

Renesas RA Family

Getting Started with GUIX Thermostat Application

Introduction

This 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 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 Renesas website. The intuitive configurators and code generators in e² 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

- e² studio IDE 2021-04 or greater
- Renesas Flexible Software Package (FSP) v3.1.0 or later
- Azure RTOS GUIX Studio V6.1.7.0 or later

Hardware

- Renesas EK-RA6M3G kit (RA6M3 MCU Group)
(<https://www.renesas.com/us/en/products/software-tools/boards-and-kits/eval-kits/ek-ra6m3g.html>)

Reference Manuals

- RA Flexible Software Package Documentation Release v3.1.0
- Azure RTOS GUIX and GUIX Studio v6.1.7.0
- Renesas RA6M3 Group User's Manual Rev.1.00
- EK-RA6M3G-v1.0 Schematics

Contents

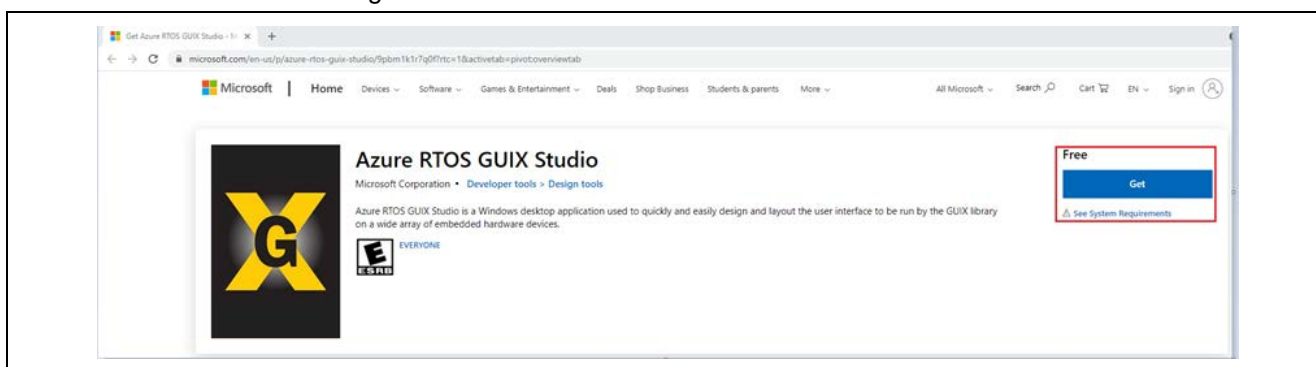
1. Installing Tools.....	2
2. Creating the Application Note Project	4
3. Using GUIX Widget Timer to Trigger A Screen Transition.....	19
4. Add Touch Driver to Thermostat_GUIX_EK_RA6M3G Project	20
5. Control LCD Backlight	27
6. Update Date/Time and Temperature	31
7. Setting Date/Time in A Full Function Project.....	34
Revision History	37

1. Installing Tools

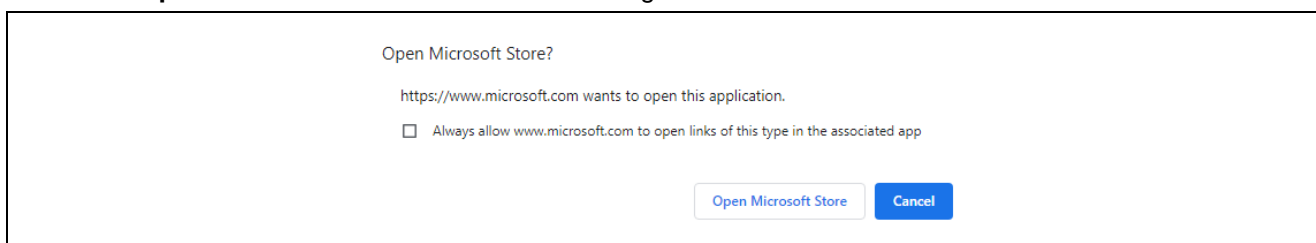
In this section you will copy the application note (AN) materials to your PC and install e² studio v2021-04/FSP v3.1.0 and Azure RTOS GUIX studio v6.1.7.0.

The steps are as follows:

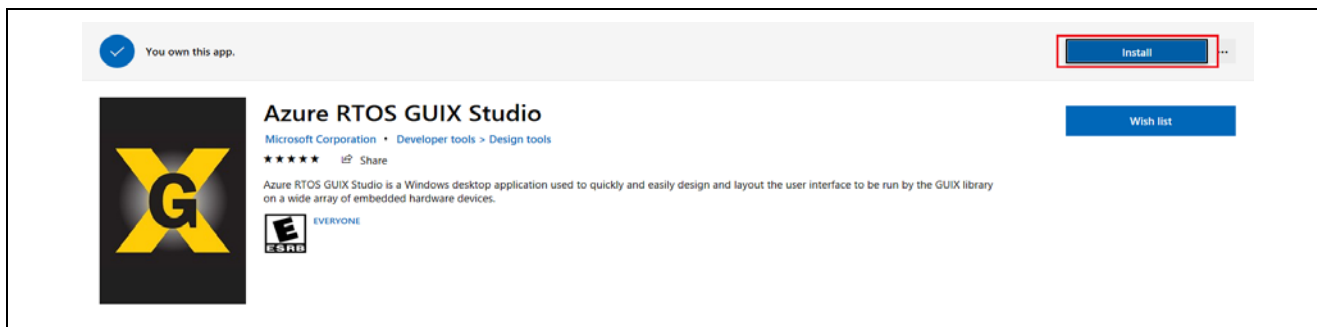
1. If you already have e² studio v2021-04 with FSP v3.1.0 or later installed, you can skip this step. Otherwise, you can download it from this [link](#).
2. You can get Azure RTOS GUIX Studio V6.1.7.0 or greater from this [link](#).
If it goes well, you will see the window in the next step on the web browser.
Note: It needs Microsoft Store working on your PC to install Azure RTOS GUIX studio.
3. Click **Get** to start installing Azure RTOS GUIX studio.



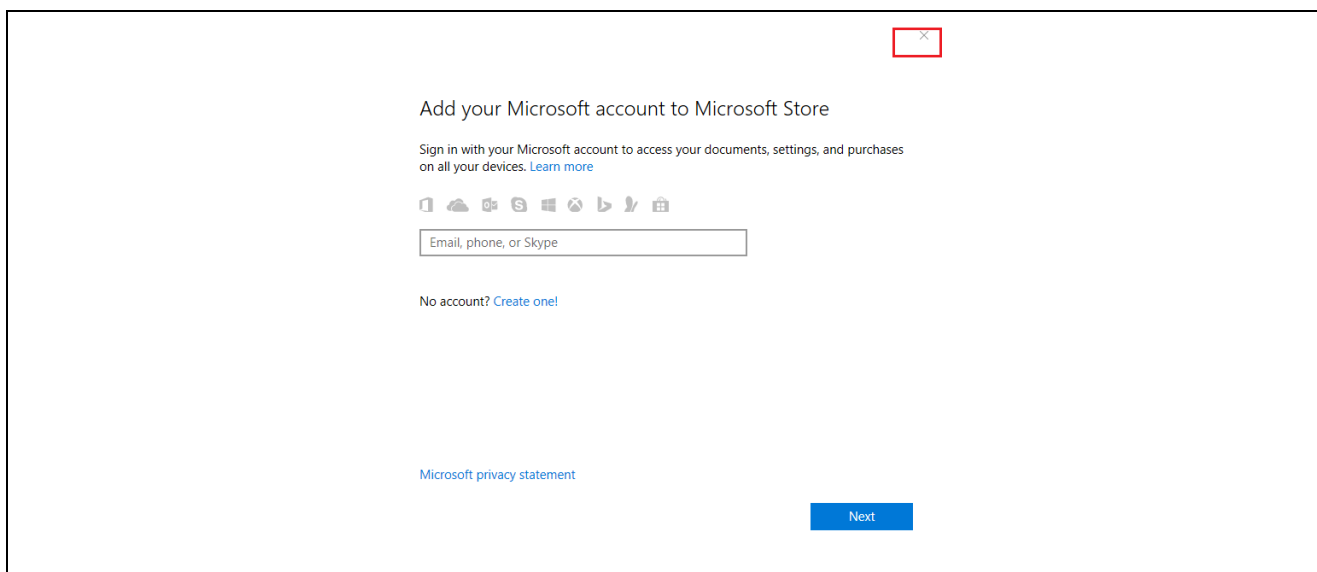
4. Click **Open Microsoft store** to continue installing Azure RTOS GUIX studio.



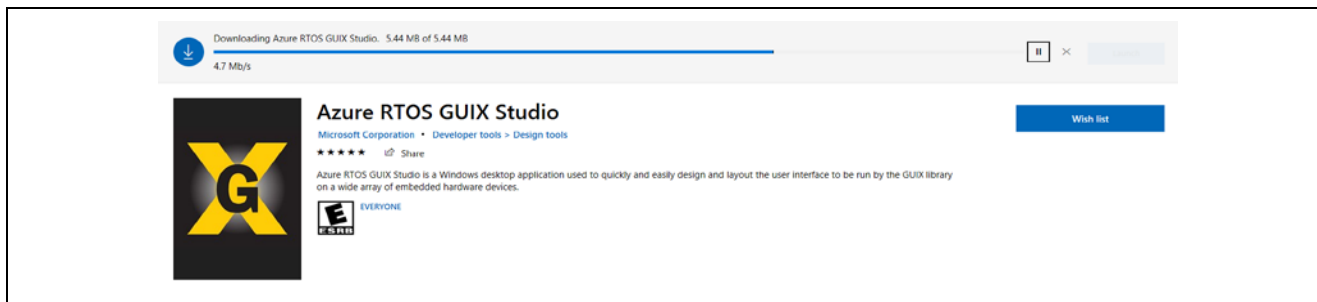
5. Click **Install** to continue. A window shows up to ask for a Microsoft account, which is seen in the next step.



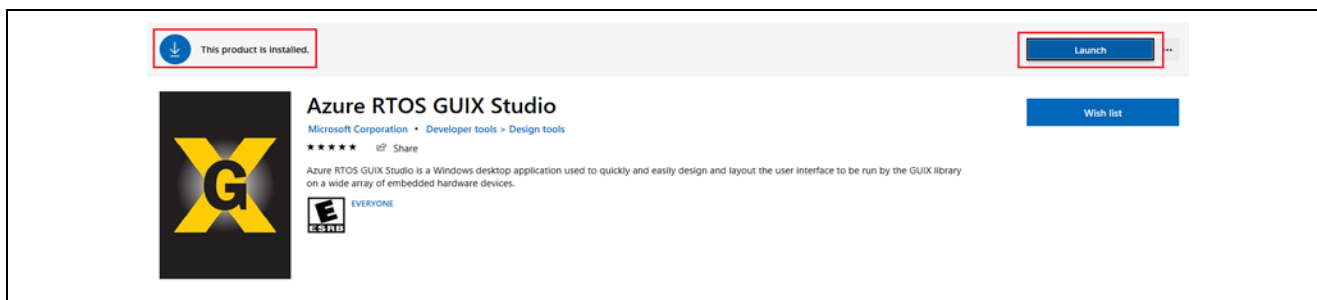
6. Ignore it by clicking **X** on the top-right to close this pop-up window and continue Azure RTOS GUIX studio installation.



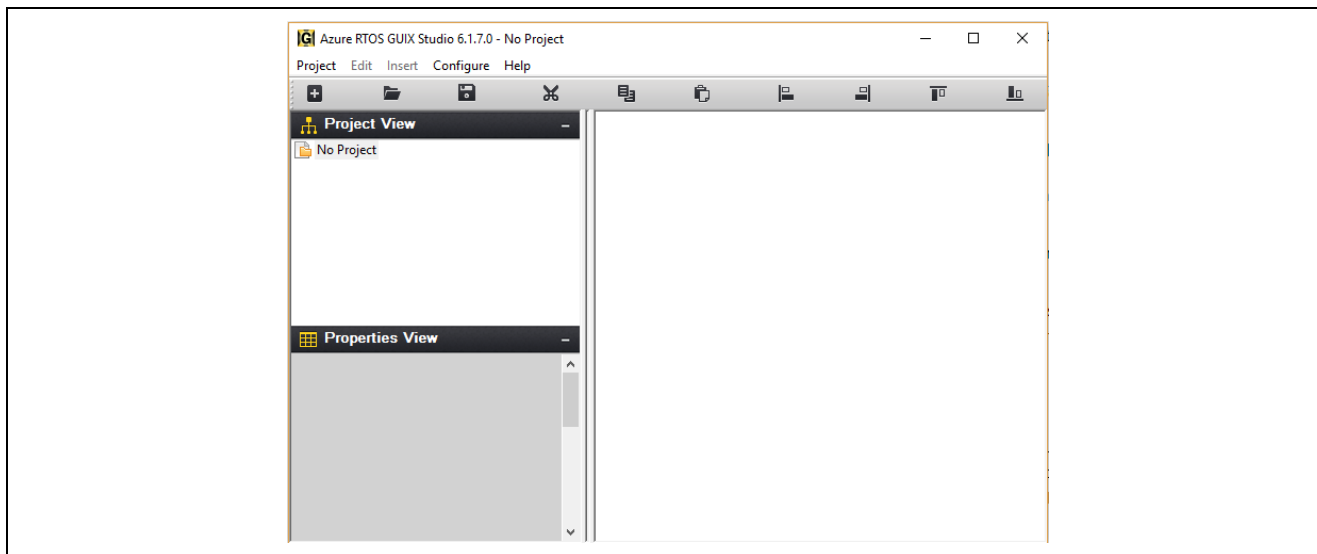
7. It starts downloading and installing Azure RTOS GUIX studio.



8. Click **Launch** to launch Azure RTOS GUI studio.



9. Azure RTOS GUIX studio 6.1.7.0 launched.



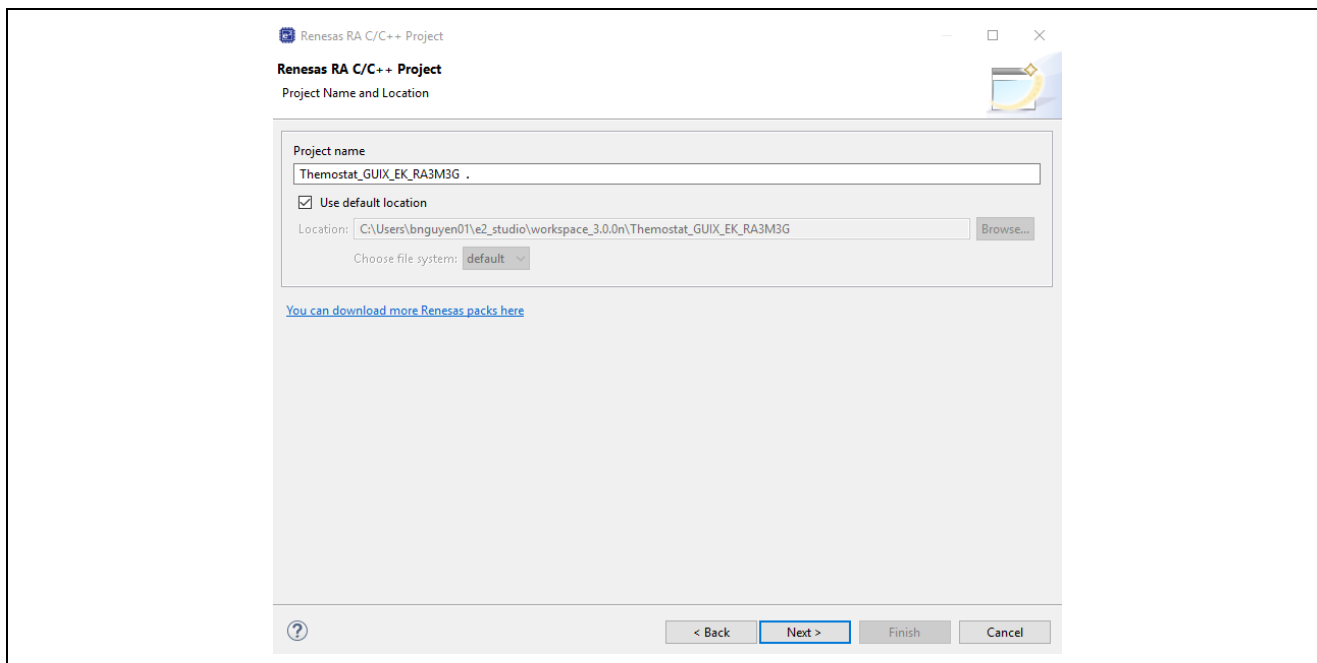
10. Close Azure RTOS GUIX studio, for now, you will open it again later in this lab.

2. Creating the Application Note Project

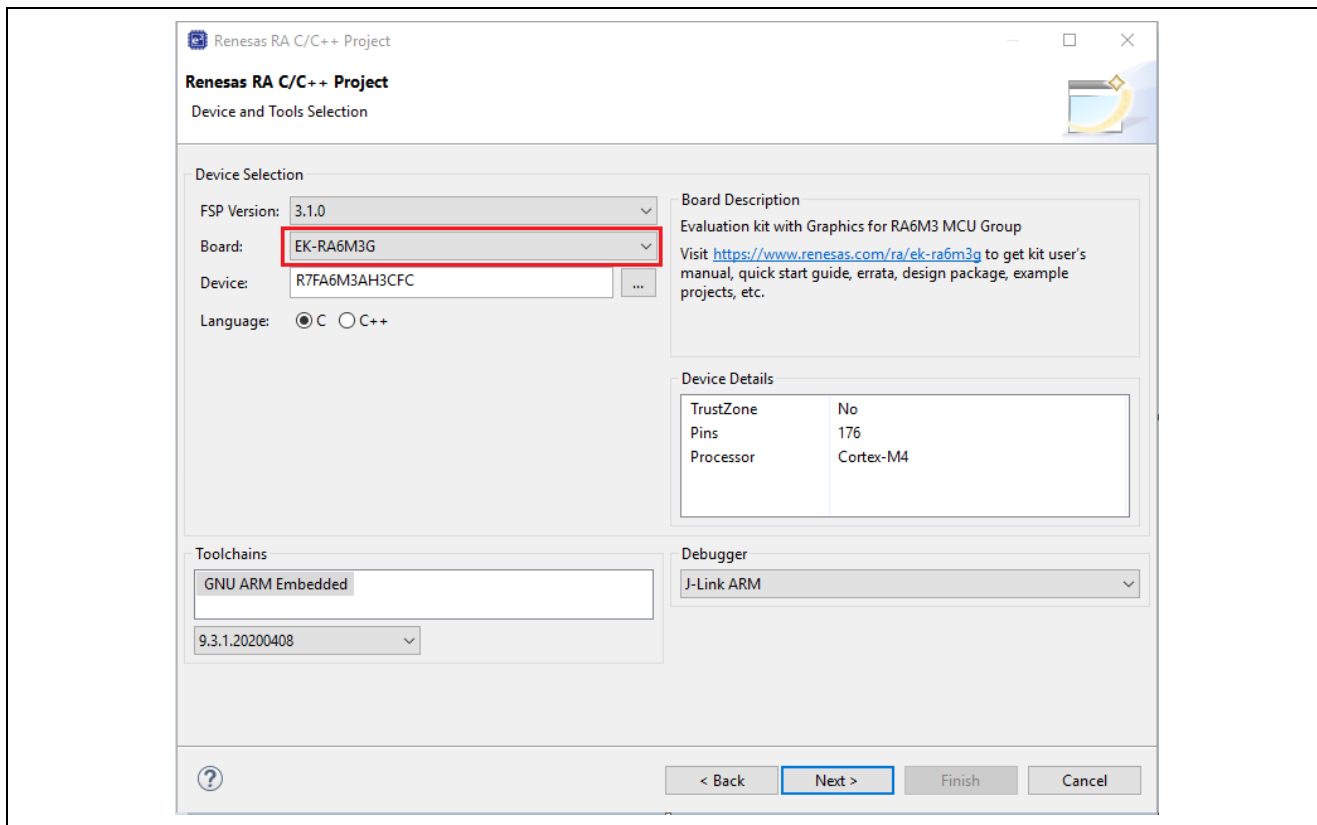
In this section, you will create a project to which you will add pre-written source code and integrate it with a pre-created Azure RTOS GUIX studio project.

The steps are as follows:

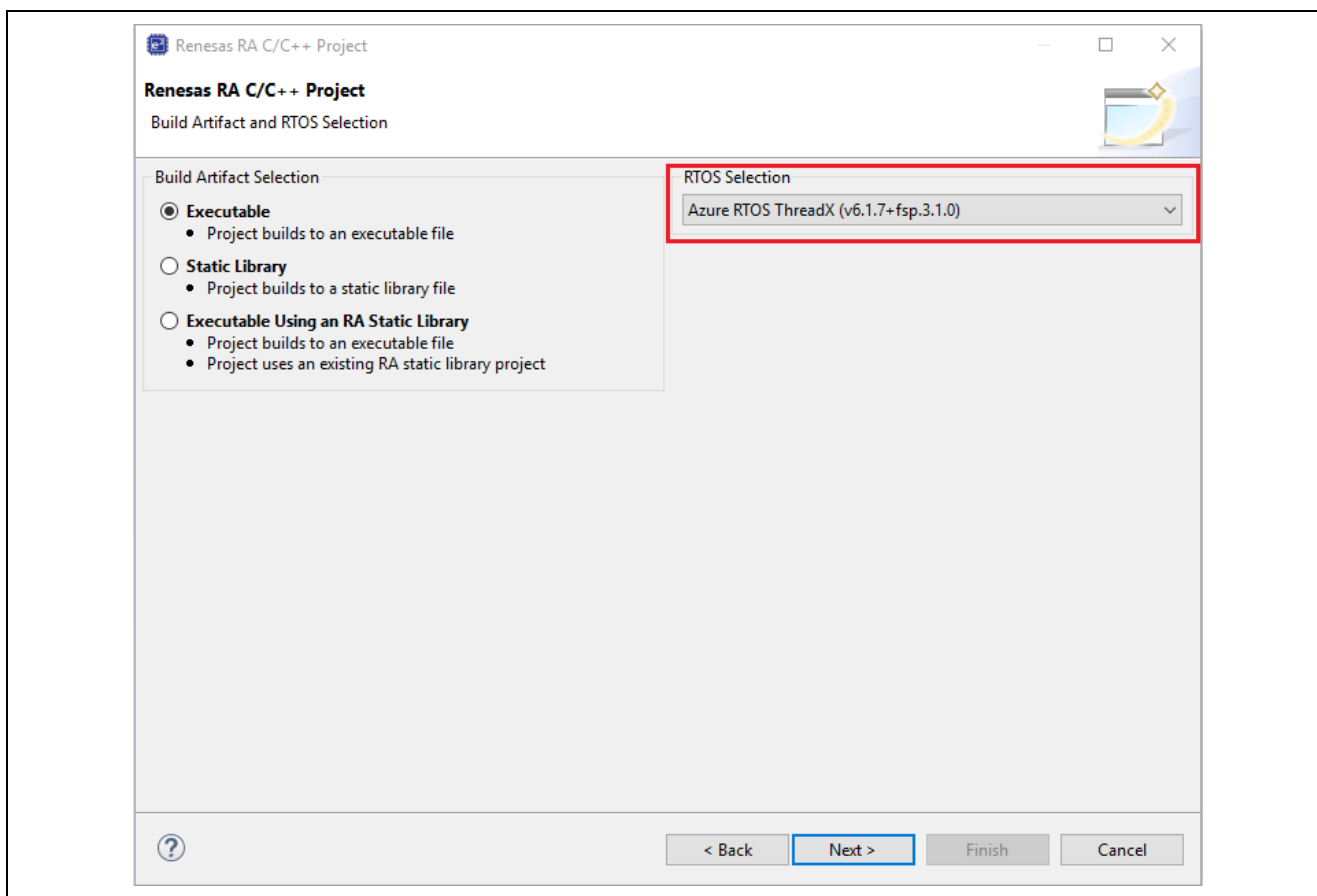
1. Create a new **RA C/C++ project**. Name it as “Thermostat_GUIX_EK_RA6M3G”.



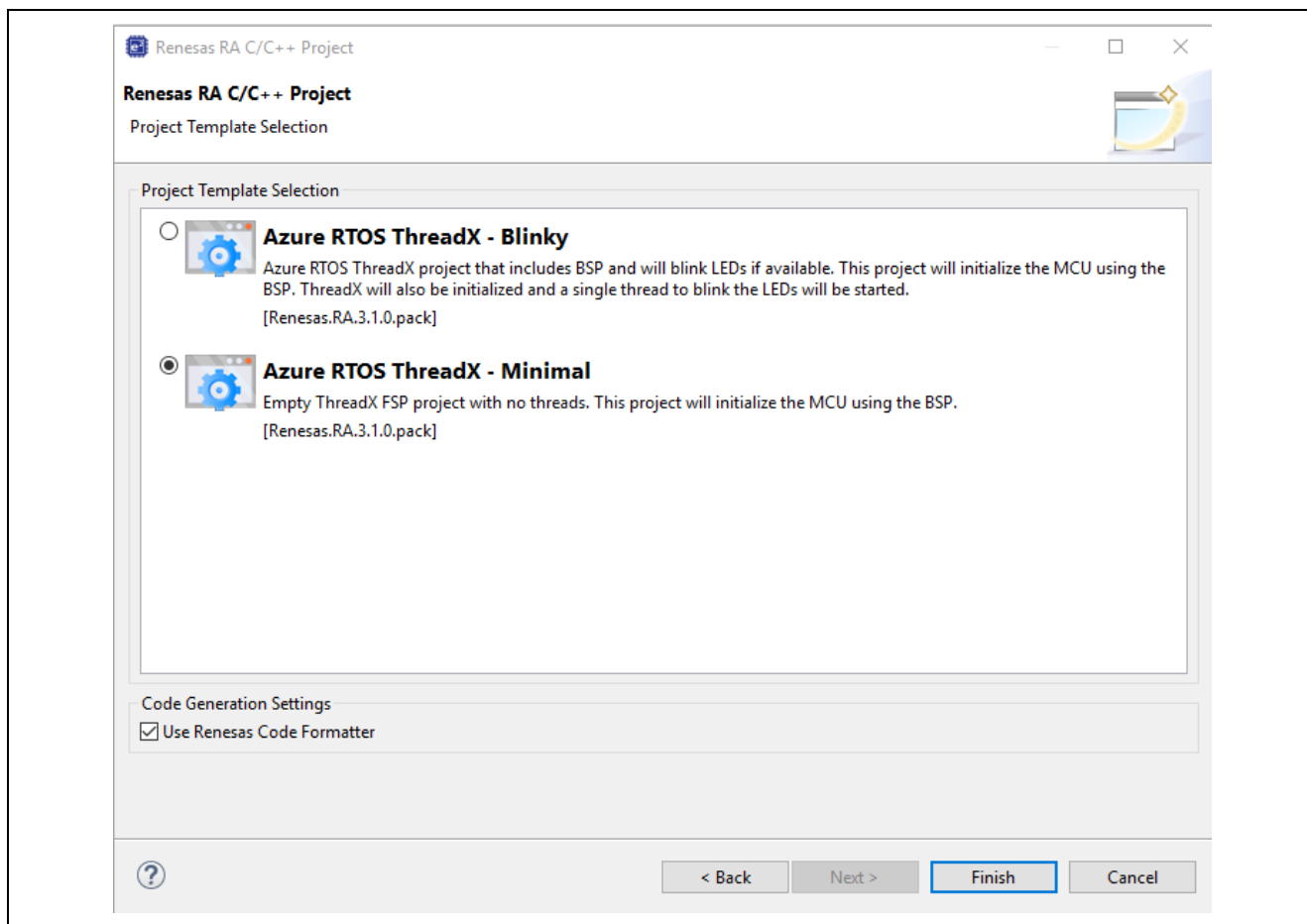
2. Set board to “EK-RA6M3”.



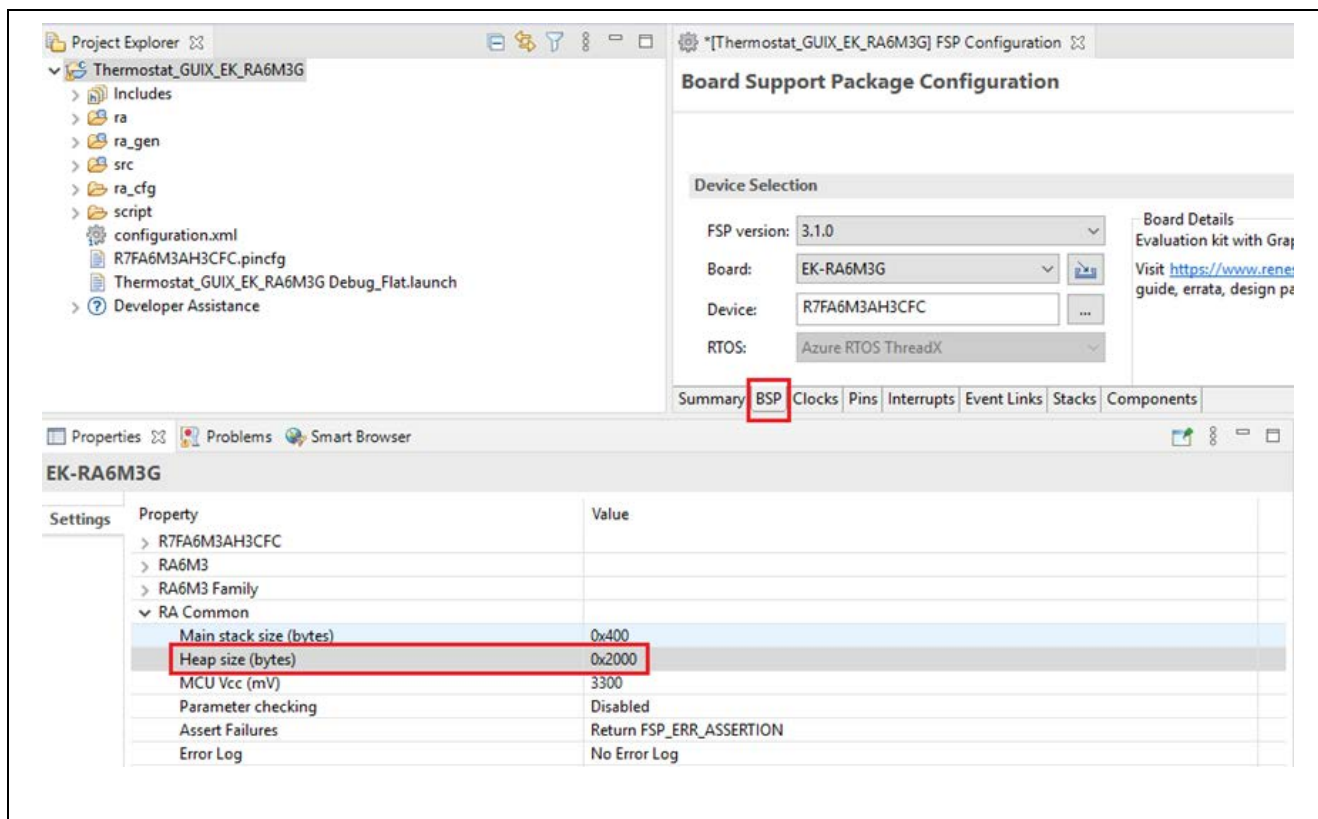
3. Select “Azure RTOS ThreadX (v6.1.7+FSP.3.1.0)”.



4. Use “Azure RTOS ThreadX-Minimal” template.



5. Open project configuration, go to **BSP** tab, change Heap size to “0x2000”.



6. Add a **new thread** and name it as **System Thread** with the following settings.

The screenshot shows the 'Stacks Configuration' window for the '[Thermostat_GUIX_EK_RA6M3G] FSP Configuration'. The 'Threads' section on the left has a 'New Thread' button highlighted with a red box. Below it, a tree view shows 'HAL/Common' expanded, with 'g_ioport I/O Port Driver on r_ioport' and 'System Thread' listed. 'System Thread' is also highlighted with a red box. The 'System Thread Stacks' section on the right has a 'New Stack >' button. Below this, an information icon and text state: 'Add stacks to the selected thread by using the 'New Stack >' toolbar button (above), clipboard.' The 'Objects' section at the bottom has a 'New Object >' button. The 'System Thread' settings table is shown below:

Property	Value
Common	
> General	
> Timer	
> Trace	
> Performance	
> RA	
> Interrupts	
Thread	
Symbol	system_thread
Name	System Thread
Stack size (bytes)	1024
Priority	3
Auto start	Enabled
Time slicing interval (ticks)	10

7. Add **Azure RTOS GUIX** to **system thread**.

The screenshot shows the 'Stacks Configuration' window with the 'System Thread' selected in the 'Threads' list. The 'New Stack >' button in the 'System Thread Stacks' section is highlighted with a red box. A context menu is open, showing the following options: 'Arm', 'Azure RTOS', 'Bootloader', 'Driver', 'Middleware', 'SEGGER', and 'Search...'. The 'Azure RTOS' option is expanded, showing a sub-menu with 'FileX', 'NetX Duo', 'USBX', and 'GUIX'. The 'GUIX' option is highlighted with a red box.

8. Don't need to make any changes in GUIX and r_glcdc driver configurations. FSP already has them configured properly for EK-RA6M3G.

*[Thermostat_GUIX_EK_RA6M3G] FSP Configuration

Stacks Configuration

Threads

- HAL/Common
 - g_ioport I/O Port Driver on r_ioport
 - System Thread
 - GUIX

Objects

New Object > Remove

System Thread Stacks

New Stack >

9. In Pin Configuration, change P603's mode to **output mode (initial high)** to enable LCD panel backlight.

Select Pin Configuration

RA6M3G-EK.pincfg [Manage configurations...](#) ☒ Generate data: g_bsp_pin_cfg

Pin Selection

Type filter text


- ▼ P6
 - ✓ P600
 - ✓ P601
 - ✓ P602
 - ✓ P603
 - P604
 - P605
 - P606
 - P607
 - ✓ P608
 - ✓ P609
 - ✓ P610
 - ✓ P611


Pin Configuration

Name	Value	Link
Symbolic Name		
Comment		
Mode	Output mode (Initial High)	
Pull up	None	
Drive Capacity	Low	
Output type	CMOS	
Input/Output		
P603	✓ GPIO	

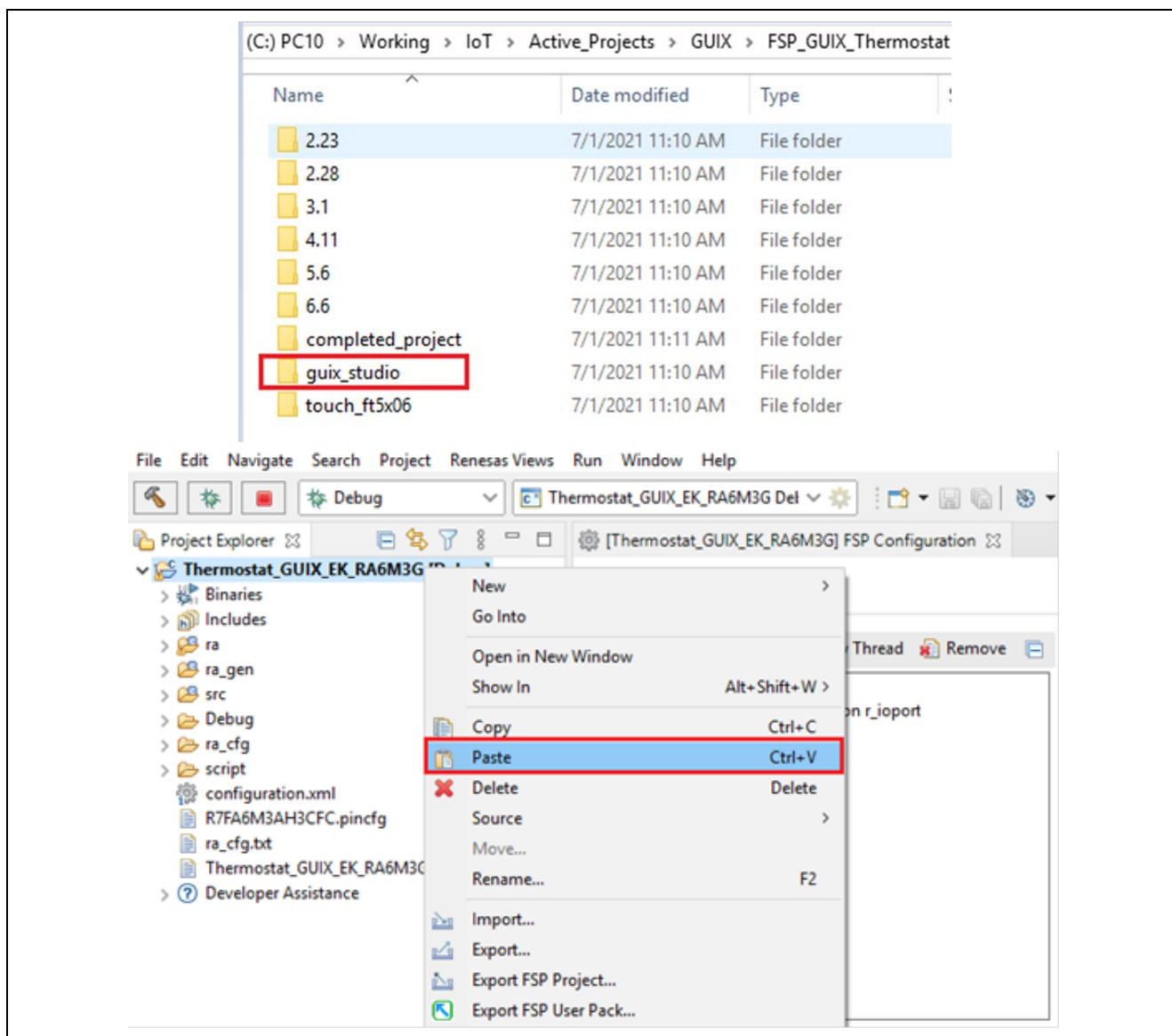
Module name: P603
Port Capabilities: BUS0: D13_DQ13
GPT7: GTIOCA

Pin Function | Pin Number

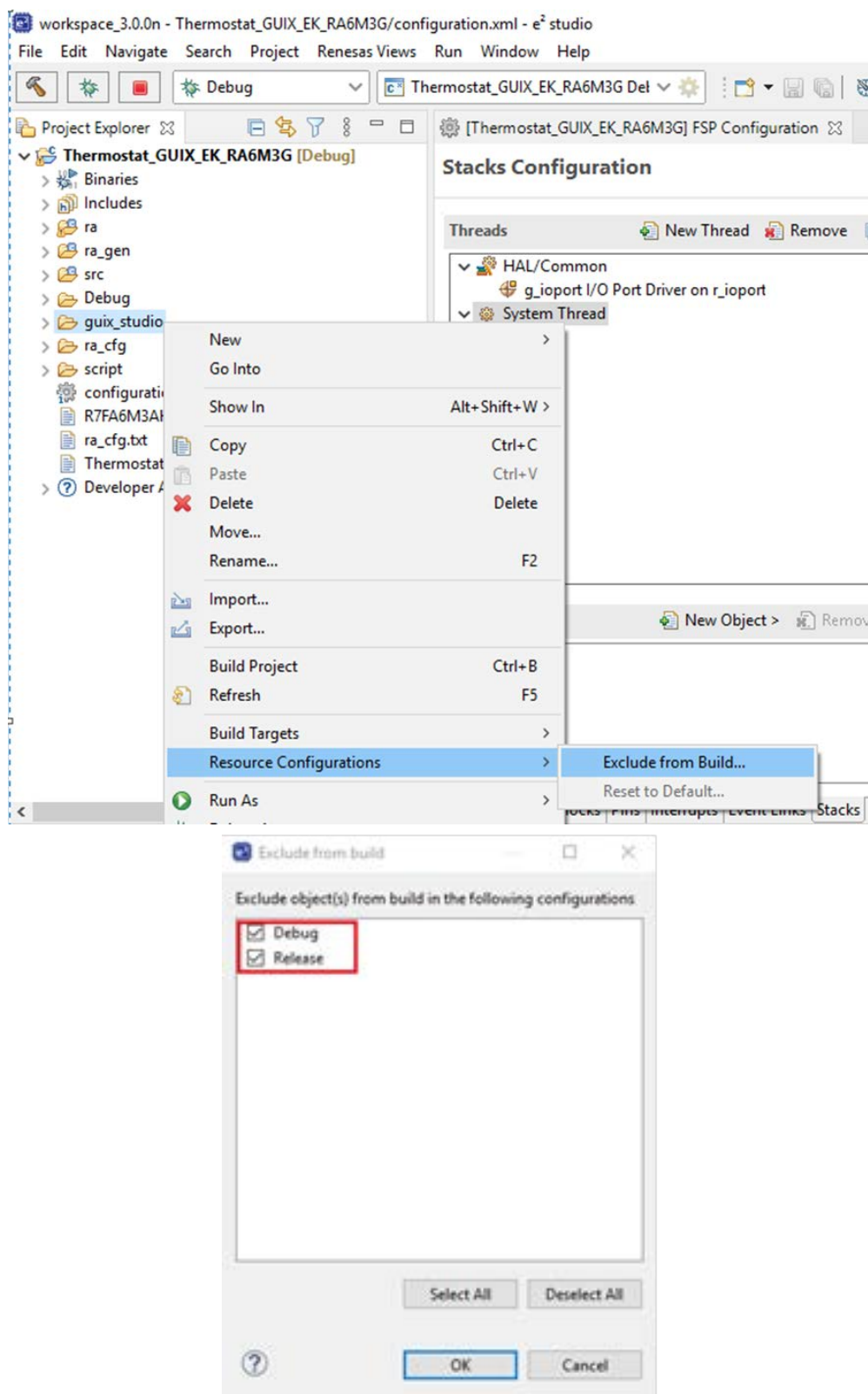
10. In RA Configurator, click  **Generate Project Content** to generate project content.

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.

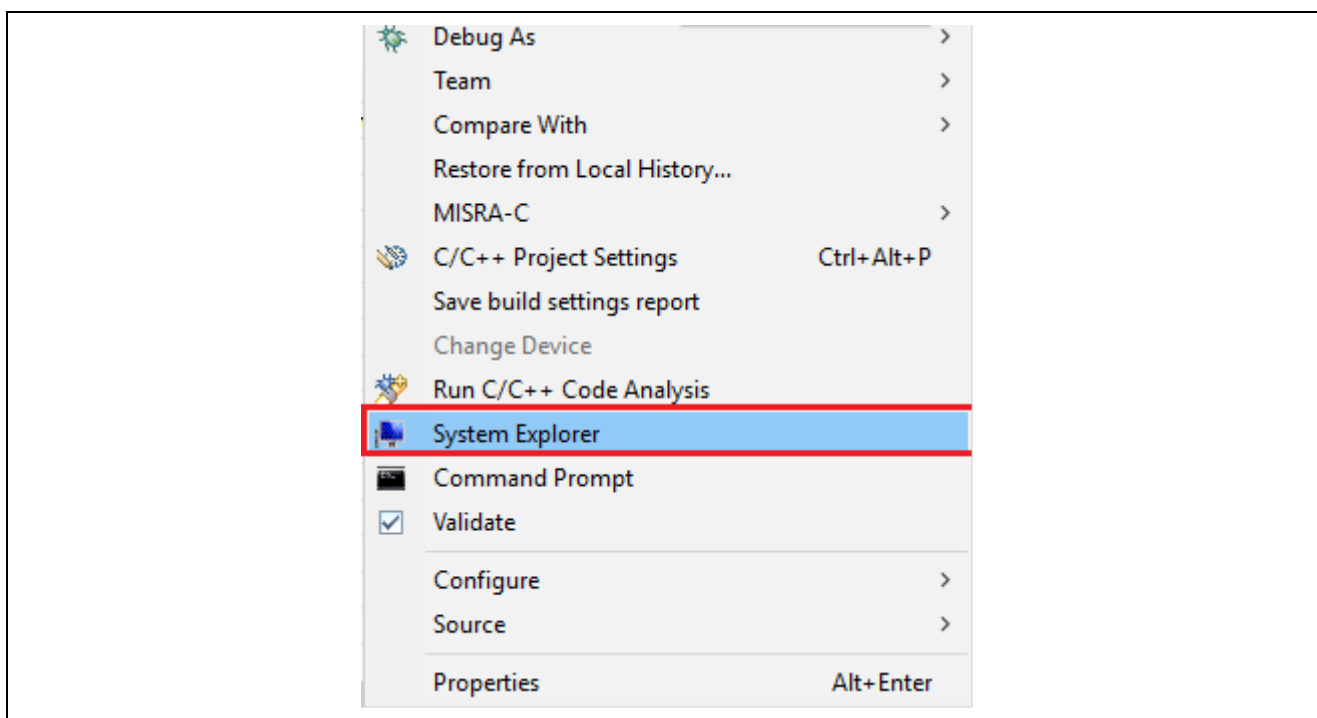
11. Copy Azure RTOS GUIX studio project to e² studio project (Thermostat_GUIX_EK_RA6M3G) by copying **guix_studio** folder in the application note (**AN**) folder (FSP_GUIX_Thermostat) and then at Thermostat_GUIX_EK_RA6M3G project and paste it.



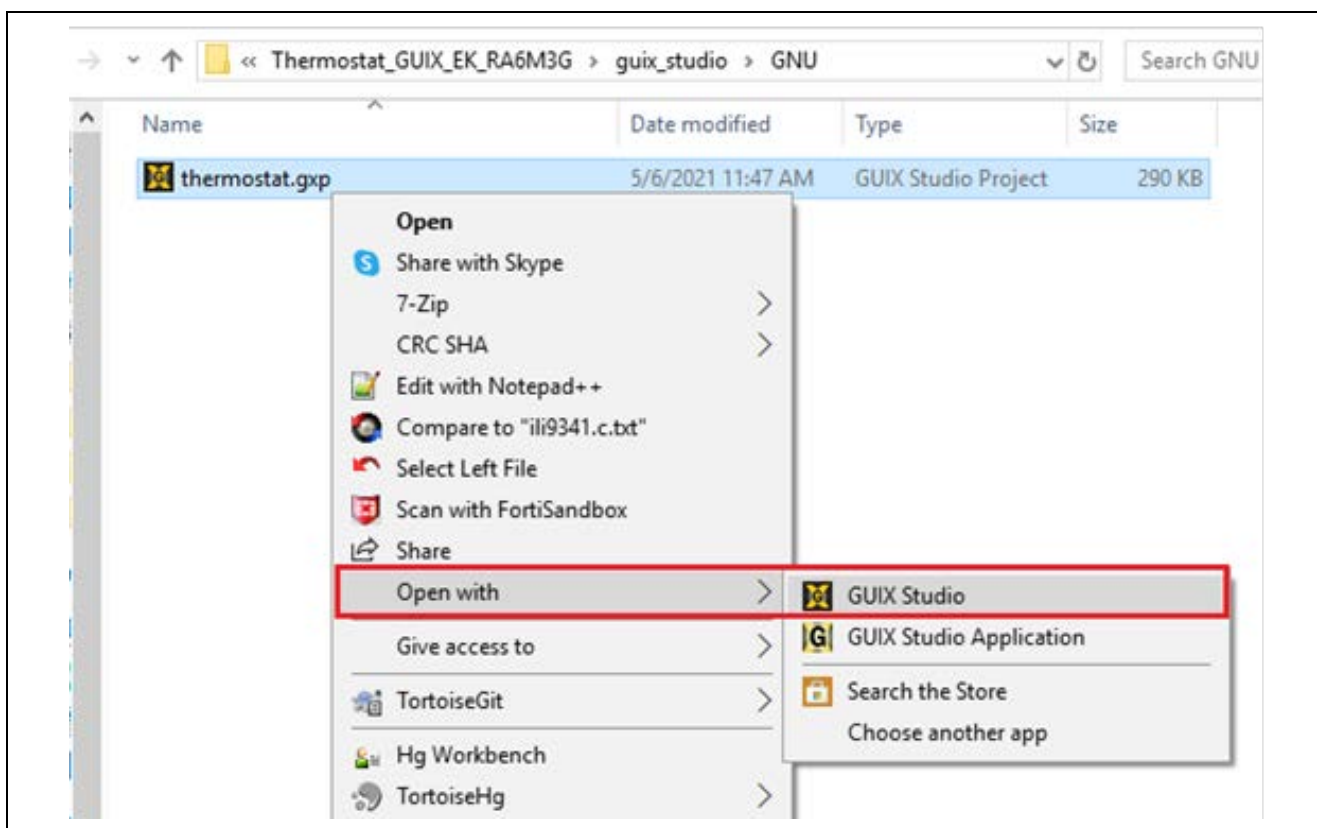
12. GUIX Studio project is now in Thermostat_GUIX_EK_RA6M3G project. In e² studio, right-click the **guix_studio** folder and exclude it from build since it contains the Azure GUIX Studio project, which won't be built by FSP.



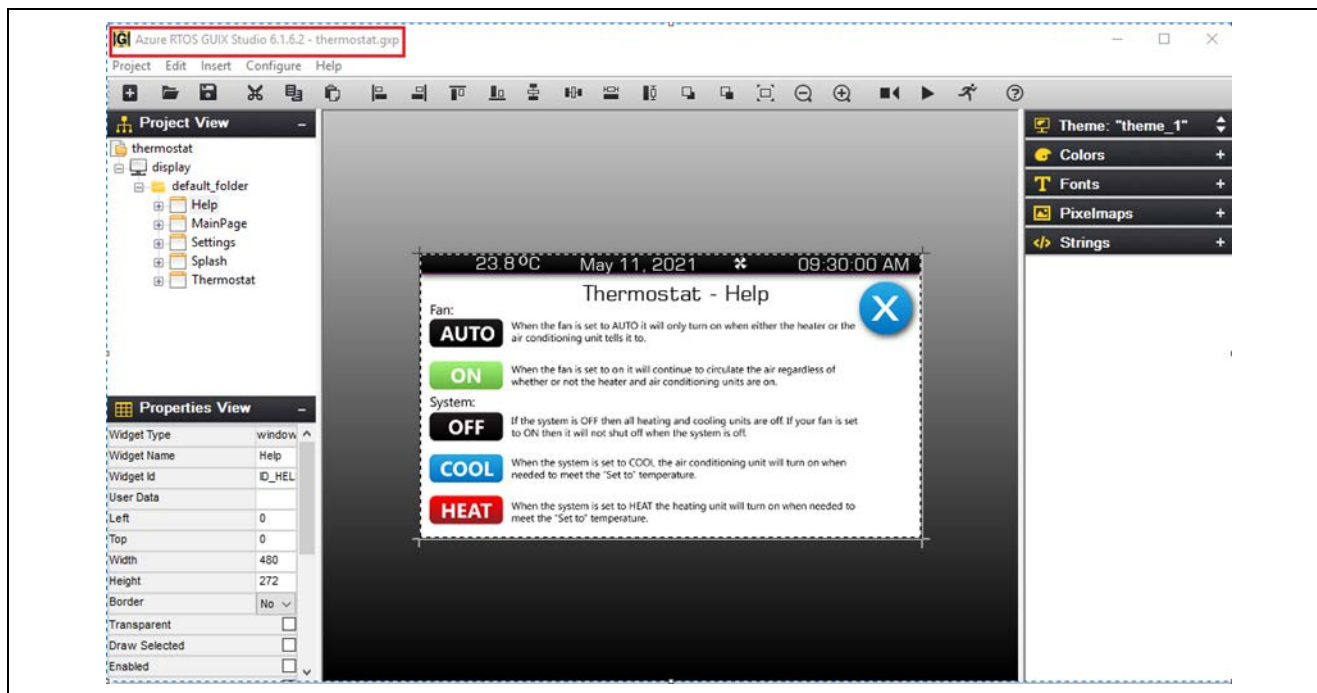
13. Get to Thermostat_GUIX_EK_RA6M3G project folder by right click the e² studio project and select **System Explorer** as shown below.



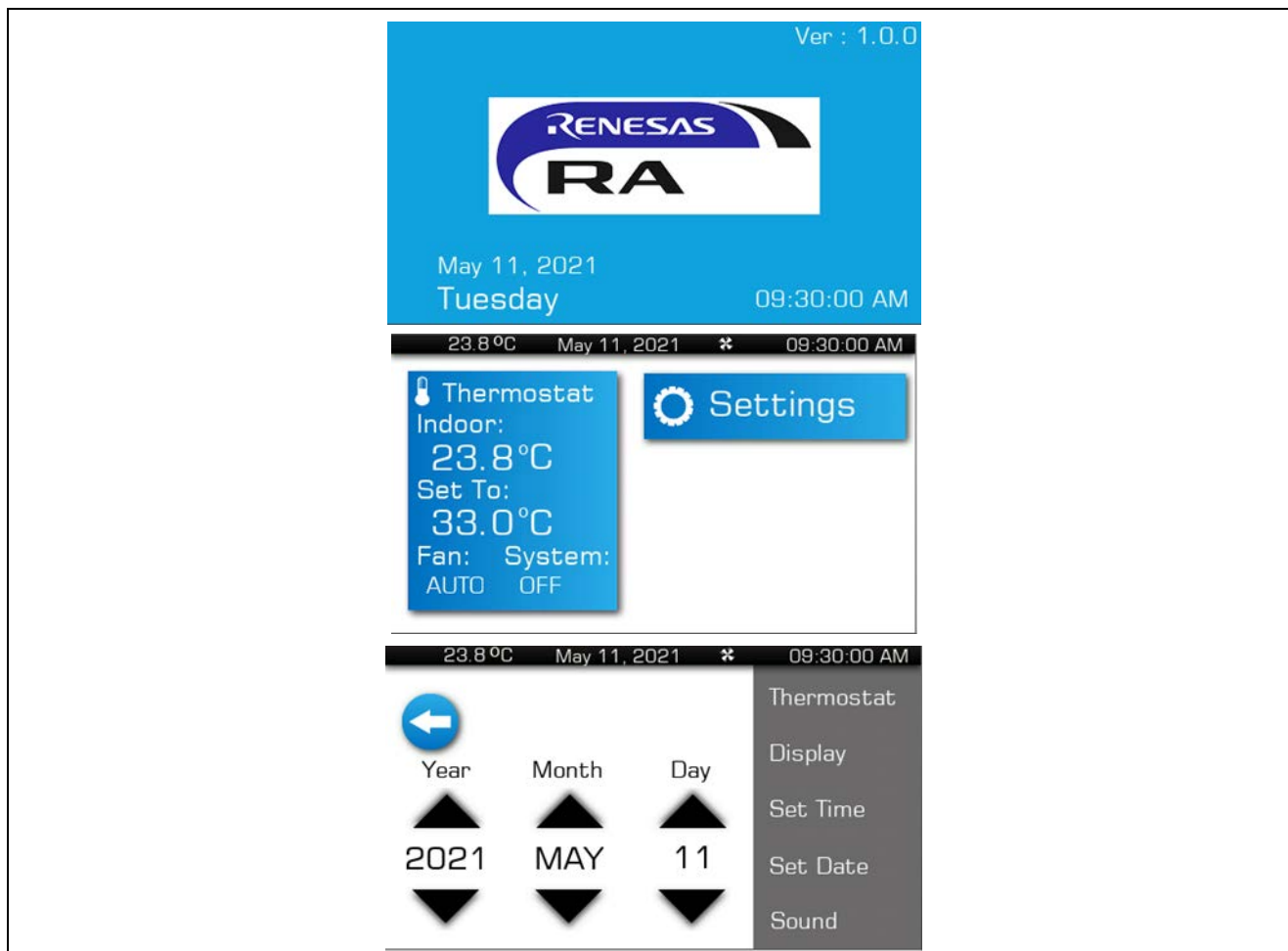
14. Open **thermostat.gpx** project file in **guix_studio>GNU** sub-folder in your Thermostat_GUIX_EK_RA6M3G folder. If you have several GUIX studio versions in your system, make sure you choose the right one, which is **v6.1.7.0 or later**.



15. 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 e² studio project.

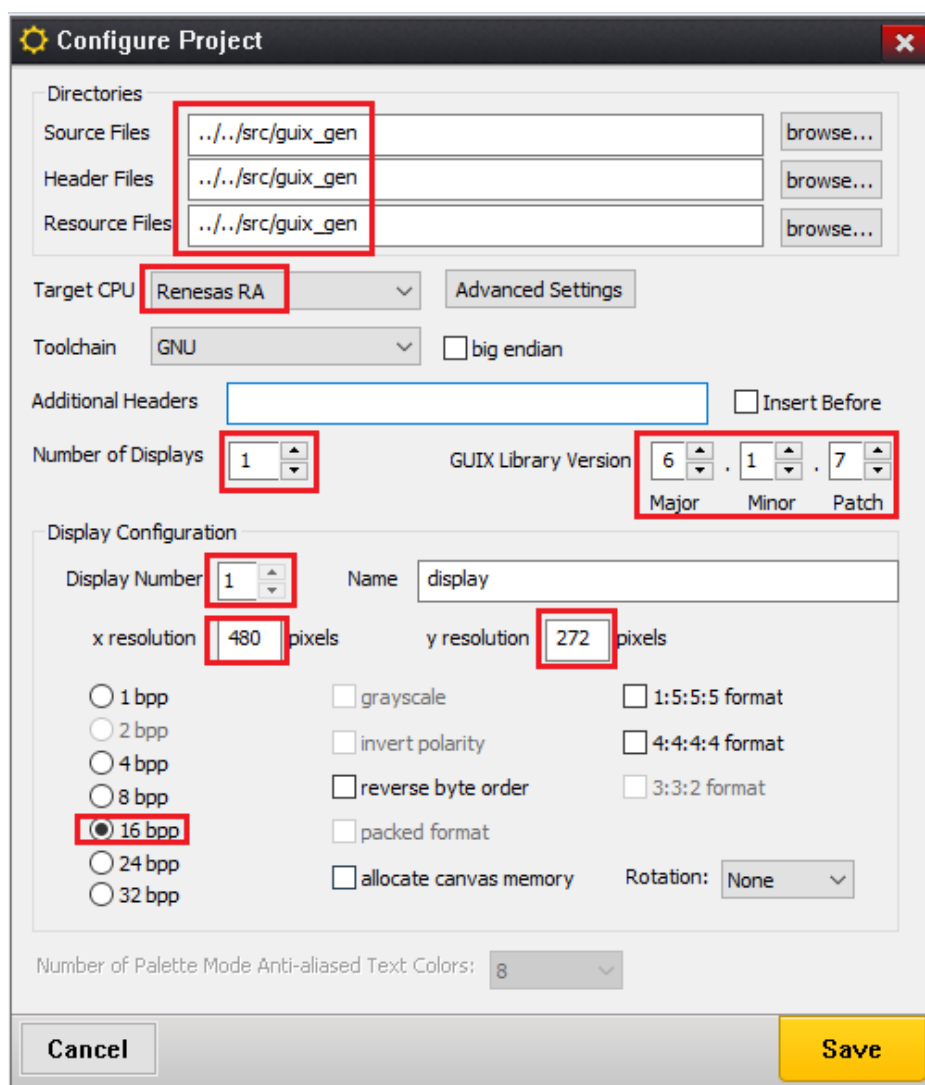


16. The Azure RTOS GUIX studio project consists of 5 screens, including Splash, MainPage, Settings, Thermostat and Help from top to bottom:

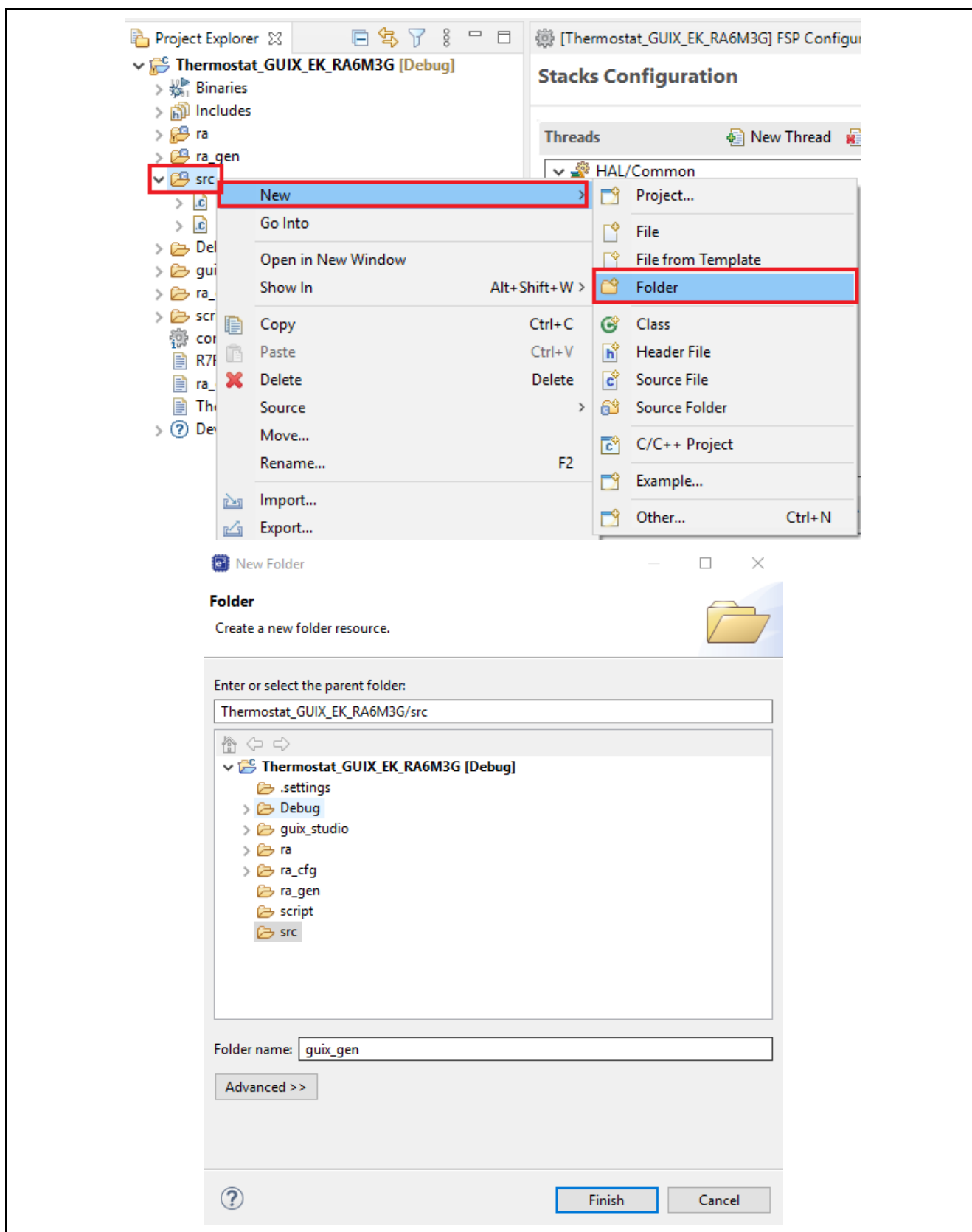




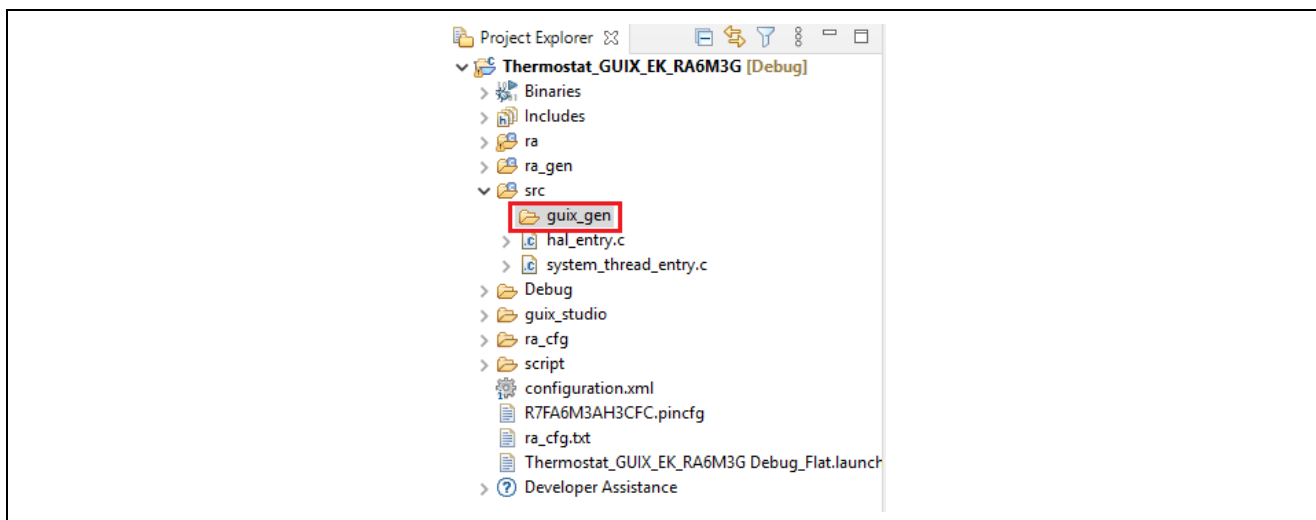
17. Click **Configure > Project/Display** and confirm the following settings.



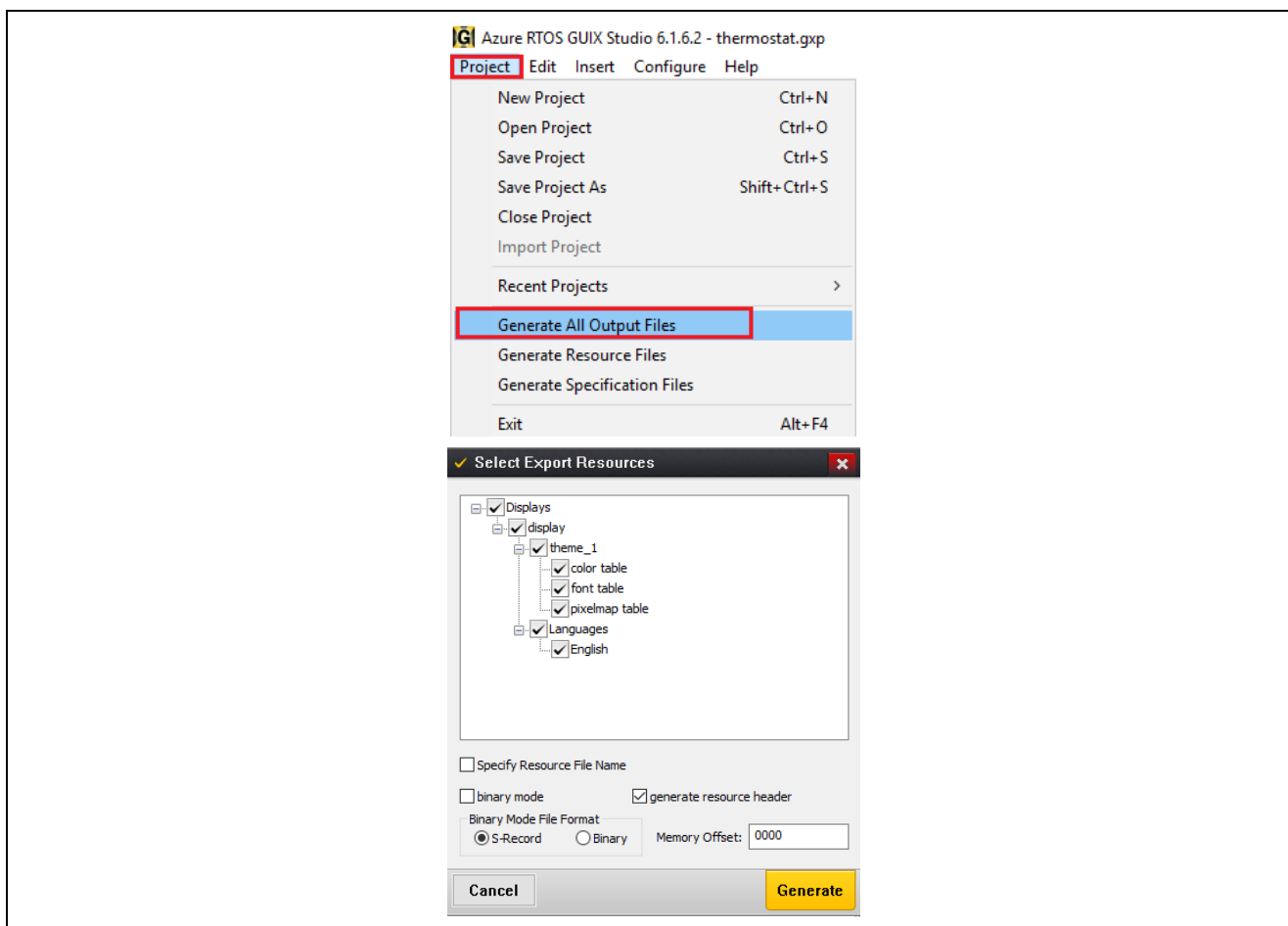
18. Go back e² studio project (Thermostat_GUIX_EK_RA6M3G), right click **src**, then select **New > Folder** and create a folder named **guix_gen**.



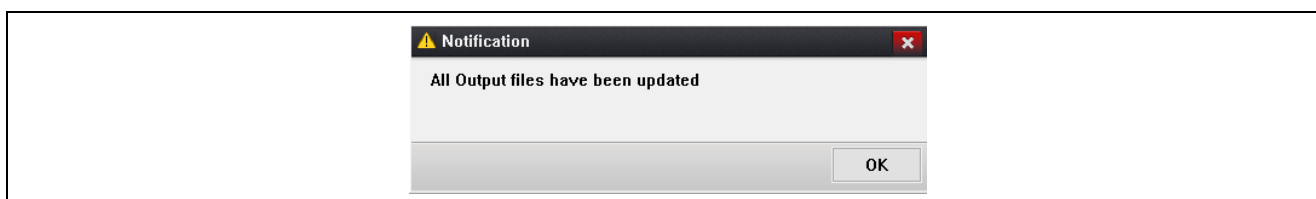
19. Confirm **guix_gen** is created before moving to next step.



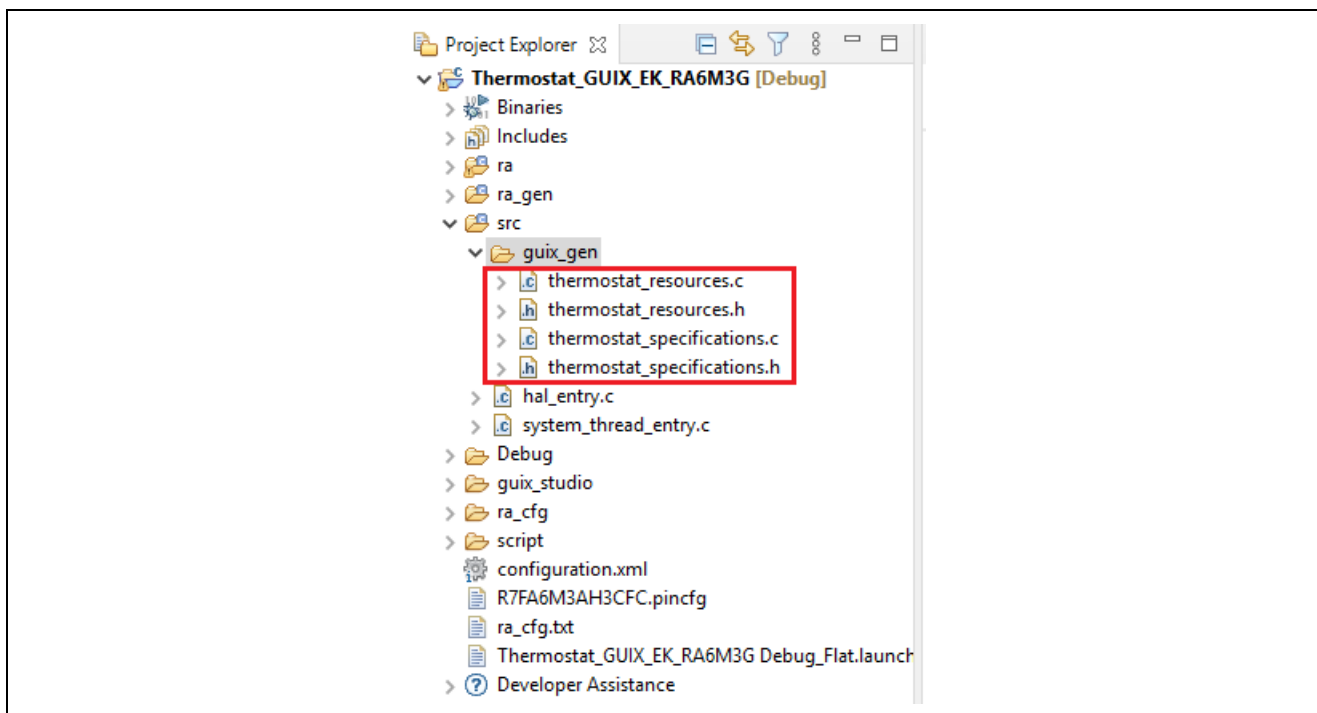
20. In Azure RTOS GUIX studio, click **Project > Generate All Output Files** to generate resource files, header files and source files of this GUIX design.



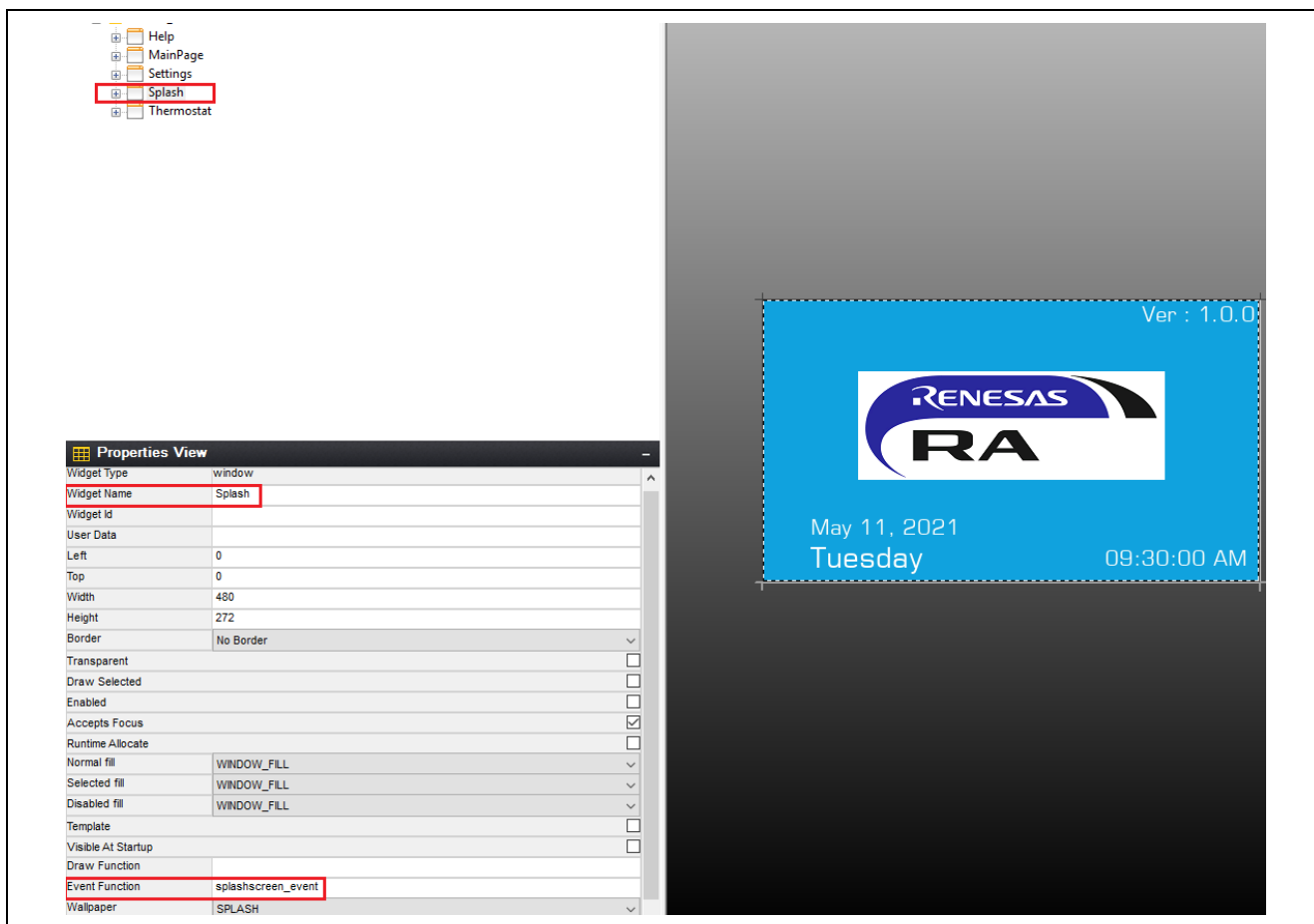
Click **Generate** to generate all output files. If succeeded, you will see below notification.



21. All output files are now in **guix_gen** folder.



22. 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 **e² studio/FSP project**. The other windows have similar definitions.



23. Copy and replace the files in **src** folder in e² studio project with the files in 2.23 folder in the AN folder:

hmi_event_handler.c

system_thread_entry.c

Build Thermostat_GUIX_EK_RA6M3G project, you will see several warnings but we will address them in later steps.

24. **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 a screen and attached it 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);
}
```

25. **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;
}
```

26. Get your EK-RA6M3G ready to run the project. Connect LCD board to **Graphics Expansion** connector on EK-RA6M3 as shown below.

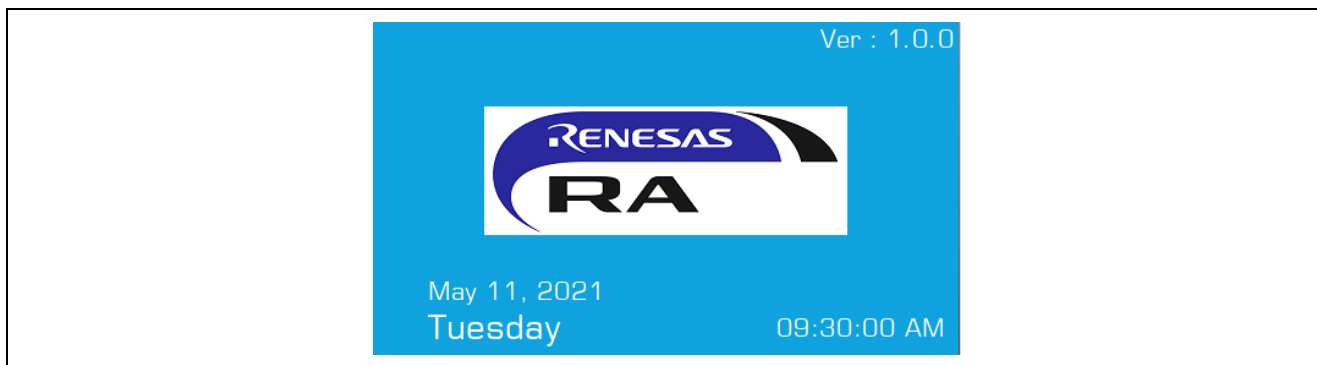


27. Connect EK-RA6M3G kit to your PC using **J10**. If you **Download and Run** Thermostat_GUIX_EK_RA6M3G project, you will see a black screen.
28. Add below code to **splashscreen_event** function in **hmi_event_handler.c** to show Splash screen.
Build the e² studio project.

```
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;
}
```

Please refer to splashscreen_event function in hmi_event_handler.c in **2.28** folder in the AN folder.

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



3. Using GUIX Widget Timer to Trigger A Screen Transition

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

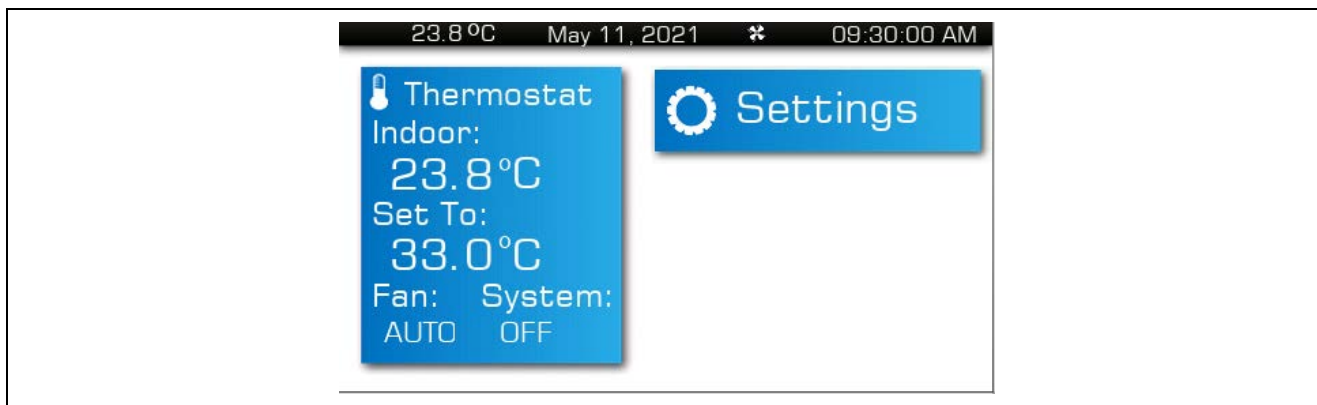
The steps are as follows:

1. Copy and replace the files in **src** folder in e² studio project with the files in **3.1** folder in the AN folder:
hmi_event_handler.c
system_thread_entry.c

2. **Code highlight:** The following code in splashscreen_event function starts a GUIX Widget timer and trigger a screen transition that hides Splash screen and shows MainPage 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;
}
```

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

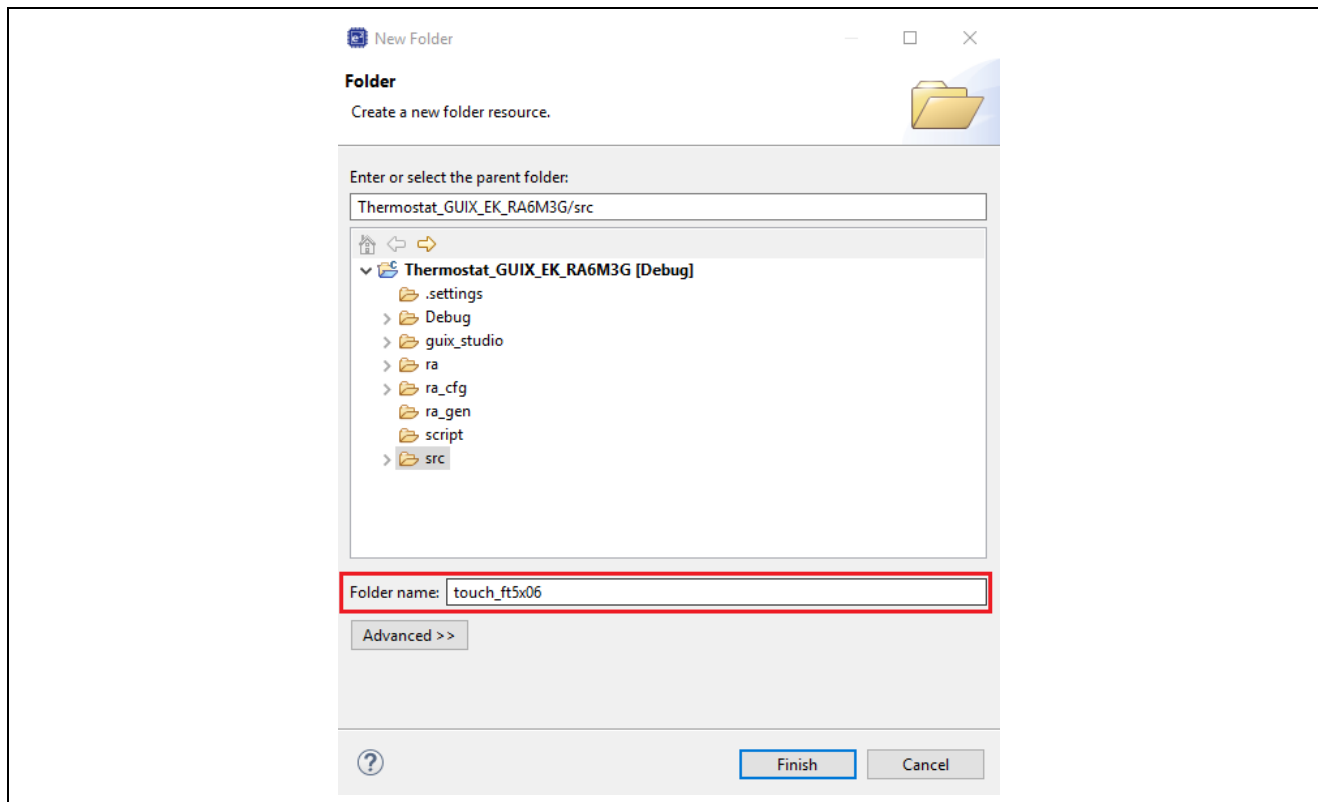


4. Add Touch Driver to Thermostat_GUIX_EK_RA6M3G Project

In this section, you will add the ft5x06 touch driver to the project to handle touch events on LCD panel.

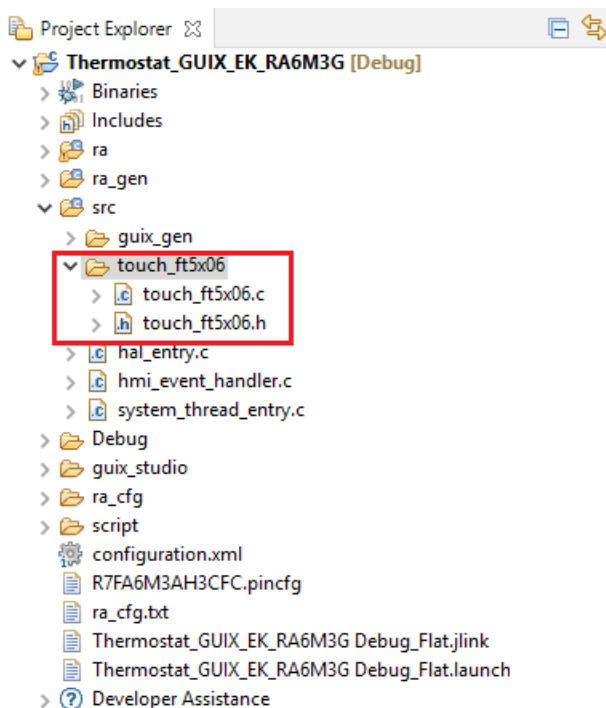
The steps are as follows:

1. In Thermostat_GUIX_EK_RA6M3G project, create a folder by right-clicking **src**, then select **New->Folder**.



Click **Finish** to create **touch_ft5x06** folder

2. Copy touch_ft5x06.c and touch_ft5x06.h from **touch_ft5x06** folder in the Lab folder to the one in e² studio project.

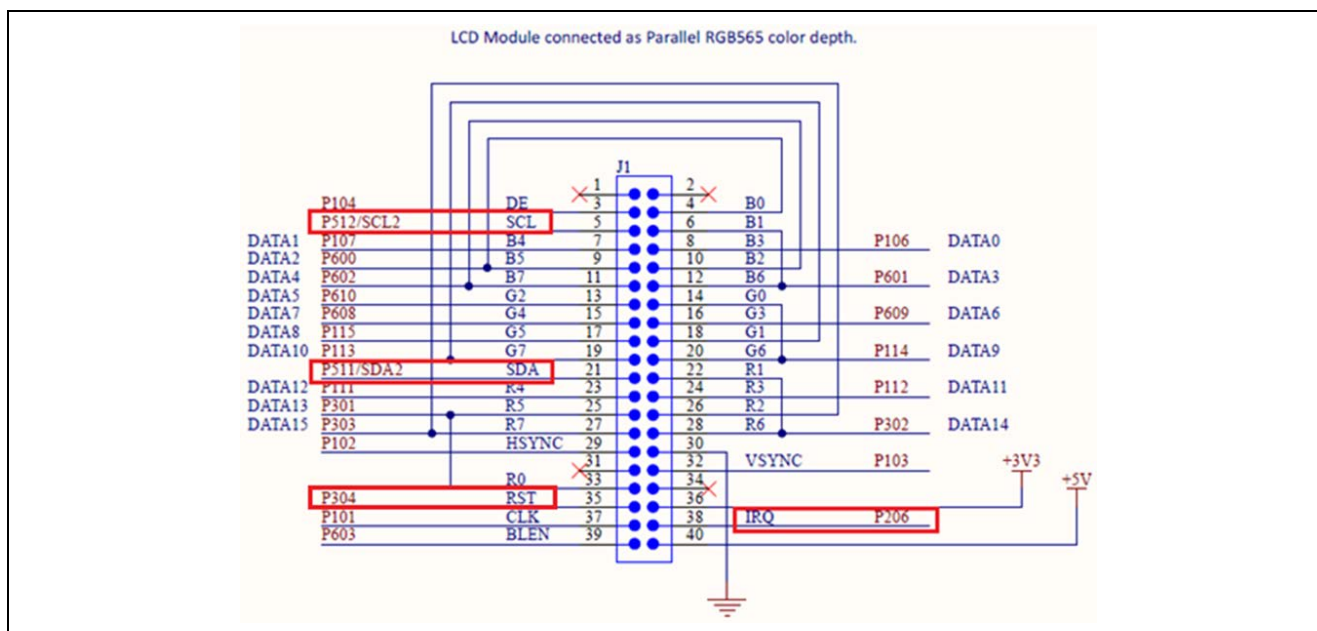


3. Open project configuration and create **Touch Thread** with below settings.

Property	Value
Common	
General	
Timer	
Trace	
Performance	
RA	
Interrupts	
Thread	
Symbol	touch_thread
Name	Touch Thread
Stack size (bytes)	1024
Priority	4
Auto start	Enabled
Time slicing interval (ticks)	10

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

- IRQ0 interrupt (P206) is used to trigger touch events.
- I2C channel 2 (P512, P511) is used to read and write data to the touch controller.
- P304 is used to reset the touch controller.



5. Since the IRQ0 (P206) needs a pull-up to function properly with the LCD board, don't change the setting that was done by FSP, as shown below.

Pin Configuration

Select Pin Configuration: RA6M3G-EK.pincfg [Export to CSV file](#) [Configure Pin Driver Warnings](#)

Manage configurations... ☒ Generate data: g_bsp_pin_cfg

Pin Selection

Type filter text

- ✓ P2
- ✓ P200
- ✓ P201
- ✓ P202
- ✓ P203
- ✓ P204
- ✓ P205
- ✓ P206
- ✓ P207
- ✓ P208
- ✓ P209
- ✓ P210
- ✓ P211
- ✓ P212
- ✓ P213

Pin Configuration

Name	Value	Link
Symbolic Name		
Comment		
Mode	Input mode	
Pull up	input pull-up	
IRQ	IRQ0-DS	
Drive Capacity	Low	
Output type	CMOS	
Input/Output	✓ GPIO	

Module name: P206
Port Capabilities: BUS0: WAIT
CTS0: TS01

Pin Function Pin Number

Summary BSP Clocks Pins Interrupts Event Links Stacks Components

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

Stacks Configuration

Threads [New Thread](#) [Remove](#)

- ✓ HAL/Common
 - g_ioport I/O Port Driver on r_ioport
- ✓ System Thread
 - GUIX
 - ✓ **Touch Thread**
 - g_touch_irq External IRQ Driver on r_icu

Objects [New Object >](#) [Remove](#)

Touch Thread Stacks

- g_touch_irq External IRQ Driver on r_icu

Summary BSP Clocks Pins Interrupts Event Links Stacks Components

Problems Console Properties Smart Browser Smart Manual Debug

g_external_irq0 External IRQ Driver on r_icu

Settings	Property	Value
API Info	Common	
	Parameter Checking	Default (BSP)
	Module g_external_irq0 External IRQ Driver on r_icu	
	Name	g_touch_irq
	Channel	0
	Trigger	Falling
	Digital Filtering	Enabled
	Digital Filtering Sample Clock (Only valid when Digital Filter	PCLK / 64
	Callback	touch_irq_cb
	Pin Interrupt Priority	Priority 5
Pins		
IRQ00	None	

7. In project configuration, add **I2C Master Driver on r_iic_master** to **Touch Thread** with below settings.

The screenshot shows the 'Stacks Configuration' window. In the 'Threads' list, 'Touch Thread' is selected and highlighted with a red box. In the 'Touch Thread Stacks' list, 'g_i2c_touch I2C Master Driver on r_iic_master' is added and highlighted with a red box. Below the lists, the 'g_i2c_touch I2C Master Driver on r_iic_master' settings are displayed in a table:

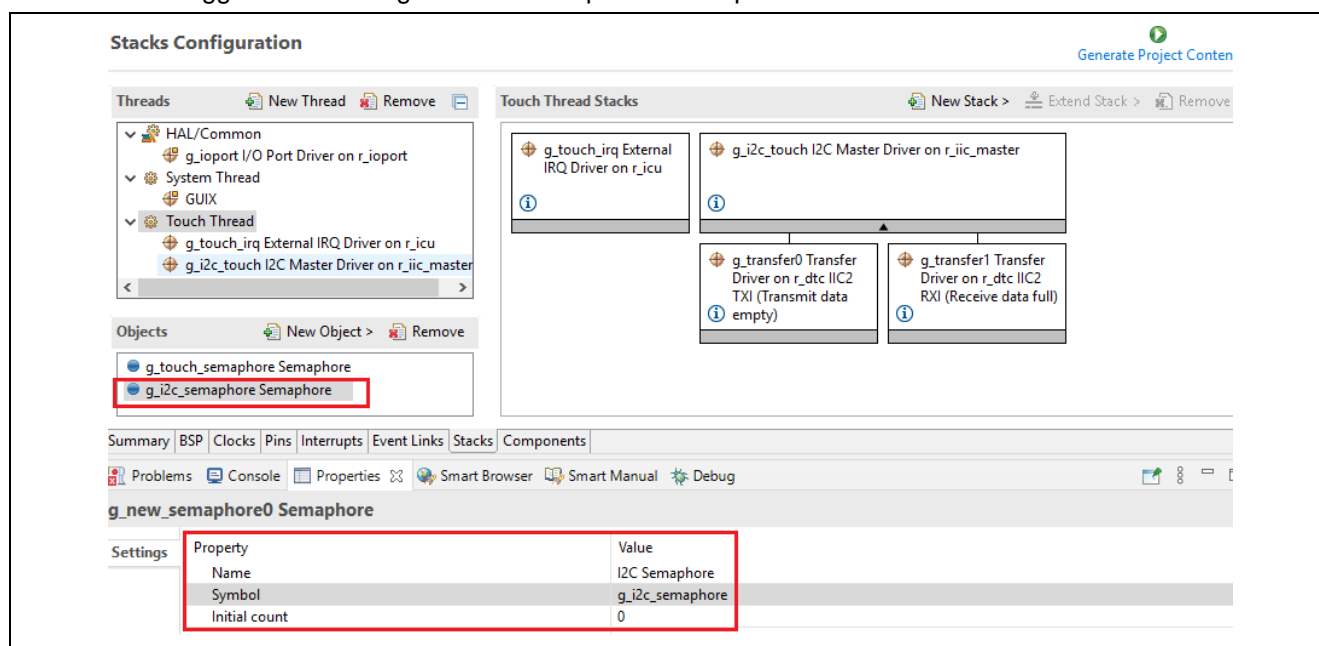
Property	Value
Parameter Checking	Default (BSP)
DTC on Transmission and Reception	Disabled
10-bit slave addressing	Disabled
Module g_i2c_touch I2C Master Driver on r_iic_master	
Name	g_i2c_touch
Channel	2
Rate	Fast-mode
Rise Time (ns)	120
Fall Time (ns)	120
Duty Cycle (%)	50
Slave Address	0x38
Address Mode	7-Bit
Timeout Mode	Short Mode
Callback	touch_i2c_callback
Interrupt Priority Level	Priority 6
Pins	
SDA	P511
SCL	P512

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

The screenshot shows the 'Stacks Configuration' window. In the 'Threads' list, 'Touch Thread' is selected. In the 'Touch Thread Stacks' list, 'g_i2c_touch I2C Master Driver on r_iic_master' is added. In the 'Objects' list, 'g_touch_semaphore Semaphore' is added and highlighted with a red box. Below the lists, the 'g_new_semaphore0 Semaphore' settings are displayed in a table:

Property	Value
Name	Touch Semaphore
Symbol	g_touch_semaphore
Initial count	0

9. In project configuration, add **I2C Semaphore** as shown below. This semaphore is used in the ft5x06 driver to trigger data reading when a touch-panel interrupt occurred.



10. In RA Configurator, click **Generate Project Content** to generate project content.
11. Copy and replace the files in **src** folder in the e² studio project with the files in **4.11** folder in the AN folder:
- hmi_event_handler.c
 - system_thread_entry.c
 - touch_thread_entry.c

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

```

/* 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)
{
    gxe.gx_event_type = GX_EVENT_PEN_UP;
    gx_system_event_send(&gxe);
}

```


13. All the screens designed in the Azure RTOS GUIX studio project are now created in `system_thread_entry.c`

```

/* Create a screen and attached it 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);
}

```

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.

```

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:
            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;
}

```

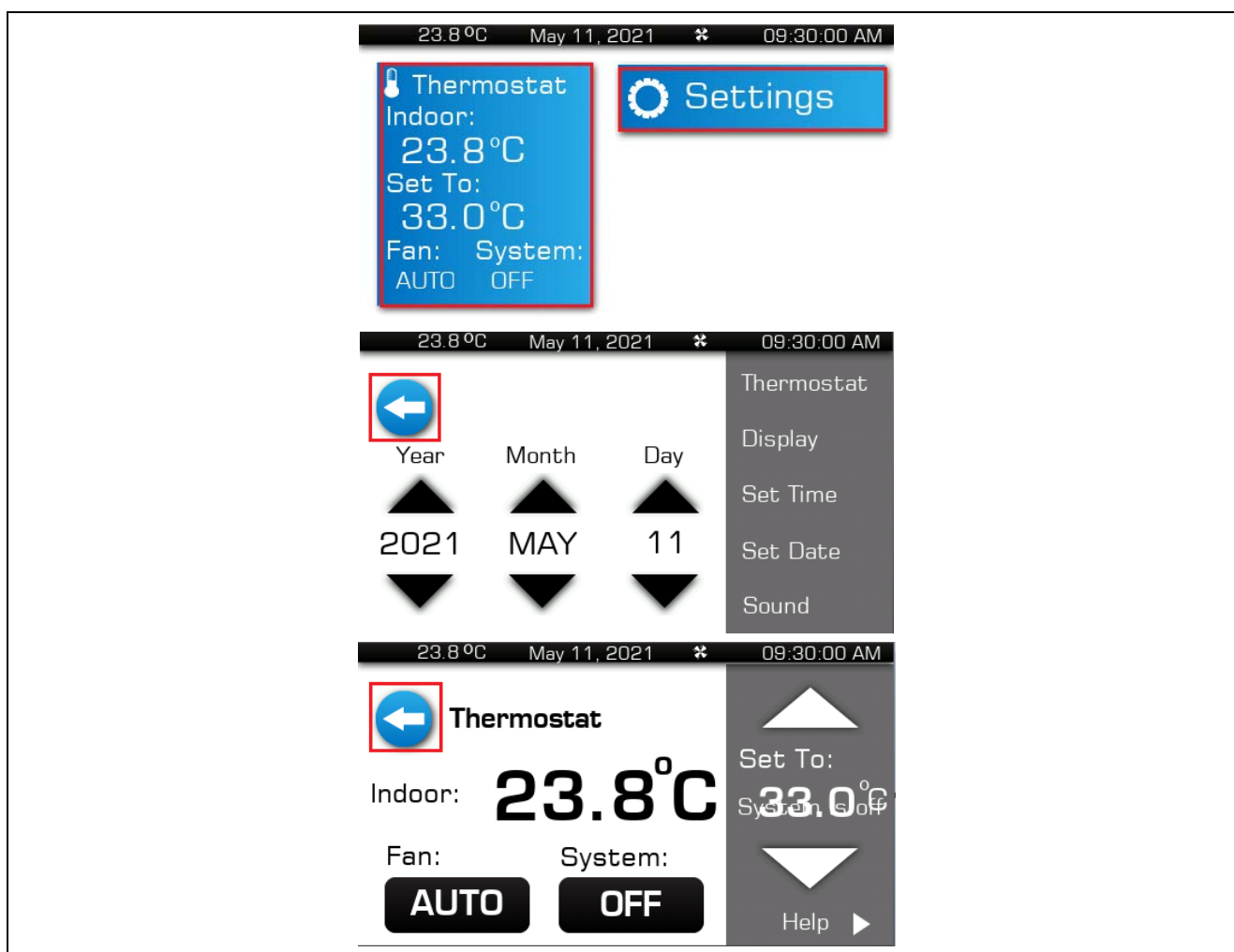
```

UINT settings_screen_event(GX_WINDOW *widget, GX_EVENT *event_ptr)
{
    UINT gx_err = GX_SUCCESS;
    switch (event_ptr->gx_event_type)
    {
        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) {
                while(1);
            }
            break;
        default:
            gx_err = gx_window_event_process(widget, event_ptr);
            if(GX_SUCCESS != gx_err) {
                while(1);
            }
            break;
    }
    return gx_err;
}

```

The above code return to MainPage screen when **Back** button on Settings screen is clicked.

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

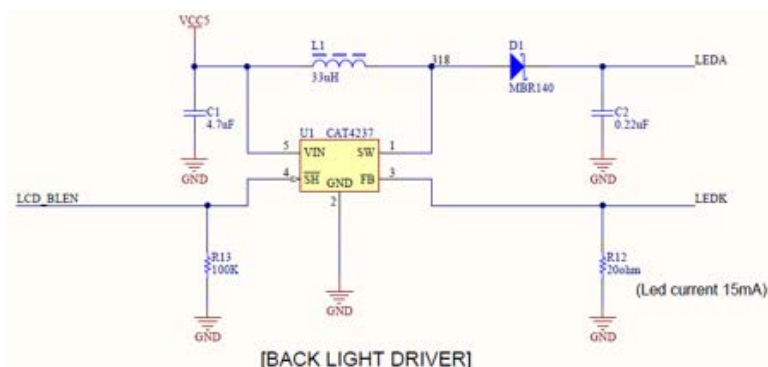


5. Control LCD Backlight

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

The steps are as follows:

1. In LCD board schematics below, the LCD_BLEN signal, which is connected to the P603 on the RA6M3 MCU, is configuring in PWM mode to control the intensity of LCD backlight.



2. To configure P603 in PWM output mode, we disable it in Pin Configuration at first. **Save this change before moving to the next step.**

Pin Configuration

Select Pin Configuration Export to CSV file Configure Pin Driver Warnings

RA6M3G-EK.pincfg Manage configurations... Generate data: g_bsp_pin_cfg

Pin Selection

Type filter text

- > P3
- > P4
- > P5
- ✓ P6
- ✓ P600
- ✓ P601
- ✓ P602
- P603**
- P604
- P605
- P606
- P607
- ✓ P608
- ✓ P609
- ✓ P610

Pin Configuration

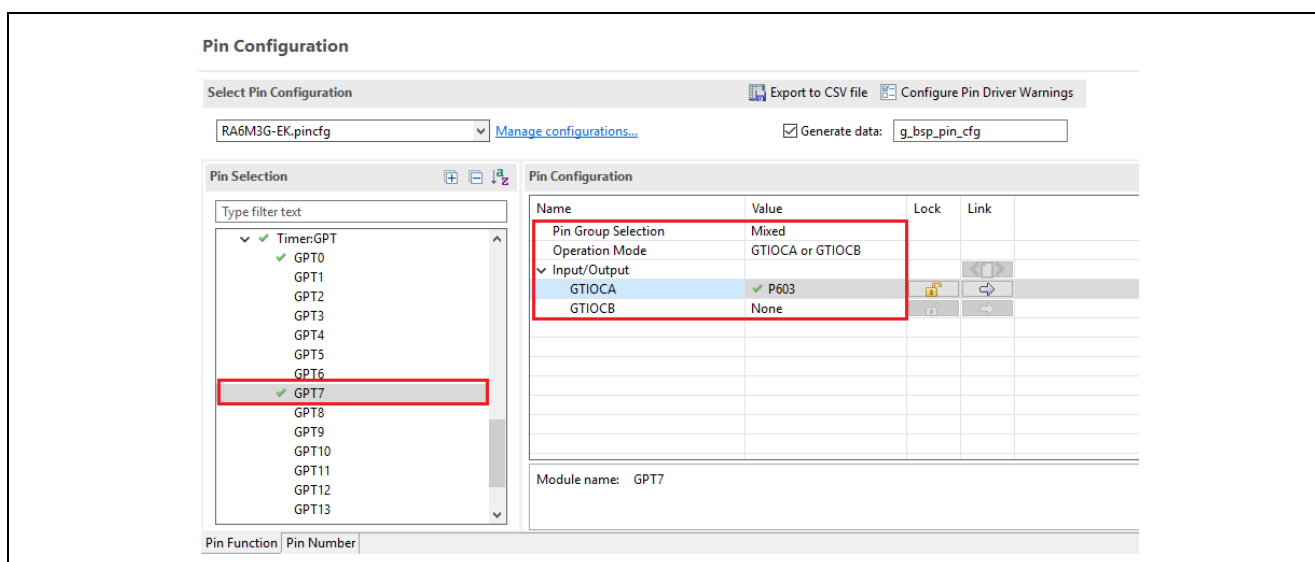
Name	Value	Link
Symbolic Name		
Comment		
Mode	Disabled	
Pull up	None	
Drive Capacity	Low	
Output type	CMOS	
Input/Output		
P603	None	

Module name: P603

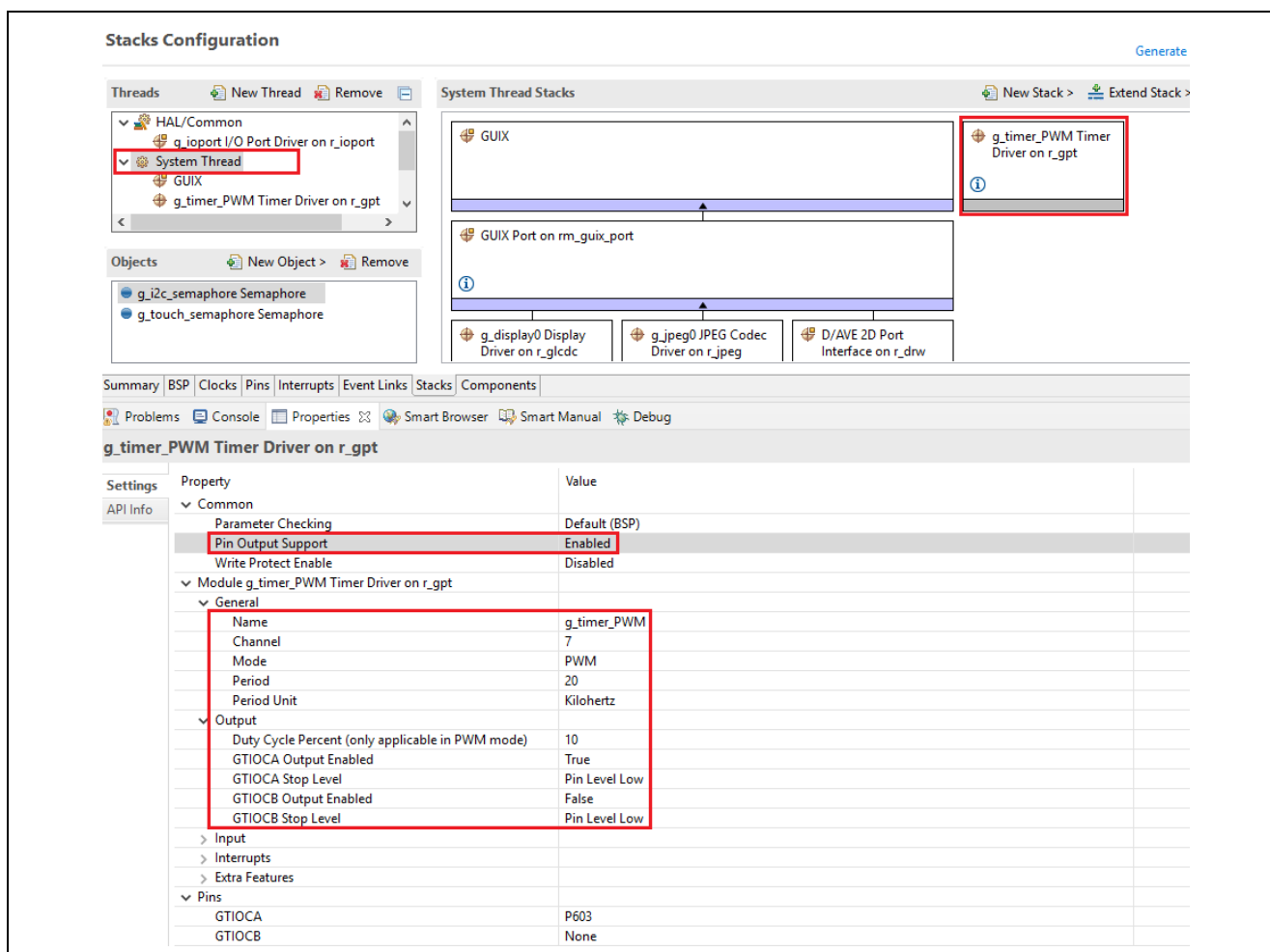
Port Capabilities: BUS0: D13_DQ13
GPT7: GTIOCA

Pin Function Pin Number

3. In Pin Configuration, set P603 as GPT7 GTIOCA output.



4. In project configuration, add **Timer Driver on r_gpt** to **System Thread** with below settings.



5. In RA Configurator, click **Generate Project Content** to generate project content.

6. Copy and replace the files in **src** folder in e² studio project with the files in **5.6** 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. */
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_GTIOCA);
```

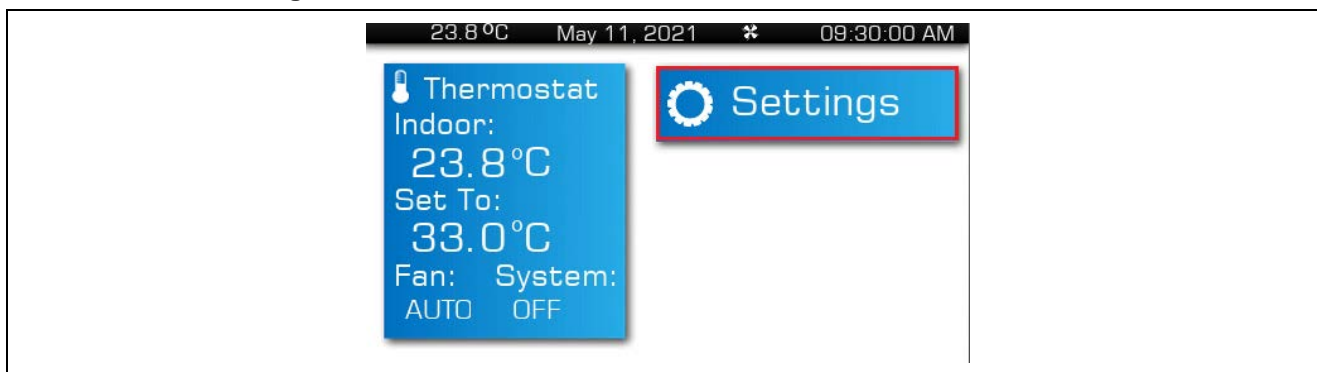
8. Looking at gpt_timer_PWM_Setup function in system_thread_entry.c, you will see brightness (**duty cycle of PWM output**) is set to 50 percent.

```
static fsp_err_t gpt_timer_PWM_setup(void)
{
    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;
    }

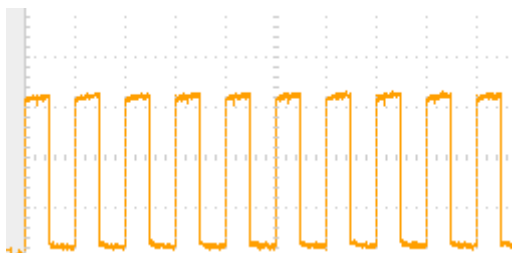
    /* Set brightness (LCD backlight) level: 50 = (45+5) */
    g_gui_state.brightness = 45;
    brightness_up(&g_gui_state.brightness);

    return err;
}
```

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



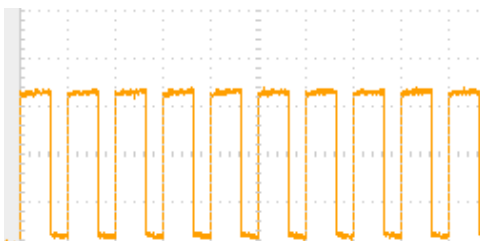
10. PWM output measured on pin P603 with brightness is set to 50%.



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



12. PWM output measured on pin P603 after changing brightness to 65%.



6. Update Date/Time and Temperature

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.

The steps are as follows:

1. In project configuration, add **RTC Driver on g_rtc** to **System Thread**.

The screenshot shows the 'Stacks Configuration' window. In the 'Threads' list on the left, 'System Thread' is selected and highlighted with a red box. In the 'System Thread Stacks' area on the right, the 'g_rtc RTC Driver on r_rtc' component is added and highlighted with a red box. Below the configuration window, the 'g_rtc RTC Driver on r_rtc' settings are displayed in a table.

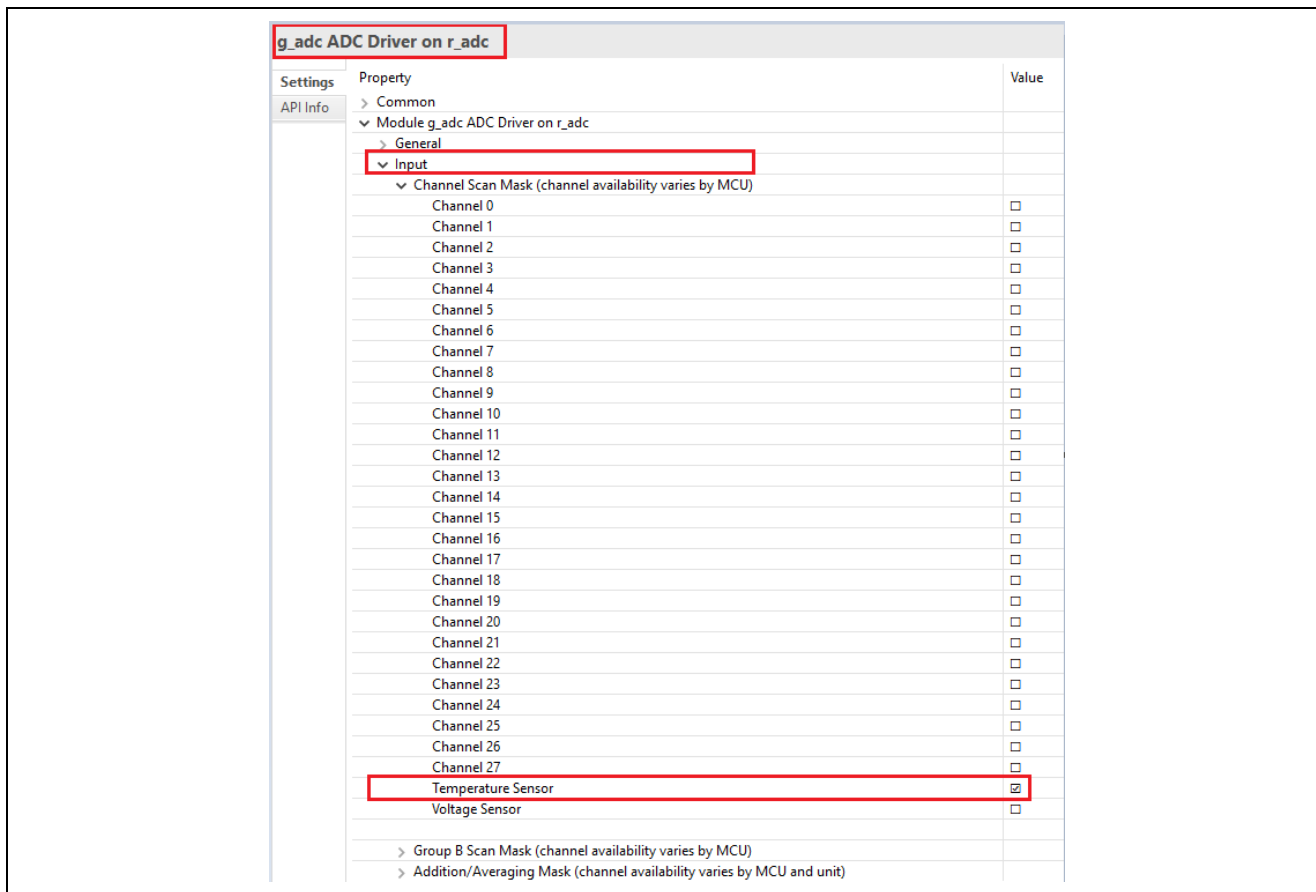
Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_rtc RTC Driver on r_rtc	
Name	g_rtc
Clock Source	LOCO
Frequency Comparison Value (LOCO)	255
Automatic Adjustment Mode	Enabled
Automatic Adjustment Period	10 Seconds
Adjustment Type (Plus-Minus)	NONE
Error Adjustment Value	0
Callback	time_update_callback
Alarm Interrupt Priority	Disabled
Period Interrupt Priority	Priority 9
Carry Interrupt Priority	Priority 12

2. In project configuration, add **ADC Driver on r_adc** to **System Thread**.

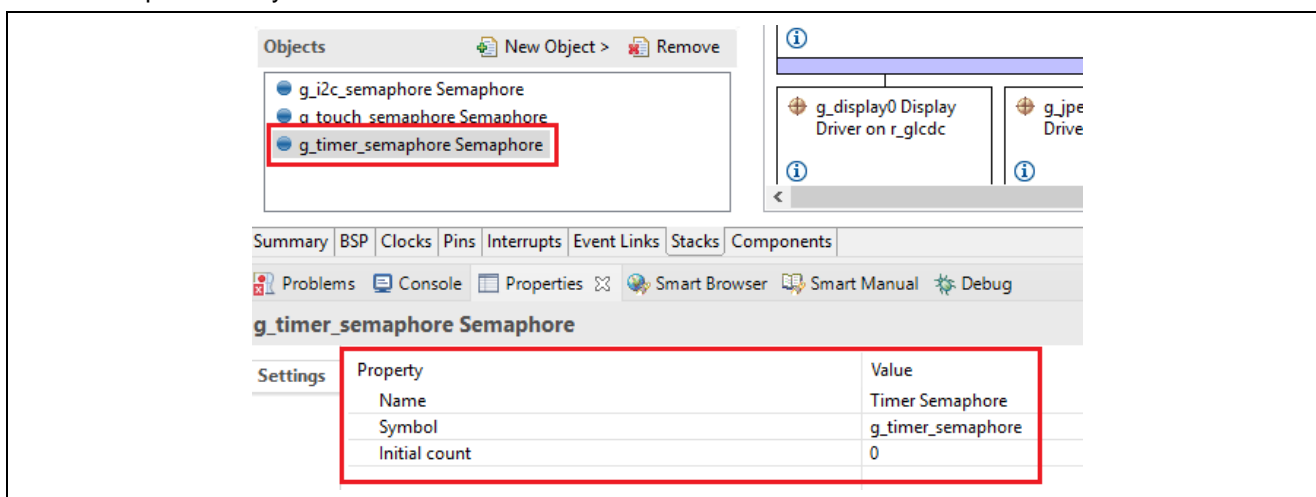
The screenshot shows the 'Stacks Configuration' window. In the 'Threads' list on the left, 'System Thread' is selected and highlighted with a red box. In the 'System Thread Stacks' area on the right, the 'g_adc ADC Driver on r_adc' component is added and highlighted with a red box. Below the configuration window, the 'g_adc ADC Driver on r_adc' settings are displayed in a table.

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_adc ADC Driver on r_adc	
General	
Name	g_adc
Unit	0
Resolution	12-Bit
Alignment	Right
Clear after read	On
Mode	Continuous Scan
Double-trigger	Disabled
Input	
Interrupts	
Extra	
Pins	

3. Select **Temperature Sensor** as input source for g_adc module.



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



5. In RA Configurator, click **Generate Project Content** to generate project content.
6. Copy and replace the files in **src** folder in e² studio project with the files in **6.6** folder in the Lab folder:
 - hmi_event_handler.c
 - system_thread_entry.c
 - system_time.c
 - system_time.h
 - system_api.h

7. In System Thread, date/time data and temperature data get updated every second. It then sends out events to trigger GUIX updates.

```
while (1)
{
    /* Wait for RTC interrupt. */
    status = tx_semaphore_get(&g_timer_semaphore, TX_WAIT_FOREVER);
    if(TX_SUCCESS != status)
    {
        APP_ERR_TRAP(FSP_ERR_ASSERTION);
    }
    /* Get date, time */
    R_RTC_CalendarTimeGet(&g_rtc_ctrl, &g_gui_state.time);
    /* Send GUIX event to update time */
    send_hmi_message(GXEVENT_MSG_TIME_UPDATE);

    /* Delay and update temperature*/
    tx_thread_sleep (10);
    /* Read die temperature */
    err = R_ADC_Read(&g_adc_ctrl, ADC_CHANNEL_TEMPERATURE, &adc_temp_data);
    /* Handle error */
    if (FSP_SUCCESS != err)
    {
        APP_ERR_TRAP(err);
    }

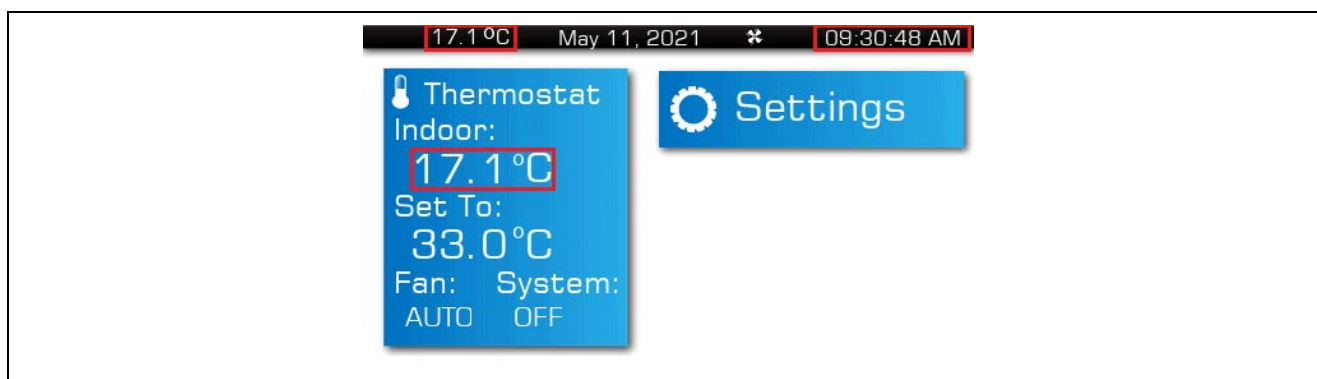
    /* Conversion of ADC temperature in celsius */
    g_gui_state.temp_c = ADCTEMP_AS_C(adc_temp_data);

    /* Send GUIX event to update time */
    send_hmi_message(GXEVENT_MSG_UPDATE_TEMPERATURE);
    tx_thread_sleep (1);
}
```

8. Following is an example of handling temperature and time update events in the MainPage screen event handler.

```
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 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:
            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;
}
```

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

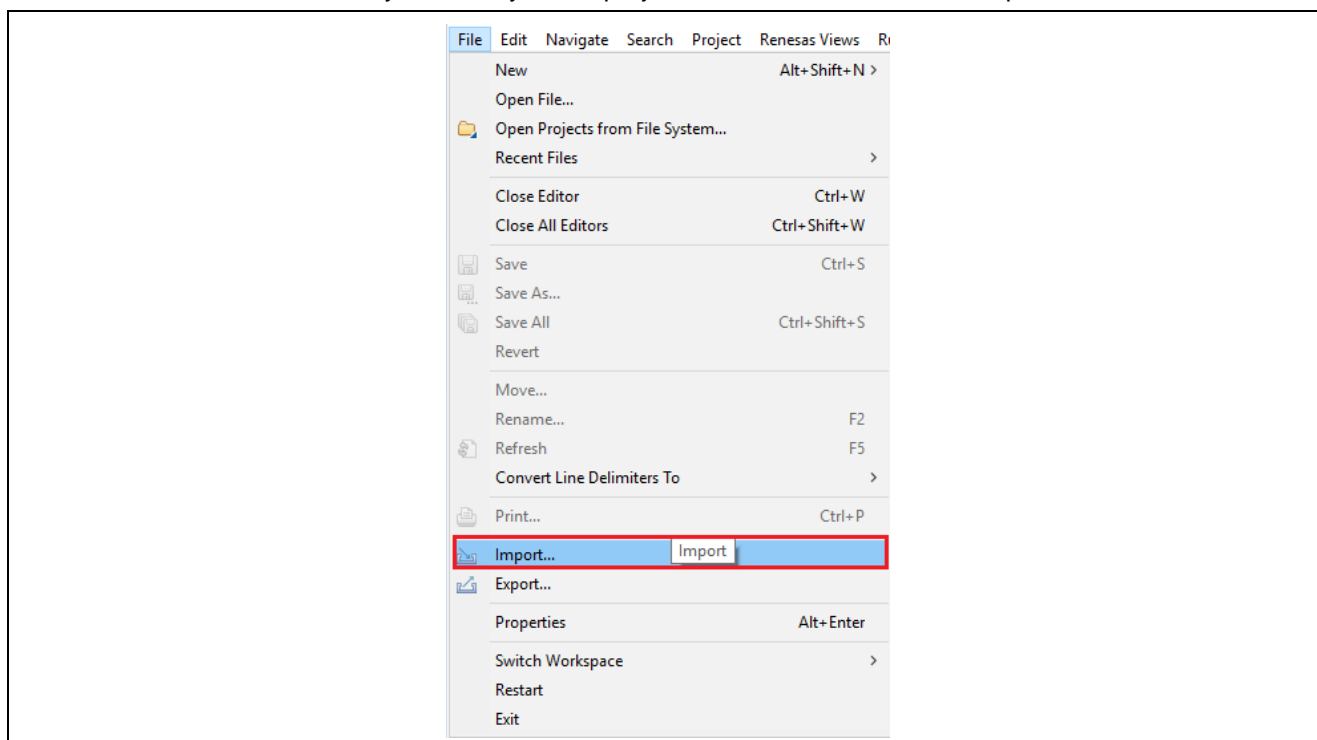


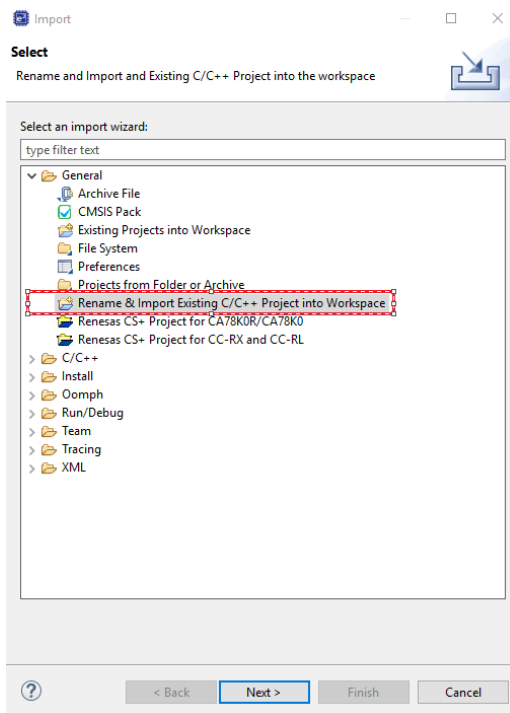
7. Setting Date/Time in A Full Function Project

In this section, you will import and run the complete Thermostat project that enable 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.

The steps are as follows:

1. You can try the completed project in **completed_project** folder that has a full function Thermostat application. Use **Rename & Import Existing C/C++ Project into Workspace** feature of **Import** menu in e² studio to do so since you already had a project with the same in the workspace.





Website and Support

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

RA Product Information	www.renesas.com/ra
RA Product Support Forum	www.renesas.com/ra/forum
RA Flexible Software Package	www.renesas.com/FSP
Renesas Support	www.renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep.14.21	—	First release document

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/