| ID_IMG_RX_LOGO | GUI_ID_IMAGE0 | イメージウィジェット(rx_logo)の管理 ID |
| ID_IMG_METER | GUI_ID_IMAGE1 | イメージウィジェット(temp_meter)の管理 ID |
| ID_BTN_TEMP | GUI_ID_BUTTON0 | ボタンウィジェット(Temp)の管理 ID |
| ID_BTN_AIR | GUI_ID_BUTTON1 | ボタンウィジェット(Air Flow)の管理 ID |
| ID_BTN_LEFT | GUI_ID_BUTTON2 | ボタンウィジェット(left_btn)の管理 ID |
| ID_BTN_RIGHT | GUI_ID_BUTTON3 | ボタンウィジェット(right_btn)の管理 ID |
| ID_BTN_HOME | GUI_ID_BUTTON4 | ボタンウィジェット(home_btn)の管理 ID |
| ID_TXT_TEMP | GUI_ID_TEXT0 | テキストウィジェット(“Temp”)の管理 ID |
| ID_TXT_AIR | GUI_ID_TEXT1 | テキストウィジェット(“Air Flow”)の管理 ID |
| ID_TXT_TEMP_VAL | GUI_ID_TEXT2 | テキストウィジェット(温度設定値)の管理 ID |
| ID_TXT_AIR_VAL | GUI_ID_TEXT3 | テキストウィジェット(風量設定値)の管理 ID |

#### 6.5.4.4　関数

空調制御処理の関数の一覧を示します。

表 6.19　関数

| 関数名 | 概要 |
|---|---|
| cb_win_temp | 温度メータ処理のコールバック関数 |
| cb_win_flow | 風量メータ処理のコールバック関数 |
| cb_ac_screen | 空調制御画面のコールバック関数 |
| ac_screen | 空調制御画面のウィンドウ生成 |

#### 6.5.4.5 関数仕様

空調制御処理の関数仕様を示します。

| cb_win_temp | |
|---|---|
| 概　要 | 温度メータ処理のコールバック関数 |
| ヘッダ | なし |
| 宣　言 | static void cb_win_temp(WM_MESSAGE * p_msg); |
| 説　明 | 温度メータの描画処理を制御します。 |
| 引　数 | WM_MESSAGE * p_msg　ウィンドウ情報 |
| リターン値 | なし |

| cb_win_flow | |
|---|---|
| 概　要 | 風量メータ処理のコールバック関数 |
| ヘッダ | なし |
| 宣　言 | static void cb_win_flow(WM_MESSAGE * p_msg); |
| 説　明 | 風量メータの描画処理を制御します。 |
| 引　数 | WM_MESSAGE * p_msg　ウィンドウ情報 |
| リターン値 | なし |

| cb_ac_screen | |
|---|---|
| 概　要 | 空調制御画面のコールバック関数 |
| ヘッダ | なし |
| 宣　言 | static void cb_ac_screen(WM_MESSAGE * p_msg); |
| 説　明 | 空調制御画面の制御を行います。 |
| 引　数 | WM_MESSAGE * p_msg　ウィンドウ情報 |
| リターン値 | なし |

| ac_screen | |
|---|---|
| 概　要 | 空調制御画面のウィンドウ生成 |
| ヘッダ | main.h |
| 宣　言 | void ac_screen(void) |
| 説　明 | GUI_CreateDialogBox 関数を実行して空調制御画面を生成します。 |
| 引　数 | なし |
| リターン値 | なし |

### 6.5.5　画像表示処理(image_screen.c)

#### 6.5.5.1　仕様

本画面は図 6.9 で示すパーツで構成されています。

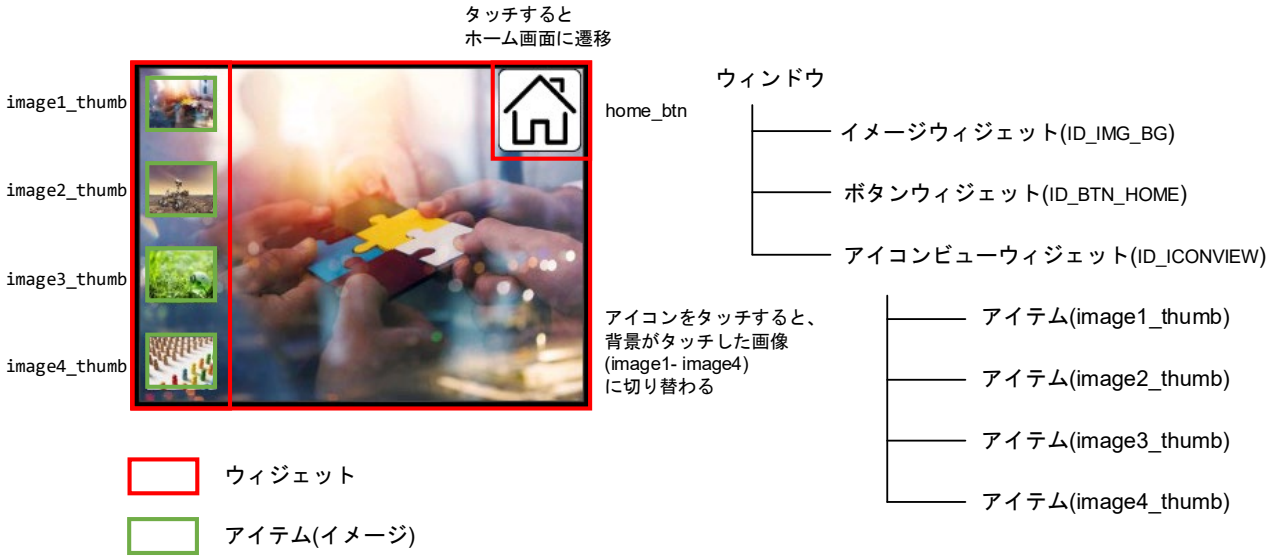



図 6.9　画像表示画面の構成

ウィンドウ内の動作はコールバック関数 cb_image_screen で管理されており、メッセージ p_msg->MsgId に応じた処理を実装しています。

表 6.20　メッセージに応じた処理

| メッセージ | 内容 |
|---|---|
| WM_CREATE | ウィンドウ生成時に 1 度だけ実行されます。<br>ウィンドウに配置する各ウィジェットを初期化（背景画像の設定やアイコンビューの設定）します。 |
| WM_PAINT | アイコンビューで選択された背景画像を IMAGE_SetBitmap 関数で描画します。 |
| WM_NOTIFY_PARENT | 各ウィジェットでイベントが発生した場合に実行されます。<br>WM_GetId(p_msg->hWinSrc)でどのボタンからのイベント通知かを判断し、ボタンに応じて処理を行います。 |
| MY_NEXT_SCREEN | ユーザ定義のメッセージです。<br>メインループからの WM_SendMessageNoPara(g_window, MY_NEXT_SCREEN)を受けて実行されます。 |
| MY_BACK_SCREEN | ユーザ定義のメッセージです。<br>メインループからの WM_SendMessageNoPara(g_window, MY_BACK_SCREEN)を受けて実行されます。 |

#### 6.5.5.2　変数

画像表示処理のグローバル変数の一覧を示します。

表 6.21　グローバル変数

| 型 | 変数名 | 内容 |
|---|---|---|
| extern GUI_CONST_STORAGE GUI_BITMAP | bmhome_btn | home_btn のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmimage1 | image1 のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmimage2 | image2 のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmimage3 | image3 のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmimage4 | image4 のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmimage1_thumb | image1_thumb(image1 のサムネイル画像)のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmimage2_thumb | image2_thumb(image2 のサムネイル画像)のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmimage3_thumb | image3_thumb(image3 のサムネイル画像)のデータ変数 |
| extern GUI_CONST_STORAGE GUI_BITMAP | bmimage4_thumb | Image4_thumb(image4 のサムネイル画像)のデータ変数 |
| extern WM_HWIN | g_window | 表示中のアクティブウィンドウのハンドル |

#### 6.5.5.3　定数

画像表示処理の定数の一覧を示します。

表 6.22　定数

| 定数名 | 設定値 | 内容 |
|---|---|---|
| ICONVIEW_WIDTH | 60 | アイコンビューの横幅 |
| ICONVIEW_HEIGHT | LCD_GetYSize() | アイコンビューの高さ(LCD の高さを取得する関数) |
| ICON_WIDTH | 50 | アイコンの横幅 |
| ICON_HEIGHT | 38 | アイコンの高さ |
| ID_IMG_BG | GUI_ID_IMAGE0 | イメージウィジェット(背景画像)の管理 ID |
| ID_BTN_HOME | GUI_ID_BUTTON0 | ボタンウィジェット(home_btn)の管理 ID |
| ID_ICONVIEW | GUI_ID_ICONVIEW0 | アイコンビューウィジェットの管理 ID |

#### 6.5.5.4　関数

画像表示処理の関数の一覧を示します。

表 6.23　関数

| 関数名 | 概要 |
|---|---|
| cb_image_screen | 画像表示画面のコールバック関数 |
| image_screen | 画像表示画面のウィンドウ生成 |

#### 6.5.5.5　関数仕様

画像表示処理の関数仕様を示します。

| cb_image_screen | |
|---|---|
| 概　要 | 画像表示画面のコールバック関数 |
| ヘッダ | なし |
| 宣　言 | static void cb_image_screen(WM_MESSAGE * p_msg); |
| 説　明 | 画像表示画面の制御を行います。 |
| 引　数 | WM_MESSAGE * p_msg　ウィンドウ情報 |
| リターン値 | なし |

| image_screen | |
|---|---|
| 概　要 | 画像表示画面のウィンドウ生成 |
| ヘッダ | main.h |
| 宣　言 | void image_screen (void); |
| 説　明 | WM_CreateWindow 関数を実行して画像表示画面を生成します。 |
| 引　数 | なし |
| リターン値 | なし |

### 6.5.6　フォント表示処理(font_screen.c)

#### 6.5.6.1　仕様

本画面は図 6.10 で示すパーツで構成されています。

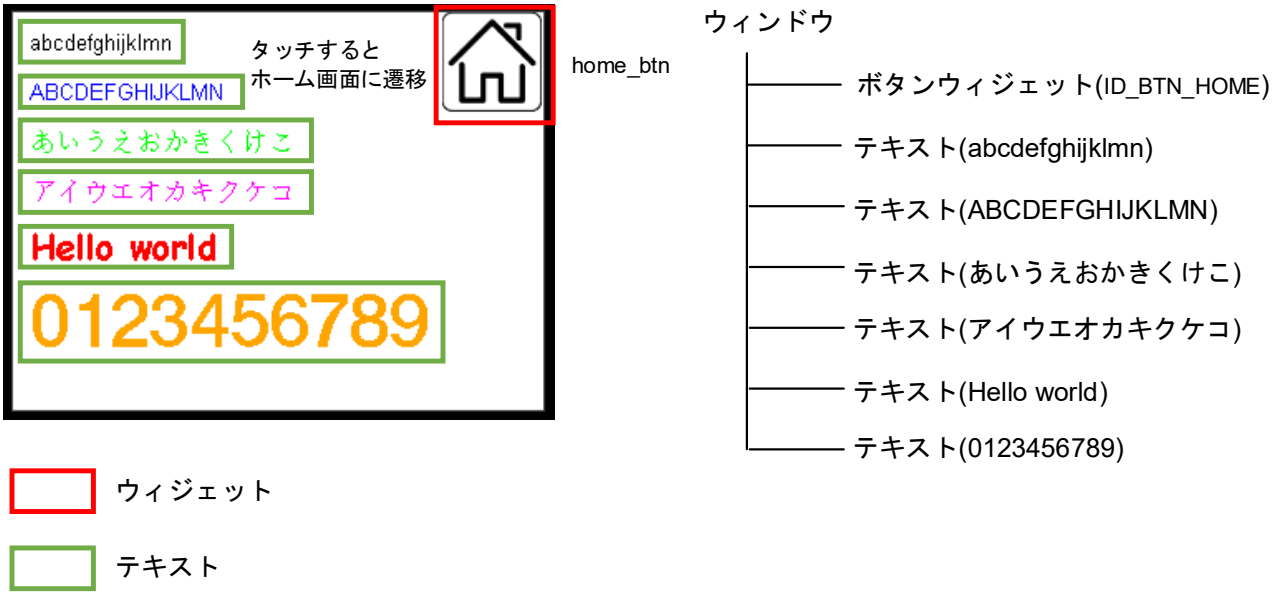



図 6.10　フォント表示画面の構成

ウィンドウ内の動作はコールバック関数 cb_font_screen で管理されており、メッセージ p_msg->MsgId に応じた処理を実装しています。

表 6.24　メッセージに応じた処理

| メッセージ | 内容 |
|---|---|
| WM_CREATE | ウィンドウ生成時に 1 度だけ実行されます。<br>ウィンドウに配置するボタンウィジェットを初期化します。 |
| WM_PAINT | 各フォントを描画します。 |
| WM_NOTIFY_PARENT | 各ウィジェットでイベントが発生した場合に実行されます。<br>WM_GetId(p_msg->hWinSrc)でホームボタン(home_btn)からのイベント通知かを判断し、ホーム画面へ遷移します。<br>遷移前には WM_DeleteWindow(p_msg->hWin)で RX ロゴ画面のダイアログを削除します。 |
| MY_NEXT_SCREEN | ユーザ定義のメッセージです。<br>メインループからの WM_SendMessageNoPara(g_window, MY_NEXT_SCREEN)を受けて実行されます。 |
| MY_BACK_SCREEN | ユーザ定義のメッセージです。<br>メインループからの WM_SendMessageNoPara(g_window, MY_BACK_SCREEN)を受けて実行されます。 |

フォント表示画面で使用しているフォントはすべて emWin に付属しているフォントです。使用しているフォントを表 6.25 に示します。

表 6.25　使用フォント一覧

| フォント | 内容 |
|---|---|
| GUI_FONT_16_ASCII | アルファベット(ASCII)フォント、16h |
| GUI_FONT_16_HK | 日本語(ひらがな&カタカナ)フォント、16h |
| GUI_FONT_COMIC24B_ASCII | コミックのようなフォント(ASCII)、24h |
| GUI_FONT_D24X32 | 数字フォント、24w×32h |

日本語フォントの表示には emWin 付属の U2C で文字コードに変換したものを使用しています。ひらがな(str_hiragana_utf8)を例にすると、先頭の“¥xe3¥x81¥x82”が“あ”の文字コードとなります。

```
static const char * str_hiragana_utf8[] =
{
    "\xe3\x81\x82\xe3\x81\x84\xe3\x81\x86\xe3\x81\x88\xe3\x81\x8a\xe3\x81\x8b\xe3\x81\x8d\xe3\x81\x8f\xe3\x81\x91\xe3\x81\x93"
};

static const char * str_katakana_utf8[] =
{
    "\xe3\x82\xa2\xe3\x82\xa4\xe3\x82\xa6\xe3\x82\xa8\xe3\x82\xaa\xe3\x82\xab\xe3\x82\xad\xe3\x82\xaf\xe3\x82\xb1\xe3\x82\xb3"
};
```

図 6.11　ひらがな＆カタカナの文字コード

### 6.5.6.2　変数

フォント表示処理のグローバル変数の一覧を示します。

表 6.26　グローバル変数

| 型 | 変数名 | 内容 |
|---|---|---|
| extern GUI_CONST_STORAGE GUI_BITMAP | bmhome_btn | home_btn のデータ変数 |
| extern WM_HWIN | g_window | 表示中のアクティブウィンドウのハンドル |
| static const char * | str_hiragana_utf8[] | ひらがな(あいうえおかきくけこ)の文字コード |
| static const char * | str_katakana_utf8[] | カタカナ(アイウエオカキクケコ)の文字コード |

### 6.5.6.3　定数

フォント表示処理の定数の一覧を示します。

表 6.27　定数

| 定数名 | 設定値 | 内容 |
|---|---|---|
| ID_BTN_HOME | GUI_ID_BUTTON0 | ボタンウィジェット(home_btn)の管理 ID |

### 6.5.6.4　関数

フォント表示処理の関数の一覧を示します。

表 6.28　関数

| 関数名 | 概要 |
|---|---|
| cb_font_screen | フォント表示画面のコールバック関数 |
| font_screen | フォント表示画面のウィンドウ生成 |

#### 6.5.6.5 関数仕様

フォント表示処理の関数仕様を示します。

| cb_font_screen | |
|---|---|
| 概　要 | フォント表示画面のコールバック関数 |
| ヘッダ | なし |
| 宣　言 | static void cb_font_screen(WM_MESSAGE * p_msg); |
| 説　明 | フォント表示画面の制御を行います。 |
| 引　数 | WM_MESSAGE * p_msg　ウィンドウ情報 |
| リターン値 | なし |

| font_screen | |
|---|---|
| 概　要 | フォント表示画面のウィンドウ生成 |
| ヘッダ | main.h |
| 宣　言 | void font_screen (void); |
| 説　明 | WM_CreateWindow 関数を実行してフォント表示画面を生成します。 |
| 引　数 | なし |
| リターン値 | なし |

## 6.6　リソース

本サンプルプログラムのリソースを以下に示します。

下記の条件における参考値です。

- C/C++ Compiler Package for RX Family V3.05.00
- 最適化レベル：レベル 2

### 6.6.1　全体リソース

表 6.29　サンプルプログラム全体のリソース

| 種類 | 使用量（バイト） | 備考 |
|---|---|---|
| RAM | 269,155 | このうち、emWin のワークサイズは 250K バイト確保しています。 |
| ROM | 1,055,073 | |
| スタック | 1,355 | |

### 6.6.2　各画面のリソース

各画面のリソースを以下に示します。なお、FIT モジュールやメイン処理関連のリソース、emWin で使用するキャッシュ(150K バイト)は含みません。

#### 6.6.2.1　ホーム画面処理

表 6.30　ホーム画面のリソース

| 種類 | 使用量（バイト） | 詳細 |
|---|---|---|
| RAM | 1,269 | emWin ワーク(1,269 バイト) |
| ROM | 81,769 | プログラム、定数、使用画像 |

#### 6.6.2.2　RX ロゴ表示処理

表 6.31　RX ロゴ表示処理のリソース

| 種類 | 使用量（バイト） | 詳細 |
|---|---|---|
| RAM | 521 | 変数(28 バイト)<br>+ emWin ワーク(493 バイト) |
| ROM | 21,503 | プログラム、定数、使用画像 |

#### 6.6.2.3　空調制御処理

表 6.32　空調制御処理のリソース

| 種類 | 使用量（バイト） | 詳細 |
|---|---|---|
| RAM | 42,568 | 変数(39 バイト)<br>+ emWin ワーク(42,529 バイト)<br>(このうち、メモリデバイスで 39,200 バイト使用) |
| ROM | 88,956 | プログラム、定数、使用画像 |

#### 6.6.2.4　画像表示処理

表 6.33　画像表示処理のリソース

| 種類 | 使用量（バイト） | 詳細 |
|---|---|---|
| RAM | 1,181 | 変数(8 バイト)<br>+ emWin ワーク(1,173 バイト) |
| ROM | 641,048 | プログラム、定数、使用画像 |

#### 6.6.2.5　フォント表示処理

表 6.34　フォント表示処理のリソース

| 種類 | 使用量（バイト） | 詳細 |
|---|---|---|
| RAM | 397 | 変数(8 バイト)<br>+ emWin ワーク(389 バイト) |
| ROM | 11,428 | プログラム、定数、使用画像 |

## 6.7　使用ツール

### 6.7.1　QE for Display

本サンプルコードでは QE for Display を使用しています。QE for Display はルネサス製 RX マイコンに対応する統合開発環境 e² studio 用のプラグインです。表示機器を搭載した組み込みシステム開発において各種ツールや FIT モジュールと連携してグラフィカルな I/F で設定することができ、効率的な開発をサポートします。

詳細は下記 URL をご参照ください。

https://www.renesas.com/jp/ja/software-tool/qe-display-development-assistance-tool-display-applications

### 6.7.2　QE for Capacitive Touch

本サンプルコードでは QE for Capacitive Touch を使用しています。QE for Capacitive Touch はルネサス製 RX マイコンに対応する統合開発環境 e² studio 用のプラグインと、CS+や IAR EWRX で連携できる単体版（ダウンロードパッケージ内に同梱されています）があります。 静電容量式タッチ機能を使用した組み込みシステム開発において、タッチインタフェースの初期設定や感度のチューニングを容易に行うことができ、効率的な開発をサポートします。

詳細は下記 URL をご参照ください。

https://www.renesas.com/jp/ja/software-tool/qe-capacitive-touch-development-assistance-tool-capacitive-touch-sensors

# 7. 画面の更新速度に関する補足

2.3 画面の更新速度について で説明した内容は emWin ライブラリの使用方法に基づく方法ですが、ハードウェアの観点から以下の内容も合わせてご検討ください。

## 7.1 通信ボーレート

通信ボーレートが高いほど画面の更新速度を上げることができますが、ボーレートの設定には制限がある場合があります。RX671 の場合、LCD との通信インタフェースで使用している RSPI は最大 40MHz に設定可能ですが、システムクロック(ICLK)や周辺モジュールクロック A(PCLKA)を 80MHz にする必要があります。このような最大ボーレートとのトレードオフで最大動作周波数を下げる場合、最大ボーレートを設定してもユーザアプリケーションや emWin の処理負荷によっては画面の更新速度が遅くなることがあります。ユーザアプリケーションと emWin の処理負荷を考慮し、適切な動作周波数とボーレートを検討してください。

## 7.2 DMA 転送機能の選択

emWin FIT モジュールでは、LCD との通信インタフェースに RSPI または SCI(簡易 SPI モード)を用いる場合、DMA 転送機能を使用して通信間のオーバヘッドを削減し、通信効率を上げることができます。RX ファミリには DMAC と DTC の 2 種類、または DTC の 1 種類が搭載されており、転送速度は DTC よりも DMAC の方が速くなります。そのため、通常では DMAC を使用されることを推奨します。

## 7.3 コンパイルオプション

画面の更新速度を上げるためにはコンパイルオプションによるコード効率の向上が挙げられます。最適化レベルを最大にし、実行性能重視(-speed オプション)を指定することで、通信のオーバヘッドをより短縮することができます。

# 8. プロジェクトの設定情報

## 8.1 Smart Configurator

サンプルプログラムで使用している FIT モジュールおよび e² studio の Smart Configurator の設定を下記に示します。Smart Configurator の設定における各表の項目、設定内容は設定画面の表記で記載しています。各 FIT モジュールの詳細は、各 FIT モジュールのドキュメントを参照してください。

表 8.1 BSP モジュールの設定

| 分類 | 項目 | 設定、説明 |
|---|---|---|
| スマート・コンフィグレータ >> クロック | | プロジェクト作成時にボード選択で「EK-RX671」を選択した場合のデフォルト設定です。 |
| | VCC 設定 | 3.3(V) |
| | メインクロック設定 | 動作：チェックを入れる<br>発振源：発振子<br>周波数：24 (MHz)<br>発振安定時間：9980 (μs) |
| | PLL 回路設定 | 分周比：x1<br>逓倍比：x10.0 |
| | サブクロック発振器設定 | 動作：チェックを入れる<br>周波数：32.768 (kHz)<br>発振器ドライブ能力：標準 CL<br>発振安定時間：2000 (ms) |
| | HOCO クロック設定 | 停止：チェックを外す |
| | LOCO クロック設定 | 停止：チェックを外す |
| | システムクロック設定 | クロックソース：PLL<br>システムクロック(ICLK)：x1/2　120 (MHz)<br>周辺モジュールクロック(PCLKA)：x1/2　120 (MHz)<br>周辺モジュールクロック(PCLKB)：x1/4　60 (MHz)<br>周辺モジュールクロック(PCLKC)：x1/4　60 (MHz)<br>周辺モジュールクロック(PCLKD)：x1/4　60 (MHz)<br>FlashIF クロック(FCLK)：x1/4　60 (MHz) |
| | IWDT 専用クロック設定 | 停止：チェックを外す |

表 8.2 GPIO モジュールの設定

| 分類 | 項目 | 設定、説明 |
|---|---|---|
| スマート・コンフィグレータ >> コンポーネント >> r_gpio_rx | | デフォルトの設定 (変更なし) |

表 8.3 CMT モジュールの設定

| 分類 | 項目 | 設定、説明 |
|---|---|---|
| スマート・コンフィグレータ >> コンポーネント >> r_cmt_rx | | デフォルトの設定 (変更なし) |

表 8.4 BYTEQ モジュールの設定

| 分類 | 項目 | 設定、説明 |
|---|---|---|
| スマート・コンフィグレータ >> コンポーネント >> r_byteq | | デフォルトの設定 (変更なし) |

表 8.5 DMAC モジュールの設定

| 分類 | 項目 | 設定、説明 |
|---|---|---|
| スマート・コンフィグレータ >> コンポーネント >> r_dmaca_rx | | デフォルトの設定 (変更なし) |

表 8.6 RSPI モジュールの設定

| 分類 | 項目 | 設定、説明 |
|---|---|---|
| スマート・コンフィグレータ >> コンポーネント >> r_rspi_rx | | 下記の変更以外はデフォルトの設定。 |
| Configurations | | |
| | Dummy data of reception<br>(RSPI_CFG_DUMMY_TXDATA) | 0x00 に変更 |
| リソース >> RSPI | | |
| | RSPI0 | チェックを入れる |
| | RSPCKA | 使用する：チェックを入れる |
| | MOSIA 端子 | 使用する：チェックを入れる |
| | MISOA 端子 | 使用する：チェックを入れる |
| スマート・コンフィグレータ >> 端子 >> シリアルペリフェラルインタフェース >> RSPI0 | | 下記の設定以外はチェックを外す。 |
| | RSPCKA | 使用する：チェックを入れる<br>端子割り当て：PC5 に設定 |
| | MOSIA | 使用する：チェックを入れる<br>端子割り当て：PC6 に設定 |
| | MISOA | 使用する：チェックを入れる<br>端子割り当て：PC7 に設定 |

表 8.7 SCI モジュールの設定

| 分類 | 項目 | 設定、説明 |
|---|---|---|
| スマート・コンフィグレータ >> コンポーネント >> r_sci_rx | | 下記の変更以外はデフォルトの設定。 |
| Configurations | | |
| | Use ASYNC mode<br>(SCI_CFG_ASYNC_INCLUDED) | Not (0)に変更 |
| | Use SSPI mode<br>(SCI_CFG_SSPI_INCLUDED) | Include (1)に変更 |
| | Byte value to transmit while clocking in data in SSPI mode<br>(SCI_CFG_DUMMY_TX_BYTE) | 0x00 に変更 |
| リソース >> SCI | | |
| | SCI1 | チェックを入れる |
| | SCK1 端子 | 使用する：チェックを入れる |
| | RXD1/SMISO1/SSCL1 端子 | 使用する：チェックを入れる |
| | TXD1/SMOSI1/SSDA1 端子 | 使用する：チェックを入れる |
| スマート・コンフィグレータ >> 端子 >> シリアルコミュニケーションインタフェース >> SCI1 | | 下記の設定以外はチェックを外す。 |
| | SCK1 | 使用する：チェックを入れる<br>端子割り当て：P17 に設定 |
| | RXD1 | 使用する：チェックを入れる<br>端子割り当て：P15 に設定 |
| | TXD1 | 使用する：チェックを入れる<br>端子割り当て：P16 に設定 |

表 8.8 Touch QE モジュールの設定

| 分類 | 項目 | 設定、説明 |
|---|---|---|
| スマート・コンフィグレータ >> コンポーネント >> rm_touch_qe | | デフォルトの設定 (変更なし) |

表 8.9 CTSU モジュールの設定

| 分類 | 項目 | 設定、説明 |
|---|---|---|
| スマート・コンフィグレータ >> コンポーネント >> r_ctsu_qe | | 下記の変更以外はデフォルトの設定。 |
| リソース >> CTSU | | |
| | TSCAP 端子 | 使用する |
| | TS1 端子 | 使用する |
| | TS5 端子 | 使用する |

表 8.10 emWin モジュールの設定

| 分類 | 項目 | 設定、説明 |
|---|---|---|
| スマート・コンフィグレータ >> コンポーネント >>　r_emwin_rx | | QE for Display を使用しない場合、以下の設定を行ってください。<br>下記の変更以外はデフォルトの設定。 |
| Configurations | | |
| | Work area size for GUI<br>(EMWIN_GUI_NUM_BYTES) | 256000 に設定<br>(デフォルト値 100KB にキャッシュ機能で必要なサイズ 150KB を加算したサイズです) |
| | Horizontal LCD size<br>(EMWIN_XSIZE_PHYS) | 240 に設定 |
| | Vertical LCD size<br>(EMWIN_YSIZE_PHYS) | 320 に設定 |
| | LCD orientation<br>(EMWIN_DISPLAY_ORIENTATION) | ORIENTATION_CCW に設定 |
| | LCD interface<br>(EMWIN_LCD_IF) | LCD_IF_RSPI |
| | Select LCD Driver IC<br>(EMWIN_LCD_DRIVER_IC) | LCD_DRV_IC_ILI9341 |
| | Communication baud rate of LCD interface<br>(EMWIN_LCD_BAUDRATE) | 30000000 |
| | Use or unuse display cache<br>(EMWIN_GUI_USE_CACHE) | Use：チェックを入れる |
| | Display Signal Pin<br>(EMWIN_DISP_SIGNAL_PIN) | GPIO_PORT_5_PIN_0 |
| | Backlight Pin<br>(EMWIN_BACKLIGHT_PIN) | GPIO_PORT_5_PIN_2 |
| | Data/Command Pin<br>(EMWIN_DATA_CMD_PIN) | GPIO_PORT_5_PIN_1 |
| | Chip Select Pin<br>(EMWIN_LCD_CS_PIN) | GPIO_PORT_C_PIN_3 |
| | Touch interface<br>(EMWIN_TOUCH_IF) | TOUCH_IF_SCI_SPI |
| | Touch interface channel number<br>(EMWIN_TOUCH_IF_NUMBER) | 1 に設定 |
| | Communication baud rate of touch interface<br>(EMWIN_TOUCH_BAUDRATE) | 2000000 |
| | Use Touch IC Reset Pin<br>(EMWIN_USE_TOUCH_IC_RESET_PIN) | Not use Touch IC Reset Pin：チェックを外す |
| | Touch Chip Select Pin<br>(EMWIN_TOUCH_CS_PIN) | GPIO_PORT_1_PIN_4 |

## 8.2　QE for Display

QE for Display の設定を図 8.1 に示します。QE for Display を使用する際は下記の設定にしてください。

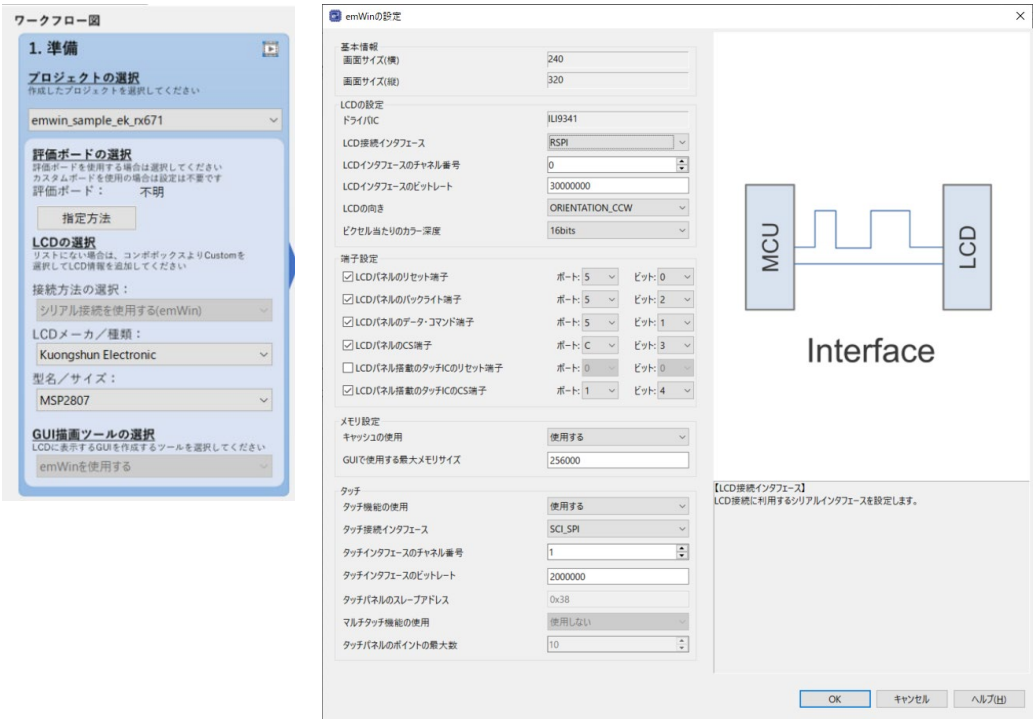



図 8.1　QE for Display の設定

## 8.3　QE for Capacitive Touch

QE for Capacitive Touch の設定を図 8.2 に示します。調整結果は動作確認したボードによる参考データです。使用されるボードでタッチキーが反応しないときはタッチキーの再調整を行ってください。

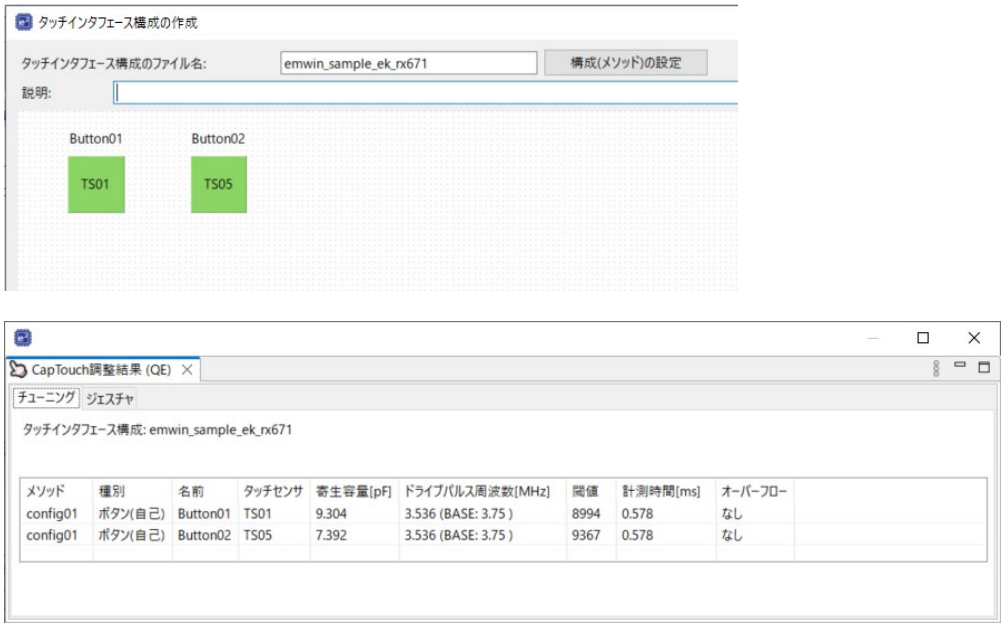



図 8.2　QE for Capacitive Touch の設定

# 9. 参考ドキュメント

ユーザーズマニュアル：ソフトウェア

・emWin Graphic Library with Graphical User Interface User Guide & Reference Manual
（https://www.segger.com/downloads/emwin/UM03001）

・RX スマート・コンフィグレータ ユーザーガイド: e$^2$ studio 編(R20AN0451)

ユーザーズマニュアル：ハードウェア

・RX671 グループ ユーザーズマニュアル ハードウェア編 (R01UH0899)

・EK-RX671 ユーザーズマニュアル (R20UT5234)

・EK-RX671 CPU ボード回路図 (R20UT5233)

（各デバイスの最新版をルネサス エレクトロニクスホームページから入手してください）

テクニカルアップデート／テクニカルニュース

（最新の情報をルネサスエレクトロニクスホームページから入手してください）

ユーザーズマニュアル：開発環境

RX ファミリ CC-RX コンパイラ ユーザーズマニュアル（R20UT3248）

（最新版をルネサス エレクトロニクスホームページから入手してください）

## 改訂記録

| Rev. | 発行日 | 改訂内容 | |
|---|---|---|---|
| | | ページ | ポイント |
| 1.00 | June.30.23 | - | 初版 |
| | | | |

# 製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、$V_{IL}$（Max.）から $V_{IH}$（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、$V_{IL}$（Max.）から $V_{IH}$（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

# ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

## 本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

## お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

## 商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。