

Renesas RA Family

"GUIX Thermostat" for EK-RA8D1 MIPI Display

Introduction

This application, which is a Thermostat application, provides a reference for developing complex multi-threaded applications with a touchscreen graphical Human Machine Interface (HMI) by using Renesas FSP and Azure RTOS GUIX. It describes steps to create a basic GUIX for FSP, integrates a touch driver, and handles multiple hardware accesses, system updates, and event handling.

This application is developed using the Renesas RA Flexible Software Package (FSP), which provides a quick and versatile way to build secure connected Internet of Things (IoT) devices using the Renesas RA family of Arm microcontrollers (MCUs). RA FSP provides production-ready peripheral drivers to take advantage of the RA FSP ecosystem along with Azure RTOS GUIX library and Azure RTOS.

This application note assumes that you are familiar with the concepts associated with writing multi-threaded applications under a Real-Time Operating System (RTOS) environment, such as Azure RTOS. This application note makes use of RTOS features such as threads and semaphores. Prior experience in using Azure RTOS would be helpful for easy understanding of the provided application project. For more detailed information on Azure RTOS features, refer to the Azure RTOS User Manual.

The Graphics application is developed using the Renesas e² studio Integrated Solution Development Environment (IDE). e² studio is integrated with the FSP platform installer, which can be downloaded from the Renesas website. The intuitive configurators and code generators in e² studio and FSP will help the application developers create complex multi-threaded graphics applications very quickly. This application note walks you through all the necessary steps in creating, building, and running a complex graphics project, including the following:

- Board setup.
- Install tools.
- Build and run applications.
- Azure RTOS GUIX Studio project integration.
- Setup Azure RTOS GUIX Studio project.
- Add touch Driver.
- Create FSP GUIX project.
- Hardware Setup.
- Using the General Purpose Timer to drive a PWM backlight control signal.

Required Resources

Development tools and software

- e² studio IDE Version: 2024-07 (24.4.0) or greater
- Renesas Flexible Software Package (FSP) v5.5.0.
- Azure RTOS GUIX Studio V6.4.0.

Hardware

- Renesas EK-RA8D1 kit (RA8D1 MCU Group)
 - MIPI LCD included.
 - Micro USB cable included.

Reference Manuals

- RA Flexible Software Package Documentation Release v5.5.0
- Azure RTOS GUIX and GUIX Studio v6.4.0.0
- Azure RTOS ThreadX v6.4.0
- Renesas RA8D1 Group User's Manual Rev.1.1.0
- EK-RA8D1-v1.0 Schematics

Purpose

This document will guide you through the setup of an Azure RTOS GUIX touchscreen interface Thermostat application in e² studio. This document will show how to configure the drivers and library included with the FSP. These will allow you to set up the MIPI LCD Display Controller, the LCD's touch screen drivers, and RTOS objects to communicate with application tasks. It also shows the steps necessary to create a simple GUI interface using the Azure RTOS GUIX Studio editor. In addition, this app note will cover project setup along with basic debugging operations. When it is running, the application will respond to touchscreen actions, presenting a basic graphical user interface (GUI).

Intended Audience

The intended audience is users who want to design GUI applications.

Contents

1. Download and Installing Tools	4
1.1 Overview.....	4
1.2 Procedural Steps	4
2. Create the Application Project and Enable Backlight	5
2.1 Overview.....	5
2.2 Procedural Steps	5
3. Using GUIX Widget Timer to Trigger a Screen Transition	21
3.1 Overview	21
3.2 Procedural Steps	21
4. Add Touch Driver to Thermostat_GUIX_EK_RA8D1_MIPi Project	23
4.1 Overview.....	23
4.2 Procedural Steps	23
5. Control MIPI LCD Backlight	32
5.1 Overview	32
5.2 Procedural Steps	32
6. Update Date/Time and Temperature	37
6.1 Overview	37
6.2 Procedural Steps	37
7. Testing and Debugging the Complete Thermostat Project.....	43
7.1 Overview	43
7.2 Procedural Steps	43
8. Website and Support	45
Revision History	46

1. Download and Installing Tools

1.1 Overview

In this section, you will copy the application note (AN) materials to your PC and install e² studio v2024-04/FSP v5.5.0 and Azure RTOS GUIX Studio v6.4.0.0.

1.2 Procedural Steps

1.2.1 If you already have e² studio with FSP v5.5.0 or later installed, you can skip this step. Otherwise, you can download it from this [link](#).

1.2.2 You can get Azure RTOS GUIX Studio v6.4.0.0 or greater from this [link](#). If it goes well, you will see the window in the next step on the web browser.

Note: Microsoft Store needs to work on your PC to install Azure RTOS GUIX Studio.

1.2.3 Click **Download** to local PC and start installing Azure RTOS GUIX Studio.

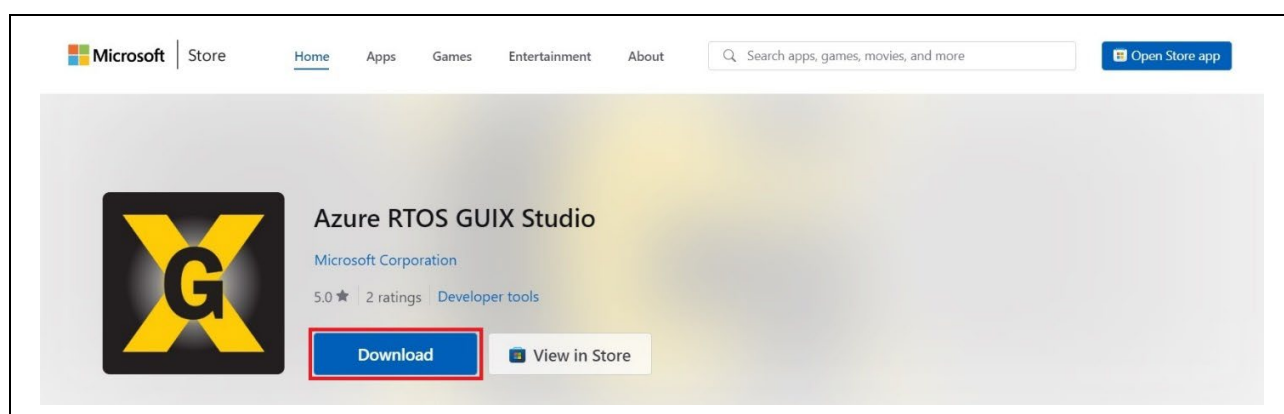


Figure 1. Get Azure RTOS GUIX Studio

1.2.4 Either you will be directed to the Microsoft Store for installation, or a .exe file will begin downloading. If the latter, click the downloaded file “**Azure RTOS GUIX Studio installer.exe**” and continue to install Azure RTOS GUIX studio.

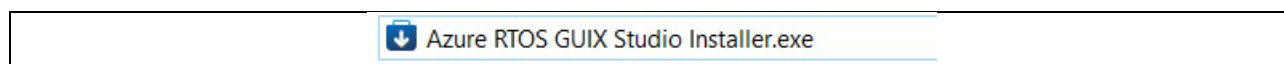
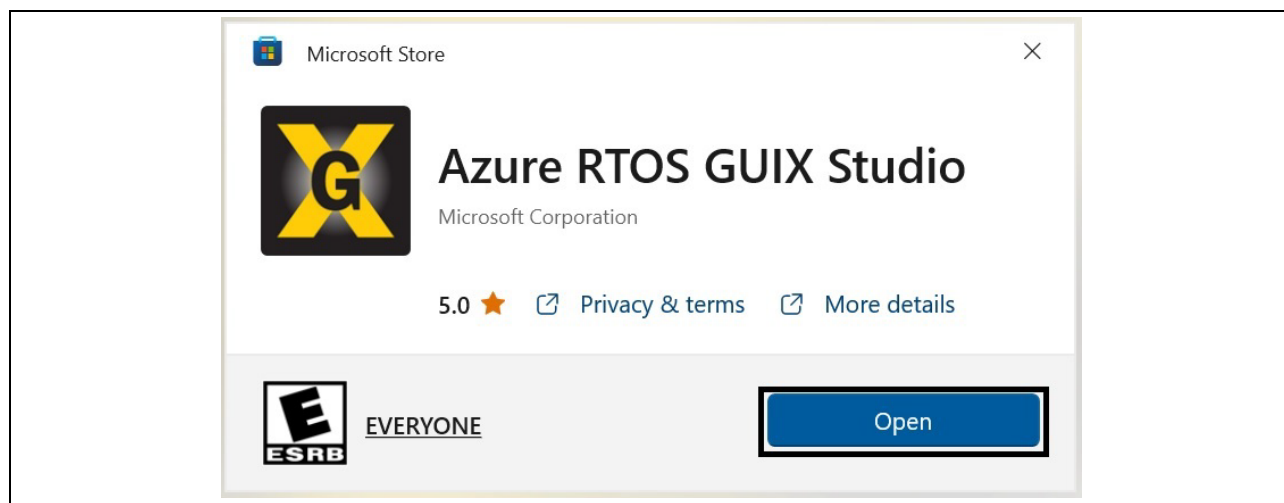


Figure 2. Install Azure RTOS GUIX Studio

1.2.5 Click “Open” to open the “Azure RTOS GUIX Studio” App.



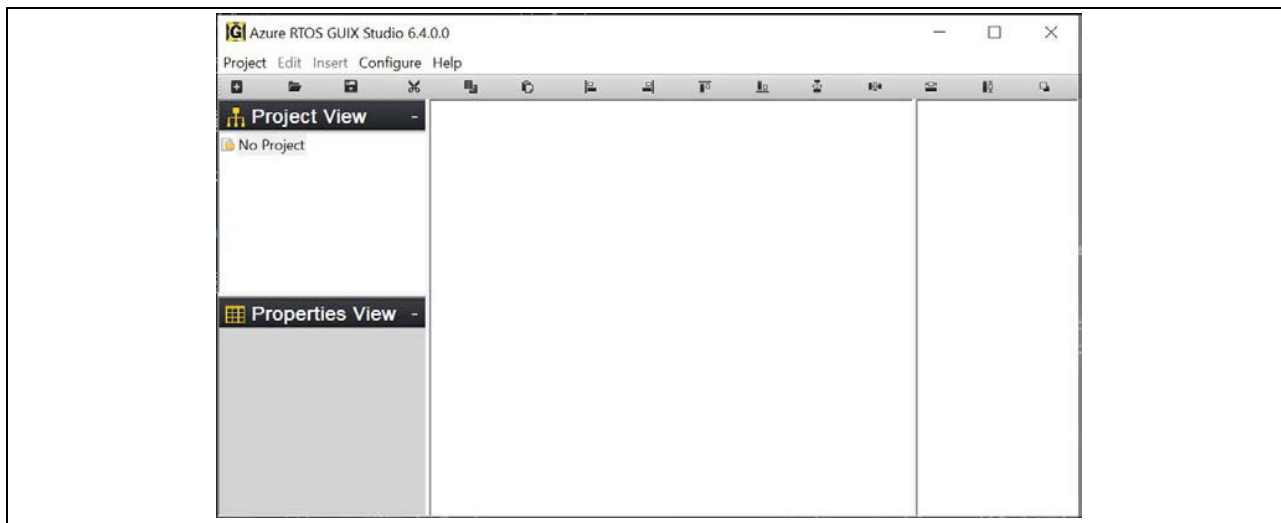


Figure 3. Click Open to Launched “Azure RTOS GUIX Studio”.

1.2.6 Close Azure RTOS GUIX Studio for now; you will need to open it again later.

2. Create the Application Project and Enable Backlight

2.1 Overview

In this section, you will create a project to which you will add pre-written source code and integrate it with a pre-created Azure RTOS GUIX studio project.

2.2 Procedural Steps

2.2.1 Create a new RA C/C++ project. Name it Thermostat_GUIX_EK_RA8D1_MIPI.

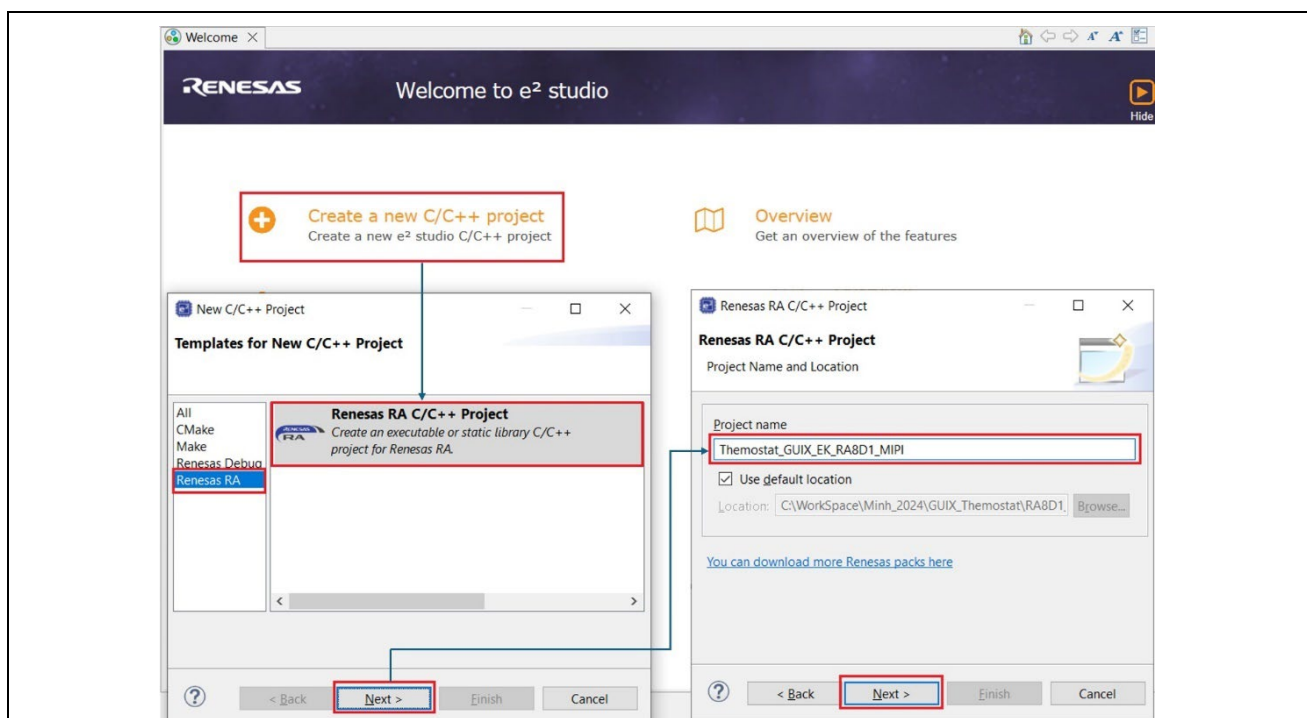


Figure 4. Create New Project

2.2.2 Set the board to EK-RA8D1.

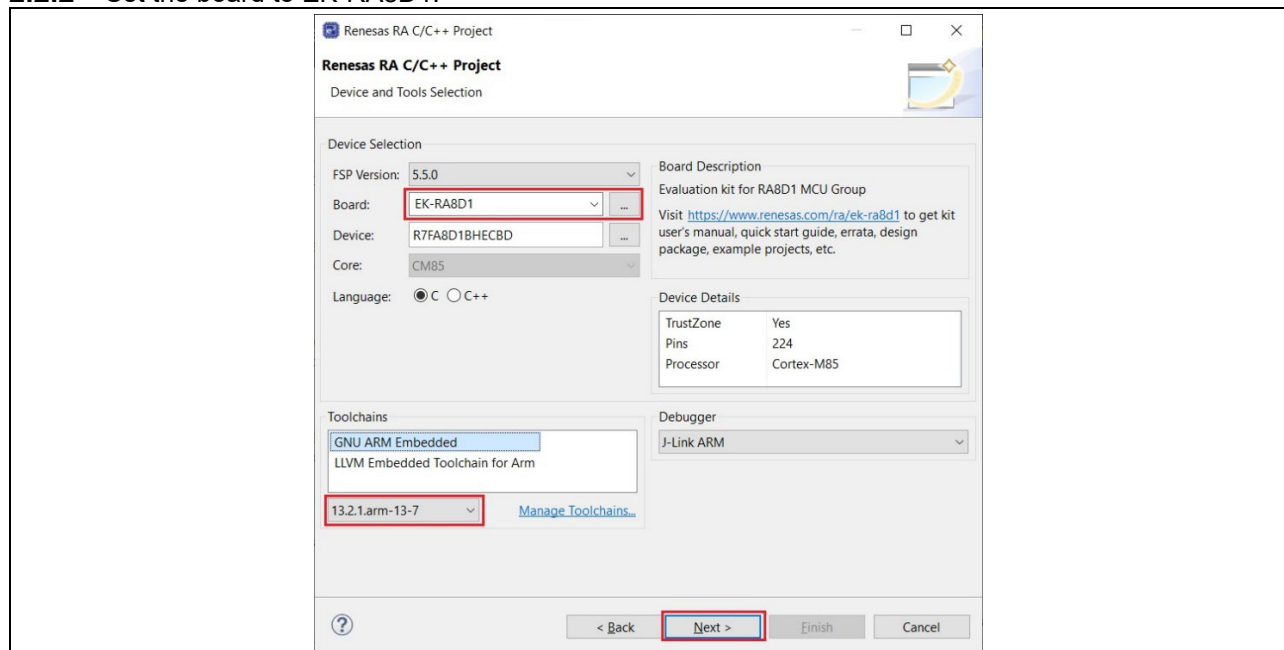


Figure 5. Select and Set board to EK-RA8D1

2.2.3 First, select the Project Type as **Flat** and click Next. Then, set the Build Artifact to **Executable** and the RTOS to **Azure RTOS ThreadX (v6.4.0+fsp5.5.0)** and click Next.

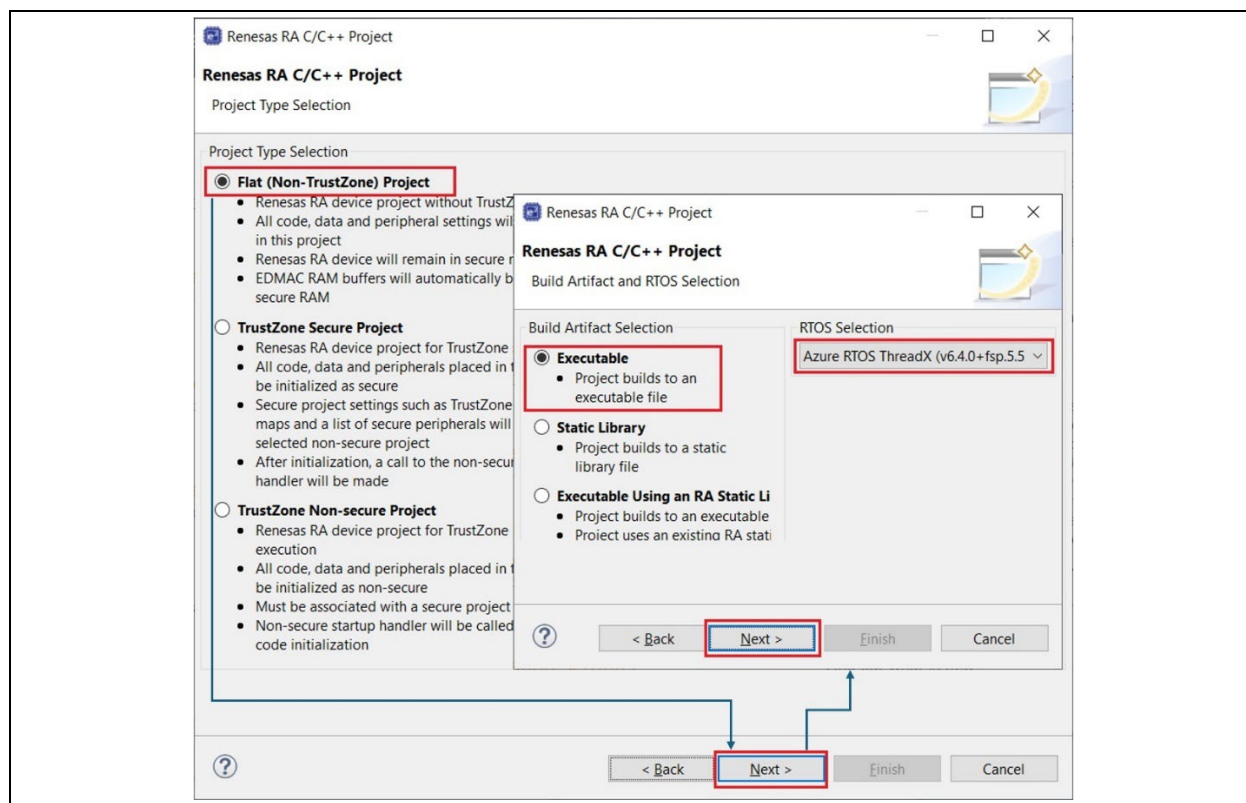


Figure 6. Select Azure RTOS ThreadX

2.2.4 Use Azure RTOS ThreadX - Minimal template and click **Finish**.

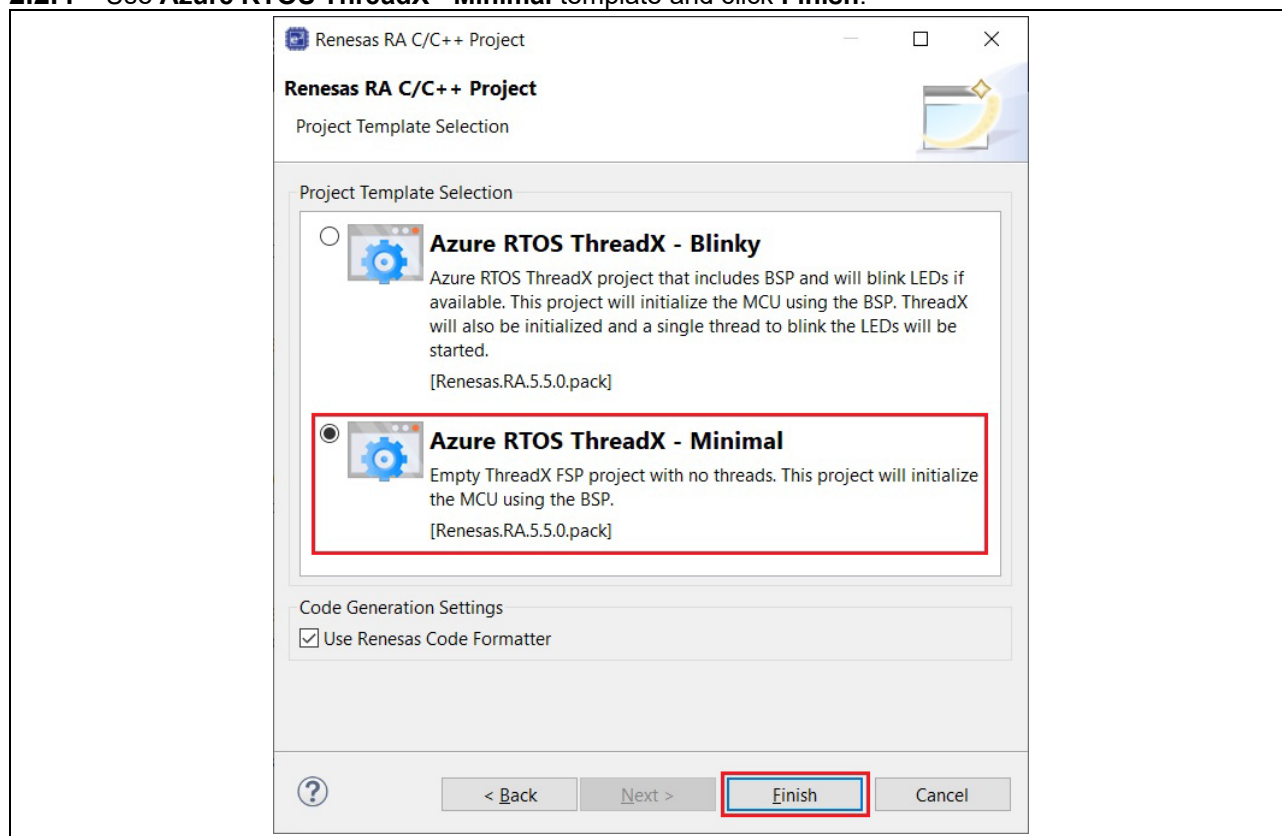


Figure 7. Selecting Azure RTOS ThreadX – Minimal Template

2.2.5 Open the project's configuration.xml file and go to the **BSP** tab. Use the Properties tab while in the FSP Configuration View to view and edit the settings for the **SDRAM** to match the picture below. Also set the **Main stack size(bytes)** to **0x2000** and the **Heap size (bytes)** to **0x2000**.

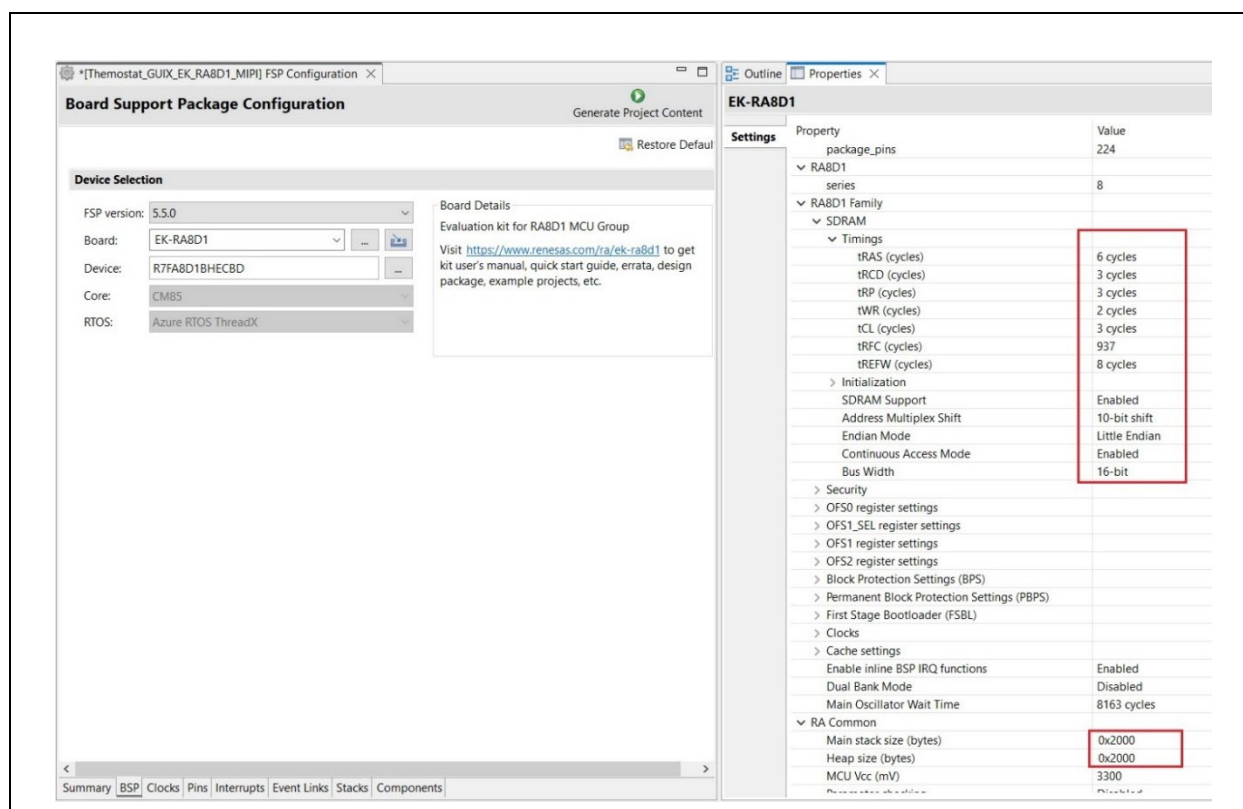


Figure 8. Setting SDRAM, Main stack size and Heap size Bytes Properties

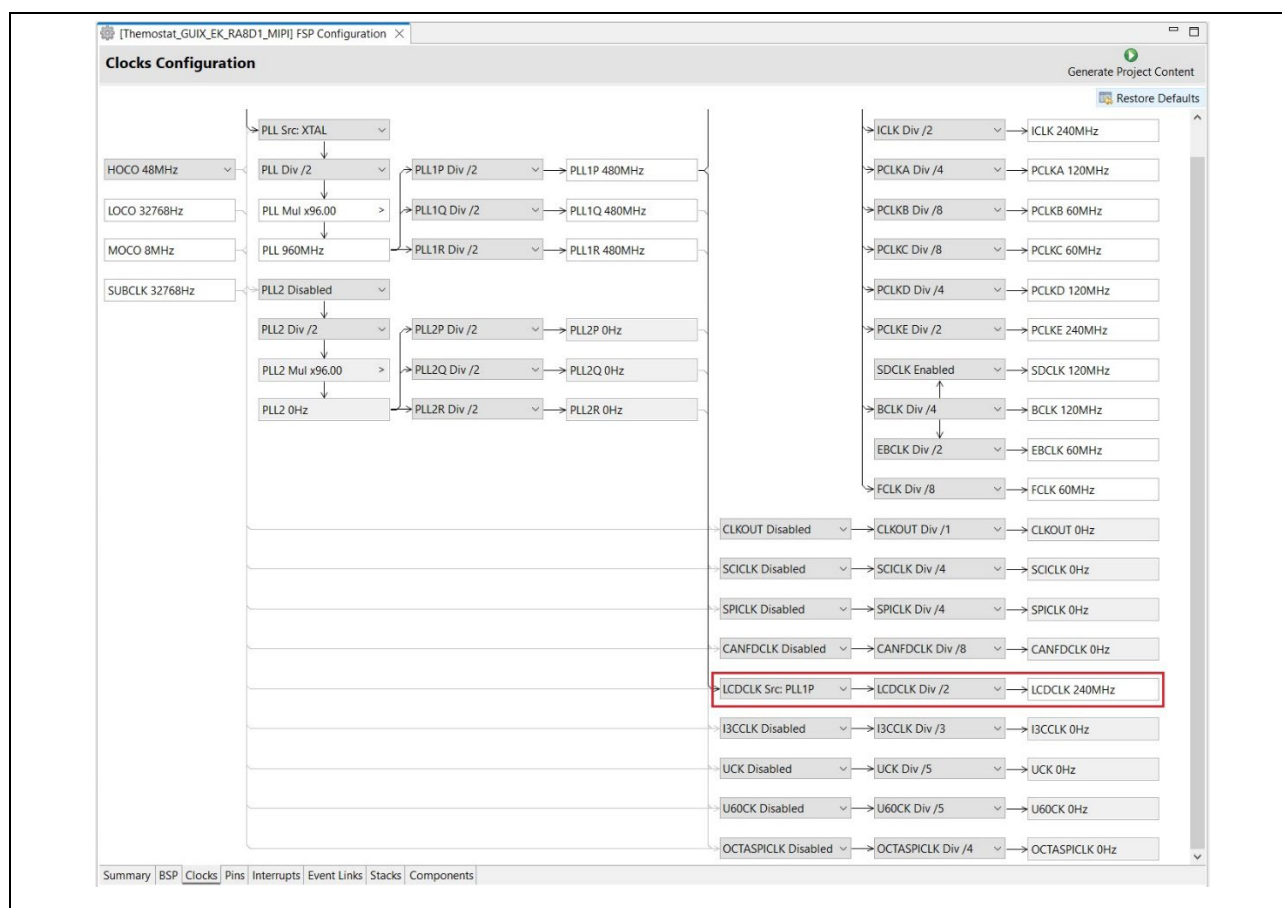
Note on GUI Storage in MCU’s Memory::

This project uses an Azure RTOS GUIX project that contains multiple full-screen pixel maps. Calculations for the memory footprint of one image are given by: 1 image = 480 x 854 pixels x 24bpp / 8 = approximately 1.3 MB. This Thermostat_GUIX_EK_RA8D1_MIPI project has multi-images stored in the code flash memory, which is larger than 2 MB of code flash memory. It will overflow the region. The alternative is to use the SDRAM to store the pixel maps. The SDRAM memory is available built-in on board (SDRAM is 512Mbit, which is 64Mbyte). Following section 2 and Figure 12, step # 2.2.9 Settings properties for **Graphics LCD**: “Input > Framebuffer > Section for framebuffer allocation > . sdram (it points to SDRAM and use SDRAM). After step 2.2.30: Generate Project Content and Build Project”, you will see the SDRAM initialized through the generated macro in hal_entry.c file BSP_CFG_SDRAM_ENABLED.

```
#if BSP_CFG_SDRAM_ENABLED

/* Setup SDRAM and initialize it. Must configure pins first. */
R_BSP_SdramInit(true);

#endif
```

Figure 9. SDRAM Initialization Macro**2.2.6 Click tab “Clocks” and set “Clocks” for the LCDCLK****Figure 10. Clocks Setting For LCDCLK**

2.2.7 Back in the Stacks Configuration tab, click add “**New Thread**”, name it **System Thread**, and set the properties as shown in the image below:

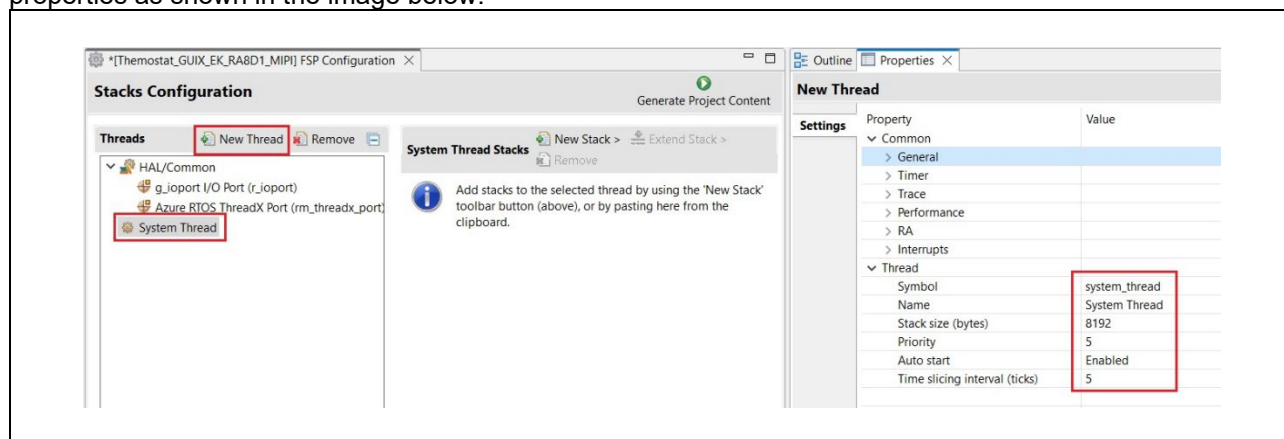


Figure 11. Adding the System Thread and setting its properties

2.2.8 With the **System Thread** selected in the Threads window, click **Add New Stack > Graphics > Azure RTOS GUIX**.

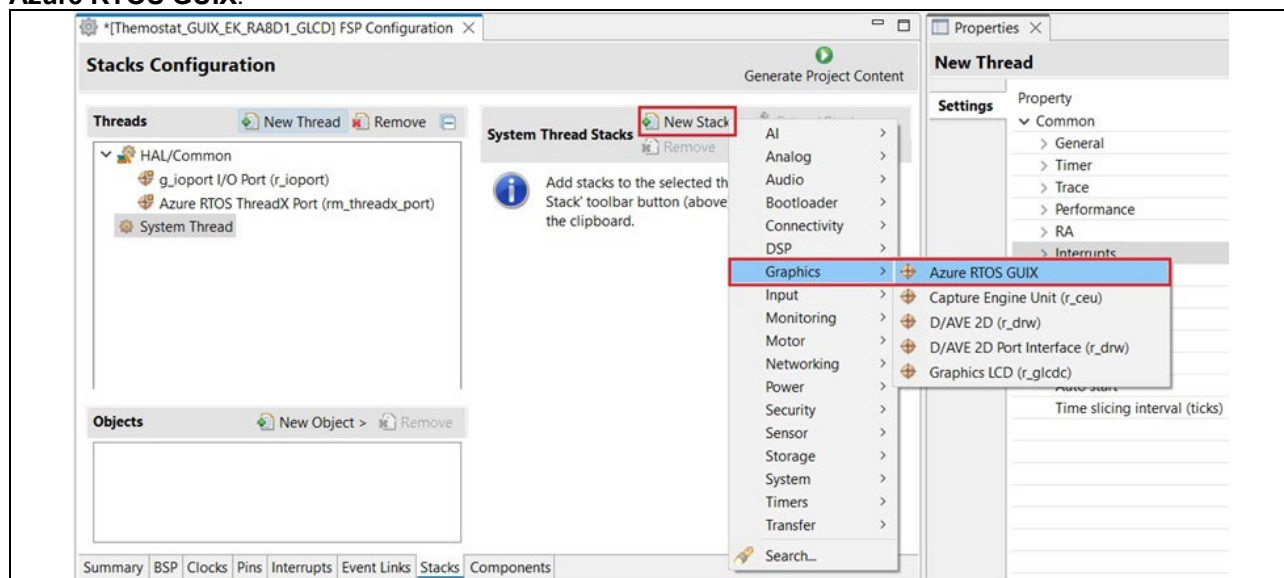


Figure 12. Add Azure RTOS GUIX

2.2.9 Use the properties settings as illustrated in the images below for the **Azure RTOS GUIX** library, the **Azure RTOS GUIX Port (rm_guix_port)** middleware, and the **Graphics LCD (r_glcdc)** module.

The figure consists of two screenshots from the FSP Configuration tool, showing the configuration of the Azure RTOS GUIX Port and the Graphics LCD module.

Top Screenshot: Azure RTOS GUIX Port (rm_guix_port) Settings

The **Stacks Configuration** window shows the **System Thread Stacks** configuration. The **Azure RTOS GUIX Port (rm_guix_port)** is highlighted in the stack. The **Settings** panel for **Azure RTOS GUIX Port (rm_guix_port)** is shown on the right.

Property	Value
DRW Buffer Cache	Enabled
Module Azure RTOS GUIX Port (rm_guix_port)	
Display Rotation	
Screen Orientation	CW (90 degrees)
Use Canvas Buffer	Disabled
Canvas Buffer Memory Section	sdram
JPEG Decoding	
Name	g_rm_guix_port0
Target Display Layer	Graphics Layer 1
Callback Function	NULL

Bottom Screenshot: g_display Graphics LCD (r_glcdc) Settings

The **Stacks Configuration** window shows the **Azure RTOS GUIX Stacks** configuration. The **g_display Graphics LCD (r_glcdc)** is highlighted in the stack. The **Settings** panel for **g_display Graphics LCD (r_glcdc)** is shown on the right.

Property	Value
Parameter Checking	Default (BSP)
Color Correction	Off
Module g_display Graphics LCD (r_glcdc)	
General	
Name	g_display
Interrupts	
Input	
Graphics Layer 1	
General	
Enabled	Yes
Horizontal size	480
Vertical size	854
Horizontal position	0
Vertical position	0
Color format	RGB8888 (32-bit)
Line descending mode	Disabled
Framebuffer	
Framebuffer name	fb_background
Number of framebuffers	2
Section for framebuffer allocation	sdram
Line Repeat	
Fading	
Graphics Layer 2	
Output	
Timing	
Horizontal total cycles	559
Horizontal active video cycles	480
Horizontal back porch cycles	5
Horizontal sync signal cycles	2
Horizontal sync signal polarity	Low active
Vertical total lines	894
Vertical active video lines	854
Vertical back porch lines	20
Vertical sync signal lines	3
Vertical sync signal polarity	Low active
Data Enable Signal Polarity	High active
Sync edge	Rising edge
Format	

Figure 13. Setting Properties of Azure RTOS GUIX Port

2.2.10 Left-click on **Add MIPI DSI Output (Optional) > New > MIPI Display (r_mipi)** to add the MIPI DSI and Phy Output modules. Then, set the interrupt callback routine name for the **MIPI Display (r_mipi_dsi)** as shown in the image below.

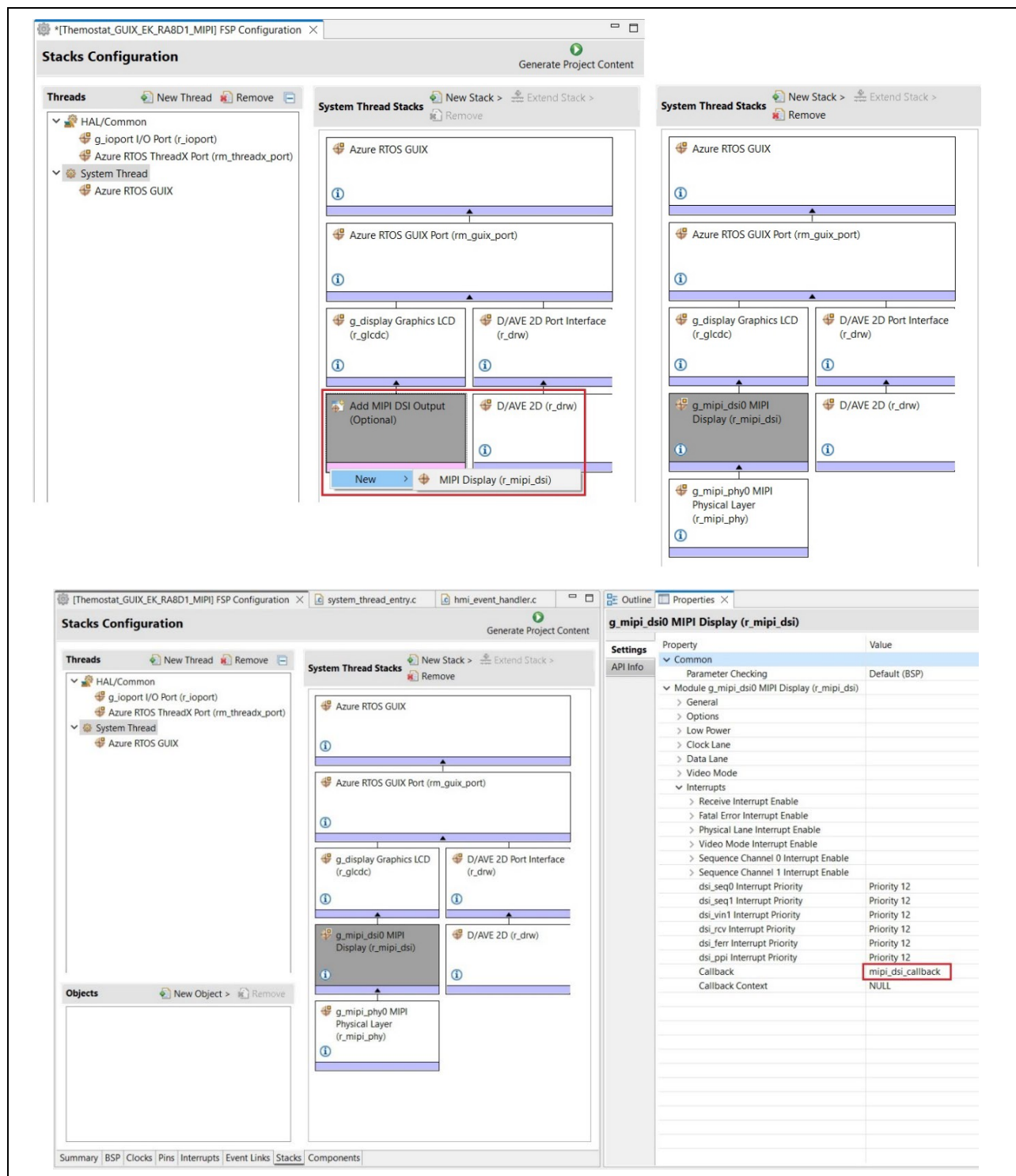


Figure 14. Adding MIPI Display module and setting properties

2.2.11 Navigate to the **Pins** tab to see the MCU **Pins Configuration**. Change P4> P404's Mode to **Output mode (Initial High)** to enable LCD panel backlight.

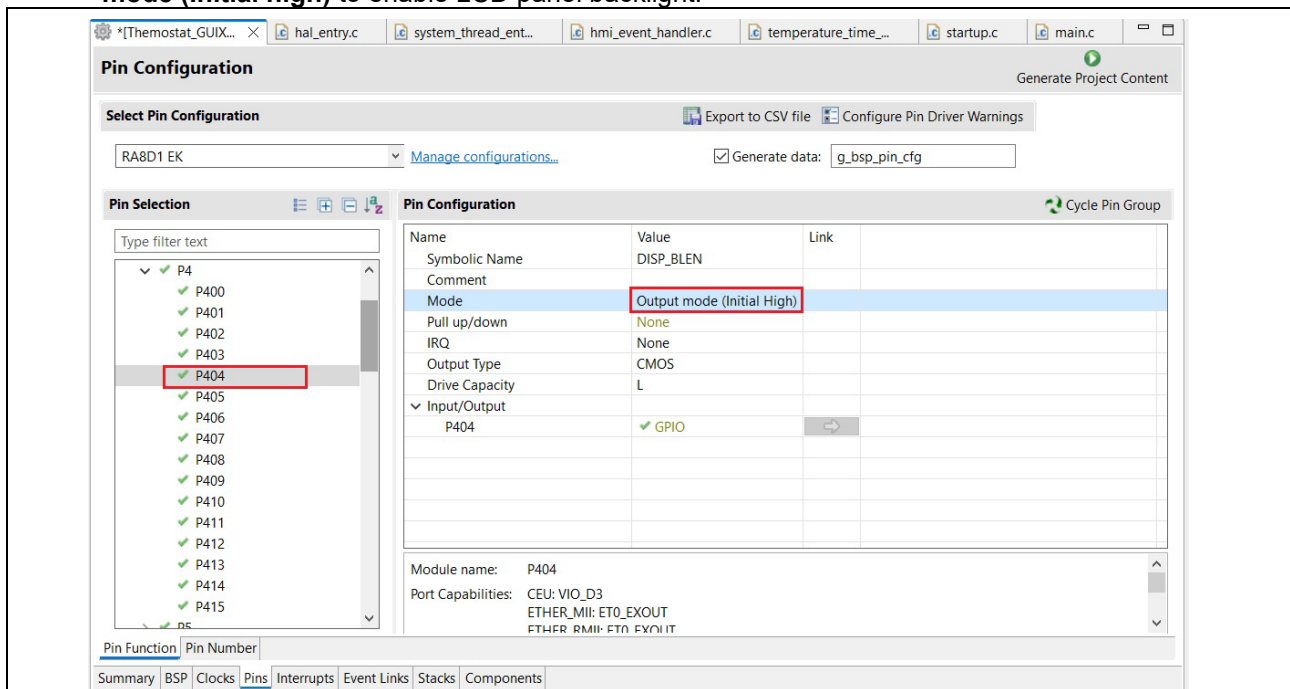



Figure 15. Settings PWM Timer properties

2.2.12 Save the configuration.xml file. In the FSP Configurator, click **Generate Project Content** to generate module support files as configured for this project. Make sure the project is selected in the Project Explorer, and click  to build the project files. It may take a long time to finish building an Azure RTOS/GUIX project on your PC.

2.2.13 From the source file folder **FSP_GUIX_Thermostat** accompanying this application note, copy the Azure RTOS GUIX Studio project folder named **guix_studio** into e² studio project file tree.

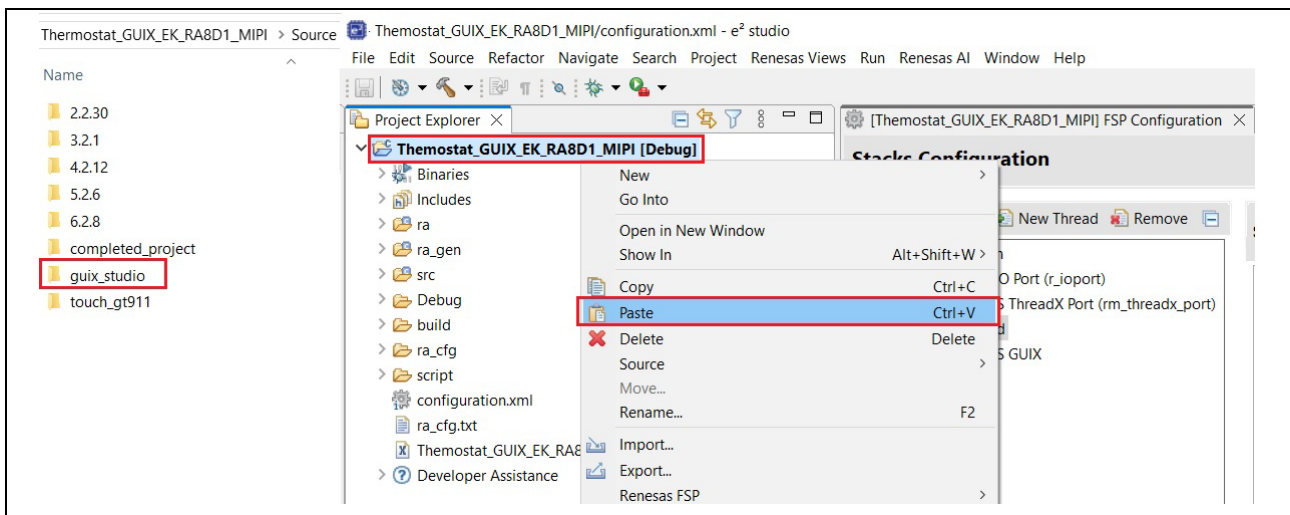


Figure 16. Copying the Azure RTOS GUIX Studio Project to e² studio

2.2.14 The GUIX Studio project is now in the Thermostat_GUIX_EK_RA8D1_MIPI e² studio project, but the GUIX project cannot be built by the compiler, so the folder must be removed from the build. In the e² studio Project Explorer, right-click on “**guix_studio**” folder and select **Resource Configurations > Exclude from Build...** ensure that both the **Debug** and **Release** are checked, and click **OK**.

The “guix_studio” folder holds the GUIX Studio thermostat project. This folder differs from the GUIX project’s resource and specification files. Those .c and .h files contain the GUIX project’s pixel maps and fonts, and they will be generated at a later step and included in the e² studio project build.

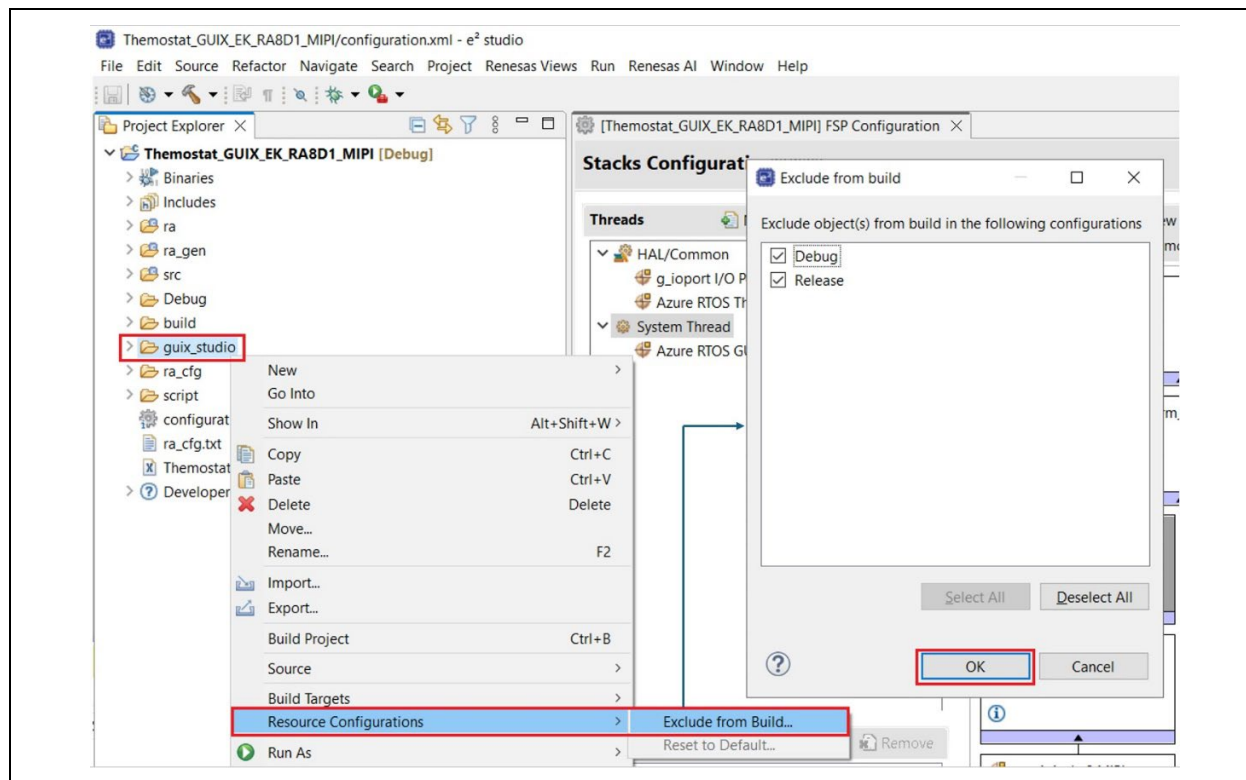


Figure 17. Excluding the guix_studio folder from the build

2.2.15 Open the Thermostat_GUIX_EK_RA8D1_MIPI project folder by right-clicking the e² studio project name in the Project folder and selecting “System Explorer,” as shown below.

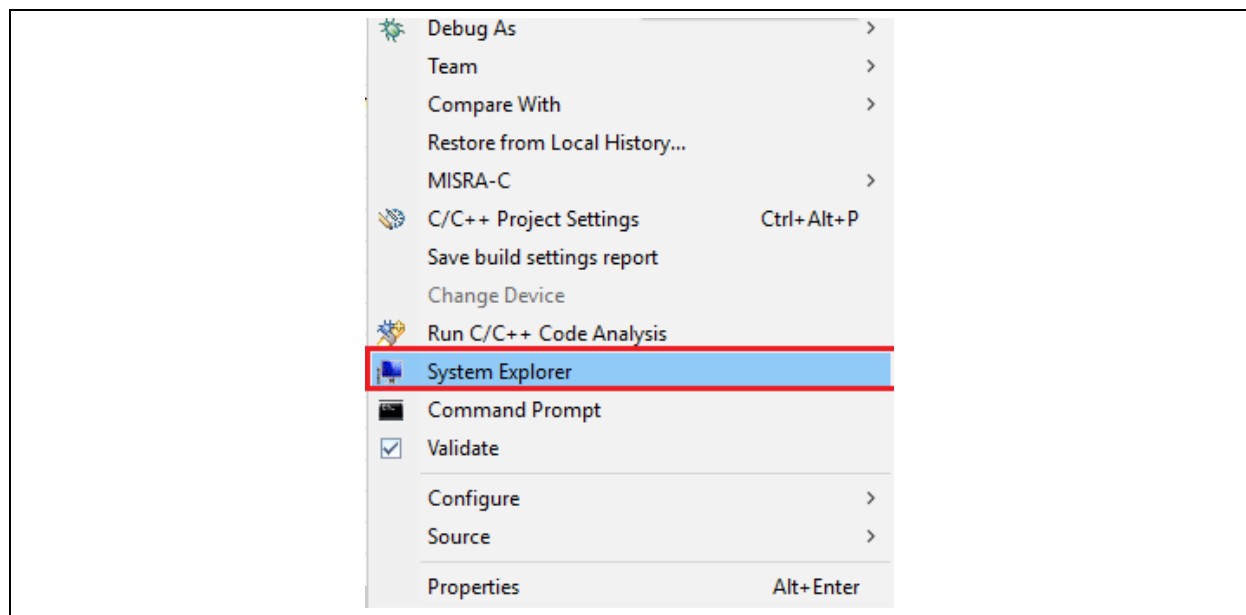


Figure 18. Selecting System Explorer

2.2.16 Navigate to the sub-folder “**guix_studio > GNU**” and double-click on the **thermostat.gpx** project file to open it with GUIX Studio. Remember that the GUIX Studio version needs to be **v6.4.0.0 or later**.

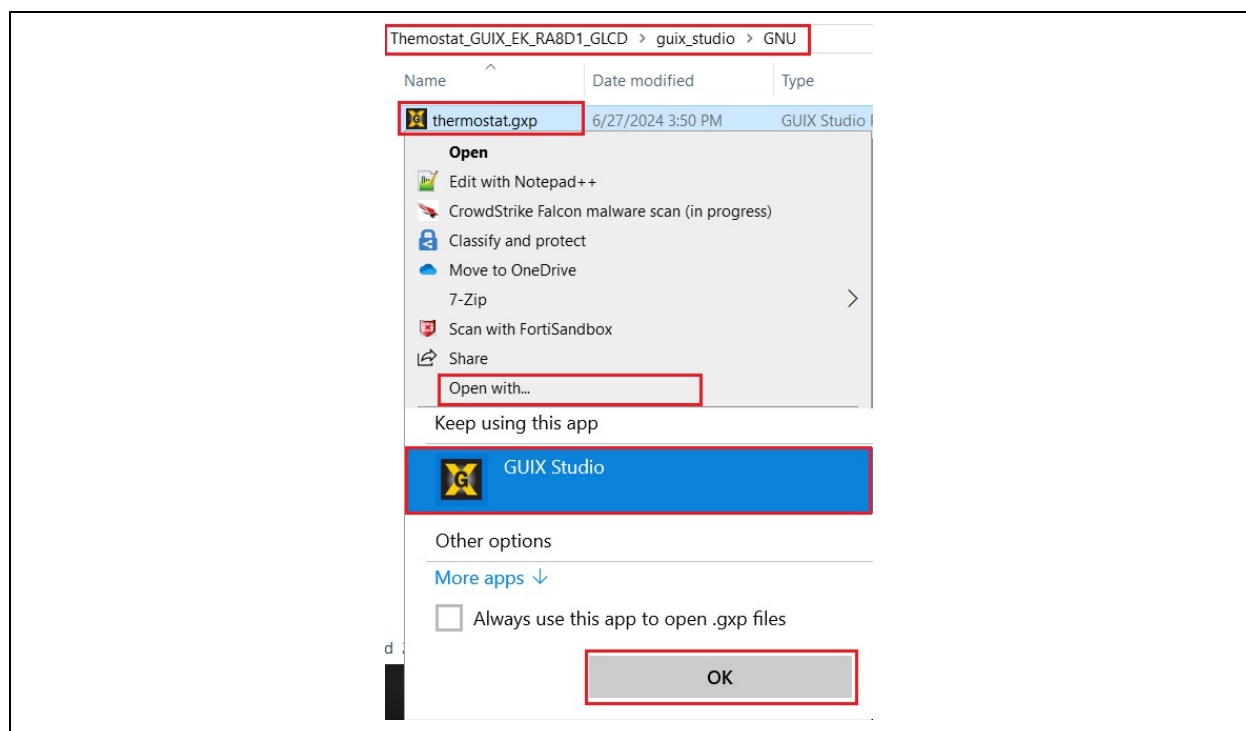


Figure 19. Opening the Project File

2.2.17 The GUIX Studio project is a completed design of the Thermostat GUI application. The next several steps describe the process to generate the project's resource application code and integrate the code files to the e² studio project Thermostat_GUIX_EK_RA8D1_MIPi.

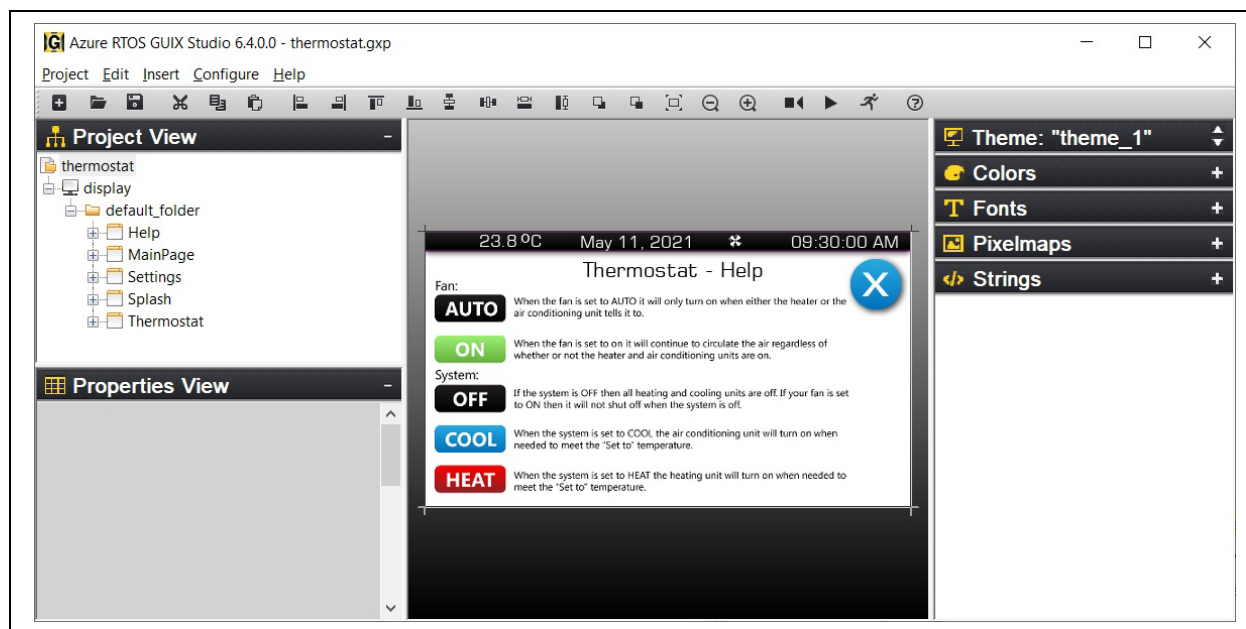


Figure 20. GUIX Studio Thermostat Application View

2.2.18 The Azure RTOS GUIX Studio project consists of 5 screens, including the Splash, Main Page, Settings, Thermostat, and Help, from the top to bottom layer:



Figure 21. Azure RTOS GUIX Studio Project Screens

2.2.19 Click on the GUIX Studio option **Configure > Project/Display** and confirm the following settings.

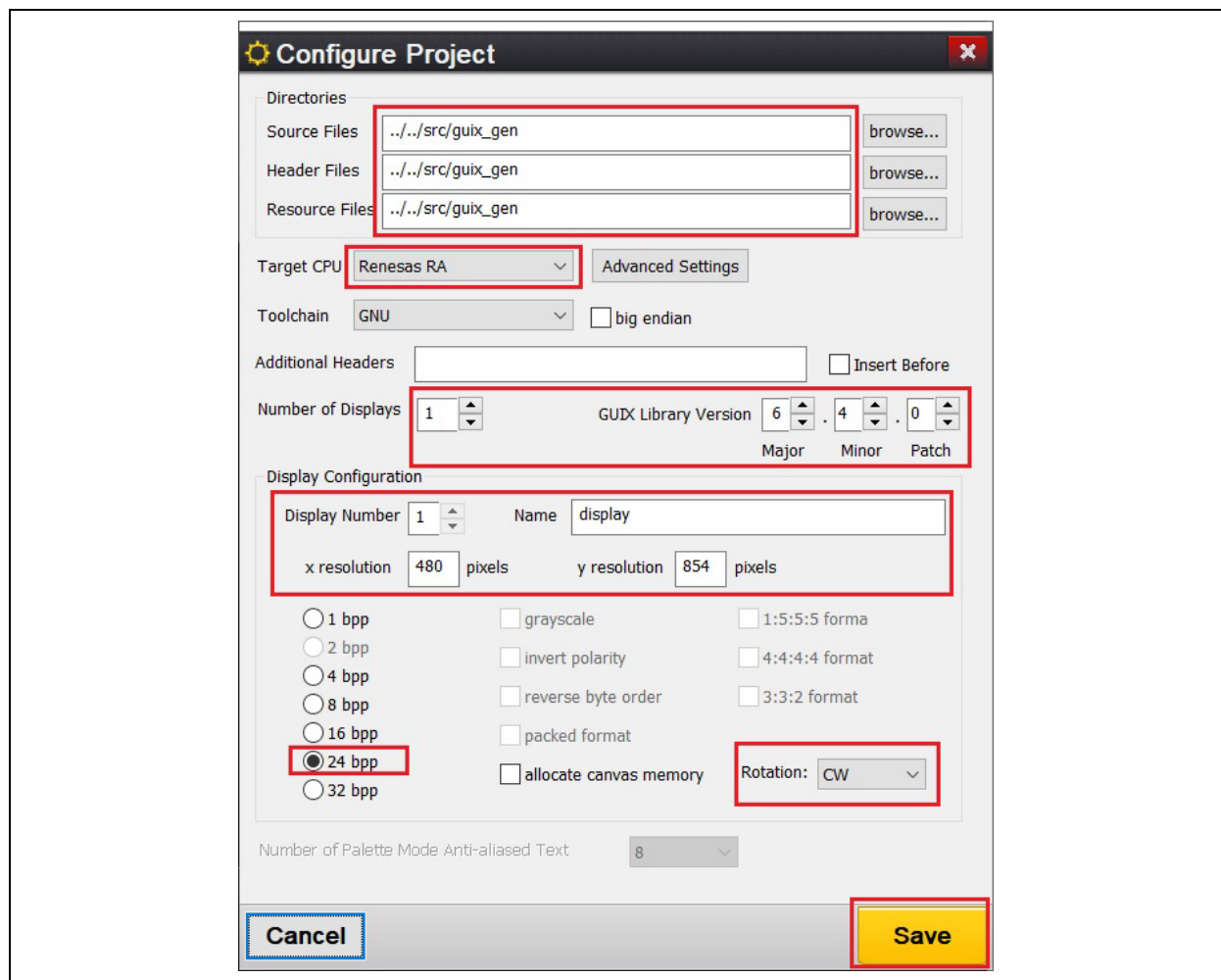


Figure 22. Configure Project Settings

2.2.20 Go back e² studio project (Thermostat_GUIX_EK_RA8D1_MIPI), right-click on the project file “src”, then select **New > Folder** and create a folder named “guix_gen”.

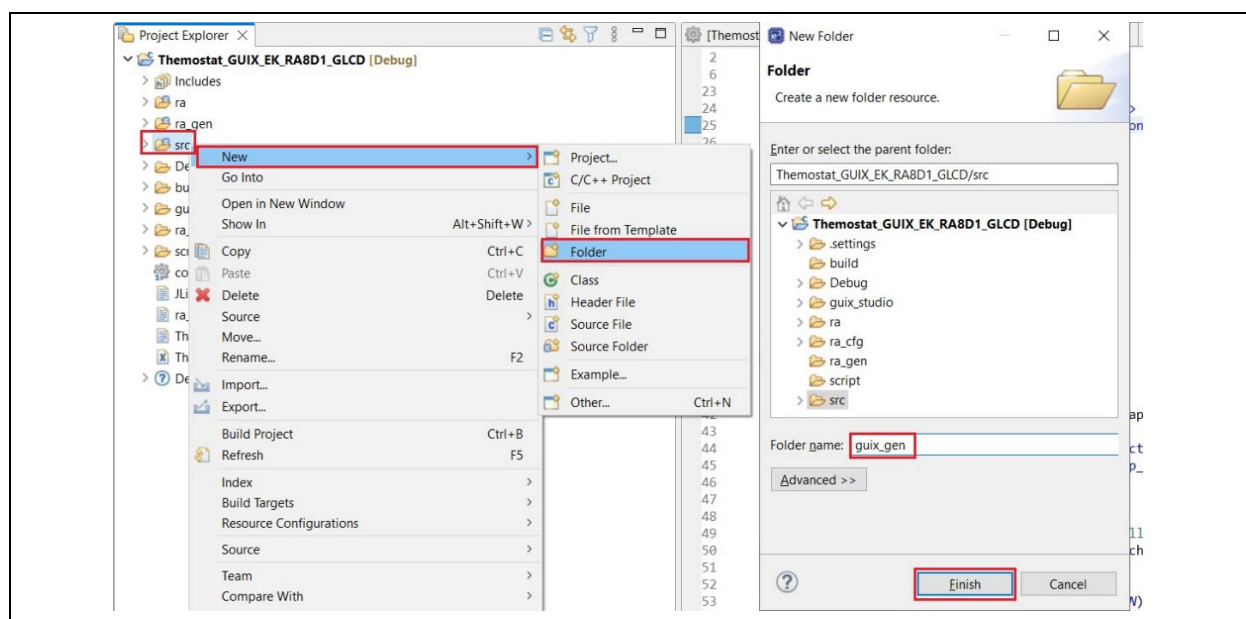


Figure 23. Creating a “guix_gen” in e² studio Project

2.2.21 Confirm “guix_gen” is created before moving to next step.

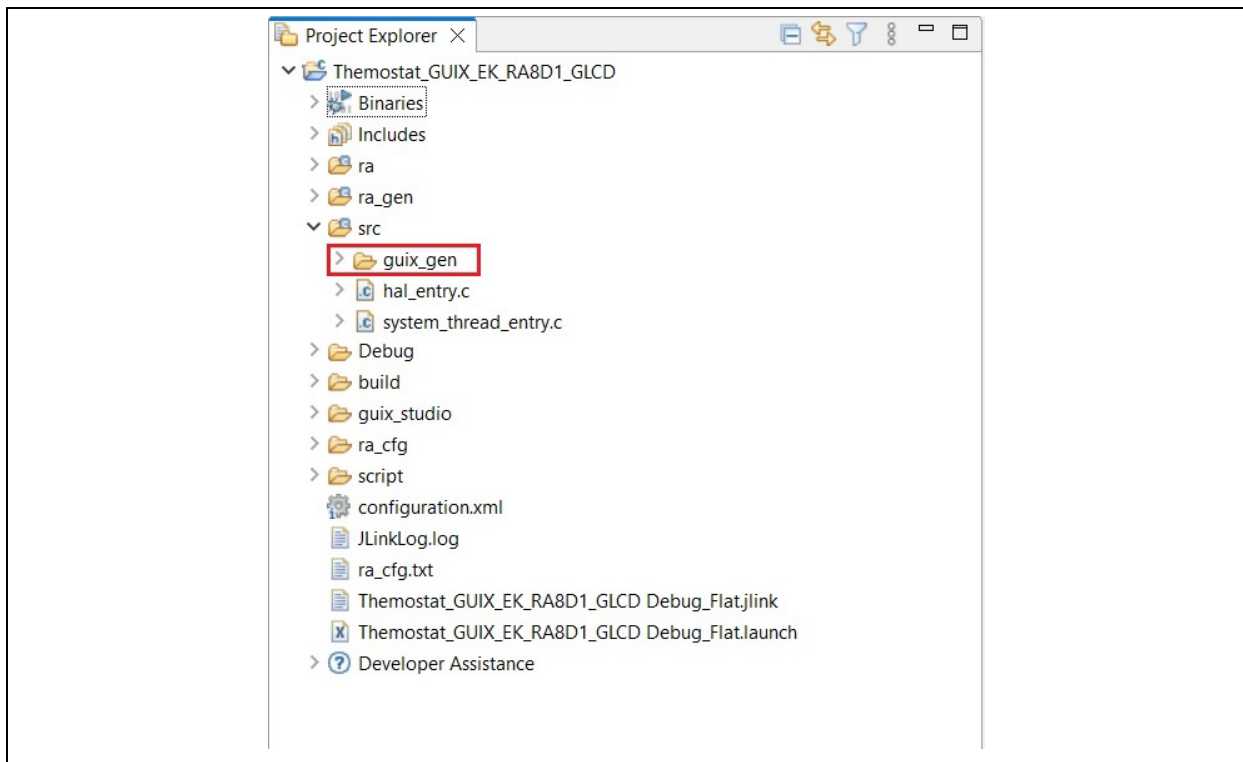


Figure 24. Confirming creation of “guix_gen”

2.2.22 In Azure RTOS GUIX Studio, click **Project > Generate All Output Files** to generate resource files, header files, and source files of this GUIX design.

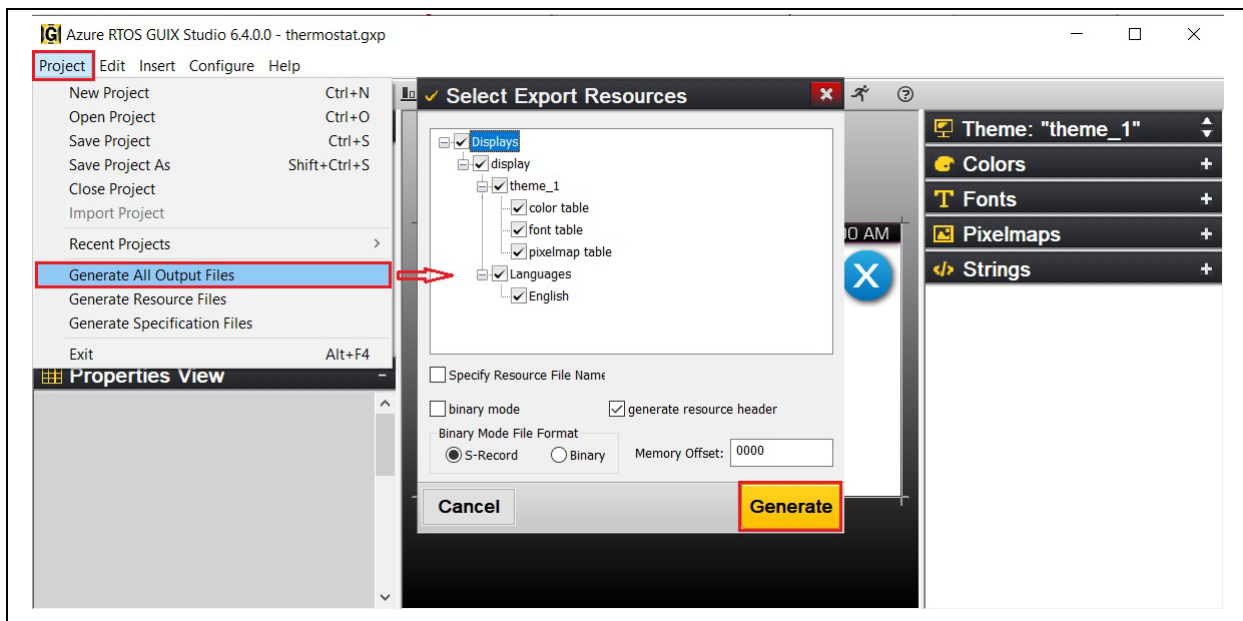


Figure 25. Clicking Generate All Output Files

2.2.23 Click **Generate**. If successful, you will see the notification below.



Figure 26. All Output Files Updated Notification

2.2.24 All output files are now in “guix_gen” folder.

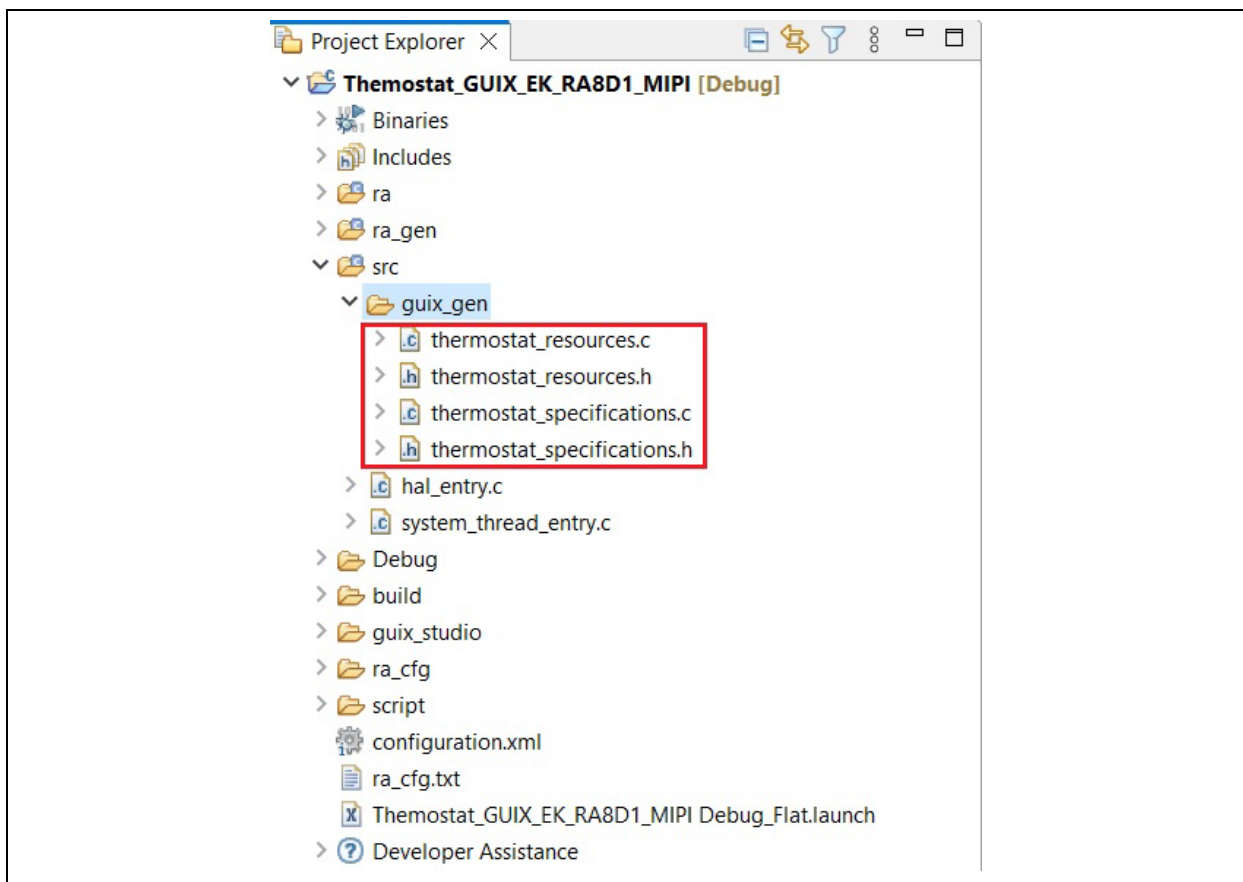


Figure 27. Location of Output Files

2.2.25 From the GUIX project’s Project View window, select the “**Splash**” screen to note the names given for the **Widget Name** and **Event Function** definitions. These definitions are used to create a screen and handle it in the **e² studio/FSP project**. The other windows have similar definitions.

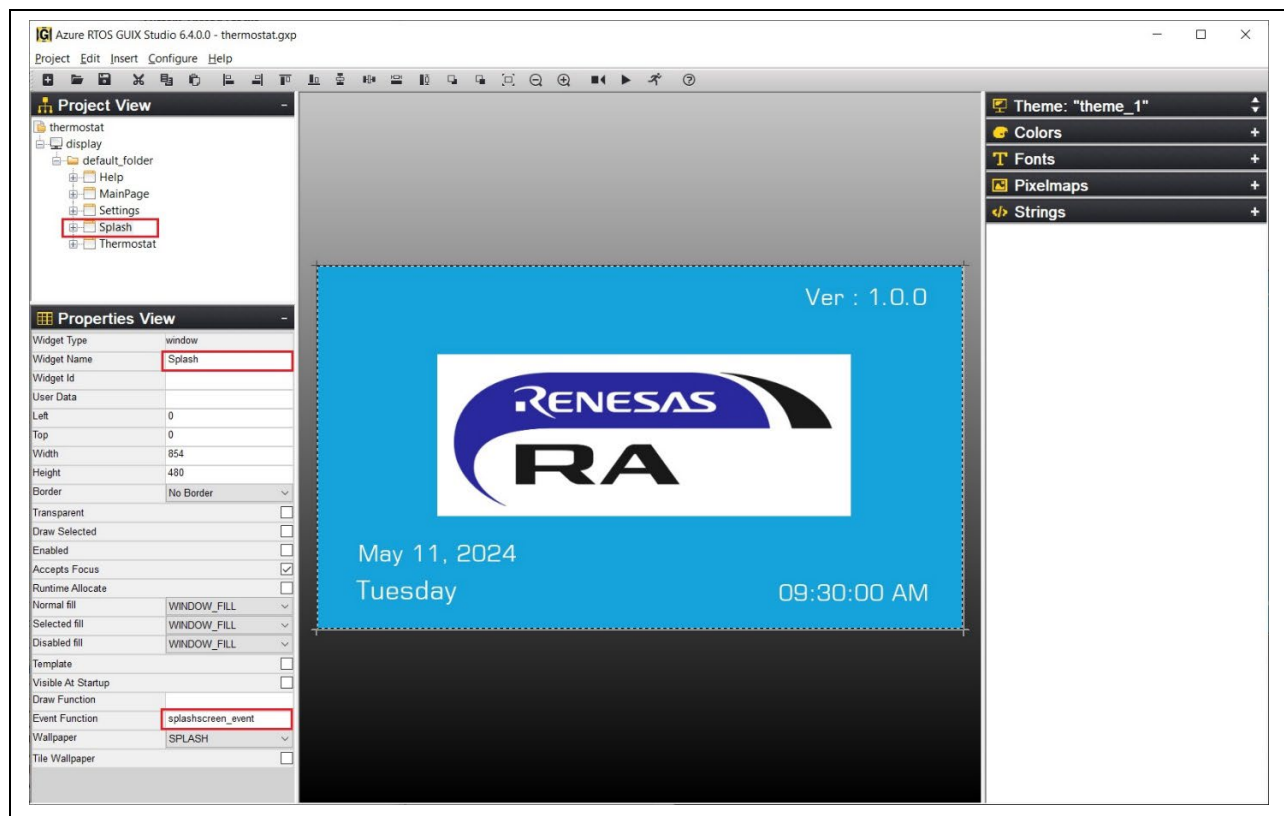


Figure 28. Definitions in the Azure RTOS GUIX Studio Project

2.2.26 From the Source file folder accompanying this application note, copy the files in the **2.2.26** folder to the e² studio’s **src** folder and overwrite all files that already exist.

- hmi_event_handler.c
- system_thread_entry.c
- system_thread_entry.h

Build Thermostat_GUIX_EK_RA8D1_MIPi project. You will see several warnings, but we will address them in later steps.

2.2.27 Code highlight: The following example creates a screen based on the Widget Name in the GUIX project and attaches it to the root window. In this case, it is the “Splash” screen. Refer to system_thread_entry.c for more details.

```
/* Create the widget and attached to root window.*/
gx_err = gx_studio_named_widget_create("Splash", (GX_WIDGET *) p_root, (GX_WIDGET **) &p_splash_screen);
if(GX_SUCCESS != gx_err)
{
    APP_ERR_TRAP(FSP_ERR_ASSERTION);
}
```

Figure 29. Create the Splash screen using GUIX API

2.2.28 Code highlight: An event function associated with a screen needs to be defined to handle events on that screen. Refer to hmi_event_handler.c for more details. All event functions are empty at this point.

```

/*****
 * @brief   Handles all events on the splash screen.
 * @param[in] widget   Pointer to the widget that caused the event
 * @param[in] event_ptr Pointer to event that needs handling
 * @retval   GX_SUCCESS
 *****/
UINT splashscreen_event(GX_WINDOW *widget, GX_EVENT *event_ptr)
{
    UINT gx_err = GX_SUCCESS;

    return gx_err;
}

```

Figure 30. Handle the Splash screen events

2.2.29 Get your EK-RA8D1 ready to run the project. Connect the MIPI LCD board to the J58 connector on EK-RA8D1, as shown below.

- Switch on board SW1-6 for GLCD_OE_L set “ON”
- Switch on board SW1-7 for SDRAM set “ON”.
- All other switches, set to “OFF”

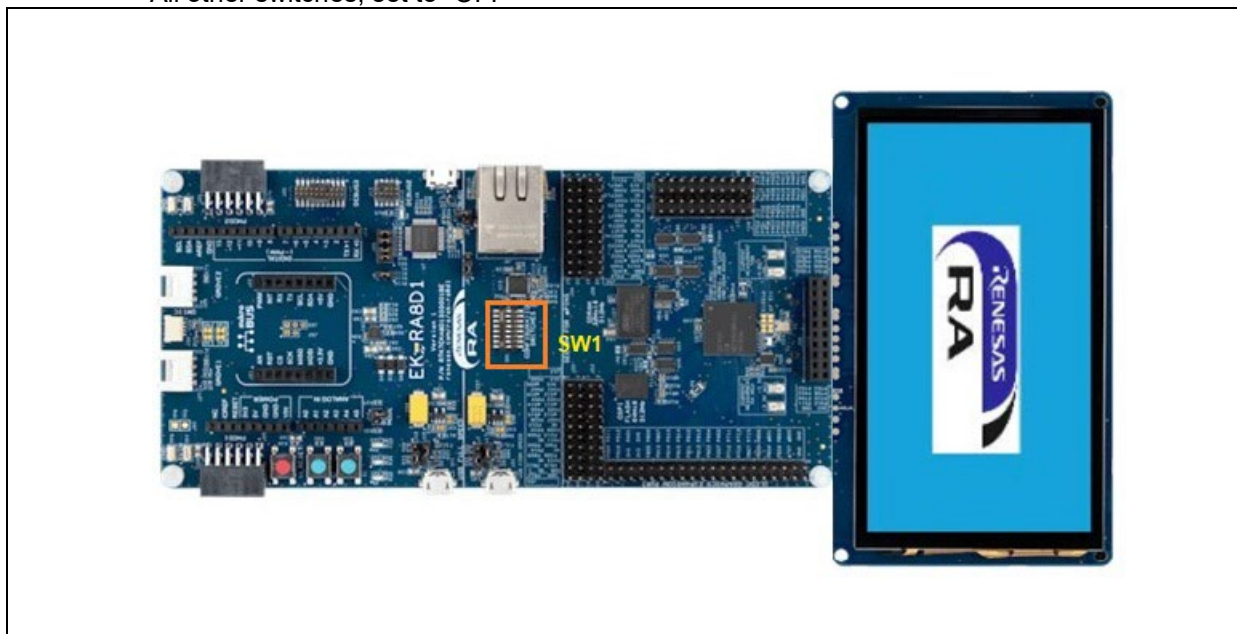




Figure 31. Connecting MIPI LCD Board to Connector of EK-RA8D1

2.2.30 Connect the micro-USB cable end to **J10** on the EK-RA8D1 board and the other end to your PC.

Build Project  and start the Thermostat_GUIX_EK_RA8D1_MIPI Debug_Flat  project; you will see a black screen.

2.2.31 Follow the code in **splashscreen_event()** function in **hmi_event_handler.c** to show The Splash screen. **Build** the e² studio project.

```
switch (event_ptr->gx_event_type)
{
    case GX_EVENT_SHOW:
        gx_err = gx_window_event_process(widget, event_ptr);
        if(GX_SUCCESS != gx_err) {
            while(1);
        }
        break;
    default:
        gx_err = gx_window_event_process(widget, event_ptr);
        if(GX_SUCCESS != gx_err) {
            while(1);
        }
        break;
}
```

Figure 32. Show the Splash screen with GUIX APIs

Please refer to the splashscreen_event function in hmi_event_handler.c in the “2.2.26” folder in the AN folder.

2.2.32 Since the project has already been downloaded, press the **Play** button twice to **run** the project. You will see the Splash screen on the LCD panel:

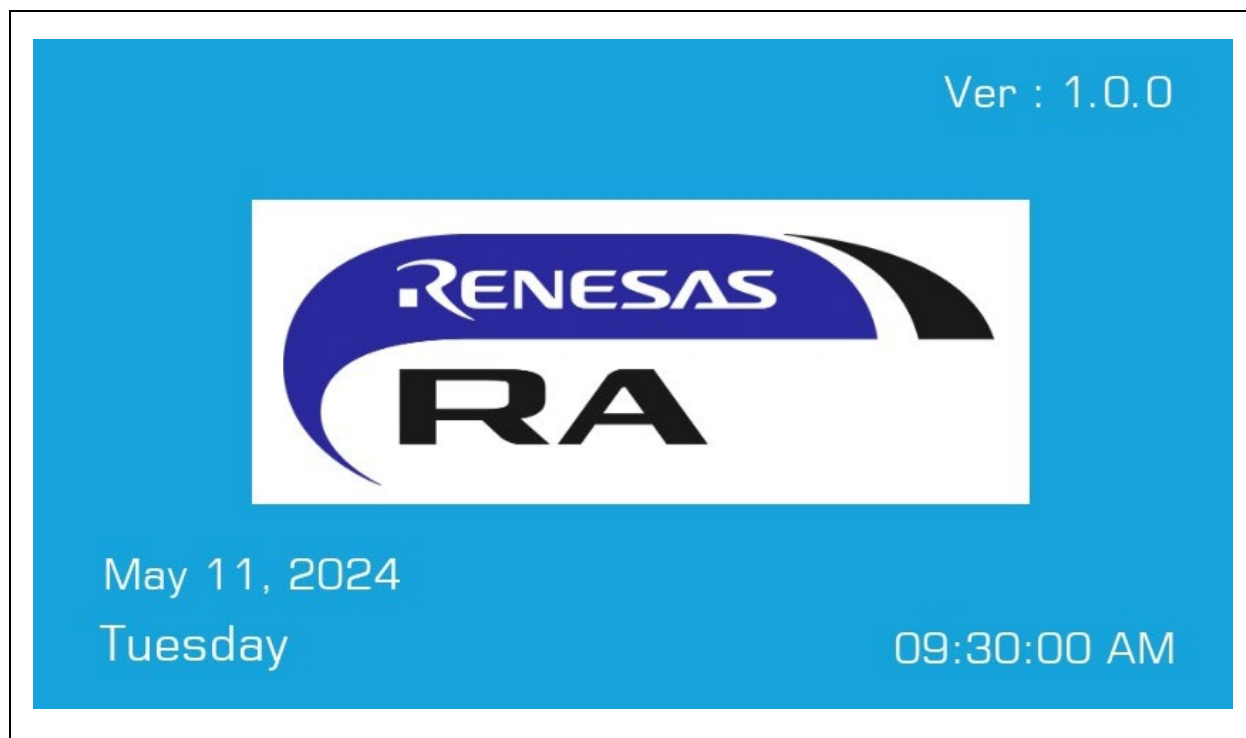


Figure 33. Splash Screen View on LCD

3. Using GUIX Widget Timer to Trigger a Screen Transition

3.1 Overview

In this section, you will implement a simple use of the GUIX Widget timer, which triggers a screen transition.

3.2 Procedural Steps

3.2.1 From the Source file folder accompanying this application note, copy the files in the **3.2.1** folder to the e² studio's **src** folder, and overwrite all files that already exist. The contents should include:

- hmi_event_handler.c
- system_thread_entry.c
- system_thread_entry.h

3.2.2 Code highlight: The following code in the **splashscreen_event()** function starts a GUIX Widget timer and triggers a screen transition that hides the Splash screen and shows the Main Page screen.

```
switch (event_ptr->gx_event_type)
{
    case GX_EVENT_TIMER:
        gx_system_timer_stop(widget, 10);
        toggle_screen(p_mainpage_screen,p_splash_screen);
        break;
    case GX_EVENT_SHOW:
        gx_system_timer_start(widget, 10 , SPLASH_TIMEOUT,
SPLASH_TIMEOUT);
        gx_err = gx_window_event_process(widget, event_ptr);
        if(GX_SUCCESS != gx_err) {
            while(1);
        }
        break;
    default:
        gx_err = gx_window_event_process(widget, event_ptr);
        if(GX_SUCCESS != gx_err) {
            while(1);
        }
        break;
}
```

Figure 34. Hide the Splash screen and show the Main Page screen

3.2.3 Build, Download, and Run the project, and you will see the transition from the **Splash screen** to the **Main Page screen** in about 3 seconds.

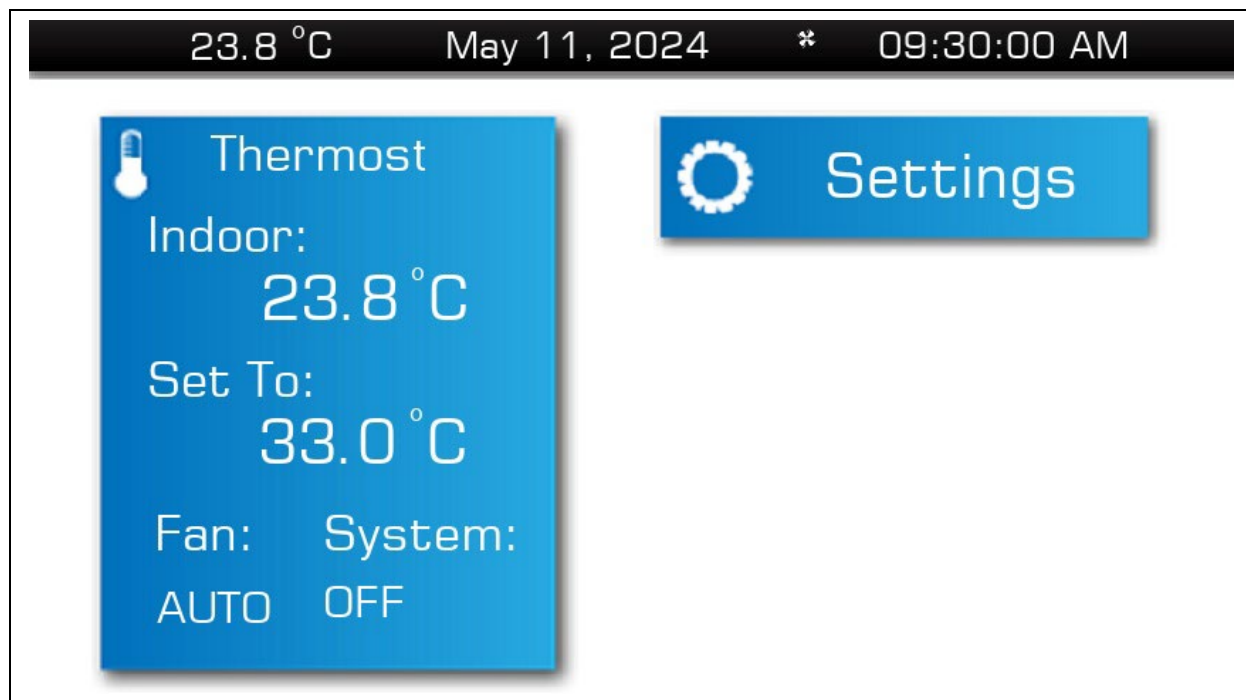


Figure 35. Main Page Screen

4. Add Touch Driver to Thermostat_GUIX_EK_RA8D1_MIPI Project

4.1 Overview

In this section, you will add the gt911 touch driver to the project to communicate between the MCU and the touch control circuitry on the MIPI LCD panel.

4.2 Procedural Steps

4.2.1 In the Thermostat_GUIX_EK_RA8D1_MIPI project, create a folder by right-clicking “src”, then select **New > Folder**. Name the folder “touch_gt911” and click **Finish** to create.

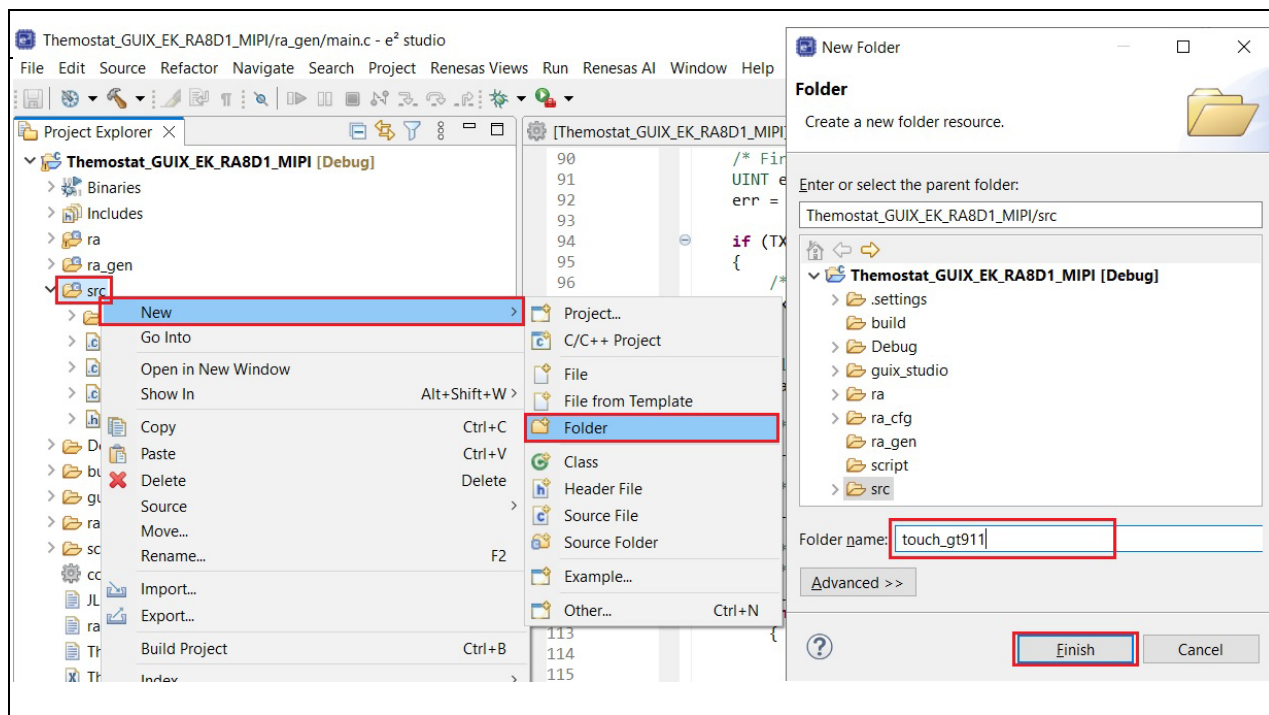


Figure 36. Creating New Folder in Thermostat_GUIX_EK_RA8D1_MIPI Project

4.2.2 Copy the contents of the Source folder **Source > “touch_gt911”** to the e² studio project **src > touch_gt911** folder you just made. The contents are the hardware driver files touch_gt911.c and touch_gt911.h, which interface to the touch IC on the MIPI LCD.

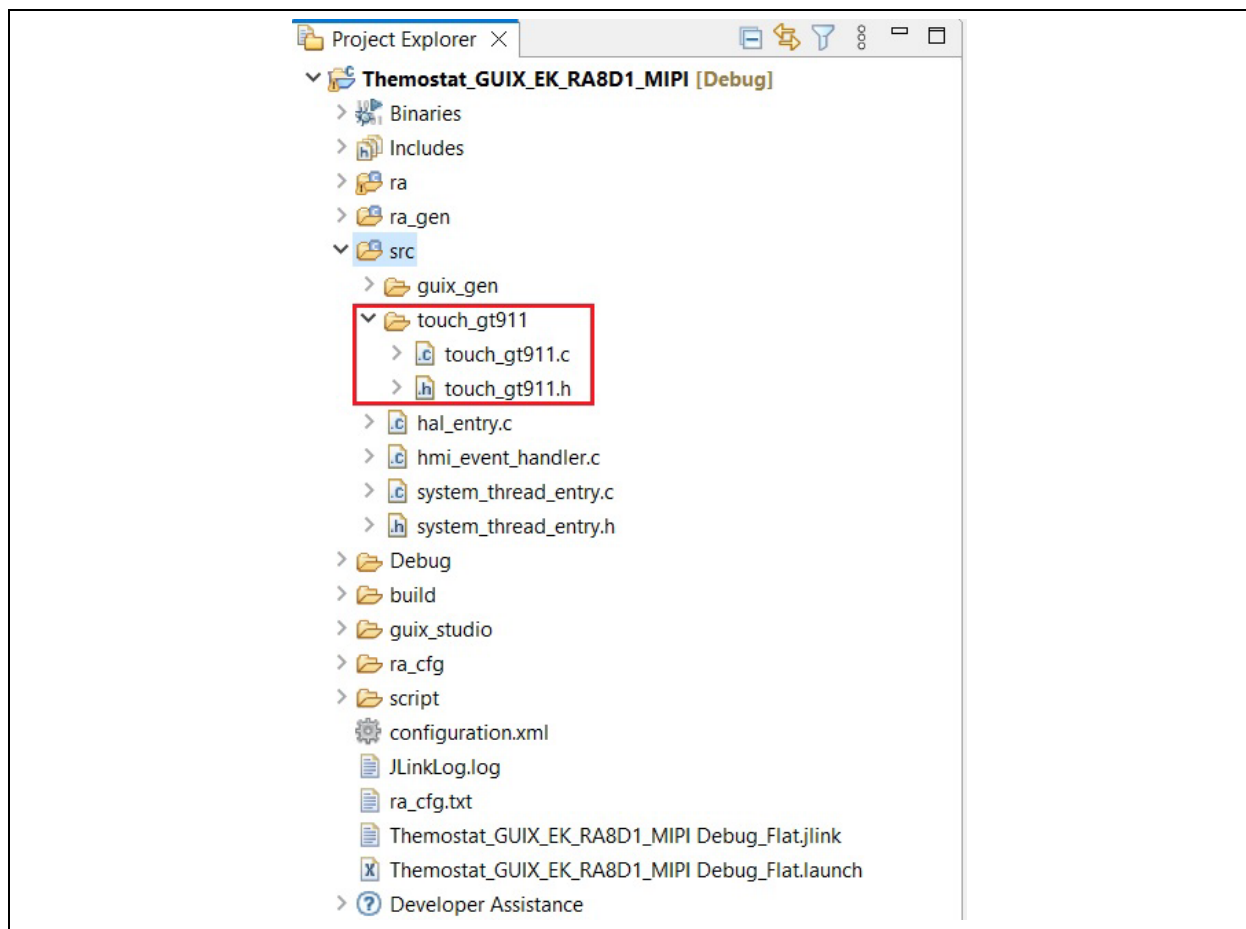


Figure 37. Copying files to the e² studio Project

4.2.3 Open the e² studio's **configuration.xml** file in the **Stacks** tab and ensure that the e² studio view FSP Configuration is selected. Use the **New Thread** button to create a **Touch Thread** with the settings illustrated below.

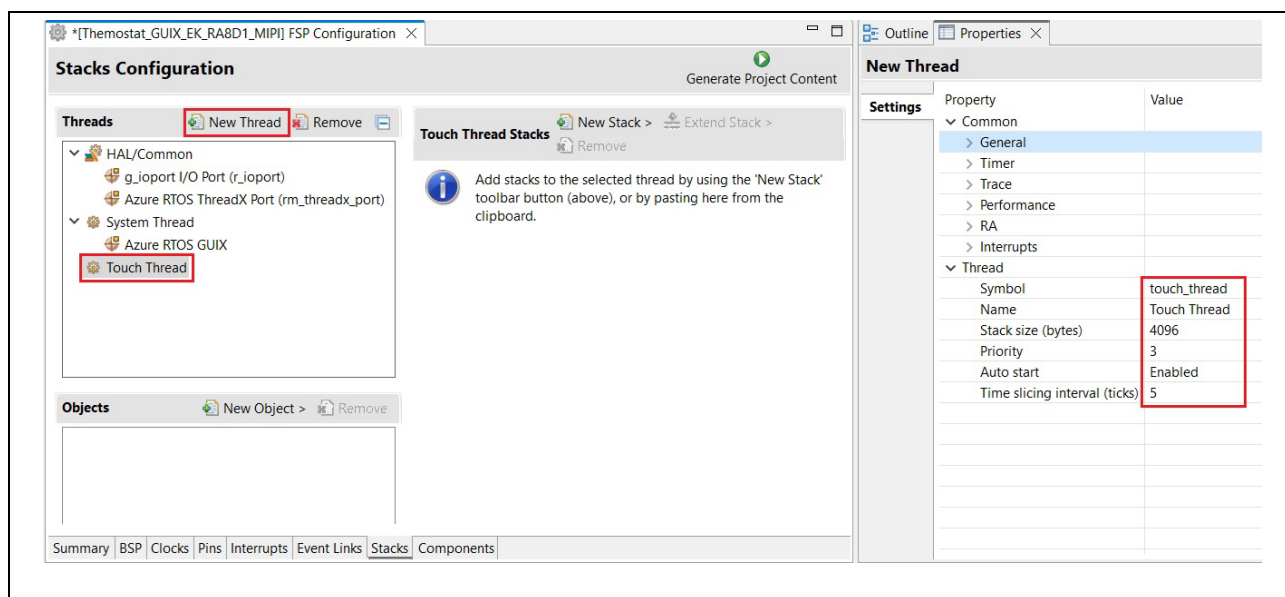


Figure 38. Creating Touch Thread

4.2.4 The image below shows the MIPI interface from the EK-RA8D1 MCU schematic. The pins marked in red are used for touch panel control on the MIPI LCD board:

- IRQ3 interrupt (DISP_INT = P510) is used to trigger touch events.
- IIC channel 1 (ICC_SDA = P512, ICC_SCL = P511) is used to read and write data via I2C communication protocol to the touch controller.
- DISP_RST = PA01 is used to reset the touch controller.

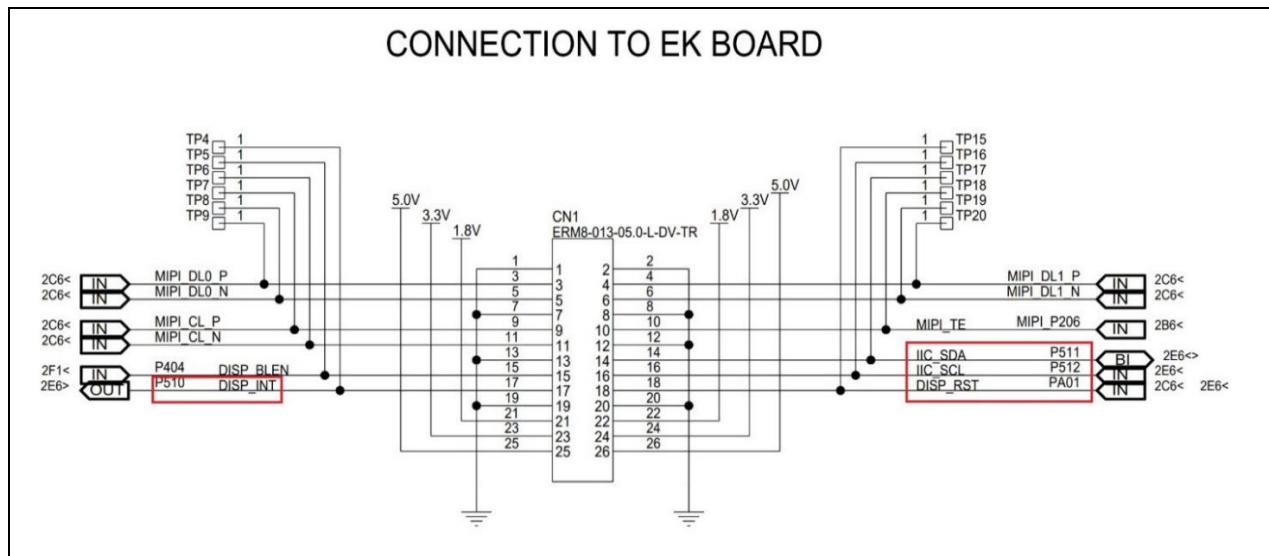


Figure 39. MIPI LCD Pin address

4.2.5 Next from the **Pins** tab, go to **PA > PA01** and set the DISP_RST IRQ (PA01) with the following configuration:

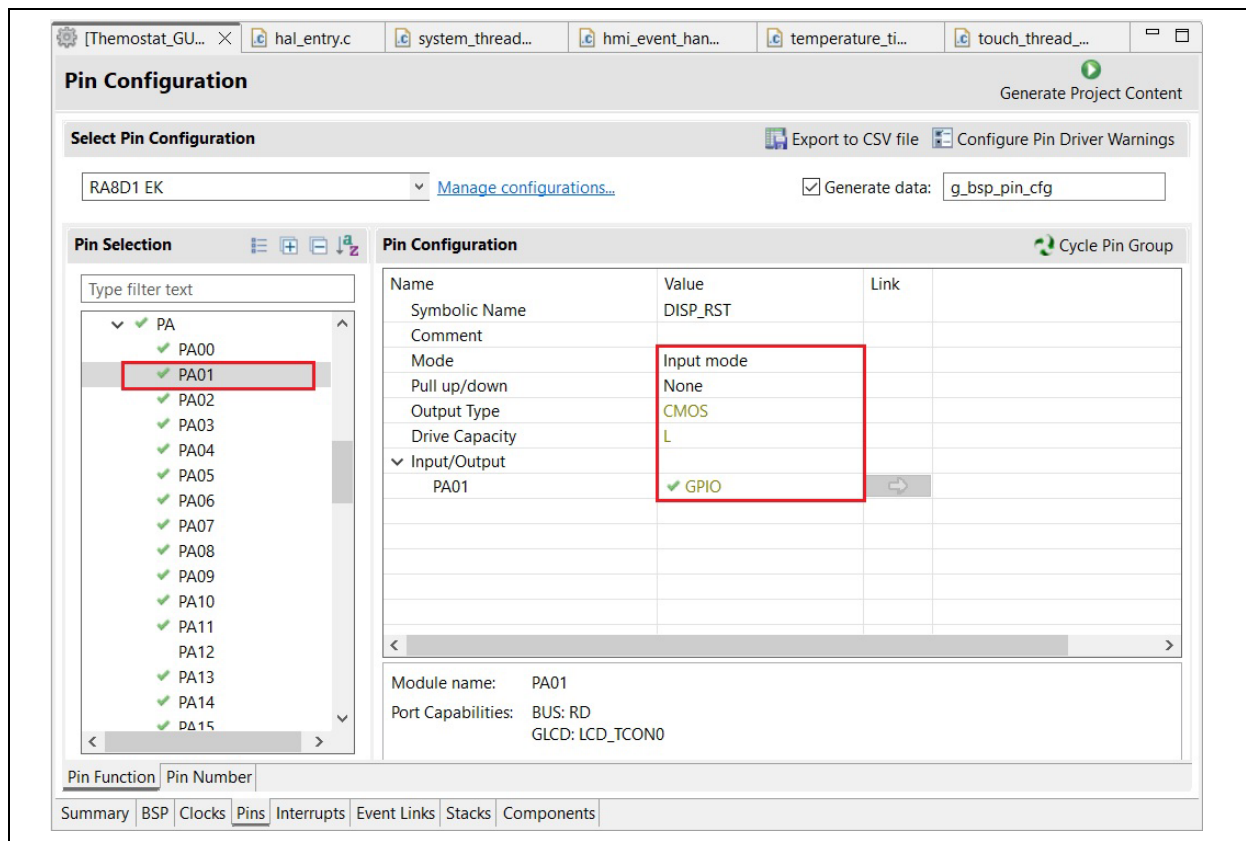


Figure 40. Setting DISP_RST IRQ Pin Configuration

4.2.6 Back in the **Stacks** tab, make sure the **Touch Thread** is selected from the Threads window and use the **New Stack** button to add the **Input > External IRQ Driver (r_icu)** to the Touch Thread.

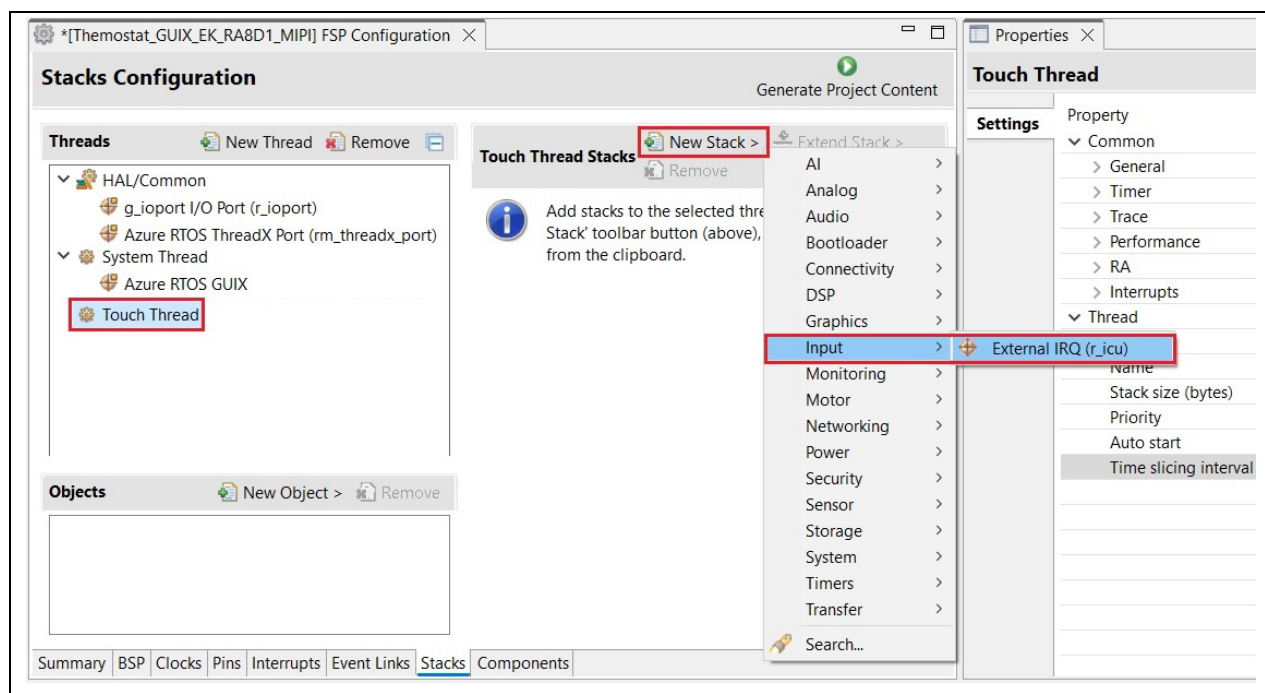


Figure 41. Adding External IRQ Driver on r_icu to Touch Thread

4.2.7 From the Properties window, copy the settings from the image below for the **External IRQ (r_icu)**. Then Click the yellow arrow “<unavailable>” from the Pins row to navigate to the Pins tab and set **P510** as the Input/Output field for **IRQ3**.

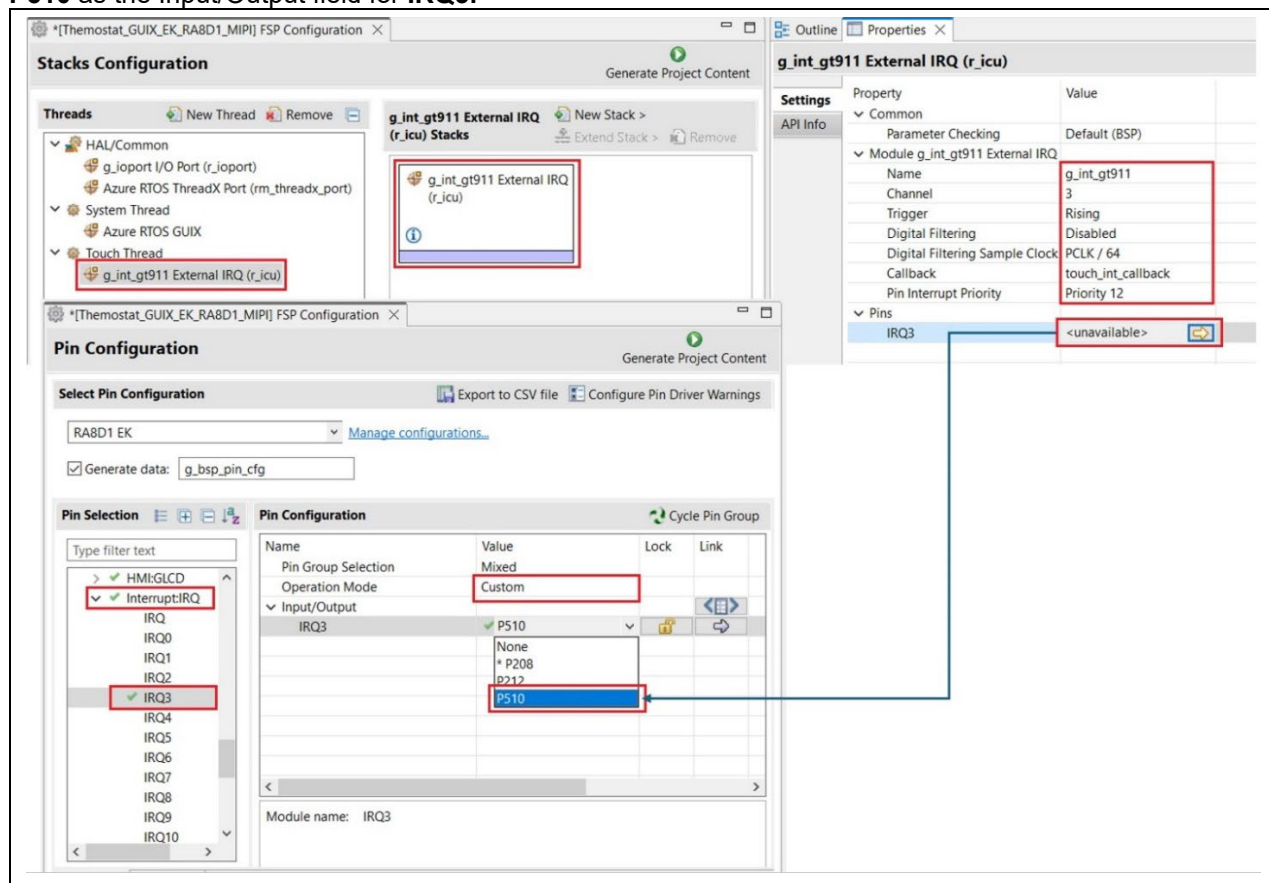


Figure 42. Settings External IRQ Properties

4.2.8 Back in the **Stacks** tab of e² studio, follow a similar procedure as the External ICU (r_icu) module to add the hardware module **Connectivity > I2C Master Driver (r_iic_master)** to the **Touch Thread** and set the right properties as shown in the images below.

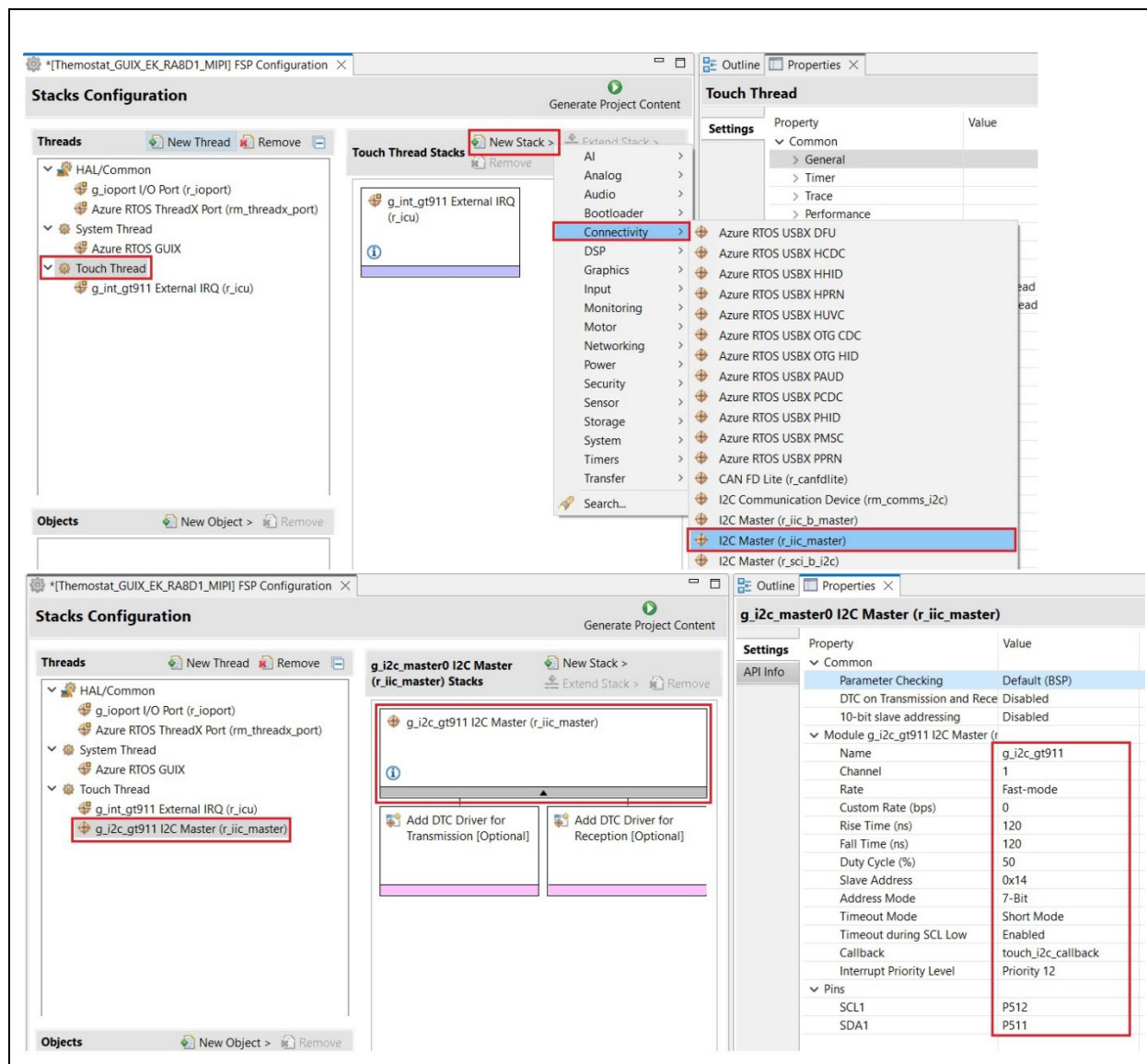


Figure 43. Adding I2C Master Driver on r_iic_master to Touch Thread

4.2.9 From the **Objects** window, use the **New Object > Semaphore** to add the Semaphore GT911 IRQ with the symbol **g_semaphore_gt911_irq**. Set the Initial Count to **0**. This semaphore is used in the gt911 driver to trigger data reading when a touch-panel interrupt occurs.

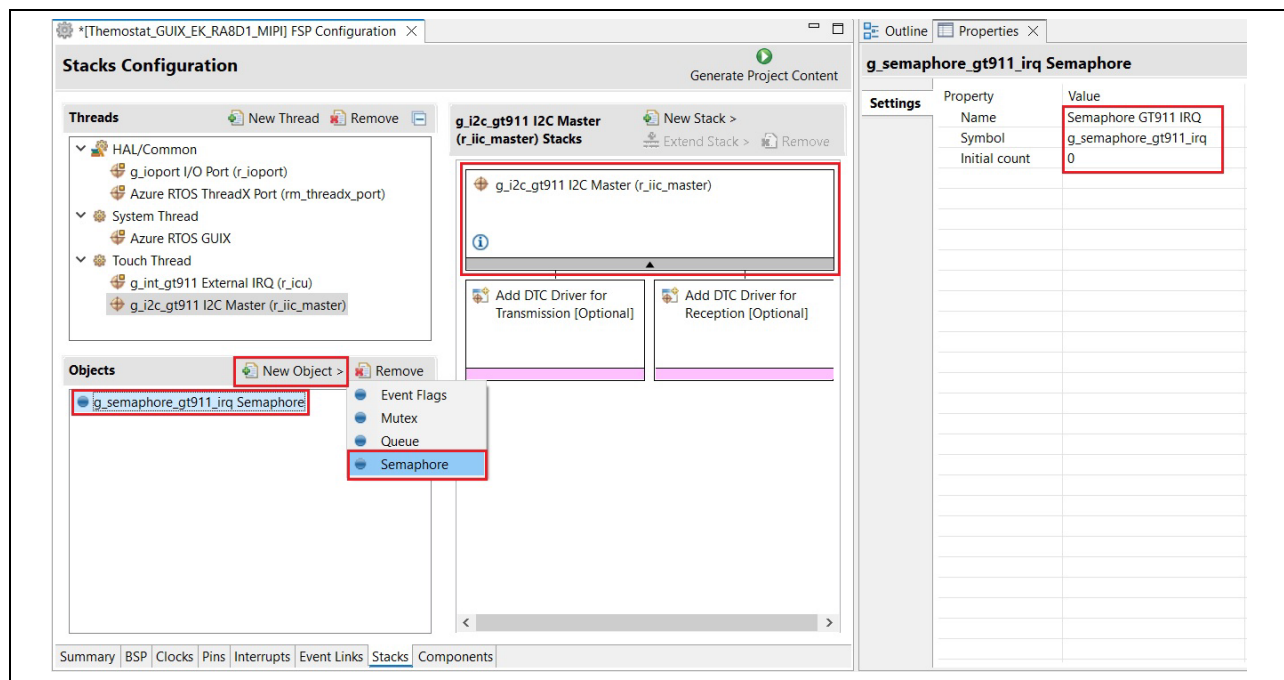


Figure 44. Adding Touch Semaphore

4.2.10 Now add a second: the **Semaphore GT911 I2C** with the symbol **g_semaphore_gt911_i2c**. Set the Initial Count to **0**. This semaphore signals the Touch thread when a touch event occurs. The Touch thread then sends the touch event to GUIX for processing.

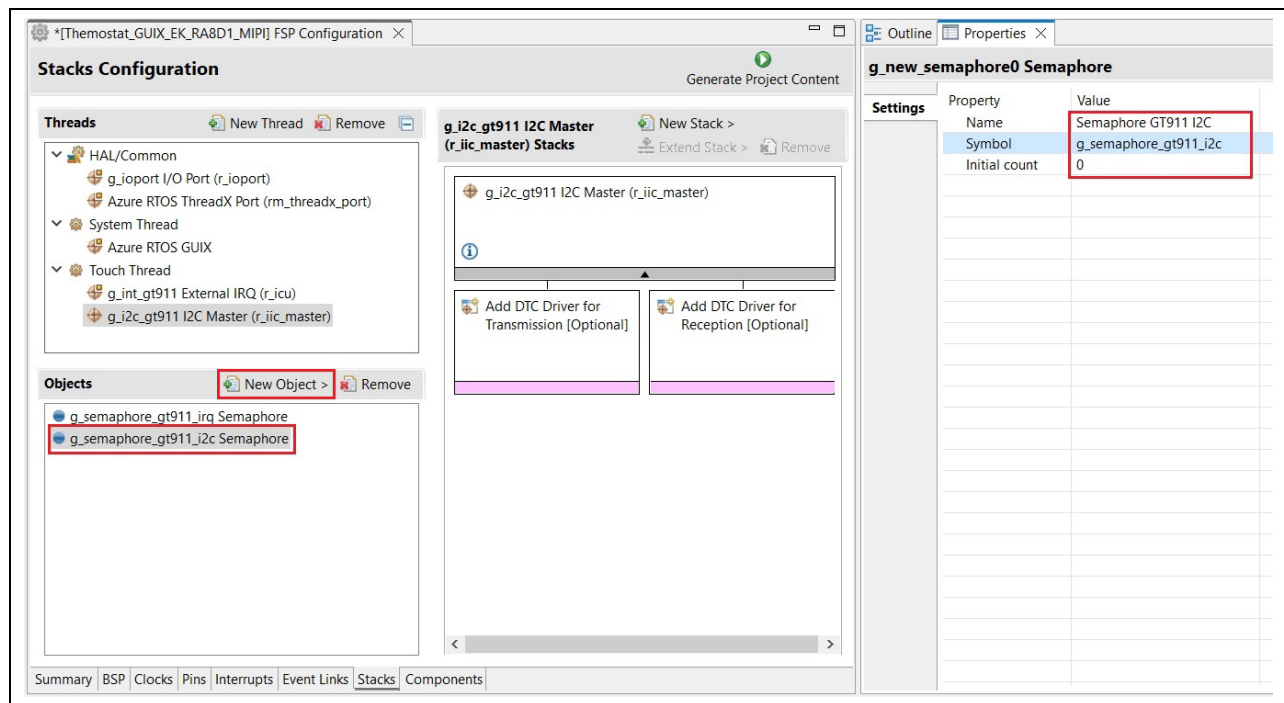


Figure 45. Adding I2C Semaphore

4.2.11 Use the **Add New Object > Event Flags** to add the **GT911 Event Flags** with the symbol **g_event_flags_gt911**, as shown below.

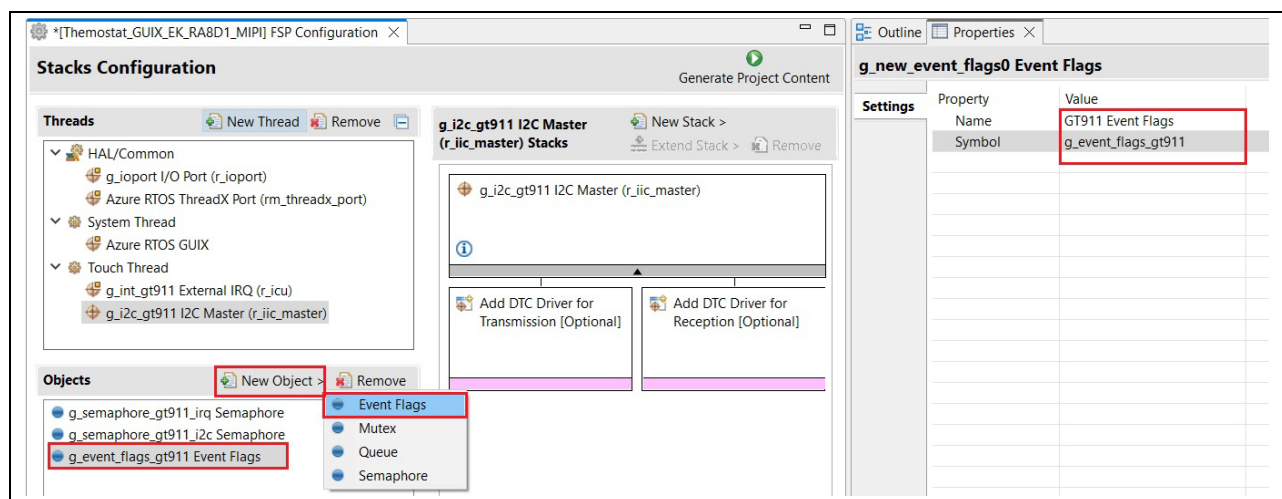


Figure 46. Adding Event Flags

4.2.12 In RA Configurator, click [Generate Project Content](#) to generate project content.

4.2.13 From the Source file folder accompanying this application note, copy the files in the **4.2.13** folder to the e² studio's **src** folder and overwrite all files that already exist. The contents of **4.2.13** are:

- SEGGER_RTT (folder)
- common_utils.h
- hmi_event_handler.c
- system_api.h
- system_cfg.h
- system_thread_entry.c
- system_thread_entry.h
- touch_thread_entry.c

4.2.14 Code highlight: Review the code below from `touch_thread_entry.c` which is responsible for retrieving touch data when a touch IRQ happens and sending information about the touch event to GUIX for processing.

```

/* Get touch data from the FT5X06 */
ft5x06_payload_get (&touch_data);

/* Send touch data*/
if (1 == touch_data.num_points)
{
    gxe.gx_event_payload.gx_event_pointdata.gx_point_x = touch_data.point[0].x;
    gxe.gx_event_payload.gx_event_pointdata.gx_point_y = touch_data.point[0].y;
    gxe.gx_event_type = GX_EVENT_PEN_DOWN;
    gx_system_event_send (&gxe);
}
else if (GX_EVENT_PEN_DOWN == gxe.gx_event_type) // @suppress("10.2b If statement")
{
    gxe.gx_event_type = GX_EVENT_PEN_UP;
    gx_system_event_send (&gxe);
}

```

Figure 47. Touch control code in touch_thread_entry.c

4.2.15 All the screens designed in the Azure RTOS GUIX Studio project are now created in system_thread_entry.c

```
/* Create the widget and attached to root window.*/
gx_err = gx_studio_named_widget_create("Splash", (GX_WIDGET *) p_root, (GX_WIDGET **) &p_splash_screen);
if(GX_SUCCESS != gx_err)
{
    APP_ERR_TRAP(FSP_ERR_ASSERTION);
}

gx_err = gx_studio_named_widget_create ("Settings", GX_NULL, (GX_WIDGET **) &p_settings_screen);
if(GX_SUCCESS != gx_err)
{
    APP_ERR_TRAP(FSP_ERR_ASSERTION);
}

gx_err = gx_studio_named_widget_create ("MainPage", GX_NULL, (GX_WIDGET **) &p_mainpage_screen);
if(GX_SUCCESS != gx_err)
{
    APP_ERR_TRAP(FSP_ERR_ASSERTION);
}

gx_err = gx_studio_named_widget_create ("Thermostat", GX_NULL, (GX_WIDGET **) &p_thermostat_screen);
if(GX_SUCCESS != gx_err)
{
    APP_ERR_TRAP(FSP_ERR_ASSERTION);
}

gx_err = gx_studio_named_widget_create ("Help", GX_NULL, (GX_WIDGET **) &p_help_screen);
if(GX_SUCCESS != gx_err)
{
    APP_ERR_TRAP(FSP_ERR_ASSERTION);
}
```

Figure 48. Create all screens defined in the GUIX project

4.2.16 The code is marked in red in hmi_event_handler.c handles touch events when the Thermostat and Settings buttons are clicked. Refer to hmi_event_handler.c for more details.

```

* @brief Handles all events on the main screen.
UINT mainpage_event(GX_WINDOW *widget, GX_EVENT *event_ptr)
{
    UINT gx_err = GX_SUCCESS;
    switch (event_ptr->gx_event_type)
    {
        case GXEVENT_MSG_UPDATE_TEMPERATURE:
            /** Update temperature text. */
            update_local_temp_string();
            update_text((GX_WIDGET *) widget, ID_TEMP_TEXT, g_local_temp_str);
            update_text((GX_WIDGET *) widget, ID_TEMP_TEXT_2, g_local_temp_str);
            break;
        case GXEVENT_MSG_TIME_UPDATE:
            update_time ((GX_WIDGET *) widget, &g_gui_state);
            update_date((GX_WIDGET *) widget, &g_gui_state);
            break;
        case GXEVENT_MSG_FAN_TOGGLE:
            if (g_gui_state.fan_on)
            {
                show_hide_widget((GX_WIDGET *) widget, ID_FAN_ICON, 1);
            }
            else
            {
                show_hide_widget((GX_WIDGET *) widget, ID_FAN_ICON, 0);
            }
            break;

        case GX_SIGNAL(ID_THERMO_BUTTON, GX_EVENT_CLICKED):
            /** Shows the thermostat control screen. */
            toggle_screen (p_thermostat_screen, p_mainpage_screen);
            break;

        case GX_SIGNAL(ID_SETTINGS_BUTTON, GX_EVENT_CLICKED):
            /** Shows the settings screen and saves which screen the user is currently viewing. */
            toggle_screen (p_settings_screen, widget);
            break;

        case GX_SIGNAL(ID_DAY_DOWN, GX_EVENT_CLICKED):
            /** Create message to set date. */
            p_message->msg_id.event_b.class_code = SF_MESSAGE_EVENT_CLASS_TIME;
            p_message->msg_id.event_b.code = SF_MESSAGE_EVENT_SET_DATE;
            g_gui_state.time.tm_mday -= 1;
            time.tm_mday = -1;
            p_message->msg_payload.time_payload.time = time;
            send_message(p_message);
            break;

        case GX_SIGNAL(ID_BACK_BUTTON, GX_EVENT_CLICKED):
            /** Returns to main screen. */
            toggle_screen (p_mainpage_screen, widget);
            break;

        case GX_EVENT_SHOW:
            gx_err = gx_window_event_process(widget, event_ptr);
            if(GX_SUCCESS != gx_err) {
                APP_ERR_TRAP(FSP_ERR_ASSERTION);
            }
            settings_menu_highlight(widget, ID_SETTINGS_MENUITEM_NONE);
            settings_item_show (widget, ID_SETTINGS_CONTENT_NONE, cur_settings_screen);
            cur_settings_screen = ID_SETTINGS_CONTENT_NONE;
    }
}

```

Figure 49. Handle button events from touch input

4.2.17 Build, Download, and Run the e² studio project. At this point, you will be able to go back and forth from the Main Page screen to the Thermostat screen and Settings screen using the Thermostat and Settings buttons on the Main Page screen and the “Back” button on the other two screens.

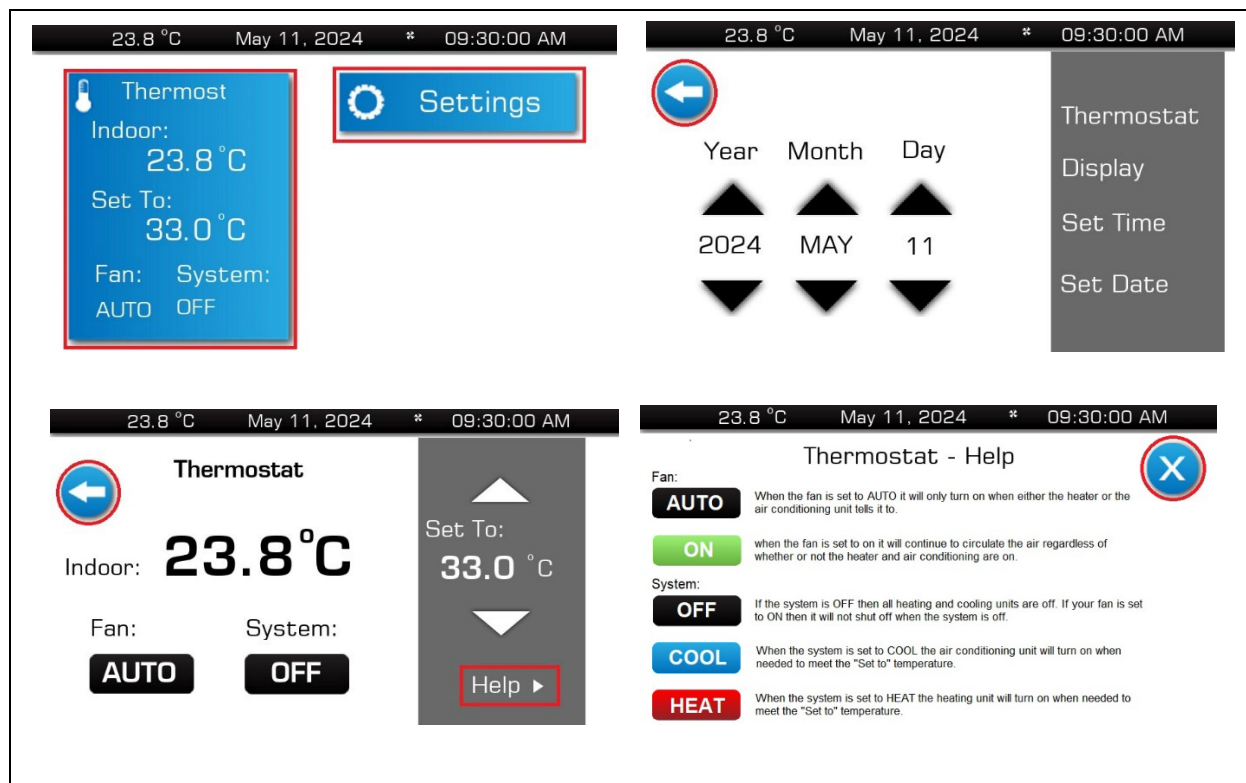


Figure 50. Navigating between the Main Page Screen and the Thermostat Screen

5. Control MIPI LCD Backlight

5.1 Overview

In this section, you will use a PWM output pin of a GPT timer to control the intensity (brightness) of the LCD backlight.

5.2 Procedural Steps

5.2.1 In the LCD board schematics below, the DISP_BLEN signal, which is connected to the P404 on the RA8D1 MCU, is configured in PWM mode to control the intensity of the backlight display.

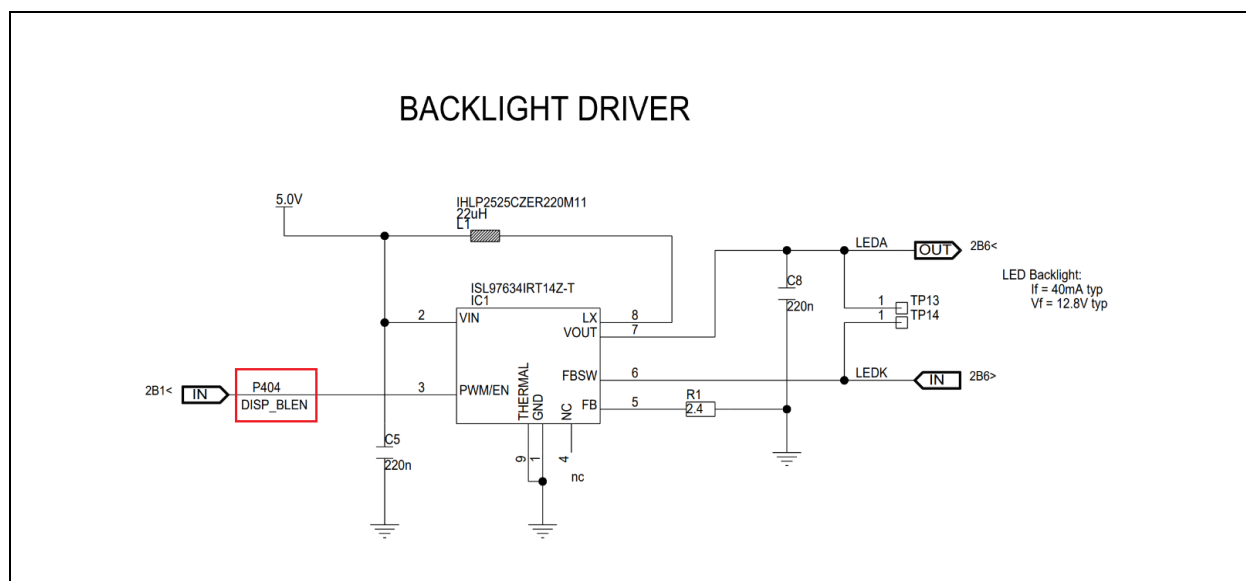


Figure 51. MIPI LCD Board Schematic

5.2.2 To configure P404 in PWM output mode, we first need to disable its current configuration from the Pins tab of the configuration.xml. Navigate to **P4 > P404** and set the **Mode** to **Disabled**. **Save this change to the configuration.xml before moving to the next step.**

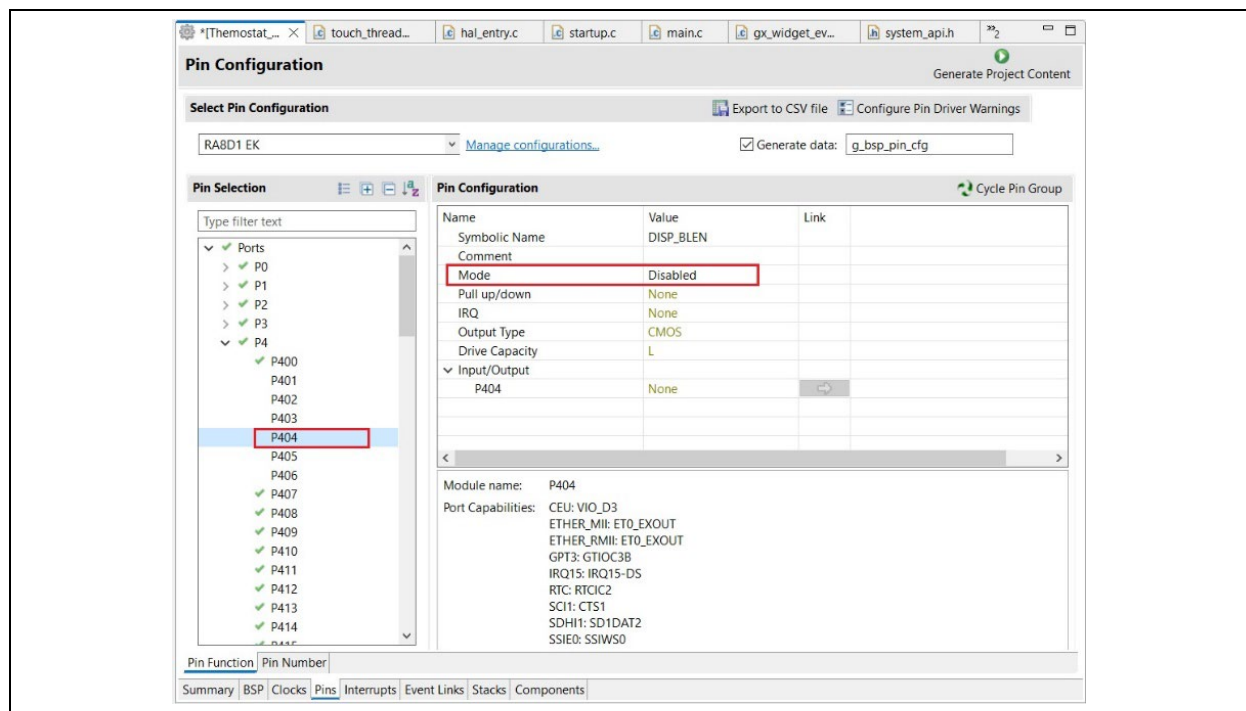


Figure 52. Disabling P404 in Pin Configuration

5.2.3 Still, in the Pins tab, navigate to the **Timers:GPT > GPT3**. Set the Pin Group Selection to **Mixed**, Operation Mode to **GTIOCA or GTIOCB**, and then set the Input/Output of **GTIOCB** to **P404** as shown in the image below:

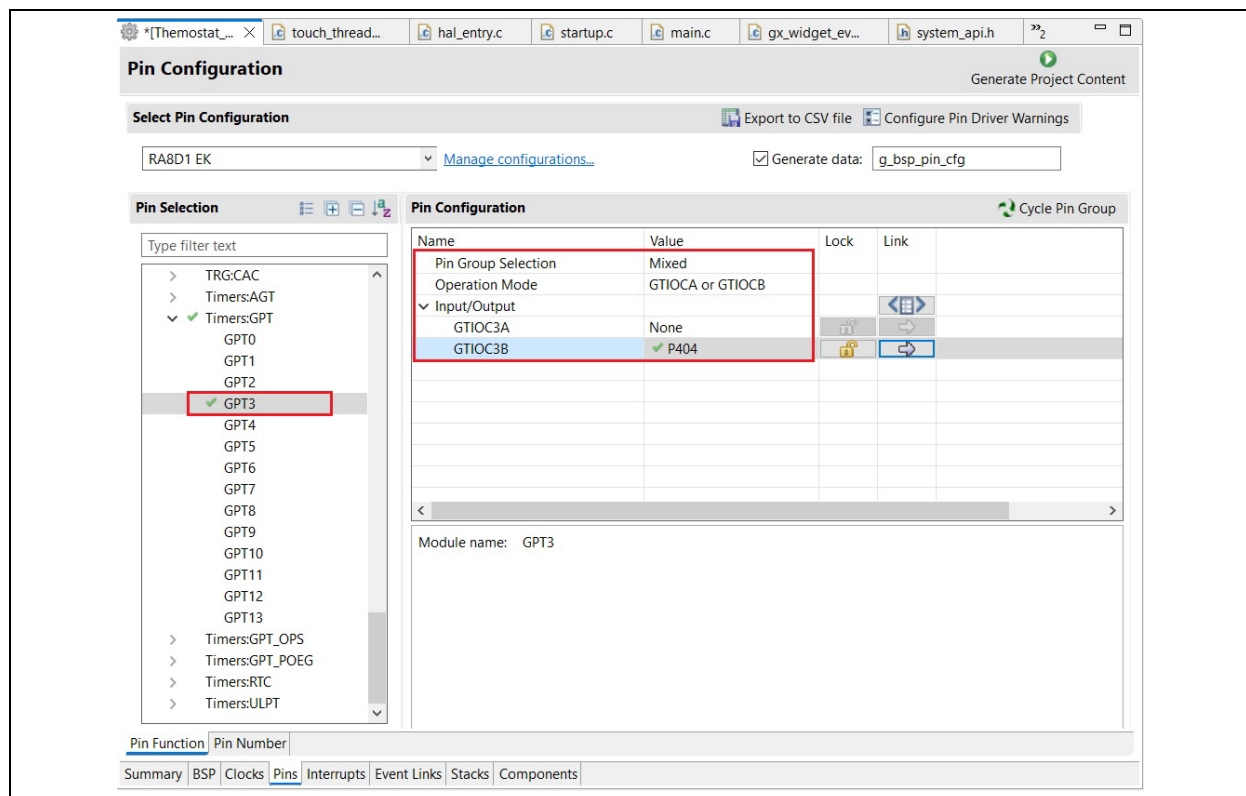


Figure 53. Setting P404 as GPT3 - GTIOCB Output in Pin Configuration

5.2.4 Navigate to the **Stacks** tab. Make sure the **System Thread** is selected in the Threads window and click on **Add New Stack > Timers > Timer, General PWM (r_gpt)**. Use the image to make sure the settings in the Properties tab of the newly added GPT module are set correctly for this application.

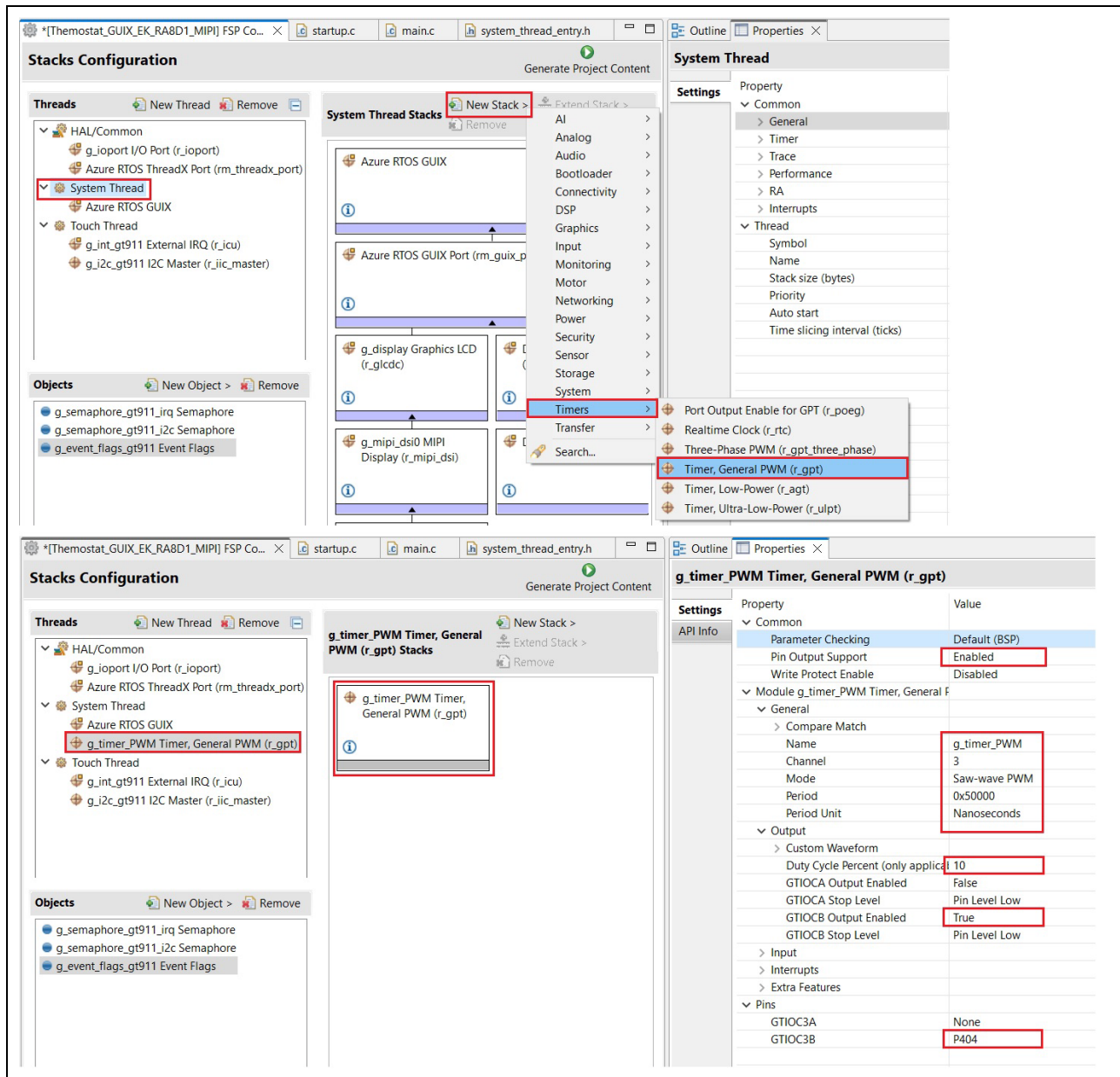


Figure 54. Adding Timer Driver on r_gpt to System Thread

Even though the duty cycle of PWM output is purposely set to **10%** here, it will be changed to **50%** later in the code.

5.2.5 In RA Configurator, click **Generate Project Content** to save the configuration.xml and generate project content.

5.2.6 From the Source file folder accompanying this application note, copy the files in the **5.2.6** folder to the e² studio's **src** folder and overwrite all files that already exist. The contents of **5.2.6** are:

- common_utils.h
- hmi_event_handler.c
- system_thread_entry.c

- system_thread_entry.h
- brightness.c
- brightness.h
- system_api.h
- system_cfg.h
- touch_thread_entry.c

5.2.7 Code Highlight: The brightness_up() and brightness_down() functions in brightness.c are used to set the **PWM duty cycle**, as shown below:

```
/* Get the current period setting. */
R_GPT_InfoGet(&g_timer_PWM_ctrl, &info);

/* Calculate the desired duty cycle based on the current period. Note that if the period could be larger than
 * UINT32_MAX / 100, this calculation could overflow. */
duty_cycle_count = (uint32_t) ((info.period_counts * brightness)/GPT_PWM_MAX_PERCENT);
err = R_GPT_DutyCycleSet(&g_timer_PWM_ctrl, duty_cycle_count, GPT_IO_PIN_GTIOCB);
if (FSP_SUCCESS == err)
{
    *p_brightness = (uint8_t)brightness;
}
```

Figure 55. Function to update the duty cycle during run-time

5.2.8 Code Highlight: Look at gpt_timer_PWM_Setup() function in system_thread_entry.c, you will see that brightness (**duty cycle of PWM output**) is set to 50 percent.

```
* @brief This function is setting up GPT/PWM timer
static fsp_err_t gpt_timer_PWM_setup(void) // @suppress("8.1b, 8.1f Non-API function naming")
{
    fsp_err_t err = FSP_SUCCESS;
    /* Open GPT */
    err = R_GPT_Open(&g_timer_PWM_ctrl, &g_timer_PWM_cfg);
    if(FSP_SUCCESS != err)
    {
        return err;
    }
    /* Enable GPT Timer */
    err = R_GPT_Enable(&g_timer_PWM_ctrl);
    /* Handle error */
    if (FSP_SUCCESS != err)
    {
        return err;
    }
    /* Start GPT timer */
    err = R_GPT_Start(&g_timer_PWM_ctrl);
    if(FSP_SUCCESS != err)
    {
        return err;
    }

    /* Set brightness (LCD backlight) level: 50 = (45+5) */
    g_gui_state.brightness = 45;
    brightness_up(&g_gui_state.brightness);

    return err;
}
```

Figure 56. Setting the duty cycle of PWM output in the System Thread

5.2.9 Build, Download, and Run the e² studio project. By clicking the **Settings** button on the **Main Page** screen, you can access the **Settings** screen.

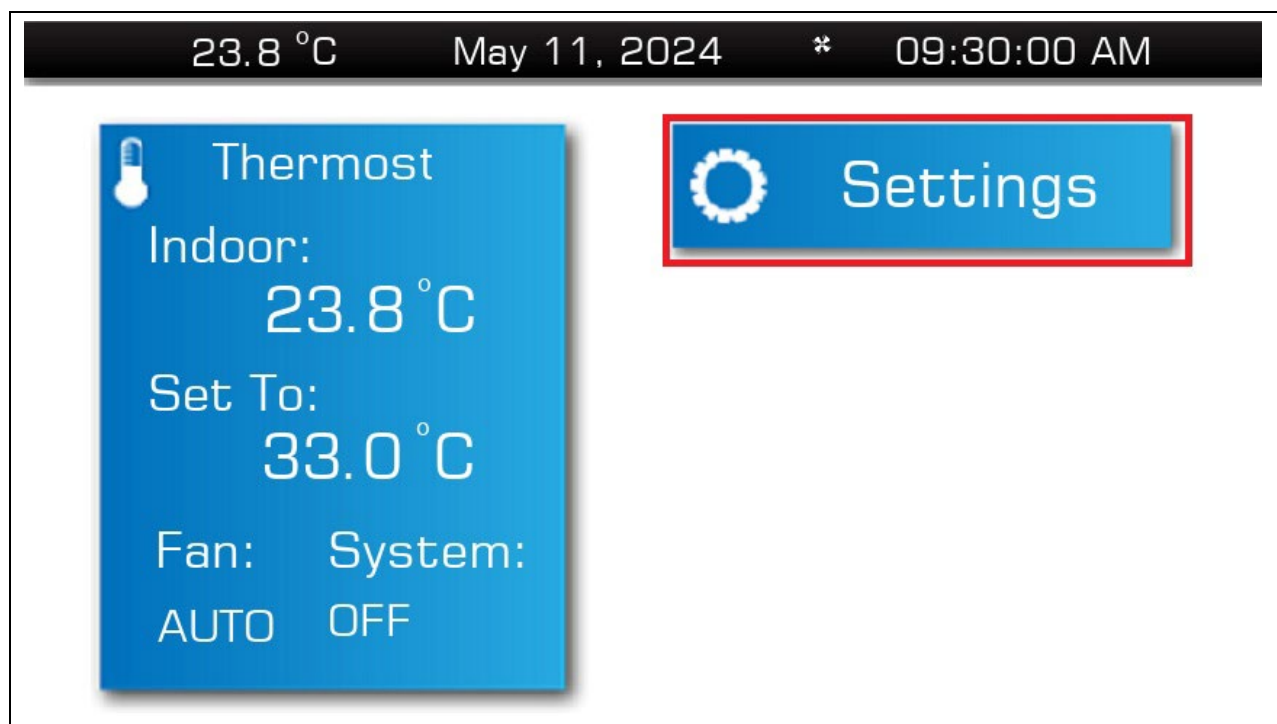


Figure 57. Settings Button on Main Page Screen

5.2.10 PWM output can be measured on pin P404. Here, the brightness is set to 50%.

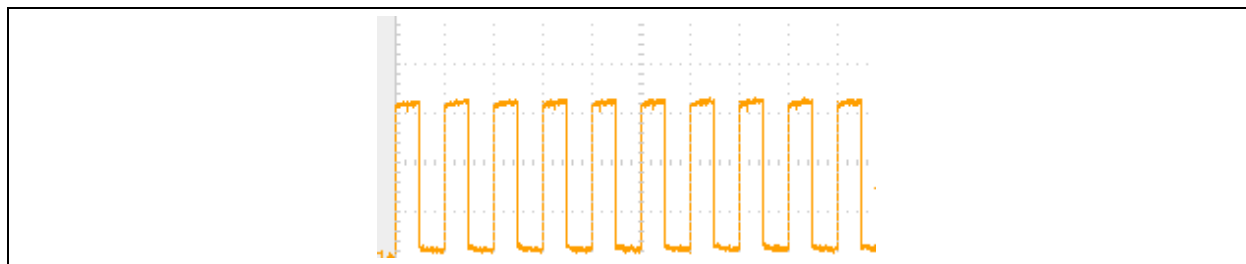


Figure 58. PWM Output on P603 at 50% Brightness

5.2.11 Click the “**Display**” menu on the **Settings** screen; you can use the “**Up**” and “**Down**” buttons to change the brightness of the LCD backlight.

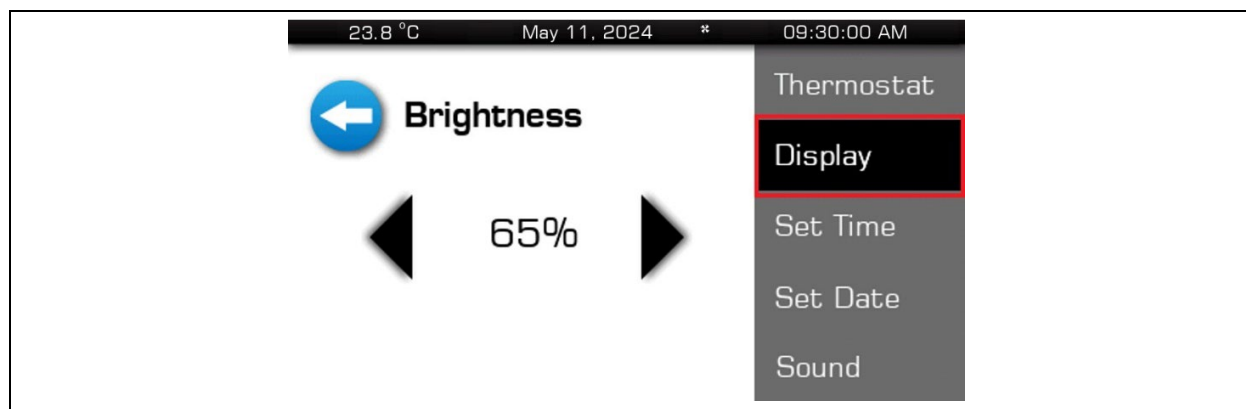


Figure 59. Display on Settings Screen

5.2.12 PWM output was measured on pin P404 after changing brightness to 65%.

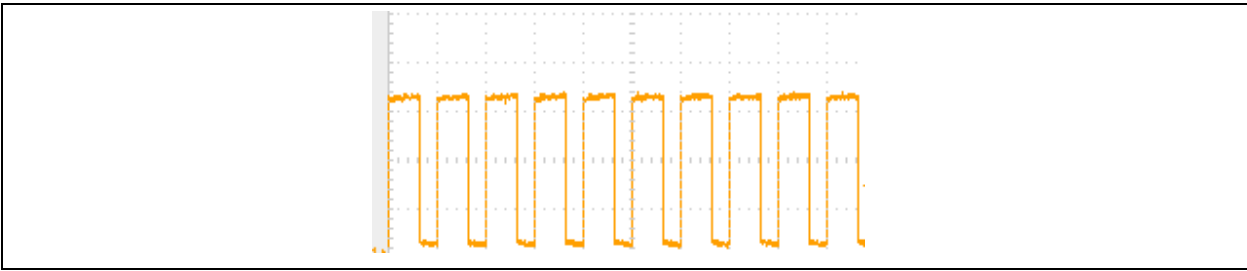


Figure 60. PWM Output on P404 at 65% Brightness

6. Update Date/Time and Temperature

6.1 Overview

In this section, you will enable the RTC controller as a timekeeper and one ADC channel to read the MCU die’s temperature sensor and use it as the Thermostat temperature data.

6.2 Procedural Steps

6.2.1 In the configuration.xml under the Stacks tab, create a **New Thread > Temperature Time Thread**, and match the image below to set the properties of the new thread.

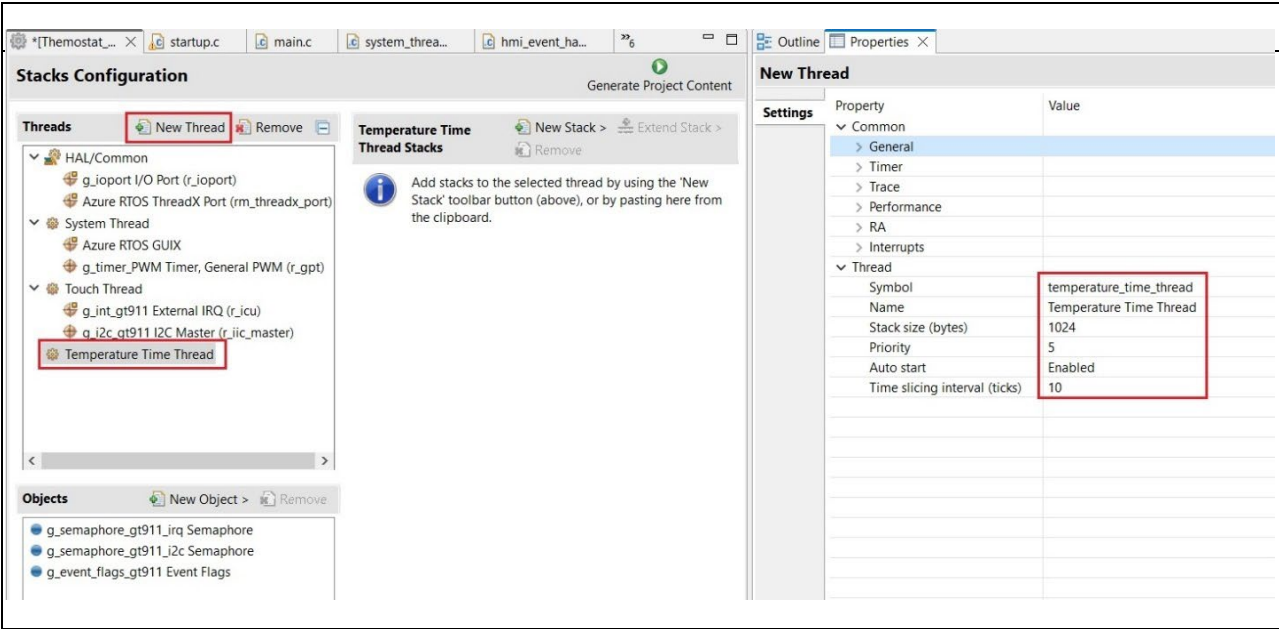


Figure 61. Create Temperature Time Thread

6.2.2 Make sure the **Temperature Time Thread** is selected and hit **Add New Stack > Timers > Realtime Clock (r_rtc)**. Match the r_rtc settings to those in the image below:

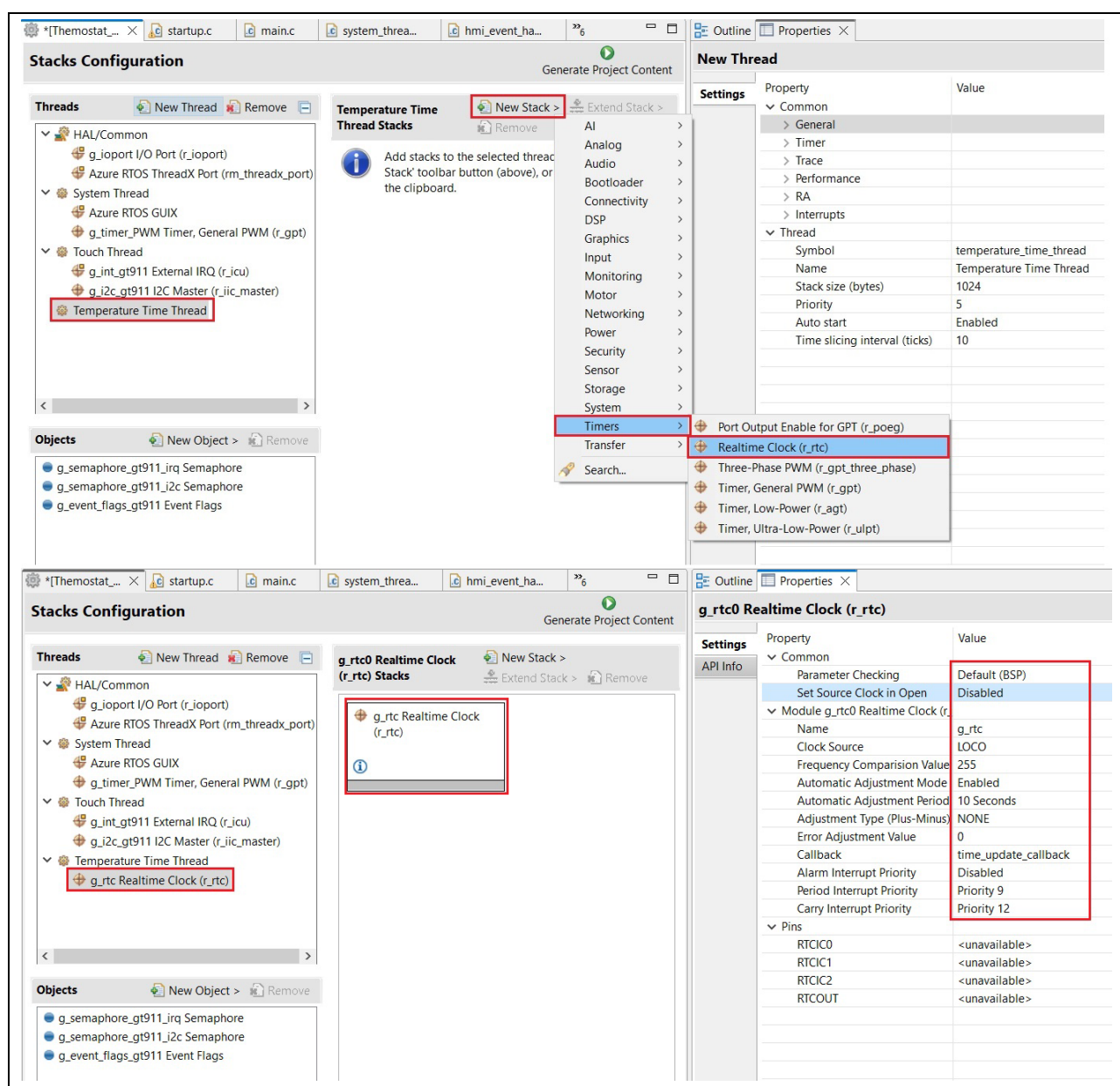


Figure 62. Adding RTC Driver on g_rtc to Temperature Time Thread and Setting Properties

6.2.3 The System Thread is where the temperature value will be read from the ADC. Select the **System Thread** in the Threads window and press **Add New Stack > Analog > ADC (r_adc)**. Match the `r_adc` settings to those in the image below:

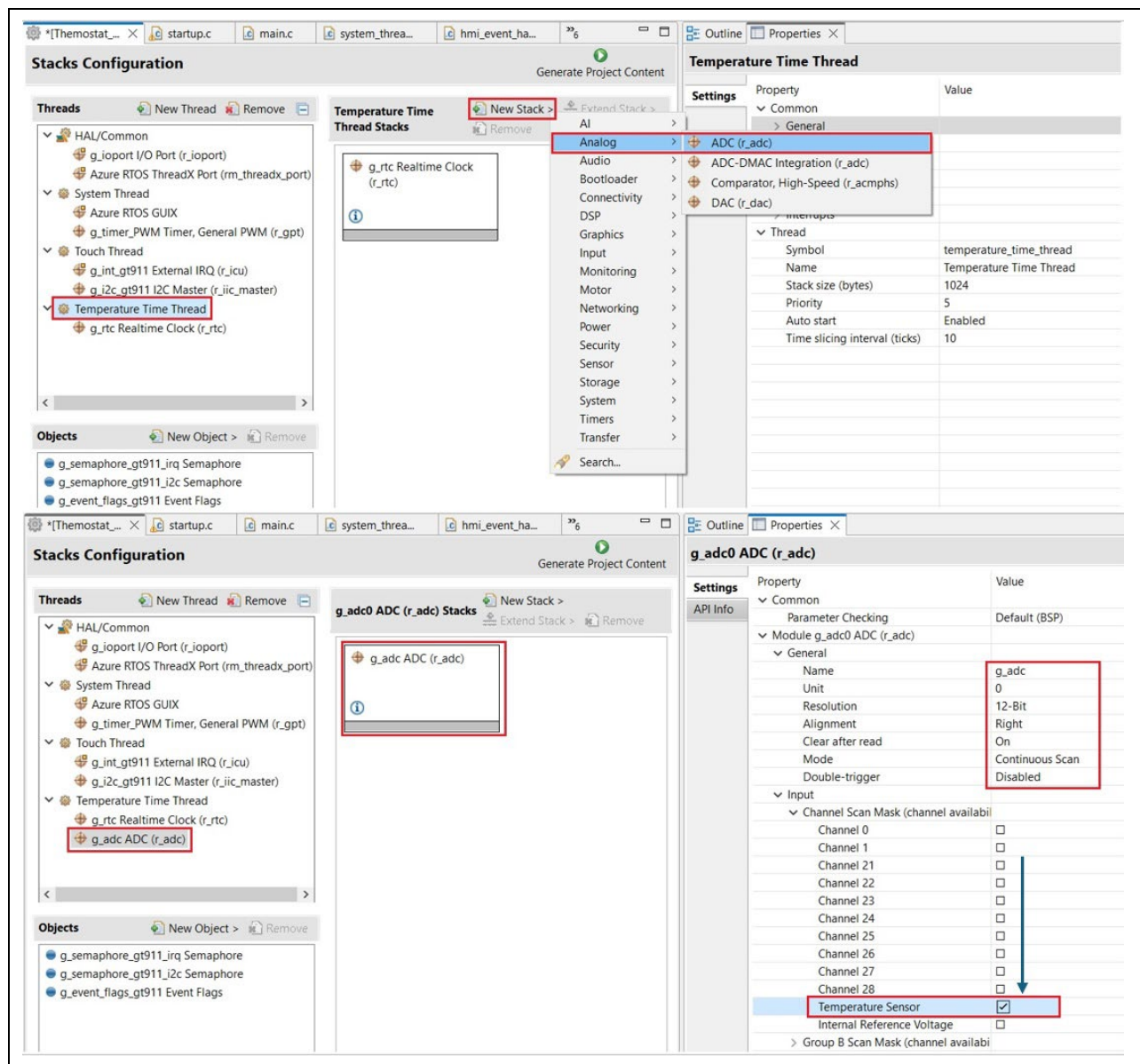


Figure 63. Adding ADC Driver on `r_adc` to System Thread and Setting Properties

- 6.2.4** Use the Objects window button **Add New Object Button > Semaphore** to add the **Timer Semaphore** with the symbol **g_timer_semaphore**. Set the Initial Count to **0**. This semaphore will trigger the timing between the System Thread and the Temperature Time Thread.

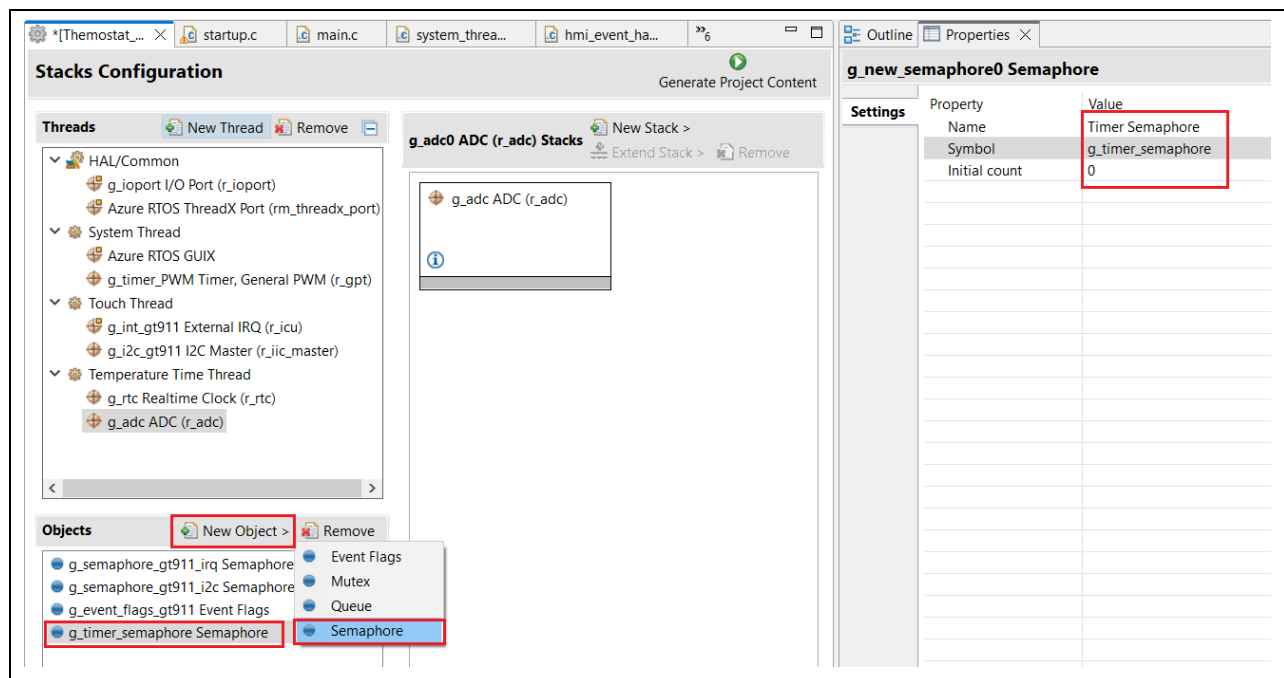


Figure 64. Creating g_timer_semaphore

- 6.2.5** Use the **New Object > Queue** to create a **System Message Queue** with the symbol **system_msg_queue**. Give it a Message Size of **16** and a Queue Size of **64**, as shown below.

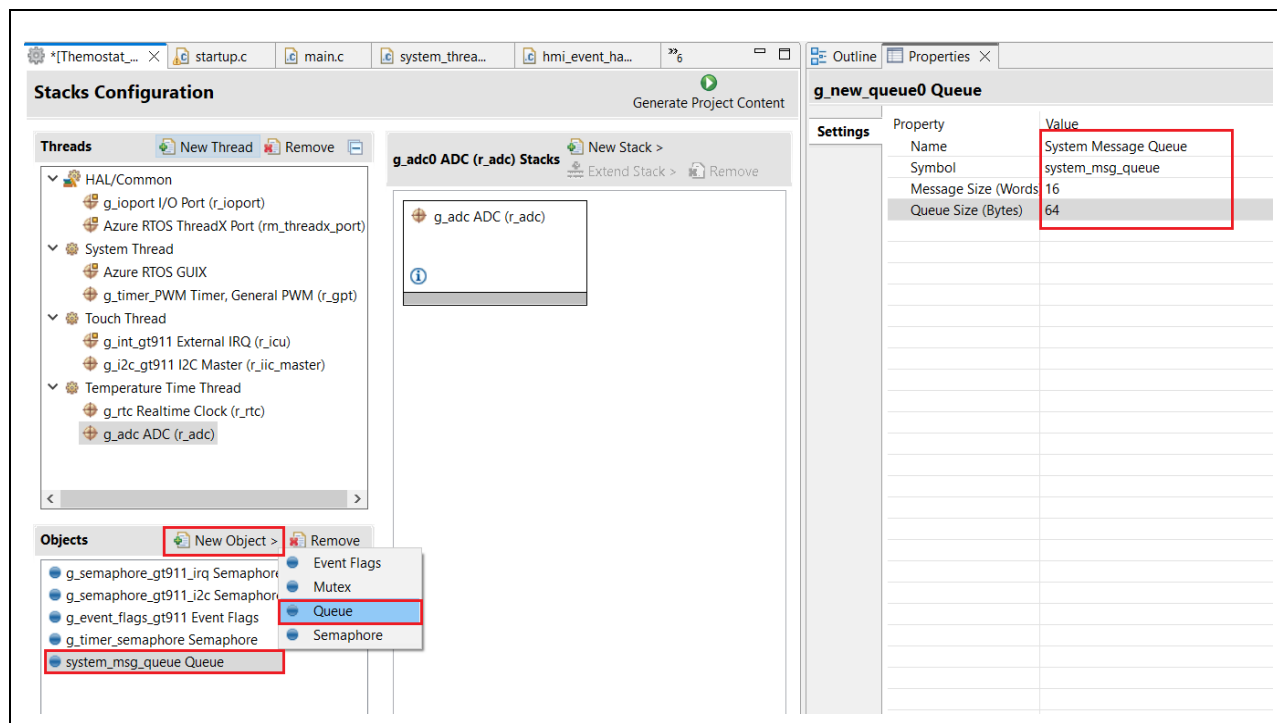


Figure 65. Creating system_msg_queue Queue

6.2.6 Use the **New Object > Mutex** to create a **System Mutex** with the symbol **g_sys_mutex**. **Disable** the Priority Inheritance.

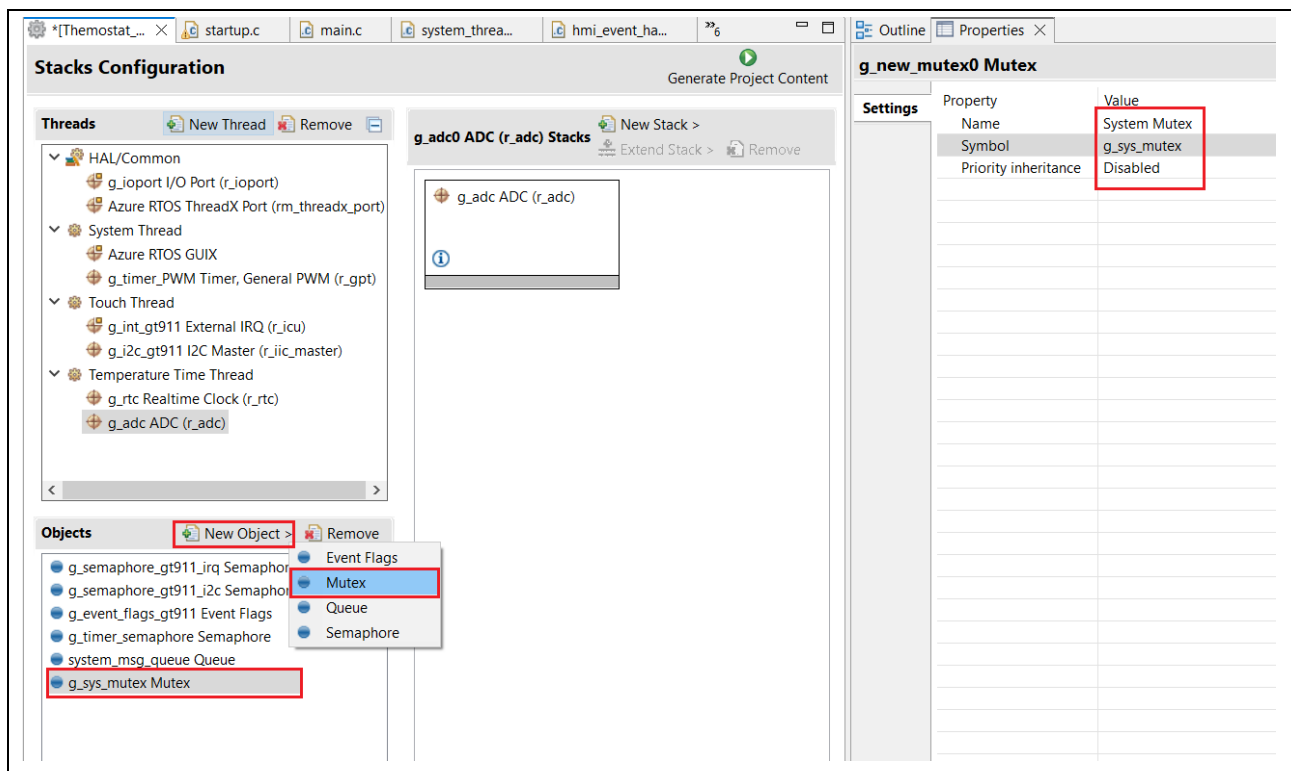


Figure 66. Creating g_sys_mutex Queue

6.2.7 In RA Configurator, click **Generate Project Content** to save the configuration.xml and generate project content.

6.2.8 From the Source file folder accompanying this application note, copy the files in the **6.2.8** folder to the e² studio's **src** folder and overwrite all files that already exist. The contents of **6.2.8** are:

- hal_entry.c
- brightness.c
- brightness.h
- common_utils.h
- hmi_event_handler.c
- system_api.h
- system_cfg.h
- system_thread_entry.c
- system_thread_entry.h
- system_time.c
- system_time.h
- temperature_time_thread_entry.c
- touch_thread_entry.c

6.2.9 Code Highlight: In System Thread, date/time data and temperature data get updated every second. It then sends out events to trigger GUIX updates. The following is an example of handling temperature and time update events in the Main Page screen event handler.

```
case GXEVENT_MSG_UPDATE_TEMPERATURE:
    /** Update temperature text. */
    update_local_temp_string();
    update_text((GX_WIDGET *) widget, ID_TEMP_TEXT, g_local_temp_str);
    update_text((GX_WIDGET *) widget, ID_TEMP_TEXT_2, g_local_temp_str);
    break;
case GXEVENT_MSG_TIME_UPDATE:
    update_time ((GX_WIDGET *) widget, &g_gui_state);
    update_date((GX_WIDGET *) widget, &g_gui_state);
    break;
```

Figure 67. Update the temperature and time each second

6.2.10 Build, Download, and Run the e² studio project. You will see time and temperature updates every second.

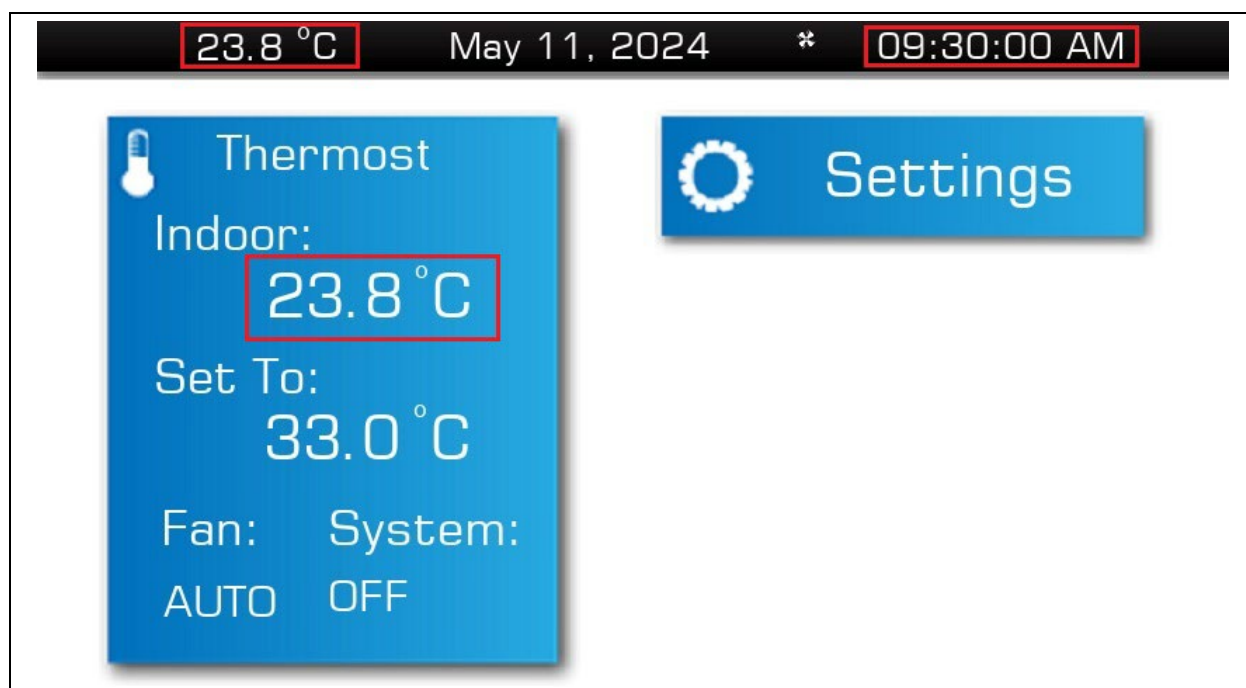


Figure 68. Time, Temperature on Main Page Screen

7. Testing and Debugging the Complete Thermostat Project

7.1 Overview

In this section, you will import and run the complete Thermostat project that enables the date and time settings. Upon the user presses the date and time buttons on the settings screen, a message will be sent to the system thread to update the date and time; then the system thread will send a GUIX event to trigger a time display update on screens.

7.2 Procedural Steps

- 7.2.1** You can try the completed project in the completed_project folder; the project name is Thermostat_GUIX_EK_RA8D1_MIPI. Use the **“Rename & Import Existing C/C++ Project into Workspace”** feature of the **Import** menu in e2 studio to do so since you already had a project with the same in the workspace.

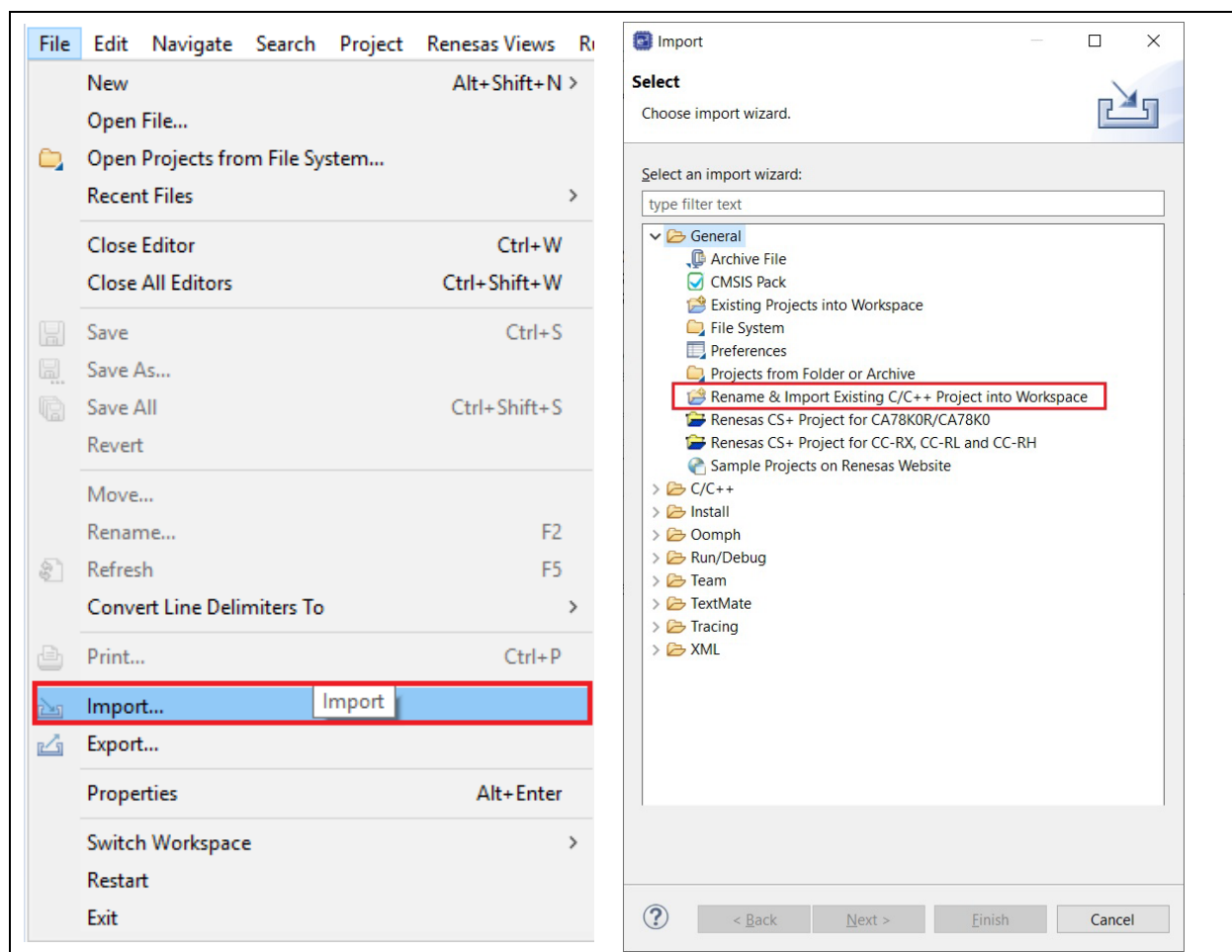


Figure 69. Rename & Import Existing C/C++ Project into Workspace on Import Menu

7.2.2 Once the project is imported, open the configuration.xml, click Generate Project Content, and build the project. There should be no errors or warnings, and you can hit Run twice to proceed with testing the application.

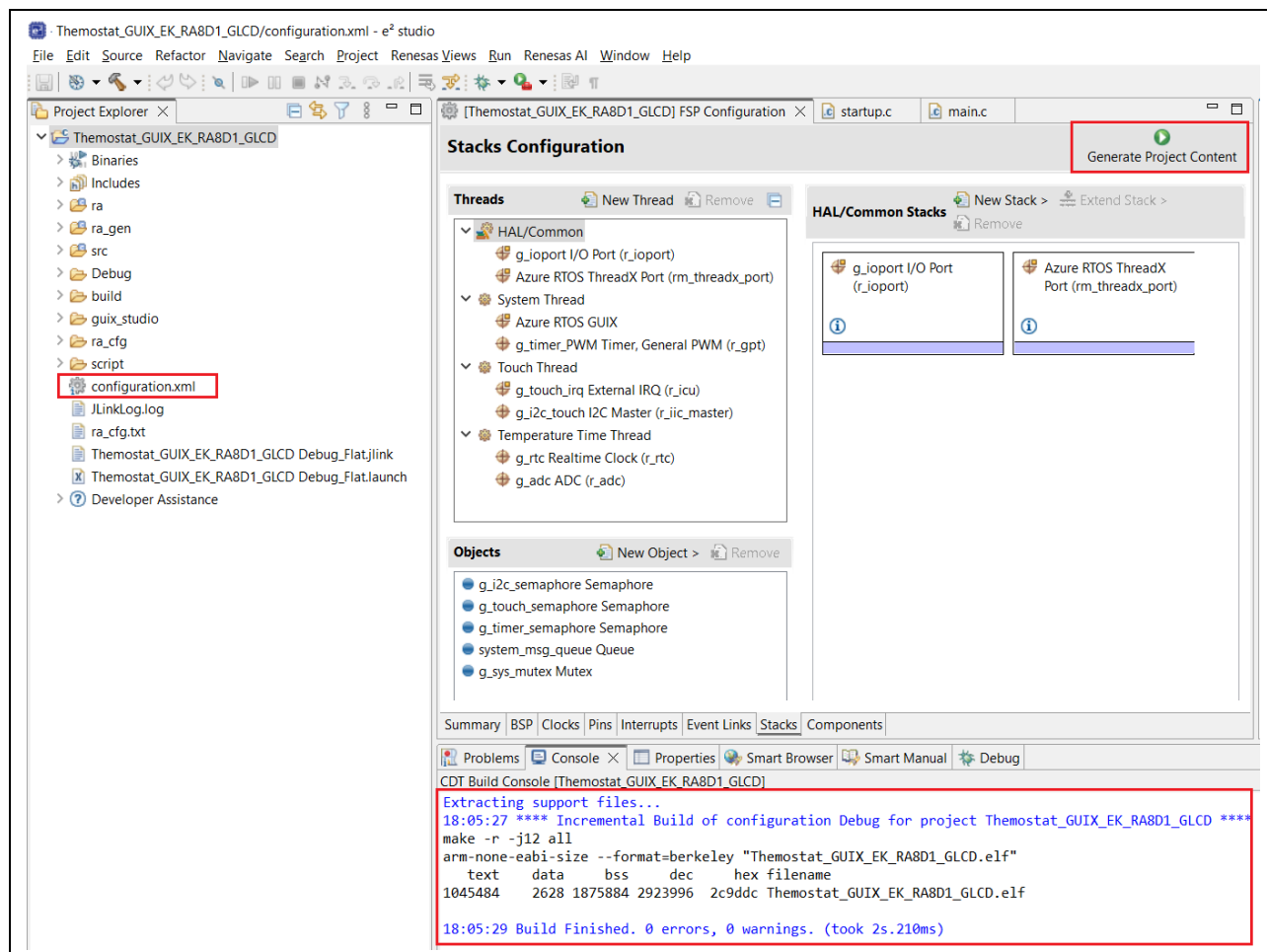


Figure 70. Project imported, Generate Project Content and Build Project.

8. Website and Support

Visit the following URLs to learn about key elements of the RA family, download components and related documentation, and get support:

RA Product Information	EK-RA8D1 - Evaluation Kit for RA8D1 MCU Group Renesas
RA Product Support Forum	renesas.com/ra/forum
RA Flexible Software Package	renesas.com/FSP
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct.15.24	—	Initial release

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/.