RX671 Group, RX130 Group

External Serial LCD Display Demonstration using RX Family and Segger emWin Library

Introduction

This document describes how to use emWin library to develop display application using external serial LCD controller with 32-bit RX MCU family.

Target Device

RX671 Group

RX130 Group

Operation Check Tool

Target Board for RX130

Target Board for RX671

Contents

1. Overview of Demonstration

1.1. Demonstration Overview

Figure 1-1 shows the overview of the demonstration.
In this demonstration, the LCD display is realized by connecting the PMOD LCD module, which is included in RX Renesas Starter Kit+, to Target Board for RX130 or RX671.

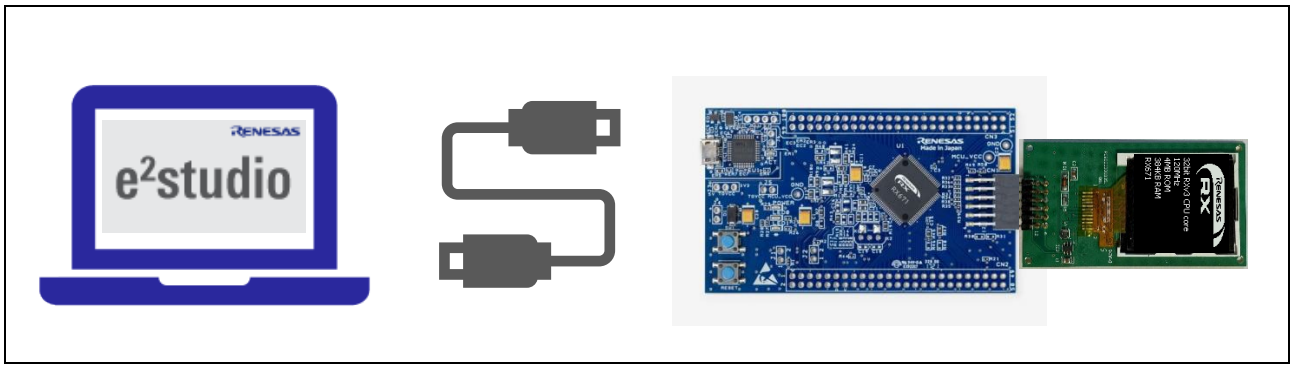


Figure 1-1 Demonstration Overview

1.2. Project Structure

Figure 1-2 shows the project structure of the demonstration. The Application folder contains executed routine in Src folder and C-converted display data in Images folder. The GUI and Config folders contain emWin library and its configurations.

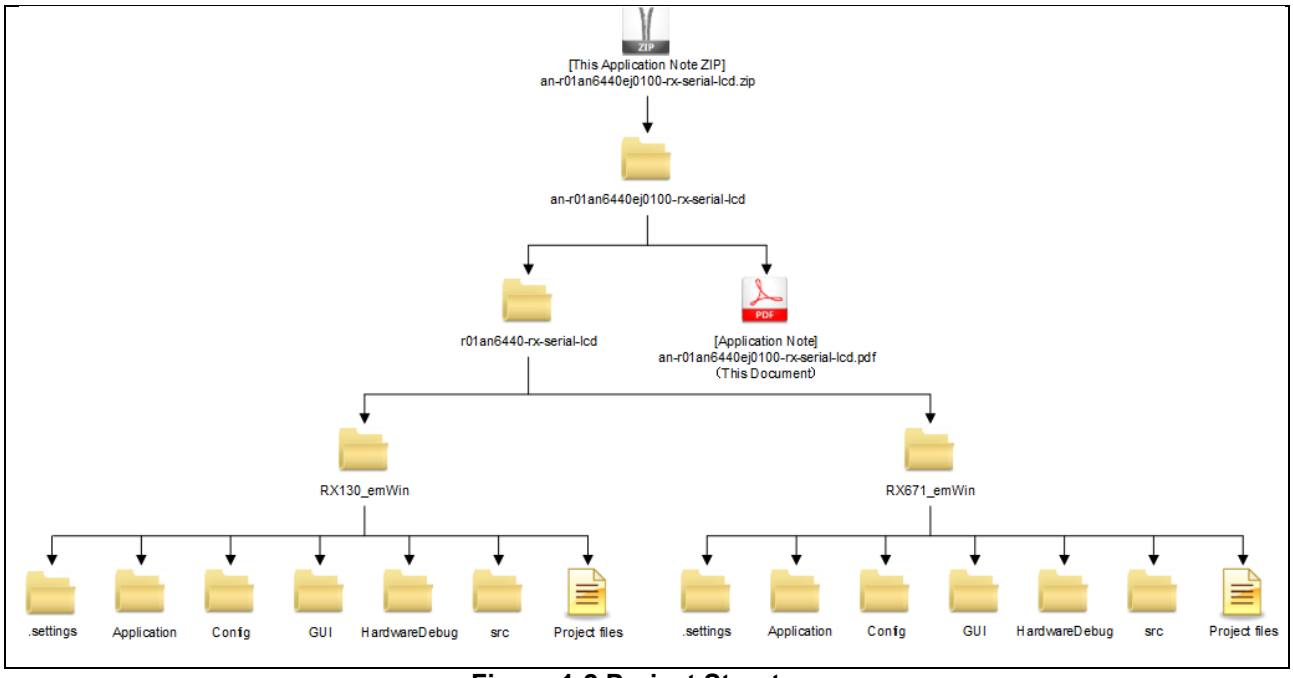


Figure 1-2 Project Structure

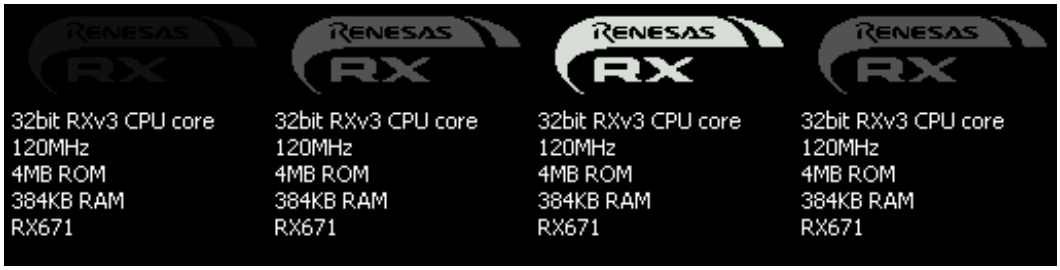
1.3. Demonstration Operation

This project provides 4 types of the demonstrations.

1.3.1 Intro Demo

The intro shows a flashing Renesas logo and some information about the Target Board the application is running on.

Below is the example of the demonstration running on the Target Board for RX671.



1.3.2. Moving Renesas Logo

The second part of the demonstration consists of only a single window but animates to properties of the window.

1.3.3. Slide Show

The third part of the demonstration is a slideshow of six different images.

1.3.4. Progress bar

The last part of the demonstration shows an animated progress bar and a value ranging from 0 to 100.

2 Hardware Preparation

2.1. Constitution of the Hardware

Table 2.1 shows hardware used in this demo.

Table 2.1 Hardware List

Item	Provider	Description
Target Board for RX671	Renesas Electronics Corporation	Evaluation board equipped with RX671
Target Board for RX130	Renesas Electronics Corporation	Evaluation board equipped with RX130
Pmod LCD module	Renesas Electronics Corporation	PMOD-LCD module included in the RX Renesas Starter Kit

2.2. Hardware Modification

2.2.1. Target Board for RX130

The PMOD connector is not fitted on Target Board for RX130 on shipment. Therefore, users need to solder a PMOD connector to run the demo.

2.2.2. Target Board for RX671

Target Board for RX671 supports Pmod Type 6A by shipment default. However, the LCD uses SPI communication, so we must modify the board by removing SS13 and SS14 connection and shorted SC1 and SC2. Please refer to Table 5-1 of Target Board of RX671 User's Manual for more detail.

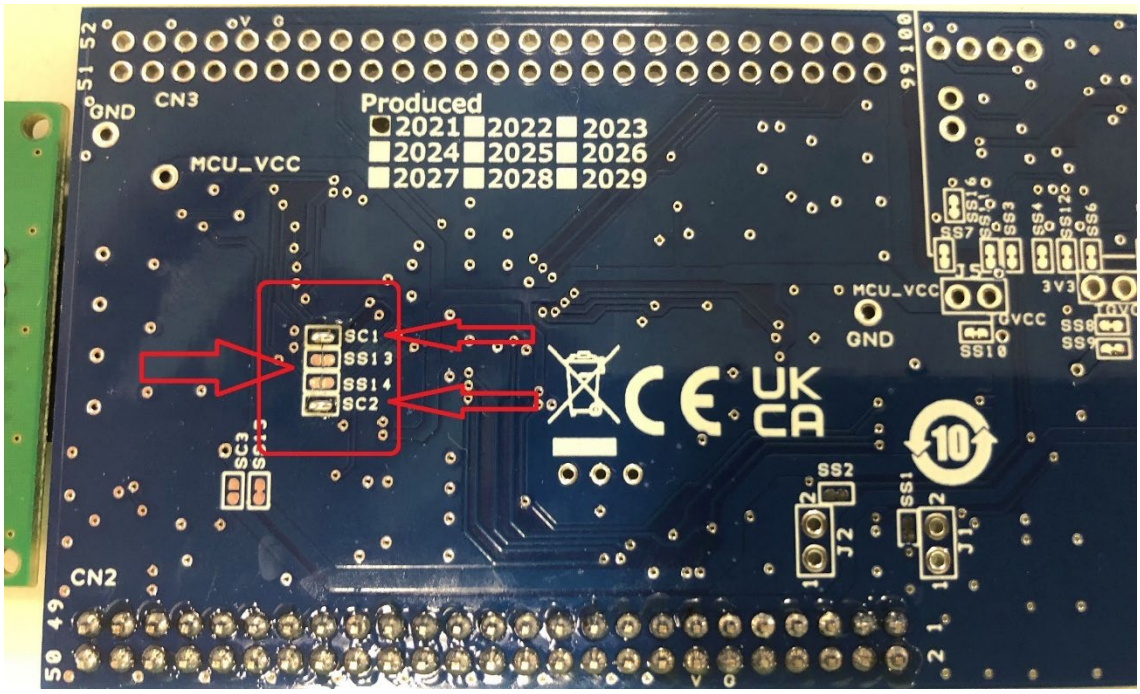


Figure 2-2 RX671 Target Board Modification

2.3. How to Run the Demonstration

The demonstration packages for Target Board for RX130 or RX671 containing a project for e² studio. To run the demonstration, you may follow the steps below.

1. Start e² studio.
2. Once e² studio is started go to "File → Import…".

3. Select “General → Existing Projects into Workspace” and click on “Next”.
4. In the “Import Projects” dialog and browse to the root directory of the project, e.g. “C:\RX130_emWin\”.
5. Under “Projects” the project should be listed now. Make sure the checkbox is checked.
6. Click “Finish”. Now the project should be shown in the “Project Explorer” of e² studio.
7. Right click on the project and select “Build Project”. The “Smart Configurator” generates the missing source files now and the project will be compiled.
8. Connect the RX130/RX671 Target Board with a micro USB cable to the host PC.
9. Right click on the project in the project explorer and select “Debug As → Renesas GDB Hardware Debugging”.
10. Wait till the applications has been downloaded to the target board.
11. Press F8 to let the application run to main(), press F8 again to let run freely.

3. Testing Conditions of the Operation

3.1. Testing Conditions of the Operation

The sample code of this application has been confirmed to work under the conditions shown in Table 3.1

Table 3.1 Testing Conditions of the Operation

Items	Conditions
MCU	Target Board for RX671: • R5F5671EHDFP 100-pin LFQFP • Internal ROM 2MB, Internal RAM 384KB
	Target Board for RX130: • R5F51308ADFP 100-pin LFQFP • Internal ROM 512KB, Internal RAM 48KB
Power supply	USB connecter: 5V input
IDE (Integrated Development Environment)	Renesas e² studio 2022-01
C compiler	CC-RX V3.04.00
Debugger	On-board E2-Lite
Version of the sample code	Rev.1.00

3.2. Memory Size

The ROM and RAM size used of RX130 in this sample code are shown in Table 3.2 and Table 3.3 respectively (At optimization level 2)

Table 3.2 ROM Size

Size (KB)	Description.
110	Program, emWin library, Renesas drivers
194	Resources, images
26	Other
Total: 330KB	

Table 3.3 RAM Size

Size (KB)	Description.
30	emWin memory pool
1	Stack
1	Initialized data
3	Unitialized data
Total: 35KB	

The ROM and RAM size used of RX671 in this sample code are shown in Table 3.4 and Table 3.5, respectively (At optimization level 2)

Table 3.4 ROM Size

Size (KB)	Description.
116	Program, emWin library, Renesas drivers
194	Resources, images
27	Other
Total: 337KB	

Table 3.5 RAM Size

Size (KB)	Description.
200	emWin memory pool
5	Stack
1	Initialized data
4	Unitialized data
Total: 210KB	

4. Software

4.1. Software Hierarchy

Figure 4-1 shows the software hierarchy of this demonstration.

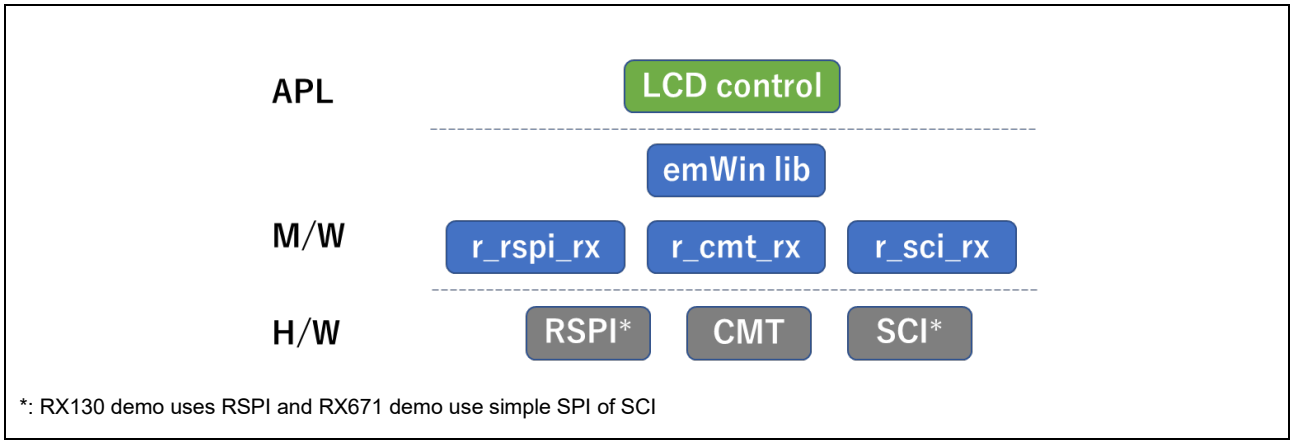


Figure 4-1 Software Hierarchy

4.2. Flowchart

Figure 4-2 shows the flow of a typical emWin application. All emWin example application start with the function MainTask(). This makes it easy to change the GUI application. The MainTask() can be called either directly from main() or started as a dedicated task.

First function being called in MainTask() is GUI_Init() which initializes emWin and the hardware interface like SPI. Also a timer is created which provides emWin with a time base. A time base is important because no dynamic operations would work without it (e.g. animations).

After GUI_Init() the GUI gets created with the emWin API.

At the end of MainTask() an endless loop is executed which periodically manages touch input, redrawing of windows as well software timers. This loop keeps emWin “alive”.

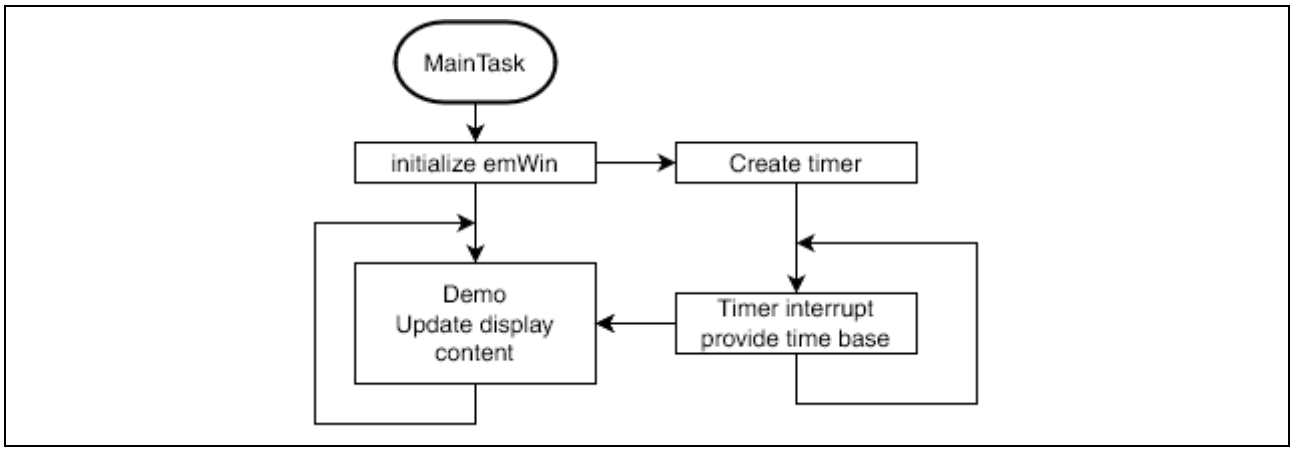


Figure 4-1 Flowchart

4.3. LCD Configuration

This section gives a brief overview of the emWin configuration and the interface to the LCD controller.

The only configuration files which need to be modified by the user are the following:

- GUIConf.c
- GUIConf.h
- LCDConf.c
- LCDConf.h

4.3.1. GUIConf.c

In this file, the memory is assigned to emWin. emWin will allocate memory only from the pool assigned here. The size of the memory pool depends a lot on the scope of the application.

On the Target Board of RX671, the driver uses a cache. The memory for this cache will also taken from the pool assigned to emWin. So, make sure this pool is large enough.

4.3.2. GUIConf.h

As this file offers only compile time relevant options it is not further described here.

It is recommended to NOT change this file when using a pre-compiled emWin version. Changes to this file might affect header files but do not affect already built parts of emWin. Changing this file might lead to unpredictable behavior.

4.3.3. LCDConf.c

Within this file the interface of emWin to the display controller, as well as the emWin driver is set up.

In LCD_X_Config() the emWin driver is configured and function pointers for the communication are set to the driver. The communication is done via an SPI interface and uses the SPI driver set by Renesas.

4.3.3.1. Creating a New Driver Instance

The display used for the Target Board for RX130 and Target Board for RX671 uses a Sitronix ST7715 display controller. As the video memory of the controller is not directly addressable, an emWin driver for an indirectly accessible frame buffer has to be used.

In this case it is the GUIDRV_FlexColor driver which already supports the ST7715. More information about the GUIDRV_FlexColor driver can be found here:

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

The first step is to initialize a driver instance for layer 0 (you could also drive a second display as layer 1 connected to another SPI channel). The driver instance is created by calling the following:

```
GUI_DEVICE_CreateAndLink(GUIDRV_FLEXCOLOR, GUICC_565, 0, 0);
```

The first parameter sets the proper driver API, here it is GUIDRV_FLEXCOLOR.

With the second parameter the color format is selected, named color conversion. The correct color format depends on the LCD controller and in which order it “expects” the colors. In this case we set the format to the emWin color conversion GUICC_565 which corresponds to 16bit BGR.

The third parameter should be 0 and the last one is the layer index, in this case also 0.

Please also take a look at the API description of GUI_DEVICE_CreateAndLink() which can be found here:

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

4.3.3.2. Set Layer Size

The next step is to to set a proper layer size. Typically this is the physical display size in pixel.

Depending on the selected display orientation x- and y-size have to be swapped.

4.3.3.3. Configure The Driver

As the GUIDRV_FlexColor driver is used for indirectly accessible display controllers, the GUIDRV_FlexColor driver needs to know how to communicate with the controller.

This is done by setting a couple of functions in a GUI_PORT_API structure to read and write data as well as commands to the display controller.

An 8 bit SPI interface should be used in this case, so only the 8-bit-related function pointers need to be set:

```
PortAPI.pfWrite8_A0  = _Write8_A0;
PortAPI.pfWrite8_A1  = _Write8_A1;
PortAPI.pfWriteM8_A0 = _WriteM8_A0;
PortAPI.pfWriteM8_A1 = _WriteM8_A1;
PortAPI.pfRead8_A0   = _Read8_A0;
PortAPI.pfRead8_A1   = _Read8_A1;
PortAPI.pfReadM8_A0  = _ReadM8_A0;
PortAPI.pfReadM8_A1  = _ReadM8_A1;
```

This structure is used for setting function pointers for the GUIDRV_FlexColor and for selecting the proper mode:

```
GUIDRV_FlexColor_SetFunc(pDevice, &PortAPI, GUIDRV_FLEXCOLOR_F66709,
GUIDRV_FLEXCOLOR_M16C1B8);
```

The first parameter is a pointer to the driver instance and the second one is a pointer to the previously initialized GUI_PORT_API structure.

The third parameter selects a function set of the GUIDRV_FlexColor driver which matches the specifications of the ST7715 controller. In the emWin user manual you will find a list with all supported display controllers and the correct function sets.

The last parameter selects the interface type, the desired color depth and enables a possible display cache. GUIDRV_FLEXCOLOR_M16C1B8 can be read as follows:

M16 – 16 bit mode

C1 – Display cache enabled (C0 would be disabled)

B8 – 8 bit interface

More information can be found here:
https://www.segger.com/doc/UM03001_emWin.html#GUIDRV_FlexColor_SetFunc

The last step is to set a configuration. In this case the display orientation is set as well as the start and end position of the video memory. If the x- and y-axis are being swapped, the start and end positions need to adapted.

```
  Config.Orientation = DISPLAY_ORIENTATION;
  Config.FirstCOM = (DISPLAY_ORIENTATION & GUI_MIRROR_Y) ? 3 : 1;
  Config.FirstSEG = 2;
  if (DISPLAY_ORIENTATION & GUI_SWAP_XY) {
    Temp = Config.FirstCOM;
    Config.FirstCOM = Config.FirstSEG;
    Config.FirstSEG = Temp;
  }
  GUIDRV_FlexColor_Config(pDevice, &Config);
```

Please refer to the manual for further information:
https://www.segger.com/doc/UM03001_emWin.html#GUIDRV_FlexColor_Config

4.3.3.4. Display Driver Callback Function

The function LCD_X_DisplayDriver() is a callback function which is called by the driver for several operations. Here, we react only on following commands:

LCD_X_INITCONTROLLER – Sent when it's time to initialize the LCD controller

LCD_X_ON – Sent to turn on the display

LCD_X_OFF – Sent to turn off the display

When reacting on LCD_X_INITCONTROLLER, the interface used to communicate with the display is set up. Once this is done the LCD controller has been initialized. These steps have to be done by the user, the display controller initialization is not part of the emWin code. In this case though, this has already been done.

A full list of commands can be found here:

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

4.3.4. LCDConf.h

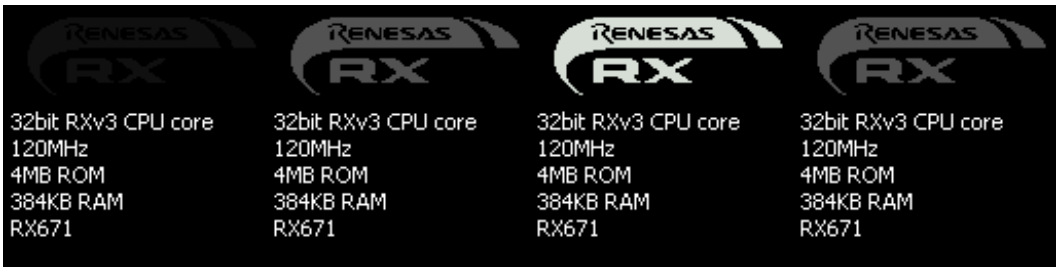
This file is empty in most cases. It can be used to make user functions located in LCDConf.c available to other modules.

4.4. Demo Configuration

Below you will find some short descriptions of the individual parts of the demonstration application running on the Target Board for RX130 and Target Board for RX671.

This description will not cover every single aspect of the demonstration, but will add some hints on some specific topics like setting up an animation or sending custom messages to a window.

4.4.1. Intro



The intro shows a flashing Renesas logo and some information about the target board the application is running on.

This demo consists of two windows. One is used as parent window, covering the entire screen and a child window to display the Renesas logo.

4.4.1.1. Parent Window

At first the parent window is created. The callback function of the parent window _cbPowerOn() reacts on two messages.

WM_CREATE will be send to a window upon creation and allows the user to react on the window creation. Within this event a second window is created as child of the current window. The second window is used to show the flashing logo.

WM_PAINT is sent to a window when the window should be drawn. In this case the window will clear its content with black color and draw some description text about the current device. Please note, all positioning information used here are relative to the windows x0 and y0 position.

4.4.1.2. Child Window

The child window will be created with the callback function _cbLogo(). Within this callback function an animation is created upon the WM_CREATE event and a bitmap is drawn on WM_PAINT.

The animation is used to animate a value from 0 to 255. This value is used to create gray shades from 0xFF000000 to 0xFFFFFFFF.

At first the user has to define a structure which contains some information about the current animation progress as well as some data used for the animation.

The structure looks like this:

```
typedef struct {
  WM_HWIN hWin;
  U32     Value;
  U32     MaxValue;
  int     Dir;
} ANIM_DATA_VALUE;
```

hWin is set to the current window handle. In this case the handle of the window is used to display the logo. With the handle, it is easy to trigger a redraw of this window.

Value this member is used to calculate a new color in the WM_PAINT event of the window.

MaxValue is the maximum value calculated by the animation. Here it is set to 0xFF.

Dir is used to indicate a direction. 0 runs towards 0xFF and 1 runs back to 0x00.

The animation gets created with a pointer to an animation callback function shown below:

```
static void _Animate255(GUI_ANIM_INFO * pInfo, void * p) {
  ANIM_DATA_VALUE * pData;
  pData = (ANIM_DATA_VALUE *)p;
  pData->Value = (pData->MaxValue * pInfo->Pos) / GUI_ANIM_RANGE;
  if (pData->Dir) {
    pData->Value = pData->MaxValue - pData->Value;
  }
  if (pInfo->State == GUI_ANIM_END) {
    pData->Dir ^= 1;
  }
  WM_InvalidateWindow(pData->hWin);
}
```

In this callback function we calculate a value depending on the position (pInfo->Pos) on the time line of the animation. The position ranges from 0 to 32767 (GUI_ANIM_RANGE).

After calculating a value, a redraw of the window is triggered by calling WM_InvalidateWindow(). This will cause the Window Manager to redraw the given window on the next call of GUI_Exec().

The WM_PAINT event of _cbLogo() redraws the logo with a color calculated by the value from the animation. The code below does this:

```
    GUI_SetBkColor(GUI_BLACK);
    GUI_Clear();
    Color = (0x010101 * AnimData.Value) | 0xFF000000;
```

```
GUI_SetColor(Color);

GUI_DrawBitmap(&RENESAS_RX_LOGO, 4, 2);
```

A special characteristic of 1bpp is used to draw the bitmap. If a 1bpp emWin bitmap is saved without a palette, the currently set fore- and background colors are used to draw the bitmap. In this case GUI_BLACK is used as the background color and the calculated color is used as the foreground color.

4.4.2. Moving Renesas Logo

The second part of the demo consists of only a single window but animates to properties of the window.

At first a window is created to draw the Renesas logo. It has the size of the logo bitmaps and will be moved up and down on the screen while changing the color of the logo. The callback function you might want to refer to is named _cbLogoColor().

The animation consists of three animation items starting at different points of the overall length of the animation.

The first animation item starts with the animation and runs for the entire duration of the animation. This item is used to calculate an index from 0x00 to 0xFF.

The other two items are used to set the y-position of the window. The first item starts when the animation starts and ends in the middle of the overall duration. This animation item moves the logo down.

The third item starts at the middle of the animation and last till its end. This one is used to move the logo up.

Within the WM_PAINT event of _cbLogoColor() the logo is drawn. The same characteristic already used in the first part of the demo is used again, but this time the logo is drawn using colors from a color gradient.

Instead of calculating a color an index from 0x00 to 0xFF, calculated by the first animation item, is used to access an array of already defined colors. This array represents a gradient with 256 colors.

4.4.3. Slide Show

The third part of the demonstration is a slideshow of six different images.

To show the images, a single window is created using the callback function _cbSlide().

On WM_CREATE a timer is created and attached to this window by calling WM_CreateTimer(). Every time the timer expires it sends a WM_TIMER message to the window it is attached to.

Within the WM_TIMER event an index is incremented and the window will be marked as invalid to trigger a redraw. After showing the last image, the window will delete itself.

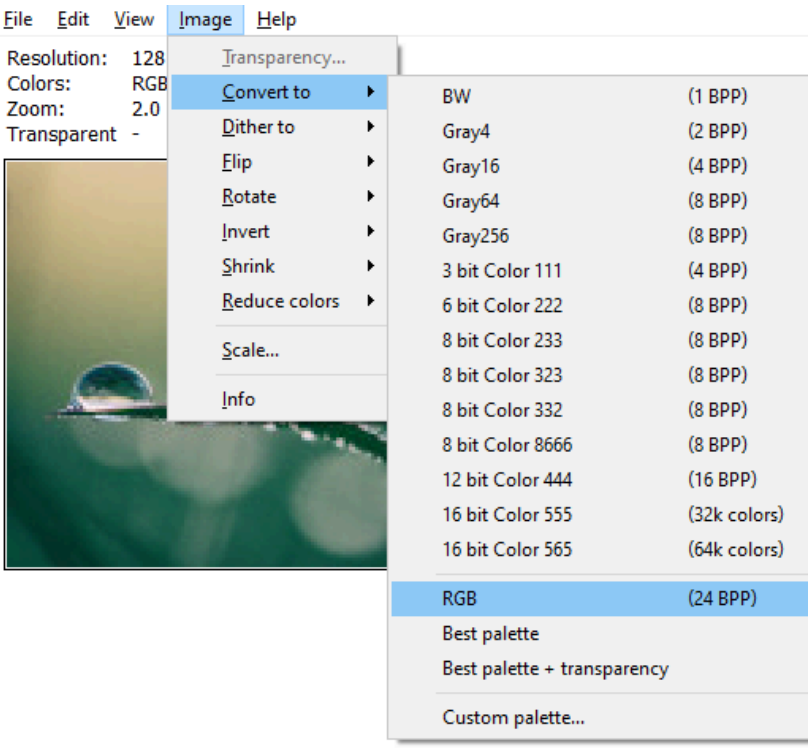
On WM_PAINT the window draws an image depending on the index set in WM_TIMER.

The images shown in this part of the demo are stored as 16bpp bitmaps. This offers the best quality as these bitmaps have the some color depths as used on the display. The downside is a higher memory requirement.

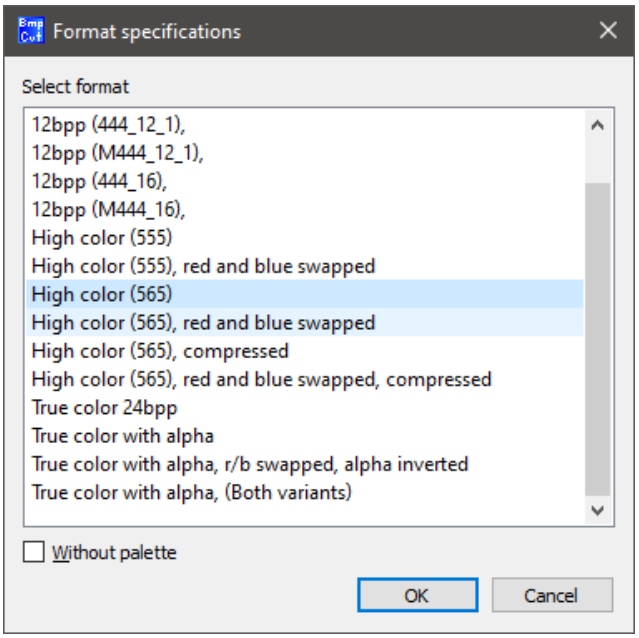
The best drawing performance is always achieved by using the bitmap format with the same color settings as the driver uses. If both formats are matching, no conversion is required when writing pixel data to the display.

In this case the driver uses a 16bpp BGR color format (GUICC_565), therefore the bitmap should be stored also in 16bpp BGR. The corresponding color format is GUI_DRAW_BMP565.

The images are converted with the Bitmap Converter. Before an image is saved, it has to be made sure that no alpha channel is present. Simply remove the alpha channel by changing the color space to RGB only as shown below:



When saving the image as a C-file, select “High Color (565)” as the desired format.



4.4.4. Progress Bar

The last part of the demonstration shows an animated progress bar and a value ranging from 0 to 100.

The screen is composed of three windows. One used as parent window covering the entire screen and displaying the static text “Progress”. One TEXT widget used to display the value. And a third one to show the moving progress bar.

The parent window clears the background and displays the static text.

A TEXT widget is used to display the dynamic string. This widget is used as it makes it easier to update only the required area, which is the widget’s area. A widget is a predefined window object offering specific functionality. As most widgets rely on user interaction, they were not applicable in this demo. Please refer to the manual for further information on widgets.

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

The third window is used to display the progress bar.

On WM_CREATE of the parent window, the other two windows are created. The TEXT widget does not receive a specific callback function, as it uses the default one. After creating the TEXT widget some properties are set, such as colors, font and alignment.

The window which shows the progress bar uses the callback function _cbProgbar() which defines the look of the progress bar.

A simple animation is also set up at this point which counts a value from 0 to 100.

Every time a new value is calculated by the animation, a custom message without data is send to the parent window callback. This is done by calling WM_SendMessageNoPara(). A custom message can be defined as shown below. Just make sure it is out of the range of the messages predefined by emWin, as this would lead to undefined behavior.

```
#define MSG_NEW_ANIM_DATA    (WM_USER + 0x00)
```

The callback function reacts on the message and sets a new string to the TEXT widget. The widget will be automatically invalidated and updated by the Window Manager.

To update the progress bar, a message is sent from the parent window to the progress bar window, only this time with the current progress value as the parameter.

The callback function _cbProgbar() reacts on the sent message MSG_NEW_ANIM_DATA and stores the attached data in a static variable. After that, the progress bar window invalidates itself to trigger a redraw with the new value.

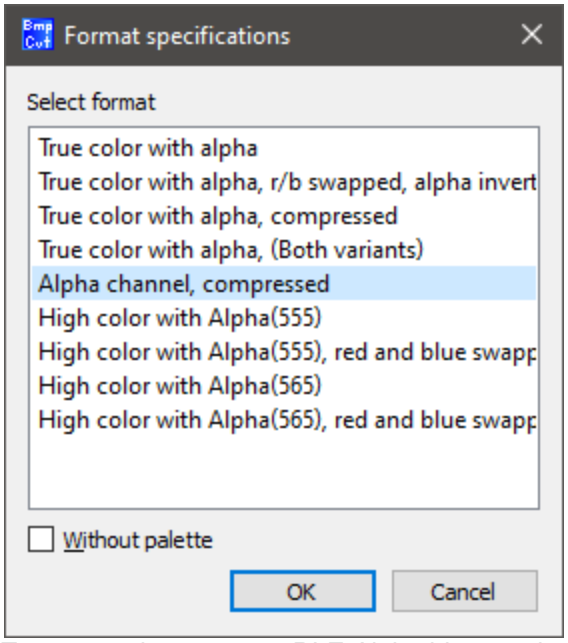
At first, a rectangle with the window dimensions is retrieved when reacting on WM_PAINT in _cbProgbar(). This rectangle is used to fill the progress bar with white. Now the rectangle is reduced by two pixel in each direction and a second rectangle in black is drawn into the progress bar. This results in a white frame and any possible content within this region has been cleared.

Now the rectangle is reduced by two pixels again and is used as a clipping rectangle. With a clipping rectangle set, it is not possible to draw outside of this region defined by the clipping rectangle. Here, we don't want the angular shape exceed the right hand side of the progress bar.

After setting the clipping rectangle, the rectangle is modified again by using the progress bar value sent previously via the message. This rectangle is filled with GUI_YELLOW.

To achieve an angular rectangle, an RLE-Alpha bitmap is used. This bitmap format has the functionality to be drawn in any color, as the 1bpp bitmap, but with a transparent background as well as anti-aliasing for soft edges.

An RLE-Alpha bitmap stores only the alpha values of an image with run-length encoding to reduce the memory footprint.



To save an image as an RLE-Alpha bitmap, simply select "Alpha channel, compressed" as the format when saving an image with the Bitmap Converter.

This format can be quite useful for icons as it allows to change the icon color during runtime.

5. Note On Creating an Application

5.1. Notes on the Target Board for RX671

On the Target Board for RX671 a display cache is used to render the display content before sending it to the display. This improves the drawing performance as sending one large buffer is more efficient than sending each pixel on its own. Furthermore, it can help to avoid flickering effects, but this depends also on the interface speed.

5.2. Notes on the Target Board for RX130

When running the demo on the Target Board for RX130 a display cache is not available as this device offers not enough internal RAM. Fortunately, automatic memory devices provided by the Window Manager can be used. A window created with the flag WM_CF_MEMDEV will draw its content into a memory device and send the memory device content to the display in one go. This improves the performance of the drawing operations because larger blocks are being sent to the display.If there is not enough memory to create a memory device with the size of a window, the pixel data will be split into multiple smaller memory devices (banding). Still, with banding the drawing performance is faster than without any kind of cache.

As it is not possible to read the display content on the Target Board for RX130 and no display cache can be used, the memory device offers another advantage. Without a cache or the possibility to read it wouldn't be possible to use drawing operations which require color mixing (e.g. anti-aliasing or semi-transparent bitmaps). This is because the current background color of the transparent pixel has to be known. With memory devices however, it is possible to draw the background into the device and mix the foreground into it. This has to be considered by the user when developing an application and the user has to make sure that no transparent windows are being used and that the background is always defined.

5.3. General notes:

5.3.1. Window Manager

The Window Manager of emWin is commonly used to provide a structure to the screen by splitting it up into smaller parts, each for a specific operation. For example, one window manages the drawing of a text and another window draws an image.

Splitting up the screen content into multiple smaller parts will also improve performance, as only small parts that are independent of each other are required to be redrawn. For example, a counter would only need to redraw a single number instead of the entire screen.

Every window created with the Window Manager needs a callback function which allows it to react on specific events like painting (WM_PAINT), creation (WM_CREATE) or deletion (WM_DELETE). By reacting on such events the user can completely define the window behavior, as well as the window appearance. More information can be found here:

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

To process events, redrawing or possible touch input, it is necessary to keep the Window Manager "alive". This is done by calling GUI_Exec() periodically. Typically this happens in a loop that is commonly called a "super loop".

This demo also uses a super loop to create and delete its single components that can be found in MainTask(). To wait for a single part of the demo to be finished, a macro is used to check the existence of a window and to perform a GUI_Exec():

```
#define WAIT4WIN(hWin)  while (WM_IsWindow(hWin)) { GUI_Exec(); }
```

This macro will return from the while-loop as soon as the window has been deleted.

Instead of GUI_Exec(), the user could also call GUI_Delay(int Delay). GUI_Delay() internally calls GUI_Exec() every 5 ms until the given delay period has been reached.

5.3.2. Windows

The Window Manager is a great tool to keep the screen content organized. To do this, the screen can be split up in several areas by using windows.

To create a window, the following function can be used:

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

The position x0 and y0 are always relative to the parent window. If the parameter hParent is NULL the desktop window is used as parent.

The Style parameter uses an OR-combination of WM_CF_… flags and it should at least contain WM_CF_SHOW. Otherwise, no window will appear on the screen. Please refer to the manual for further create flags.

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

The parameter cb is a pointer to a callback function which defines the behavior and the appearance of the window. Within this callback function the user can react on messages and decide how to handle them (e.g. WM_PAINT to define the look of the window).

A list of messages can be found under the link below:

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

Below is a very simple example of a callback function which fills the window area with a red color. For further examples you can refer to the code of the demo.

```
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 member pMsg->hWin always refers to the window the callback function belongs to. This makes it easy to modify the window or retrieve properties of it.

5.3.3. Images

All images used in the application have been converted using the Bitmap Converter. The Bitmap Converter is a tool which is supplied with emWin. This tool allows the user to convert images into emWin specific formats (bitmaps). Although it is possible to draw common image formats like PNG, JPG or GIF directly, emWin bitmaps offer a better performance. The user also has the option to optimize for a faster drawing speed or for a lower memory footprint depending on the requirements.

Some emWin bitmaps can also be drawn with a color set on runtime so it is not necessary to save a bitmap multiple times in a different color. This can e.g. come in handy for icons.

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

5.3.4. Fonts

With the Font Converter tool, any font installed on the Windows host PC can be converted into an emWin font. emWin fonts can be created with or without anti-aliasing. It is also possible to reduce the amount of characters included in the font to a minimum (e.g. only those characters required by the application). This is

useful when the application uses a language which has typically a high amount of different characters e.g. like Japanese or Chinese.

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

5.3.5. Animations

Simply broken down, animations only increment a value from 0 to 32767 in a specific period of time. This mechanism can be used to animate other values like window coordinates, color spaces or any other value.

An animation is created by calling the function below.

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

Period is time the animation lasts in milliseconds.

MinTimePerSlice is the minimum time between two calls of the animation callback function.

pVoid is a pointer to user defined data, typically a structure holding information about the current animation status.

pfSlice is a pointer to a slice function being called periodically. It can be NULL and is not used in this demo.

After creating an animation the user has to add items to animation. To do so call the following function:

```
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 is the handle of a previously created animation.

ts is the start time of this item within the overall period of the animation.

te is the end time of this item within the overall period of the animation.

pVoid is is a pointer to user defined data.

pfAnim is a pointer to a user defined callback function. It is called every time the minimum time slice has expired.

After adding an item to the animation it can be started by calling GUI_ANIM_StartEx().

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

hAnim is the handle of a previously created animation.

NumLoops is the amount how many times the animation should be executed. If this is -1, the animation will run indefinitely.

pfOnDelete is a pointer to a function which will be called when the animation gets deleted. This can also be NULL.

Below you can find some examples on how to utilize the animation module.

You might also want to refer to the animation chapter of the emWin user manual for further information.

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

Examples

If you are looking for further examples or information you might want to take a look at the emWin Wiki page and the examples page within this Wiki.

https://wiki.segger.com/emWin

https://wiki.segger.com/emWin_Examples

6. Reference Documents

User's Manual:

Target Board for RX671 User's Manual: R20UT4894EJ0100

Target Board for RX130 User's Manual: R20UT4169EJ0101

(Get the latest version from the Renesas Electronics website.)

User's Manual: emWin

UM03001_emWin.pdf

(Get the latest version from the SEGGER website https://www.segger.com/downloads/emwin/)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	May.31.2022	–	First edition issued
-	-	-	-

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 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. 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.4.0-1 November 2017)

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/.