RX Family

GUI Sample Program using Serial LCD and emWin Library

Introduction

In this application note, we describe how to develop a GUI application using serial TFT-LCD display with Segger's graphics library emWin.

The sample program uses the basic functions provided by emWin, such as window manager, widgets, memory devices, animations, etc., which are also explained in this application note. Please note that this sample program does not use the GUI design tool AppWizard.

Note:

- Running this sample program requires separate SPI-compatible TFT-LCD display and wiring components to connect with the target board EK-RX671
- Because the link size of this sample program exceeds 128K bytes, it cannot be built if the free evaluation version's trial period of CC-RX has expired. If you have purchased the compiler, make sure to register the license key to the License Manager.

Target Device

RX671 Group

If you apply this application note to another microcontroller, please modify it according to the specifications of that microcontroller and thoroughly evaluate it.

Contents

1. Overview

This sample program consists of the following screens. The LCD display is equipped with touchscreen functions, which could be used to control the operation through buttons displayed. In addition, transitions between screens can be controlled by the touch keys designed on the target board.

- Home Screen
- RX Logo Display
- Air Conditioning Control
- Image Display
- Font Display

1.1 Home Screen

This is the first screen displayed when this sample program is run. It allows the transition to the other four screens. The main functions of emWin being used here are the Window Manager, Dialog, and Button Widget.

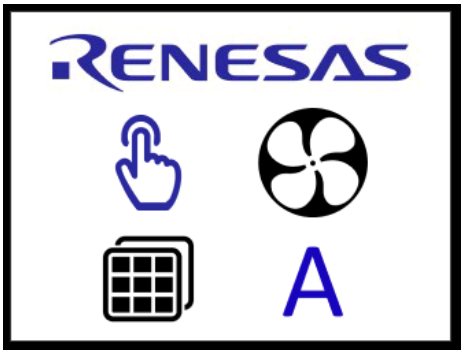


Figure 1.1 Home Screen

1.2 RX Logo Display

The RX logo, which is displayed in the center of the screen, can be moved freely around the screen by dragging it. When the drag is released, the RX logo returns to the center of the screen. The main functions of emWin being used here are the Window Manager, Button Widget, and Animation.

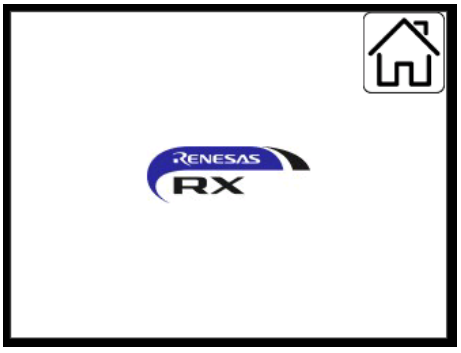


Figure 1.2 RX Logo Display

1.3 Air Conditioning Control

This screen resembles the control of the temperature and airflow. By touching the left and right arrows when either "Temp" or "Air Flow" is selected, you can change each setting value. The state of the temperature meter and the air flow meter changes according to the set value. The main features of emWin being used here are the Window Manager, Dialog, Button Widget, Timer, and Memory Devices.

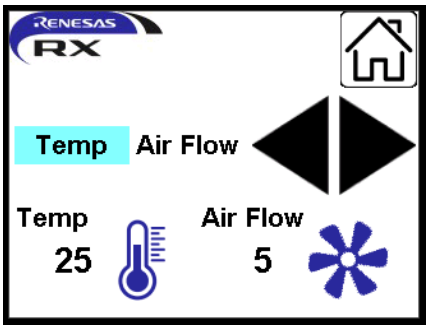


Figure 1.3 Air Conditioning Control

1.4 Image Display

This screen displays images in full screen. Image is switched by touching the thumbnail on the left. The main features of emWin being used here are the Window Manager and Iconview Widget.

Figure 1.4 Image Display

1.5 Font Display

This displays several fonts included in the emWin library. The main features of emWin being used are the Window Manager and Font Display.

Figure 1.5 Font Display

1.6 Screen Switch by Touch Keys

You can switch between screens as shown below by touch keys on the target board.

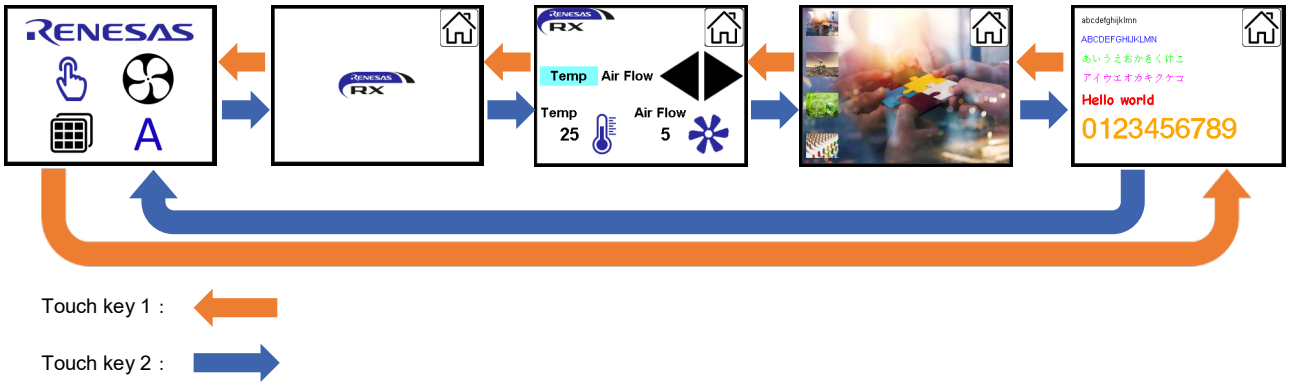


Figure 1.6 Screen Switch Sequence

2. Main Functions of emWin

2.1 Display Driver (Flex Color Driver) and Color Mode

emWin provides a variety of display drivers compatible with various types of LCD controllers and color modes for displaying various colors. This sample program uses the display driver named "Flex Color" which is compatible with the connected LCD controller (ILI9341), and the color mode defined as "GUICC_565" for 16-bit RGB.

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

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

2.2 Memory Management

emWin is a lightweight graphics library, but the amount of memory used can vary greatly depending on the images and designs used, making memory management extremely important. In particular, the RAM used as a workspace for functions such as cache and memory devices, requires careful consideration.

The workspace used in emWin is specified by the configuration parameter "EMWIN_GUI_NUM_BYTES" in the emWin FIT module. It is recommended to set this with ample room for user applications.

2.3 Screen Update Speed

When using an LCD with a serial interface, the screen update speed is significantly affected by the screen design and communication speed. In particular, the extensive use of windows and widgets, or slow data communication speeds, can reduce the overall screen update performance. As a result, flickering may occur, or screen updates may be visually perceptible. emWin provides the following features to mitigate these issues. Also, please refer to Chapter 7 for additional information on screen update speeds from a hardware perspective.

2.3.1 Cache

Normal screen updates are performed for each window or widget, resulting in repeated data transmissions to the LCD. By using a cache, all screen update data can be written out to the cache and then sent to the LCD in one go. This reduces access to the LCD and can increase the screen update speed. However, a cache requires memory for a full screen, so sufficient RAM capacity is necessary.

RAM required for cache: Display width (px) x height (px) x bytes per pixel (bpp)

To use the cache, set "EMWIN_GUI_USE_CACHE" to "1" in the configuration settings of the emWin FIT module, and set the size required for the cache to "EMWIN_GUI_NUM_BYTES".

In this sample program, the cache is enabled. The RAM required for the cache is 240px x 320px x 2bpp = 150KB, which can be accommodated within the built-in RAM (384KB) of the RX671.

2.3.2 Memory Devices

Memory devices are buffers that can be used in the following ways:

1. Buffer for image data read from an external device.
2. Buffer used for operations such as rotating images, changing color, α value, blurring and blending.
3. Buffer for screens that take time to draw (used in the same way as a cache)

Memory devices are dynamic buffers, and if multiple memory devices are enabled simultaneously, RAM capacity for that number of enabled memory devices is required. These are allocated from the configuration parameter "EMWIN_GUI_NUM_BYTES" of the emWin FIT module.

There are several types of API functions to create memory devices, and you need to select the appropriate API depending on the purpose. For example, the GUI_MEMDEV_CreateFixed32 function must be used to perform rotation processing.

Moreover, the MCUs which do not have enough memory capacity for cache function can instead use memory devices to increase the screen update speed. By dividing the screen into multiple sections and writing to memory devices of an allocable size, the performance will be higher than when not using a cache. To apply a memory device to a screen, refer to 2.5 Windows.

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

In this sample program, memory device is used in the 1.3 Air Conditioning Control screen.

2.4 Window Manager

The emWin window manager is a feature managing the screen in units called windows.

The screen content is divided into multiple parts (windows) and managed, and the performance is improved by independently drawing each part.

The operation of created windows is managed by callback functions corresponding to events of windows creation (WM_CREATE) and drawing processing (WM_PAINT), for example.

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

In order to execute events such as window drawing and touch panel input, it is necessary to execute the GUI_Exec function in the main loop. When the GUI_Exec function is executed, the callback functions of each window are executed.

You can use the GUI_Delay function instead of the GUI_Exec function. The GUI_Delay function waits for the time specified in the argument (in ms units). During the wait, the GUI_Exec function is called at least once within the GUI_Delay function. Also, the waiting time specified in the argument of the GUI_Delay function is a minimum period, and when the processing time of the callback function by the GUI_Exec function is long, it may be over the specified time. Therefore, please use the GUI_Delay function as a wait function for processes that are expected to take less time than the specified waiting period.

2.5 Windows

There are different types and statuses of windows.

<Types>

- Child/Grandchild Window

These are windows that are defined relative to another window, referred to as the parent. If the parent window moves, its child windows move accordingly. Child windows are always contained within the parent window. A new child window (grandchild window) can also be created within a child window.

- Parent Window

This is the window that serves as the parent for the child windows.

- Desktop Window

This is the bottommost window that covers the entire screen. It becomes the default (active) window when no other windows are defined. Therefore, all windows are descendants of this desktop window. If multiple layers are defined, a desktop window is created for each layer.

<Status>

- Active Window (Current Window)

This is the window currently being used for drawing operations. It may not necessarily be the topmost window.

- Show/Hide Window

These are windows that are visible/invisible. You can specify visibility/invisibility when creating a window, or you can toggle visibility/invisibility using API functions.

Basically, a window can be created using the following API functions.

```
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

These indicate the position of the window. In the case of a child window, it's the position relative to the parent window.

• width, height

These indicate the size of the window.

• hParent, (WM_CreateWindowAsChild function)

This specifies the parent window. If you specify NULL, the desktop window becomes the parent window.

• Style

Flags ("WM_CF_xxxx") are specified. Multiple flags can be specified by OR operand. If WM_CF_SHOW is included, the window will be displayed from the start. Also, specifying WM_CF_MEMDEV will use a memory device for window drawing. For details on the flags, please refer to the URL below.

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

• cb

This is a pointer to a callback function that defines the behavior of the window. Within this callback function, you write processes according to events (messages) such as WM_PAINT. For details on the messages, please refer to the URL below.

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

Below is an example of a callback function that fills the window area with red.

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

The pMsg argument of the callback function could specify various information. For example, the above pMsg->MsgId stores the event message, and pMsg->hWin allows access to the properties of the window to which the callback function belongs.

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

• NumExtraBytes

It specifies the number of bytes to allocate for user data. For more details, please refer to the description of the WM_SetUserData function.

2.6 Dialogs

Dialogs are a function that allows collective management of windows or parts (widgets) within a window by defining their placement in a structure. This feature is particularly useful for organizing multiple widgets within a window or when the placement will not change.

Basic dialogs can be created using the following API functions.

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

• paWidget

It specifies a pointer to a structure variable. The variable of GUI_WIDGET_CREATE_INFO is defined and each widgets, including windows, are configured.

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

It indicates the total number of widgets contained in the dialog. Users can directly set a numerical value or can use GUI_COUNTOF("the variable of structure") to dynamically accommodate the size of the structure variable.

• cb

It points to a callback function that defines the behavior of the dialog. It is similar to a normal window, but the initialization message and processing content may differ.

```
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);
  }
}
```

In Dialogs, the WM_INIT_DIALOG message is executed only once during initialization. It is used to configure the widgets within the dialog.

• hParent

It specifies the parent window. If NULL is specified, the desktop window becomes the parent window.

•x0, y0

It indicates the position of the dialog. It is relative to the parent window.

The screens designed using the GUI Builder, which is provided with emWin, are composed of these dialogs. For more details on Dialogs, please refer to the URL below.
https://www.segger.com/doc/UM03001_emWin.html#Dialogs

In this sample program, dialogs are used in screens 1.1 Home Screen and 1.3 Air Conditioning Control

2.7 Images

emWin supports popular image formats such as PNG, JPG, and GIF.

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

All the images used in this sample program are converted to C source code using the Bitmap Converter provided with emWin. This tool allows you to convert images to the emWin-specific format (bitmap). By converting images to emWin bitmaps in advance, processing time can be reduced for images that are used only within the system.

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

The images used in this sample program are output in the following formats. This output format is efficiently processed when matched with the display driver (Flex Color driver) described in section 2.1 Display Drivers and Color Mode (GUICC_565).

· High color (565) ... 16-bit BGR565 image

· High color with Alpha (565) ... 16-bit BGR565 image with transparency

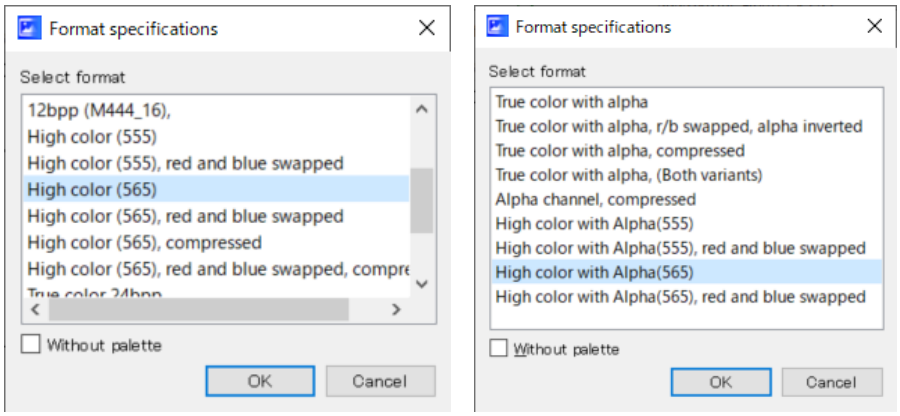


Figure 2.1 Format of Images

2.8 Fonts

emWin provides a mechanism to use a wide variety of fonts. It is possible to use fonts included in the library or import fonts in SIS or CBF formats from external memory.

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

The Font Converter provided with emWin allows to convert fonts installed on a Windows PC into emWin format fonts. Users can specify whether anti-aliasing is enabled or only the characters to be used in the application is applied. However, please note that font licensing is not included, and users must obtain the necessary licenses.

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

To Japanese fonts, emWin supports SJIS encoding and other formats, but it is recommended to use UTF-8 whenever possible. The U2C tool provided with emWin allows you to convert UTF-8 text data into character code. It can be used in the same way for languages other than Japanese. When saving UTF-8 text files, please save them with the "BOM (Byte Order Mark)".

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

This sample program uses only the fonts included with emWin.

2.9 Animation

emWin offers animation mechanism that dividing a given period of time 32768 by default and counting from 0 to 32768. This allows to animate the coordinates, color codes, and other values of animation objects.

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

An animation is created by calling the following function:

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

•Period

The duration of the animation in milliseconds.

•MinTimePerSlice

The execution period of the animation processing (callback function) in milliseconds.

• pVoid

A pointer to user-defined data.

• pfSlice

A pointer to the callback function. A NULL could be set. This function is executed before and after the callback function of the animation item. If multiple animation items are active simultaneously, it is executed before the first animation item and after the last animation item. In this case, the values GUI_ANIM_START("0") and GUI_ANIM_END("2") are passed to the first argument.

After creating an animation, users need to add animation items.

```
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

The handle of the animation.

• ts, te

The start and end times of this item within the animation's duration.

• pVoid

A pointer to user-defined data.

• pfAnim

A pointer to the callback function of the animation item. This callback function is executed when the time matches the period set by MinTimePerSlice in the GUI_ANIM_Create function, during period from the start time to the end time of this item.

After setting the animation items, the animation starts by executing the GUI_ANIM_StartEx function.

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

• hAnim

The handle of the animation.

• NumLoops

Specifies the number of times the animation is executed. Set to "-1" for infinitive execution.

• pfOnDelete

A pointer to the callback function called when the animation is deleted. A NULL could be set.

In this sample program, animation is used in 1.2 RX Logo Display.

2.10 Notes

To use anti-aliasing and semi-transparent bitmaps function, it is necessary to obtain the color information of the background and foreground pixels that are overlapped. Therefore, the use of a cache or memory device is required.

2.11 Additional Information

For more information about emWin, please refer to the following URLs from Segger.

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

3. Evaluation Environment

This sample program has been evaluated under the following conditions.

Table 3.1 Evaluation Environment

Items	Description
Target MCU	R5F5671EHDFB (RX671 Group)
Operating Frequency	• Main clock: 24MHz • PLL:240MHz (Main clock divided by 1 and multiplicated by 10) • System clock (ICLK): 120MHz (PLL divided by 2) • Peripheral module clock A(PCLKA): 120MHz (PLL divided by 2) • Peripheral module clock B(PCLKB): 60MHz (PLL divided by 4) • Peripheral module clock C(PCLKD): 60MHz (PLL divided by 4) • Peripheral module clock D(PCLKD): 60MHz (PLL divided by 4) • FlashIF clock (FCLK): 60MHz (PLL divided by 4)
Operating Power	3.3V
Integrated Development Environment	Renesas Electronics e^2 studio Version 2023-04
C Compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.05.00
	Compiler Option -lang = c99
iodefine.h version	V1.00
Endian	Little endian
Operation Mode	Single chip mode
Processor Mode	Supervisor mode
Sample Project Versioon	Version 1.00
Emulator	Onboard Emulator (E2 Lite Emulator)
Target Board	EK-RX671 (Part Number: RTK5EK6710S00001BE)
Target LCD	MSP2807 (Manufacturer: Kuongshun Electronic Limited)

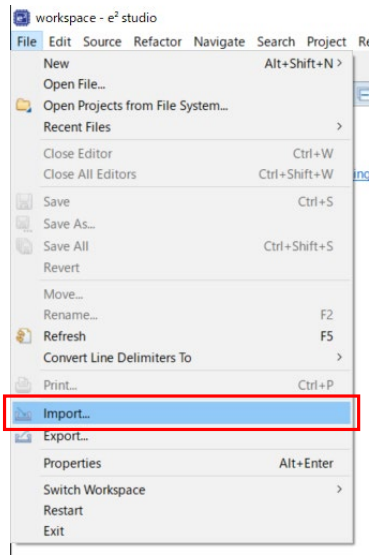
4. How to Run the Sample Project

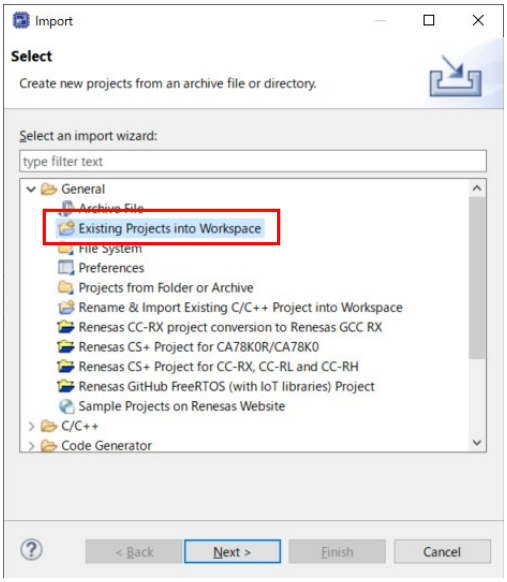
4.1 Prepare the Hardware

In order to run this sample code, you will need to set up the jumpers on the EK-RX671 and connect it to an LCD (MSP2807). Please prepare the necessary parts in advance. Refer to section 5 Hardware Description for information on how to set up the jumpers and connect the devices.

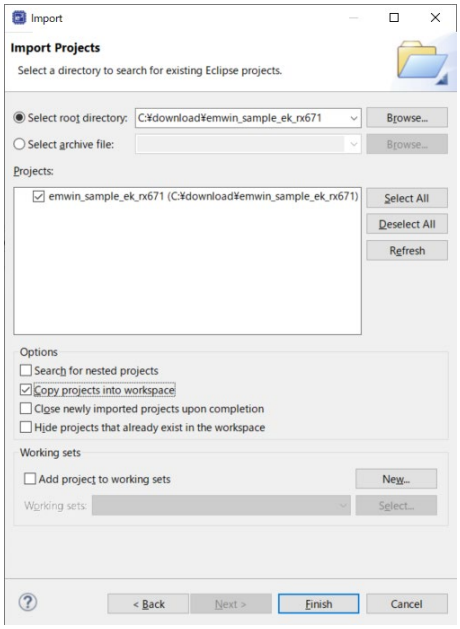
4.2 Import the Project

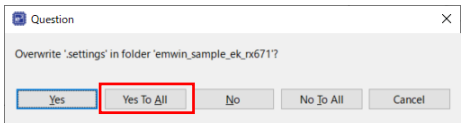
Go to "File" → "Import" to open the "Import Select" window. Select "Existing Projects into Workspace" from the "General" category, then click "Next"



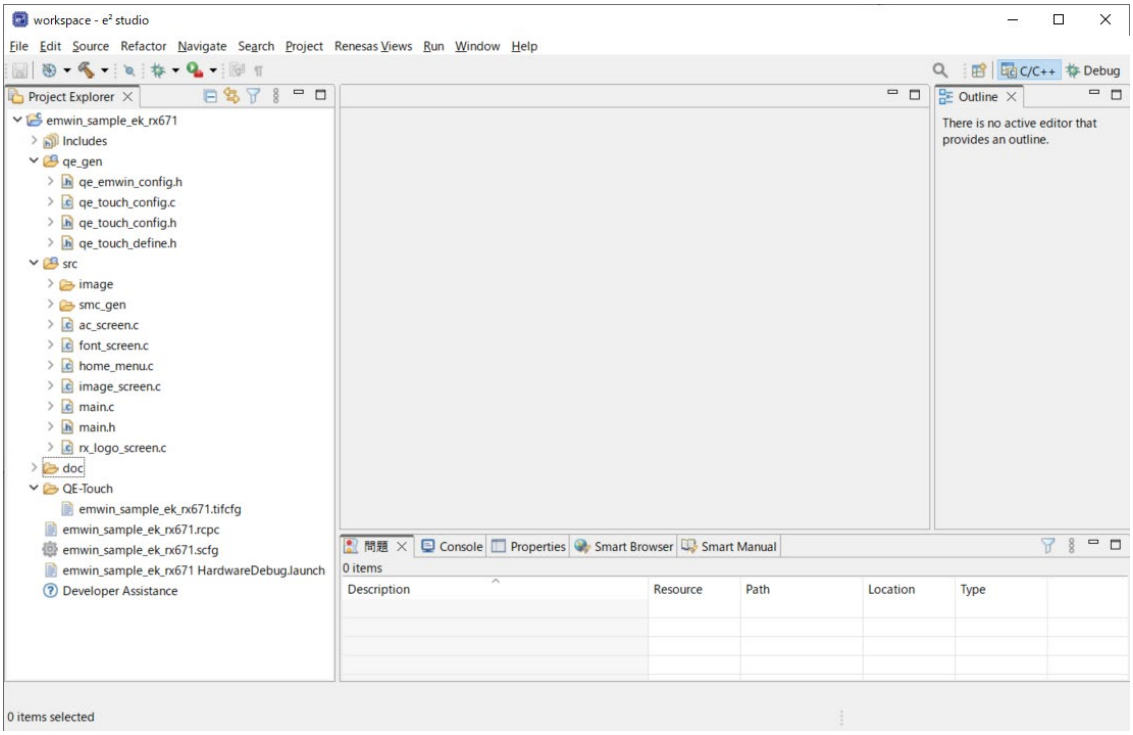


In the "Import Projects" window that appears, click "Browse" under "Select root directory" and choose the folder of the downloaded sample project. Once the project is recognized, check the "Copy projects into workspace" option and click "Finish" If a "Question" window appears, click "Yes to All"

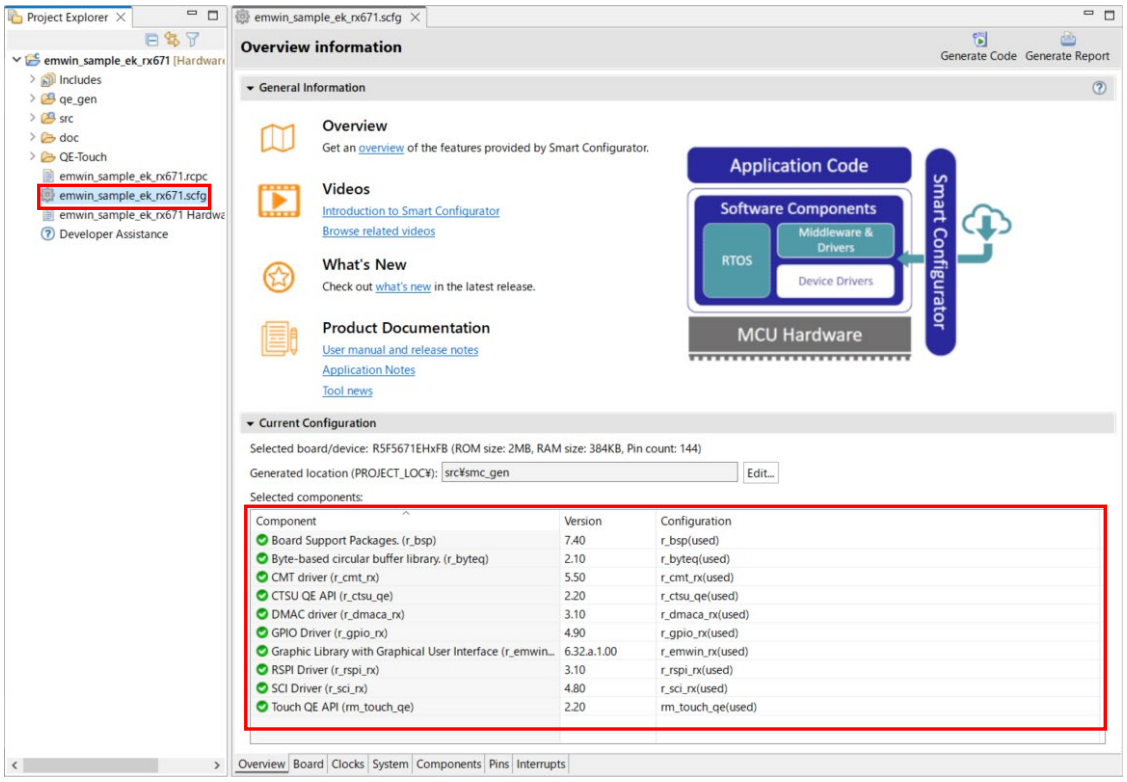




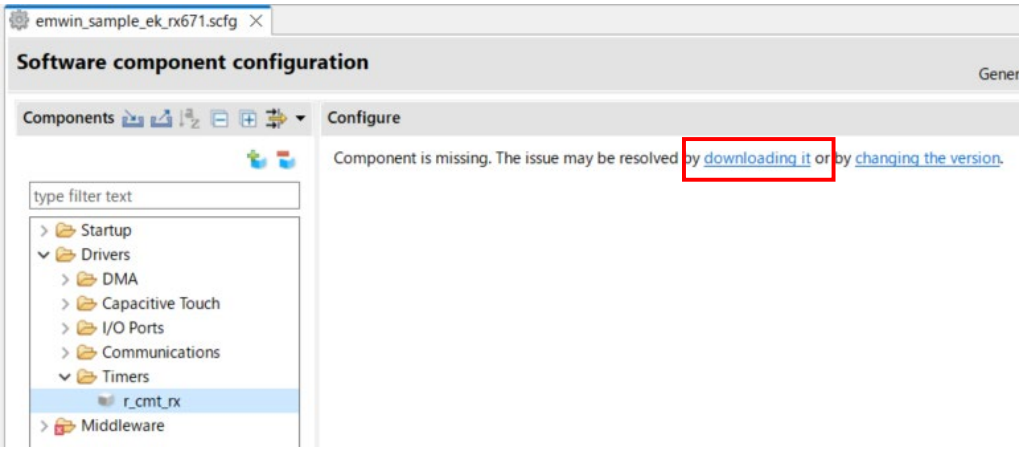
Once the import is complete, the sample project will be displayed in the Project Explorer.



Launch the Smart Configurator. Check the "Selected Components" in the "Overview" tab and make sure that the target version is downloaded to PC.

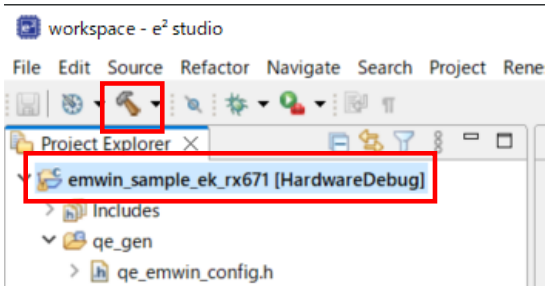


If it has not been downloaded, you can download the target version by clicking on the target module from the "Components" tab.

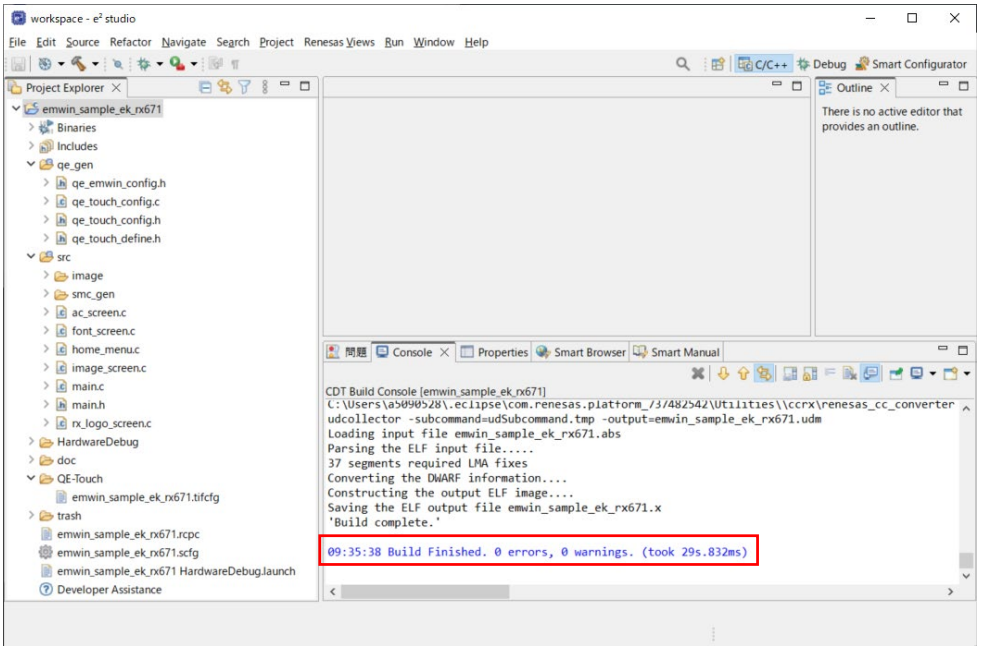


4.3 Build the Project

Select the project and click the build button.

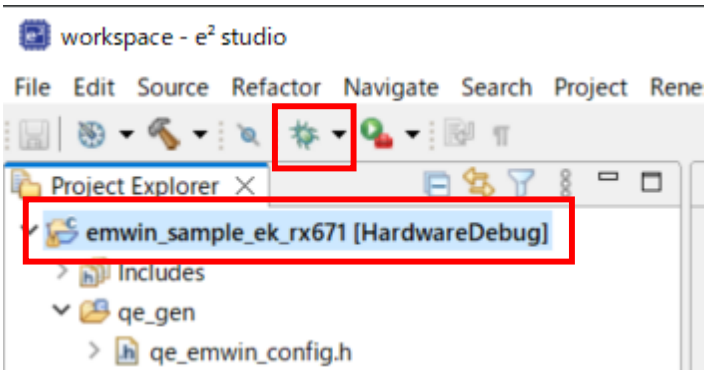


Make sure that the build is completed successfully. If the build fails, it's possible that the compiler mentioned in Section 3 Evaluation Environment is not installed, or the trial period for the free evaluation version has ended. Please check the toolchain settings in the project properties and set a usable compiler.

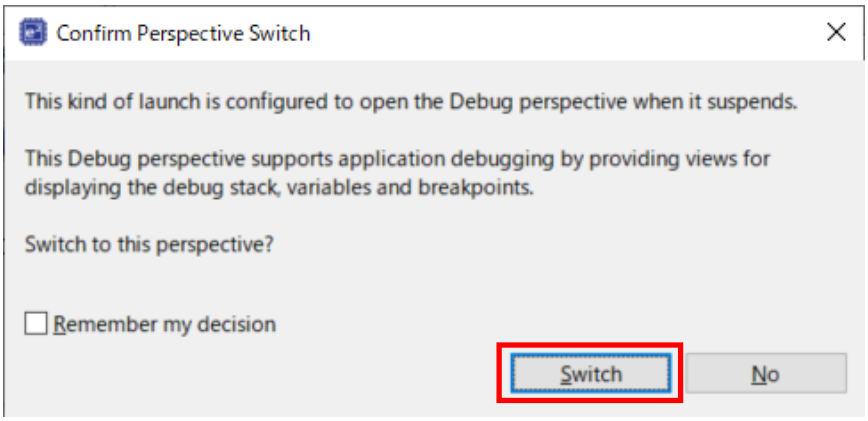


4.4 Connect Debugger and Execute the Program

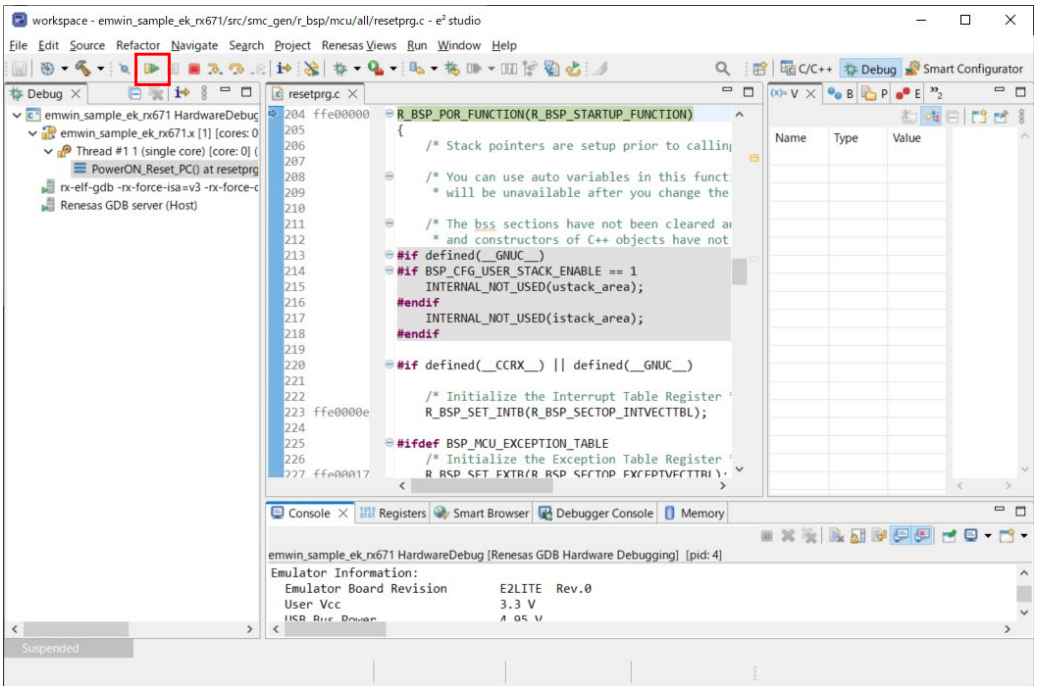
Select the project and click the debug button.



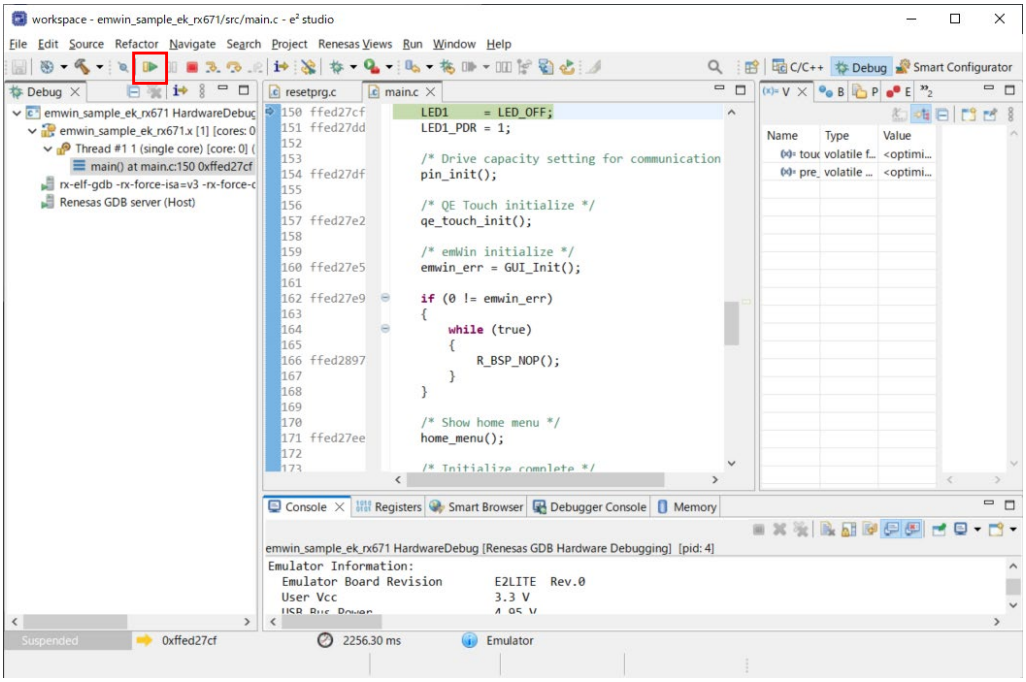
After the debugging connection is initiated, the "Confirm Perspective Switch" window will appear. Click on the "Switch" button.



Once the perspective has switched to debugging, click the "Resume" button.



Click the "Resume" button once again to execute the program.



5. Hardware Description

5.1 Hardware Configuration

Figure 5.1 shows the hardware configuration.

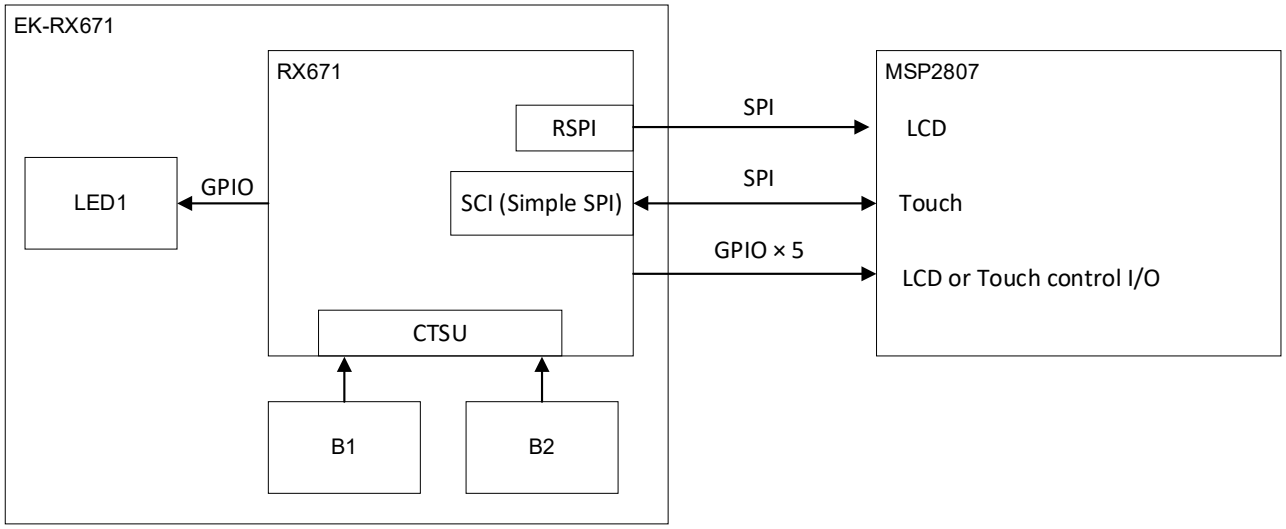


Figure 5.1 Hardware Configuration

Also, to run the sample program, the jumper settings for the EK-RX671 shown in Table 5.1 are required.

Table 5.1 Jumper settings

Jumper	Setting	Description
J30	Open	To use P16

Table 5.2 Evaluated LCD

LCD Part Number	Resolution	LCD Controller	Touch Panel Controller	Operating Power	Note
MSP2807 (Kuongshun Electronic Limited)	240×320	ILI9341 (ILITEK)	XPT2046 (Xptek)	3.3V	The sample program could be run on a product equivalent to this one.

5.2 Connection to LCD

Table 5.3 shows the connection between the EK-RX671 and the LCD. Please connect the LCD according to this table.

Table 5.3 Connection Between EK-RX671 and the LCD

EK-RX671		emWin Control Pin		LCD (MSP2807)		
Pin No.	Signal			Pin No.	Signal	Description
J2-1	3.3V	-		1	VCC	3.3V
J2-35	GND	-		2	GND	GND
J2-29	PC3	EMWIN_LCD_CS_PIN		3	CS	CS signal of LCD
J2-23	P50	EMWIN_DISP_SIGNAL_PIN		4	RESET	Reset signal of LCD
J2-22	P51	EMWIN_DATA_CMD_PIN		5	DC/RS	Data/Command signal of LCD
J2-26	PC6	MOSIA		6	SDI(MOSI)	Data input of LCD
J2-27	PC5	RSPCKA	⇔	7	SCK	Clock signal of LCD
J2-21	P52	EMWIN_BACKLIGHT_PIN		8	LED	Backlight control signal
J2-25	PC7	MISOA(Not used)		9	SDO(MISO)	Not used
J2-11	P17	SCK1		10	T_CLK	Touch panel clock
J2-14	P14	EMWIN_TOUCH_CS_PIN		11	T_CS	CS signal of touch panel
J2-12	P16	SMOSI1		12	T_DIN	Data input of touch panel
J2-13	P15	SMISO1		13	T_DO	Data output of touch panel
-				14	T_IRQ	Not used

5.3 Used Pins and Their Functions

Table 5.4 shows the used pins and their functions.

Table 5.4 Used Pins and Their Functions

Devices	Pin Name	Input/Output	Description
LCD (MSP2807)	PC6/MOSIA	Output	Data output
	PC7/MISOA	Input	Data input(Not used) Please pull up or pull down.
	PC5/RSPCKA	Output	Clock output
	PC3	Output	CS signal output
	P50	Output	LCD reset signal output
	P51	Output	Data/Command signal output
	P52	Output	Backlight control signal output
Touch Panel (MSP2807)	P17/SCK1	Output	Clock output
	P16/SMOSI1	Output	Data output
	P15/SMISO1	Input	Data input
	P14	Output	CS signal output
Touch keys (EK-RX671)	PC4/TSCAP	Input	For LPF connnection
	P33/TS1	Input	Touch button 1
	P24/TS5	Input	Touch button 2
LED	P56	Output	Control LED

6. Software Description

6.1 Software Configuration

Figure 6.1 shows the software configuration. The emWin FIT module consists of the emWin library, which serves as the core, and the interface that connects the RX MCU and the library. The Application control the screen using only the emWin API.

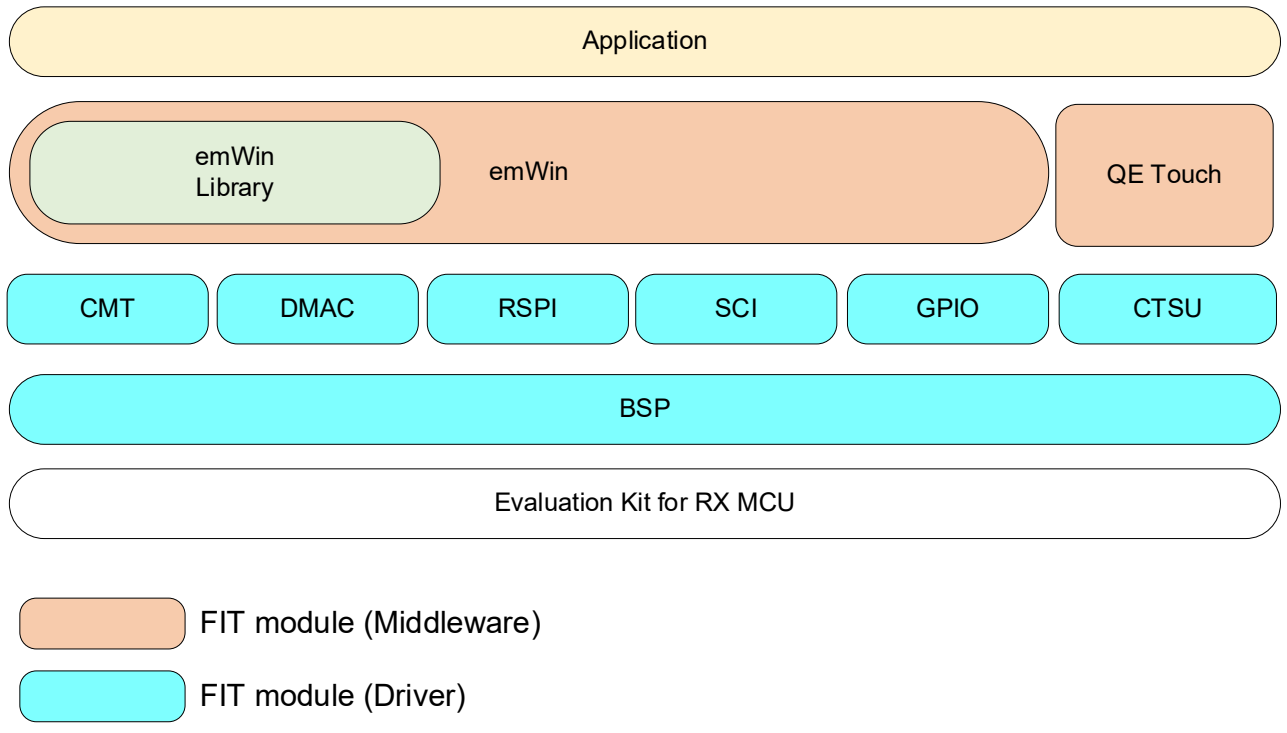


Figure 6.1 Software Configuration

6.2 Used FIT Module

Table 6.1 shows the FIT modules used in the sample program.

Table 6.1 Used FIT Modules

Module	Document Title	Document Number
BSP	RX Family Board Support Package Module Firmware Integration Technology	R01AN1685
emWin	RX Family emWin v6.32 Module Firmware Integration Technology	R01AN6771
CMT	RX Family CMT Module Firmware Integration Technology	R01AN1856
GPIO	RX Family GPIO Module Firmware Integration Technology	R01AN1721
RSPI	RX Family RSPI Module Firmware Integration Technology	R01AN1827
SCI	RX Family SCI Module Firmware Integration Technology	R01AN1815
DMAC	RX Family DMAC Module Firmware Integration Technology	R01AN2063
CTSU	RX Family QE CTSU Module Firmware Integration Technology	R01AN4469
QE Touch	RX Family QE Touch Module Firmware Integration Technology	R01AN4470
BYTEQ	RX Family BYTEQ Module Firmware Integration Technology	R01AN1683

6.3 Project Structure

Figure 6.2 shows the project structure.

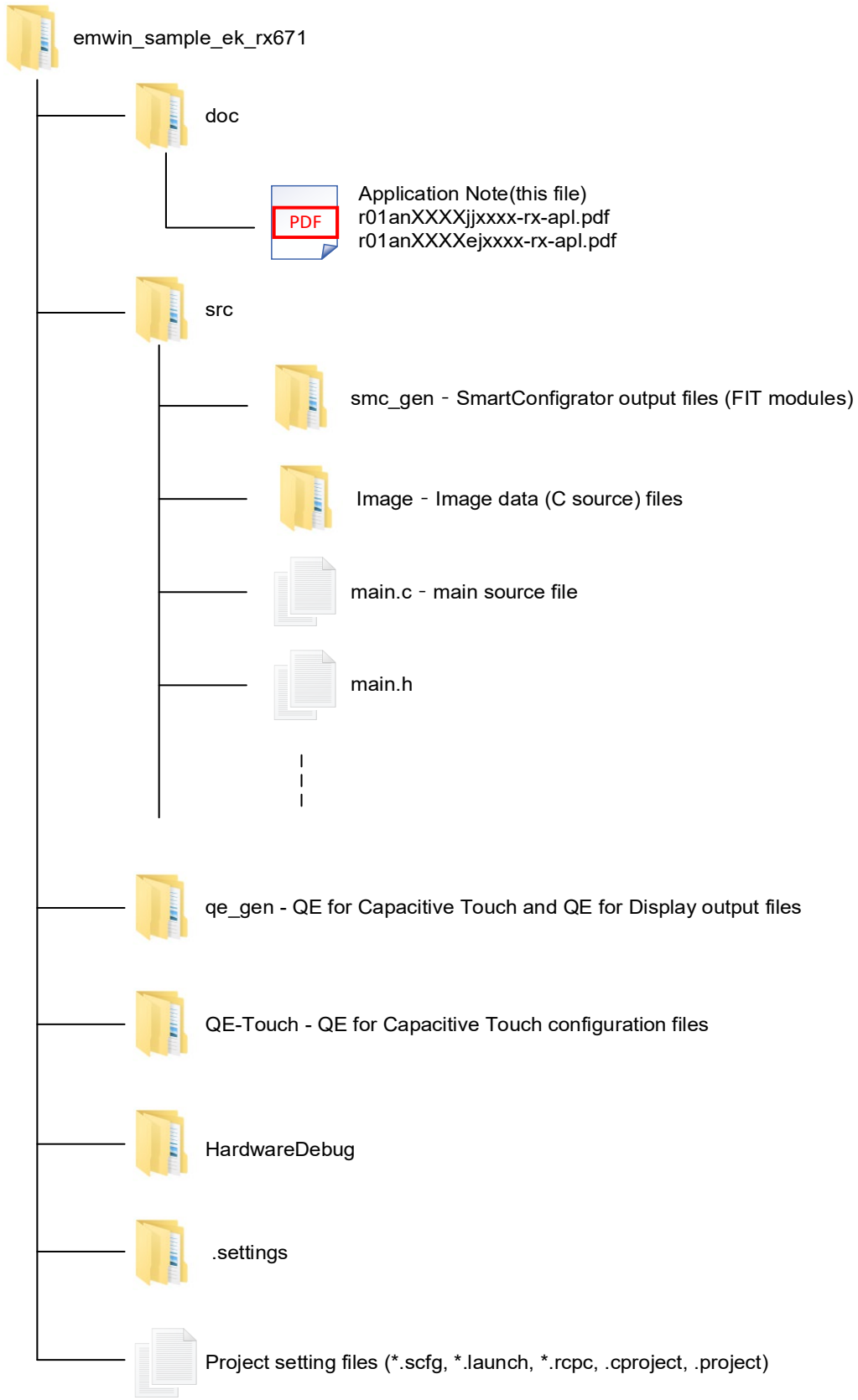


Figure 6.2 Project Structure

6.4 File Structure

The files used in this sample program are shown below. Please note that files generated from the Smart Configurator or QE are not included.

Table 6.2 Files used in this sample program

File Name		Description	Notes
Src Folder			
	main.c	Main process	
	main.h	Common header	
	home_menu.c	Home Screen process	
	rx_logo_screen.c	RX Logo Display process	
	ac_ screen.c	Air Conditioning Control process	
	image_ screen.c	Image Display process	
	font_screen.c	Font Display process	
src\image Folder			
	renesas_logo.c	Renesas logo	Generated by Bitmap Converter
	icon1.c	Icon 1 of Home Screen	
	icon2.c	Icon 2 of Home Screen	
	icon3.c	Icon 3 of Home Screen	
	icon4.c	Icon 4 of Home Screen	
	rx_logo.c	RX logo	
	home_btn.c	Home button	
	image1.c	Image 1	
	image1_thumb.c	Thumbnail of Image 1	
	image2.c	Image 2	
	image2_thumb.c	Thumbnail of Image 2	
	image3.c	Image 3	
	image3_thumb.c	Thumbnail of Image 3	
	image4.c	Image 4	
	image4_thumb.c	Thumbnail of Image 4	
	left_btn_down.c	Left button (when pressed)	
	left_btn_up.c	Left button (when not pressed)	
	right_btn_down.c	Right button (when pressed)	
	right_btn_up.c	Right button (when not pressed)	
	mode_btn_off.c	Mode selection (when pressed)	
	mode_btn_on.c	Mode selection (when not pressed)	
	temp_meter.c	Temperature meter image	
	windmill.c	Air flow meter image	

6.5 Processes in Detail

This section explains the details of each process. Because this explanation uses the functions and variables from the sample program, please refer to the sample program for further clarification.

6.5.1 Main Process (main.c)

6.5.1.1 Specification

After initializing emWin and the touch key, the main loop updates the GUI and scans the touch key.

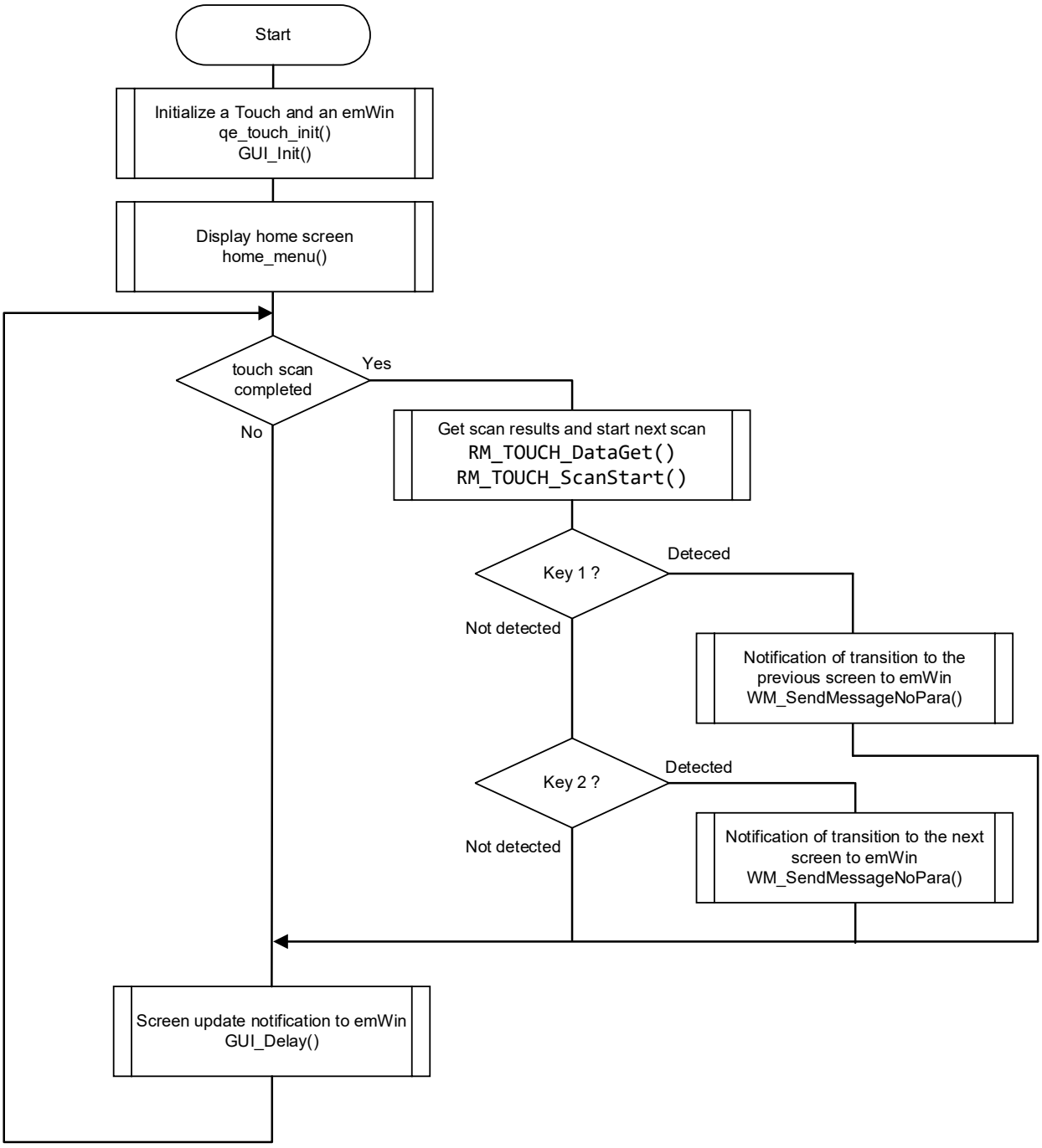


Figure 6.3 Overview Flow of Main Process

6.5.1.2 Variables

This shows the list of global variables.

Table 6.3 Global Variables

Type	Variable Name	Description
WM_HWIN	g_window	The handle of the active window being displayed

6.5.1.3 Constants

This shows the list of constants shared in the sample program.

Table 6.4 Constants（Shared in the sample program: main.h）

Constant	Setting Value	Description
LCD_SIZE_X	320	The width of the screen
LCD_SIZE_Y	240	The height of the screen
HOME_BTN_POS_X	259	X coordination of Home Button
HOME_BTN_POS_Y	0	Y coordination of Home Button
MY_NEXT_SCREEN	WM_USER + 0	User-defined message: Next screen
MY_BACK_SCREEN	WM_USER + 1	User-defined message: Previous screen
BT1_ON	1	Detection judgement value of touch key 1
BT2_ON	2	Detection judgement value of touch key 2
LED_ON	1	LED on
LED_OFF	0	LED off
LED1	PORT5.PODR.BIT.B6	PODR register bit of LED1 (P56)
LED1_PDR	PORT5.PDR.BIT.B6	PDR register bit of LED1 (P56)

6.5.1.4 Functions

This shows the list of functions of the main process.

Table 6.5 Functions (main.c)

Functions	Description
main	Main process
pin_init	Drive capacity setting process for communication pins
qe_touch_init	Initialization and initial scan process of touch keys

6.5.1.5 Function Specification

This shows the function specifications of the main process.

main	
Description	Main Process
Header	No
Declaration	void main(void)
Explanation	It initializes emWin and the touch keys. After that, it updates the GUI and scans the touch keys in the main loop.
Arguments	No
Return Value	No

pin_init	
Description	Drive capacity setting process for communication pins
Header	No
Declaration	void pin_init(void)
Explanation	It sets the drive capacity of the pins used for communication with the LCD.
Arguments	No
Return Value	No

qe_touch_init	
Description	It initialize touch key initialization and performs initial scanning
Header	No
Declaration	void qe_touch_init(void)
Explanation	After the terminal settings and initialization of the touch keys (execution of the RM_TOUCH_Open function), it performs the initial scan.
Arguments	No
Return Value	No

6.5.2 Home Screen Process (home_menu.c)

6.5.2.1 Specification

The Home Screen consists of parts as shown in Figure 6.4.

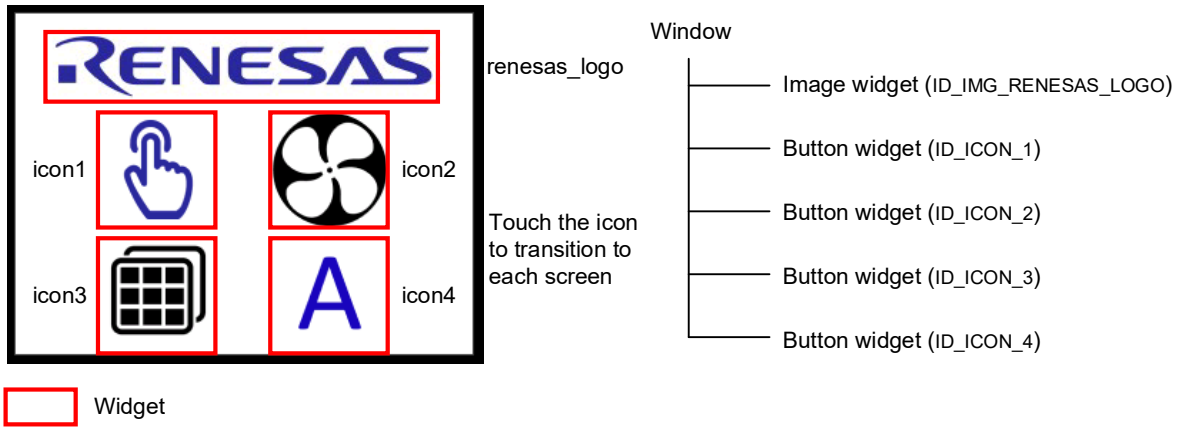


Figure 6.4 Home Screen Structure

The Home Screen is composed using Dialog function. By using it, users can set information about windows and widgets all together using the structure variable g_dialog_home_menu. The operation within the dialog is managed by the callback function cb_home_menu, and the processing is executed according to the message p_msg->MsgId.

Table 6.6 Processing According To Messages

Message	Description of Processing
WM_INIT_DIALOG	Executed only once when the dialog is generated. It initializes each widget placed in the dialog (background setting and the button image setting)
WM_NOTIFY_PARENT	Executed when an event occurs in each widget. It determines from which button the event notification came using WM_GetId(p_msg->hWinSrc), and transitions to the next screen according to the button. Before transitioning, the home screen dialog is deleted using WM_DeleteWindow(p_msg->hWin).
MY_NEXT_SCREEN	A user-defined message. It is executed upon receiving WM_SendMessageNoPara(g_window, MY_NEXT_SCREEN) from the main loop.
MY_BACK_SCREEN	A user-defined message. It is executed upon receiving WM_SendMessageNoPar(g_window, MY_BACK_SCREEN) from the main loop.

6.5.2.2 Variables

Below is a list of global variables for Home Screen processing.

Table 6.7 Global Variables

Type	Variable Name	Description
extern GUI_CONST_STORAGE GUI_BITMAP	bmrenesas_logo	Data variable of renesas_logo
extern GUI_CONST_STORAGE GUI_BITMAP	bmicon1	Data variable of icon1
extern GUI_CONST_STORAGE GUI_BITMAP	bmicon2	Data variable of icon2
extern GUI_CONST_STORAGE GUI_BITMAP	bmicon3	Data variable of icon3
extern GUI_CONST_STORAGE GUI_BITMAP	bmicon4	Data variable of icon4
extern WM_HWIN	g_window	Handle of the active window currently displayed
static const GUI_WIDGET_CREATE_INFO	g_dialog_home_menu[]	Management information of widgets used in the dialog

6.5.2.3 Constants

Below is a list of constants for Home Screen processing.

Table 6.8 Constants

Constants	**Setting Value**	**Description**
ID_WINDOW_HOME	GUI_ID_USER + 0	Management ID of the window
ID_IMG_RENESAS_LOGO	GUI_ID_IMAGE0	Management ID of image widget (renesas_logo)
ID_ICON_1	GUI_ID_BUTTON1	Management ID of button widget (icon1)
ID_ICON_2	GUI_ID_BUTTON2	Management ID of button widget (icon2)
ID_ICON_3	GUI_ID_BUTTON3	Management ID of button widget (icon3)
ID_ICON_4	GUI_ID_BUTTON4	Management ID of button widget (icon4)

6.5.2.4 Function

Below is a list of functions for Home Screen processing.

Table 6.9 Functions

Function	**Description**
cb_home_menu	Callback function of the Home Screen
home_menu	Home Screen dialog creation

6.5.2.5 Function Specification

The function specification for Home Screen processing is as follows.

cb_home_menu	
Description	Callback function of the home screen
Header	No
Declaration	static void cb_home_menu(WM_MESSAGE * p_msg);
Explanation	Controls the Home Screen
Arguments	WM_MESSAGE * p_msg Pointer to the message of the dialog (window)
Return Value	No

home_menu	
Description	Home Screen dialog creation
Header	main.h
Declaration	void home_menu(void);
Explanation	Create the Home Screen by executing the GUI_CreateDialogBox function.
Arguments	No
Return Value	No

6.5.3 RX Logo Display Processing (rx_logo_screen.c)

6.5.3.1 Specification

The RX Logo Display consists of parts as shown in Figure 6.5.

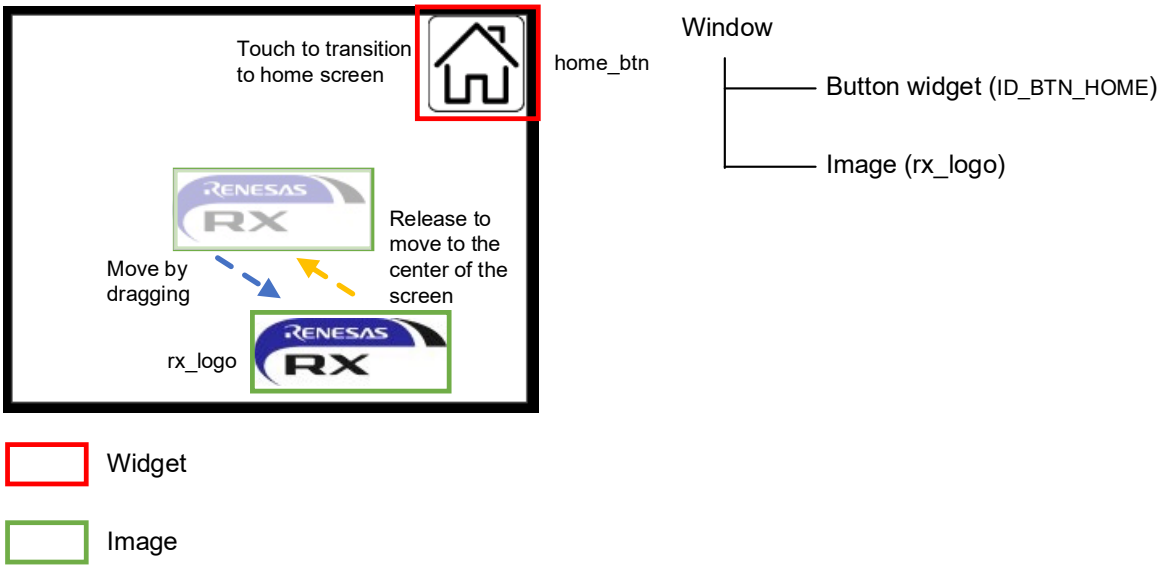


Figure 6.5 RX Logo Display Screen

The operation within the window is managed by the callback function cb_rx_logo_screen, and processing is executed according to the message p_msg->MsgId.

Table 6.10 Processing According to Message

Message	Description of Processing
WM_CREATE	Executed only once when the window is created. It initializes the button widget placed within the window and sets the initial coordinates of the RX logo.
WM_PAINT	Clears the screen and draws the RX logo.
WM_TOUCH	Executed when touch operation is being performed. When the RX logo is touched, the WM_SetCapture function is executed to occupy the touch event during drag operation. During dragging, it obtains the current touch coordinates, updates the coordinates of the RX logo, and instructs screen update with the WM_InvalidateWindow function. When the RX logo is released, the WM_ReleaseCapture function releases the touch event and performs the animation processing.
WM_NOTIFY_PARENT	Executes when an event occurs in the widget. WM_GetId(p_msg->hWinSrc) determines if it's an event notification from the home button (home_btn) and transitions to the Home Screen if it is determines. Before transitioning, the RX logo screen dialog is removed with WM_DeleteWindow(p_msg->hWin).
MY_NEXT_SCREEN	User-defined message. It is executed upon receiving WM_SendMessageNoPara(g_window, MY_NEXT_SCREEN) from the main loop.
MY_BACK_SCREEN	User-defined message. It is executed upon receiving WM_SendMessageNoPara(g_window, MY_BACK_SCREEN) from the main loop.

The animation feature is used for the processing of returning the RX logo to the center of the screen after drag operation.

The anim_setup function initializes and starts the animation. Once the animation starts, the cb_anim_func function is executed every ANIM_SLICE time (100ms) set by the GUI_ANIM_Create function. The cb_anim_func function updates the coordinate information of the RX logo and instructs the screen update with the WM_InvalidateWindow function.

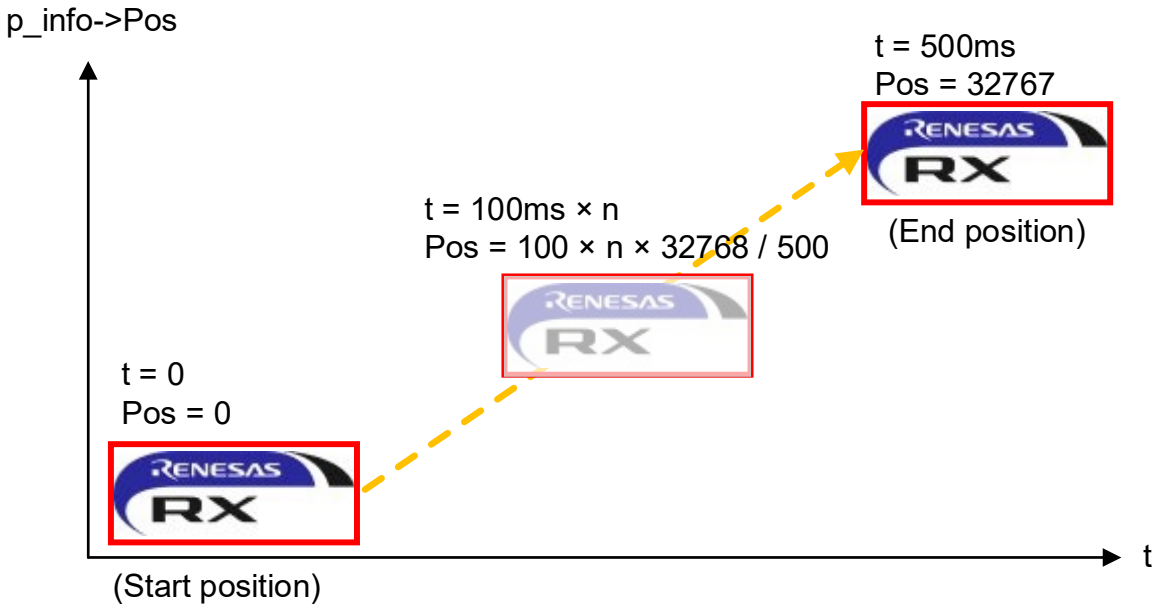


Figure 6.6 Overview of The RX Logo Animation

6.5.3.2 Variables

Below is a list of global variables for RX Logo Display processing.

Table 6.11 Global Variables

Type	Variable Name	Description
extern GUI_CONST_STORAGE GUI_BITMAP	bmhome_btn	Data variable of home_btn
extern GUI_CONST_STORAGE GUI_BITMAP	bmrx_logo	Data variable of rx_logo
WM_HWIN	g_window	Handle of the active window being displayed
static GUI_POINT	g_logo_pos	Coordinates of rx_logo

6.5.3.3 Constants

Below is a list of constants for RX Logo Display processing.

Table 6.12 Constants

Name	Setting Value	Description
ANIM_PERIOD	500	Animation execution time (ms)
ANIM_SLICE	100	Animation processing execution cycle (ms)
BM_RX_LOGO	&bmrx_logo	Pointer to the data variable of rx_logo
ID_BTN_HOME	GUI_ID_BUTTON0	Management ID of button widget (home_btn)

6.5.3.4 Function

Below is a list of functions for RX Logo Display processing.

Table 6.13 Function

Name	Description
cb_anim_func	Execution of animation processing
anim_setup	Initialization and start of animation
cb_rx_logo_screen	Callback function of RX Logo Screen
rx_logo_screen	RX Logo Screen window creation

6.5.3.5 Function Specification

The function specification for RX Logo Display processing is as follows.

cb_anim_func		
Description	Execution of animation processing	
Header	No	
Declaration	static void cb_anim_func(GUI_ANIM_INFO * p_info, void * p_void);	
Explanation	This is a callback function of animation items registered by the GUI_ANIM_AddItem function. It is executed periodically. It updates the coordinates of the RX logo and instructs the screen update using the WM_InvalidateWindow function.	
Arguments	GUI_ANIM_INFO * p_info	Animation information
	void * p_void	Pointer to user data
Return Value	No	

anim_setup		
Description	Initializes and starts the animation.	
Header	No	
Declaration	static void anim_setup(WM_HWIN h_win);	
Explanation	It stores the current coordinates of the RX logo in the anim_data variable and initializes the animation processing using the GUI_ANIM_Create and GUI_ANIM_AddItem functions. The anim_data variable, which is passed an argument during initialization, can be referenced by the cb_anim_func callback function.	
Arguments	WM_HWIN h_win	Window handle
Return Value	No	

cb_rx_logo_screen		
Description	Callback function of RX Logo Screen	
Header	No	
Declaration	static void cb_rx_logo_screen(WM_MESSAGE * p_msg);	
Explanation	It controls the behavior of the RX logo screen.	
Arguments	WM_MESSAGE * p_msg	Window handle
Return Value	No	

rx_logo_screen	
Description	RX Logo Screen window creation
Header	main.h
Declaration	void rx_logo_screen(void);
Explanation	It executes the WM_CreateWindow function to generate the RX Logo Screen
Arguments	No
Return Value	No

6.5.4 Air Conditioning Control Processing (ac_screen.c)

6.5.4.1 Specification

The Air Conditioning Control Display consists of parts shown in Figure 6.7.

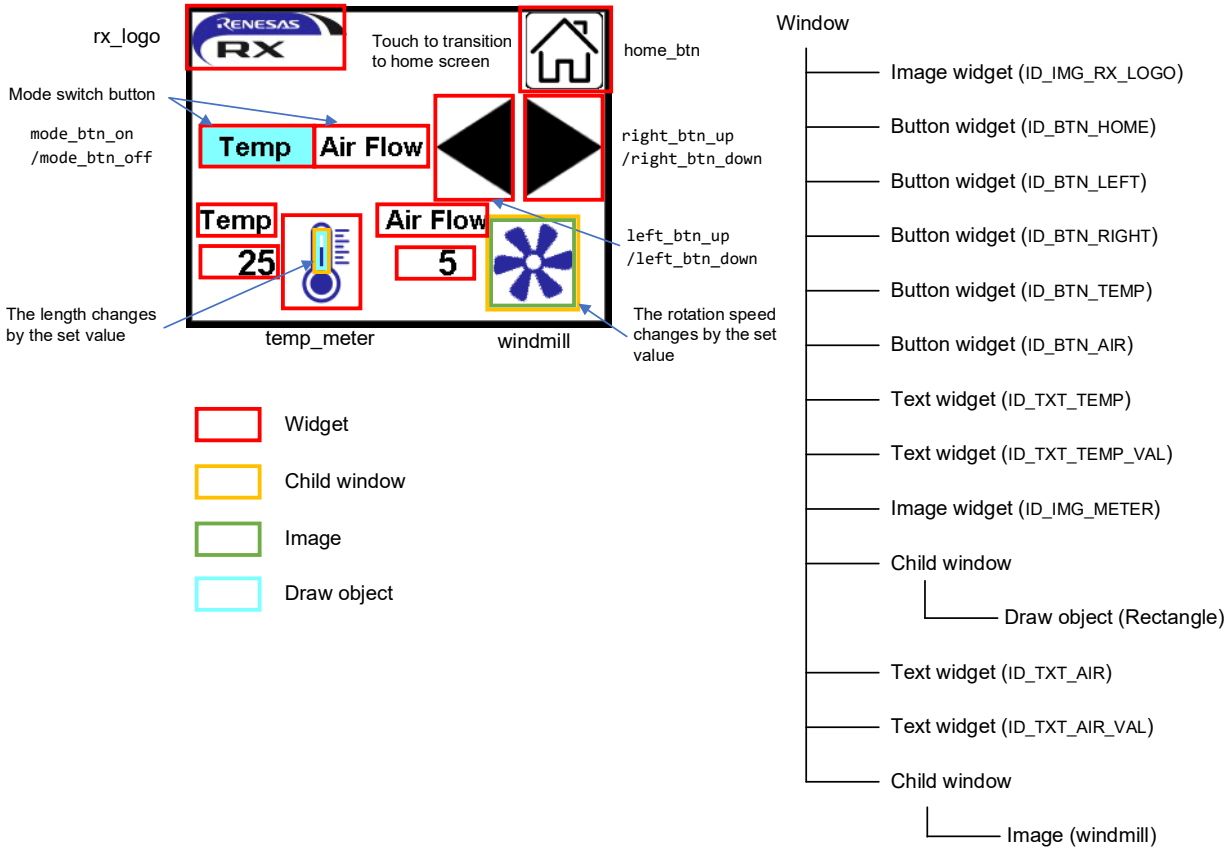


Figure 6.7 Air Conditioning Control Screen

The operation inside the window is managed by the callback function cb_ac_screen, and processing is executed according to the message p_msg->MsgId.

Table 6.14 Processing According to Message (cb_ac_screen)

Message	Description of Processing
WM_INIT_DIALOG	Executed only once when the window is created. It initializes the widgets inside the window and creates child windows.
WM_NOTIFY_PARENT	Executed when an event occurs in a widget. It determines the source of the event using WM_GetId(p_msg->hWinSrc) and performs respective processing.
WM_DELETE	Deletes the child windows for the temperature meter and fan speed meter.
MY_NEXT_SCREEN	User-defined message It is executed upon receiving WM_SendMessageNoPara(g_window, MY_NEXT_SCREEN) from the main loop.
MY_BACK_SCREEN	User-defined message It is executed upon receiving WM_SendMessageNoPara(g_window, MY_BACK_SCREEN) from the main loop

In this sample program, two child windows are created for the temperature meter and fan speed meter. The callback functions for each window are cb_win_temp and cb_win_flow, respectively.

Table 6.15 Processing According to Messages (cb_win_temp)

Message	Description of Processing
WM_PAINT	It obtains the temperature setting value from the parent window using WM_GetUserData and calculates the dimensions of the rectangle to be drawn. Then, the rectangle is drawn using GUI_FillRectEx.。

Table 6.16: Processing Based on Messages (cb_win_flow)

Message	Description of Processing
WM_CREATE	Executed only once when the window is created. It creates two memory devices and a timer for rotation processing. GUI_MEMDEV_CreateFixed32 and WM_CreateTimer functions are used. The purpose of the created memory devices and timer is as follows: Memory Device (h_mem[0]): Stores the image before rotation. Memory Device (h_mem[1]): Stores the image after rotation Timer: Window drawing cycle (50ms)
WM_PAINT	Clears the window and draws the image from Memory Device (1) to this window using GUI_MEMDEV_WriteAt.
WM_TIMER	Executed after a specified time interval. It obtains the fan speed setting value from the parent window using WM_GetUserData and calculates the rotation angle of the image. Then, the rotated image is drawn to Memory Device (1) using GUI_MEMDEV_Rotate. Afterwards, WM_InvalidateWindow is used to instruct screen update, and WM_RestartTimer restarts the timer.
WM_DELETE	Executed when the window is deleted. It deletes the memory devices.

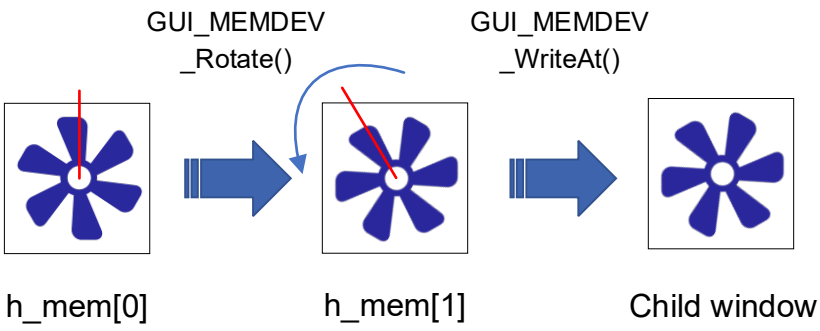


Figure 6.8 Rotation Processing Using Memory Devices

6.5.4.2 Variables

Below is a list of global variables for Air Conditioning Control Display processing.

Table 6.17 Global Variables

Type	Variable Name	Description
extern GUI_CONST_STORAGE GUI_BITMAP	bmhome_btn	Data variable of home_btn
extern GUI_CONST_STORAGE GUI_BITMAP	bmrx_logo	Data variable of rx_logo
extern GUI_CONST_STORAGE GUI_BITMAP	bmleft_btn_up	Data variable of left_btn_up
extern GUI_CONST_STORAGE GUI_BITMAP	bmleft_btn_down	Data variable of left_btn_down
extern GUI_CONST_STORAGE GUI_BITMAP	bmright_btn_up	Data variable of right_btn_up
extern GUI_CONST_STORAGE GUI_BITMAP	bmright_btn_dow n	Data variable of right_btn_down
extern GUI_CONST_STORAGE GUI_BITMAP	bmwindmill	Data variable of windmill
extern GUI_CONST_STORAGE GUI_BITMAP	bmtemp_meter	Data variable of temp_meter
extern GUI_CONST_STORAGE GUI_BITMAP	bmmode_btn_on	Data variable of mode_btn_on
extern GUI_CONST_STORAGE GUI_BITMAP	bmmode_btn_off	Data variable of mode_btn_off
WM_HWIN	g_window	The handle of the active window being displayed
static const GUI_WIDGET_CREATE_INFO	g_dialog_ac_scre en []	The management information for the widgets used in the dialog.

6.5.4.3 Constants

Below is a list of constants for Air Conditioning Control Display processing.

Table 6.18 Constants

Name	Setting Value	Description
TIMER_PERIOD	50	Air flow meter window update period in ms
WINDMILL_POS_X	230	X-coordinate of the air flow meter window
WINDMILL_POS_Y	160	Y-coordinate of the air flow meter window
SET_VAL_MIN	0	Minimum setting value for temperature and air flow
TEMP_VAL_MAX	40	Maximum setting value for temperature
AIR_VAL_MAX	20	Maximum setting value for air flow
TEMP_METER_WIDTH	6	Width of the meter window
TEMP_METER_HEIGHT	33	Height of the meter window
TEMP_METER_COLOR	0xFF9D282A	Color code for the meter
ID_WINDOW_AIR_CONDITIONING	GUI_ID_USER + 0x00	Management ID for the window
ID_IMG_RX_LOGO	GUI_ID_IMAGE0	Management ID for the image widget (rx_logo)
ID_IMG_METER	GUI_ID_IMAGE1	Management ID for the image widget (temp_meter)
ID_BTN_TEMP	GUI_ID_BUTTON0	Management ID for the button widget (Temp)
ID_BTN_AIR	GUI_ID_BUTTON1	Management ID for the button widget (Air flow)
ID_BTN_LEFT	GUI_ID_BUTTON2	Management ID for the button widget (left_btn)
ID_BTN_RIGHT	GUI_ID_BUTTON3	Management ID for the button widget (right_btn)
ID_BTN_HOME	GUI_ID_BUTTON4	Management ID for the button widget (home_btn)
ID_TXT_TEMP	GUI_ID_TEXT0	Management ID for the text widget ("Temp")
ID_TXT_AIR	GUI_ID_TEXT1	Management ID for the text widget ("Air Flow")
ID_TXT_TEMP_VAL	GUI_ID_TEXT2	Management ID for the text widget (Temperature Setting Value)
ID_TXT_AIR_VAL	GUI_ID_TEXT3	Management ID for the text widget (Air Flow Setting Value)

6.5.4.4 Functions

Below is a list of functions for Air Conditioning Control Display processing.

Table 6.19 Functions

Name	Description
cb_win_temp	Callback function of temperature meter processing
cb_win_flow	Callback function of airflow meter processing
cb_ac_screen	Callback function of Air Conditioning Control screen
ac_screen	Window creation of Air Conditioning Control screen

6.5.4.5 Function Specification

The function specification for Air Conditioning Control Display is as follows.

cb_win_temp	
Description	Callback function of temperature meter processing
Header	No
Declaration	static void cb_win_temp(WM_MESSAGE * p_msg);
Explanation	It controls drawing process of the temperature meter
Arguments	WM_MESSAGE * p_msg Window information
Return Value	No

cb_win_flow	
Description	Callback function of airflow meter processing
Header	No
Declaration	static void cb_win_flow(WM_MESSAGE * p_msg);
Explanation	It controls drawing process of the airflow meter
Arguments	WM_MESSAGE * p_msg Window information
Return Value	No

cb_ac_screen	
Description	Callback function of Air Conditioning Control screen
Header	No
Declaration	static void cb_ac_screen(WM_MESSAGE * p_msg);
Explanation	It controls the screen of Air Conditioning Control Display
Arguments	WM_MESSAGE * p_msg Window information
Return Value	No

ac_screen	
Description	Window creation of Air Conditioning Control screen
Header	main.h
Declaration	void ac_screen(void)
Explanation	It execute the GUI_CreateDialogBox function to generate the Air Conditioning Control screen.
Arguments	No
Return Value	No

6.5.5 Image Display Processing (image_screen.c)

6.5.5.1 Specification

The Image Display consists of parts shown in Figure 6.9.

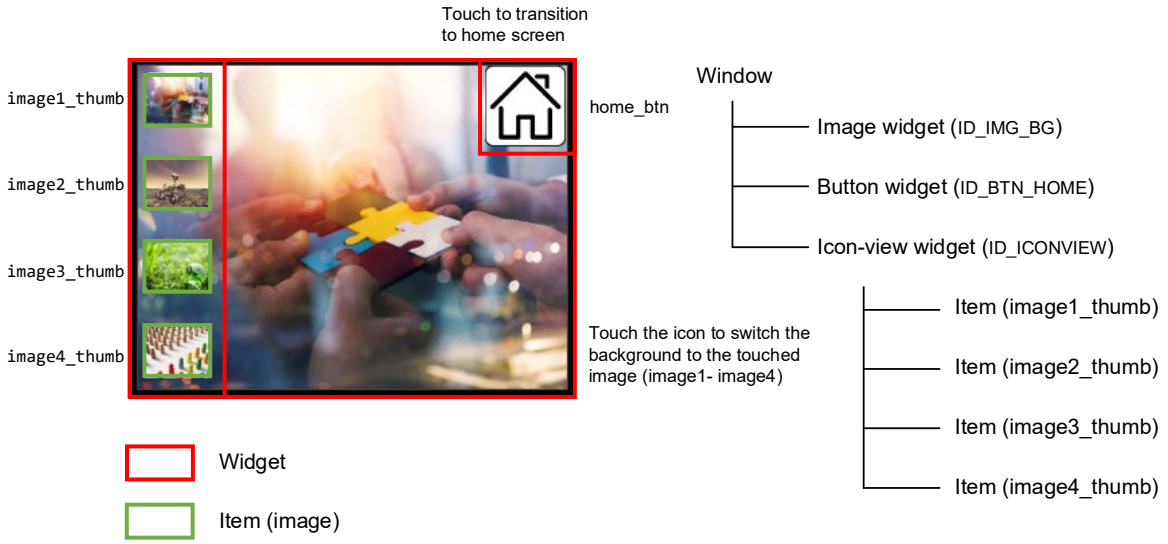


Figure 6.6 Image Display Screen

Operations in the window are managed by the callback function cb_image_screen, and processing is executed according to the message p_msg->MsgId.

Table 6.20 Processing According to Message

Message	Description of Processing
WM_CREATE	Executed only once when the window is created. It initializes each widget placed in the window (background image setting and icon view setting).
WM_PAINT	It draws the background image selected in the icon view with the IMAGE_SetBitmap function
WM_NOTIFY_PARENT	Executed when an event occurs in each widget. WM_GetId(p_msg->hWinSrc) is used to determine which button notified the event, performe processing according to the button.
MY_NEXT_SCREEN	User-defined message. It is executed upon receiving WM_SendMessageNoPara(g_window, MY_NEXT_SCREEN) from the main loop.
MY_BACK_SCREEN	User-defined message. It is executed upon receiving WM_SendMessageNoPara(g_window, MY_BACK_SCREEN) from the main loop.

6.5.5.2 Variables

Below is a list of global variables for Image Display processing.

Table 6.21 Global Variables

Type	Variable Name	Description
extern GUI_CONST_STORAGE GUI_BITMAP	bmhome_btn	Data variables of home_btn
extern GUI_CONST_STORAGE GUI_BITMAP	bmimage1	Data variables of image1
extern GUI_CONST_STORAGE GUI_BITMAP	bmimage2	Data variables of image2
extern GUI_CONST_STORAGE GUI_BITMAP	bmimage3	Data variables of image3
extern GUI_CONST_STORAGE GUI_BITMAP	bmimage4	Data variable of image4
extern GUI_CONST_STORAGE GUI_BITMAP	bmimage1_thumb	Data variable for image1_thumb (Thumbnail of image1)
extern GUI_CONST_STORAGE GUI_BITMAP	bmimage2_thumb	Data variable for image2_thumb (Thumbnail of image2)
extern GUI_CONST_STORAGE GUI_BITMAP	bmimage3_thumb	Data variable for image3_thumb (Thumbnail of image3)
extern GUI_CONST_STORAGE GUI_BITMAP	bmimage4_thumb	Data variable for image4_thumb (Thumbnail of image4)
extern WM_HWIN	g_window	Handle of the active window being displayed

6.5.5.3 Constants

Below is a list of constants for Image Display processing.

Table 6.22 Constants

Name	Setting Value	Description
ICONVIEW_WIDTH	60	Width of icon view
ICONVIEW_HEIGHT	LCD_GetYSize()	Height of icon view (function to get the height of LCD)
ICON_WIDTH	50	Icon width
ICON_HEIGHT	38	Height of icon
ID_IMG_BG	GUI_ID_IMAGE0	Management ID of image widget(background_image)
ID_BTN_HOME	GUI_ID_BUTTON0	Management ID of button widget(home_btn)
ID_ICONVIEW	GUI_ID_ICONVIEW0	Management ID of an icon view widget

6.5.5.4 Functions

Below is a list of functions for Image Display processing.

Table 6.23 Functions

Name	Description
cb_image_screen	Callback function of image display screen
image_screen	Window Creation of the image display screen

6.5.5.5 Function Specification

The function specification for Image Display is as follows.

cb_image_screen	
Description	Callback function of image display screen
Header	No
Declaration	static void cb_image_screen(WM_MESSAGE * p_msg);
Explanation	It controls the Image Display screen
Arguments	WM_MESSAGE * p_msg Window information
Return Value	なし

image_screen	
Description	Window Creation of the image display screen
Header	main.h
Declaration	void image_screen (void);
Explanation	It execute WM_CreateWindow function to create Image Display screen
Arguments	No
Return Value	No

6.5.6 Font Display Processing (font_screen.c)

6.5.6.1 Specification

The Font Display consists of parts shown in Figure 6.10.

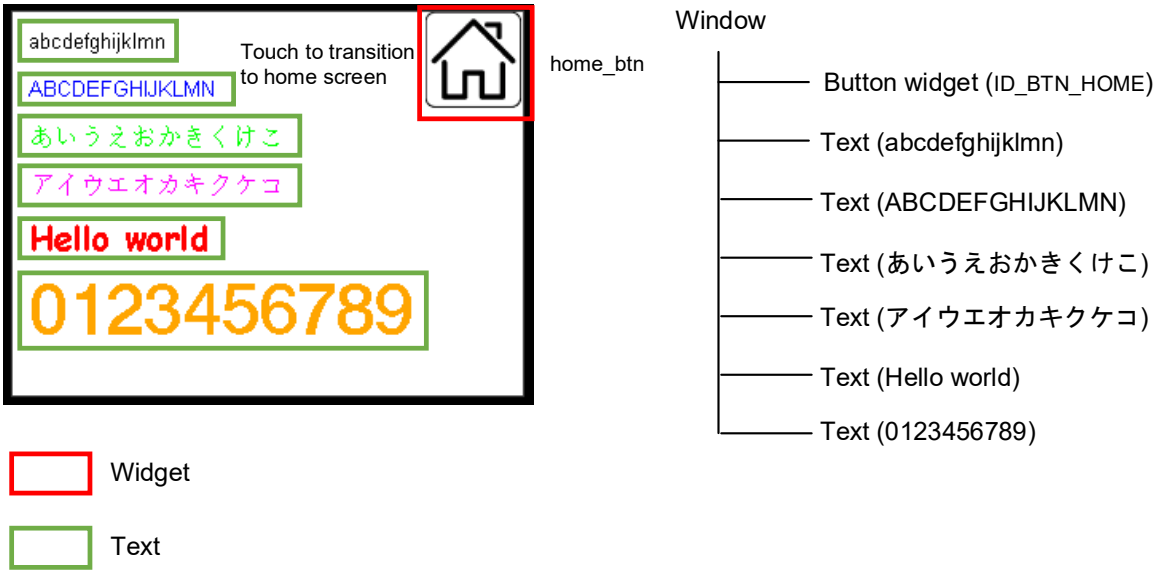


Figure 6.7 Font Display

Operations in the window are managed by the callback function cb_font_screen, and processing is executed according to the message p_msg->MsgId.

Table 6.20 Processing According to Message

Message	Description of Processing
WM_CREATE	Executed only once when a window is created. It initializes the button widgets to be placed in the window.
WM_PAINT	Draws each font.
WM_NOTIFY_PARENT	Executed when an event occurs in each widget. WM_GetId(p_msg->hWinSrc) determines if the event notification is from the home button (home_btn) and transitions to the home screen. Before transition, WM_DeleteWindow(p_msg->hWin) deletes the RX logo screen dialog.
MY_NEXT_SCREEN	User-defined message. It is executed upon receiving WM_SendMessageNoPara(g_window, MY_NEXT_SCREEN) from the main loop.
MY_BACK_SCREEN	User-defined message. It is executed upon receiving WM_SendMessageNoPara(g_window, MY_BACK_SCREEN) from the main loop.

All fonts used in the font display screen are provided by emWin. The fonts used are listed in Table 6.25.

Table 6.14 Lists of Used Fonts

Font	Description
GUI_FONT_16_ASCII	Alphabet (ASCII) font, 16h
GUI_FONT_16_HK	Japanese (Hiragana & Katakana) font, 16h
GUI_FONT_COMIC24B_ASCII	Comic-like font (ASCII), 24h
GUI_FONT_D24X32	Numeric font, 24w x 32h

Japanese fonts are converted to character codes using U2C including in emWin. For example, Hiragana (str_hiragana_utf8), the first character "\xe3\x81\x82" is the character code for "あ".

```
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"
};
```

Figure 6.11. Hiragana & Katakana Character Codes

6.5.6.2 Variables

Below is a list of global variables for Font Display processing.

Table 6.26 Global Variables

Type	Variable Name	Description
extern GUI_CONST_STORAGE GUI_BITMAP	bmhome_btn	Data variable of home_btn
extern WM_HWIN	g_window	Handle of the active window being displayed
static const char *	str_hiragana_utf8[]	Character code for Hiragana (あいうえおかきくけこ)
static const char *	str_katakana_utf8[]	character code for Katakana(アイウエオカキクケコ)

6.5.6.3 Constants

Below is a list of constants for Font Display processing.

Table 6.27 Constants

Name	Setting Value	Description
ID_BTN_HOME	GUI_ID_BUTTON0	Management ID for the button widget (home_btn)

6.5.6.4 Functions

Below is a list of functions for Font Display processing.

Table 6.28 Functions

Name	Description
cb_font_screen	Callback function for font display screen
font_screen	Window Creation for the font display screen

6.5.6.5 Function Specification

The function specification for Font Display is as follows.

cb_font_screen	
Description	Callback function for font display screen
Header	No
Declaration	static void cb_font_screen(WM_MESSAGE * p_msg);
Explanation	It controls the font display screen.
Arguments	WM_MESSAGE * p_msg Window information
Return Value	No

font_screen	
Description	Window Creation for the font display screen
Header	main.h
Declaration	void font_screen (void);
Explanation	It executes the WM_CreateWindow function to generate the font display screen.
Arguments	No
Return Value	No

6.6 Resources Usage

The resources for this sample program are listed below. These are reference values under the following conditions.

- C/C++ Compiler Package for RX Family V3.05.00
- Optimization level: Level 2

6.6.1 Overall Resources Usage

Table 6.29 Overall resources of the sample program

Memory	Usage (in byte)	Note
RAM	269,155	Of this, 250 Kbytes are reserved for emWin work size.
ROM	1,055,073	
Stack	1,355	

6.6.2 Resources Usage of Each Screens

The resources usage for each screen are shown following. Note that the FIT module, main processing-related resources, and the cache used by emWin (150 Kbytes) are not included.

6.6.2.1 Home Screen

Table 6.30 Resources of Home Screen

Memory	Usage (in byte)	Note
RAM	1,269	emWin work (1,269 bytes)
ROM	81,769	Programs, constants, images

6.6.2.2 RX Logo Display

Table 6.31 Resources of RX Logo Display

Memory	Usage (in byte)	Note
RAM	521	Variables (28 bytes) + emWin work (493 bytes)
ROM	21,503	Program, constants, images

6.6.2.3 Air Conditioning Control

Table 6.32 Resources of Air Conditioning Control

Memory	Usage (in byte)	Note
RAM	42,568	Variables (39 bytes) + emWin work (42,529 bytes) (Of this, 39,200 bytes are used by memory devices)
ROM	88,956	Program, constants, images

6.6.2.4 Image Display

Table 6.33 Resources of Image Display

Memory	Usage (in byte)	Note
RAM	1,181	Variables (8 bytes) + emWin Work (1,173 bytes)
ROM	641,048	Programs, constants, images

6.6.2.5 Font Display

Table 6.34 Resources for Font Display

Memory	Usage (in byte)	Note
RAM	397	Variables (8 bytes) + emWin work (389 bytes)
ROM	11,428	Programs, constants, images

6.7 Tools Used

6.7.1 QE for Display

This sample code uses QE for Display, a plug-in for e^2 studio, an integrated development environment compatible with Renesas RX microcontrollers. It supports efficient development of embedded systems with display devices by providing graphical interface settings in conjunction with various tools and FIT modules.

For details, please refer to the following URL:

https://www.renesas.com/us/en/software-tool/qe-display-development-assistance-tool-display-applications

6.7.2 QE for Capacitive Touch

This sample code uses QE for Capacitive Touch, a plug-in for e^2 studio, an integrated development environment that supports Renesas RX microcontrollers, and a stand-alone version that can be linked with CS+ and IAR EWRX (download (included in the package). In the development of embedded systems using capacitive touch functionality, this plug-in supports efficient development by making it easy to perform initial settings and sensitivity tuning of the touch interface.

For details, please refer to the following URL

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

7. Additional Explanation to Screen Update Speed

The explanation described in Section 2.3 Screen Update Speed is based on the features of the emWin library, however it also should be considered from a hardware perspective.

7.1 Communication Baud Rate

The higher the communication baud rate, the faster the screen update speed can be increased, but there may be limitations on baud rate settings. In the case of the RX671, the RSPI used in the communication interface with the LCD can be set to a maximum 40 MHz, but the system clock (ICLK) and peripheral module clock A (PCLKA) must be set to 80MHz. Because of lower the operating frequency in trade-off with the maximum baud rate, the screen update speed may be slower even if the maximum baud rate is set, depending on the processing load of the user application and emWin. Be careful of the appropriate operating frequency and baud rate in light of the processing load of the user application and emWin.

7.2 Selecting the DMA Transfer Function

When the emWin FIT module uses RSPI or SCI (simple SPI mode) for the communication interface with the LCD, the DMA transfer function can be used to reduce the overhead between communications and improve communication efficiency. The RX family is equipped with two types of DMACs: DMACs are faster than DTCs in terms of transfer rate. Therefore, it is recommended that DMAC be used under normal circumstances.

7.3 Compile Options

One way to increase screen update speed is to improve code efficiency through compilation options. Maximizing the optimization level and specifying the execution performance emphasis (-speed option) can further reduce the communication overhead.

8. Project Configuration Information

8.1 Smart Configurator

The FIT module used in the sample program and the Smart Configurator settings in e² studio are shown below. For details on each FIT module, please refer to the documentation for each FIT module.

Table 8.1 BSP Module Settings

Directory	Target	Setting & Description
Smart Configurator >> Clock		Default setting when "EK-RX671" is selected for board selection when creating a project.
	VCC Setting	3.3(V)
	Main clock setting	Operation: Check Oscillation source: Crystal Frequency: 24 (MHz) Oscillation stabilization time: 9980 (µs)
	PLL circuit setting	Division ratio：x1 Multiplication ratio：x10.0
	Sub-clock oscillator setting	Operation: Check Frequency: 32.768 (kHz) Oscillator drive capability: Standard CL Oscillator stability time: 2000 (m
	HOCO clock setting	Stop: Uncheck
	LOCO clock setting	Stop: Uncheck
	System clock setting	Clock source: PLL System clock (ICLK): x1/2 120 (MHz) Peripheral module clock (PCLKA): x1/2 120 (MHz) Peripheral module clock (PCLKB): x1/4 60 (MHz) Peripheral module clock (PCLKC): x1/4 60 (MHz) Peripheral module clock (PCLKD): x1/4 60 (MHz) FlashIF clock (FCLK): x1/4 60 (MHz)
	IWDT dedicated clock setting	Stop: Uncheck

Table 8.2 GPIO Module Settings

Directory	Target	Setting & Description
Smart Configurator >> Components >> r_gpio_rx		Default setting (no change)

Table 8.3 CMT Module Settings

Directory	Target	Setting & Description
Smart Configurator >> Components >> r_cmt_rx		Default setting (no change)

Table 8.4 BYTEQ Module Settings

Directory	Target	Setting & Description
Smart Configurator >> Components >> r_byteq		Default setting (no change)

Table 8.5 DMAC Module Settings

Directory	Target	Setting & Description
Smart Configurator >> Components >> r_dmaca_rx		Default setting (no change)

Table 8.6 RSPI Module Settings

Directory	Target	Setting & Description
Smart Configurator >> Components >> r_rspi_rx		Default settings except for the following changes.
Configurations		
	Dummy data of reception (RSPI_CFG_DUMMY_TXDATA)	Changed to 0x00
Resource >> RSPI		
	RSPI0	Check
	RSPCKA Pin	Use: check
	MOSIA Pin	Use: check
	MISOA Pin	Use: check
Smart Configurator >> Pins >> Serial Peripheral Interface >> RSPI0		Uncheck except for the following settings.
	RSPCKA	Use: check Pin assignment: Set to PC5
	MOSIA	Use: check Pin assignment: Set to PC6
	MISOA	Use: check Pin assignment: Set to PC7

Table 8.7 SCI Module Settings

Directory	Target	Setting & Description
Smart Configurator >> Components >> r_sci_rx		Default settings except for the following changes.
Configurations		
	Use ASYNC mode (SCI_CFG_ASYNC_INCLUDED)	Change to Not (0)
	Use SSPI mode (SCI_CFG_SSPI_INCLUDED)	Change to Include (1)
	Byte value to transmit while clocking in data in SSPI mode (SCI_CFG_DUMMY_TX_BYTE)	Change to 0x00
Resource >> SCI		
	SCI1	Check
	SCK1 Pin	Use: check
	RXD1/SMISO1/SSCL1 Pin	Use: check
	TXD1/SMOSI1/SSDA1 Pin	Use: check
Smart Configurator >> Pins >> Serial Peripheral Interface >> SCI1		Uncheck except for the following settings.
	SCK1	Use: check Pin assignment: Set to P17
	RXD1	Use: check Pin assignment: Set to P15
	TXD1	Use: check Pin assignment: Set to P16

Table 8.8 Touch QE Module Settings

Directory	Target	Setting & Description
Smart Configurator >> Components >> rm_touch_qe		Default settings except for the following changes.

Table 8.9 CTSU Module Settings

Directory	Target	Setting & Description
Smart Configurator >> Components >> r_ctsu_qe		Default settings except for the following changes.
Resource >> CTSU		
	TSCAP Pin	Use
	TS1 Pin	Use
	TS5 Pin	Use

Table 8.10 emWin Module Settings

Directory	Target	Setting & Description
Smart Configurator >> Components >> r_emwin_rx		f you do not use QE for Display, please make the following settings. Default settings except for the following changes.
Configurations		
	Work area size for GUI (EMWIN_GUI_NUM_BYTES)	Set to 256000 (Default value of 100 KB plus 150 KB, the size required by the cache function.)
	Horizontal LCD size (EMWIN_XSIZE_PHYS)	Set to 240
	Vertical LCD size (EMWIN_YSIZE_PHYS)	Set to 320
	LCD orientation (EMWIN_DISPLAY_ORIENTATION)	Set to ORIENTATION_CCW
	LCD interface (EMWIN_LCD_IF)	LCD_IF_RSPI
	Select LCD Driver IC (EMWIN_LCD_DRIVER_IC)	LCD_DRV_IC_ILI9341
	Communication baud rate of LCD interface (EMWIN_LCD_BAUDRATE)	30000000
	Use or unuse display cache (EMWIN_GUI_USE_CACHE)	Use: Check
	Display Signal Pin (EMWIN_DISP_SIGNAL_PIN)	GPIO_PORT_5_PIN_0
	Backlight Pin (EMWIN_BACKLIGHT_PIN)	GPIO_PORT_5_PIN_2
	Data/Command Pin (EMWIN_DATA_CMD_PIN)	GPIO_PORT_5_PIN_1
	Chip Select Pin (EMWIN_LCD_CS_PIN)	GPIO_PORT_C_PIN_3
	Touch interface (EMWIN_TOUCH_IF)	TOUCH_IF_SCI_SPI
	Touch interface channel number (EMWIN_TOUCH_IF_NUMBER)	Set to 1
	Communication baud rate of touch interface (EMWIN_TOUCH_BAUDRATE)	2000000
	Use Touch IC Reset Pin (EMWIN_USE_TOUCH_IC_RESET_PIN)	Not use Touch IC Reset Pin: Uncheck
	Touch Chip Select Pin (EMWIN_TOUCH_CS_PIN)	GPIO_PORT_1_PIN_4

8.2 QE for Display

The settings for QE for Display are shown in Figure 8.1.

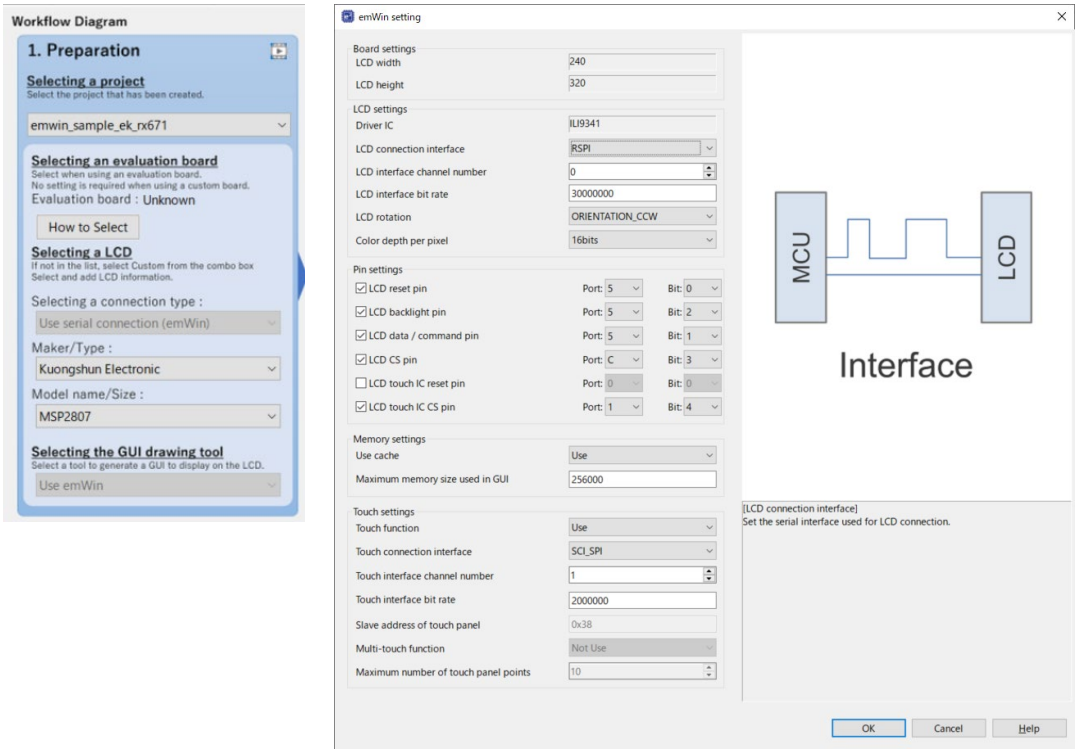


Figure 8.1 QE for Display Setting

8.3 QE for Capacitive Touch

QE for Capacitive Touch settings are shown in Figure 8.2. The adjustment results are reference data based on the board on which the operation was verified. If the touch keys do not respond on your board, please readjust the touch keys.

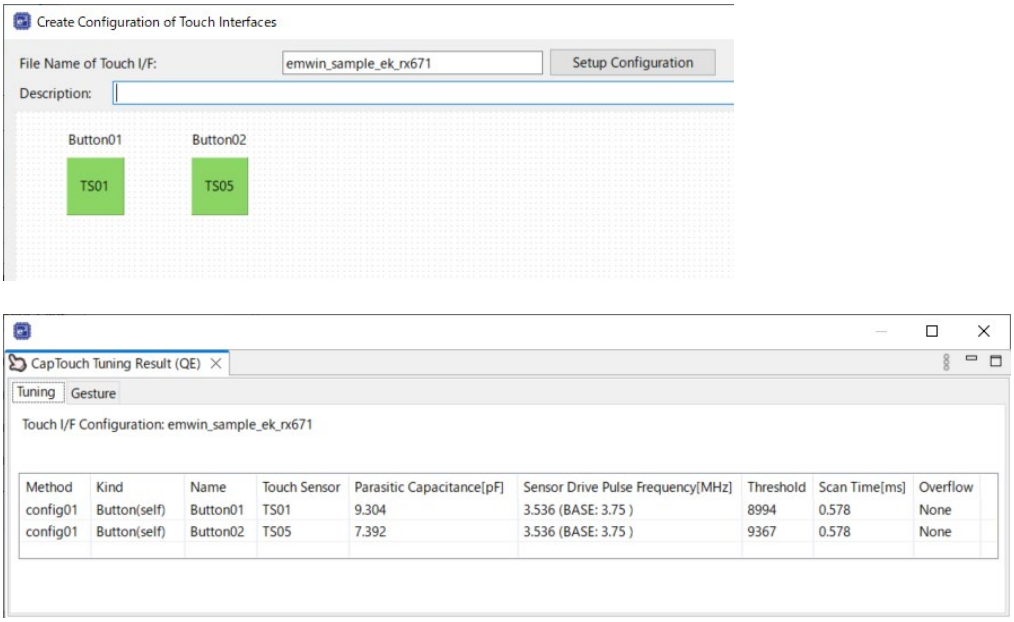


Figure 8.2 QE for Capacitive Touch Setting

9. Reference Documents

User's Manual: Software

- emWin Graphic Library with Graphical User Interface User Guide & Reference Manual
  (https://www.segger.com/downloads/emwin/UM03001)
- RX Smart Configurator User Guide: e^2 studio Edition (R20AN0451)

User's Manual: Hardware

- RX671 Group User's Manual: Hardware Edition (R01UH0899)
- EK-RX671 User's Manual (R20UT5234)
- EK-RX671 CPU Board Circuit Diagram (R20UT5233)

(Please obtain the latest version of each device from the Renesas Electronics website.)

Technical Update / Technical News

(Please obtain the latest information from the Renesas Electronics website)

User's Manual: Development Environment

RX Family CC-RX Compiler User's Manual (R20UT3248)

(Please obtain the latest version from Renesas Electronics website)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	June.30.23	-	First Edition

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.