Introduction

This application, which is a Thermostat application, provides a reference for developing complex multi-threaded applications with a touch screen graphical Human Machine Interface (HMI) by using Renesas FSP and Azure RTOS GUIX. It describes steps to create a basic GUIX for FSP, integrates touch driver, 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. “In addition, FSP also provides Ethernet, USB, File System and other middleware stacks as well. This powerful suite of tools provides a comprehensive, integrated framework for rapid development of complex embedded applications.

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 application project provided. For more detailed information on Azure RTOS features, refer to the Azure RTOS User Manual.

The Graphics application is developed using the Renesas e^2 studio Integrated Solution Development Environment (IDE). e^2 studio is integrated with the FSP platform installer, which can be downloaded from Renesas website. The intuitive configurators and code generators in e^2 studio and FSP will help the application developers in creating such 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 application.
- 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

• e^2 studio IDE Version: 2025-04 (25.4.1)
• Renesas Flexible Software Package (FSP) v6.0.0.
• Azure RTOS GUIX Studio V6.4.0.

Hardware

• Renesas EK-RA8E2 kit (RA8E2 MCU Group)

Reference Manuals

• RA Flexible Software Package Documentation Release v6.0.0
• Azure RTOS GUIX and GUIX Studio v6.4.0.0
• Renesas RA8E2 Group User’s Manual Rev.1.0.0
• EK-RA8E2-v1.0 Schematics

Purpose

This document will guide you through the setup of an Azure RTOS GUIX touch screen interface Thermostat application in e^2 studio. This document will show how to configure the drivers and library included with the FSP. These will allow you to set up the parallel LCD Display Controller, the touch screen driver, and semaphores, queue, and Mutex 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 also 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 6
2.1 Overview 6
2.2 Procedural Steps 6
3. Using GUIX Widget Timer to Trigger a Screen Transition 29
3.1 Overview 29
4. Add Touch Driver to GUIX_THERMOSTAT_EK_RA8E2 Project 32
4.1 Overview 32
4.2 Procedural Steps 32
5. Control LCD Backlight 42
5.1 Overview 42
5.2 Procedural Steps 42
6. Update Date/Time and Temperature 46
6.1 Overview 46
6.2 Procedural Steps 47
7. Testing and debugging in A Full Function Project 53
7.1 Overview 53
7.2 Procedural Steps 53
8. Website and Support 54
Revision History 56

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 v2025-04.1/ FSP v6.0.0 and Azure RTOS GUIX Studio v6.4.0.0.

1.2 Procedural Steps

1. **If you already have e² studio with FSP v6.0.0, you can skip this step. Otherwise, you can download it from this link.**
2. You can get Azure RTOS GUIX Studio v6.4.0.0 from this link. If it goes well, you will see the window in the next step on the web browser.
   Note: It needs Microsoft Store to work on your PC to install Azure RTOS GUIX Studio.
3. Click Download to local PC and start installing Azure RTOS GUIX Studio.

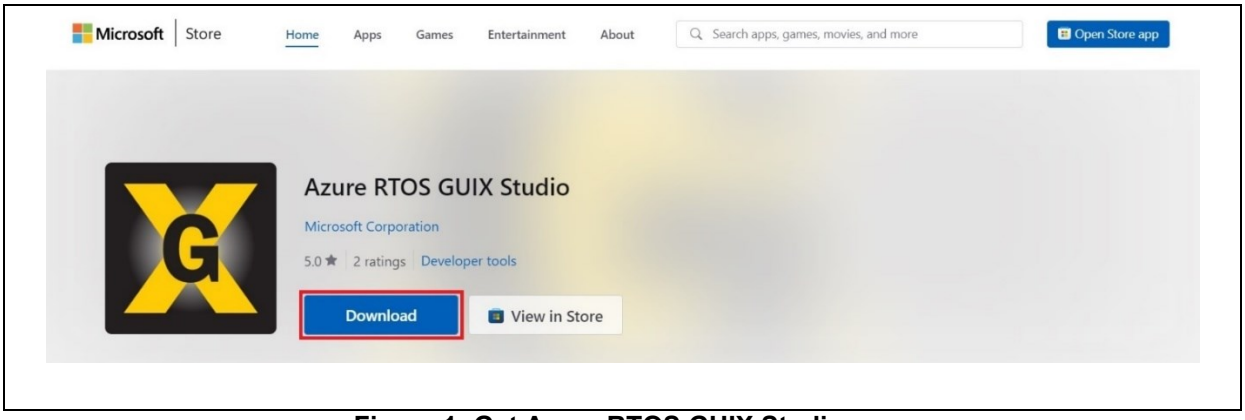


Figure 1. Get Azure RTOS GUIX Studio

4. Click downloaded file "**Azure RTOS GUIX Studio installer.exe**" and continue install Azure RTOS GUIX studio.

Figure 2. Install Azure RTOS GUIX Studio

5. Click "Open" to open "Azure RTOS GUIX Studio" App.

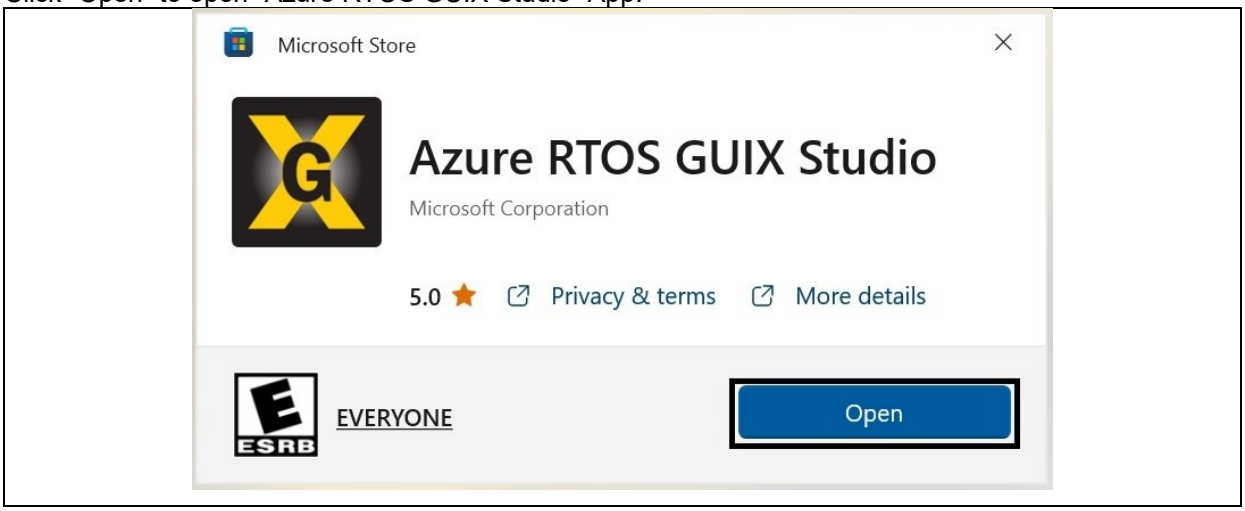


Figure 3. Click Open to Launch "Azure RTOS GUIX Studio".

6. Close Azure RTOS GUIX Studio. You will 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. This section also will show the user how to enable and use SDRAM. Setting Azure RTOS GUIX store in SDRAM.

2.2 Procedural Steps

1. Create a new RA C/C++ project. Named it **"guix_thermostat_ek_ra8e2"**.

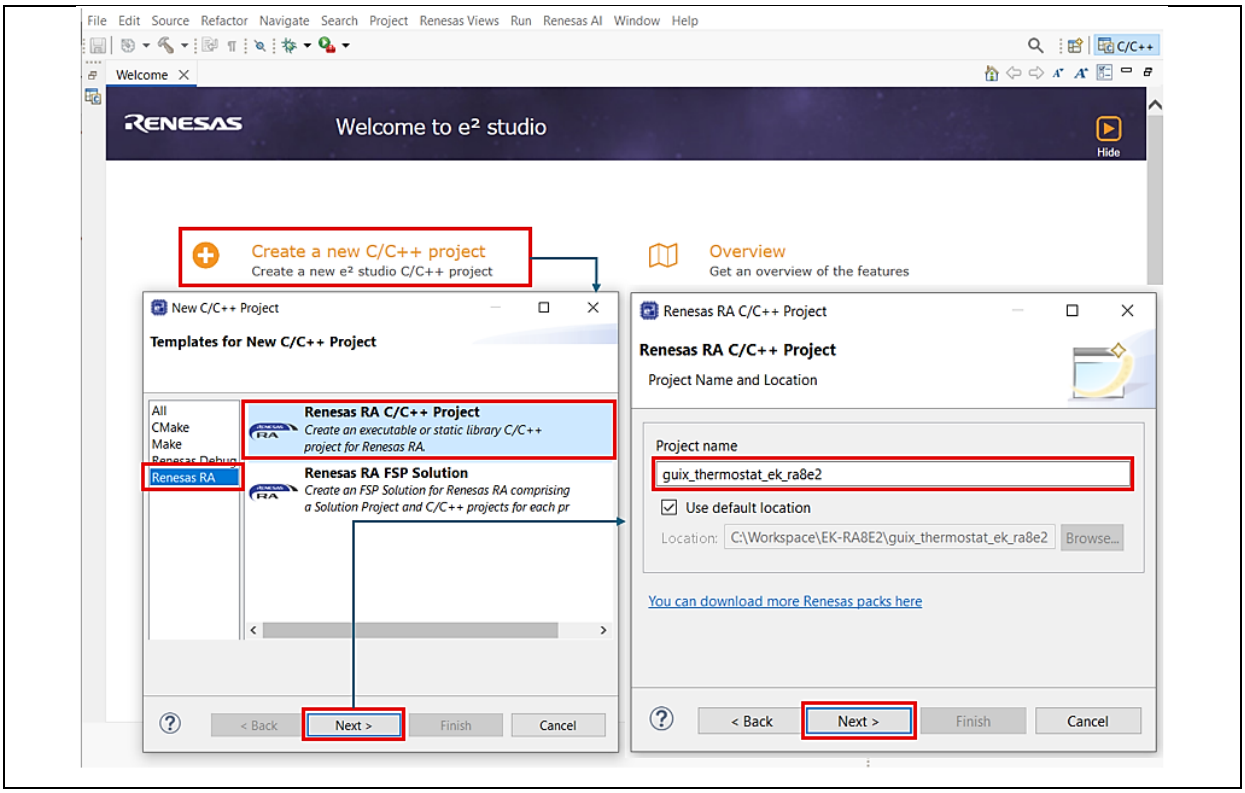


Figure 4. Create New Project

2. Select and set board to EK-RA8E2.

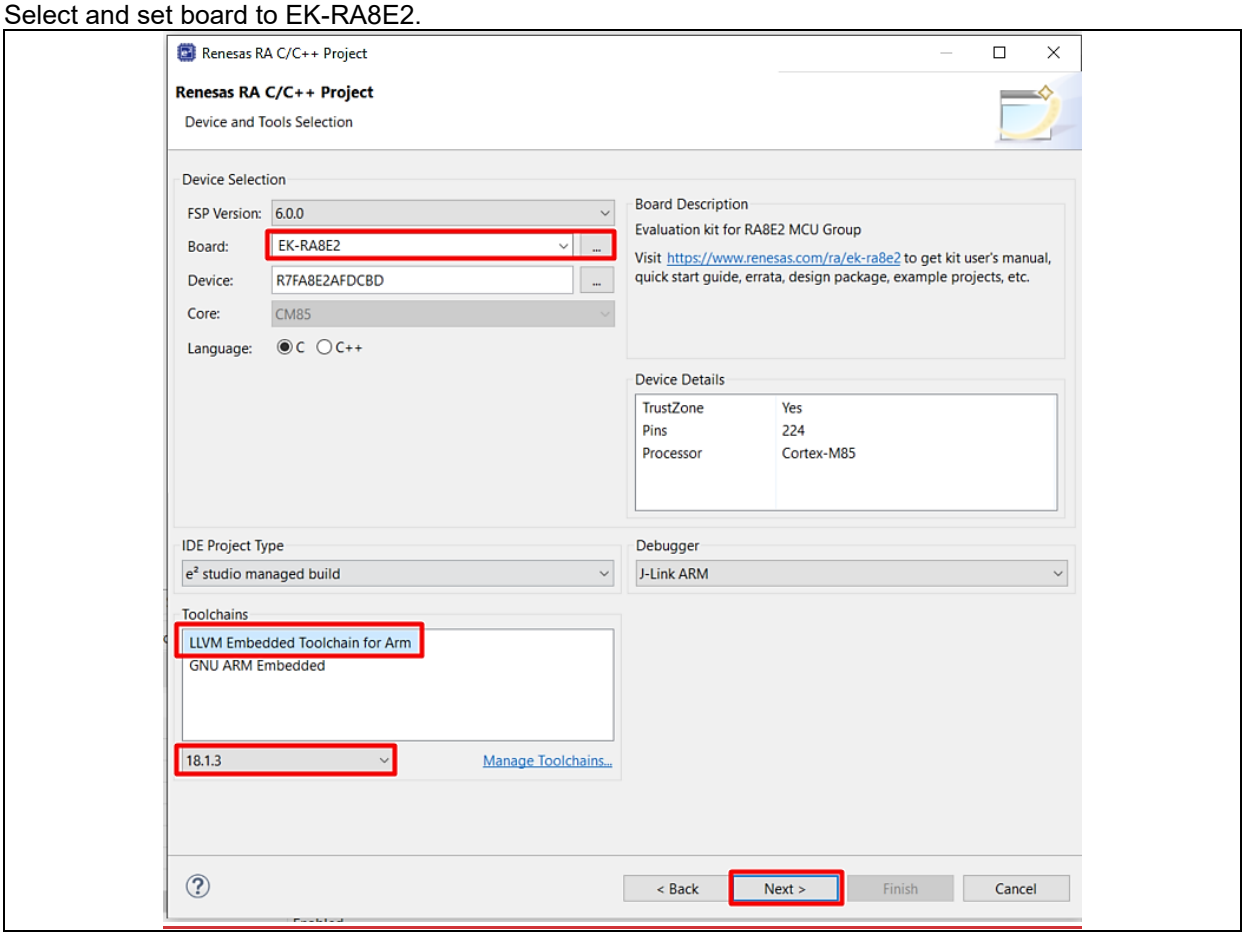


Figure 5. Select and Set board to EK-RA8E2

3. Select Project Type, Build Artifact and Azure RTOS ThreadX.

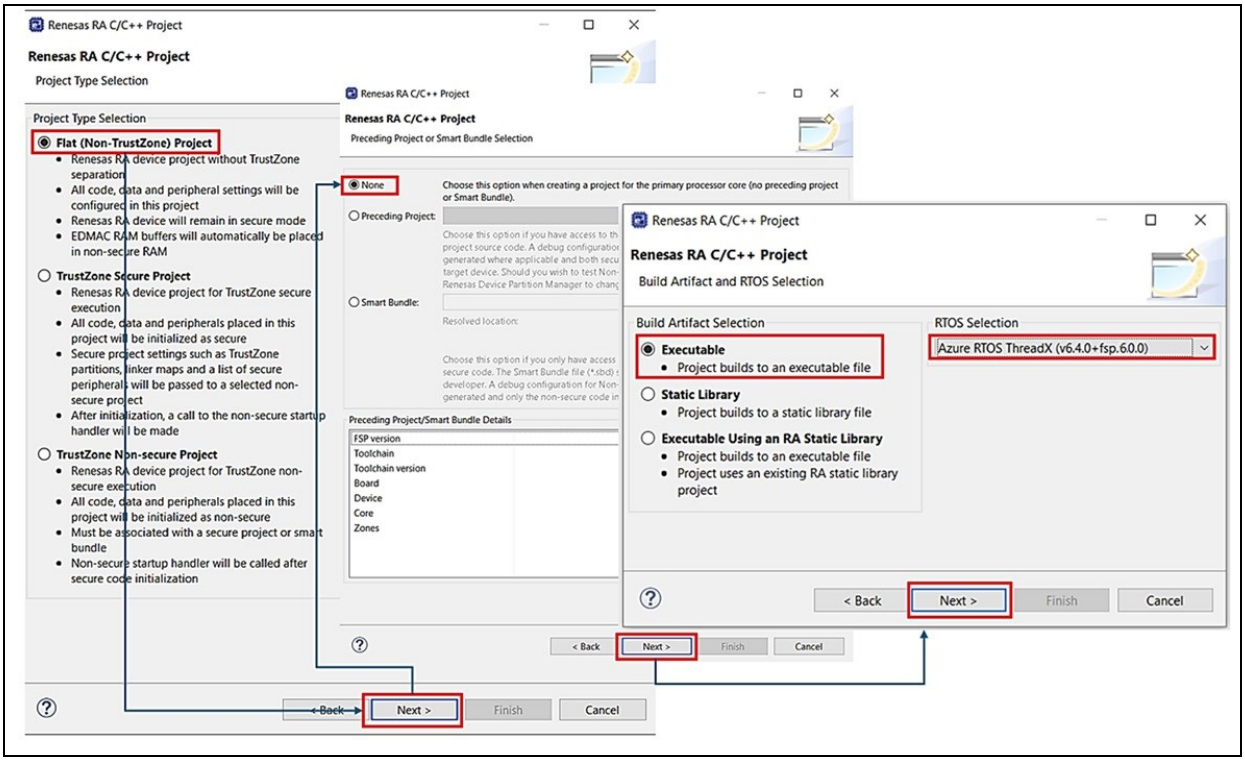


Figure 6. Select Azure RTOS ThreadX

4. Use **Azure RTOS ThreadX - Minimal** template.

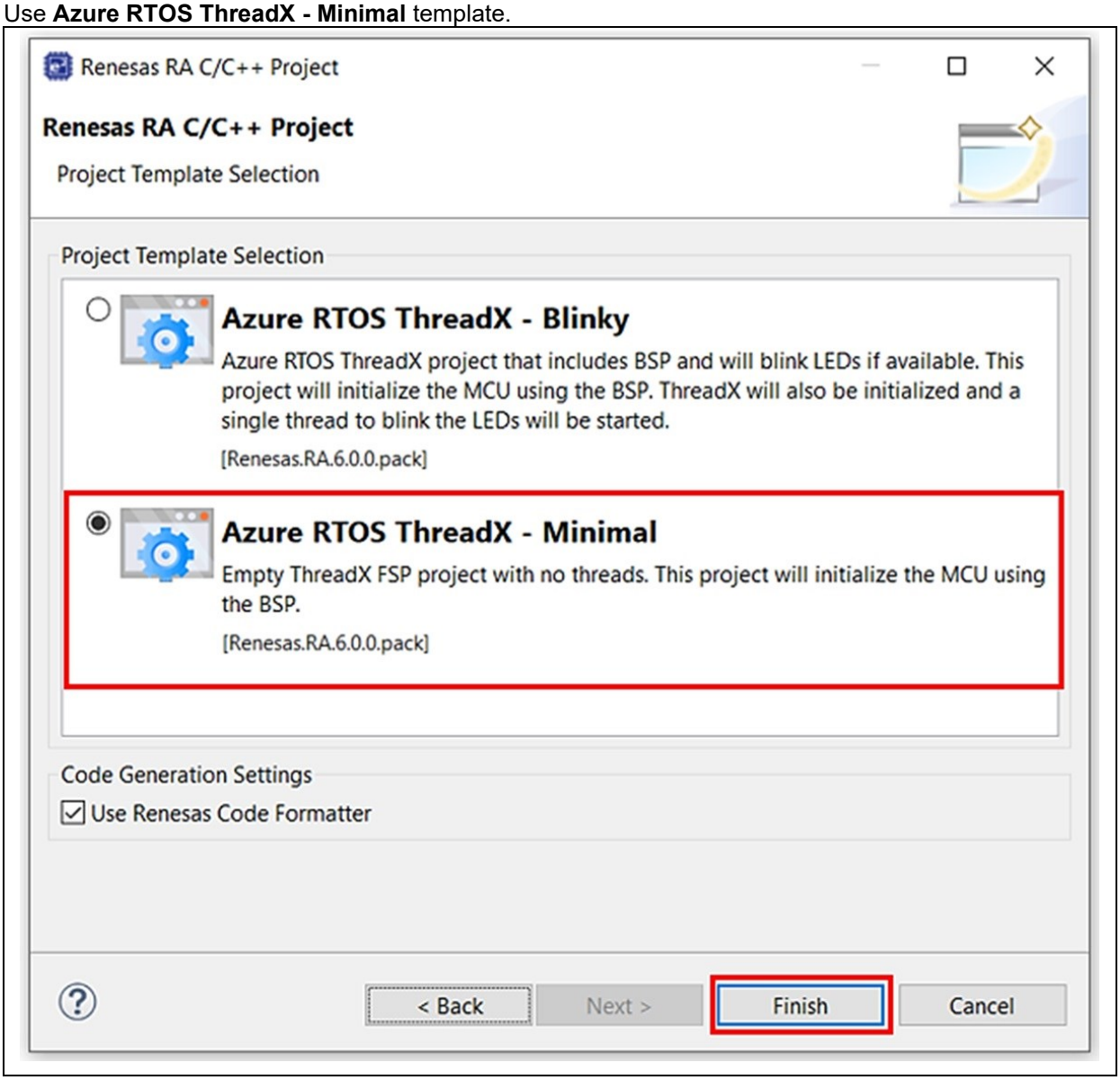


Figure 7. Selecting Azure RTOS ThreadX – Minimal Template

5. Open the project configuration and go to the **BSP** tab. Settings **SDRAM** and **Heap size (bytes)** to **0x2000**. And **Stack size (bytes)** to **0x1000**

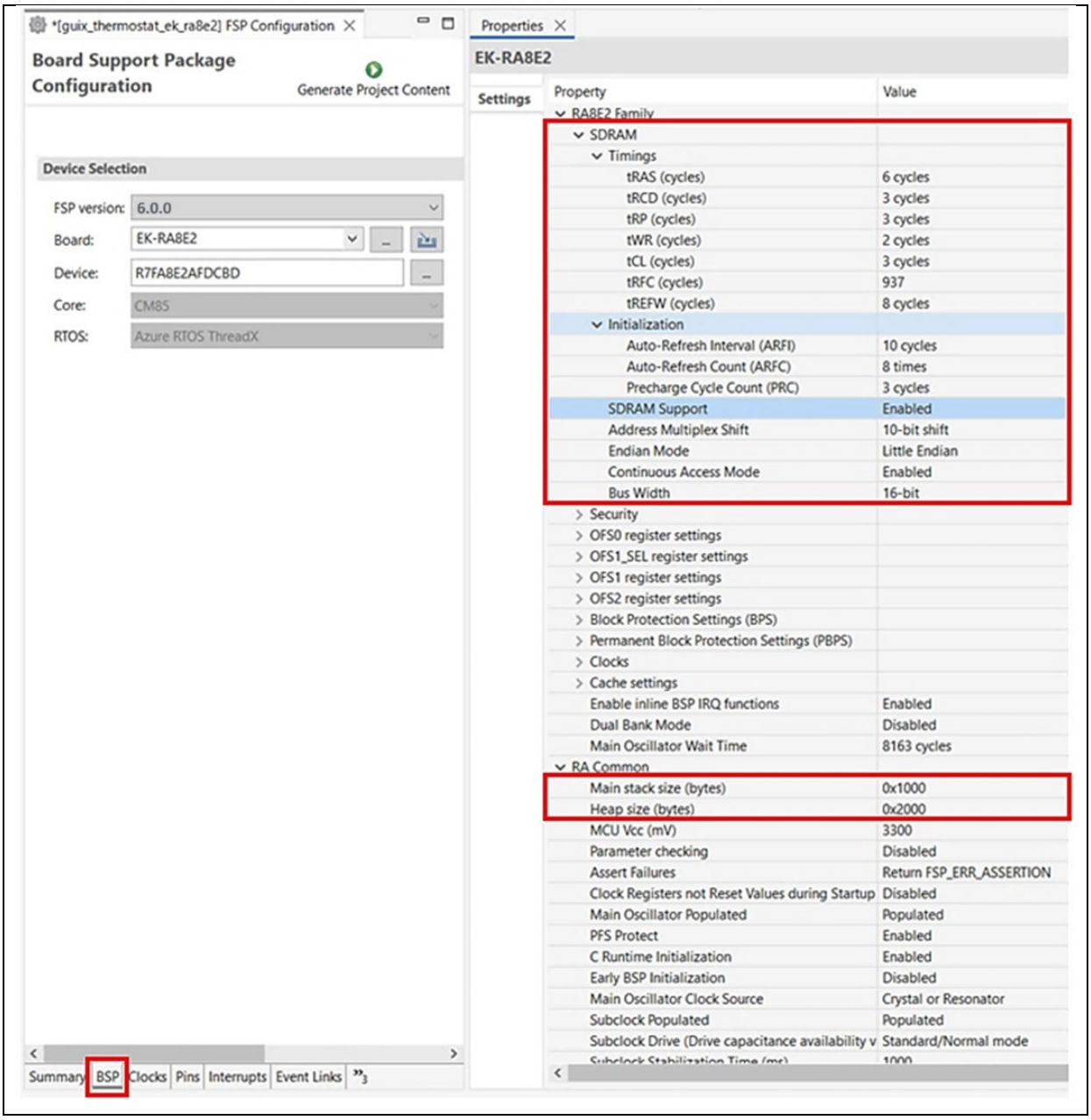


Figure 8. Setting SDRAM, Main stack and Heap size Properties

GUI Storage in MCU memory or SDRAM:

This Project is using Azure RTOS GUIX with multiple images output settings: 1 image = 800 x 480 pixels x 24bpp = approximately 1.5 MB which is larger than 1 MB of code flash memory. If it is stored in the MCU code flash memory, it does not fit. Because MCU code flash memory is limited to 1 MB. The alternative is to use the SDRAM. The SDRAM memory is available built in on board (SDRAM is 512Mbit, which is 64Mbyte). Following section 2 and Figure 13. Step # 2.2.10 Settings properties for **Graphics LCD**: “Input > Framebuffer > Section for framebuffer allocation > . sdram (it points to SDRAM and use SDRAM). After step # 2.2.32: Generate Project Content and Build Project”. You will see the SDRAM initialize generated in hal_entry.c file (“R_BSP_SdramInit(true)”).

In addition, the application stores image data in the external OSPI memory instead of the code flash, prior to being rendered by the DRW module and having its values written to SDRAM, due to limitations of the code flash.

```
#if BSP_CFG_SDRAM_ENABLED
    /* Setup SDRAM and initialize it. Must configure pins first. */
    R_BSP_SdramInit(true);
#endif
```

Figure 9: SDRAM Initialization

6. Click tab "**Clocks**" and set "Clocks" for the LCDCLK and OCTACLK

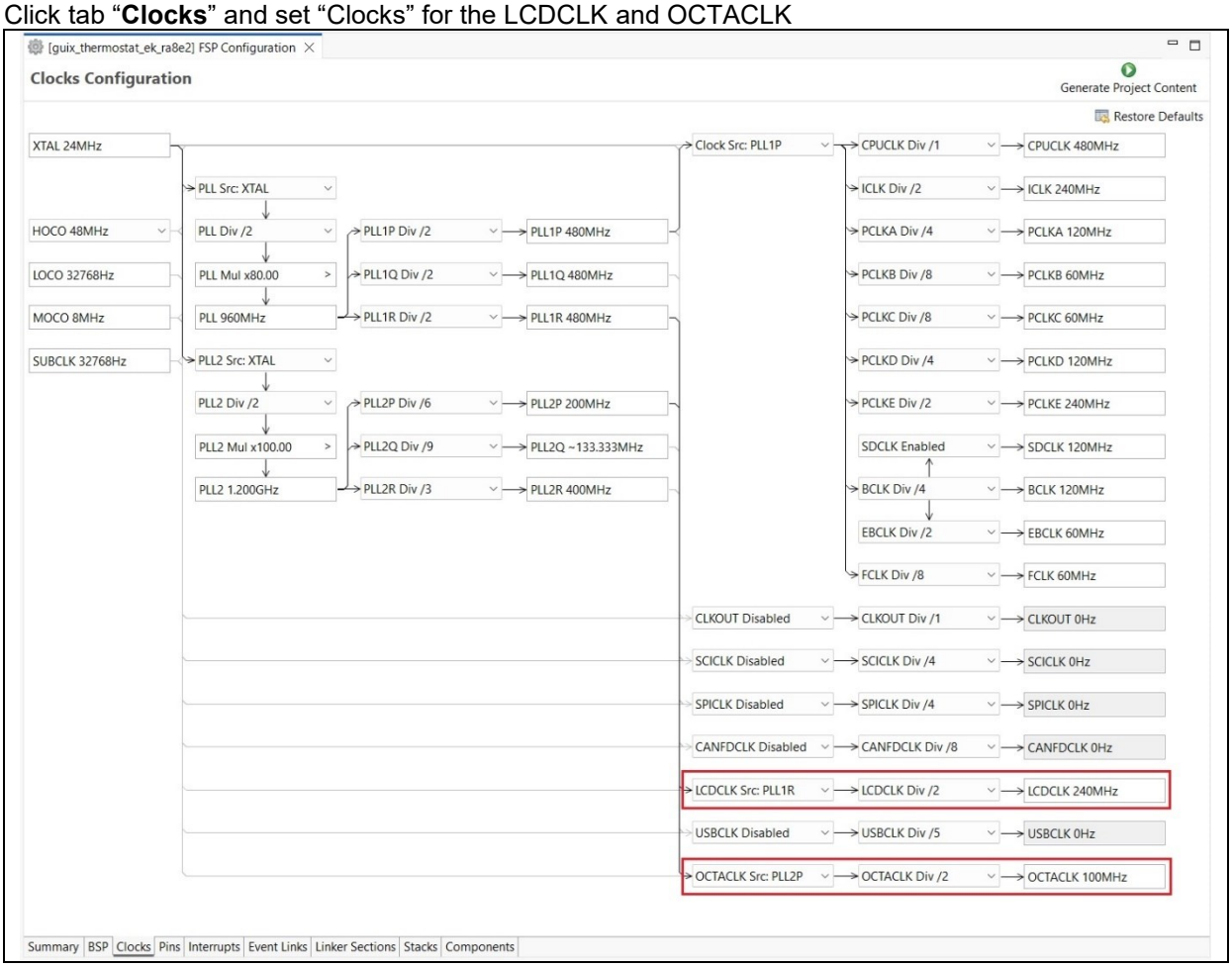


Figure 10. LCDCLK and OCTACLK Clocks Setting

7. Add "**New Thread**" and name **System Thread** and Setting Properties.

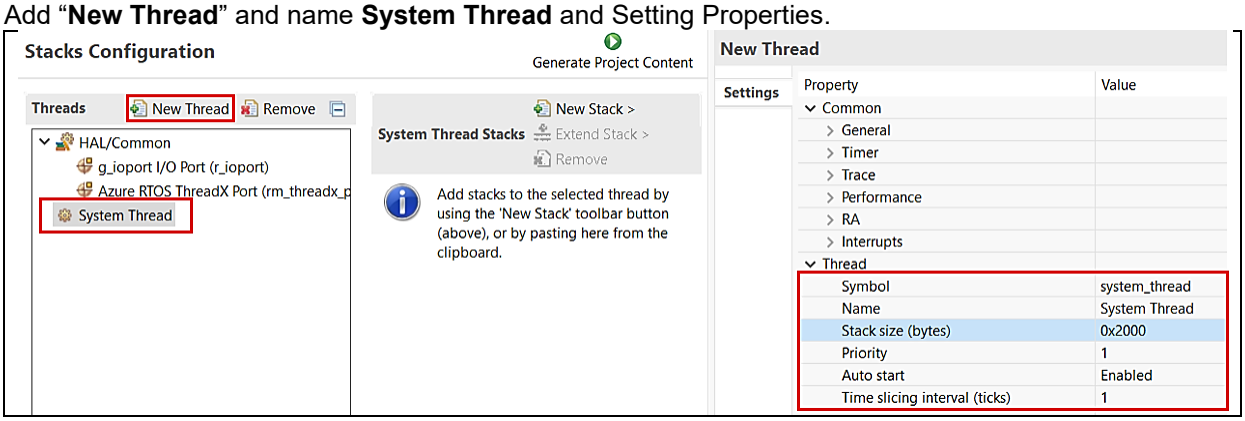


Figure 11. Add New System Thread and Setting Properties

8. Add “**New Stack**” **Azure RTOS GUIX** to System Thread.

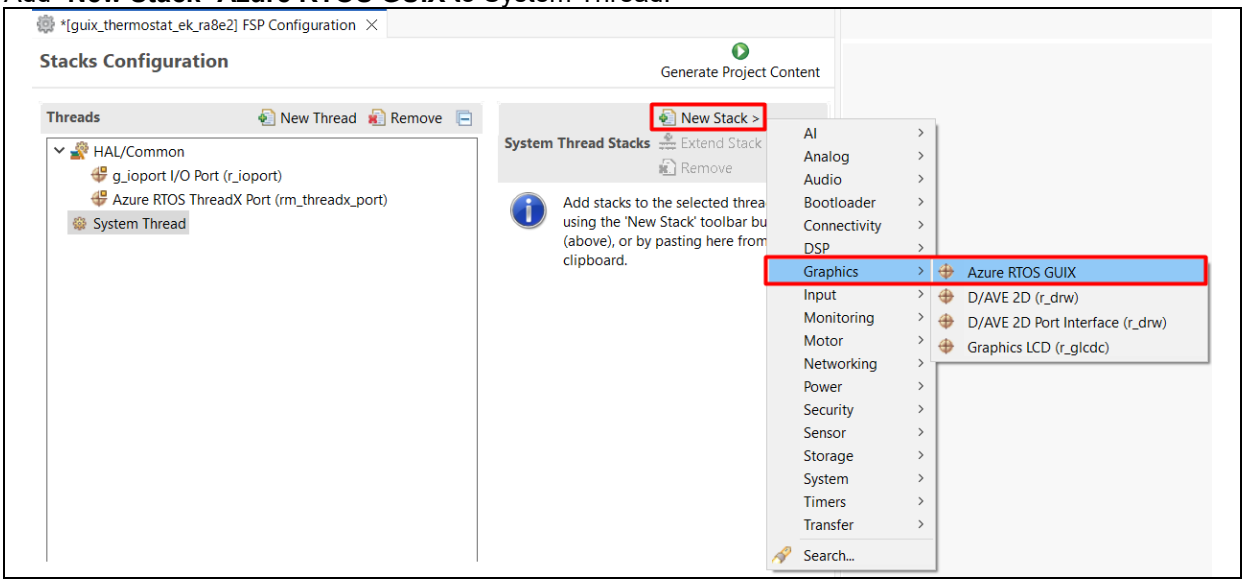


Figure 12. Add Azure RTOS GUIX

9. Setting **pins configuration** for **GLCD**.

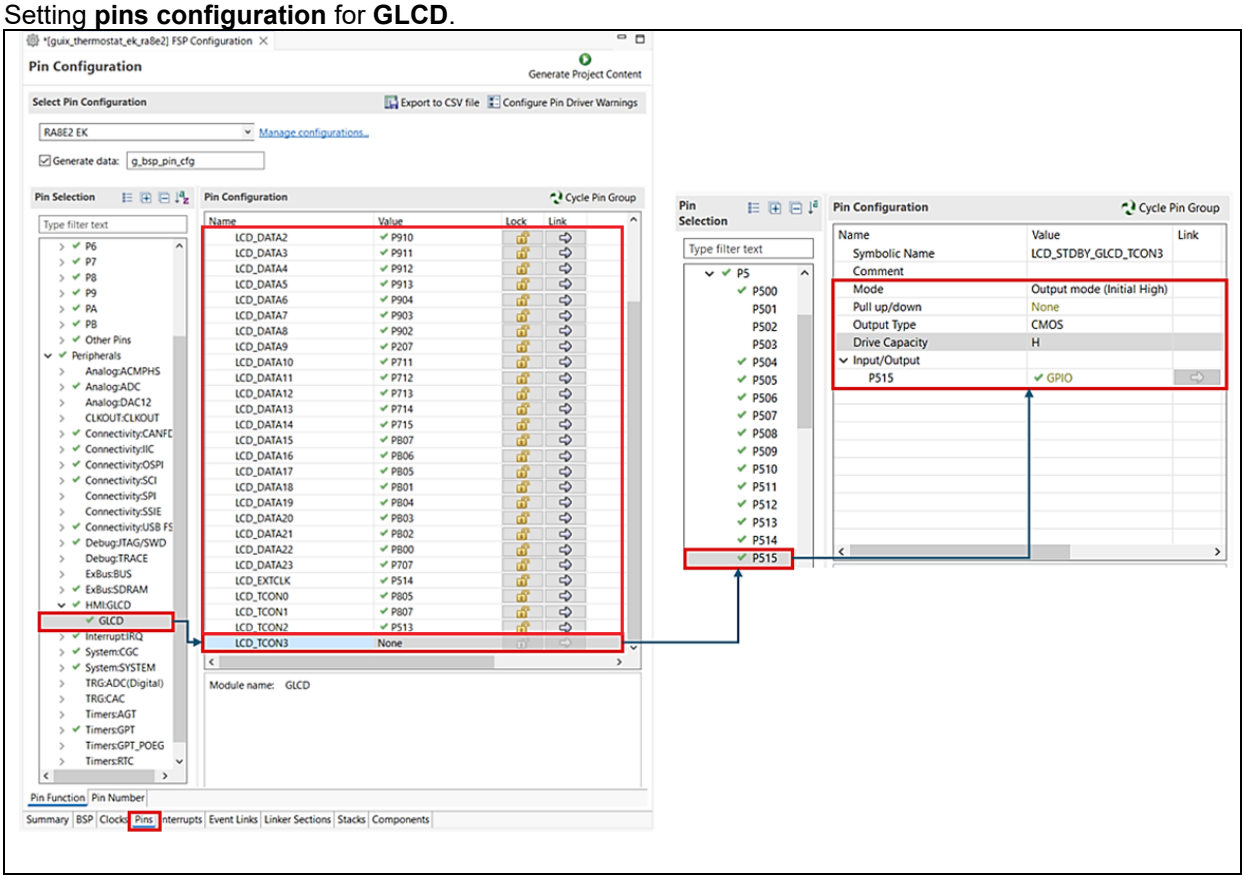


Figure 13. Setting pins configuration for GLCD

10. Settings properties for **Graphics LCD.**

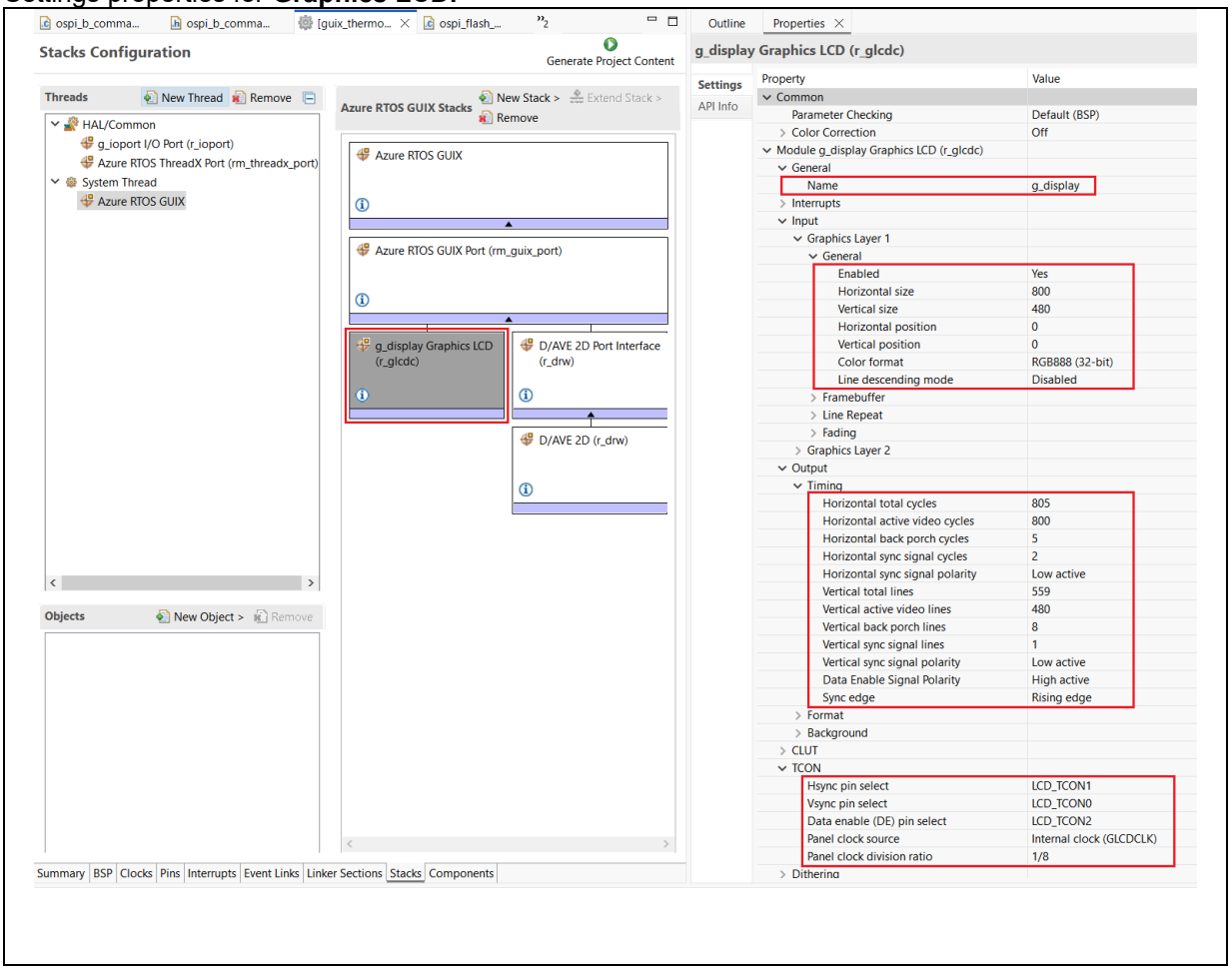


Figure 14. Setting Properties for GLCDC DISPLAY

11. **Pin Configuration**, change P404's mode to **Output mode (Initial high)** to enable LCD panel backlight.

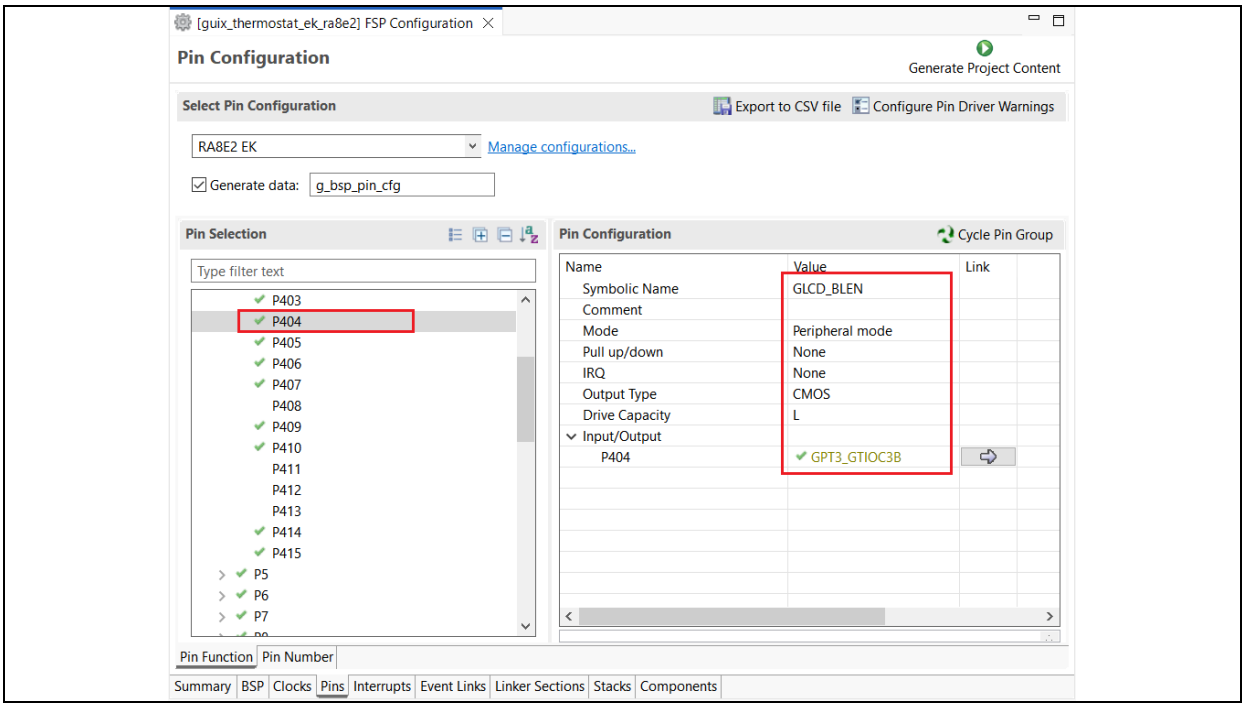


Figure 15. Settings PWM Timer properties

12. Add "**New Thread**" name **Ospi Flash Thread** and Setting Properties.

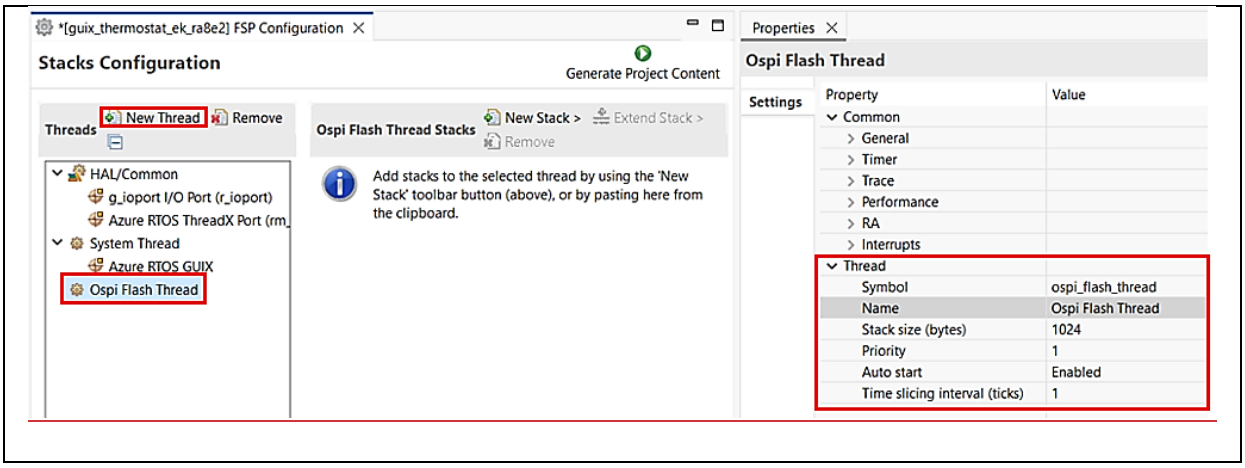


Figure 16. Add "New Thread" name Ospi Flash Thread and Setting Properties

13. Add OSPI driver (r_ospi_b).

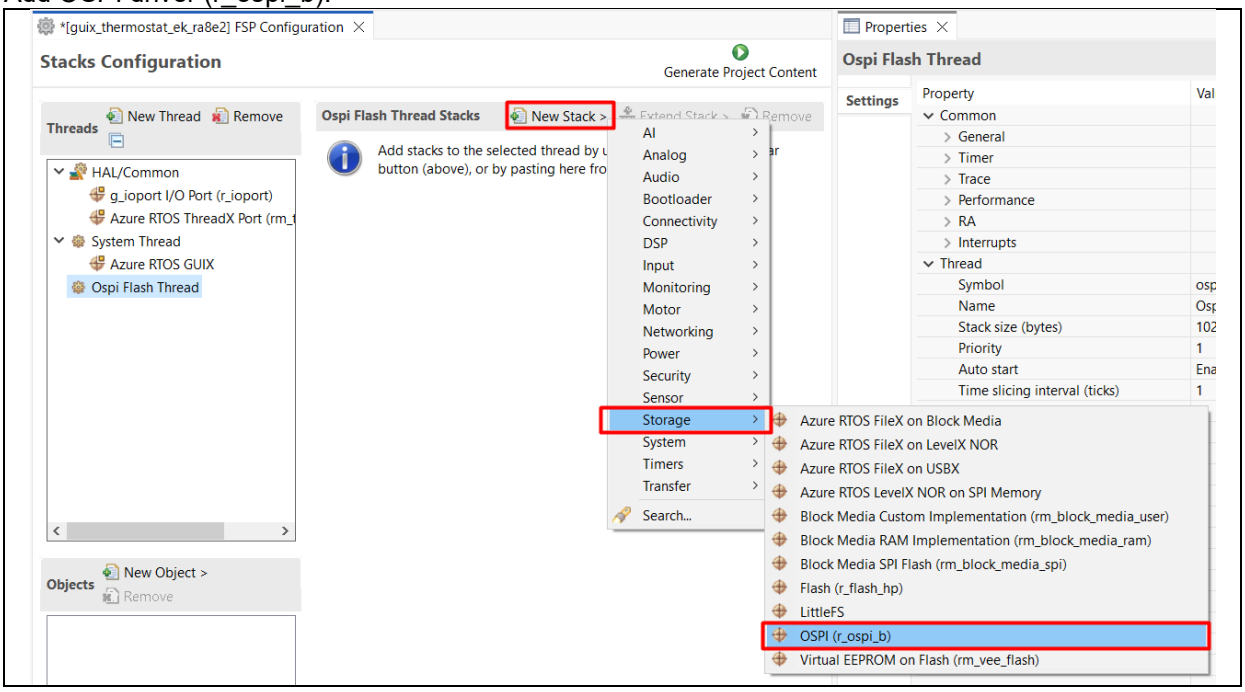


Figure 17. Add OSPI Flash to Ospi Flash Thread

14. Add DMAC Driver "r_dmac" for OSPI Flash transmission

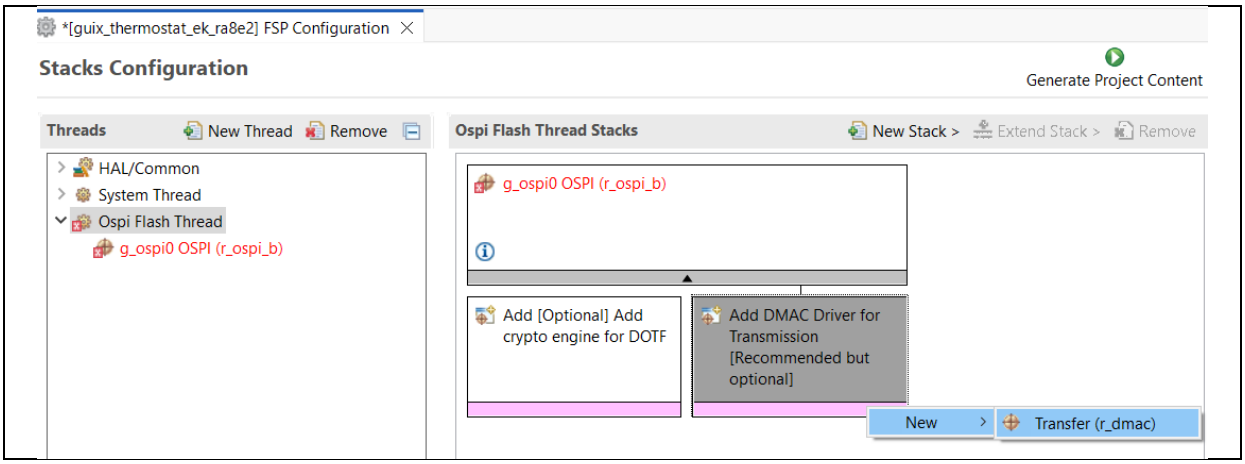


Figure 18. Add **DMAC Driver** for **transmission**

15. Setting OSPI Flash properties.

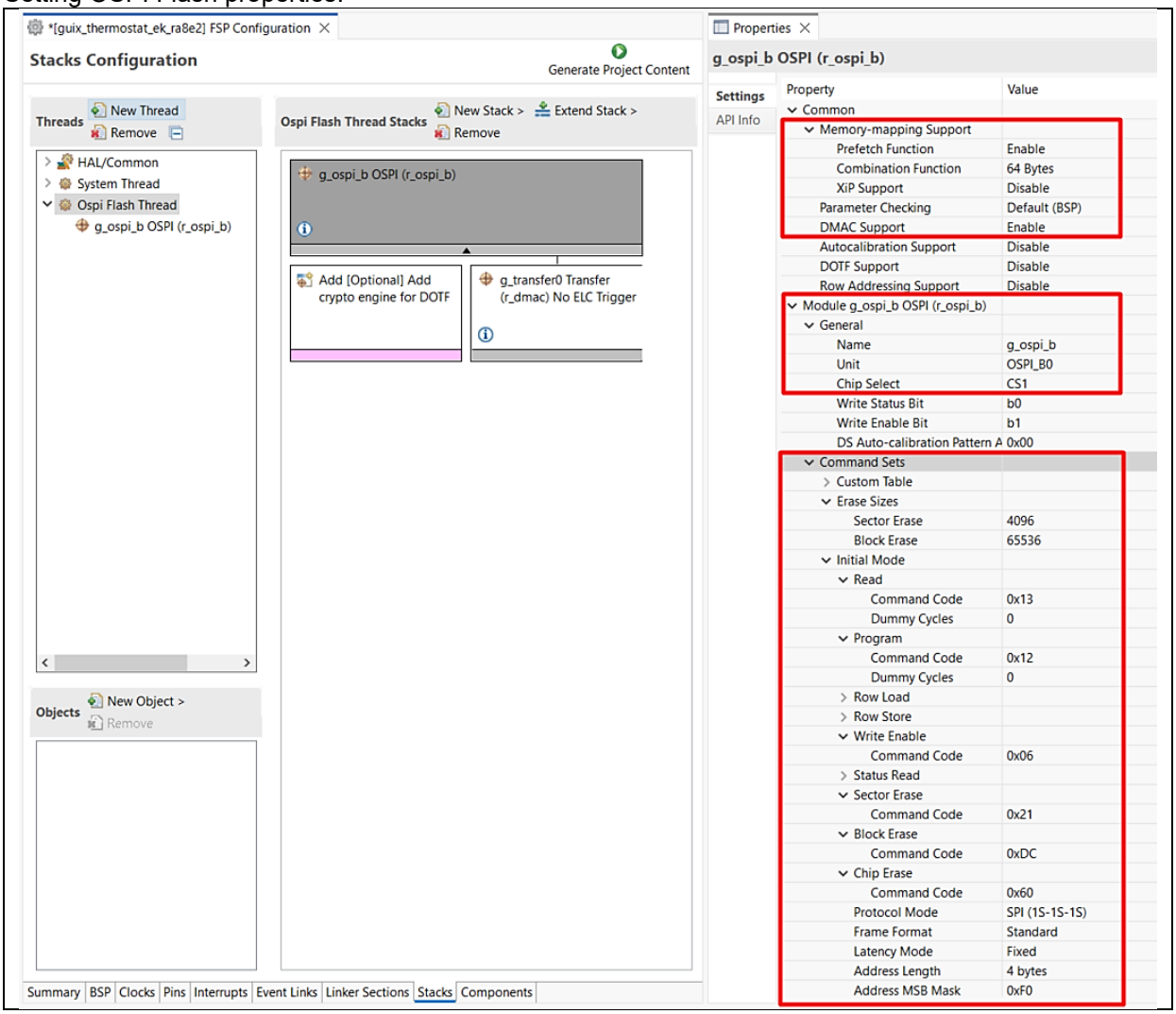


Figure 19. Set OSPI properties

16. Set OSPI pins configuration.

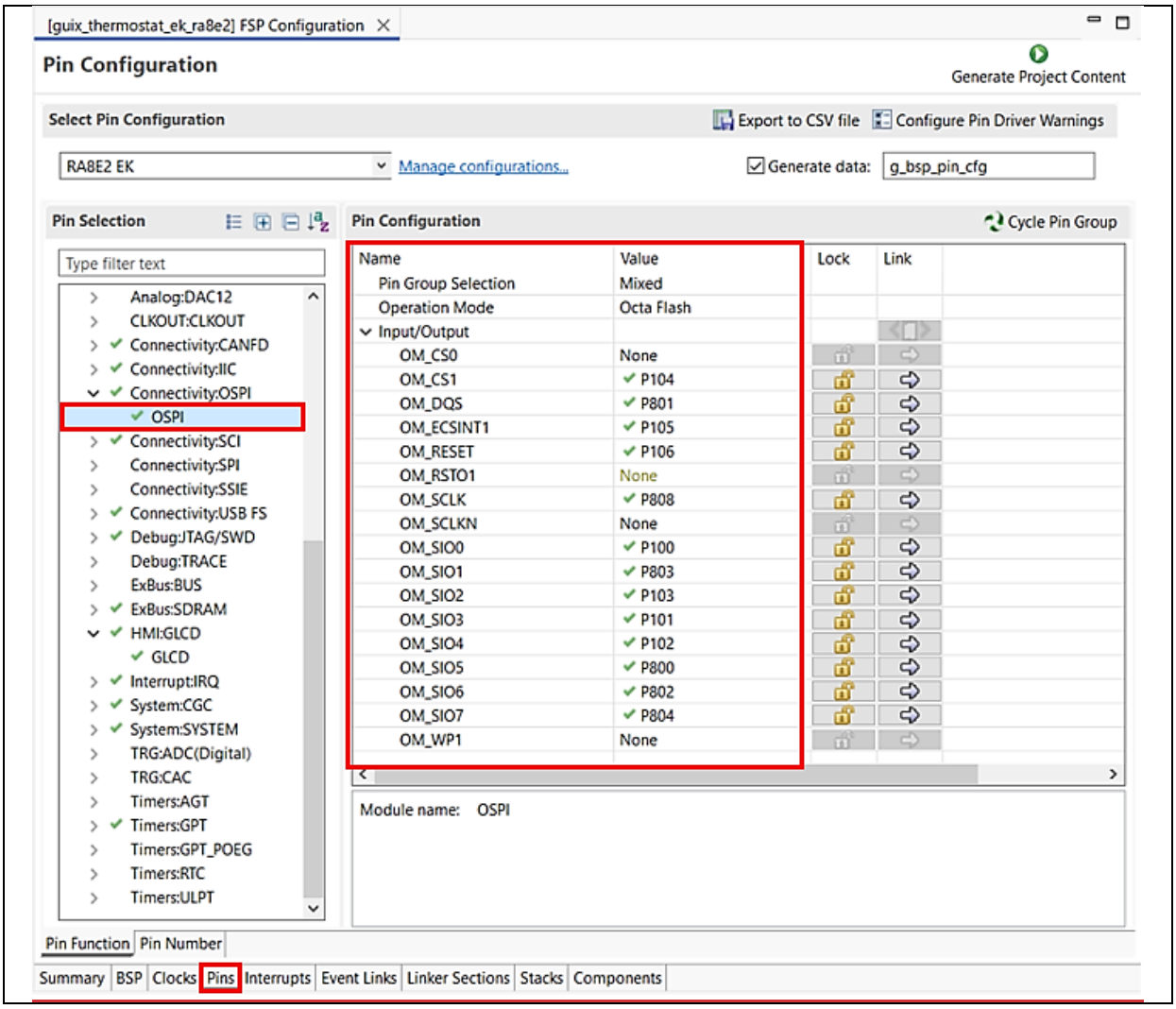


Figure 20. Set OSPI pins configuration

17. OSPI memory address map area used for CS1.

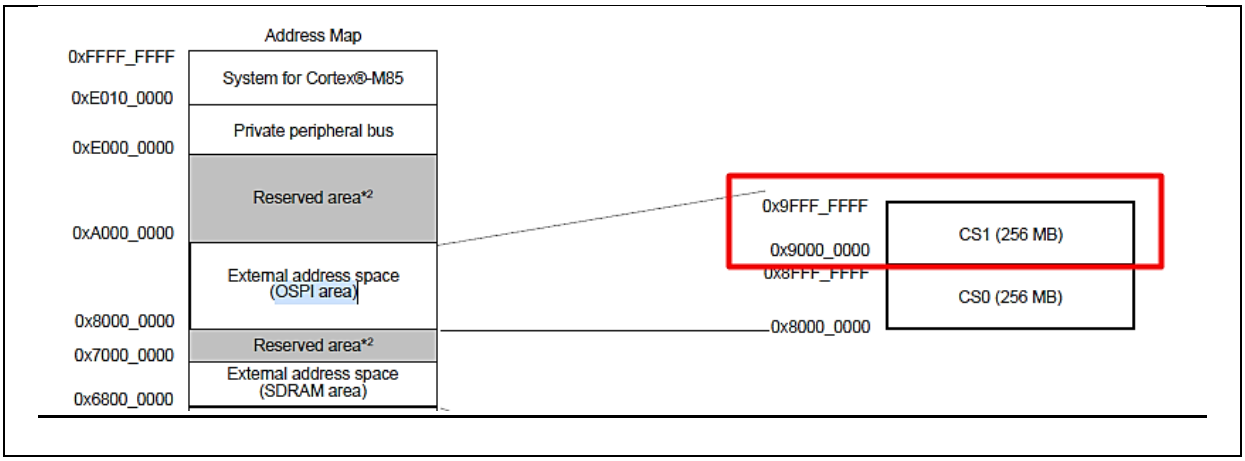


Figure 21. OSPI memory address map

18. In RA Configurator, click Generate Project Content to generate project content. Make sure project is active, click to build the project. It may take a long period of time to finish building an Azure RTOS/GUIX project on your PC.

19. Copy Azure RTOS GUIX Studio project to e² studio project (Themostat_GUIX_EK_RA8E2) by copying **"guix_studio"** folder in the application note (**AN**) folder (FSP_GUIX_Thermostat) and pasting it in the Themostat_GUIX_EK_RA8E2 project.

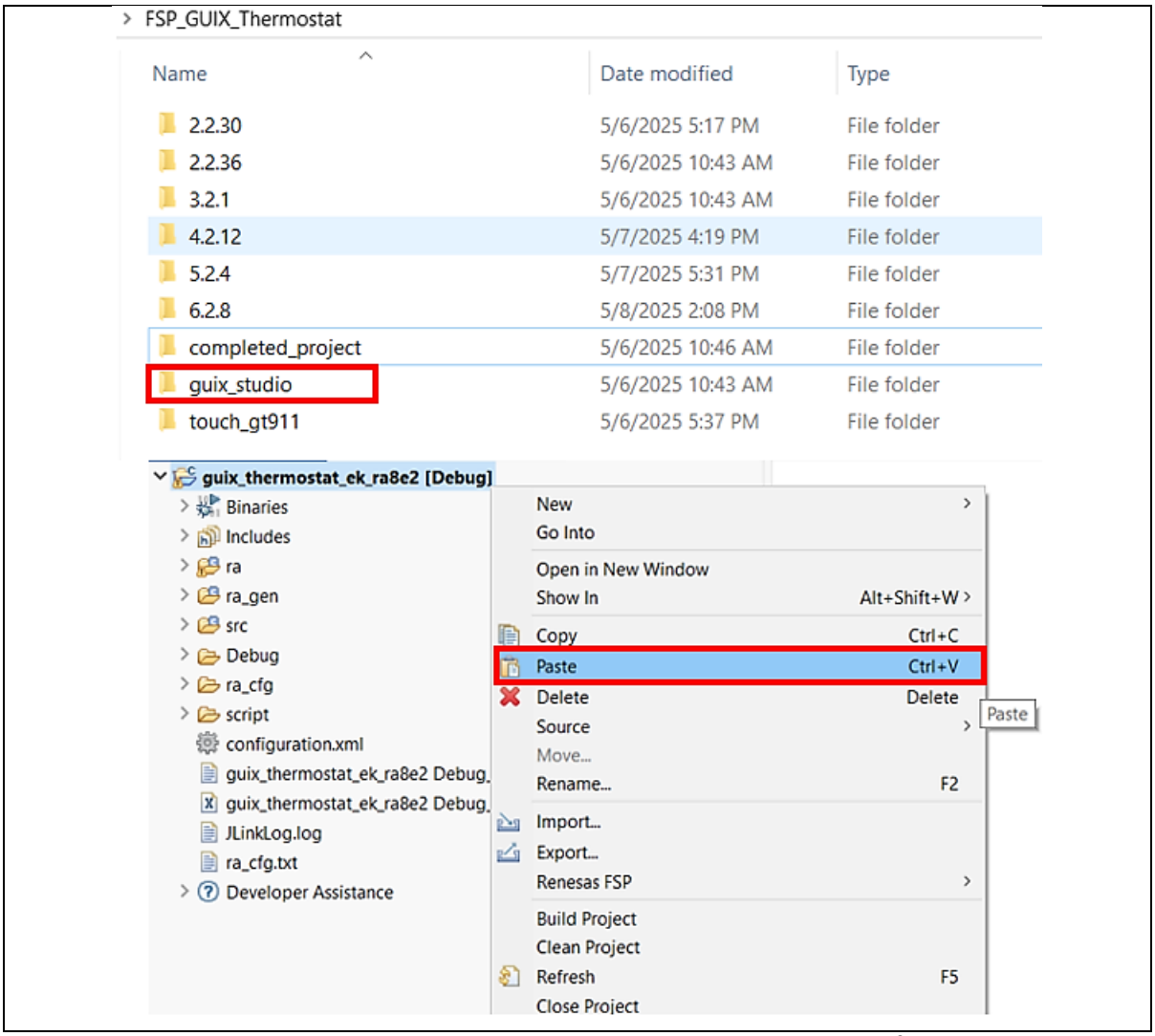


Figure 22. Copying the Azure RTOS GUIX Studio Project to e² studio

20. GUIX Studio project is now in guix_thermostat_ek_ra8e2 project. In e² studio, right-click the "**guix_studio**" folder and exclude it from the build since it contains the Azure GUIX Studio project, which will not be built by FSP.

    The guix_studio folder holds the GUIX thermostat project, the source of the graphics, and the fonts. The graphics and the fonts will be used by the GUIX thermostat project when it is compiled by the GUIX Studio application. The content in this folder will be used in a later step to generate the GUIX .c and .h source files using the GUIX Studio Application. This folder will not be compiled by e² studio IDE.

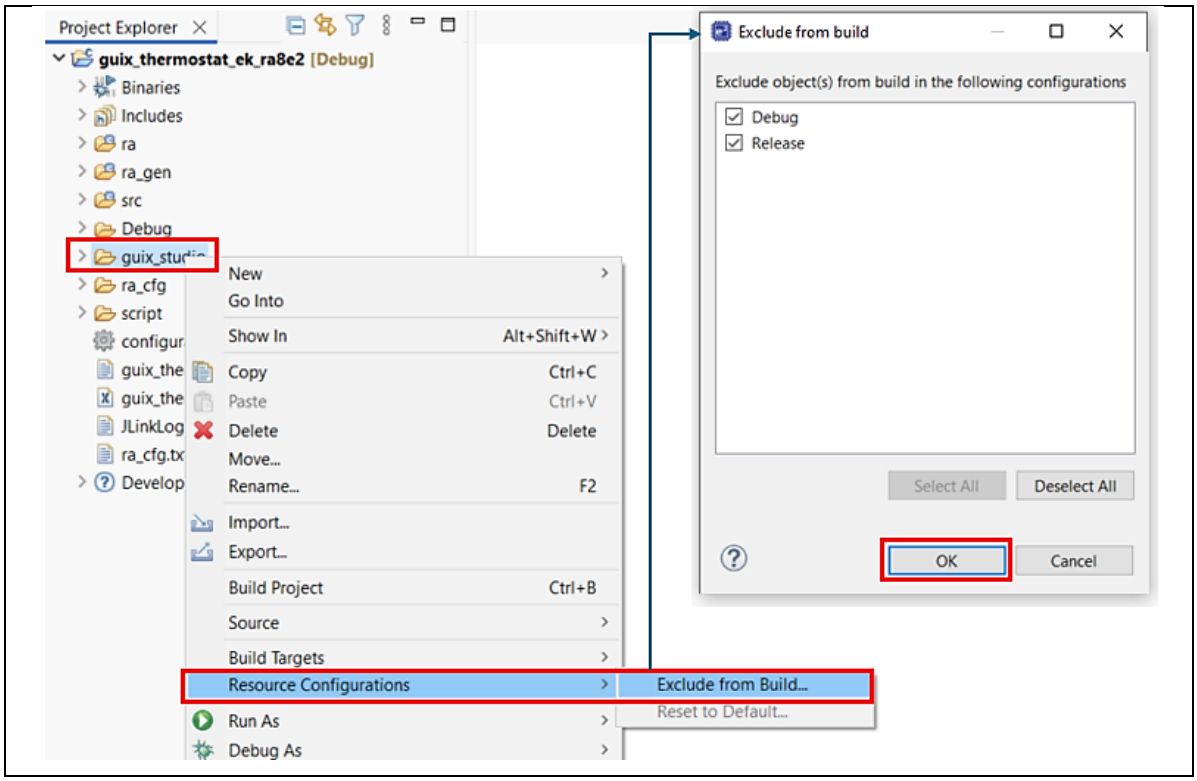


Figure 23. guix_studio folder containing

21. Get to guix_thermostat_ek_ra8e2 project folder by right clicking the e² studio project and select "**System Explorer**" as shown below.

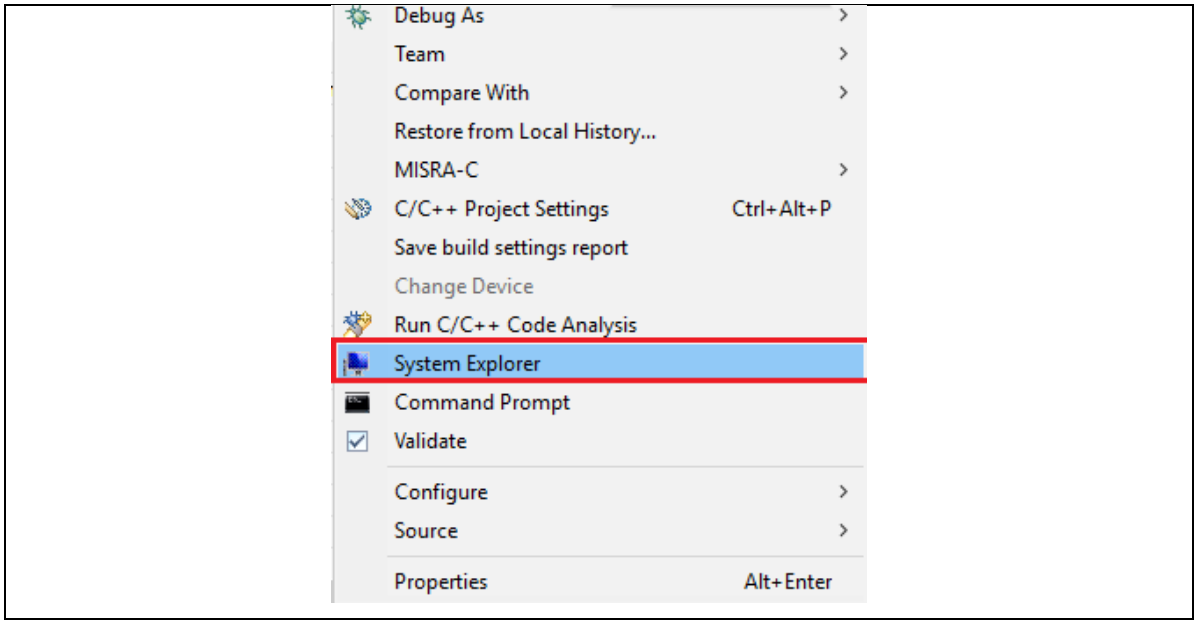


Figure 24. Selecting System Explorer

22. Open **thermostat.gpx** project file in "**guix_studio > GNU**" sub-folder in your guix_thermostat_ek_ra8e2 folder. If you have several GUIX Studio versions in your system, make sure you choose the right one, which is **v6.4.0.0**.

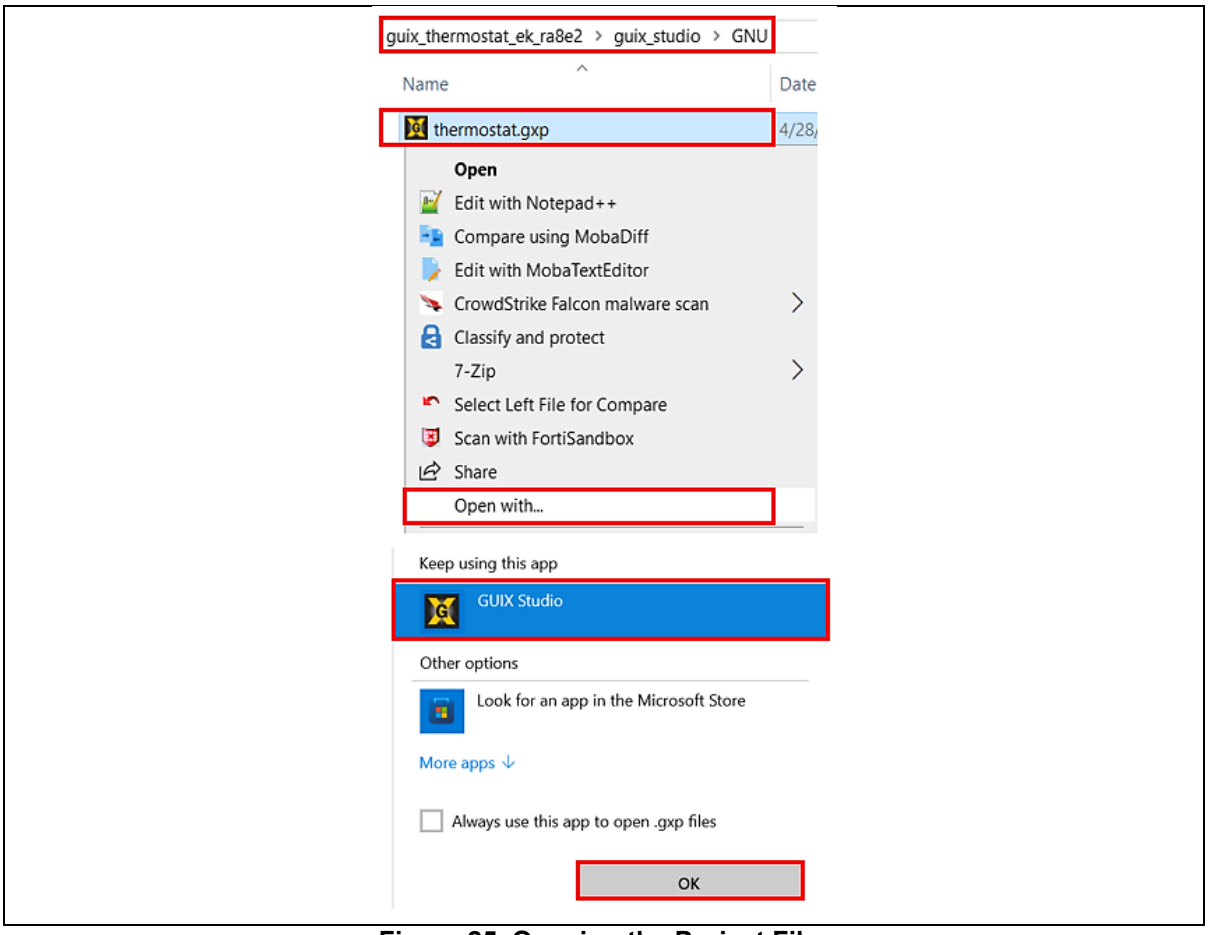


Figure 25. Opening the Project File

23. This GUIX Studio project has a complete design of this Thermostat application. The next several steps describe the process to generate resources, application code and integrate them with an e^2 studio project.

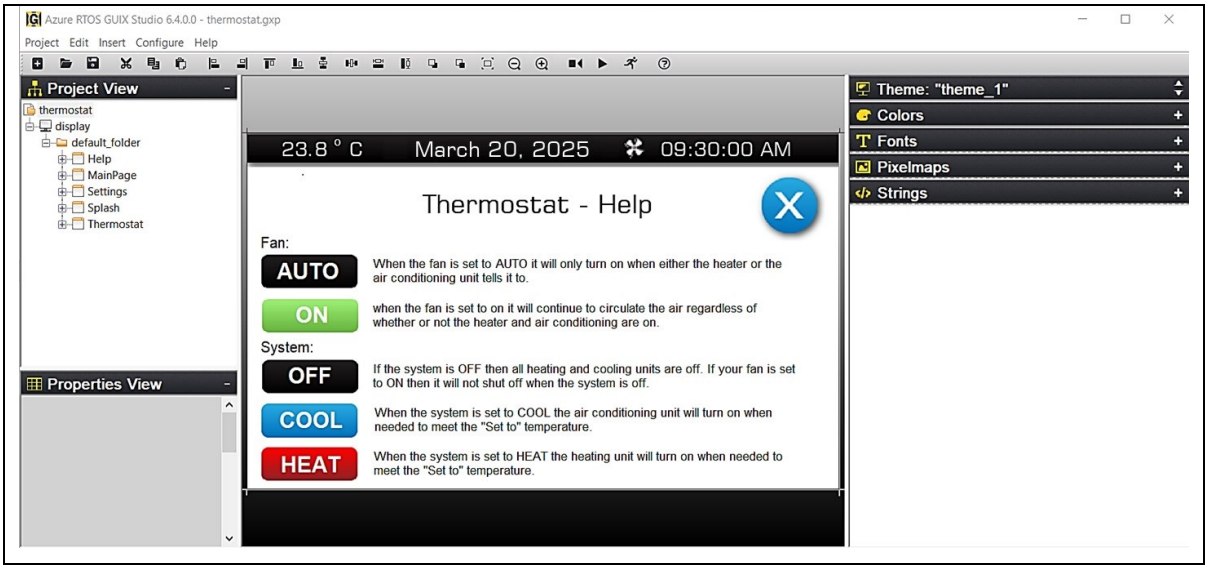


Figure 26. GUIX Studio Thermostat Application View

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

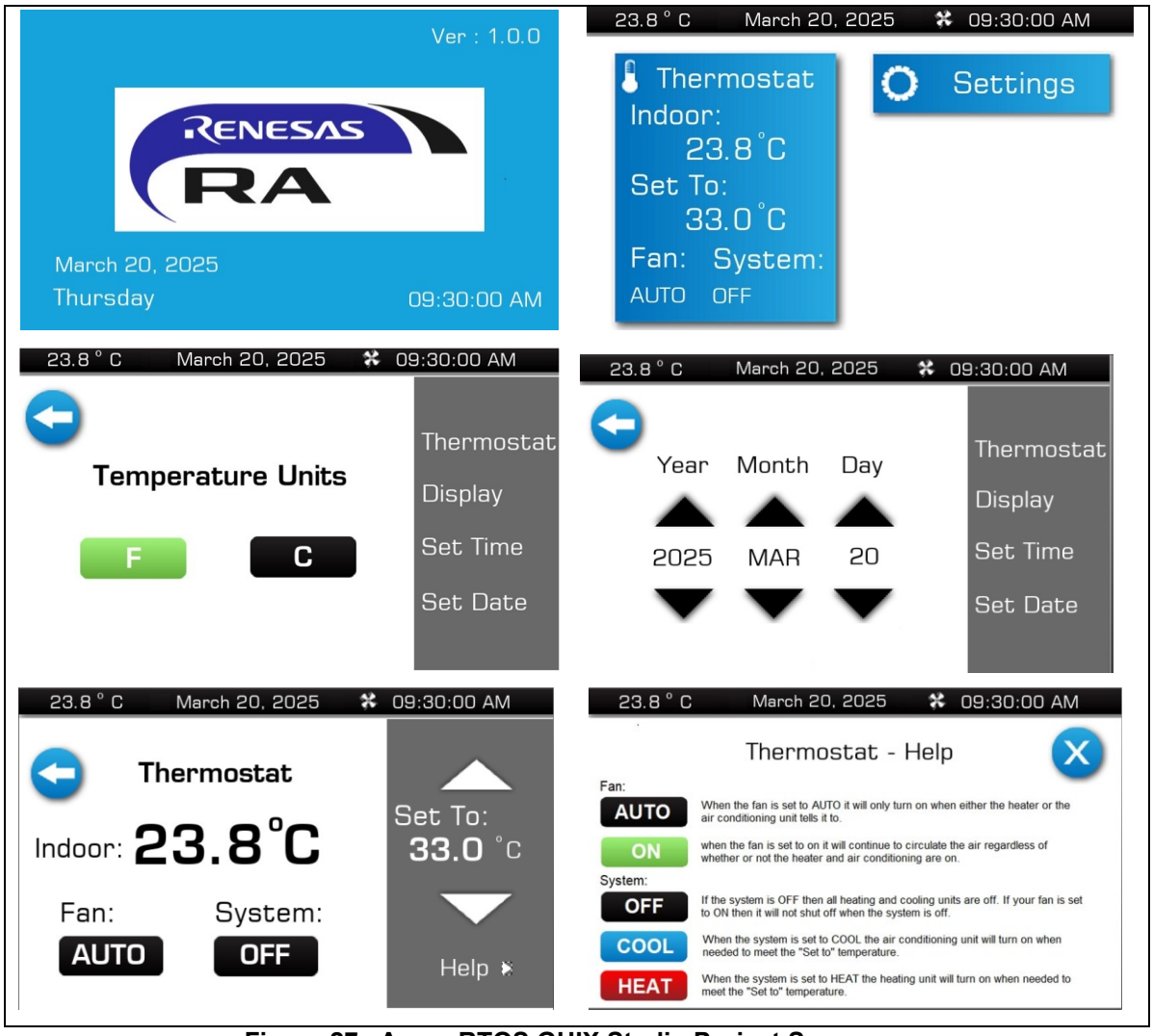


Figure 27. Azure RTOS GUIX Studio Project Screens

25. Click "**Configure->Project/Display**" and confirm the following settings.

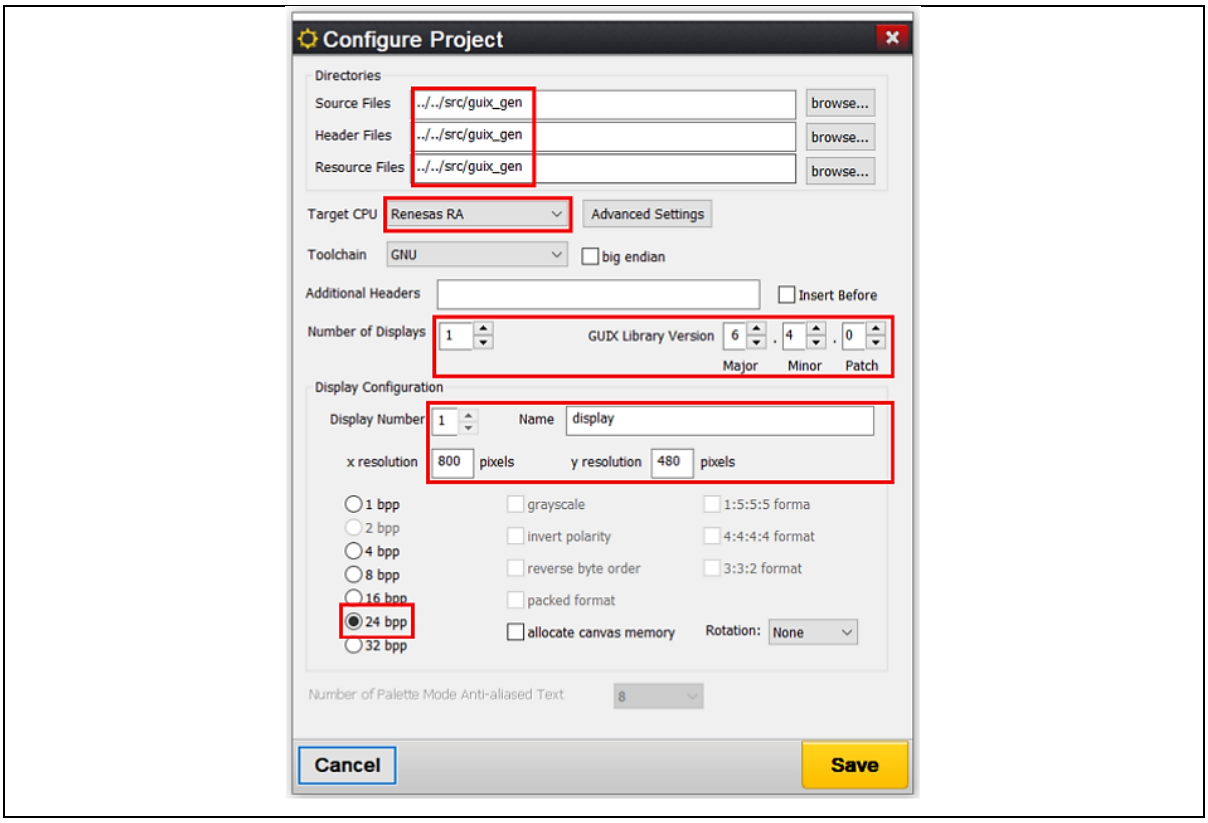


Figure 28. Configure Project Settings

26. Go back e^2 studio project (guix_thermostat_ek_ra8e2), right click "**src**", then select "**New->Folder**" and create a folder named "**guix_gen**".

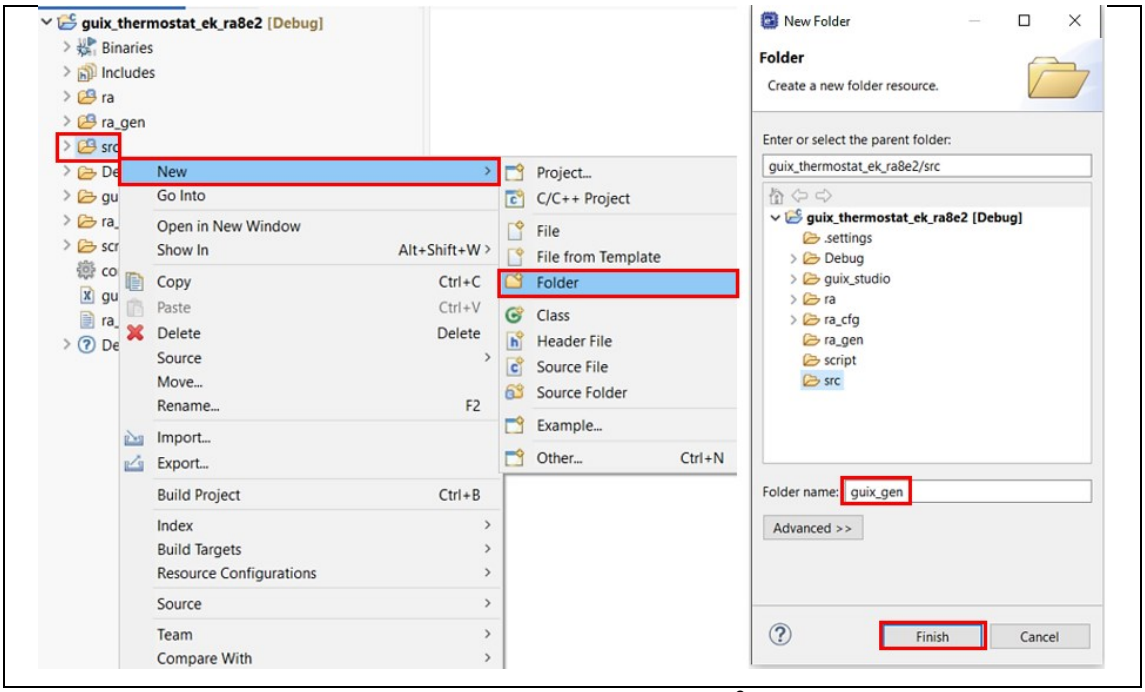


Figure 29. Creating a "guix_gen" in e^2 studio Project

27. Confirm "**guix_gen**" is created before moving to next step.

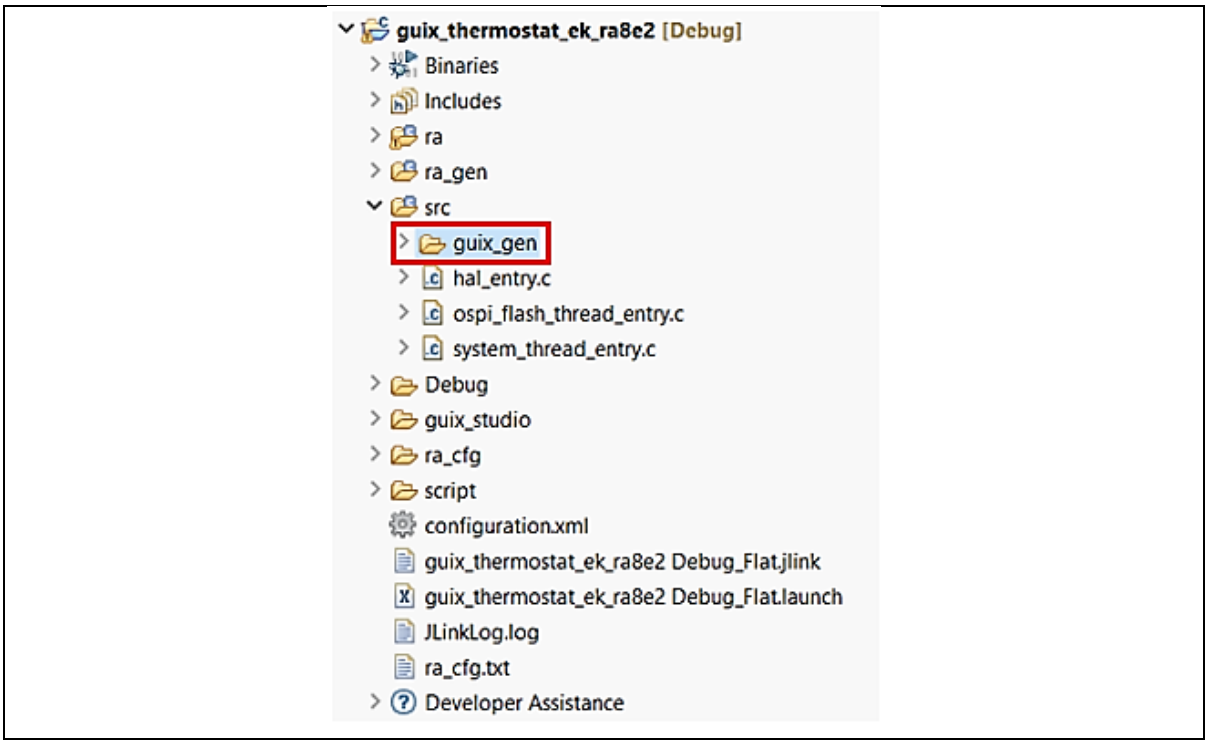


Figure 30. Confirming Creation of "guix_gen"

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

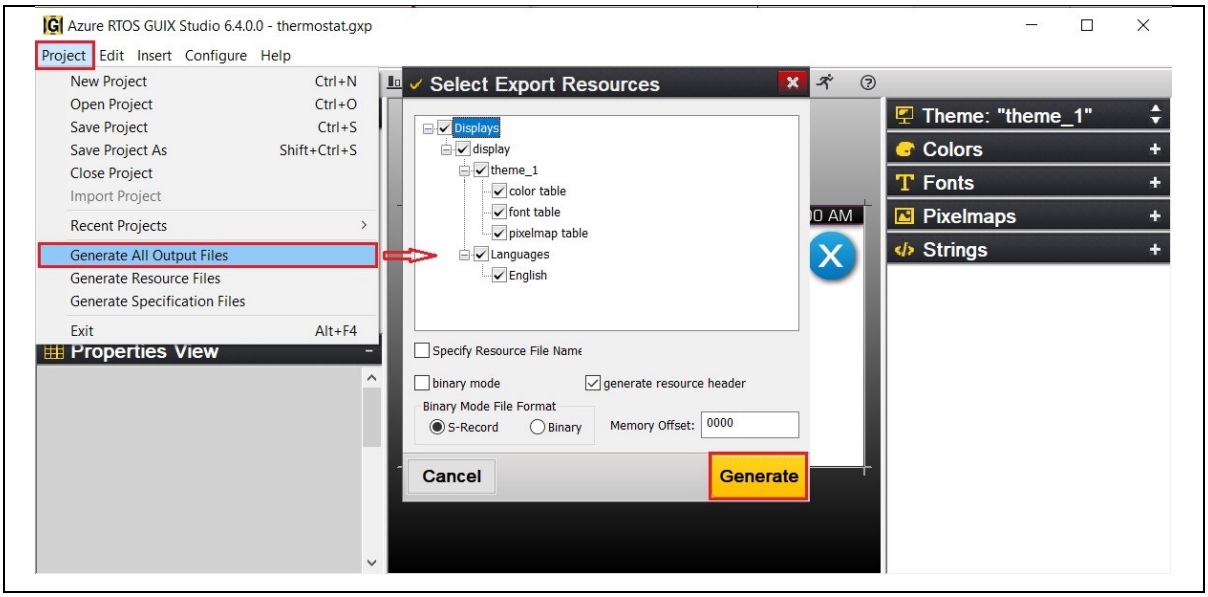


Figure 31. Clicking Generate All Output Files

29. Click **Generate** to generate all output files. If you succeeded, you would see the notification below.

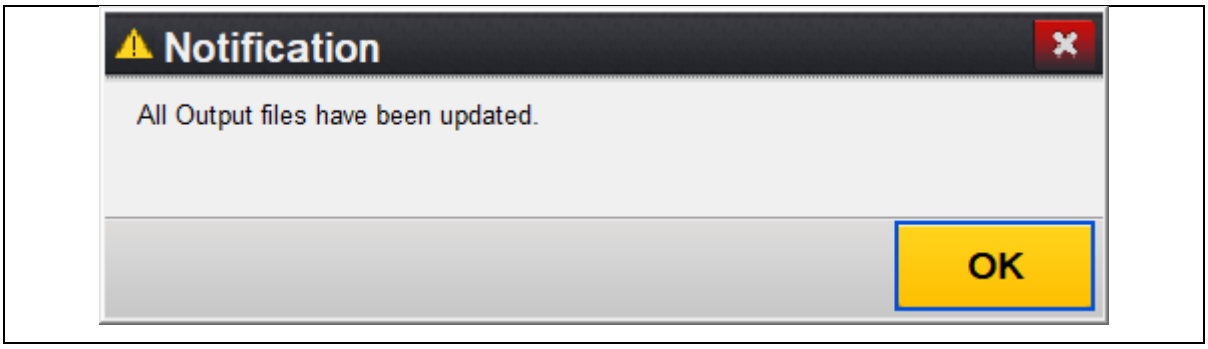


Figure 32. All Output Files Updated Notification

30. All output files are now in "**guix_gen**" folder.

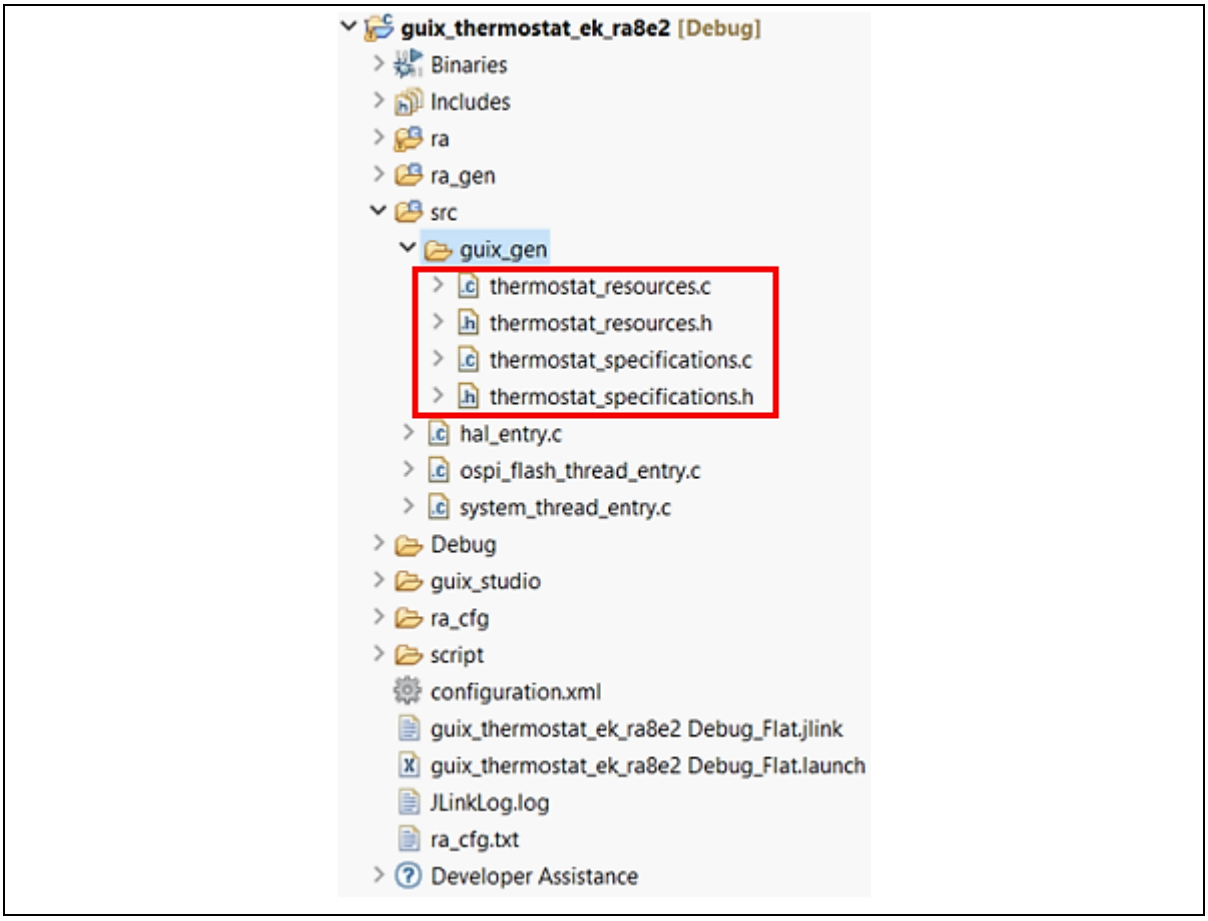


Figure 33. Location of Output Files

31. Placing GUIX Resources in External Flash Memory

Due to the limited internal flash memory on EK-RA8E2 (1Mb Code Flash), the GUI resources for this application must be stored in external OSPI flash memory to prevent overflow issue when building application project.

You need to add the **"User Mappings"** in **configuration.xml > Linker Sections** to allocate GUI resource images in external flash. This is because most image resources generated by GUIX are declared as constants (GX_CONST).

Choose **New User Mapping > OSPI_CS1 > OSPI0_CS1 Constant Data** to add New User Mapping to OSPI CS1.

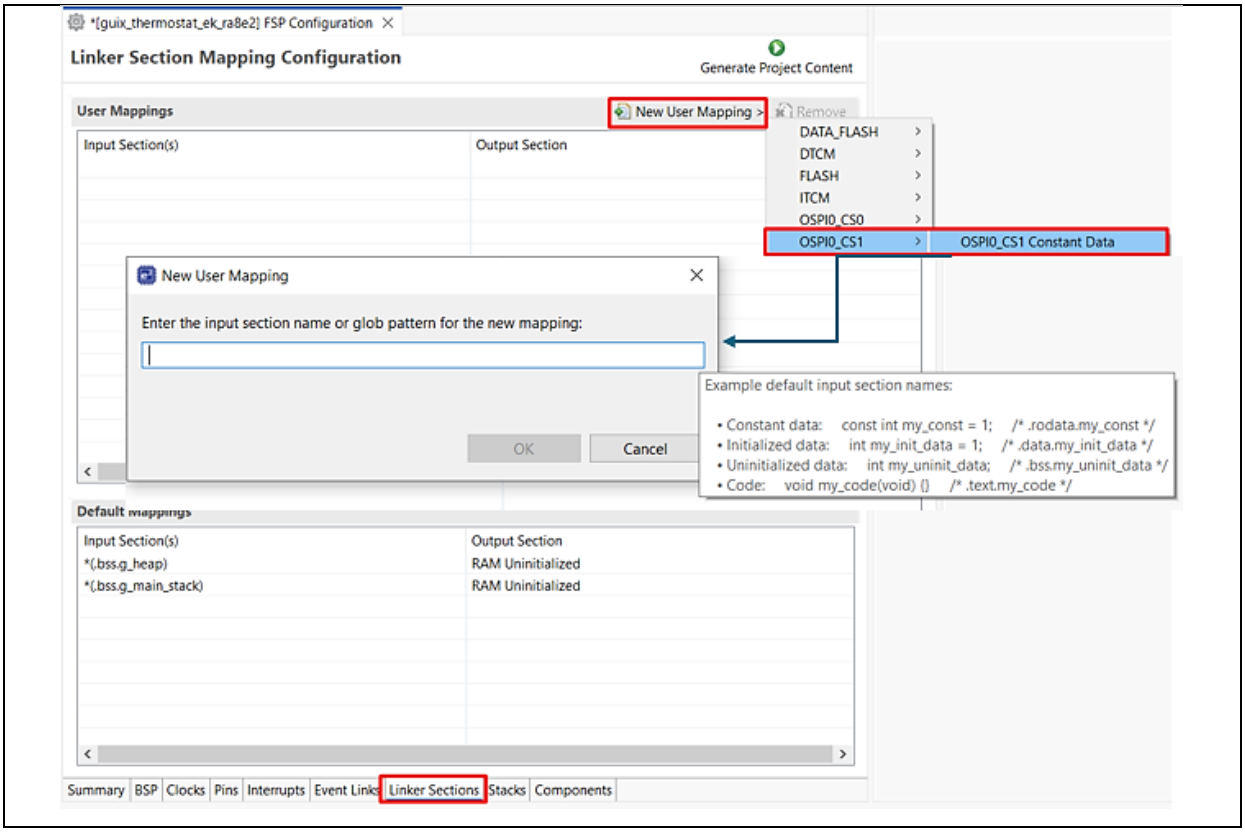


Figure 34. Add New User Mapping to allocate GUI resource to OSPI CS1

In this step, user needs to check the name of the data that needs to be allocated to the OSPI CS1 section. For example, in GUI resource file (src/guix_gen/thermostat_resources.c), there is a const array named **DISPLAY_THEME_1_RADIO_ON_pixelmap_data**. User can allocate this array to OSPI CS1 by entering the name **.rodata.DISPLAY_THEME_1*** into the input box and click **OK,** as shown below**:**

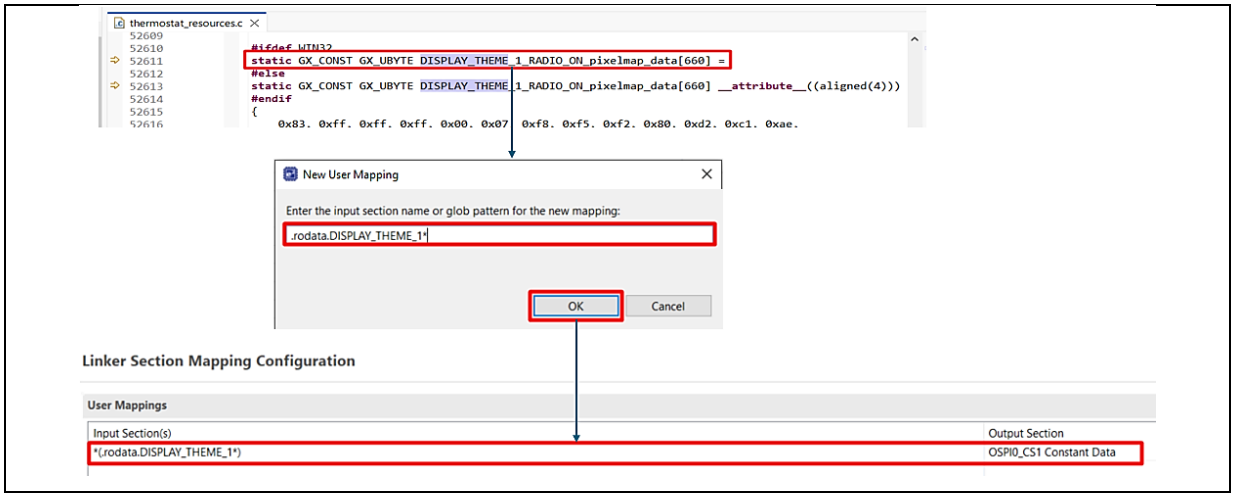


Figure 35. Add User Mapping for GUI resource - rodata

Other arrays in the resource follow a similar pattern. Please continue adding the input sections below in order to allocate the GUI resource to OSPI CS1 and run the application.

Table 1 Input sections

(.rodata.DISPLAY_THEME_1)
(.rodata.FONT_THEME_1)
(.rodata.Help)
(.rodata.MainPage)
(.rodata.Settings)
(.rodata.Splash)
(.rodata.THEME_1)
(.rodata.Thermostat)
(.rodata.display)
*(.rodata.display_theme_1_color_table)
*(.rodata.thermostat_widget_table)

Figure 36. Table Of Input Sections

After adding the input section, the linker section will appear as shown below.

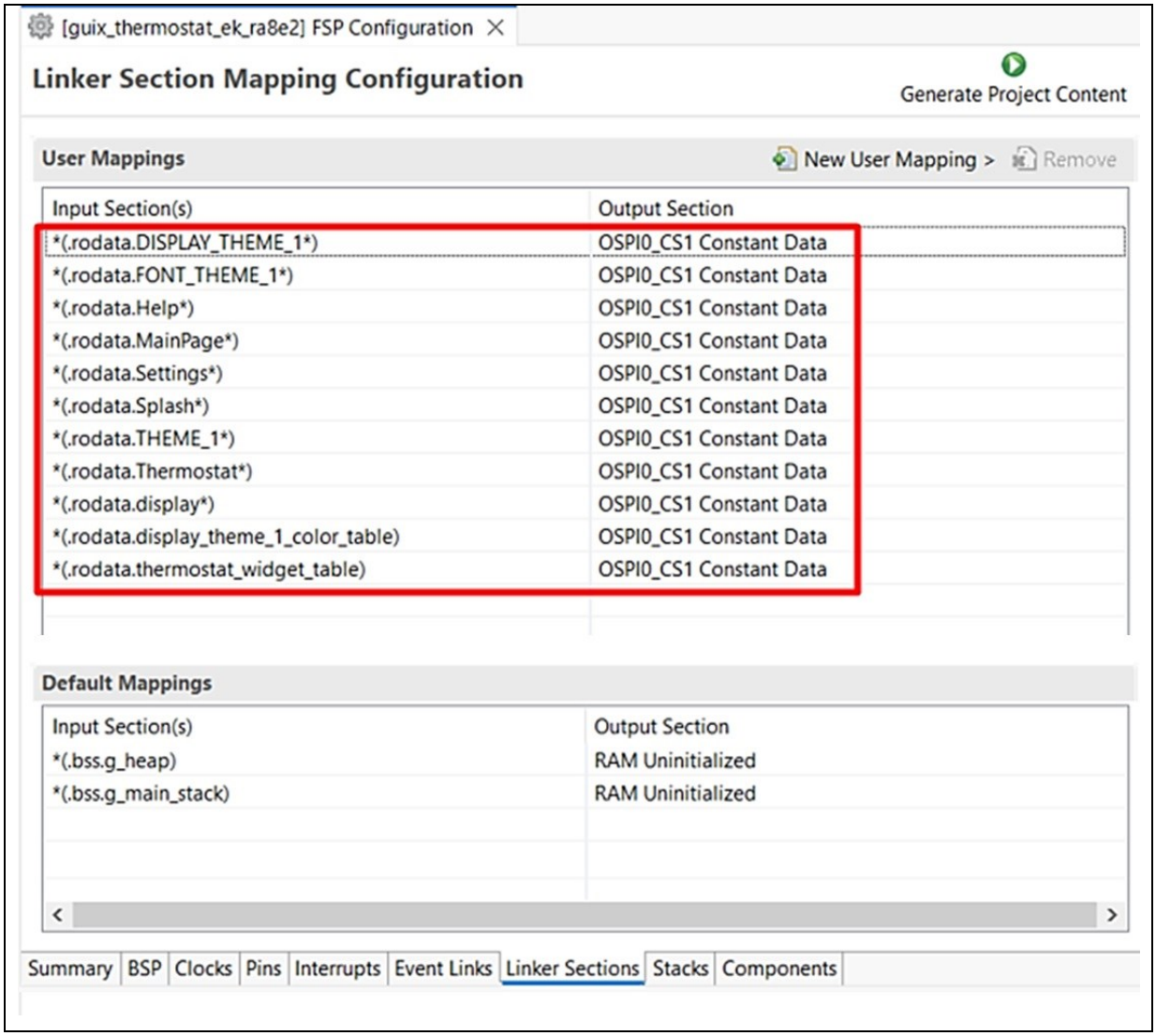


Figure 37. Linker Sections after adding all the input section

Note: The above list is intended only for the generated GUIX source included with this application. Users can check and modify it according to their corresponding source code to ensure that GUI resources are allocated to the OSPI CS1 region.

32. Enable Data cache (D-cache) to optimize performance.

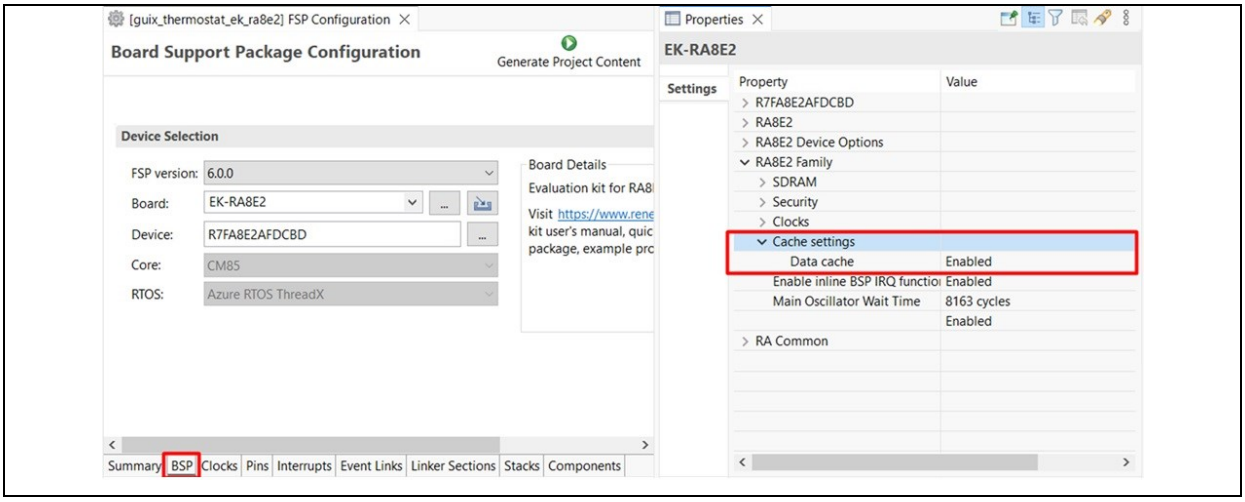


Figure 38. Enable Data cache

33. In **the Azure RTOS GUIX Studio Project**, click **"Splash"** and pick up "**Widget Name**" and "**Event Function**" definitions. These definitions are used to create a screen and handle it in **the e²studio/FSP project**. The other windows have similar definitions.

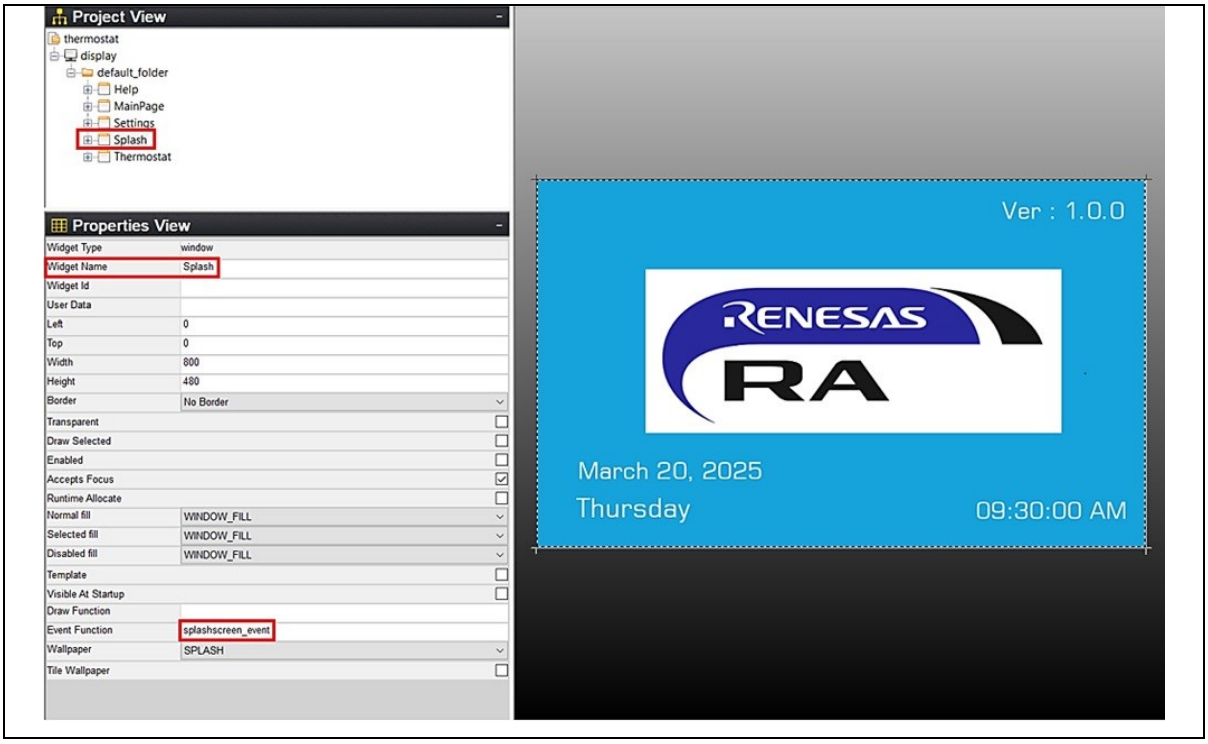
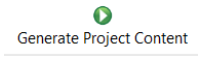


Figure 39. Definitions in the Azure RTOS GUIX Studio Project

34. Copy and replace the files in **"src"** folder in e² studio project with the files in **"2.2.34"** folder in the AN folder:

- hmi_event_handler.c
- system_thread_entry.c
- ospi_b_commands.c
- ospi_b_commands.h
- ospi_flash_thread_entry.c

In RA Configurator (configuration.xml), click 

to generate project content.?

Build guix_thermostat_ek_ra8e2 project you will see several warnings, but we will address them in later steps.

35. **Code highlight:** The application will initialize OSPI external flash and enable memory mapping. Refer to ospi_flash_thread_entry.c and related to OSPI files for more details.

```
/* Ospi Flash Thread entry function */
void ospi_flash_thread_entry(void)
{
    fsp_err_t     err            = FSP_SUCCESS;

    /* Initialize OSPI */
    err = ospi_b_init();
    if(FSP_SUCCESS != err)
    {
        APP_ERR_TRAP(FSP_ERR_ASSERTION);
    }

    while (1)
    {
        tx_thread_sleep (1);
    }
}
```

Figure 40. OSPI initialization

36. **Code highlight:** The following example creates a screen based on Widget Name in GUIX project and attached 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 41: Splash screen

37. **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 42: Splashscreen event

38. Set Switch SW4-3 OFF to Enable OSPI External flash.

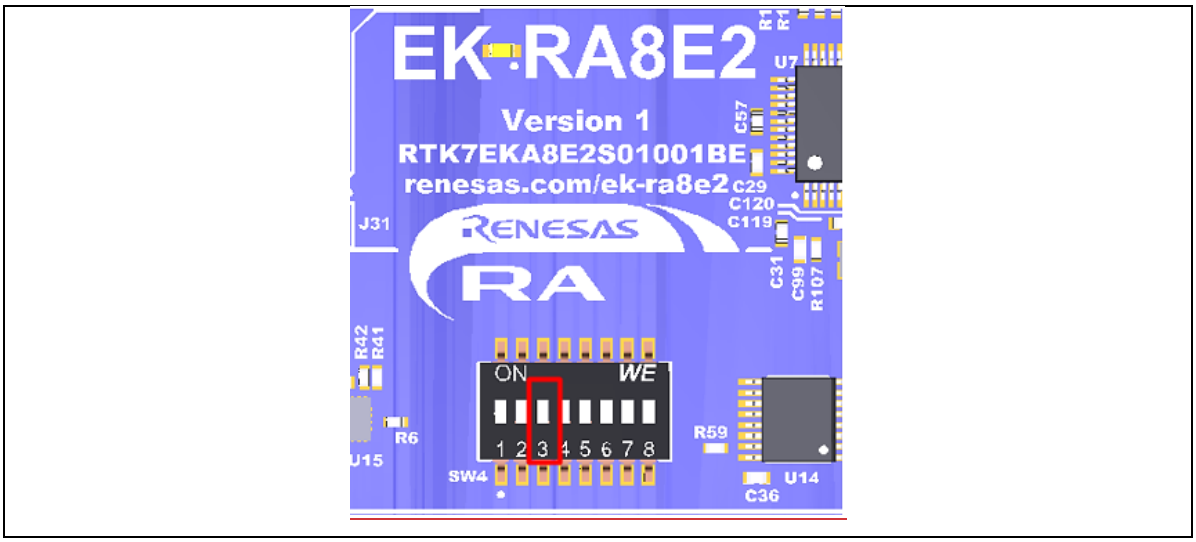


Figure 43: Enable OSPI External flash

39. Connect EK-RA8E2's Parallel Graphics Expansion board to J1 connector on EK-RA8E2 board as shown below.

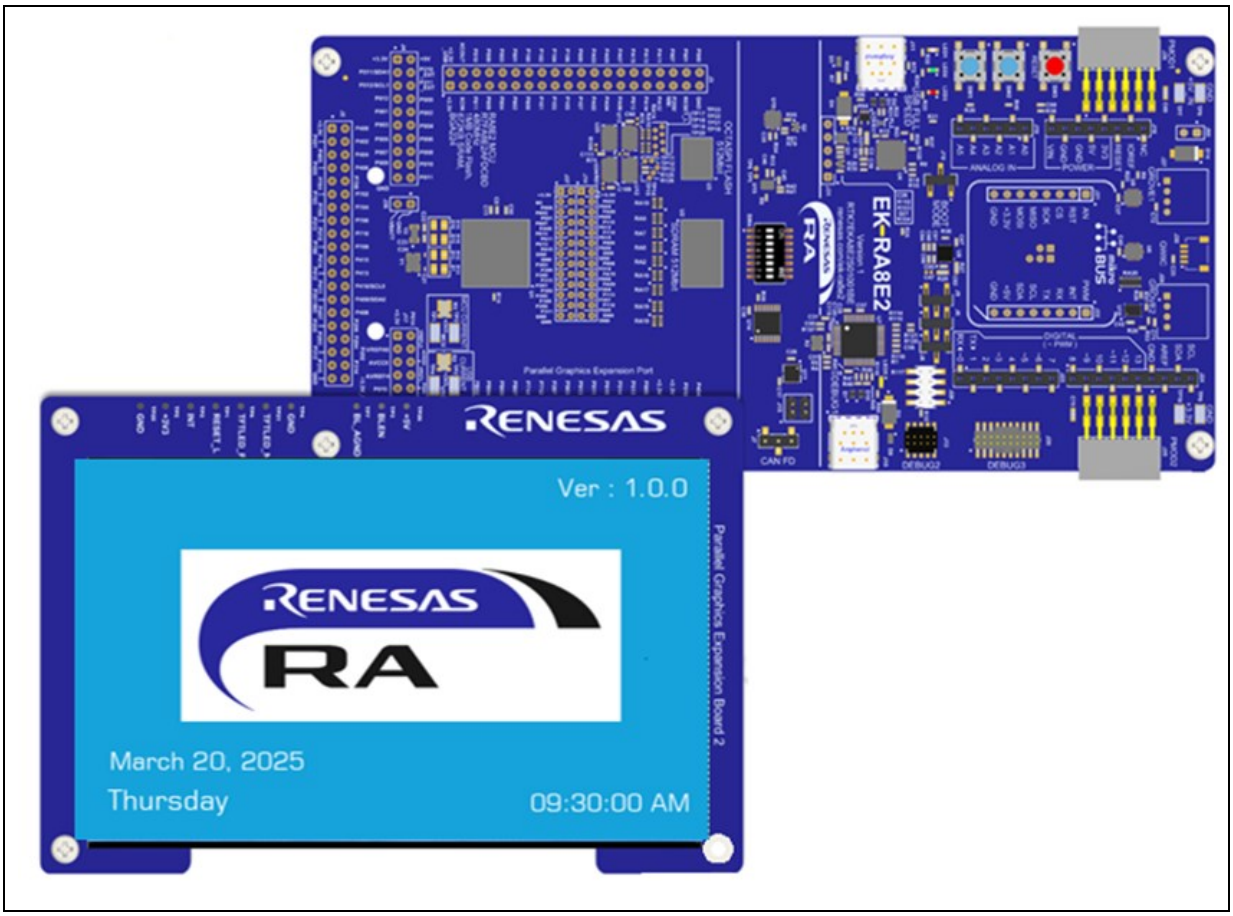


Figure 44. Connecting Parallel Graphics Expansion board to EK-RA8E2

40. Connect USB Debug using J10 on EK-RA8E2 board to your PC. Build Project  and start Debug  guix_thermostat_ek_ra8e2 project, you will see a black screen.

41. Add the following code to **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 45: Splashscreen event function

Please refer to splashscreen_event function in hmi_event_handler.c in "**2.2.34**" folder in the AN folder.

42. **Download and Run** the project, you will see the Splash screen on LCD panel.

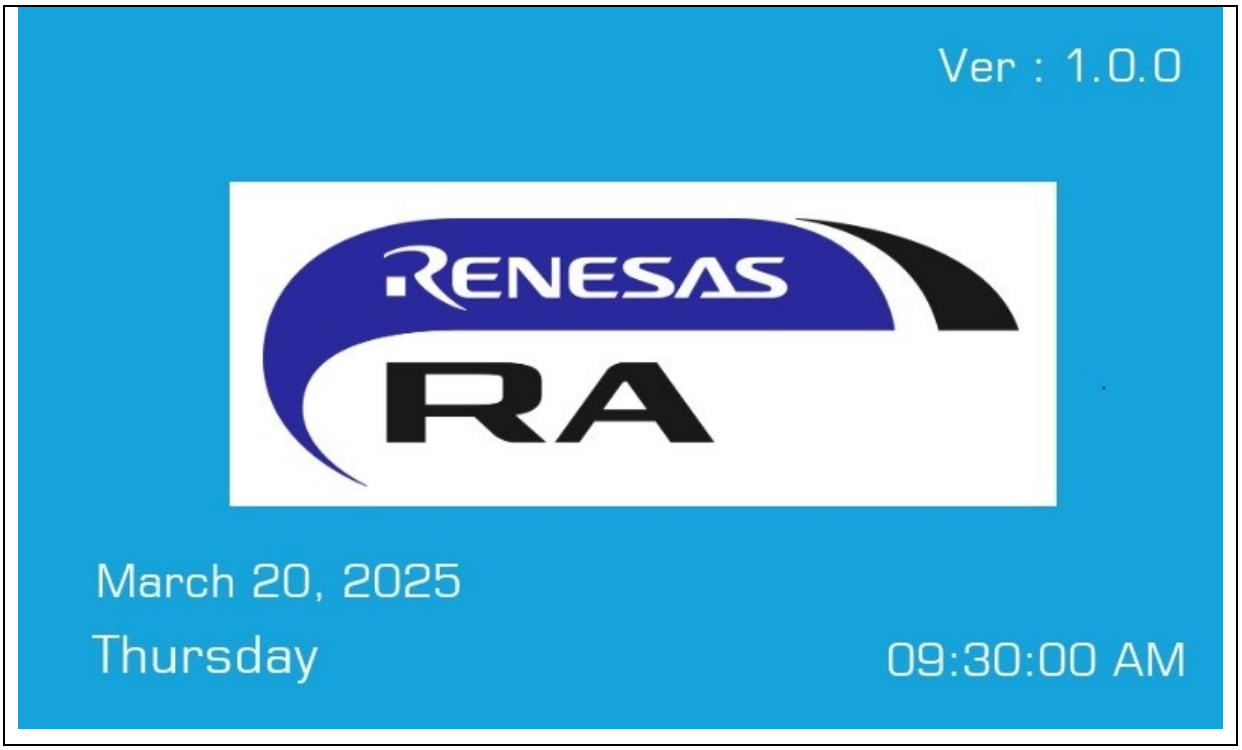


Figure 46. Splash Screen View on LCD

3. Using GUIX Widget Timer to Trigger a Screen Transition

3.1 Overview

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

3.2 Procedural Steps

1. Copy and replace these files in **"src"** folder in e² studio project with the files in **"3.2.1"** folder in the AN folder:
   - hmi_event_handler.c
   - system_thread_entry.c
   - system_api.h
2. **Code highlight**: The following code in splashscreen_event function starts a GUIX Widget timer and triggers a screen transition that hides Splash screen and shows 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 47: Splashscreen event function code

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

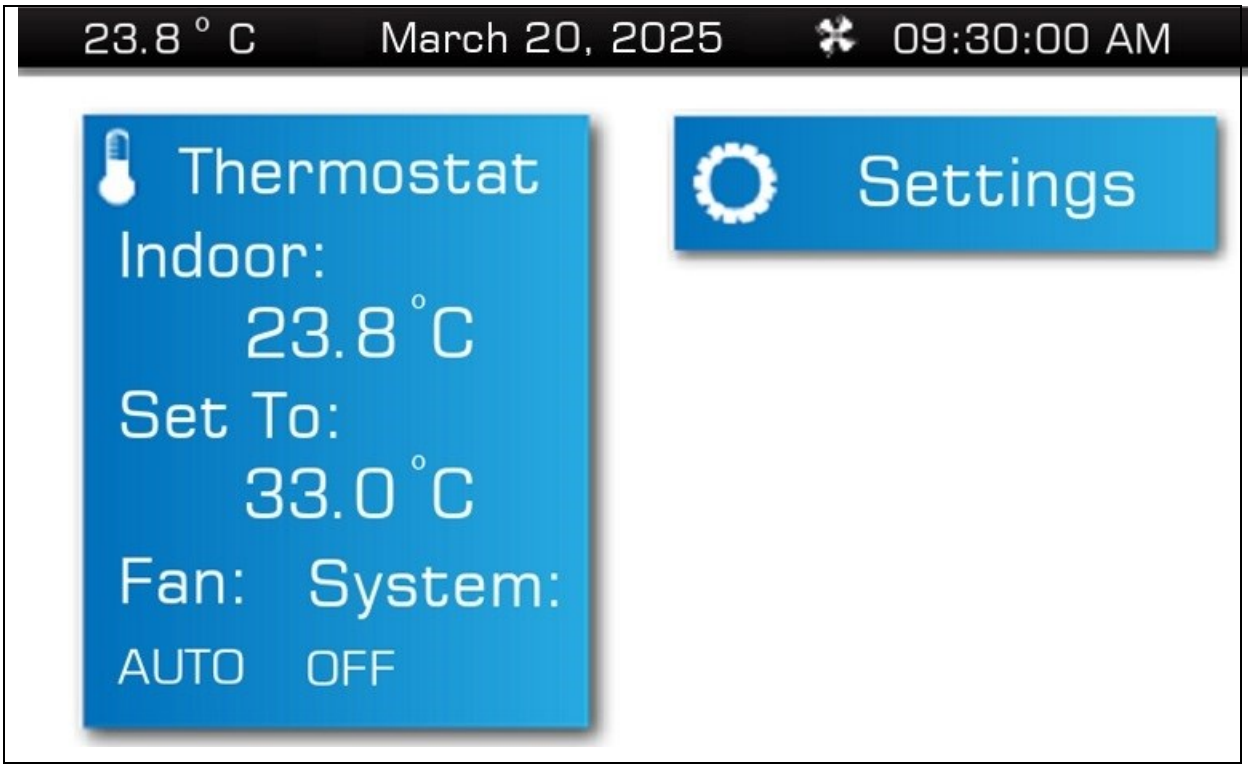


Figure 48. Main Page Screen

4. Add Touch Driver to Guix _Thermostat_EK_RA8E2 Project

4.1 Overview

In this section, you will add the gt911 touch driver to the project to handle touch events on LCD panel.

4.2 Procedural Steps

1. In guix_thermostat_ek_ra8e2 project, create a folder by right-clicking **"src",** then select **"New->Folder".**

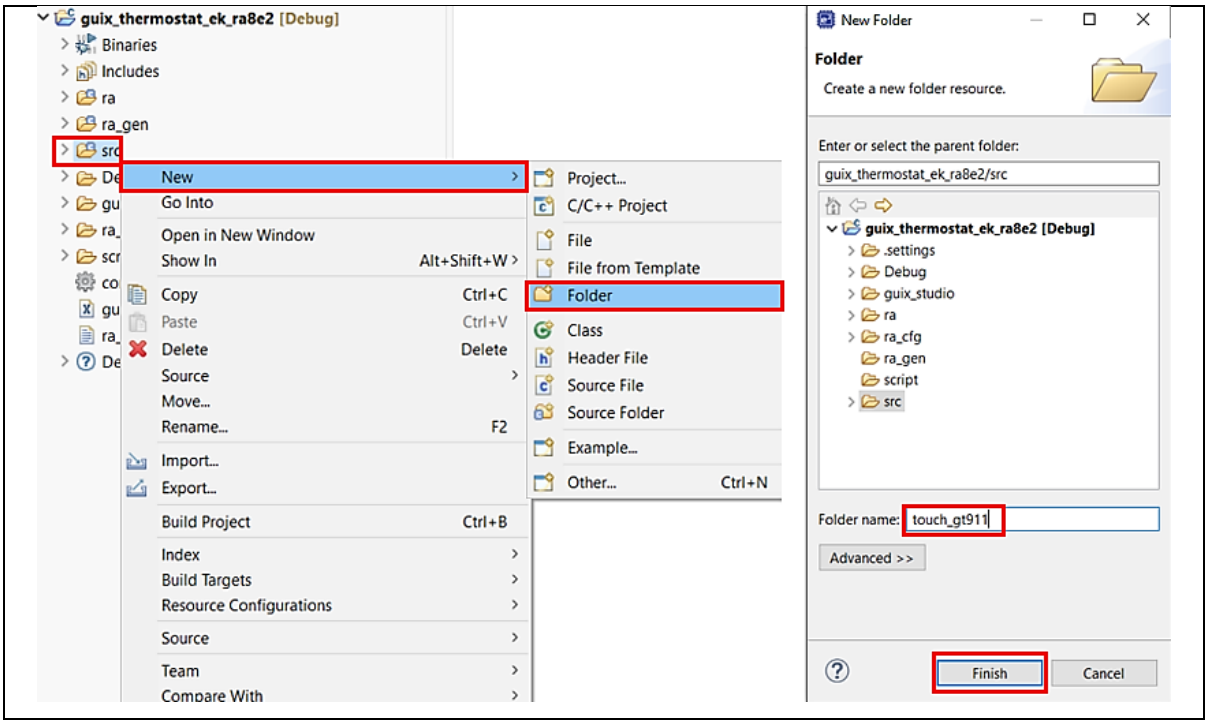


Figure 49. Creating New Folder in GUIX_THERMOSTAT_EK_RA8E2 Project

2. Copy touch_gt911.c and touch_gt911.h from "touch_gt911" folder in the Source file to the one in e² studio project.

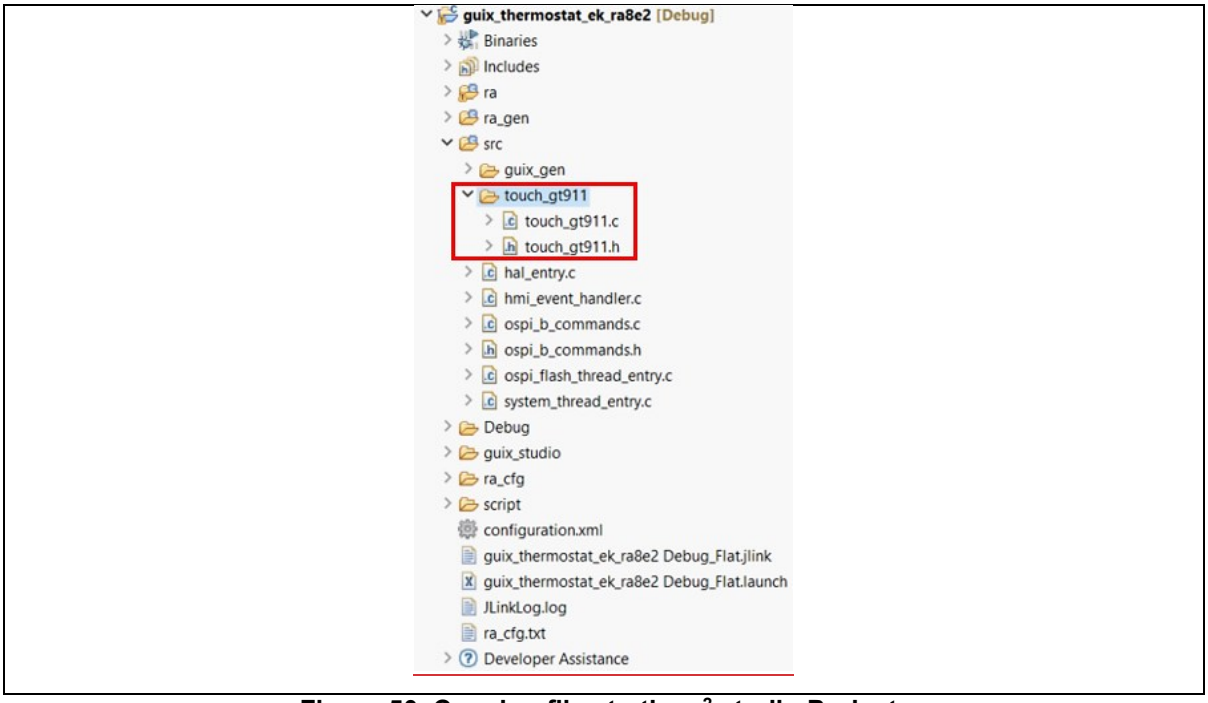


Figure 50. Copying files to the e² studio Project

3. Open project configuration and create **Touch Thread** with the settings below.

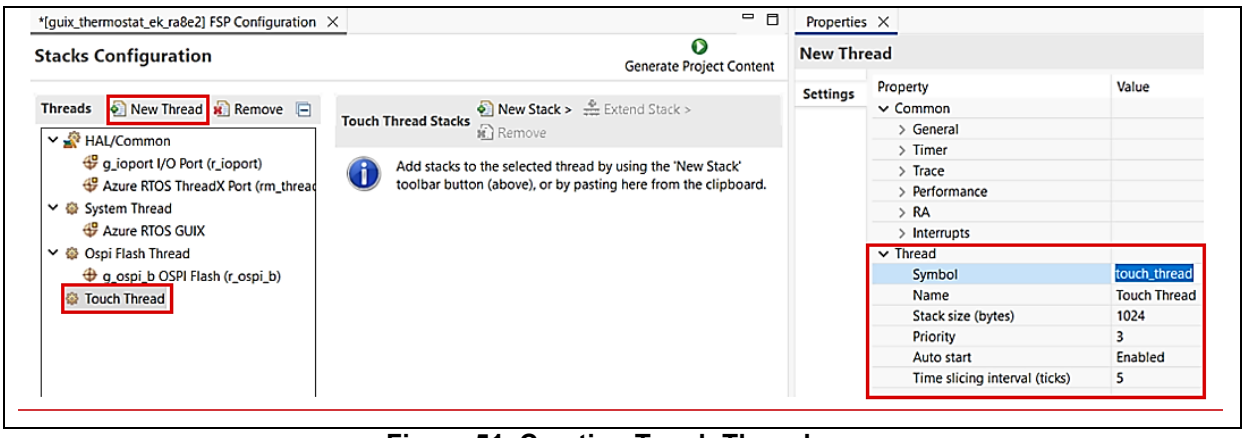


Figure 51. Creating Touch Thread

4. The pins marked in image below are used for touch panel controller on the LCD board:

   • IRQ3 interrupt (P510) is used to trigger touch events.

   • I2C channel 1 (P512, P511) is used to read and write data to the touch controller. PA01 is used to reset the touch controller.

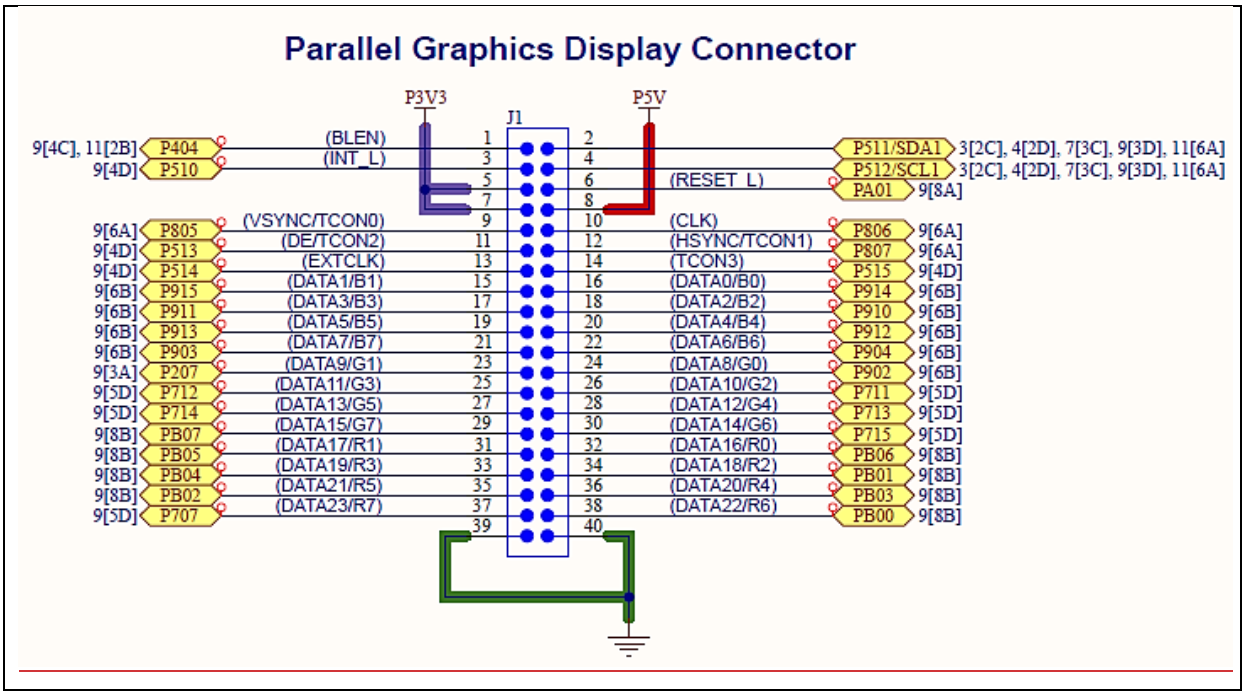


Figure 52. RA8E2 LCD Pin address

5. Setting GLCD_RST_L (PA01) Pin Configuration

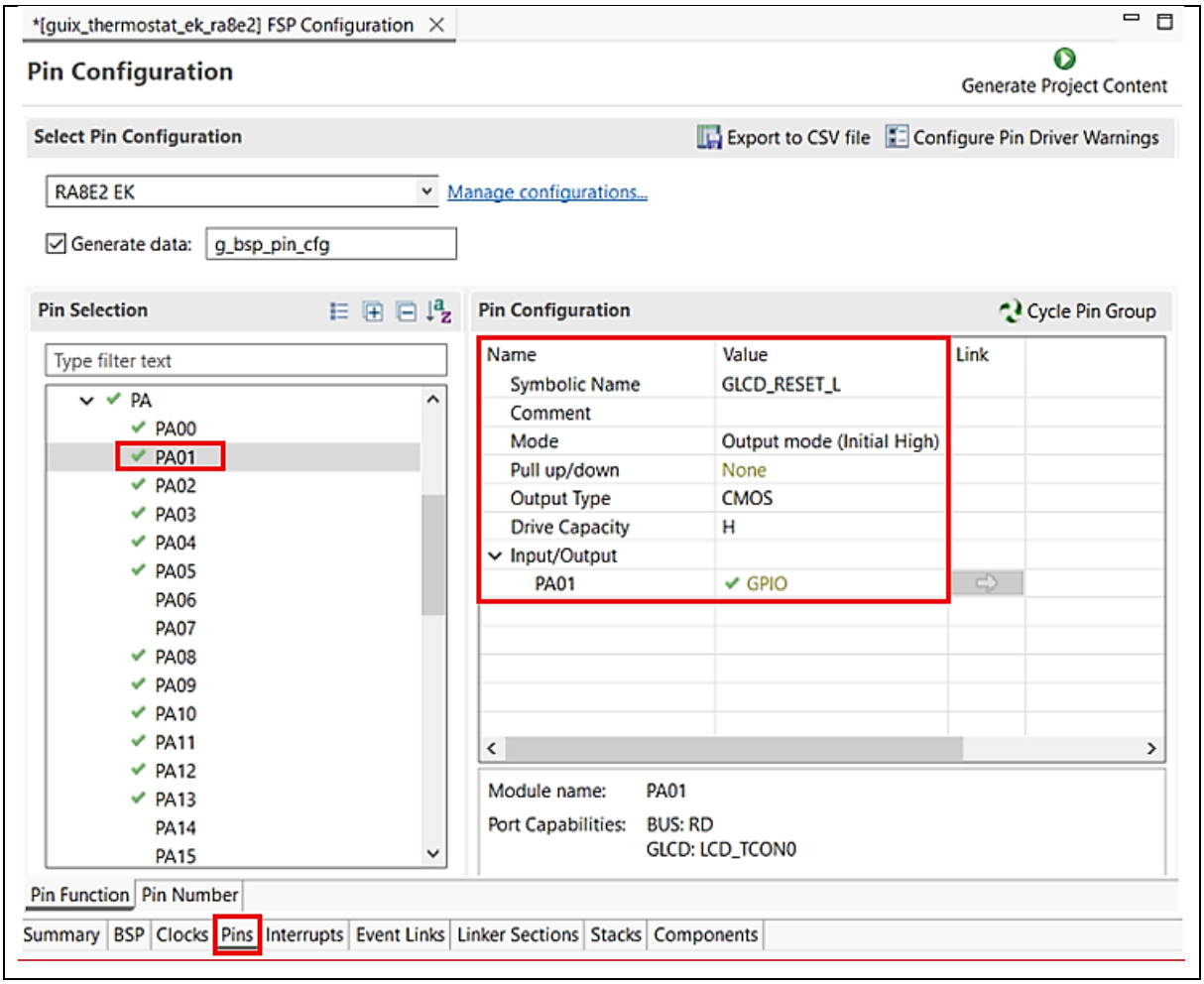


Figure 53. Setting GLCD_RST_L Pin Configuration

6. In e² studio project configuration, add **External IRQ Driver on r_icu** to **Touch Thread** with the following settings.

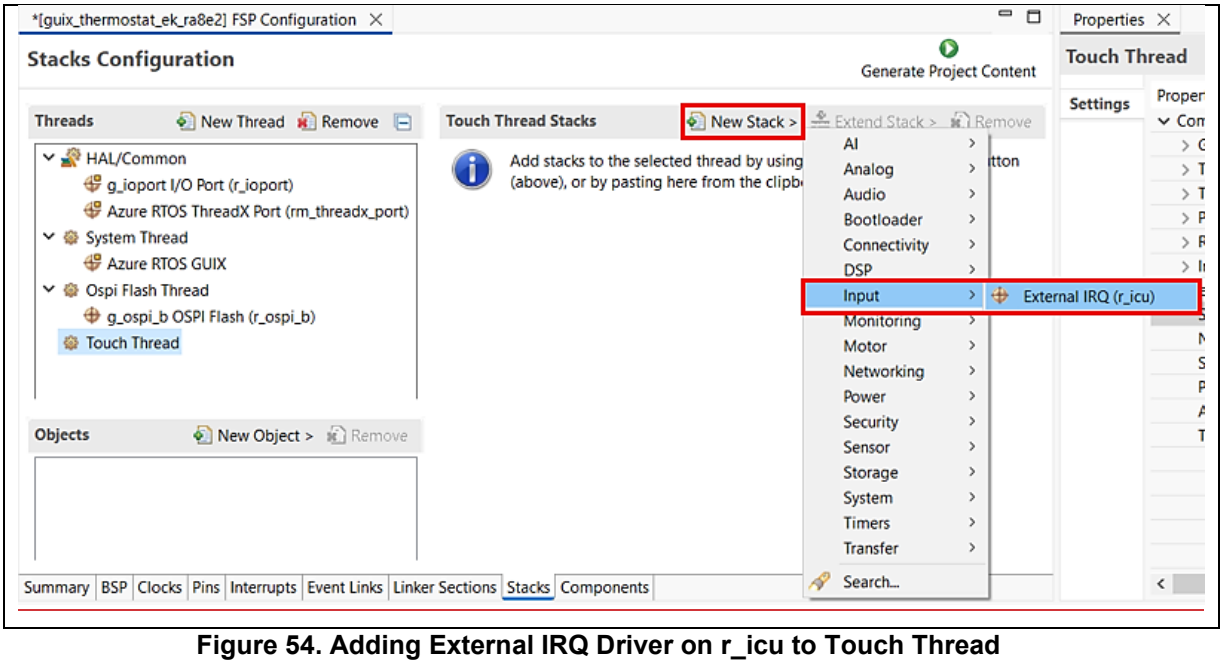


Figure 54. Adding External IRQ Driver on r_icu to Touch Thread

7. Adding and setting the **External IRQ** Properties.

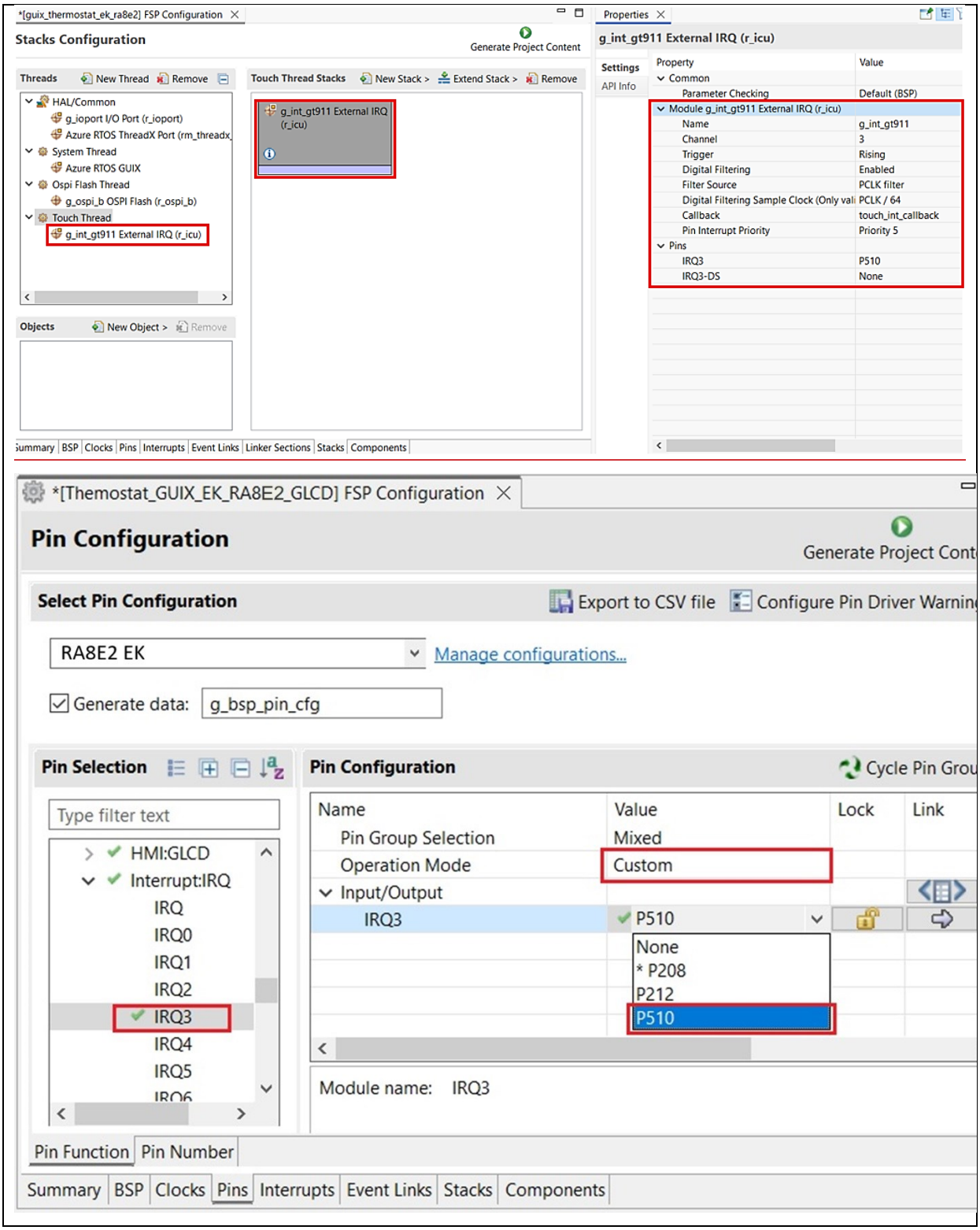


Figure 55. Settings External IRQ Properties

8. Adding and setting Properties **I2C Master Driver on r_iic_master** to **Touch Thread**

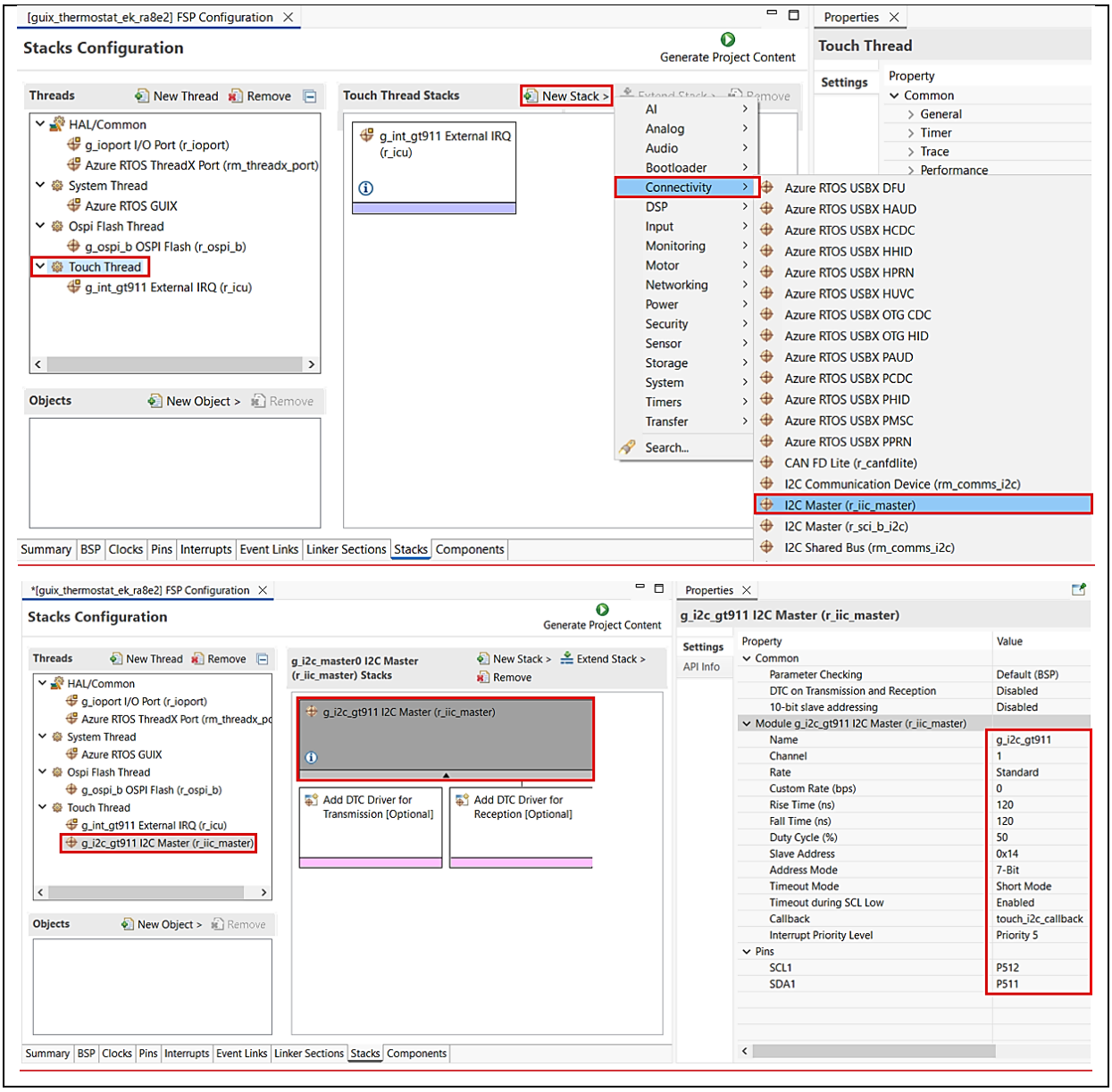


Figure 56. Adding I2C Master Driver on r_iic_master to Touch Thread

9. In project configuration, add **I2C Semaphore** as shown below. This semaphore is used in the gt911 driver to trigger data reading when a touch-panel interrupts occurs.

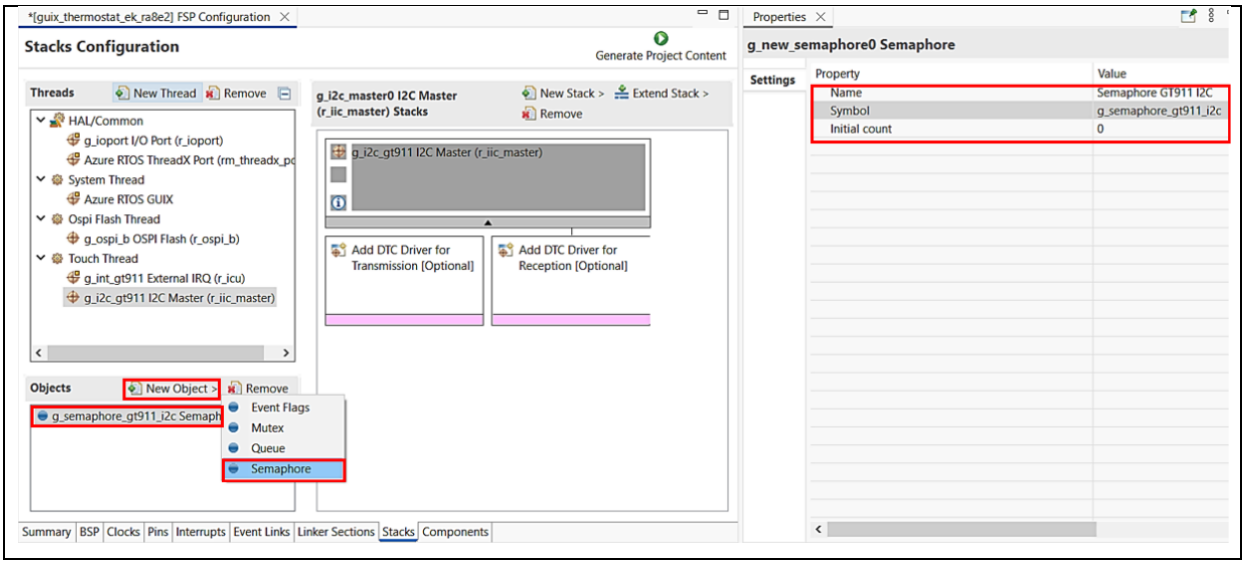


Figure 57. Adding I2C Semaphore

10. In project configuration, add **Touch Semaphore** as shown below. We use this semaphore to signal the Touch thread when a touch event occurs. The Touch thread then sends the touch event to GUIX.

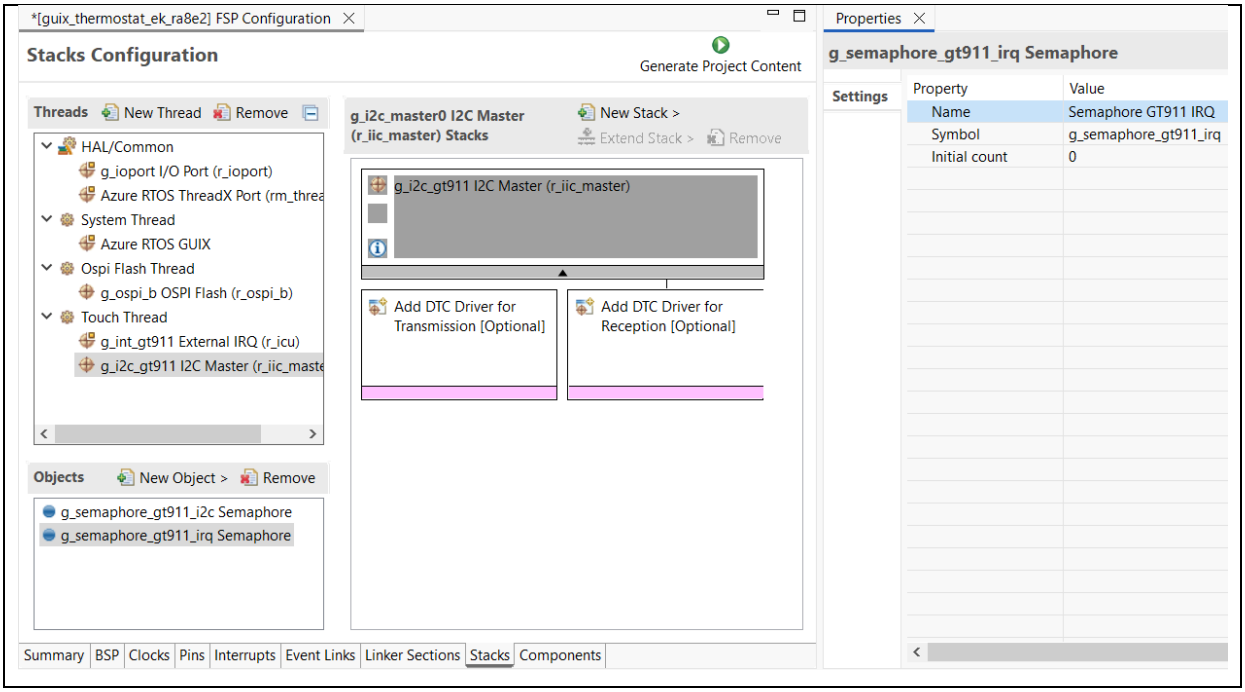


Figure 58. Adding Touch Semaphore

11. In RA Configurator, click Generate Project Content to generate project content.
12. Copy and replace these files in **"src"** folder in the e^2 studio project with the files in **"4.2.12"** folder in the AN folder:

- hmi_event_handler.c
- system_api.h
- system_thread_entry.c
- touch_thread_entry.c

13. **Code highlight:** Below code in touch_thread_entry.c get touch data and send touch event to GUIX.

```
if(status & GT911_BUFFER_READY)
{
    gt911_touch_coords.touch_count = status & GT911_MASK_TOUCH_COUNT;
    err = R_TOUCH_GT911_PointsGet(&gt911_ctrl, &gt911_touch_coords);

    if(0U < gt911_touch_coords.touch_count && GX_EVENT_PEN_UP == gxe.gx_event_type)
    {
        /* Send only the TOUCH event of the first finger to the GUIX Model View Controller. */

        translate_touch_coordinates(&gt911_touch_coords.point[0], ORIENTATION_NONE);

        gxe.gx_event_payload.gx_event_pointdata.gx_point_x       = gt911_touch_coords.point[0].x;
        gxe.gx_event_payload.gx_event_pointdata.gx_point_y       = gt911_touch_coords.point[0].y;
        gxe.gx_event_type = GX_EVENT_PEN_DOWN;
        gx_system_event_send(&gxe);

    }

    if ( 0U == gt911_touch_coords.touch_count && GX_EVENT_PEN_DOWN == gxe.gx_event_type)
    {
       gxe.gx_event_type = GX_EVENT_PEN_UP;
       gx_system_event_send(&gxe);
    }
    tx_thread_sleep (1);

    if(FSP_SUCCESS != err)
    {
        APP_ERR_TRAP(err);
    }
}
```

Figure 59. Sending touch event to GUIX

14. 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 60: Screens created in systems thread entry

15. The code marked in red in hmi_event_handler.c handle touch event when Thermostat button and Settings button 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 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_EVENT_SHOW:
            /** Update initial text fields according to system settings before the window shows. */
            /** Do default window processing first. */
            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;
    }
    return gx_err;
}
```

```
        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 61: Thermostat button and Settings button clicked

16. **Build, Download, and Run** the e² studio project. Then, you will be able to go back and forth from the Main Page screen to Thermostat screen and Settings screen using **Thermostat** and **Settings** buttons on Main Page screen and "**Back**" button on the other two screens.

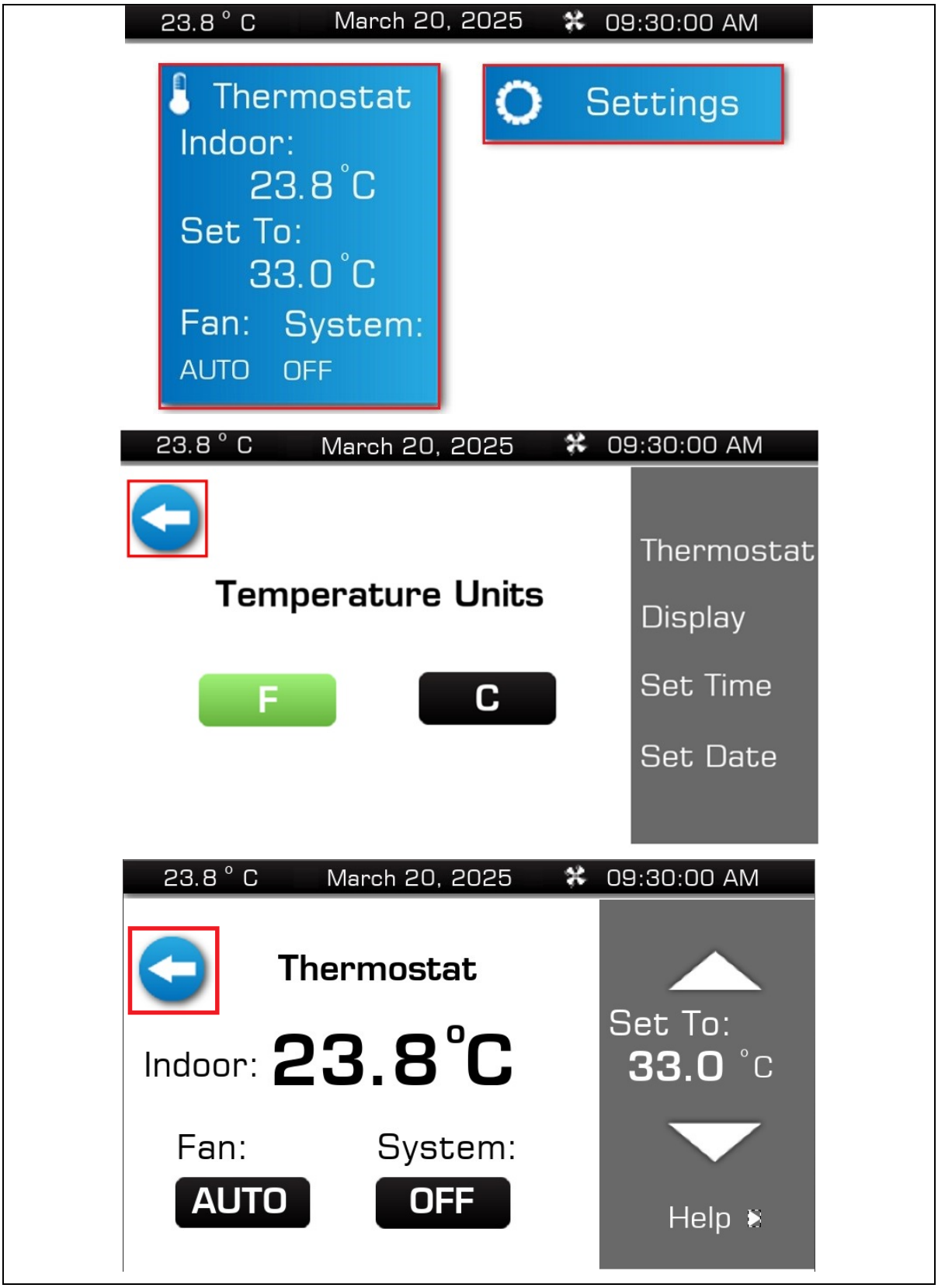


Figure 62. Navigating between Main Page Screen and Thermostat Screen

5. Control 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 LCD backlight.

5.2 Procedural Steps

1. In LCD board schematics below, the GLCD_BLEN signal, which is connected to the P404 on the RA8E2 MCU, is configured in PWM mode to control the intensity of LCD backlight.

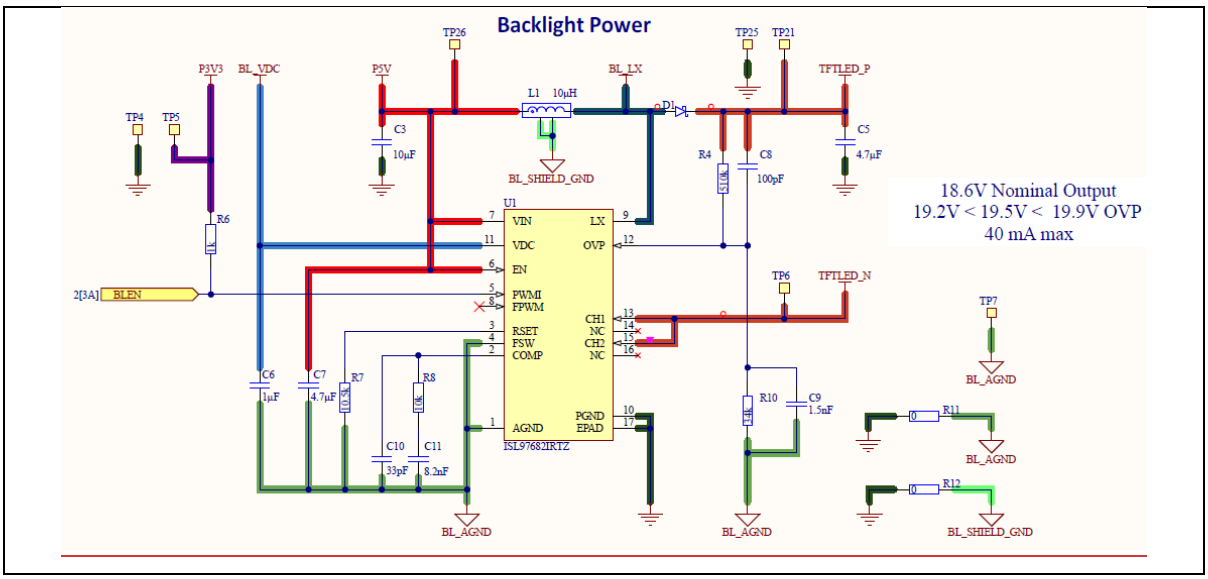


Figure 63. LCD Board Schematic

2. In Pin Configuration, set **P404** as **GPT3 - GTIOCB** output.

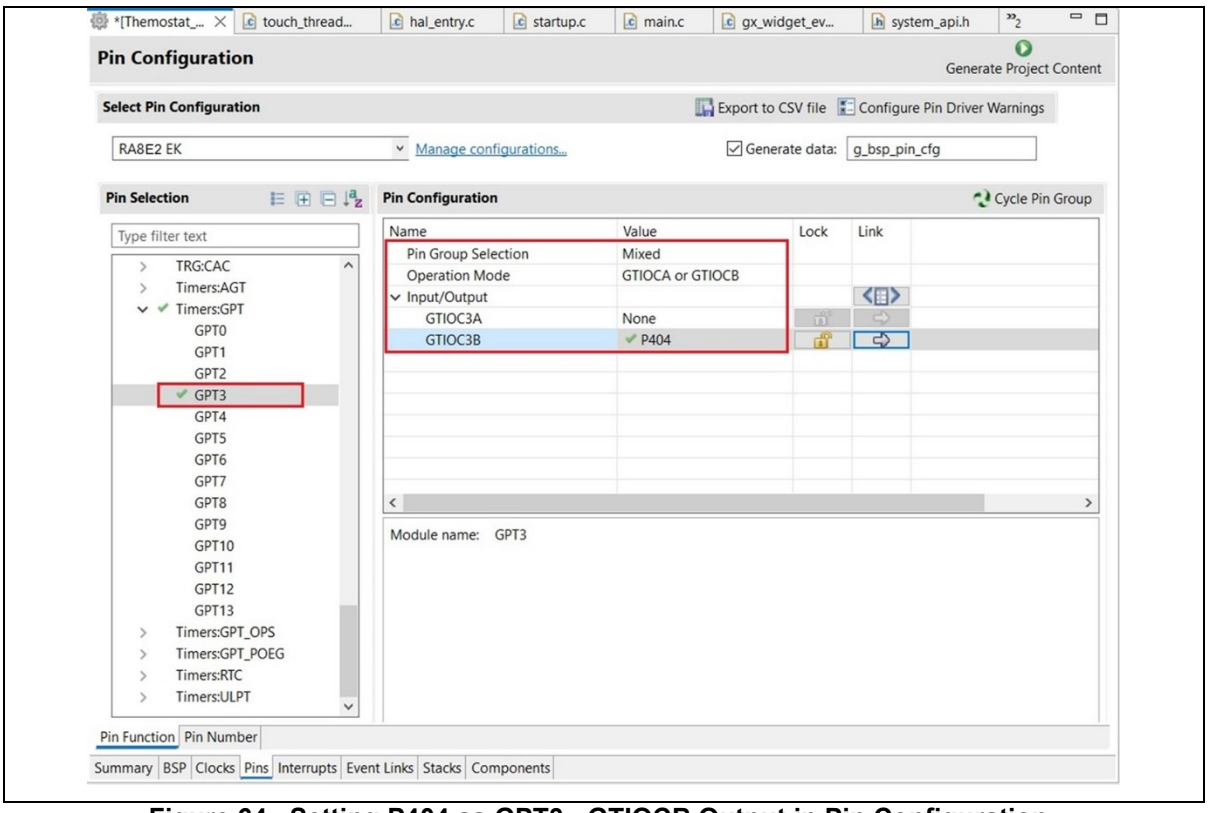


Figure 64. Setting P404 as GPT3 - GTIOCB Output in Pin Configuration

3. Add **Timer Driver** on **r_gpt** to **System Thread** with below settings.

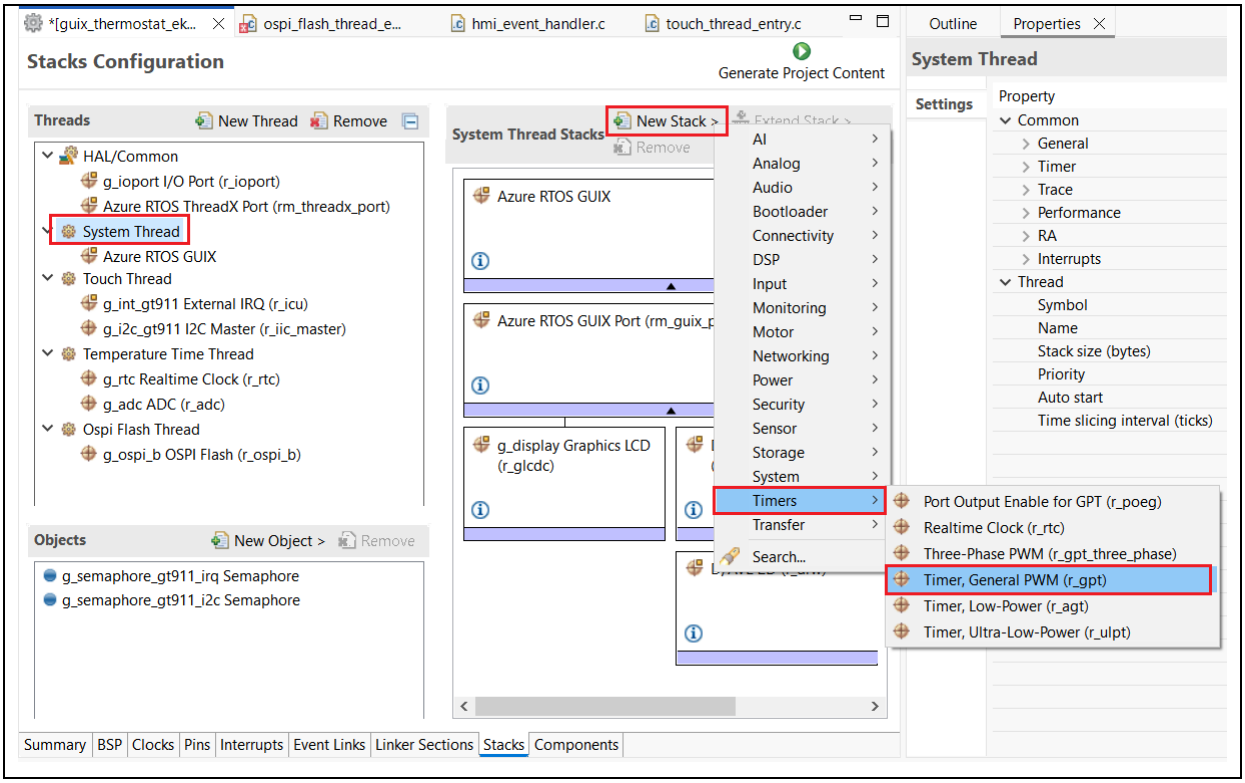


Figure 65. Adding Timer Driver on r_gpt to System Thread

4. Setting Timer Driver on r_gpt Properties.

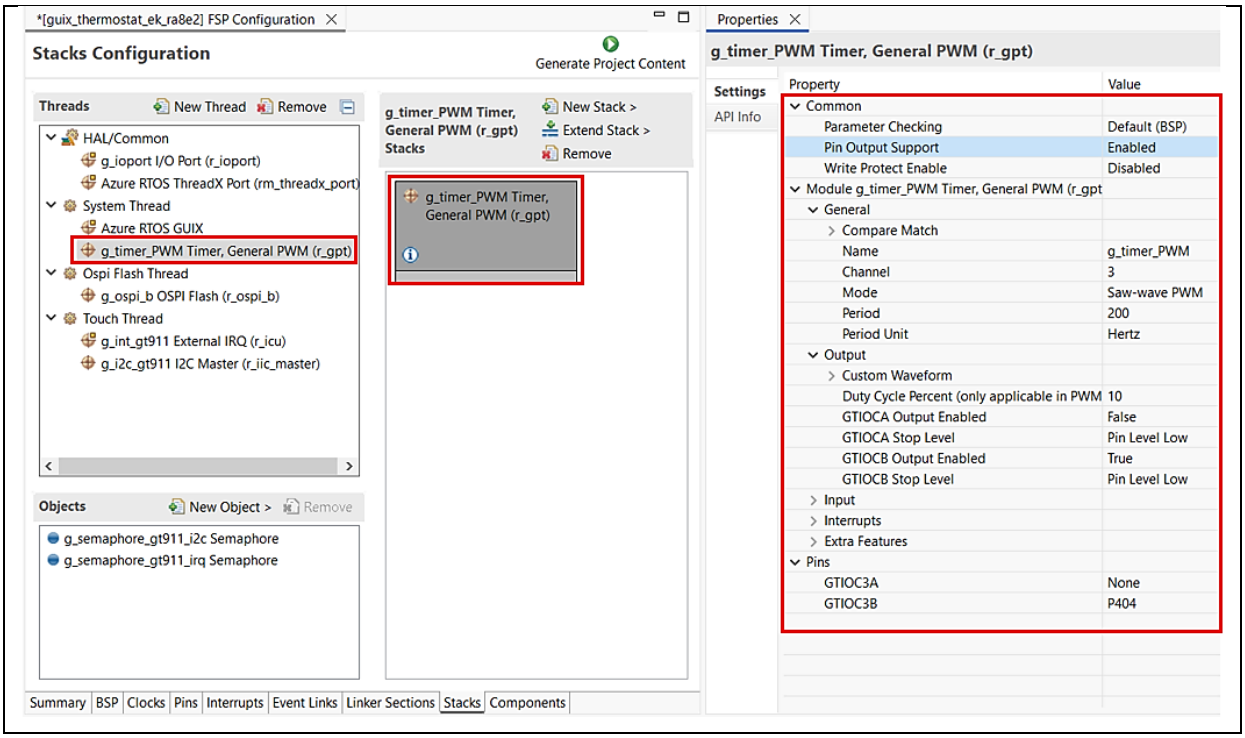
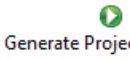


Figure 66. Adding Timer Driver on r_gpt to System Thread

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. In RA Configurator, click 

to generate project content.
6. Copy and replace these files in "**src**" folder in e^2 studio project with the files in "**5.2.6**" folder in the AN folder:
   - hmi_event_handler.c
   - system_thread_entry.c
   - brightness.c
   - brightness.h
   - system_api.h
   - system_cfg.h
7. 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 67: Setting the PWM duty cycle

8. Looking at gpt_timer_PWM_Setup function in system_thread_entry.c, you will see 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 68. duty cycle of PWM output

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

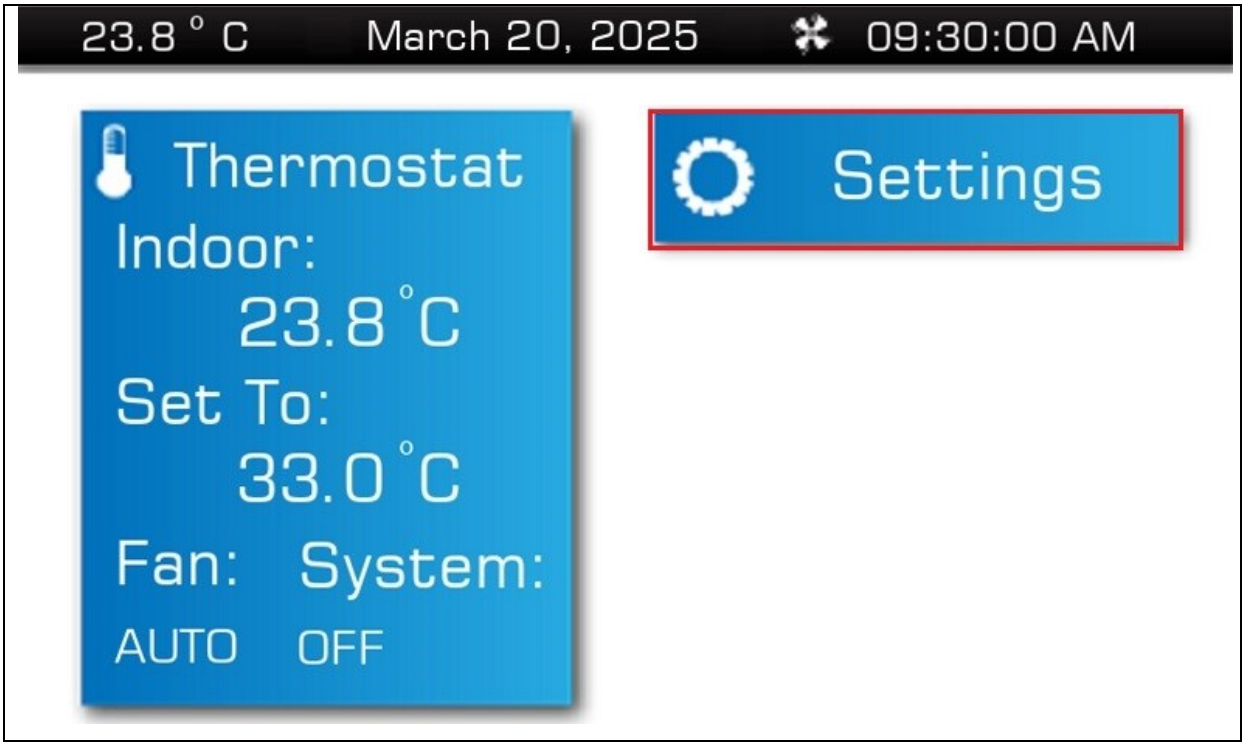


Figure 69. Settings Button on Main Page Screen

10. PWM output measured on pin P404 with brightness is set to 50%.

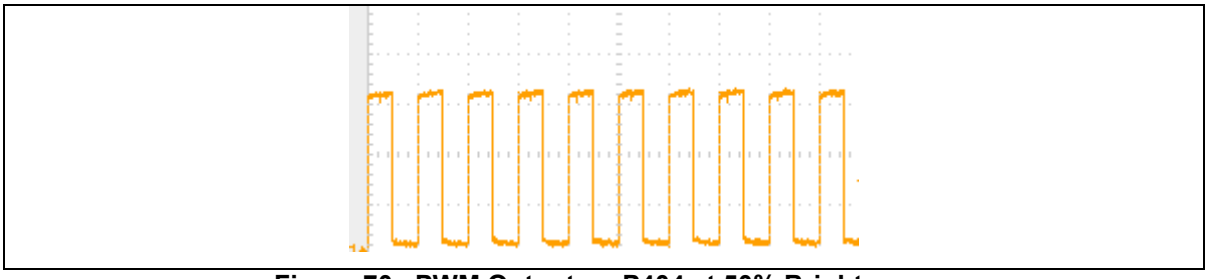


Figure 70. PWM Output on P404 at 50% Brightness

11. Click "**Display**" menu on **Settings** screen, you can use "**Up**" and "**Down**" buttons to change the brightness of LCD backlight.

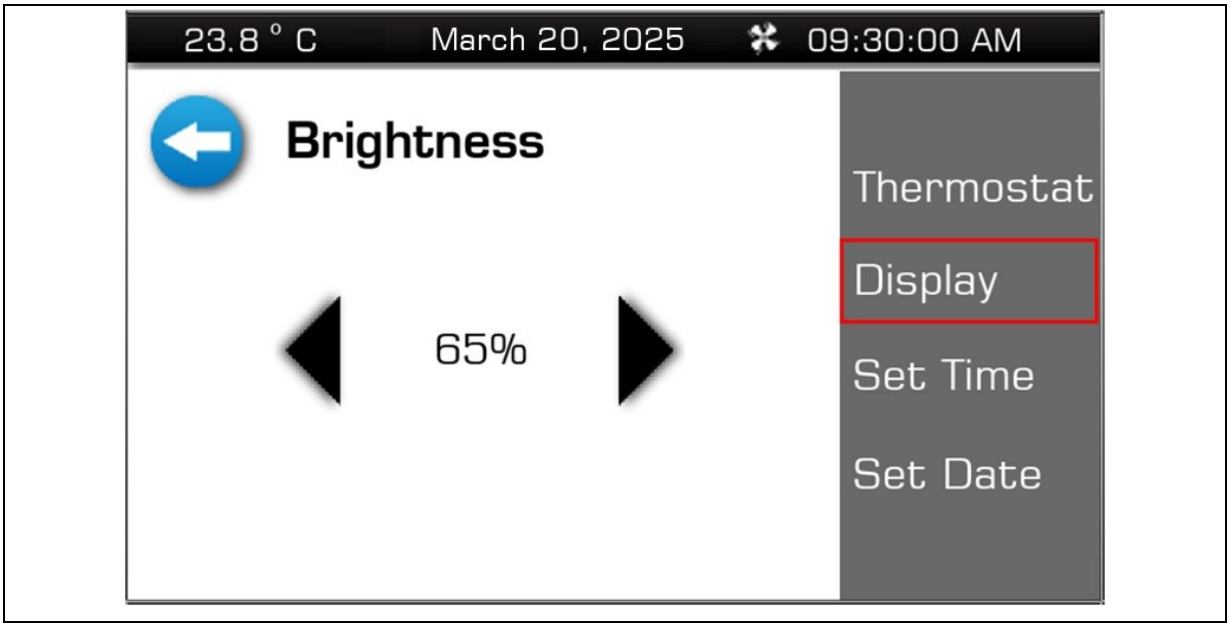


Figure 71. Display of Settings Screen

12. PWM output measured on pin P404 after changing brightness to **65%**.

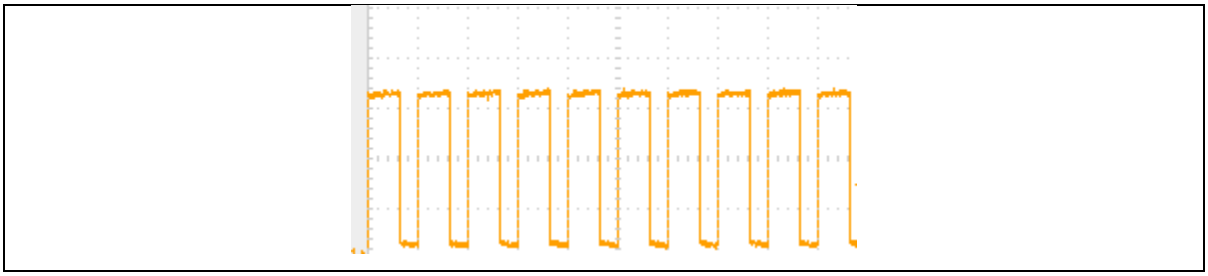


Figure 72. PWM Output on P404 at 65% Brightness

6. Update Date/Time and Temperature

6.1 Overview

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

6.2 Procedural Steps

1. In project configuration, create **Temperature Time Thread** and **set Properties**.

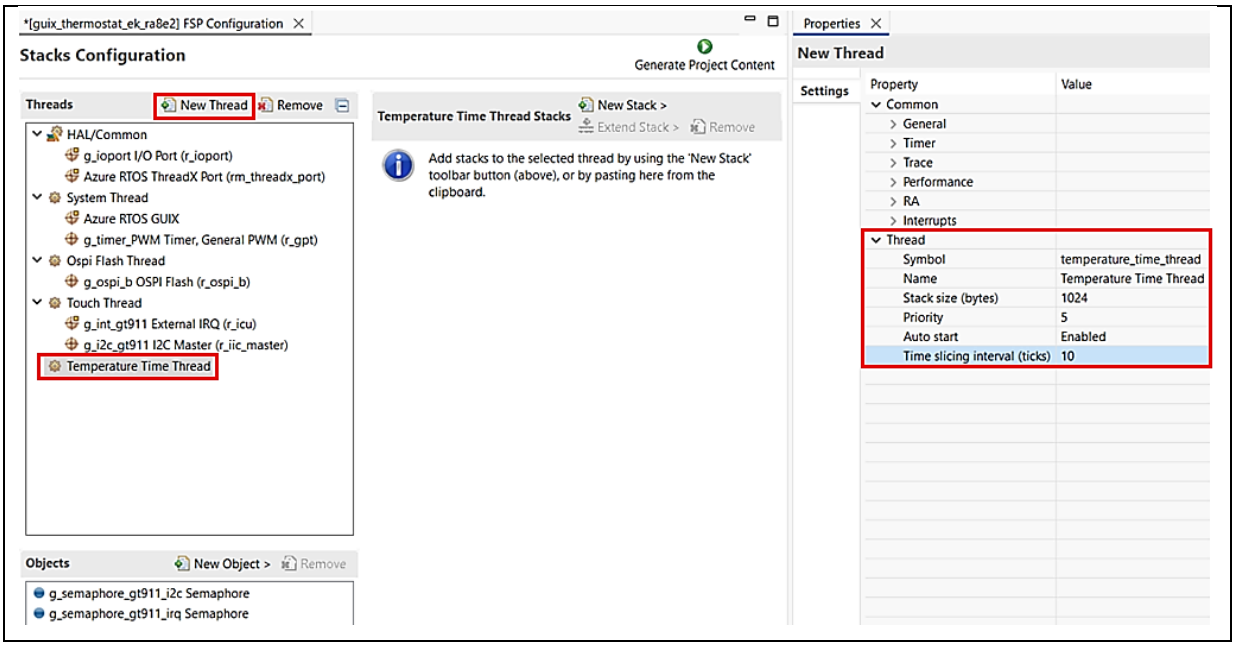


Figure 73. Create Temperature Time Thread

2. Adding Realtime Clock Driver (r_rtc) to Temperature Time Thread.

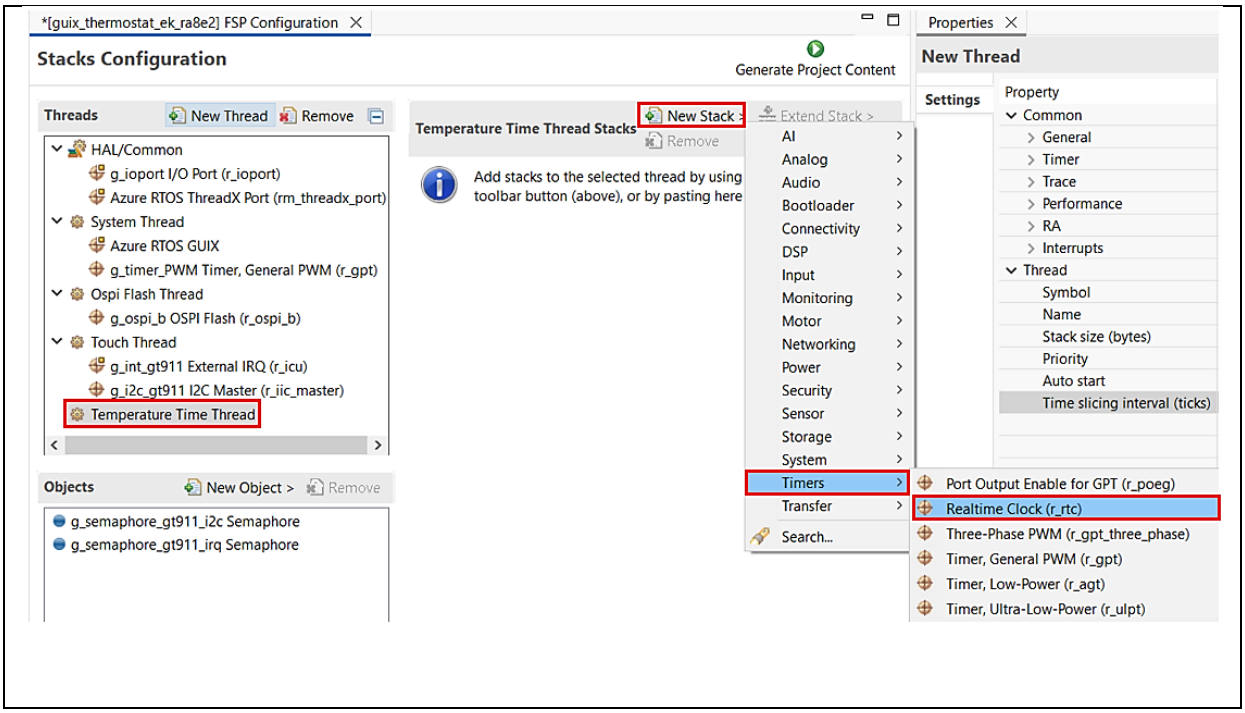


Figure 74. Adding RTC Driver.

3. Setting Realtime Clock Driver (r_rtc) Properties.

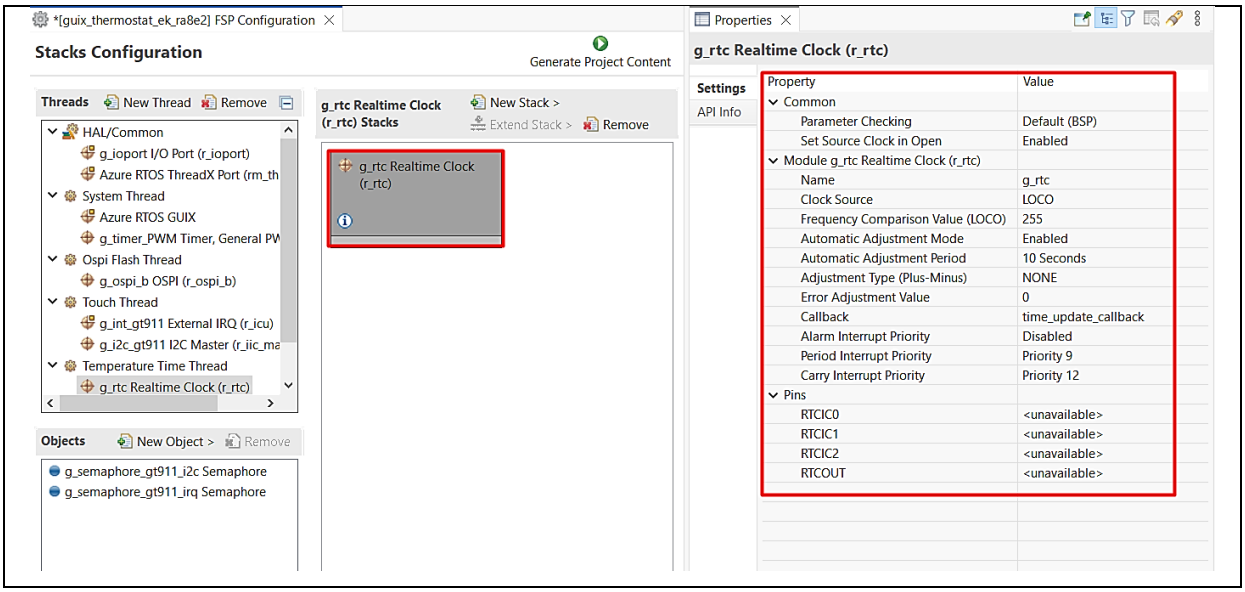
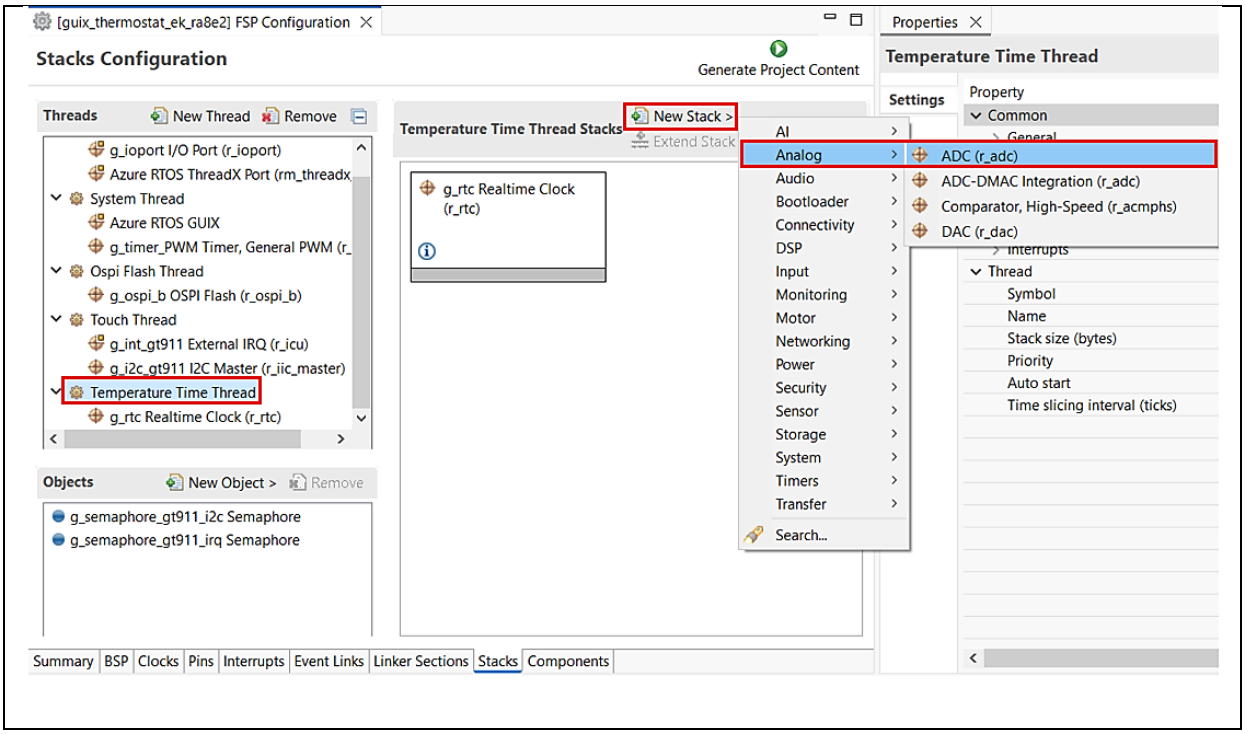


Figure 75. Setting Realtime Clock Properties.

4. Adding ADC Driver (r_adc) to Temperature Time Thread and setting Properties.



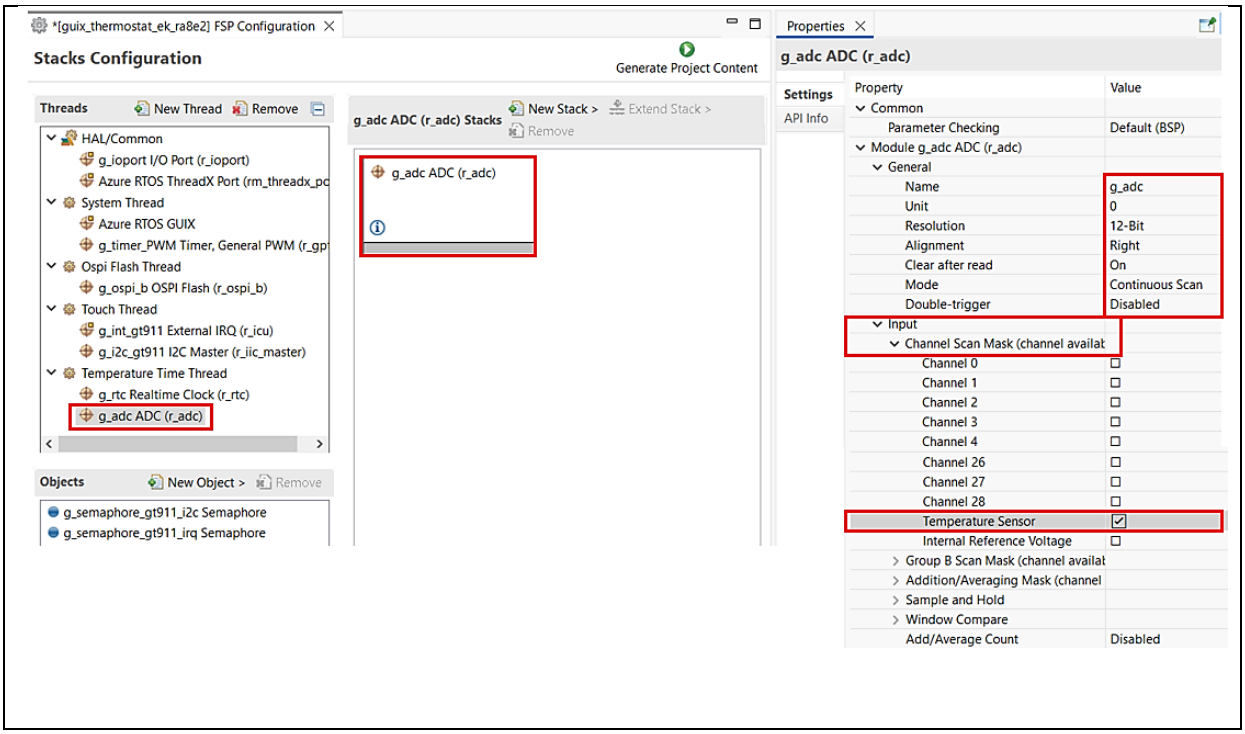


Figure 76. Adding ADC Driver on r_adc to System Thread and Setting Properties

5. Create **g_timer_semaphore** with the following settings. We use this semaphore to trigger the date and time update every second.

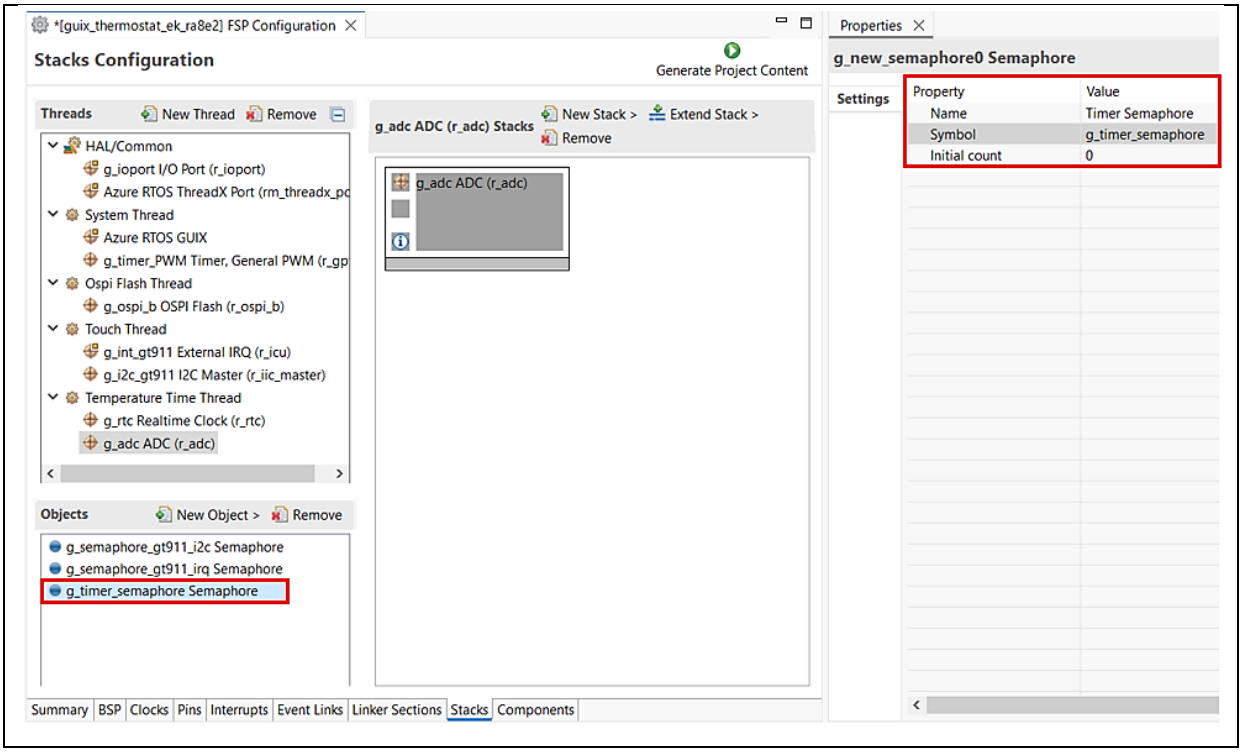


Figure 77. Creating g_timer_semaphore and setting properties

6. Create **system_msg_queue Queue** and setting Properties.

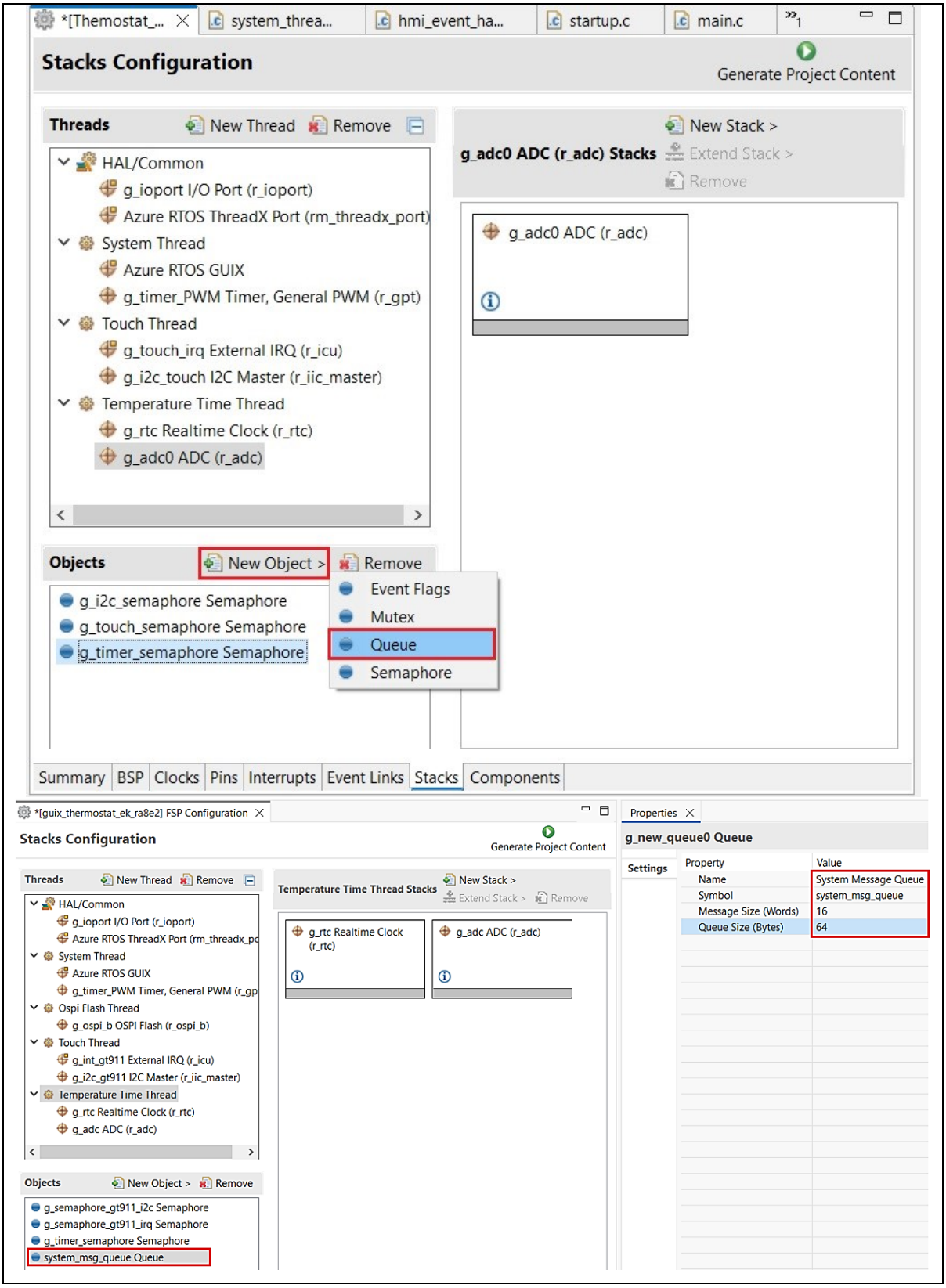


Figure 78. Creating system_msg_queue Queue and setting properties

7. Create **g_sys_mutex Mutex** and setting properties.

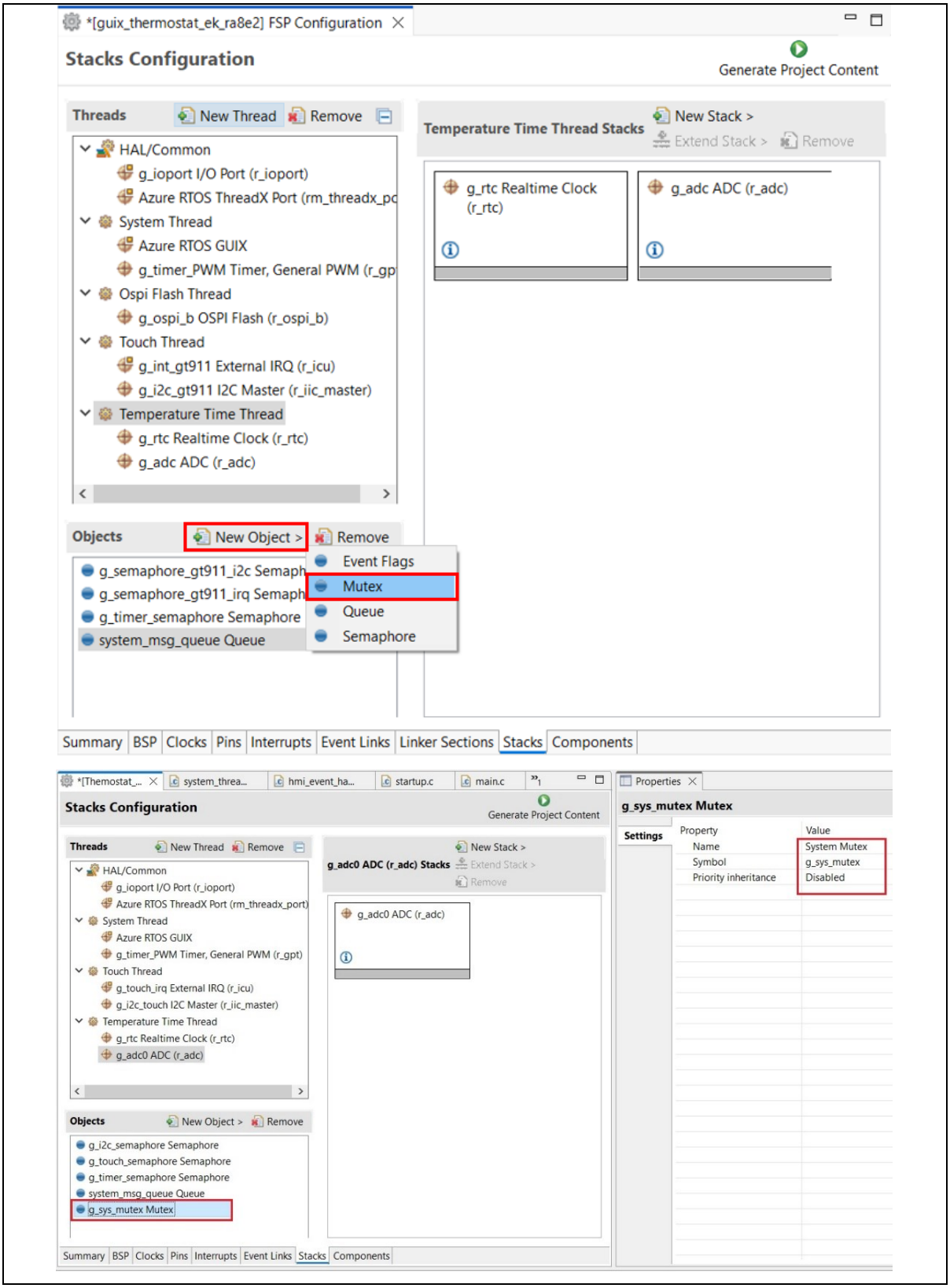


Figure 79. Creating g_sys_mutex Queue

8. In RA Configurator, click Generate Project Content to generate project content.
9. Copy and replace these files in **"src"** folder in e² studio project with the files in **"6.2.8"** folder in the Lab folder:
   - hmi_event_handler.c
   - system_thread_entry.c
   - system_time.c
   - system_time.h
   - system_api.h
   - temperature_time_thread_entry.c
10. In System Thread, date/time data and temperature data get updated every second. It then sends out events to trigger GUIX updates.
11. 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 80: Main page screen event handler

12. **Build, Download, and Run** the e² studio project. You will see time and temperature get updated every second.

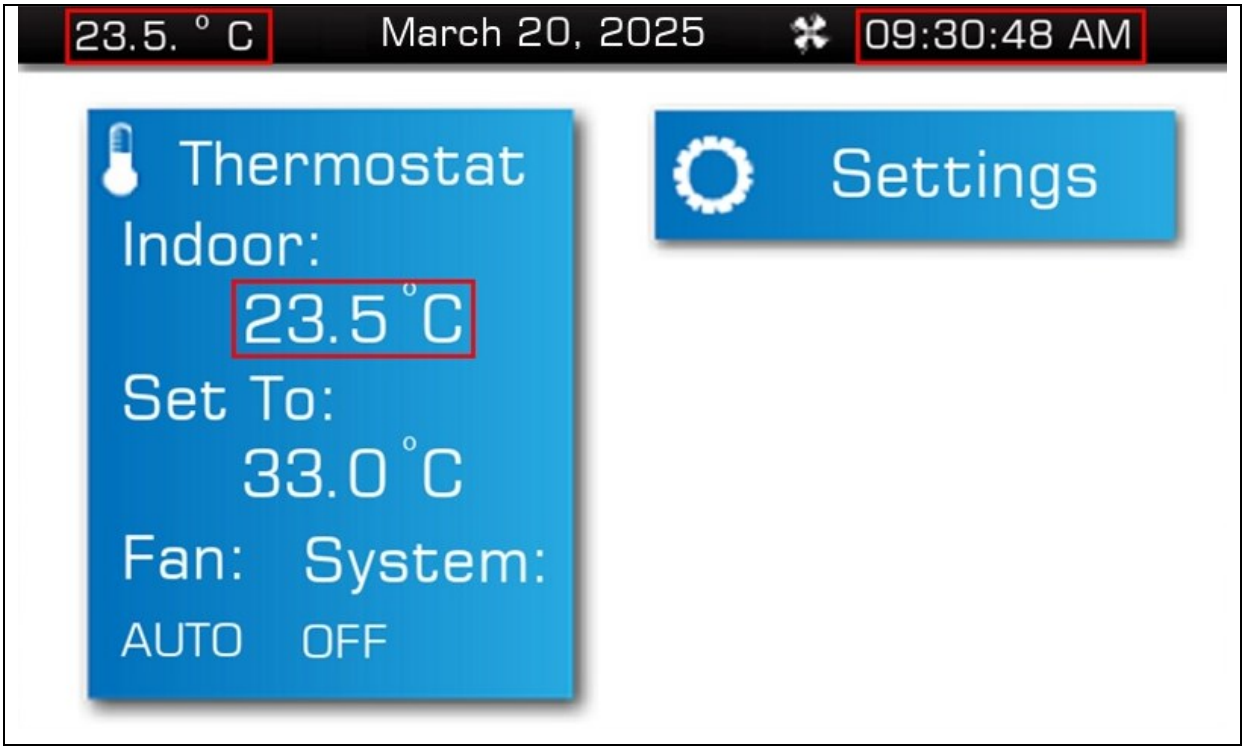


Figure 81. Time, Temperature on Main Page Screen

7. Testing and debugging in A Full Function Project

7.1 Overview

In this section, you will import and run the complete Thermostat project that enables the settings of date and time. **Upon user press 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 time display update on screens.**

7.2 Procedural Steps

1. You can try the completed project in "**completed_project**" folder that has a full functional Thermostat application. Use "**Rename & Import Existing C/C++ Project into Workspace**" feature of **Import** menu in e^2 studio to do so since you already had a project with the same in the workspace.

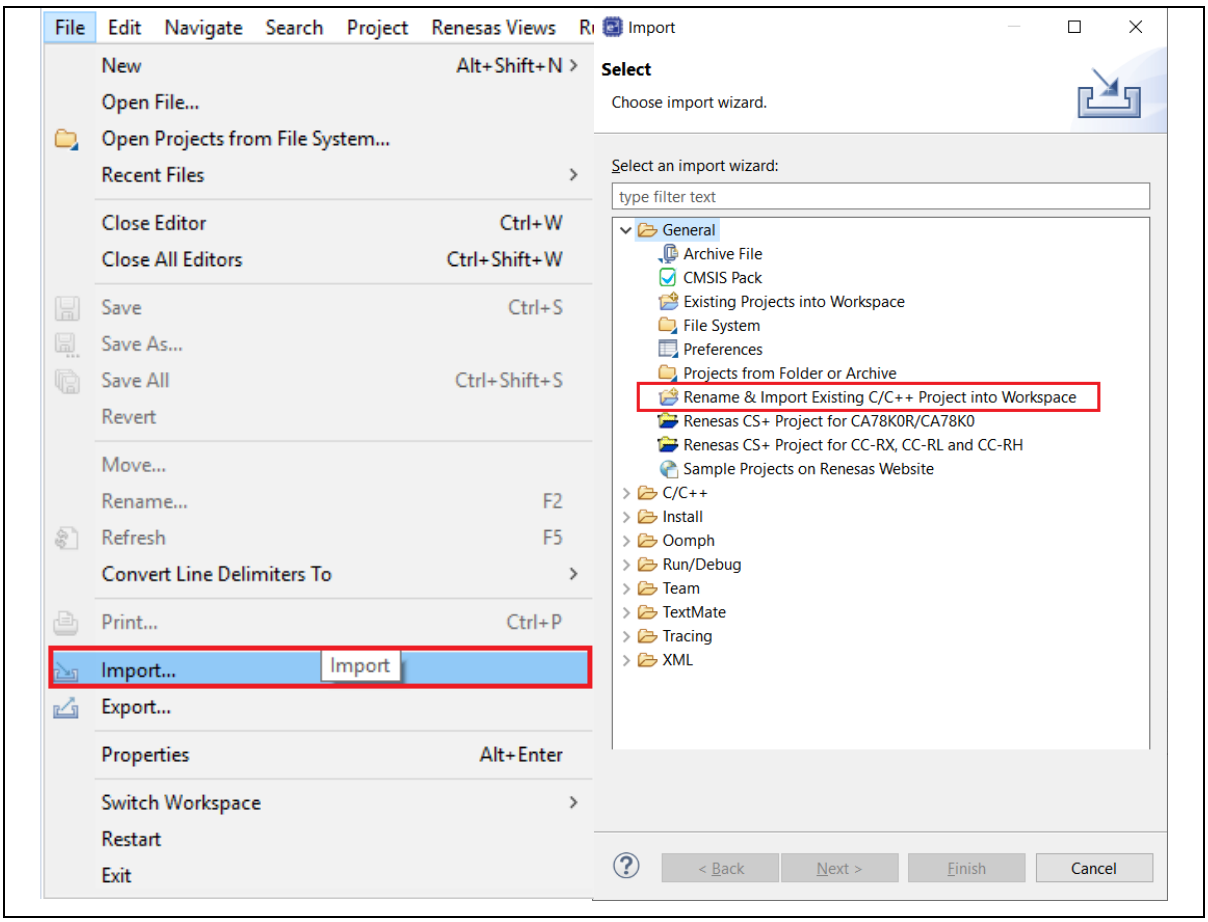


Figure 82. Rename & Import Existing C/C++ Project into Workspace on Import Menu

2. Once the project is imported, open the configuration.xml Stack, click Generate Project Content, compile the project without errors and proceed with the evaluation.

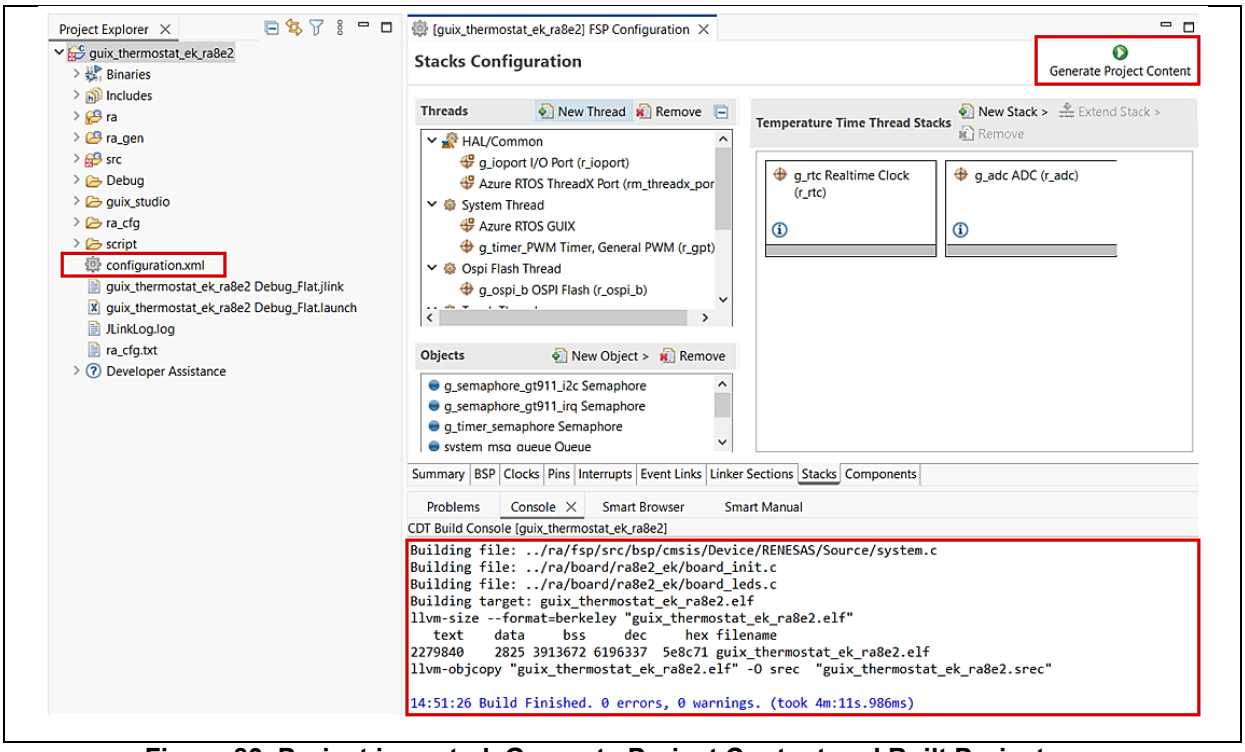


Figure 83. Project imported, Generate Project Content and Built Project.

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	RA8E2 - 480MHz Arm Cortex-M85 Based Entry-Line Graphics Microcontroller with Helium and TrustZone | 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.08.2025	—	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/.