Renesas RA Family

Getting Started with the Graphics Application on EK-RA8E2

Introduction

This application note describes the creation of an application that uses Graphical User Interfaces with an EKRA8E2 kit, referred to as a ‘graphics application’. This application is geared towards providing a reference for developing complex multi-threaded applications with a touchscreen graphical Human Machine Interface (HMI) by using the Renesas Flexible Software Package (FSP) and SEGGER AppWizard (hereinafter referred to as AppWizard).

Figure 1. Weather Panel of the Graphics Application on Renesas EK-RA8E2

It 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®-based microcontrollers (MCUs). RA FSP provides production-ready peripheral drivers to take advantage of the RA FSP ecosystem, along with the emWin (hereinafter referred to as emWin) library and FreeRTOS. In addition, Ethernet, USB, and file system stack support are also available. This powerful suite of tools provides a comprehensive, integrated framework for the 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 FreeRTOS. It makes use of RTOS features such as threads and semaphores. Knowledge of operating these with FreeRTOS can help in understanding the supplied application project in the source code. For more detailed information on FreeRTOS features, refer to the FreeRTOS User Manual.

The graphics application is developed using the Renesas e^2 studio Integrated Development Environment (IDE). This e^2 studio is a free application that you can download from the Renesas website. While building applications under the Renesas FSP Platform is considerably faster than developing similar applications in other environments, there is still a learning curve to understand the steps necessary to construct complex multi-threaded HMI applications quickly. The following steps will guide you through the necessary process:

- Board setup.
- Application overview.
- Detailed explanation of uses of the graphical screens.
- AppWizard project integration.
- AppWizard interactions setup.
- Adding an emWin widget that is not yet available in AppWizard.
- FSP configuration.
- Application design highlights.
- Using the General-Purpose Timer to drive a PWM backlight control signal.
- Importing, loading, and running the project.

Required Resources

Development tools and software

- e² studio v2025-04.1
  e² studio | Renesas
- Renesas Flexible Software Package (FSP) v6.0.0
  RA Flexible Software Package (FSP) | Renesas
- AppWizard V1.56_6.48
  SEGGER emWin GUI Library for Renesas RA Products | Renesas
  Note: The version of emWin in FSP must match the emWin version in the AppWizard. For example, the emWin version 6.48 in FSP is equivalent to 6.48 in the AppWizard V1.56_6.48, and so on.

Hardware

- Renesas EK-RA8E2

Reference Manuals

- RA Flexible Software Package Documentation Release v6.0.0
- AppWizard User Guide & Reference Manual Version 1.56
- emWin User Guide & Reference Manual Version 6.48
- Renesas RA8E2 Group User's Manual Rev.1.00
- EK-RA8E2-v1.0 Schematics

Contents

1. Board Setup

The EK-RA8E2 kit contains a few switch settings that must be configured before running the application associated with this application note. In addition to these switch settings, the boards also contain a USB debug port and connectors to access the J-Link® programming interface. Switch 4-3 must be OFF to use the pin group for communication with the external OSPI flash.

Table 1. Switch Settings for EK-RA8E2

Switch	Setting
J8	Jumper set on pins 1-2
J9	Jumper set on pins 2-3
SW4-3	OFF

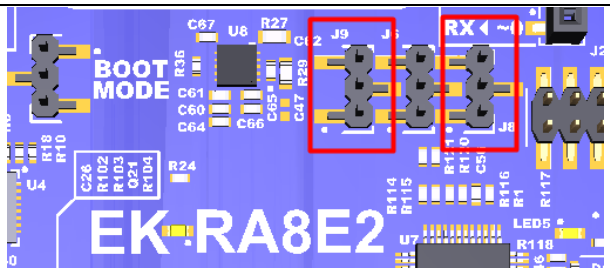

Figure 2. J8 and J9 on EK-RA8E2

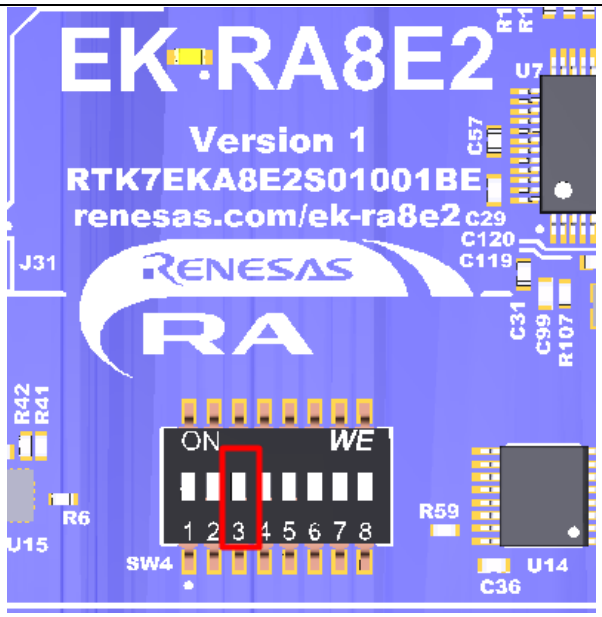

Figure 3. Switch OFF SW4-3 OSPI_OE_L

The EK-RA8E2 kit consists of two boards: the EK-RA8E2 board featuring the RA8E2 MCU with an on-chip Graphics LCD Controller, and a Parallel Graphics Expansion Board 2 featuring a 5-inch 800 x 480-pixel TFT color LCD with capacitive touch overlay. The GPIO port pin driving the backlight controller is capable of PWM output using a timer peripheral in the MCU. As a result, the intensity of the LED backlight can be adjusted by the RA8E2 MCU.

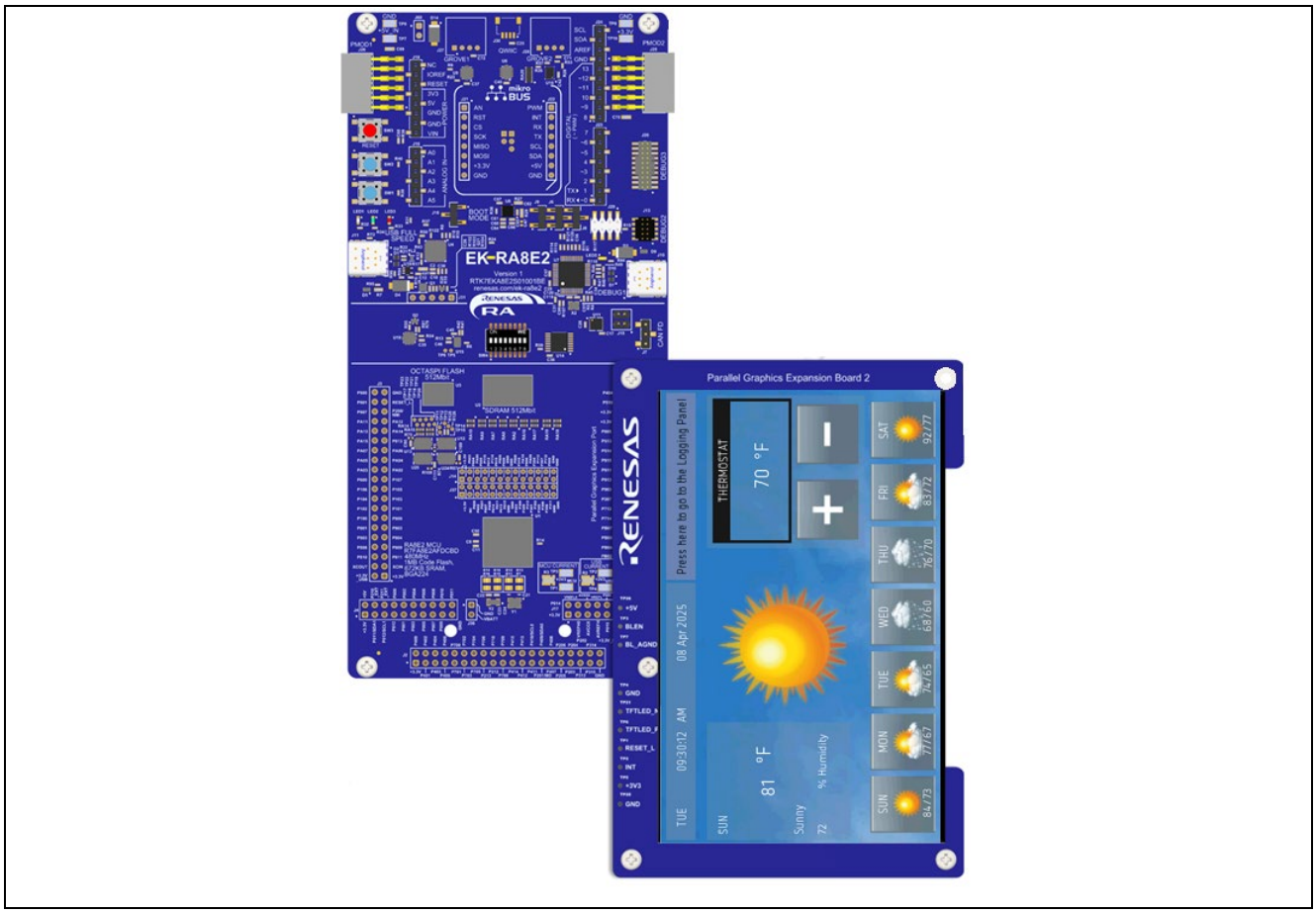

Figure 4. Overview of the EK-RA8E2 Kit with the Parallel Graphics Extension Board 2.

2. Application Overview

One of the key goals of the provided graphics application is to demonstrate how to build applications that require complex HMI screens using AppWizard and the emWin library. The following list highlights all the key features of the graphics application:

- Complex HMI design using AppWizard.
- External octal OSPI Flash to store images.
  - OSPI_B configuration.
- Multi-threaded applications using FreeRTOS
  - Semaphore object.
- GLCDC configuration
  - Framebuffer configuration.
  - TCON configuration.
- Touch Panel, I2C touch controller driver gt911.
  - External IRQ mapping.

There can be many ways to achieve the target design, and the approach described in this application note is one possible solution.

2.1 RA8E2 MCU Peripherals Used by the Graphics Application

The graphics application is complex, and it uses the Renesas RA8E2 MCU. This MCU is built around an Arm® Cortex®-M85 device. Developing complex microcontroller-based applications is usually a multi-step process:

1. The first step usually involves gathering the application requirements and performing a high-level system design that maps the requirements onto the set of hardware components. The components necessary to

fulfill those requirements include the target MCU used in the design, the tool chains required to build/debug the applications, and so forth.

2. The next step usually determines which on-board peripherals of the target MCU are used. In this step, it is often necessary to spend considerable time understanding the onboard peripherals' register map and writing the lower-level driver code necessary to expose the peripheral to the upper-level application code. Most of this work has already been done in the FSP, considerably streamlining application development.
3. Besides the on-board peripherals of the target MCU, the design often encompasses external hardware and how it is controlled. For example, the EK-RA8E2 has a Graphics Expansion board, which is controlled directly by the on-chip Graphics LCD Controller (GLCDC) of the RA8E2 MCU.
4. The last step usually details how an application will be structured on top of the selected hardware to accomplish the initial requirements.

   The graphics application requirements were first mapped to the onboard peripherals of the EK-RA8E2 kit. Figure 5 shows all the internal hardware peripherals used by the graphics application. This application note describes how each of these peripherals is configured using the FSP and the considerations that were used for each peripheral as the application is being developed.

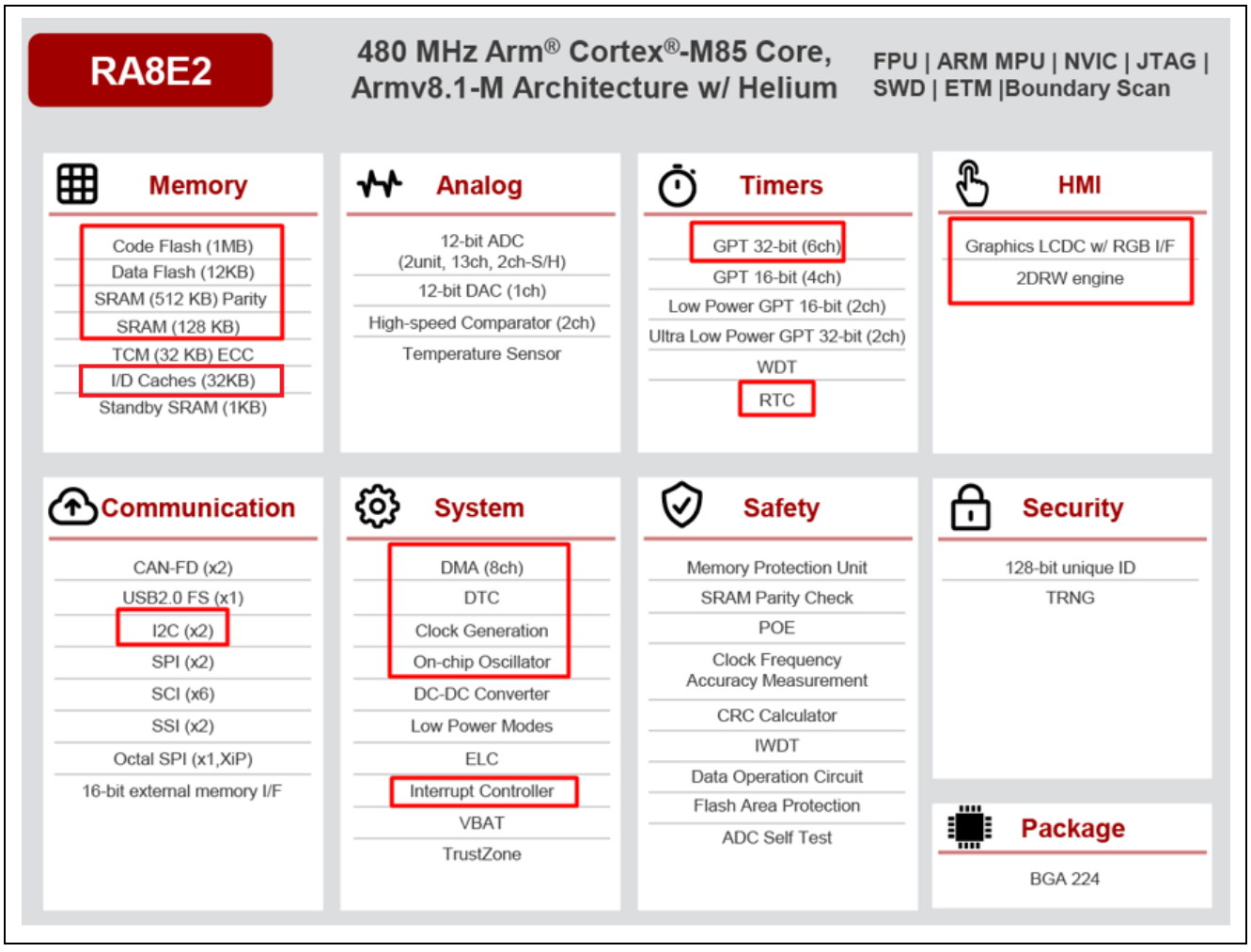

Figure 5. RA8E2 MCU Peripherals Used in the Graphics Application

2.2 Graphical User Interface (GUI)

In many human-machine interface (HMI) applications, the most daunting task may be the GUI itself. In applications requiring a graphical HMI, it is generally considered best practice to separate the business logic from the presentation. This abstracts the GUI from making decisions on what to display. Instead, it is now only concerned about how to display it. It relies on external logic to tell it what to display and when to display it.

Once you have gathered the requirements, achieved a top-level design, and identified the hardware necessary to implement that design, it is often beneficial to construct a GUI (Graphical User Interface) to help quickly communicate the look and feel of the system to others. This is where the AppWizard comes into play.

The FSP natively supports the use of AppWizard and the emWin library from SEGGER. You may choose to use emWin primitive calls directly in your application or choose to use the AppWizard to design your screens. AppWizard is a stand-alone tool that provides a point-and-click environment for generating all the screens necessary for your embedded application. Once designed, the tool outputs `.c` and `.h` files, which you then include in your application. All the application screens in the graphics application were built using the AppWizard.

2.3 Graphics Application Panels

The graphics application consists of two graphical panels: a Weather Panel and a Logging Panel. In this application, we build separate static display designs for these two panels.

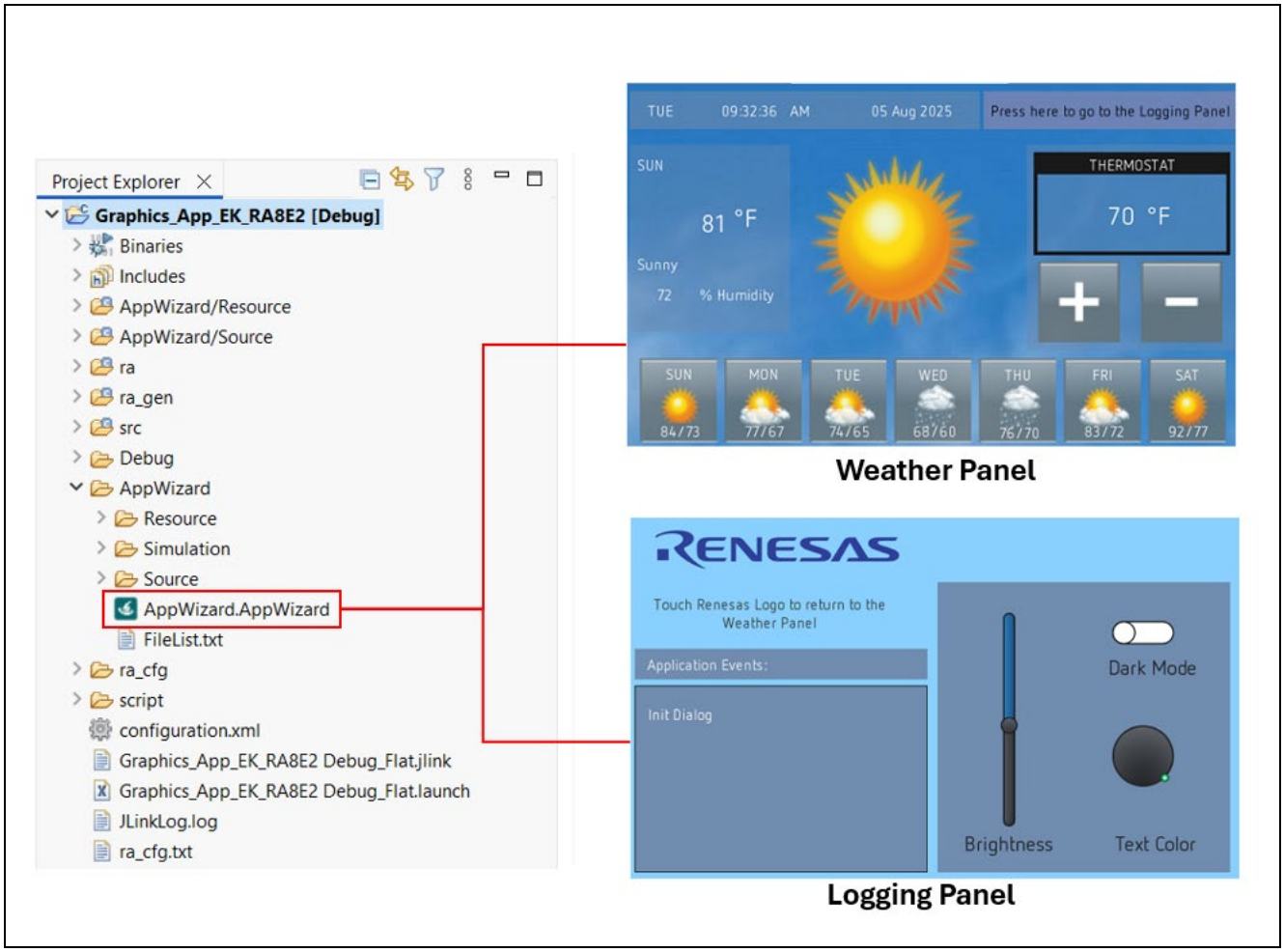

Figure 6. Screenshot of the Graphics Application

Weather Panel	This is the first screen that appears on the kit when booting up. It shows Weather forecasts by selecting days or increasing/decreasing Temperatures.
Logging Panel	This panel shows events in the **Weather Panel** and adjusts the LCD backlight or text color and background color of the Logging Editor.

3. AppWizard Overview

This section provides an overview of how graphical screens are designed and integrated into an FSP application using the AppWizard and emWin library. It is not meant to replace the AppWizard or emWin documentation. When designing graphical interfaces for the Renesas FSP platform, you are encouraged to refer to the documentation for the AppWizard and emWin library.

The AppWizard presents a graphical point-and-click environment that allows you to quickly create all the screens needed for your embedded application. You can specify the screen resolution, color depth, and

various other parameters so that what you see in the AppWizard that is running on your PC is what you will get on your embedded screens.

The AppWizard comes with a standard set of fonts and basic interface graphics, including images, text, buttons, rotaries, sliders, and more. During your screen creation phase, you can customize the display by importing image and font files into AppWizard. For optimal performance, images should be prepared in the same color format as the framebuffer. This approach eliminates runtime format conversion in emWin and ensures efficient rendering.

Figure 7. Screenshot of the Weather Panel being designed in the AppWizard

The organization of the AppWizard is straightforward. The top center window, known as the **Editor** window, contains the screen being designed. In the upper left corner, you will find the **Add objects** window. This window shows the supported window objects in the AppWizard. It allows you to click on the object icons and drag and place them in the Editor window. On the center-left is the **Hierarchic tree** window. The order in which you add items in the same level/parent determines the order in which they are drawn in the final screens, so some planning is necessary. However, you still can change the order by using drag and drop or the **Move Up** and **Move Down** buttons. As is the case with most graphical design environments, screens are laid out in a hierarchy where the main window is usually the parent, and all graphical objects contained in the window are children of that parent. The **Properties** window on the right side displays properties associated with a selected object. You may select objects from the **Hierarchic tree** window or from the **Editor** window.

The bottom left of the AppWizard screen contains **Quick Access Buttons** for managing resources such as Texts, Fonts, Images, Animations, and Variables that you use to create and interact with the screens. Through Texts resource management, AppWizard also supports multi-language designs.

The key to making any graphical design interactive is to associate events like button touches with the event handling code that implements the appropriate functionality. The **Interactions** window at the bottom center makes it easy for you to define the application's behavior regarding certain actions. These interactions can be done without any extra code, but AppWizard allows you to add your code to handle these actions and respond to GUI events.

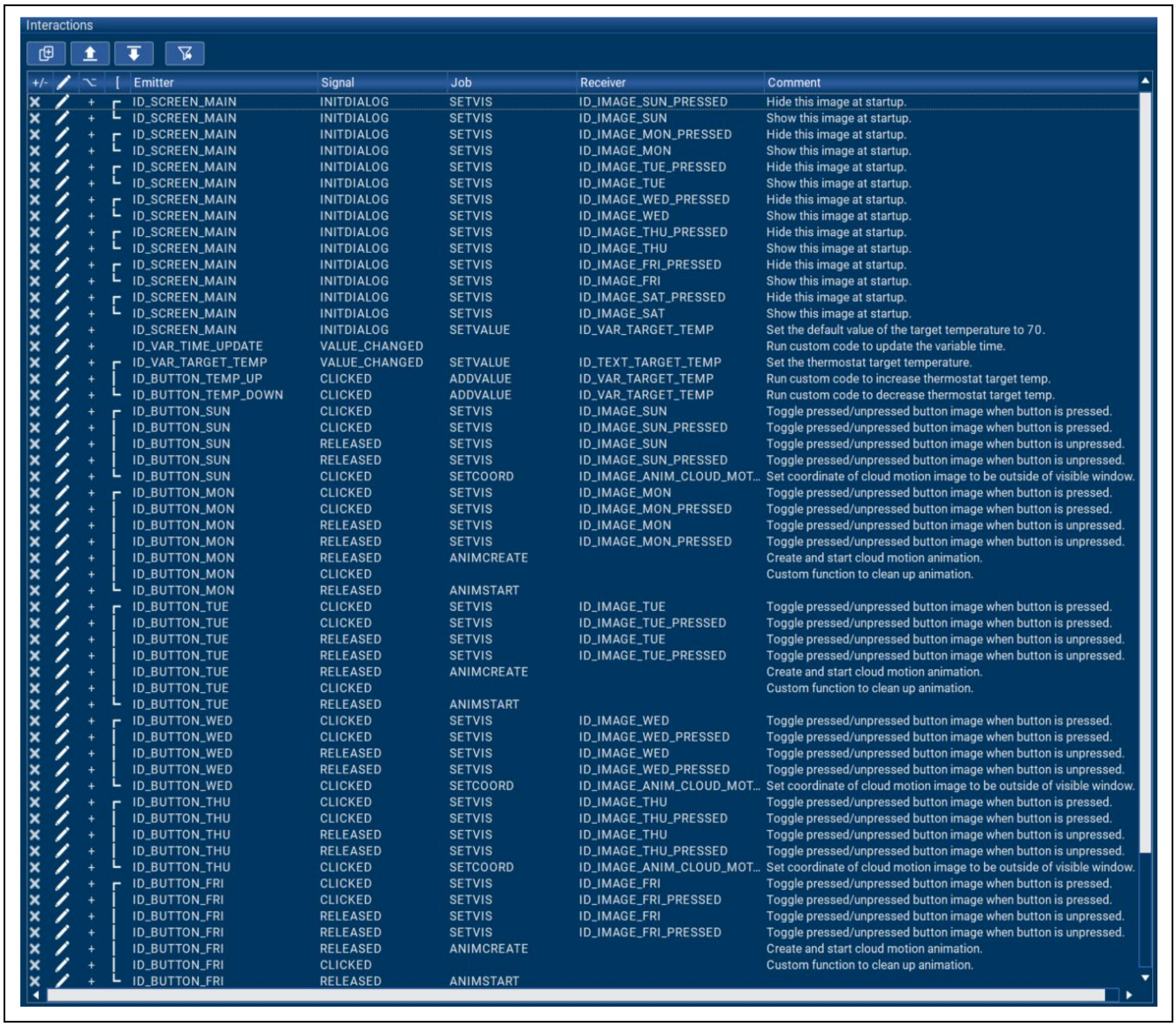

Figure 8. AppWizard Interactions Window

3.1 Create a New Project Using the AppWizard

The **Create New Project** dialog box is shown in Figure 9. This dialog box is where you specify the project-specific information, such as the basic display settings, as well as the path information for where AppWizard locates the files that result from the **Export & Save** process. The color format can be selected during project creation in AppWizard. Ensure that this format matches the framebuffer color format configured in e^2 studio under the r_glcdc stack shown in Figure 10. A mismatch may result in incorrect colors or display issues.

The AppWizard also generates a simulation project in the folder `\Simulation` located in the project folder.

When you perform **Export & Save**, the AppWizard creates `.c` and `.h` files that contain all the information necessary to render the screens you built with AppWizard on the LCD in your embedded application. The **Project Path** is where you specify the default output directory for the Source, Header, and Resource files.

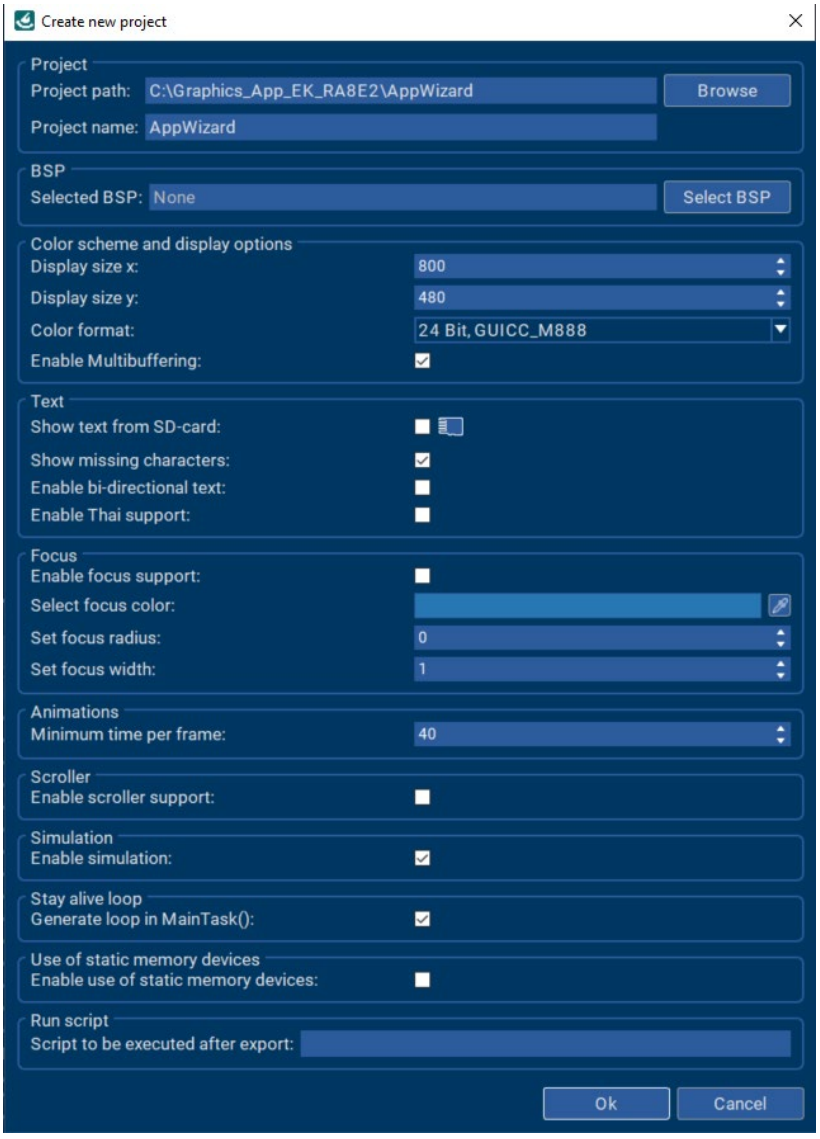

Figure 9. Create a New Project Dialog Box

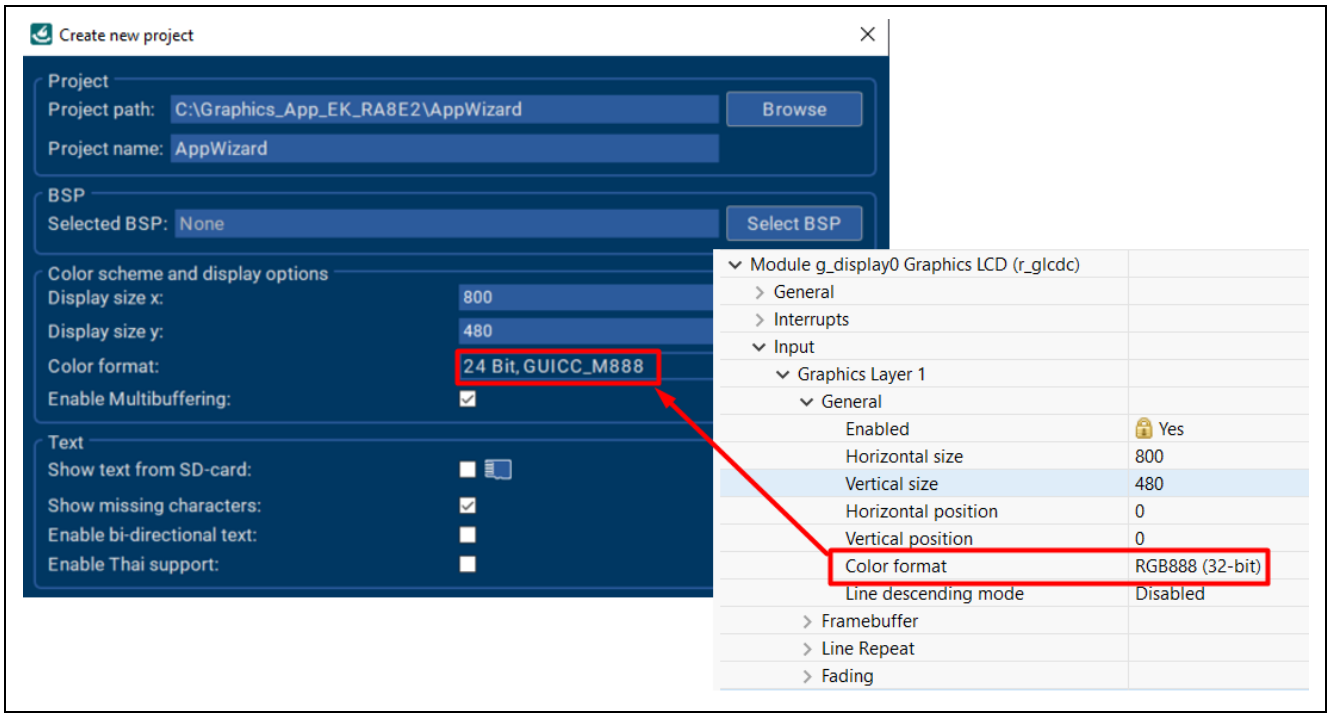

Figure 10. Example of Mapping Color Format from r_glcdc Configuration to AppWizard

It is a good practice to save the Source, Header, and Resource files relative to the e² studio project. This makes it easy to move projects from one location to another or from one PC to another. In the case of the graphics application, you can see that all the directories are located under the AppWizard folder in the project directory created by e² studio. We recommend creating the e² studio project first, then creating the AppWizard folder as an e² studio source folder before creating an AppWizard project named AppWizard under the e² studio project folder.

After generating the AppWizard, you should exclude the Simulation folders from Build and make sure that the Source and Resource folders are included to build before building the e² studio project.
To include Resource and Source folders: Right click on the Resource folder > Click Properties > Click C/C++ Build tab > Uncheck “Exclude resource from build”, do the same with the Source folder to include for build.
To exclude the Simulation folder from Build: Right Click on Simulation > Click Properties > Click C/C++ Build tab > tick “Exclude resource from build”. Refer to Figure 11 and Figure 12 for the setup. Once the setup is complete, the structure of the e² studio project, as illustrated in Figure 13.
All the necessary library and header files for the target board are generated after you finish adding the emWin stack to your e² studio project.

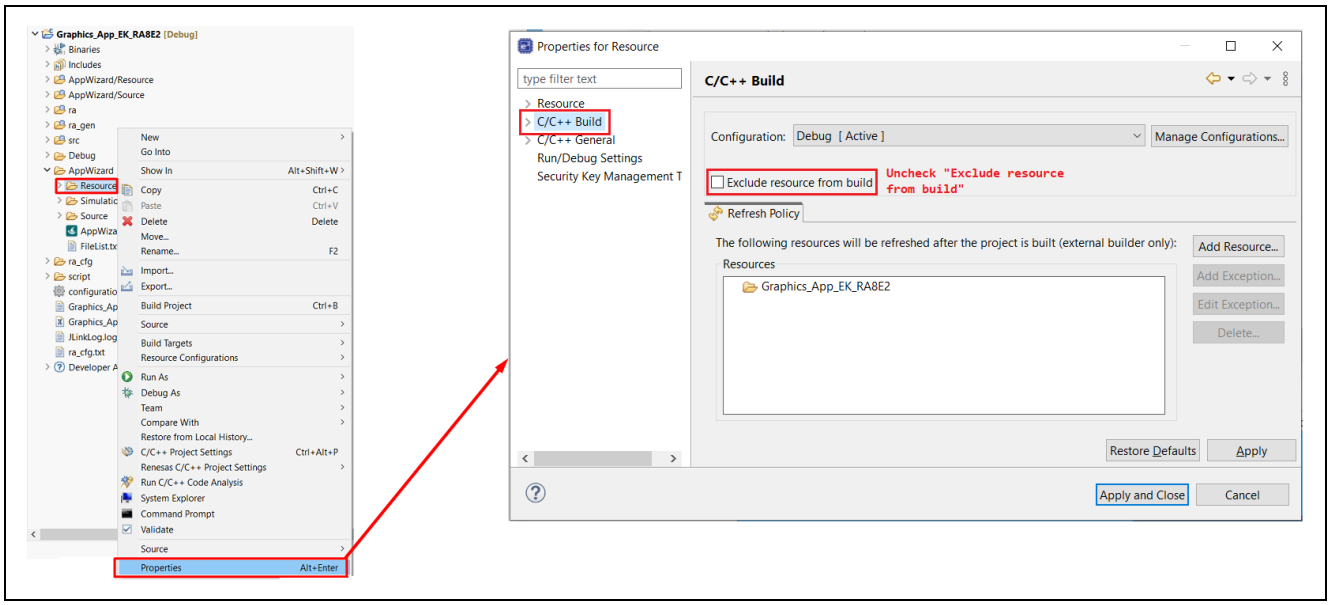

Figure 11. Include the Resources to Build

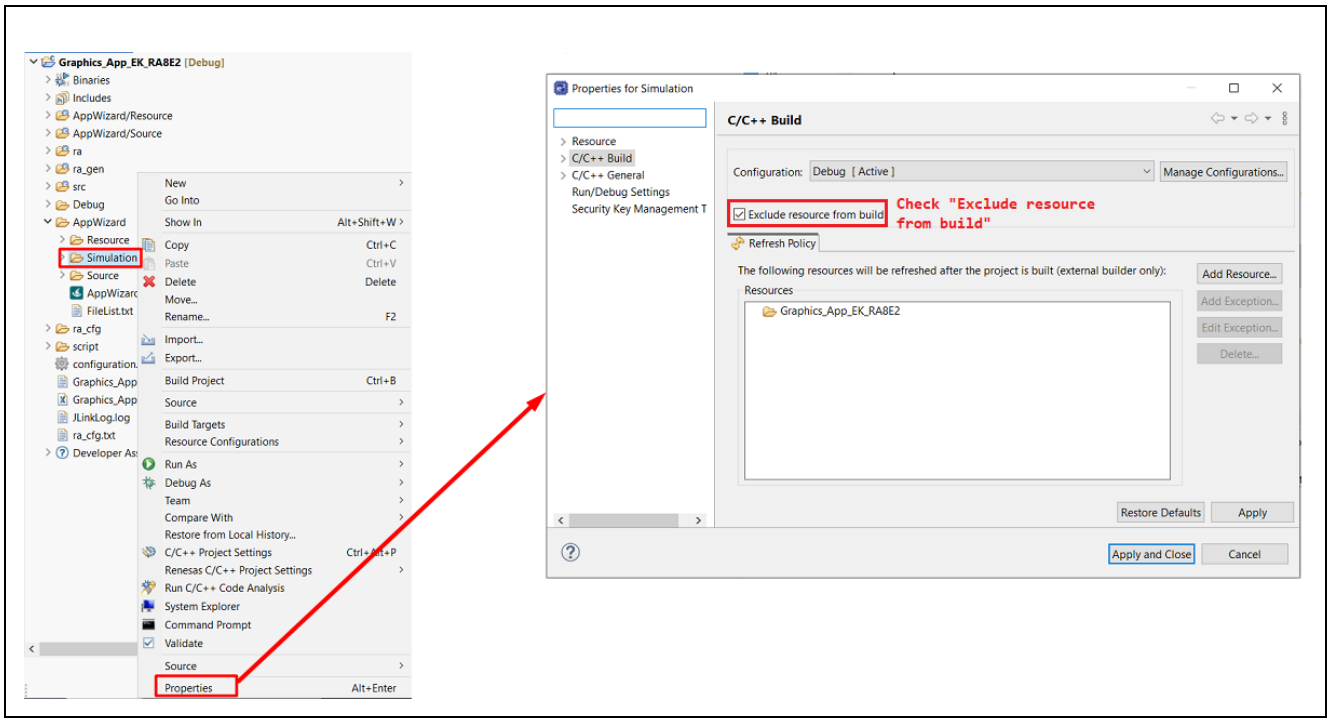

Figure 12. Exclude the Simulation Folder from the Build

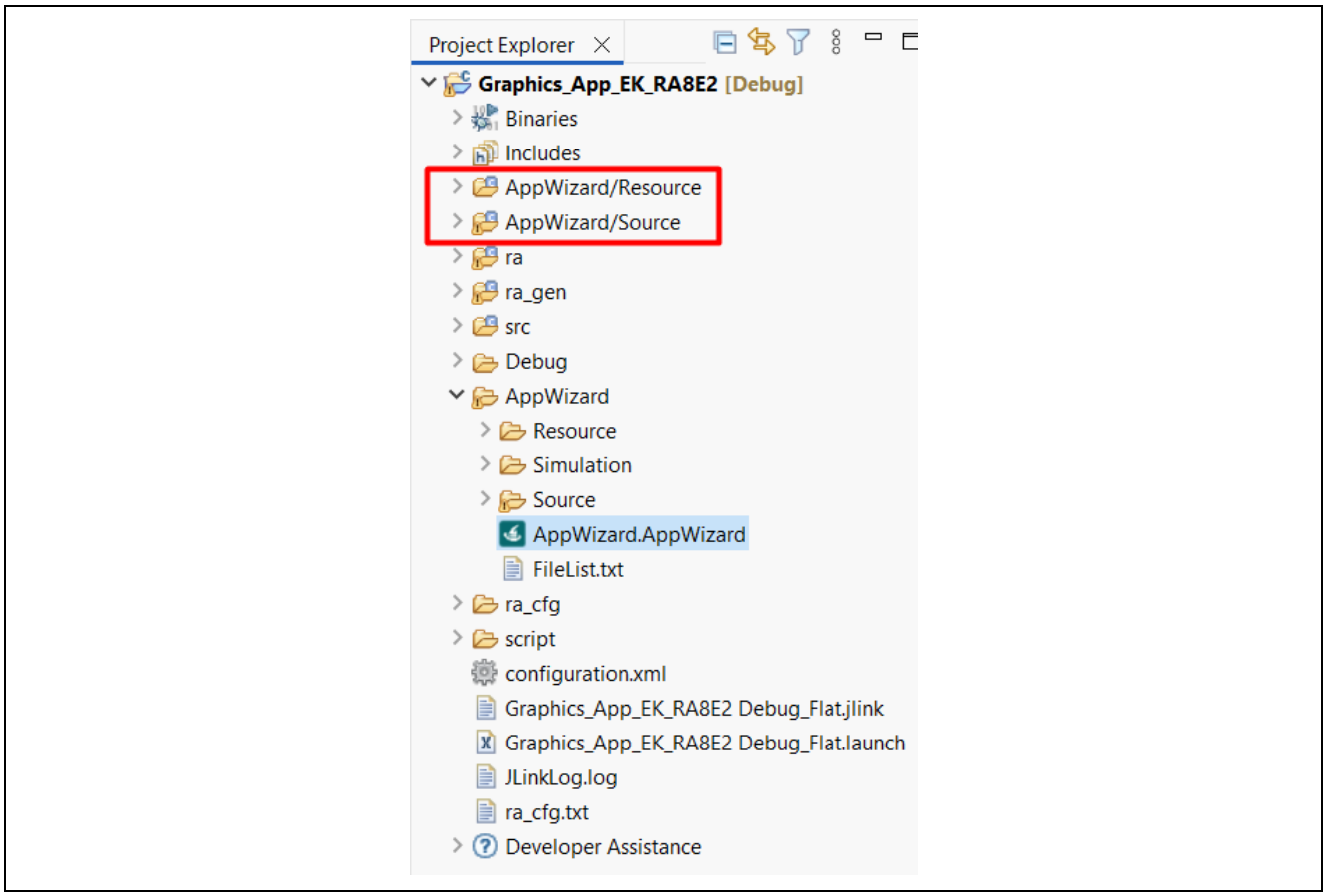

Figure 13. Project Structure View

Right click to **Project >** Click **Properties >** Click **C/C++ Build >** Click **Settings >** Click **GNU ARM Cross C Compiler > Includes >** Click **Add** to add the newly created AppWizard folder and its subfolders to the e² studio project; include the path as shown in Figure 14.

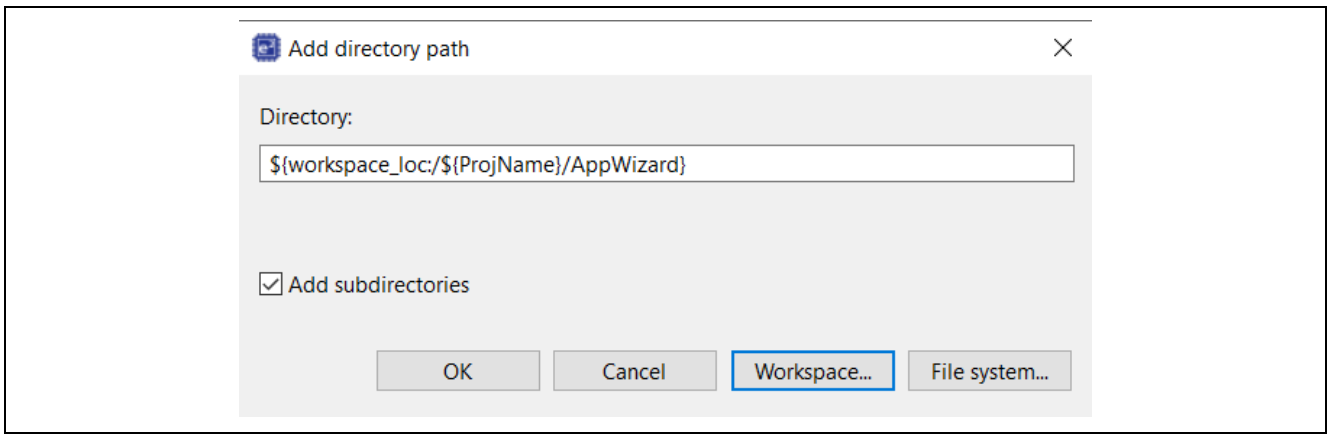

Figure 14. Adding the AppWizard Folder to the e² studio Project Includes Path

3.2 Design Weather Panel Buttons Using AppWizard

Segger provides two useful documents for AppWizard: the ‘User Guide & Reference Manual’ and the ‘Quick Start Guide’, both of which cover basic designs. The **Weather Panel** buttons, on the other hand, are more

complex and are the target of this application note. These buttons are grouped in a Window widget that includes multiple objects. For example, the window ID_WINDOW_SUN consists of:

- ID_WINDOW_SUN
  — Window widget. The placeholder to group the other widgets.
- ID_IMAGE_SUN_PRESSED
  — Image widget. Visible when the ID_BUTTON_SUN is pressed, invisible when the ID_BUTTON_SUN is released. Set bitmap using `bottom_button_trans_pressed.png`.

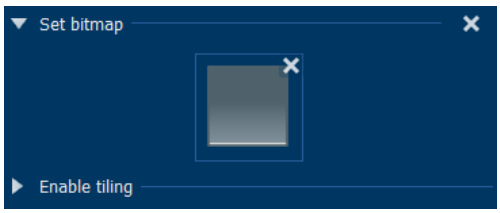

Figure 15. ID_MAGE_SUN_PRESSED Bitmap Setting

- ID_IMAGE_SUN
  — Image widget. Invisible when the ID_BUTTON_SUN is pressed, visible when the ID_BUTTON_SUN is released. Set the bitmap using `bottom_button_trans.png`.

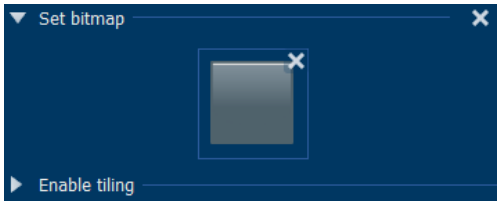

Figure 16. ID_MAGE_SUN Bitmap Setting

- ID_IMAGE_SUNNY_SUN
  — Image widget. Sunny icon. Set bitmap using `icon_sunny.png`.

Figure 17. ID_IMAGE_SUNNY_SUN Bitmap Setting

- ID_TEXT_SUN
  — Text widget. The “SUN” text.
- ID_TEXT_SUN_RANGE
  — Text widget. Shows temperature range.
- ID_BUTTON_SUN
  — Button widget. A transparent button without a bitmap image is placed on top of the other widgets. Some AppWizard interaction setups must be in place to create button-pressed/release impressions.

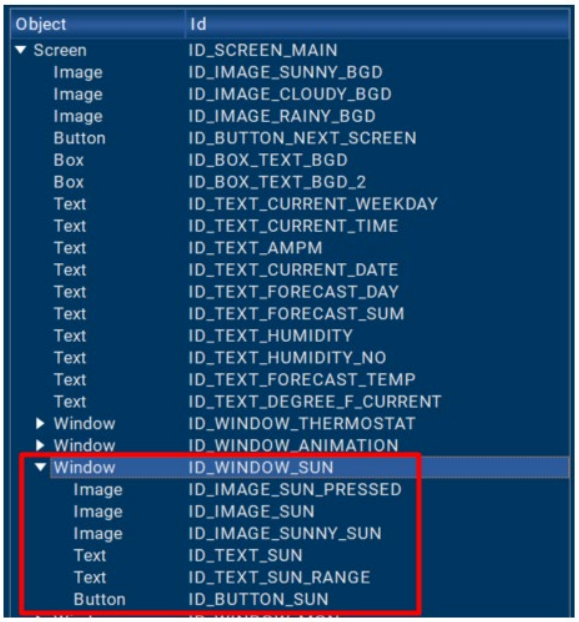

Figure 18. Design of the SUN Button Group

3.3 Setup AppWizard Interactions

Set the following interaction for the ID_BUTTON_SUN to create the button pressed/released as mentioned earlier in the Weather Panel Button Design section:

- The ID_IMAGE_SUN widget is invisible, toggling from visible to invisible when the transparent ID_BUTTON_SUN is pressed.
- The ID_IMAGE_SUN widget is visible, toggling from invisible to visible when the transparent ID_BUTTON_SUN is released.

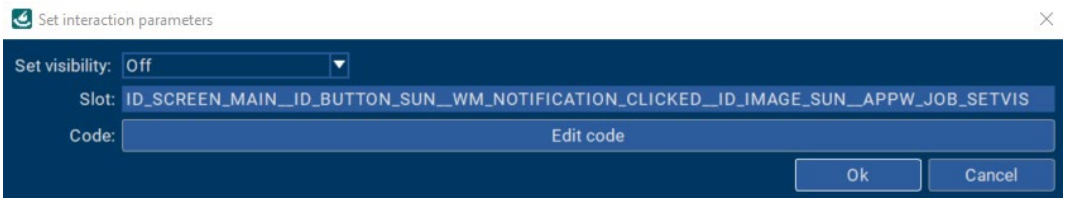

Figure 19. ID_BUTTON_SUN Interaction When Clicked

Figure 20. ID_BUTTON_SUN Interaction When Released

3.4 Add emWin Widget to AppWizard Project

You may need to use an emWin widget that is not yet supported by the AppWizard or need to create one in your custom code. The AppWizard allows that capability via the emWin API calls.

The **Logging Panel** in this graphics application features a Logging dialog created by using the Multiline Text widget.

The steps to add an emWin widget to the AppWizard project are as follows:

- Create an emWin widget by using emWin APIs in the slot routine for the AppWizard screen in the CustomCode folder.
- Handle GUI events/messages if needed via slot routines in the file `<ScreenID >Slots.c` located in the `\AppWizard\Source\CustomCode` folder.
- Figure 21 shows the function that creates the Multiline Text widget by using the `MULTIEDIT_CreateEx` API and other APIs.

Figure 21. Adding Multiline Text Widget to AppWizard Application by using emWin APIs

4. Understanding the Graphics Application

While the HMI is certainly a large part of understanding any HMI application, there are many other areas that you must understand while developing with the Renesas FSP applications. These include how the project is physically structured in e² studio, how threads and thread resources are added to the project, how threads communicate, the state machine design, and how state data is shared among cooperating threads, especially the emWin thread.

4.1 Source Code Layout

Prior to diving into the actual application code, it is best to understand the overall source code layout of an FSP project first. Renesas FSP applications generally consist of two different types of code: your code and auto-generated code. The auto-generated code can be further broken down into two sub-categories: code that is auto-generated by the FSP and code that is auto-generated by AppWizard.

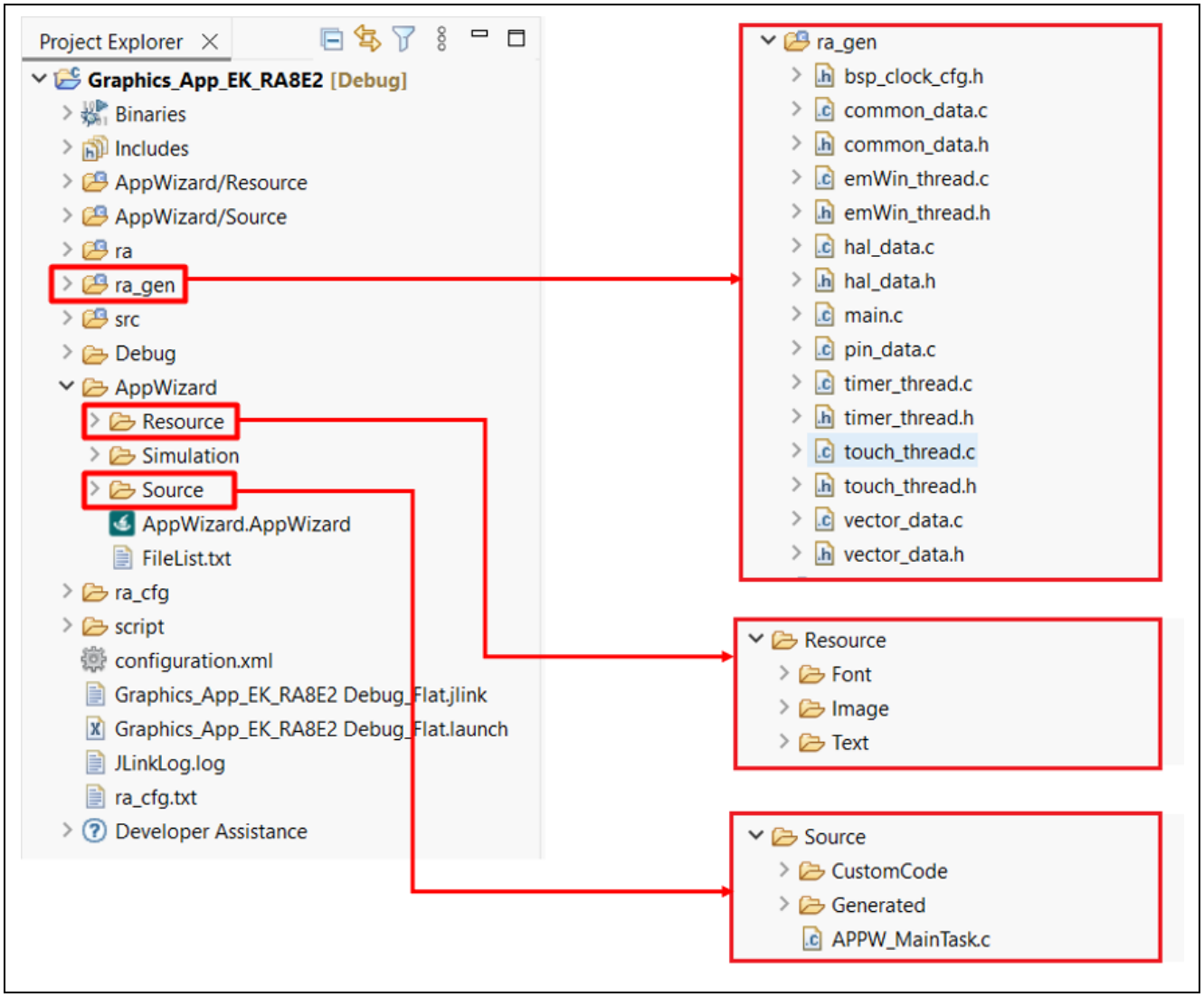

Figure 22. Graphics Application Project Source File Layout

Figure 22 shows the source code layout for the EK-RA8E2 board. FSP auto-generated code is in the `ra_gen` folder, AppWizard auto-generated code is highlighted in the `Generated` folder, and the code you generated is in the `CustomCode` folder.

Your generated code `/AppWizard/Source/CustomCode` is mainly used to handle HMI events. Your code in the `/src` folder is related to MCU peripherals and other functionalities.

4.2 Placing AppWizard Resources in External Flash Memory

When internal flash memory is insufficient, GUI resources generated by AppWizard can be stored in external OSPI flash to conserve internal memory or expand available storage.

In the AppWizard project, after exporting the generated code, all AppWizard resources are located in the `AppWizard/Resource/` directory. The `AppWizard/Resource/Image/` folder contains several `*.c` files, each defining a C array const for an image, wrapped with a specific prefix and suffix. This application demonstrates the way to store an image in an external flash using the linker section feature in e²studio. To use this project feature in e² studio, refer to the example of entries shown below.

After exporting the AppWizard project, all associated GUI resources are placed under the `AppWizard/Resource/` directory. Specifically, image assets are located in the `AppWizard/Resource/Image/` folder, where each image is represented as a `const` C array defined in a separate `*.c` file. These arrays are typically wrapped with a predefined prefix and suffix for consistency and linkage.

To reduce internal flash usage or support larger graphical assets, this application demonstrates how to store image data in external OSPI flash memory. This is achieved by leveraging the **linker section mapping** feature available in e² studio.

Refer to the example entries shown below to correctly configure and use this feature within your project.

1. Double-click on **"configuration.xml"** in the application project example. Click **Linker Sections** tab.

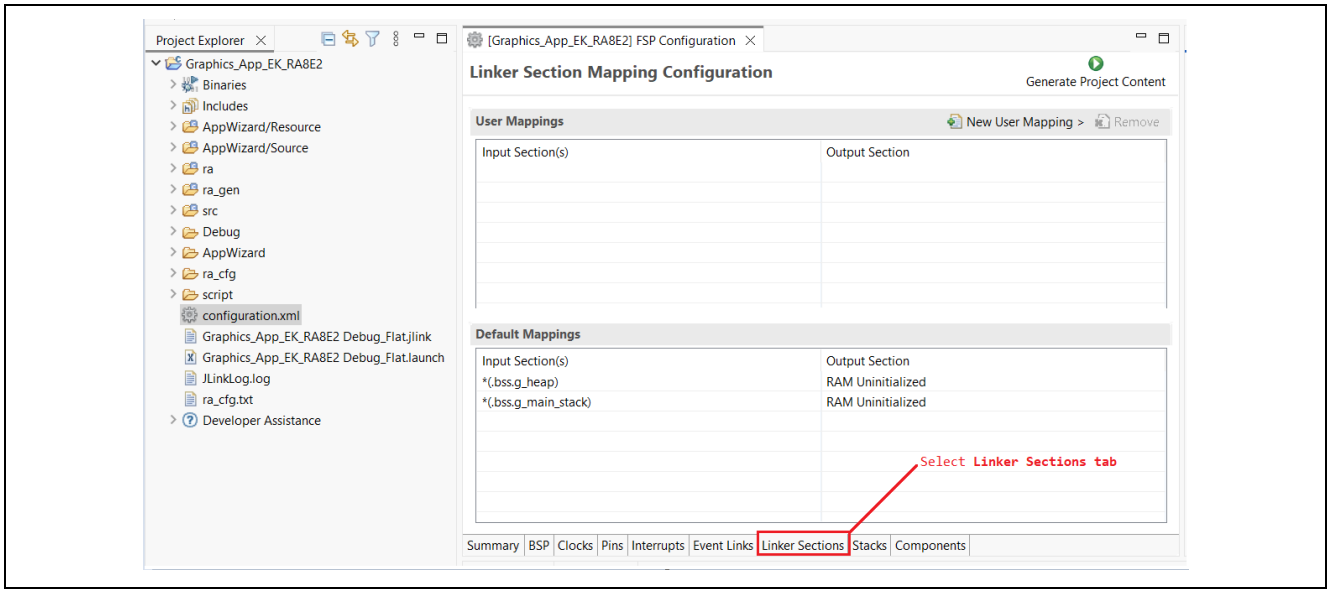

Figure 23. Linker Section Feature in e² studio

2. In the **User Mappings box, click on New User Mapping.** Choose **OSPI0_CS1 Constant Data** from **OSPI0_CS1.**

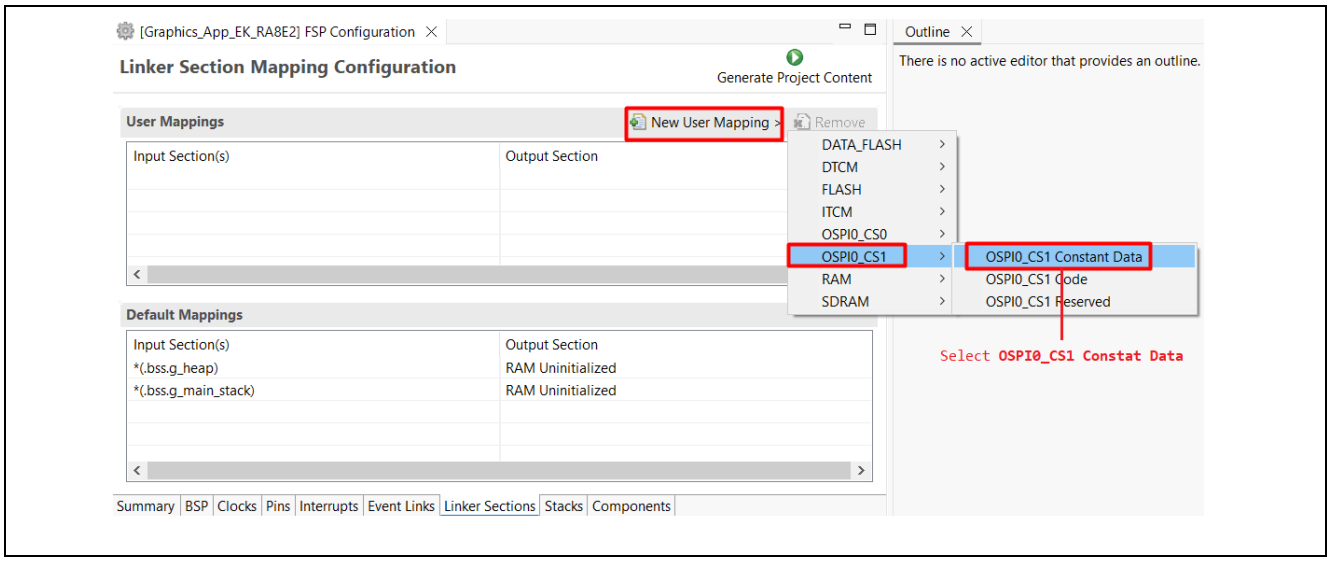

Figure 24. Example of Store Const C Array Buffer Image into the OSPI device on the EK-RA8E2

3. Enter the input section name or glob pattern for the new mapping.

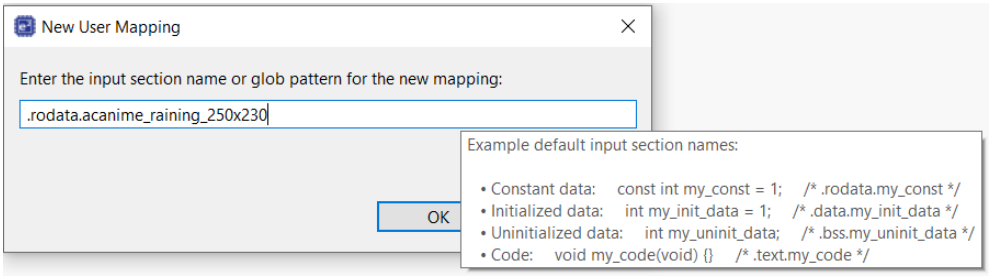

Figure 25. Example of an Input Section Name Used for Mapping Constant Data

Enter the input section name corresponding to the type of data or code.

For example, with a constant image "`buffer acanime_raining_250x230`", specify the input section as: `.rodata.acanime_raining_250x230.`

Repeat steps 2 and 3 for each additional image buffer that needs to be placed in external flash memory. As illustrated in Figure 26, all image resources will be placed within the memory space of "`Graphics_App_EK_RA8E2`" application.

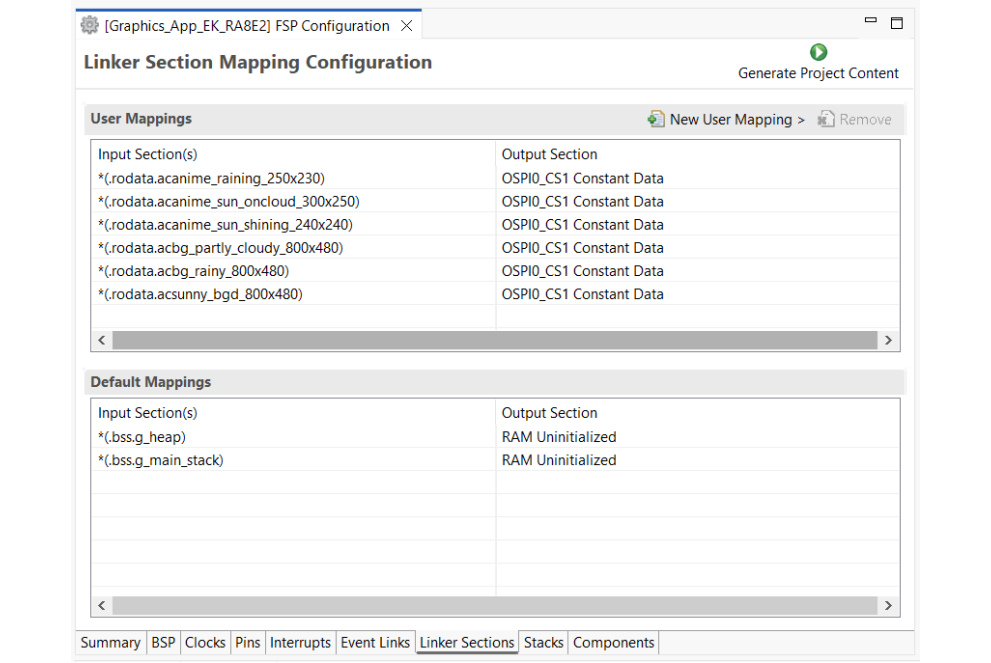

Figure 26. Example of Input Section Name in Graphics_App_EK_RA8E2 Application.

To access GUI resources stored in external OSPI flash, the OSPI_B module must be properly configured and the external OSPI flash device initialized. For detailed steps, refer to section 5.4.1.

4.3 Improve Performance by Utilizing Cortex®-M85 Core Data Cache

In the default configuration for RA8 devices, the FSP always enables the Cortex-M85 Instruction Cache (I-Cache). The FSP also allows for the optional enabling of the Cortex-M85 Data Cache (D-Cache) in the BSP configuration settings, and it is disabled by default. To use D-Cache, enable it in the BSP configuration settings.

Double-click on configuration.xml, select the **BSP** tab. **Enable Data cache** in **RA8E2 Family** > **Cache settings**

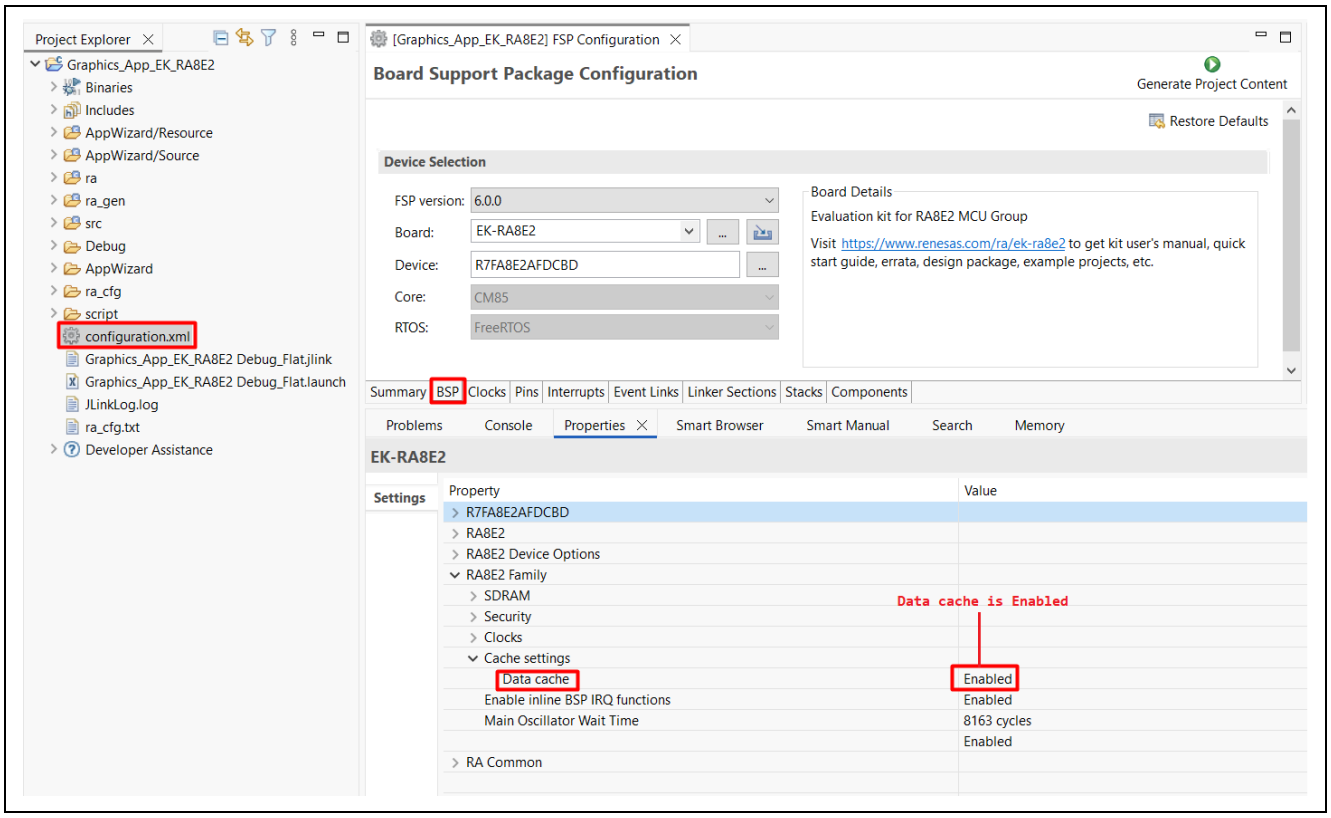

Figure 27. Example of Enable Data Cache in Application

Cache coherence should be considered when using any type of cache in a system. For further details, refer to the Cortex-M85 Caches documentation.

In this application code, the `payload` buffer for touch input at `/src/touch_gt911/touch_gt911.c` should be considered placed in the RAM uncached region.

Follow section 4.2 to add the **Input section name** `.bss.payload` to **New user mappings > RAM > RAM Uninitialized (Uncached).**

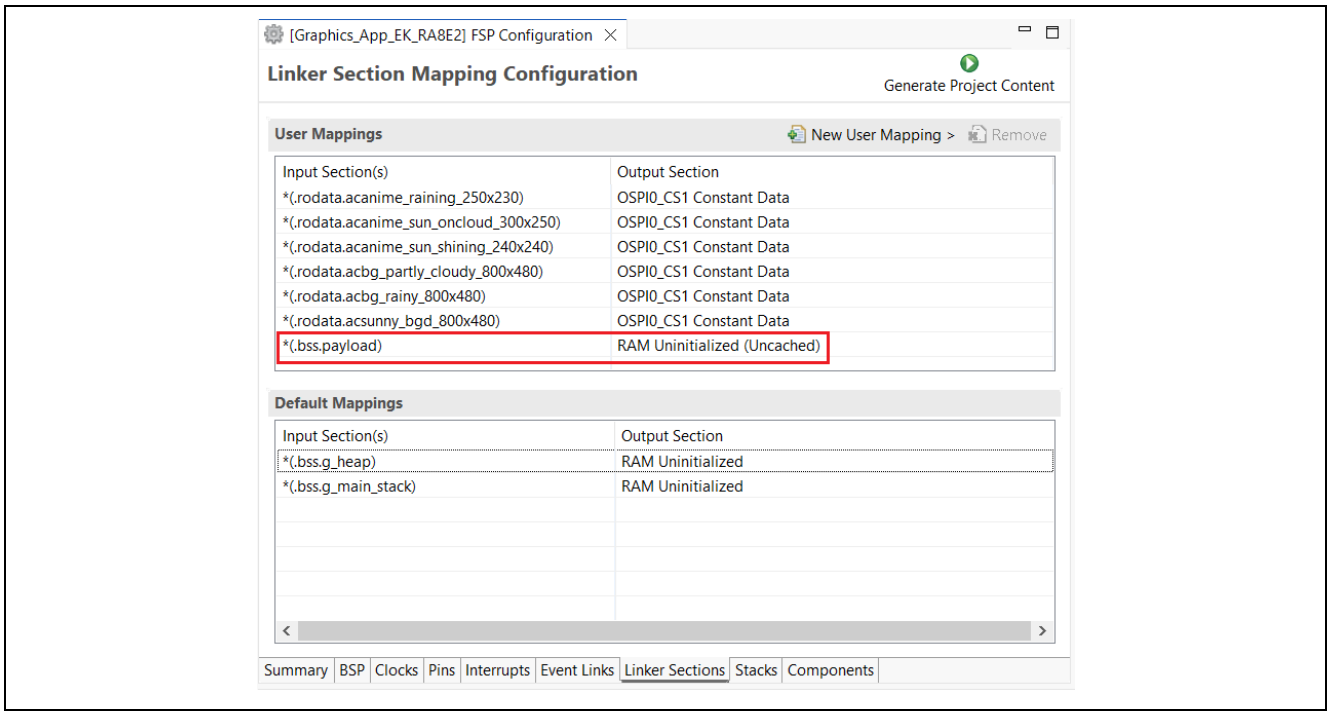

Figure 28. Example of Input Section Name *(.bss.payload) in RAM Uninitialized Uncached Section.

4.4 Application Block Diagram

As mentioned, the graphics application consists of two panels: the **Weather Panel** and the **Logging Panel**. The two application panels interface with the graphics framework through interaction, such as touch events and data (variables) changes. They communicate with FSP HAL drivers to send and receive touch sensing data, GPT PWM duty cycle, and RTC date and time.

The graphics framework includes the AppWizard framework, emWin library, emWin RA port, and interfaces with several HAL drivers such as GLCDC and D/AVE 2D. Figure 29 shows the application diagram.

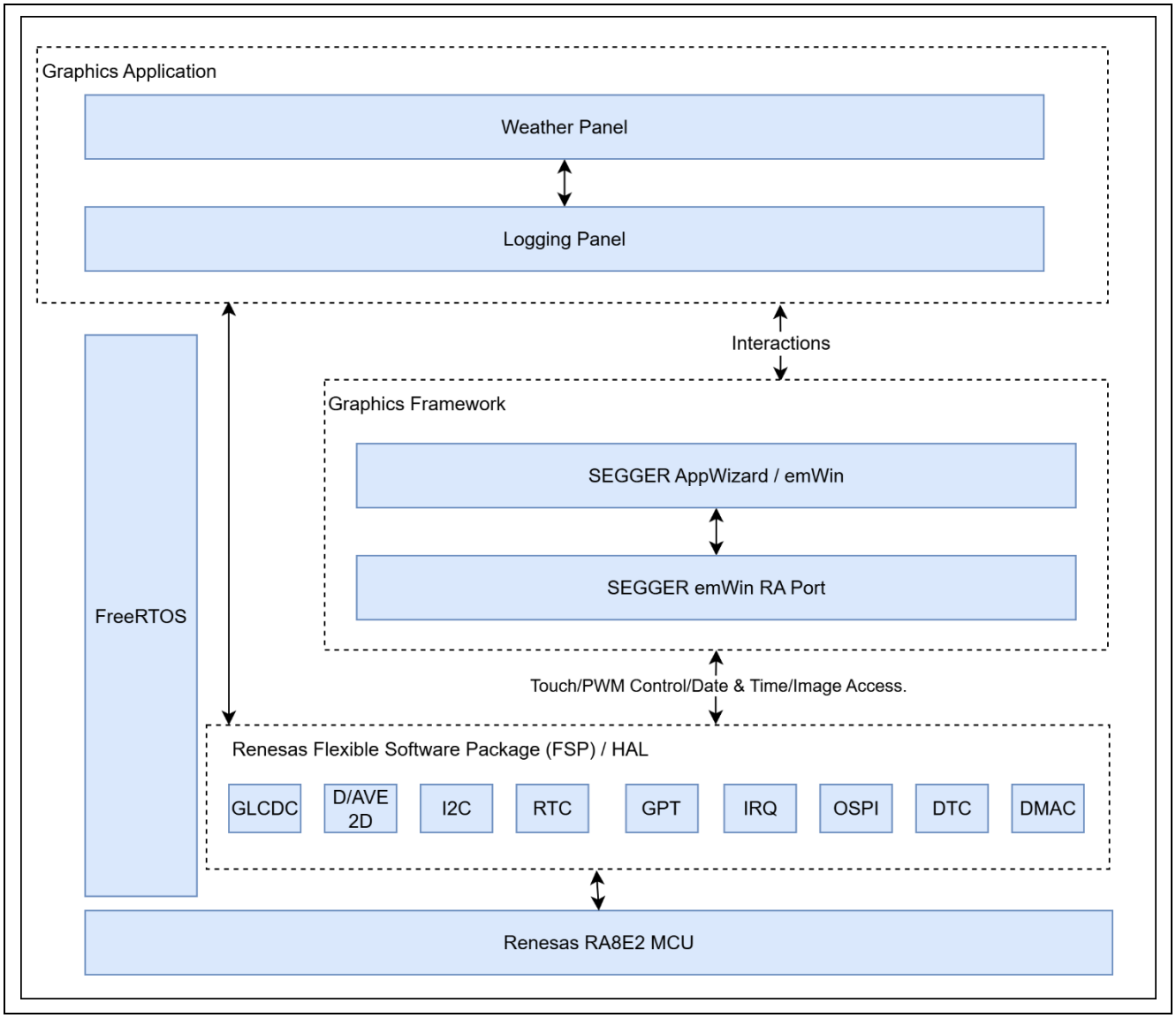

Figure 29. Application Block Diagram

4.5 Thread Overview

As mentioned in the introduction, the graphics application is a multi-threaded application that runs under FreeRTOS. There are two types of threads found in an FSP application: those created by you and those created automatically to support the operation of FSP. While it is obvious what threads you created, it is not always obvious what threads are created by FSP. The graphics application uses both user-created threads and FSP threads. Threads communicate through the emWin-type events using AppWizard and emWin APIs. The emWin thread processes data and touch events that are sent by the Touch thread and Timer thread. The FSP Configuration section details how to add your threads to your application. Figure 30 shows a high-level diagram of the threads and event flow in the graphics application. Notice the distinction between your threads and FSP threads.

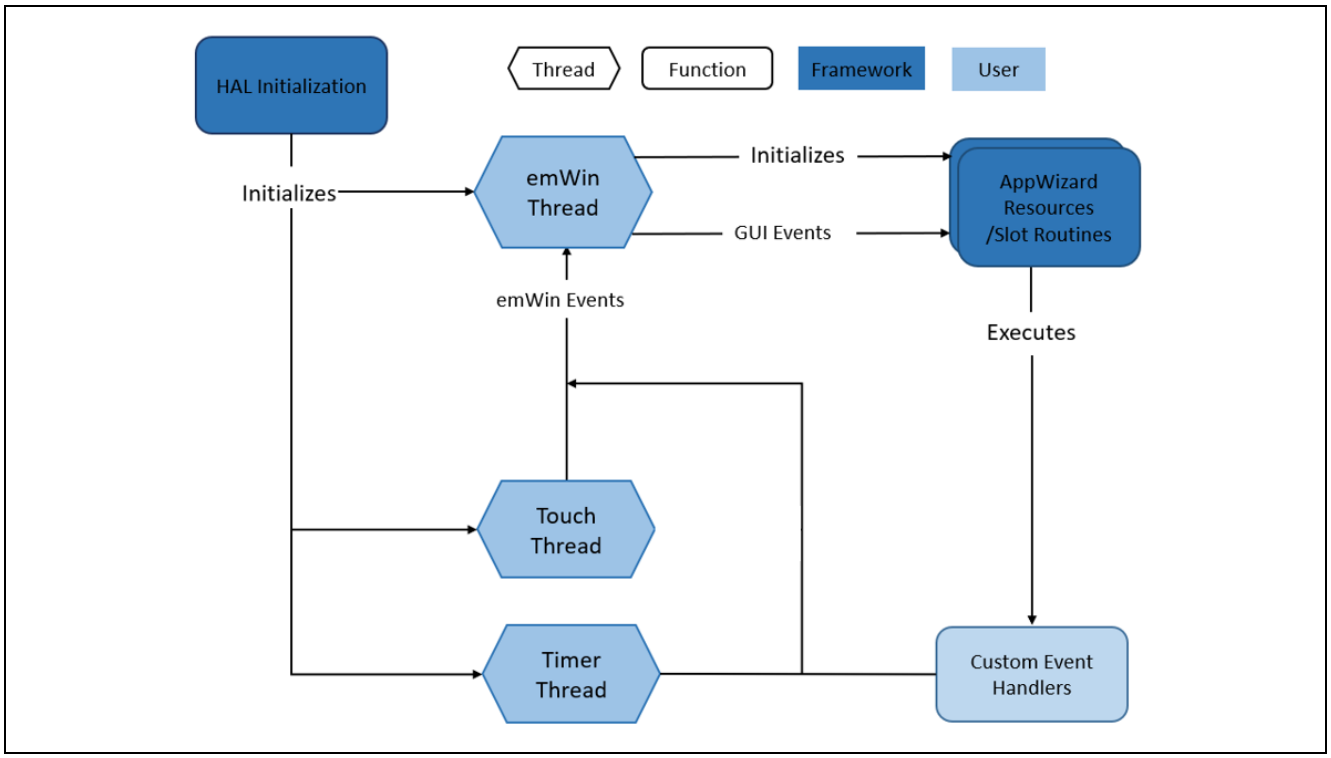

Figure 30. Graphics Application Event Flow

4.5.1 emWin Thread

The emWin thread is an HMI thread that initializes various services and resources used by the graphics application. Once this initialization is complete, the emWin thread processes touch events and window messages. If any of these inputs result in a change to the system state, the emWin invokes the AppWizard Slot routines, which are the callback routines, resulting in changes to the graphical HMI. The flowchart in Figure 31 gives the high-level view of the emWin Thread.

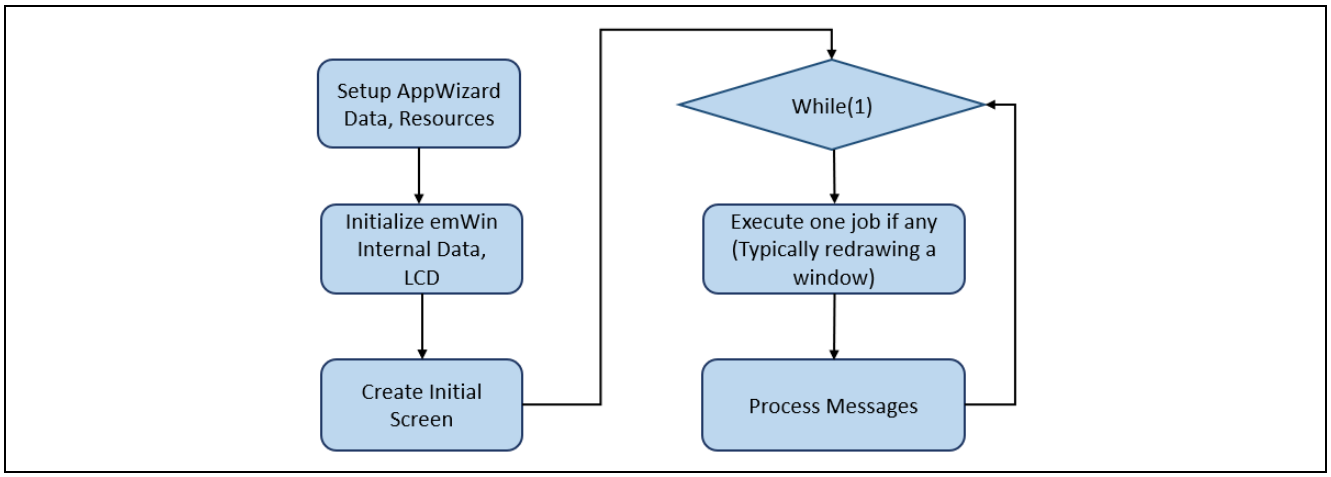

Figure 31. High-Level View of the emWin Thread

4.5.2 Touch Thread

A separate touch thread is created to read the touch sensor data. The touch sensor IC signals an event, such as a user interaction on the LCD screen, by toggling a pin connected to the MCU. In response, the touch thread reads the information from the touch sensor IC registers. Figure 32 shows the flowchart of the touch thread.

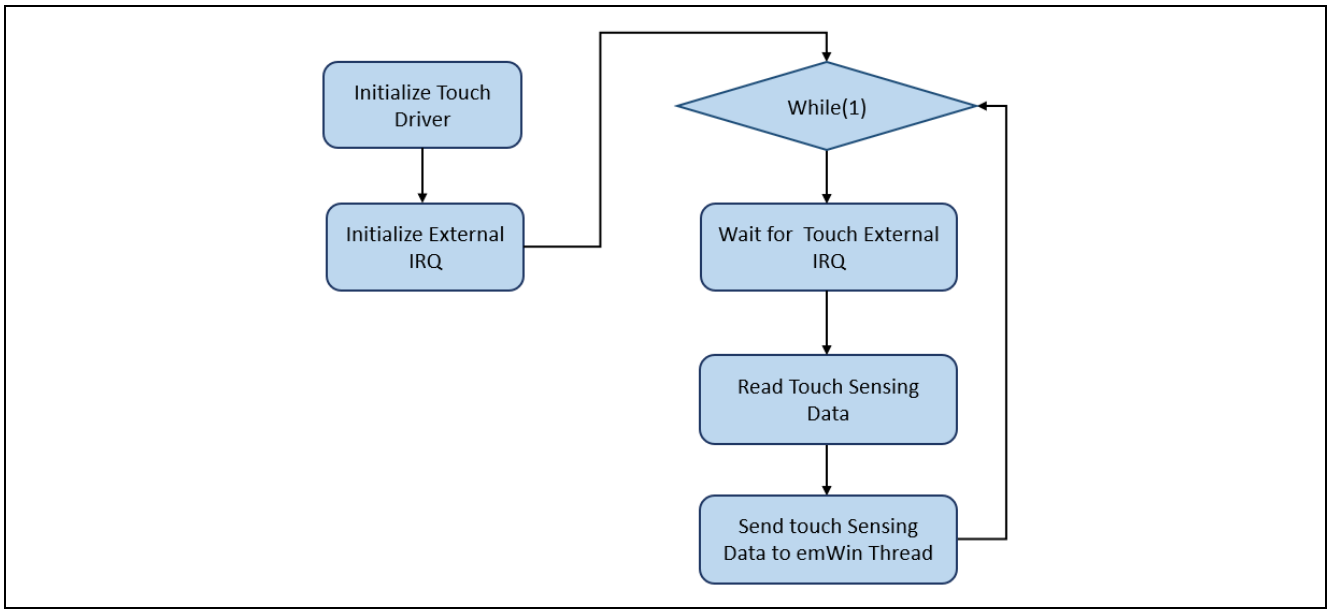

Figure 32. Touch Thread Flowchart

5. FSP Configuration

One of the first things you must do when writing an FSP application is to configure the FSP. To properly configure the FSP, you must have detailed knowledge of both the software design that you will be implementing, along with the specific hardware it will be running on. For the hardware, this includes the types of peripherals to be used on the hardware, the pins they are mapped to, whether they are internal or external to the MCU, and so on. From the software perspective, you need to decide how many threads will be used, which threads need access to what hardware components, and what additional software objects, like semaphores and queues, each thread will require. Once you have this information, you will be ready to successfully configure the FSP for your specific application needs.

In the graphics application, the FSP configuration is stored in a file named `configuration.xml`. Double-clicking on this file brings up the **RA Configuration** tab for the project.

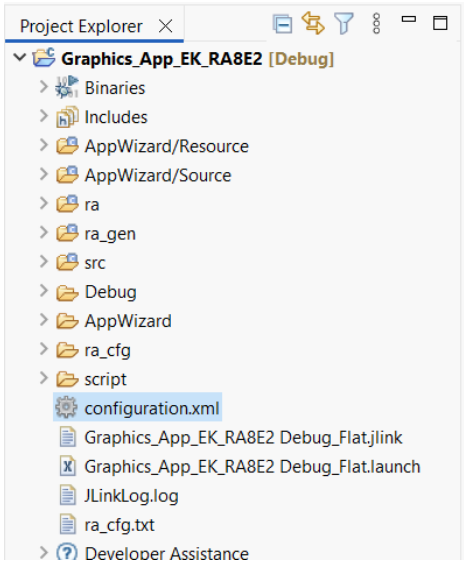

Figure 33. configuration.xml on the Project Plane

When a project is built from scratch, this configuration tab is where you will perform the initial configuration of the FSP. As you can see in Figure 34, the RA Configuration pane contains a **Summary** tab that highlights configured items and the software components currently selected for this project. In addition to the Summary tab, several other tabs are available to help you configure the FSP for your specific application.

For the purposes of this application note, we will highlight a few of the details of the FSP configuration, such as emWin, the r_glcdc driver, the touch controller, and the PWM timer, as they pertain to the graphics application. For additional details, refer to the Renesas Flexible Software Package (FSP) User's Manual on how to configure the FSP.

When you have finished configuring the project, click the **Generate Project Content**, green arrow button above the summary screen to generate all necessary files for your FSP components.

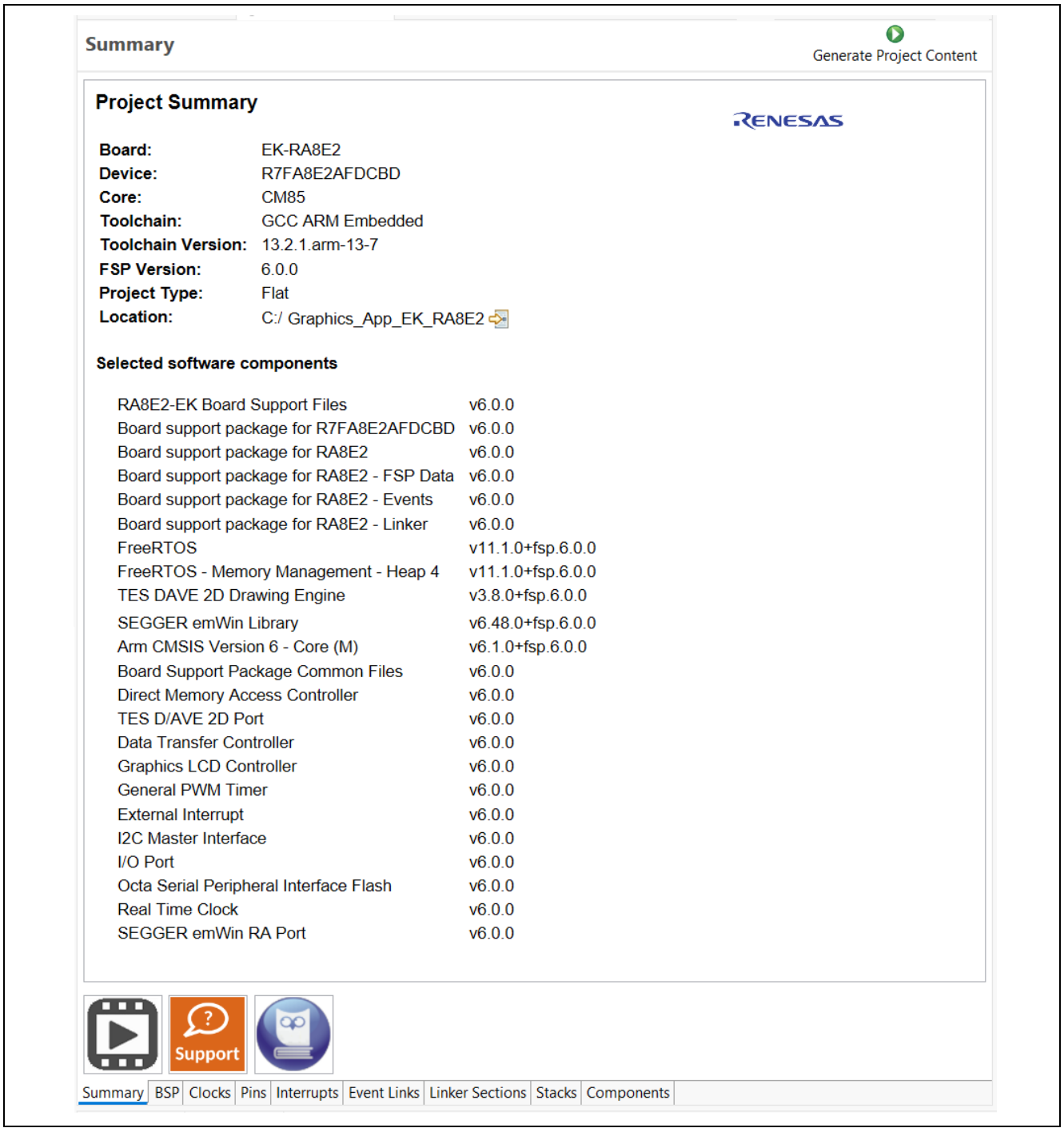

Figure 34. Summary of the Graphics Application Configuration

5.1 Components Tab

Even though the **Components** tab is the last tab showing, it is one of the first things you should configure. Selecting components first makes them available in subsequent operations, such as mapping hardware resources to specific threads in the **Stacks** tab. One of the advantages of FSP is that it will only compile the components you choose, thereby reducing the size of your overall application. As shown in Figure 35, the components are broken down into several categories.

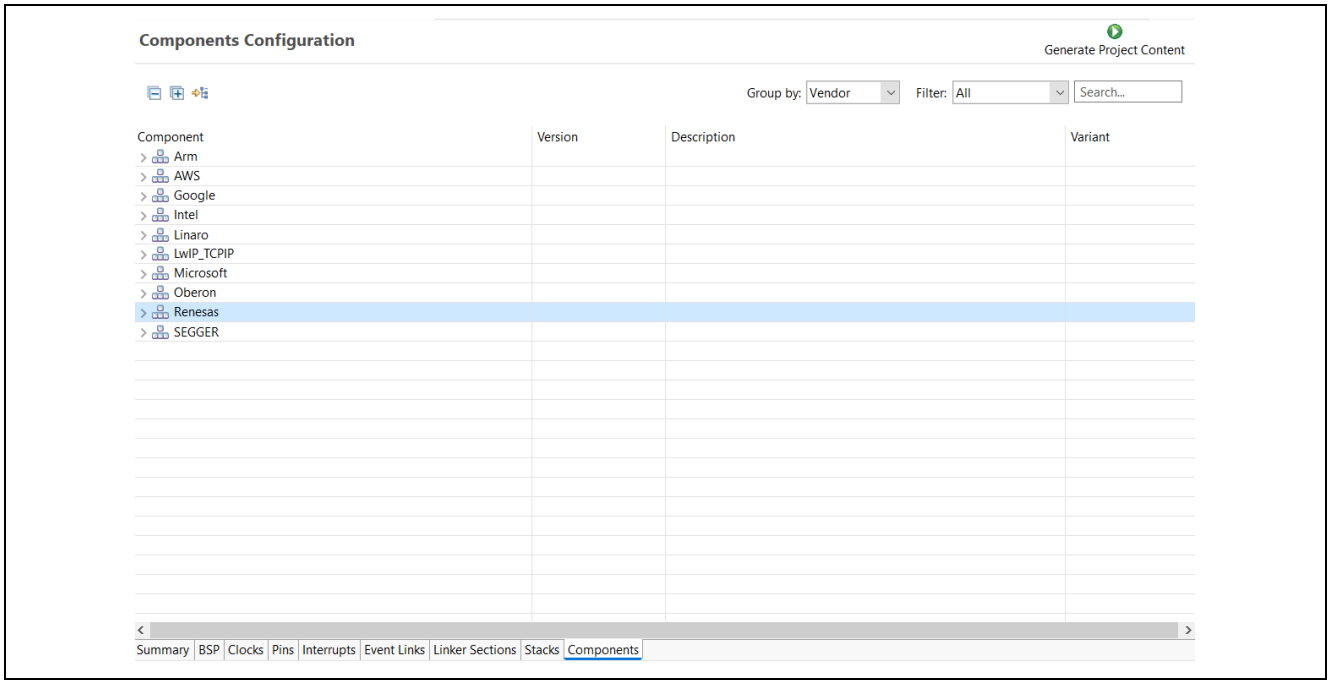

Figure 35. Components Tab Categories

You may expand any of the categories by clicking the arrow to the left of the category name.

The following table highlights the selections used for the graphics application.

Table 2. Components Used in the Graphics Application

Category	Component	Version	Description
BSP	`Ra8e2_ek`	6.0.0	RA8E2-EK Board Support Files
CMSIS	`CoreM`	6.1.0+fsp.6.0.0	Arm CMSIS Version 6 - Core (M)
Common	`fsp_common`	6.0.0	Board Support Package Common Files
GUI	`emWin`	6.48.0+fsp6.0.0	emWin Library
HAL Drivers	`r_drw`	6.0.0	TES D/AVE 2D Port
	`r_dtc`	6.0.0	Data Transfer Controller
	`r_glcdc`	6.0.0	Graphics LCD Controller
	`r_icu`	6.0.0	External Interrupt
	`r_iic_master`	6.0.0	I2C Master Interface
	`r_ioport`	6.0.0	I/O Port
	`r_rtc`	6.0.0	Real-Time Clock
	`r_dmac`	6.0.0	Direct Memory Access Controller
	`r_ospi_b`	6.0.0	Octa Serial Peripheral Interface Flash
	`r_gpt`	6.0.0	General PWM Timer
Heaps	`heap_4`	11.1.0+fsp.6.0.0	FreeRTOS - Memory Management – Heap 4
Middleware	`rm_emwin_port`	6.0.0	SEGGER emWin RA Port
RTOS	`FreeRTOS`	11.1.0+fsp.6.0.0	FreeRTOS
TES	`dave2d`	3.8.0+fsp.6.0.0	TES DAVE 2D Drawing Engine

5.2 Stacks Tab

The **Stacks** tab is where you can add and configure the threads that the FSP automatically creates for your application. You define a new thread by clicking the **New Thread** button and then entering a unique name for your new thread. Once you add a new thread, you must define the modules that the thread will use, along with any thread objects that will be used by your thread.

As an example, if you click the **Threads** panel and then single-click on the **emWin Thread**, you should see something like the screen capture shown in Figure 36. This shows that the emWin thread requires multiple modules, such as the GLCDC driver, which is used to control the LCD screen on the graphics expansion board of the EK-RA8E2 kit.

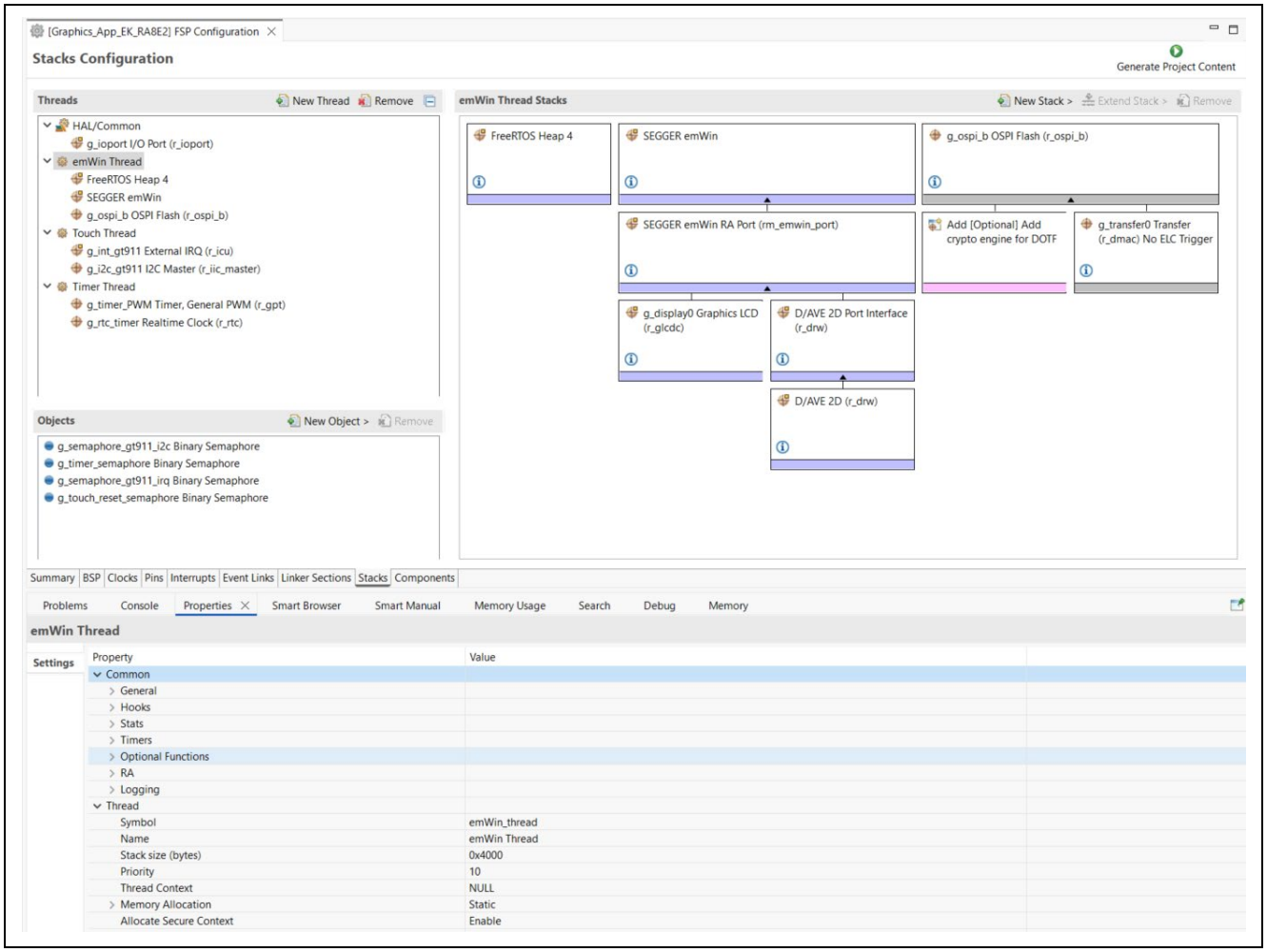

Figure 36. emWin Thread Properties and Modules Used for the Graphics Application

You can add additional modules to any thread by clicking the **New Stack** button. If you have chosen the appropriate components prior to adding modules to your threads, you should not receive any errors. As an example, Figure 37 shows you how to add a GPT timer to the Timer Thread. The timer is added by choosing **(+) New Stack** > **Timers** > **Timer, General PWM (r_gpt**)

When a middleware module such as emWin is selected, placeholders for the additional required components, such as GLCD and D/AVE2D, are automatically added below. If the FSP detects an error when adding a module, it highlights the module name in red. You may examine the errors by hovering over the module name.

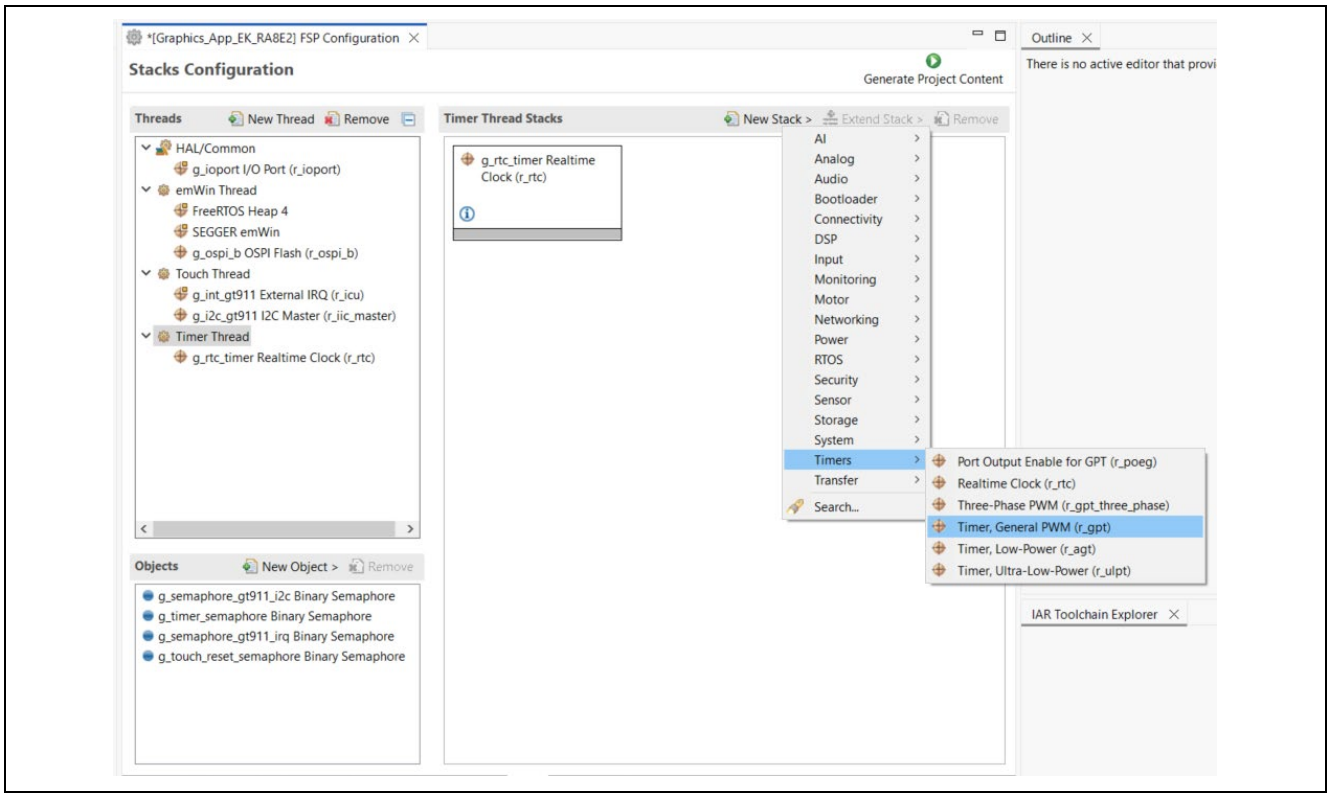

Figure 37. Adding r_gpt driver

5.3 Thread Objects

FreeRTOS supports various objects such as mutexes, queues, semaphores, and timers. In the **Objects** window, you will see that there are four semaphore objects, **g_touch_reset_semaphore**, **g_semaphore_gt911_i2c**, **g_timer_semaphore,** and **g_semaphore_gt911_irq** created for this application.

You can allocate additional thread objects by clicking on the **New Object** button next to the **Objects** window. As you can see in Figure 38, after clicking the **New Object** button in the **Objects** window, you will be presented with a drop-down list that will allow you to add the standard thread objects supported by FreeRTOS.

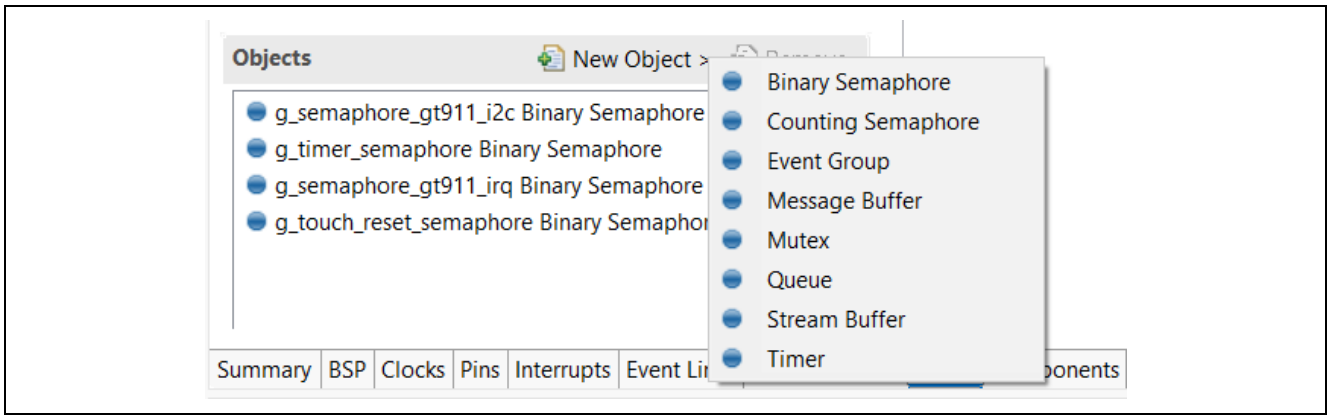

Figure 38. Objects window

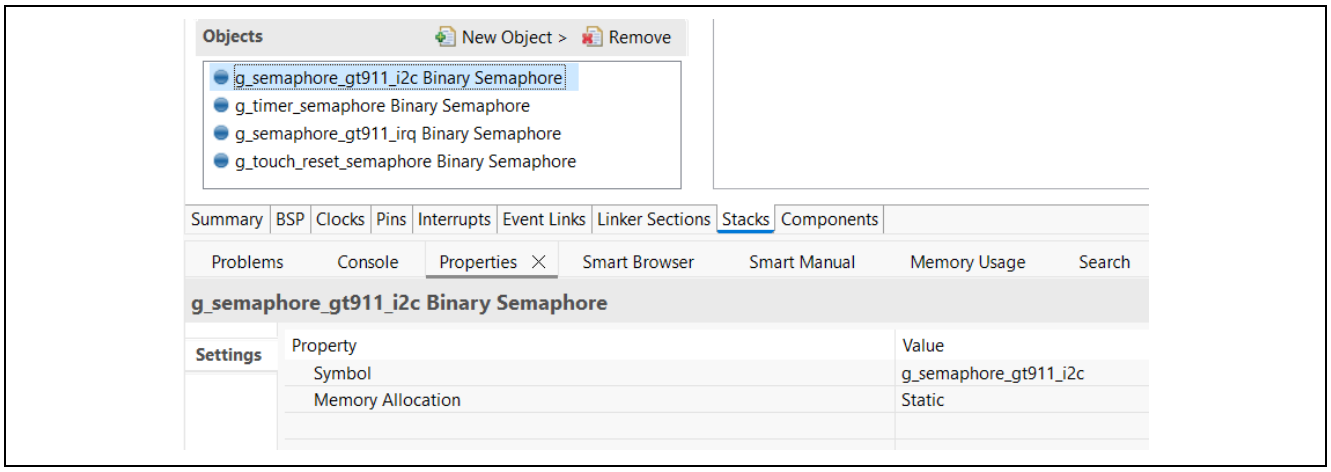

Figure 39. g_semaphore_gt911_i2c Properties

5.4 Module Configuration

Once you have added a module to your project, you need to configure its properties. The properties are dependent on the module(s) that you have added. Use the **Properties** tab to configure them. The graphics application adds the `r_glcd` driver module as part of the emWin stack. This module is used to configure the GLCDC peripheral of the Renesas RA8E2 MCU.

5.4.1 OSPI_B Configuration

To begin configuration, add the **g_ospi_b** stack to the **emWin Thread**, then open the **Properties** tab to access its settings.

The following section will walk through each configuration option available in the **r_ospi_b** configuration tab.

Channel CS1: On the EK-RA8E2 kit, the external OSPI flash device is connected to the OSPI_B controller using Chip Select 1 (CS1). The memory-mapped address space for CS1 is illustrated in Figure 40.

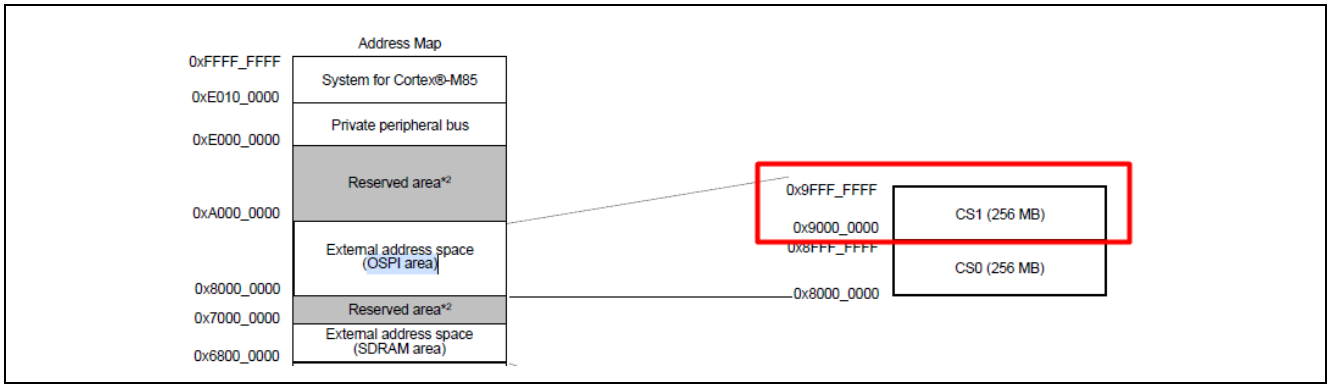

Figure 40. Address map for the OSPI area

The **Erase Sizes** and all commands in the **Initial Mode** or **High-Speed Mode**, such as **Read, Program, Write Enable, Erase, and Address Length**, in the **Command Sets** section must be configured according to the specifications of the external OSPI flash device.

Initial Mode: Communication with the external OSPI flash initially operates in 1S-1S-1S mode.

Refer to Figure 41 for the configuration example using an external OSPI flash on the EK-RA8E2 kit.

Additionally, the corresponding pins must be configured for the correct OSPI_B channel. The pin configuration is shown in Figure 42.

After completing the configuration of the r_ospi_b stack, the OSPI_B peripheral and the external OSPI flash device must be initialized before accessing any data stored in the external flash. Refer to the section 6.1.1 for initialization details.

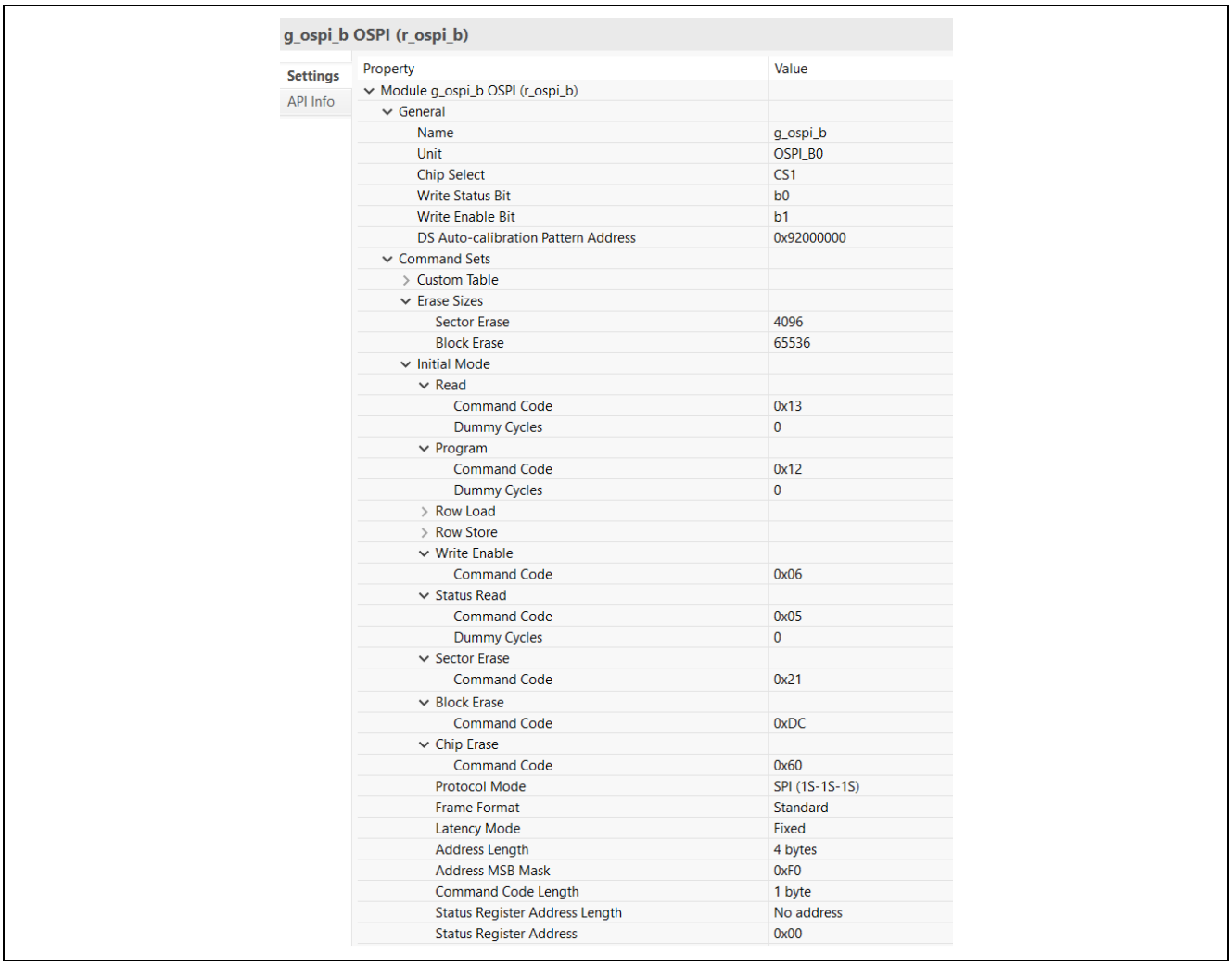

Figure 41. OSPI_B Properties Configuration using the Properties Tab

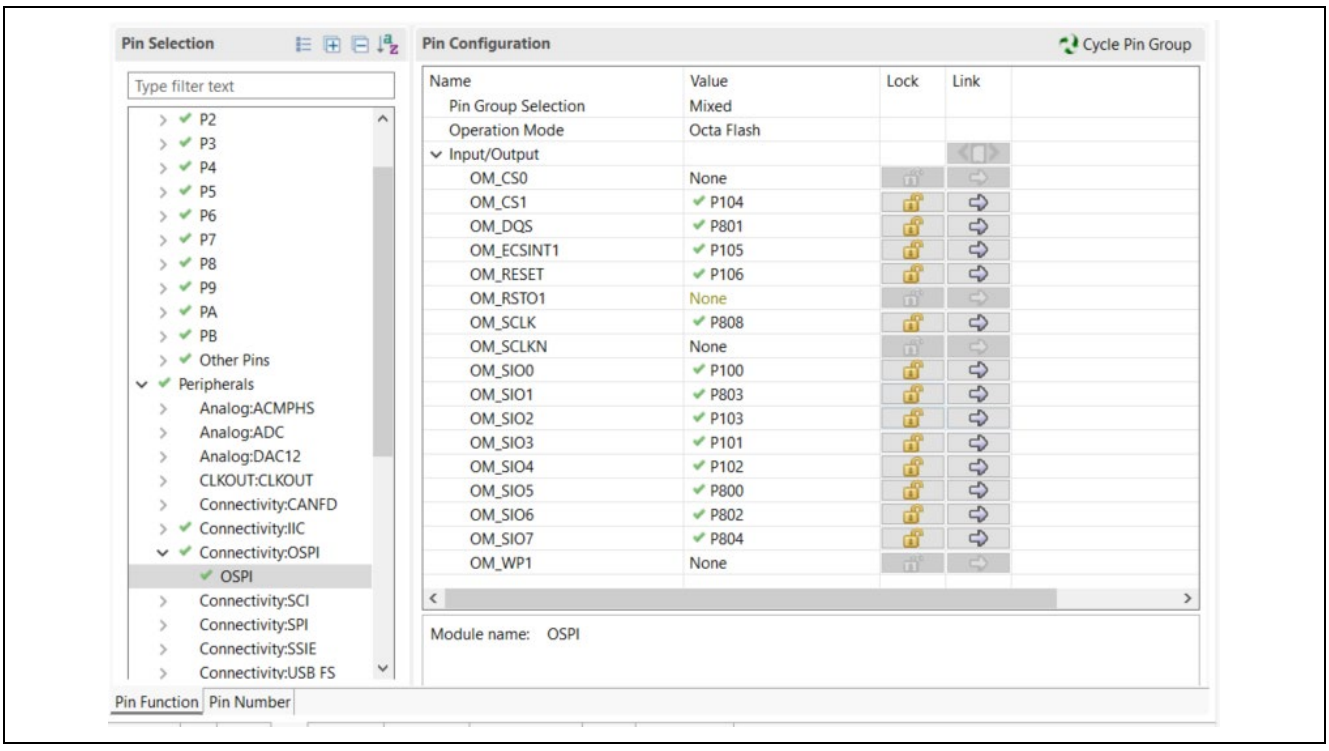

Figure 42. OSPI_B Pin Configuration

5.4.2 GLCDC Configuration

As you can see in Figure 43, selecting the **g_display0 Graphics LCD** on the **g_glcdc** module under the **SEGGER emWin Stacks** brings up the associated properties under the **Properties** tab. The first thing you will notice is that it is a lengthy list of properties within the module group. The module group is where you configure the GLCDC controller. These properties can be a bit daunting at first, but can be broken down. First, you will notice a few broad categories inside the module grouping.

- **Name:** The name given to this instance of the module **g_display0** by default.
- **Interrupts:** You set the Line Detect interrupt and other interrupts here.
- **Input:** This block of module properties defines the input to the graphics controller, most notably, the framebuffer name and the number of the framebuffers, the memory address where the framebuffer is located, and others.
- **Output:** This is the area where you define the output properties of the GLCD. This includes properties such as the total horizontal and video cycles, the active video cycles, both horizontal and vertical, front and back porch duration, and so on.
- **TCON:** You use these lines in conjunction with the **Pins** tab to map the Horizontal Sync (Hsync), Vertical Sync (Vsync), and Data Enable signals. You can specify the LCD panel clock divisor that divides the clock input into the GCLD. This division ratio currently ranges from 1/1 to 1/32.
- **Color Correction:** This is where you can add various levels of color correction, for example, brightness, contrast, and gamma, to your display. Color, contrast, and gamma correction of LCD screens are outside the scope of this application note, but this is the area where you would do that type of adjustment.

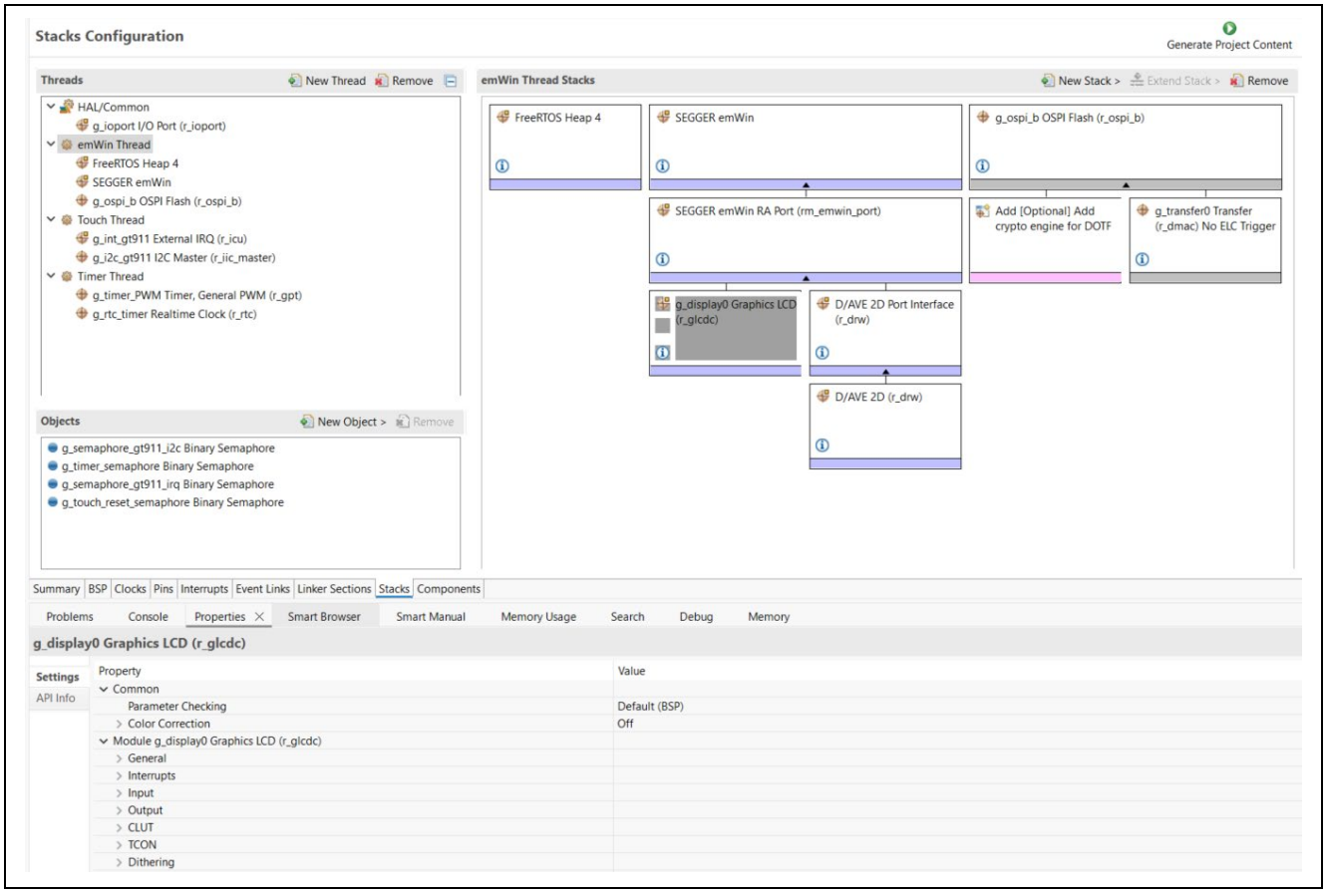

Figure 43. GLCD Properties Configuration using the Properties Tab

5.4.3 TCON Configuration

If you scroll down a little further in the **Properties** tab, you will see four TCON properties. One of these is associated with the **Panel clock division ratio**. This allows additional division of the pixel clock that is driven directly from the LCDCLK branch of the clock tree. The other three are associated with the LCD sync signals. These three signals can be confusing to new users, so how these signals map to the physical pins they are connected to is discussed here.

TCON	
Hsync pin select	LCD_TCON1
Vsync pin select	LCD_TCON0
Data enable (DE) pin select	LCD_TCON2
Panel clock source	Internal clock (GLCDCLK)
Panel clock division ratio	1/8

Figure 44. TCON Configuration for EK-RA8E2 Kit

To provide flexibility, the GLCD controller of the RA8E2 MCU provides two alternative pin groups. Each option uses different pins on the MCU to drive the data lines connected to the LCD display. It is up to the hardware designer to select pins from each group or combine them as needed. Picking one or the other may free up MCU pins that are necessary for some other part of the hardware design.

If you look at the schematics for the EK-RA8E2 kit, you can see the pin header for the LCD board. You will also notice the three pins connected to the sync signals, which are highlighted in red. The hardware designer may select data lines from either pin group or mix pins from both groups. For example, Pin Group B has been selected on the EK-RA8E2.

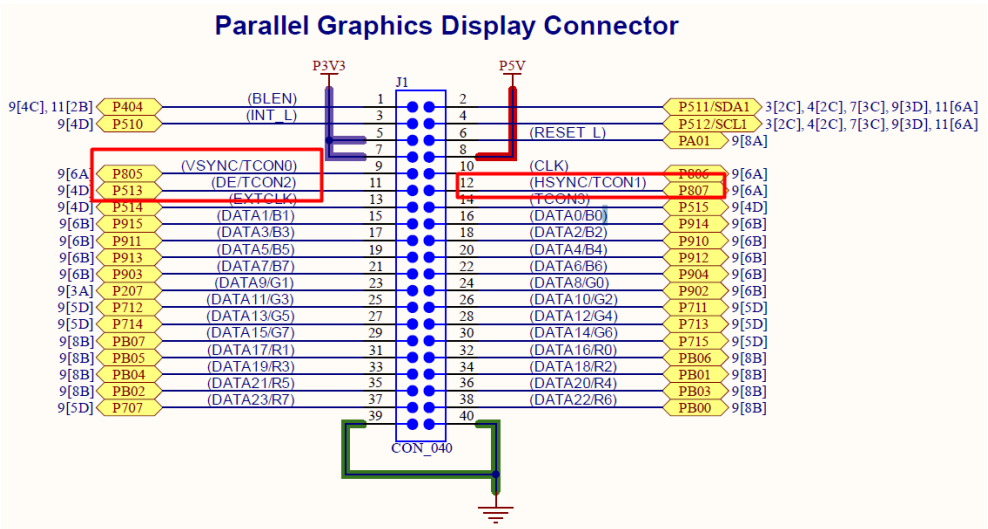

Figure 45. EK-RA8E2 LCD-Specific Signals from the Schematics

The easiest way to understand this is to go to the **Pins** tab in the RA Configuration. You will see selections for **Ports** and **Peripherals**, as shown in Figure 46. If you expand the **Peripherals** dialog, you will see all the various MCU peripherals that can be configured from this screen.

If you scroll down to the **Graphics: GLCDC** entry and click to expand it, you will see **GLCD Pin Group Selection**.

Notice that TCON0 is associated with the Port 8 Pin 05 (P805). On the schematic (P805), we see that it is connected to VSYNC, which is the vertical synchronization pin for this LCD screen. Referring to Figure 44, we see that TCON0 has been selected to drive the VSYNC signal.

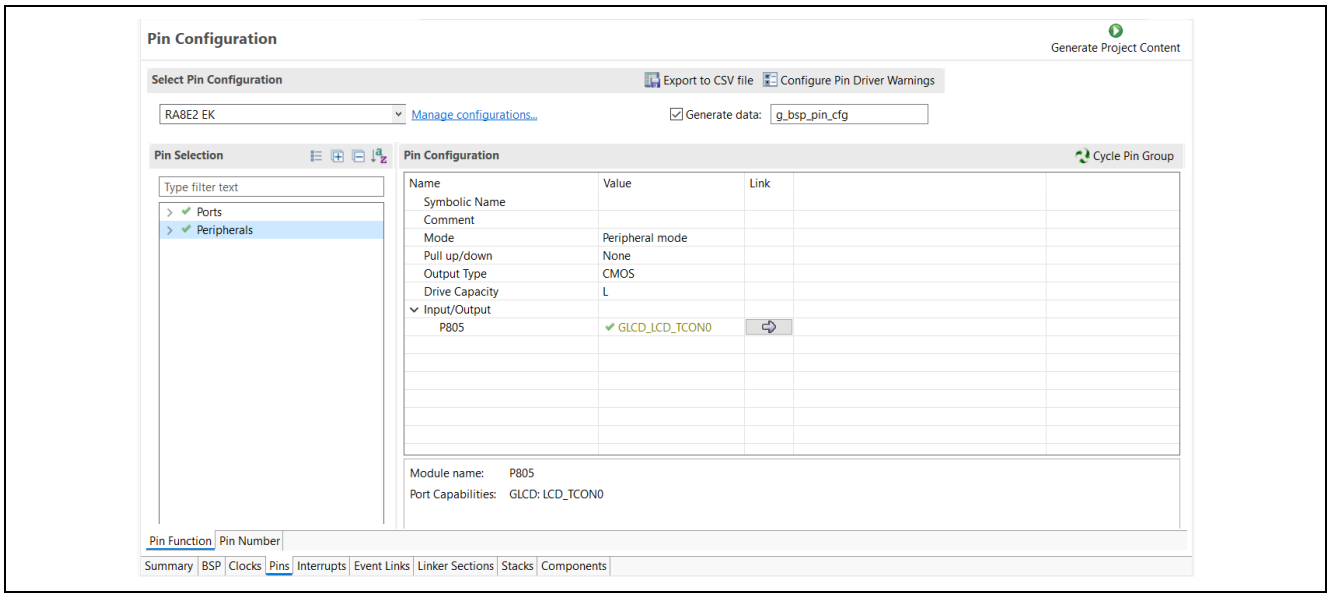

Figure 46. Pin Configuration Tab

If you look at all the LCD data lines, such as LCD_DATA0, and the pins they are connected to, they should match the pins they are connected to on the schematic. Clicking on the arrow to the right of the pin brings you directly to the associated **Pin Configuration** dialog, just as if you had selected the **Ports Group** and then the specific port and pin that you are interested in.

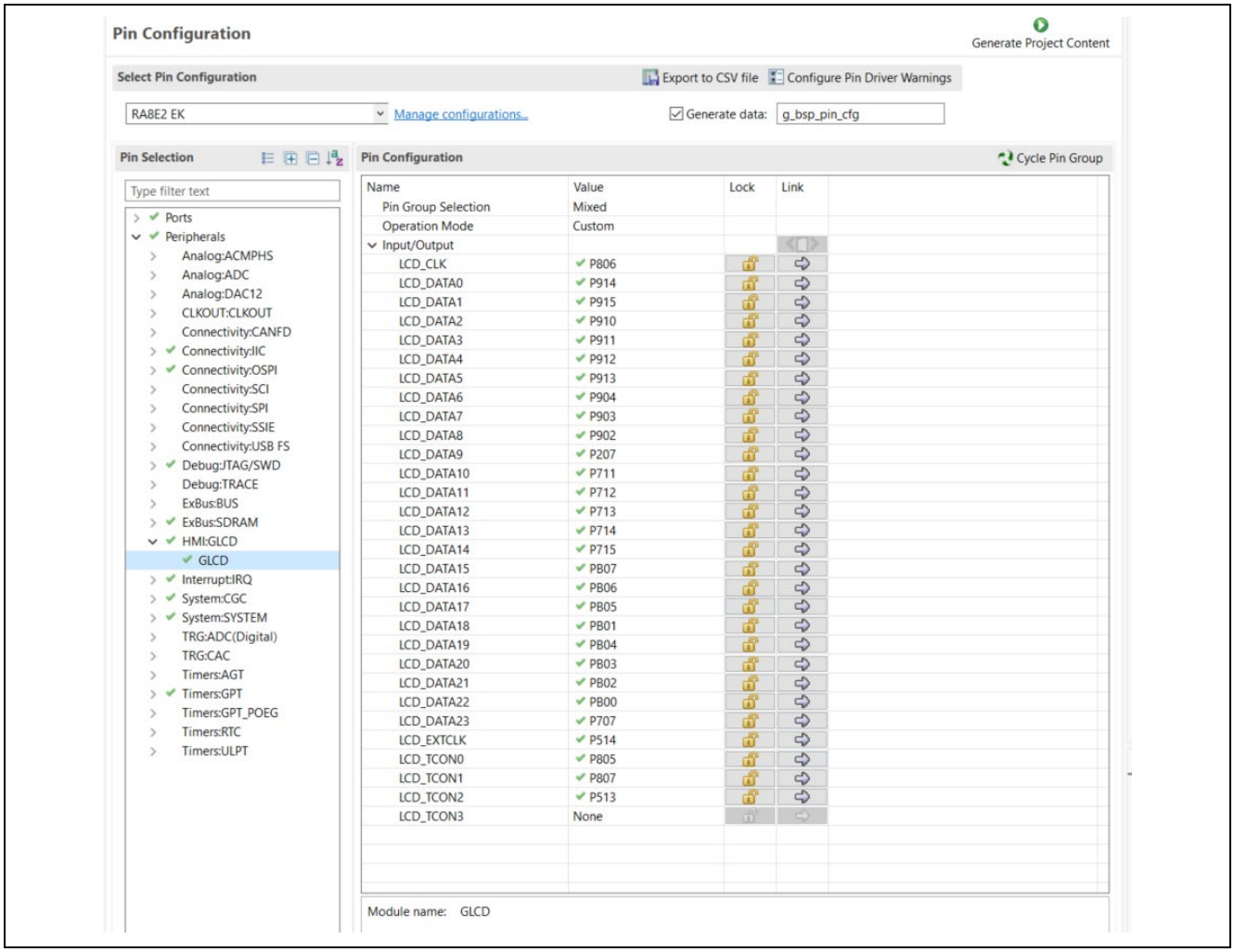

Figure 47. LCD Pin Configuration Using Configurator

For example, clicking on this arrow to the right of the LCD_TCON0 pin should bring you to the **Pin Selection Screen** that looks like Figure 48. Notice that the pin is appropriately set to **Peripheral mode**, **no Pull-up**, Drive Capacity should be adjusted (e.g., 'H') according to the desired I/O frequency, and **CMOS output type**. Clicking on the arrow button to the right of this screen brings you back to the associated peripheral screen.

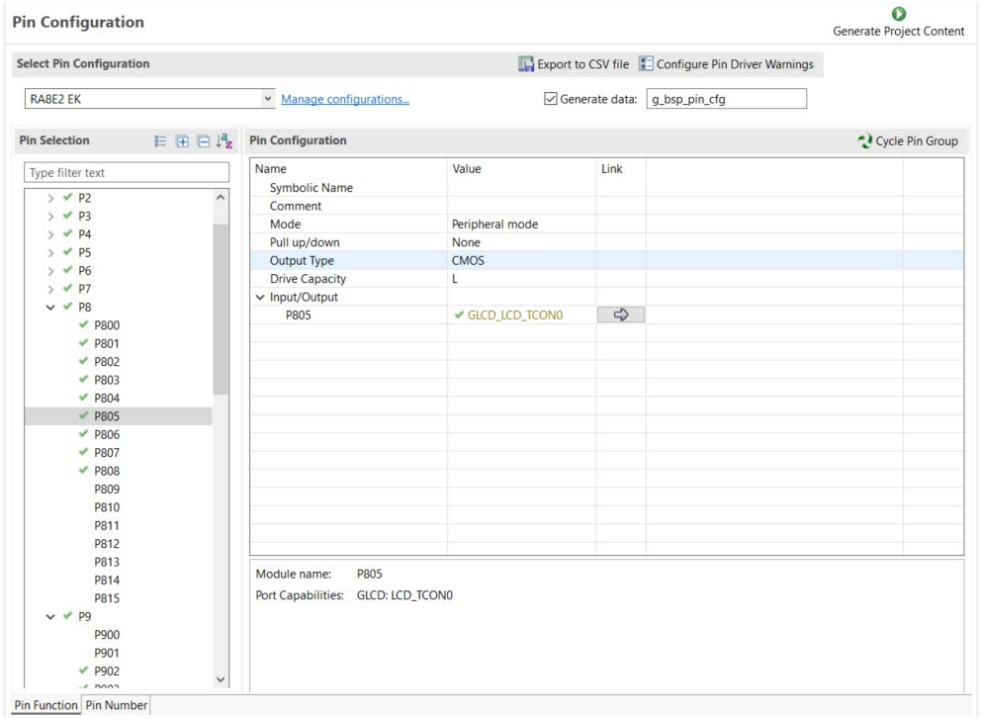

Figure 48. LCD_TCON0 Settings in Pin Selection window

5.4.4 Touch Controller Configuration

The touch event on the LCD screen is sensed by the RA8E2 MCU's external IRQ pin, and the touch sensor is read via the I2C master.

As shown in Figure 51, the interrupt signal of the Touch Controller on the LCD screen is connected to P510 on header J1 of the EK-RA8E2 board, which is the MCU IRQ channel 3. The `r_icu` and `r_iic_master` drivers are added to a Touch Thread to handle the IRQ channel 3 and I2C Master Channel 1, respectively.

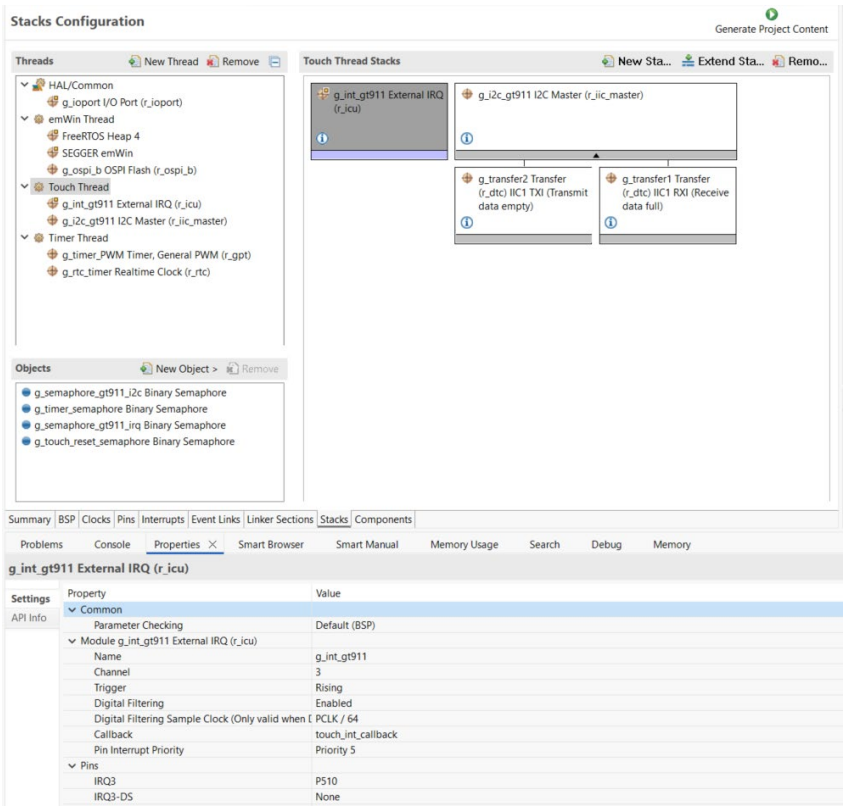

Figure 49. External Interrupt Configuration

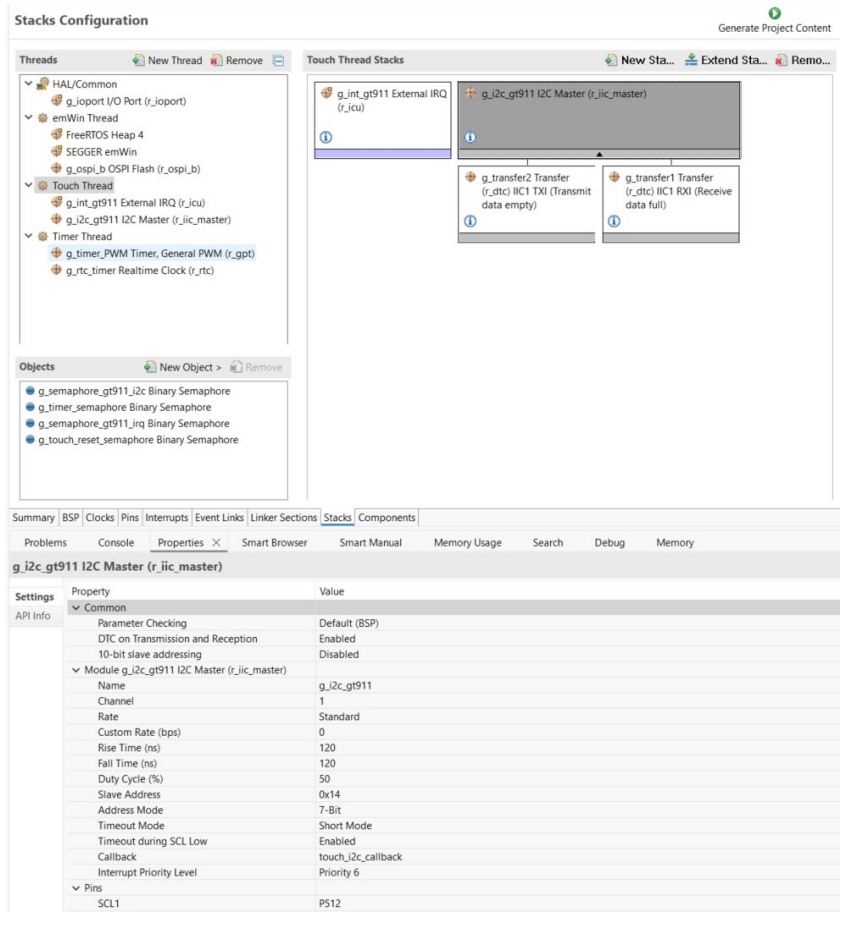

Figure 50. I2C Master Driver Configuration

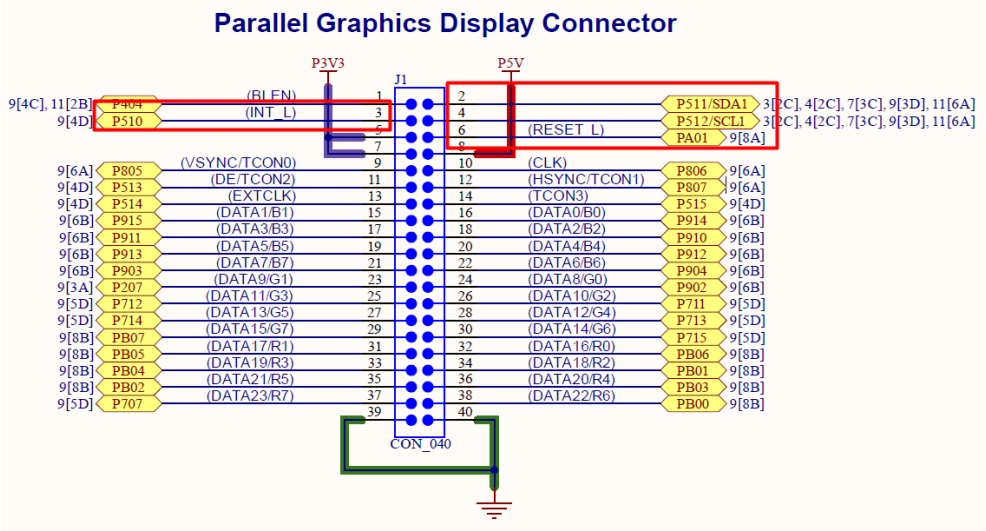

Figure 51. Touch Controller Signals

Note: When creating a project from scratch, you must add the touch driver to your project by copying the `touch_gt911` folder in this application note project to the new project. Go to **Project > Properties > C/C++ Build > Settings > GNU ARM Cross C Compiler > Includes** to add the include path.

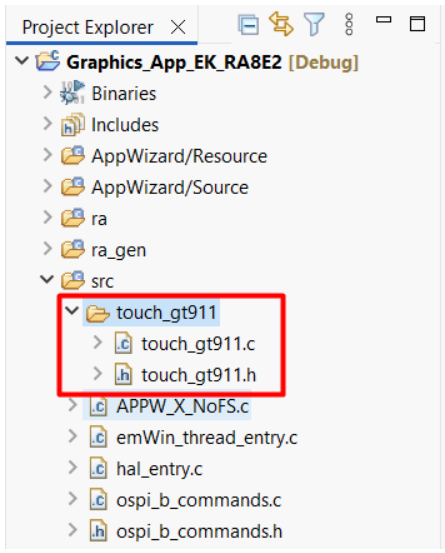

Figure 52. Touch Driver sources

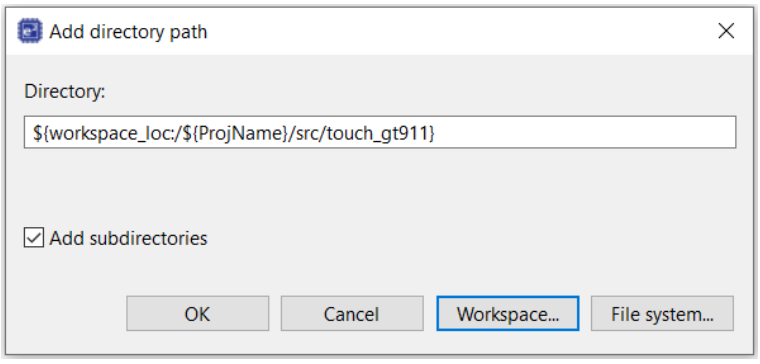

Figure 53. Add Touch Driver Folder to the List of Include Paths

5.4.5 PWM Configuration

The LCD_BLEN signal (Backlight Enable), which is connected to the P404 on the RA8E2 MCU, is configured in PWM mode to control the intensity of the LCD backlight. Figure 54 shows an excerpt from the Graphics Expansion board schematic, which shows the LCD_BLEN signal connected to the backlight controller.

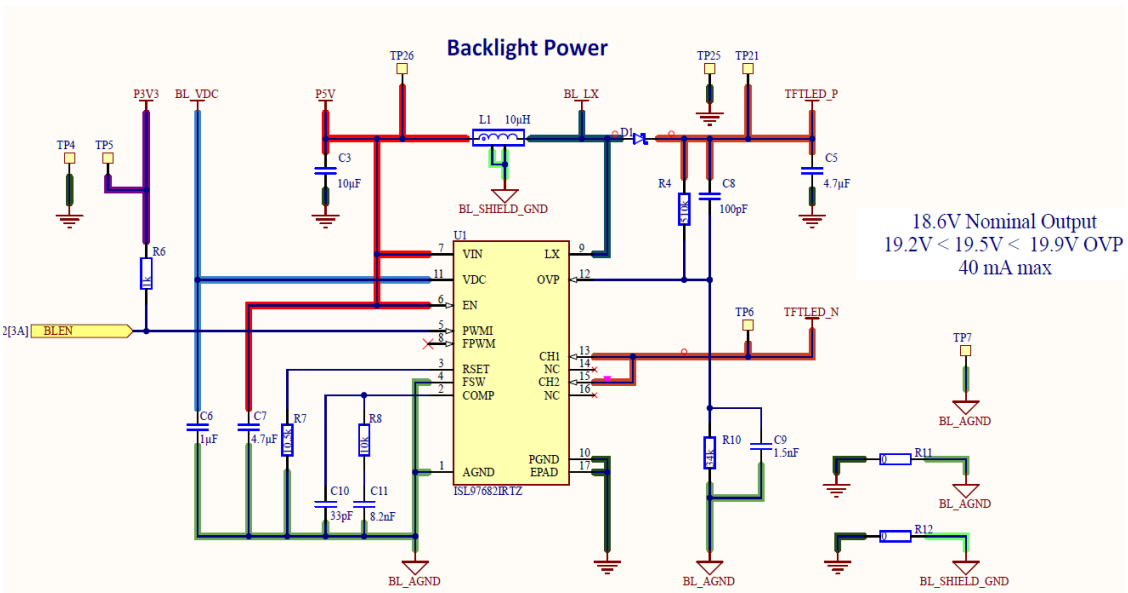

Figure 54. Backlight Control Pin on EK-RA8E2

In **Pin Configuration,** set P404 as the GTIOC3B output of the GPT channel 3. The **Pin Group Selection** is set as mixed, and the **Operation Mode** is GTIOCA or GTIOCB.

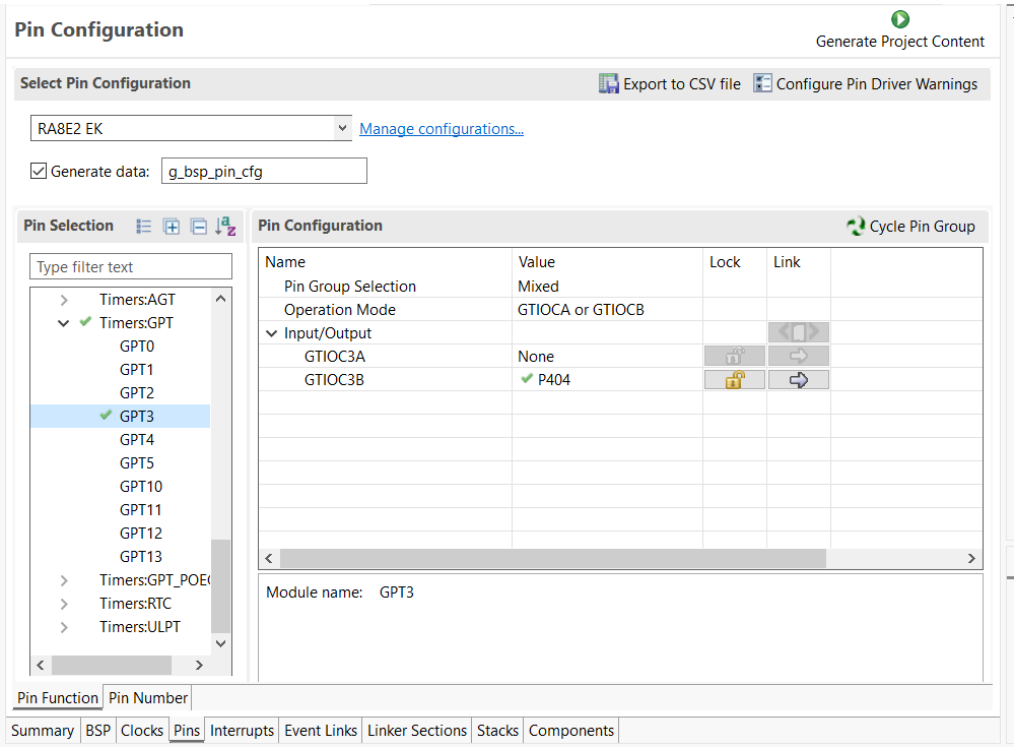

Figure 55. GPT PWM Channel 3 Pin Configuration

The `r_gpt` in the Timer thread is set in PWM mode to modulate LCD backlight intensity. In this graphics application, moving a slider in the **Logging Panel** will generate a duty cycle percentage that will be calculated into the GPT timer period and written to the counter register.

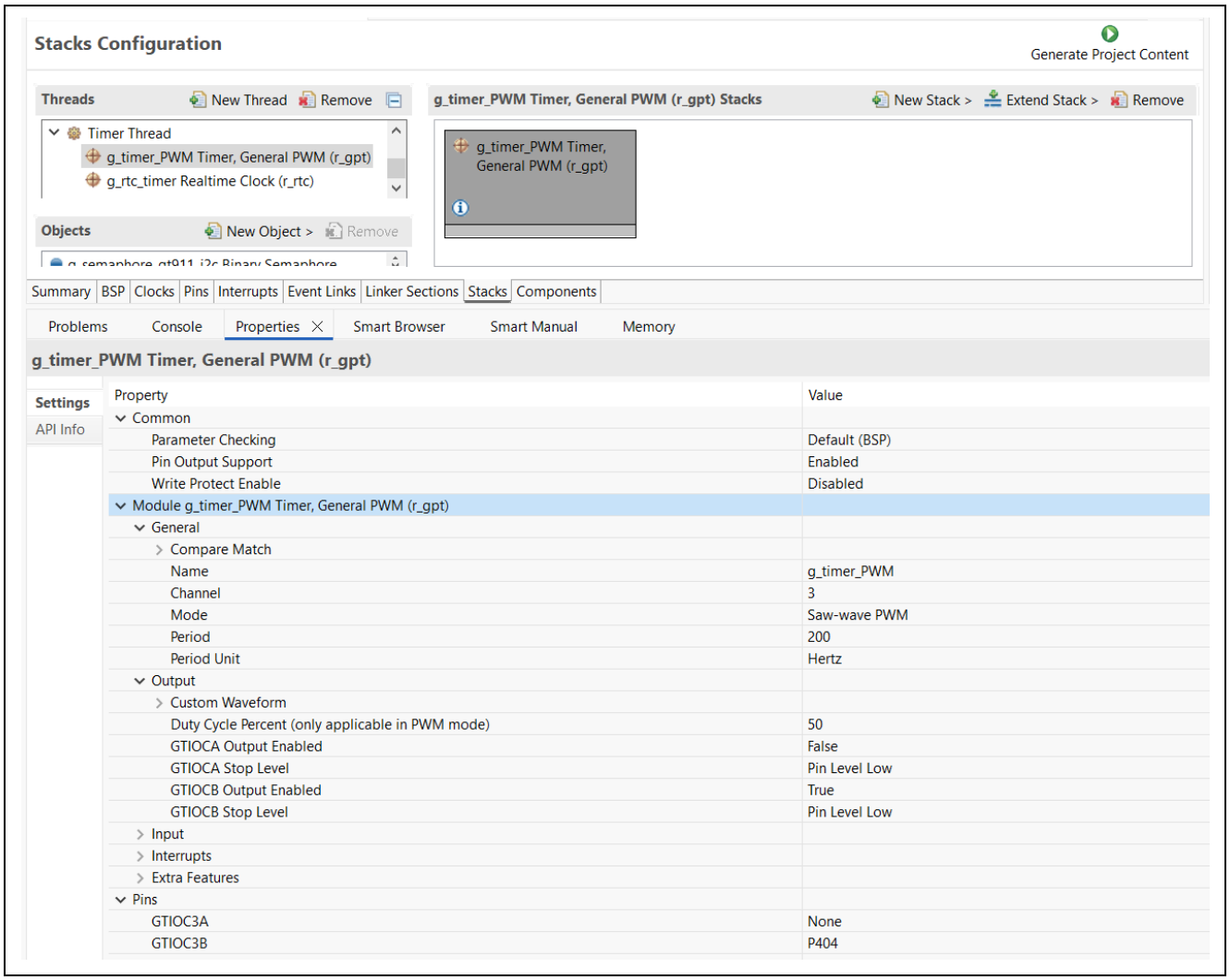

Figure 56. GPT Driver Configuration in PWM Mode

Figure 57 and Figure 58 show the AppWizard configuration for the backlight slider. Its range limits are from 5 to 100. Some interactions and custom code are needed to control the duty cycle of the PWM output as well.

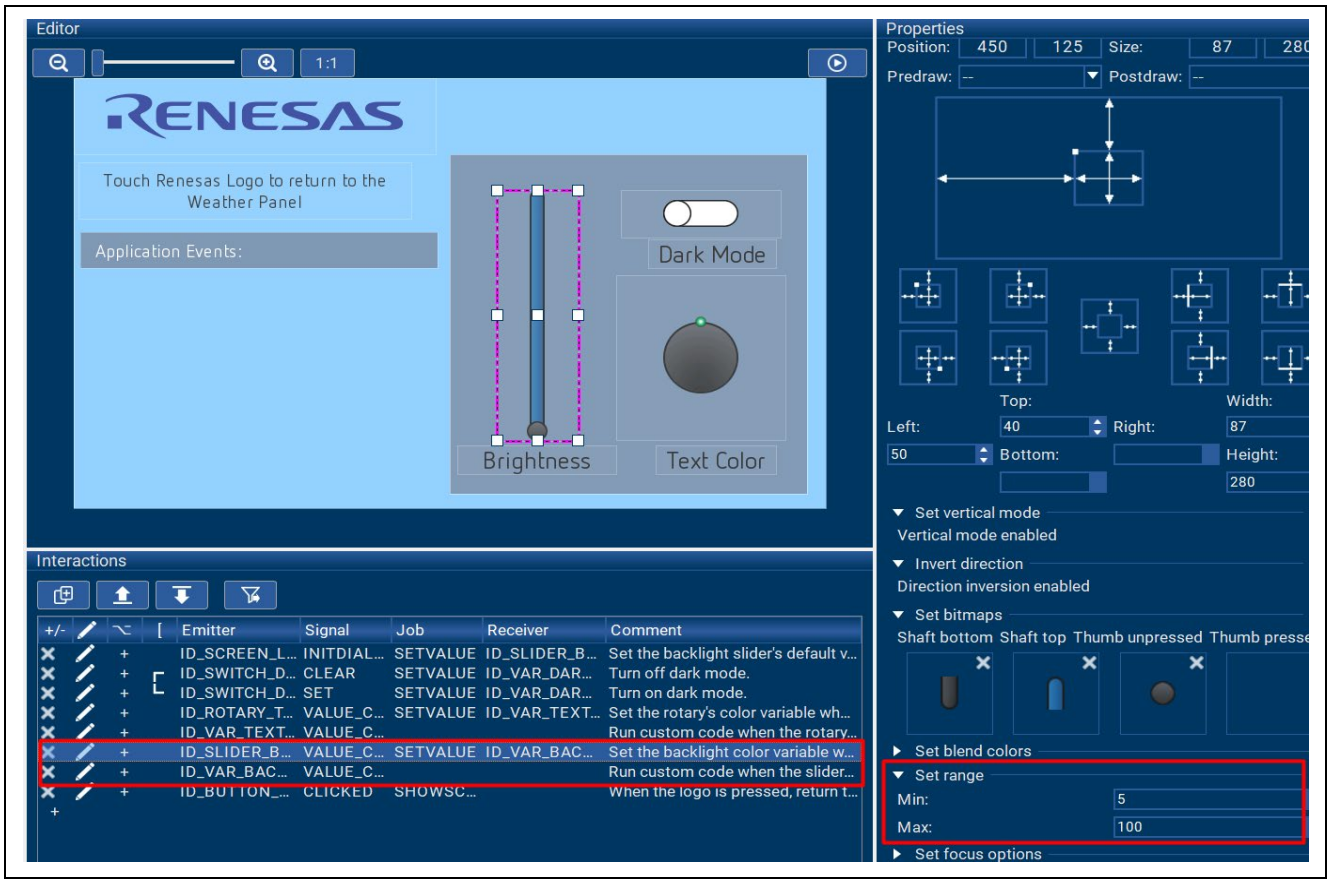

Figure 57. Slider Setup to Control LCD Backlight Intensity

```
        /* Get the Slider/ID_VAR_BACKLIGHT value*/
        gDataApp.pwm_duty_cycle = (uint8_t)APPW_GetVarData(ID_VAR_BACKLIGHT, &gui_err);
        if(gui_err)
        {
            APP_ERR_TRAP(gui_err);
        }

        current_period_count = info.period_counts;
        /* 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. A cast to uint64_t is used to prevent this. The cast is
         * not required for 16-bit timers. */

        duty_cycle_count = (uint32_t) (((uint64_t) current_period_count * gDataApp.pwm_duty_cycle)/GPT_PWM_MAX_PERCENT);
        R_GPT_DutyCycleSet(&g_timer_PWM_ctrl, duty_cycle_count, GPT_IO_PIN_GTIOCB);
    }
}
```

Figure 58. Custom Code Controls PWM Update GPT Timer Duty Cycle

6. Application Code Highlights

This section details the highlights of the graphics application. The goal of the graphics application is to show you how to develop more complex multi-threaded HMI applications using the FSP, AppWizard, and emWin library.

The key goal of the FSP is to abstract much of the complexity of interfacing with various Renesas peripherals and to quickly get you to the point where you can focus on constructing more complex applications as quickly as possible.

6.1 Threads and Main

In the FSP, `main()` is an auto-generated file that looks like the following code. The threads and objects specified during the FSP configuration are initialized in the `main()`.

```
int main(void)
{
    g_fsp_common_thread_count = 0;
    g_fsp_common_initialized = false;

    /* Create semaphore to make sure common init is done before threads start running. */
    g_fsp_common_initialized_semaphore =
#if configSUPPORT_STATIC_ALLOCATION
                      xSemaphoreCreateCountingStatic(
                      #else
            xSemaphoreCreateCounting (
#endif
                                       256,
                                       1
#if configSUPPORT_STATIC_ALLOCATION
                          , &g_fsp_common_initialized_semaphore_memory
                          #endif
                                       );

    if (NULL == g_fsp_common_initialized_semaphore)
    {
        rtos_startup_err_callback (g_fsp_common_initialized_semaphore, 0);
    }

    /* Init RTOS tasks. */
    emWin_thread_create ();
    touch_thread_create ();
    timer_thread_create ();

    /* Start the scheduler. */
    vTaskStartScheduler ();
    return 0;
}
```

Figure 59. The main () function in FSP with FreeRTOS Enabled

When you create a thread using the **New Threads** tab, the FSP creates several files. As an example, when the **emWin Thread** is added, the FSP creates three files for you: `emWin_thread.h`, `emWin_thread.c`, and `emWin_thread_entry.c`, as shown in Figure 60.

The first two files are auto-generated and, therefore, put into the `ra_gen` folder. The `emWin_thread_entry.c` file is the entry point for the **emWin Thread**, and this is where you put your application code. Auto-generated files should not be updated by the user since they will be re-generated every time you build the project or click the **Generate Project Content** button. Auto-generated files always contain some form of **do not edit** message at the top of the file.

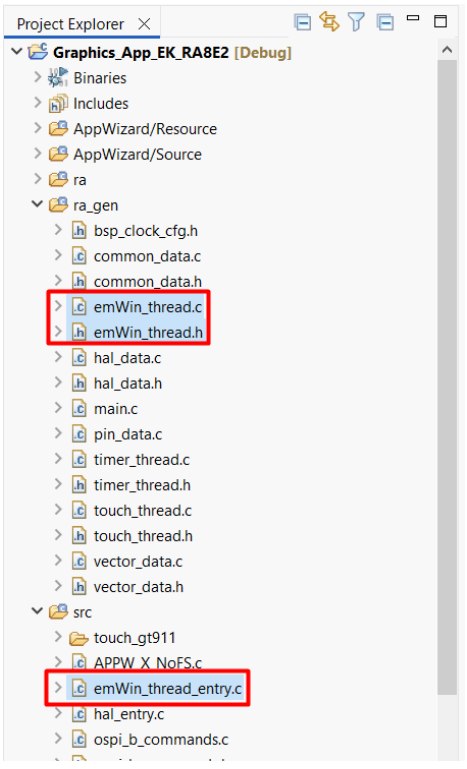

Figure 60. FSP Generated Source File Organization

6.1.1 AppWizard emWin Initialization

First, the external OSPI flash must be initialized early to allow access to image data stored on it. Add the `ospi_b_init()` function call at the beginning of the `emWin_thread_entry()` function, located in the `emWin_thread_entry.c` file, as illustrated in Figure 62. The initialization source code for the OSPI_B is shown in Figure 61.

The FSP does not automatically initialize the AppWizard system. To initialize it, simply include `GUI.h` and add the `MainTask()` API call to `emWin_thread_entry()` located in the `emWin_thread_entry.c` file as Figure 62.

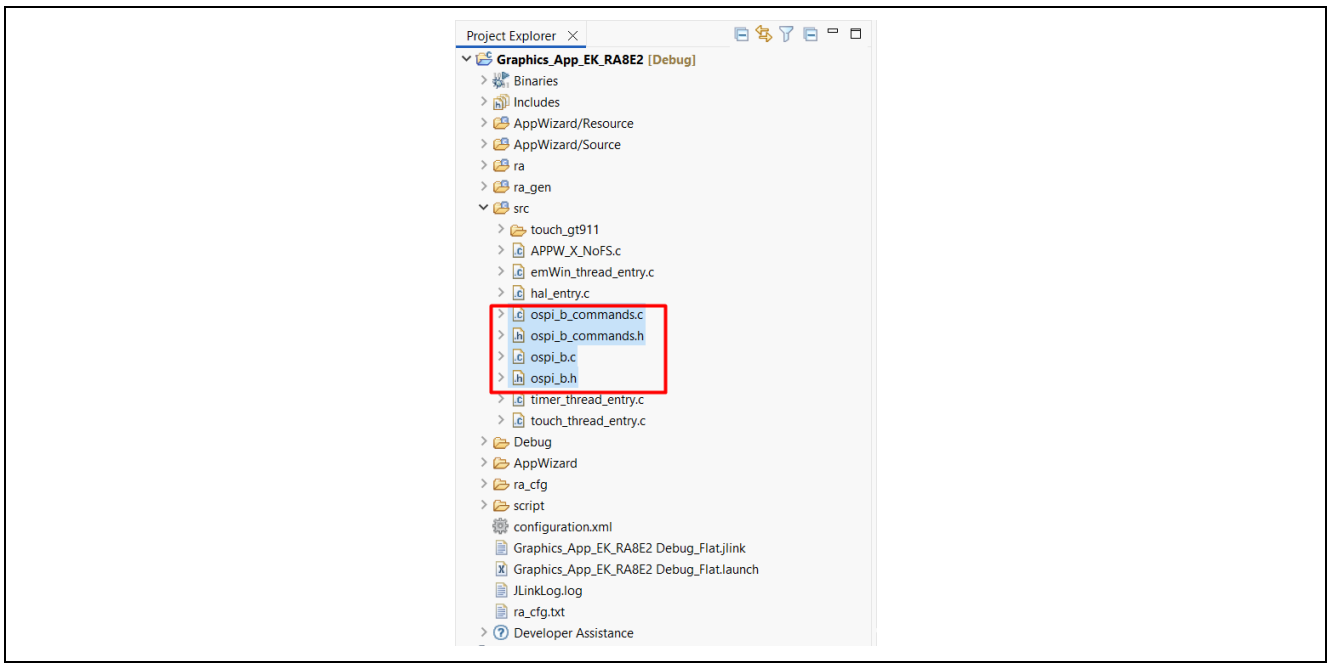

Figure 61. OSPI_B configuration sources.

```
#include "emWin_thread.h"
#include "GUI.h"
#include "ospi_b.h"

/* emWin thread entry function */
void emWin_thread_entry(void *pvParameters)
{
    FSP_PARAMETER_NOT_USED (pvParameters);

    fsp_err_t err = FSP_SUCCESS;

    /* Initialize ospi_b */
    err = ospi_b_init();
    if (FSP_SUCCESS != err)
    {
        APP_ERR_TRAP(err);
    }
    BaseType_t xResult = xSemaphoreTake(g_touch_reset_semaphore, portMAX_DELAY);

    if(xResult != pdTRUE)
    {
        APP_ERR_TRAP(xResult);
    }

    MainTask();

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

Figure 62. Initialize the External Flash and AppWizard System

6.1.2 emWin Events and Messages

Touching the screen in the graphics application causes emWin to invoke the specific callback function generated for that screen in the AppWizard. AppWizard provides the callback function with specific information about the window that caused the event and the actual event that occurred. These events are defined in `WM.h`.

You can add your code to slot routines in the file `<ScreenID > _Slots.c` located in the `\AppWizard\Source\CustomCode` folder to handle window events. The slot routines are actual callback routines generated by AppWizard. The `<ScreenID > _Slots.c` is updated whenever you add and generate new widgets or AppWizard interactions using AppWizard. However, custom code will be retained. It is a good practice to create your custom code in a separate file and call it in the appropriate slot routine.

```
* @brief       Custom code for cbID_SCREEN_MAIN in ID_SCREEN_MAIN_Slots.c
void cuscbID_SCREEN_MAIN(WM_MESSAGE * pMsg) {
  int wId                    = 0;
  int wMsg                   = 0;

  switch(pMsg->MsgId) {
      case WM_INIT_DIALOG:
          /* Get and store Images's handles */
          if(ImageHandleGet(pMsg))
          {
              APP_ERR_TRAP(FSP_ERR_INVALID_POINTER);
          }
          /* Set default weather forecast */
          if(WeatherForecastInit(pMsg))
           {
                APP_ERR_TRAP(FSP_ERR_INVALID_POINTER);
           }
          /* Set Thermostat target temperature */
          APPW_SetVarData(ID_VAR_TARGET_TEMP, gDataApp.thermo_target_temp);
          /* Save logging */
          LogDataAppend(gDataLog, sizeof(gDataLog),"\n%s", "Init Dialog");
          /* Create timer to control effects/animation*/
          ghTimer = WM_CreateTimer(pMsg->hWin, 0, ANIM_TIMER_PERIOD, 0);
          break;
      case WM_TIMER:
          /* Rainy effect*/
          if(SYS_WEATHER_RAINY == gDataApp.sys_weather_type)
          {
              if(0 == AnimRainyState)
              {
                  /* Hide 1st animation image, show 2nd animation image */
                  WM_HideWindow(hImageAnimBGD[gDataApp.sys_weather_type]);
```

Figure 63. Custom Code for The Slot Routine cb_ID_SCREEN_MAIN

6.1.3 AppWizard Variables

Variables in the AppWizard can be used to store a value. They can be accessed and changed by the GUI or from outside of the GUI. The GUI can react to a change in a variable using interactions. One of the typical uses is to update the variables in a non-GUI thread to trigger data exchange between the emWin and non-GUI threads.

```
/* Timer Thread entry function */
/* pvParameters contains TaskHandle_t */
void timer_thread_entry(void *pvParameters)
{
    FSP_PARAMETER_NOT_USED (pvParameters);

    /* Set up GPT/PWM timer using for LCD back light control */
    if(gpt_timer_PWM_setup())
    {
        __asm("BKPT #0\n");
    }

    /* Set up RTC timer */
    if(rtc_timer_setup())
    {
        __asm("BKPT #0\n");
    }

    while (1)
    {
        /* Wait for interrupt from RTC timer */
        xSemaphoreTake(g_timer_semaphore, portMAX_DELAY);

        /* Get date, time */
        R_RTC_CalendarTimeGet(&g_rtc_timer_ctrl, &RtcTimeCurrent);

        /* Trigger GUI update*/
        APPW_SetVarData(ID_VAR_TIME_UPDATE, 1);

        vTaskDelay (1);
    }
}
```

Figure 64. AppWizard Variable Update in Timer Thread

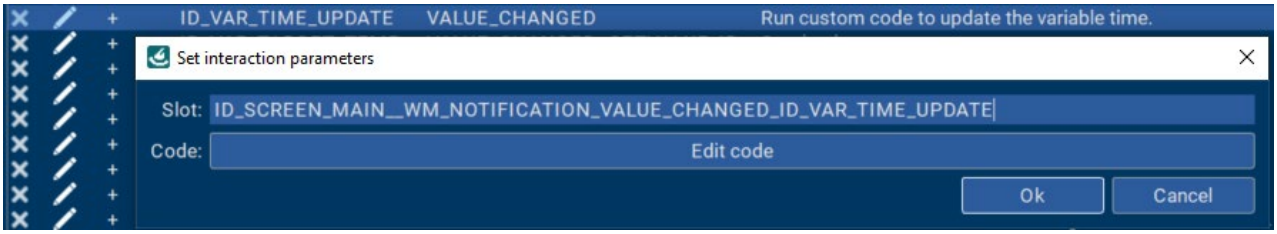

Figure 65. Set up the Interaction to perform a Date, Time Update when ID_VAR_TIME_UPDATE is changed in AppWizard

7. Importing and Building the Project

To bring the graphics application into the e² studio, follow these steps:

1. Launch e² studio.
2. In the workspace launcher, browse to the workspace location of your choice.
3. Close the **Welcome** window.
4. In e² studio, go to **File** > **Import.**
5. In the **Import** dialog box, pick Existing Projects in **Workspace.**
6. Select the archive file.
7. Select the Graphics_App_EK_RA8E2 project and click **Finish**.
8. Open **configuration.xml**.
9. Click on **Generate Project Content** on the FSP configurator window.
10. Now build the project.

8. Downloading the Executable to the EK-RA8E2 Kit

To connect and run the code, follow these steps:

1. Connect your PC to the USB port DEBUG using a USB cable.
2. Go to **Run > Debug Configurations.**
3. Click **Debug**. The program will break at the reset handler.
4. Click **Switch** to the **Debug perspective** when prompted by the e² studio.
5. Click **Resume** > **Resume**.
6. The **Weather Panel** will show as in Figure 66. You can select the forecast day or adjust the thermostat temperature. Touch the top right corner to move to the **Logging Panel**.

Figure 66. The Weather Panel

The **Logging Panel** allows you to adjust the LCD backlight using the slider or change the **Logging Dialog** text color and background color using the rotary and the switch, respectively. The logging buffer resets when it reaches the limit of 256 bytes. Touch the Renesas logo to go back to the **Weather Panel**.

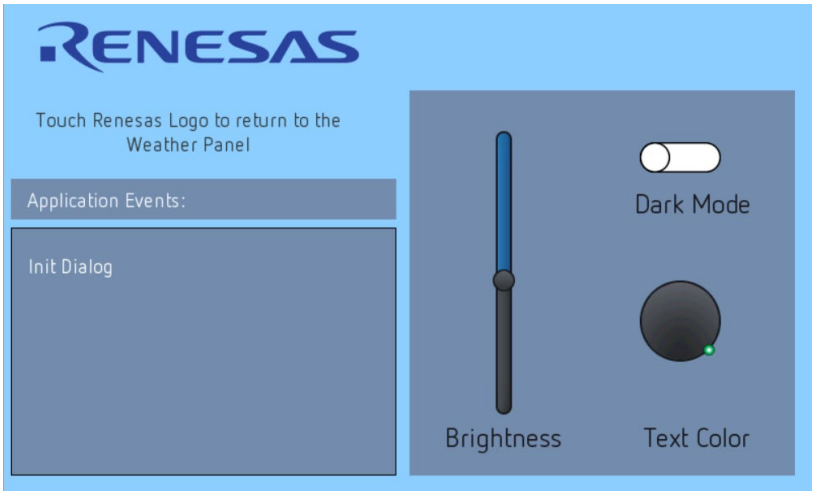

Figure 67. The Logging Panel

9. e² studio Tricks

The e² studio IDE has a handy feature that you can use to ensure that the images you are seeing on your LCD screen are coming from your framebuffer. To use this feature, make sure to connect the e² studio to your board and run the program under the debugger. Ensure that your **Memory** tab is open in the **Console** window, normally located at the bottom of the screen in **Debug** view. Click the small green plus (**+**) sign in the Monitors Pane to add a memory monitor. You should see a **Monitor Memory** dialog, as shown in Figure 68. Enter the Framebuffer **&fb_background[0]** or **&fb_background[1]** and click the **OK** key.

A new tab should now appear under the **Memory** tab that displays the contents of the memory area you specified for the memory monitor.

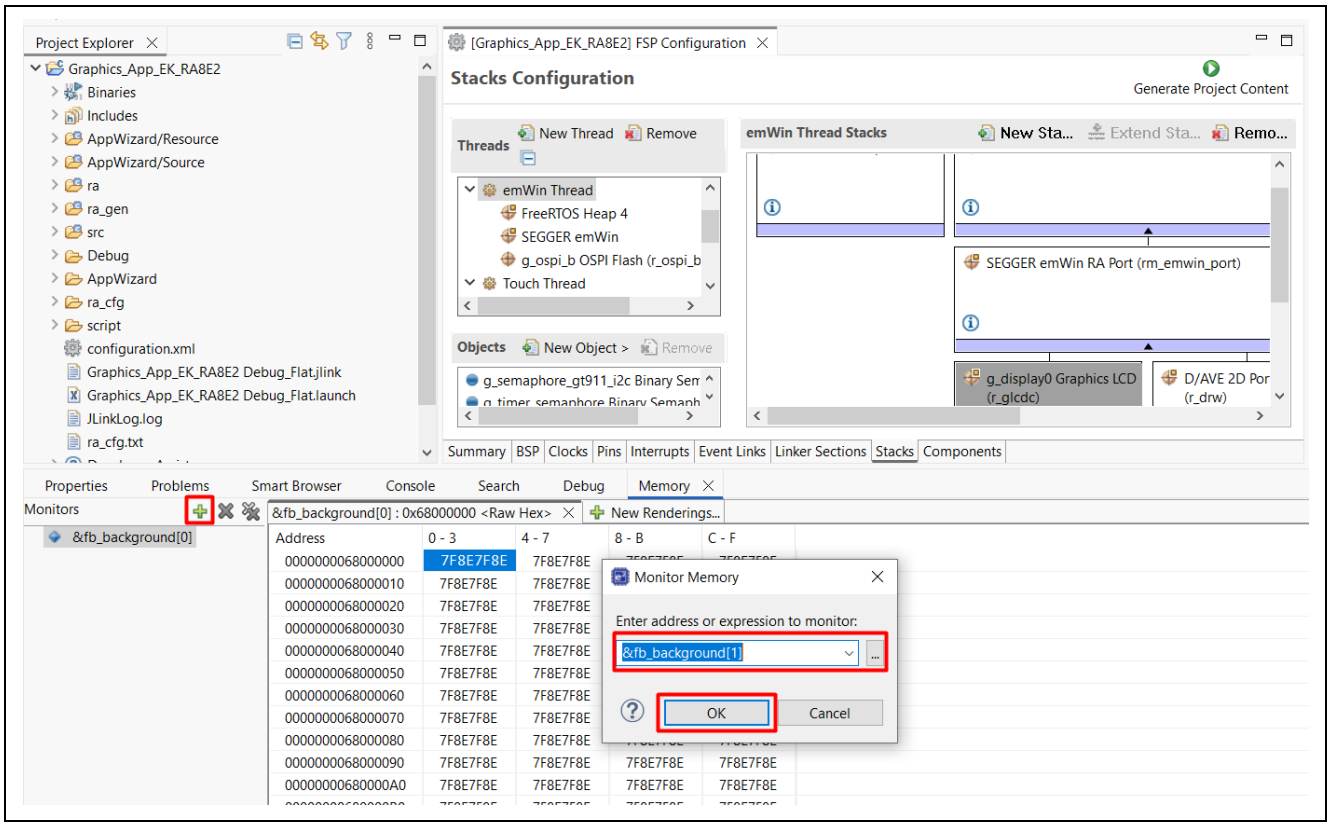

Figure 68. Using the Memory Monitor to Display the Framebuffer Contents

You should now see the contents of the selected framebuffer memory area displayed in the memory monitor you just created. If you know what the hex value of every pixel should be on your display, you would be able to use this memory monitor to definitively say that your image is being stored in the framebuffer. However, as most of us do not know the hex values associated with our pixels, we will let the memory monitor do the work for us.

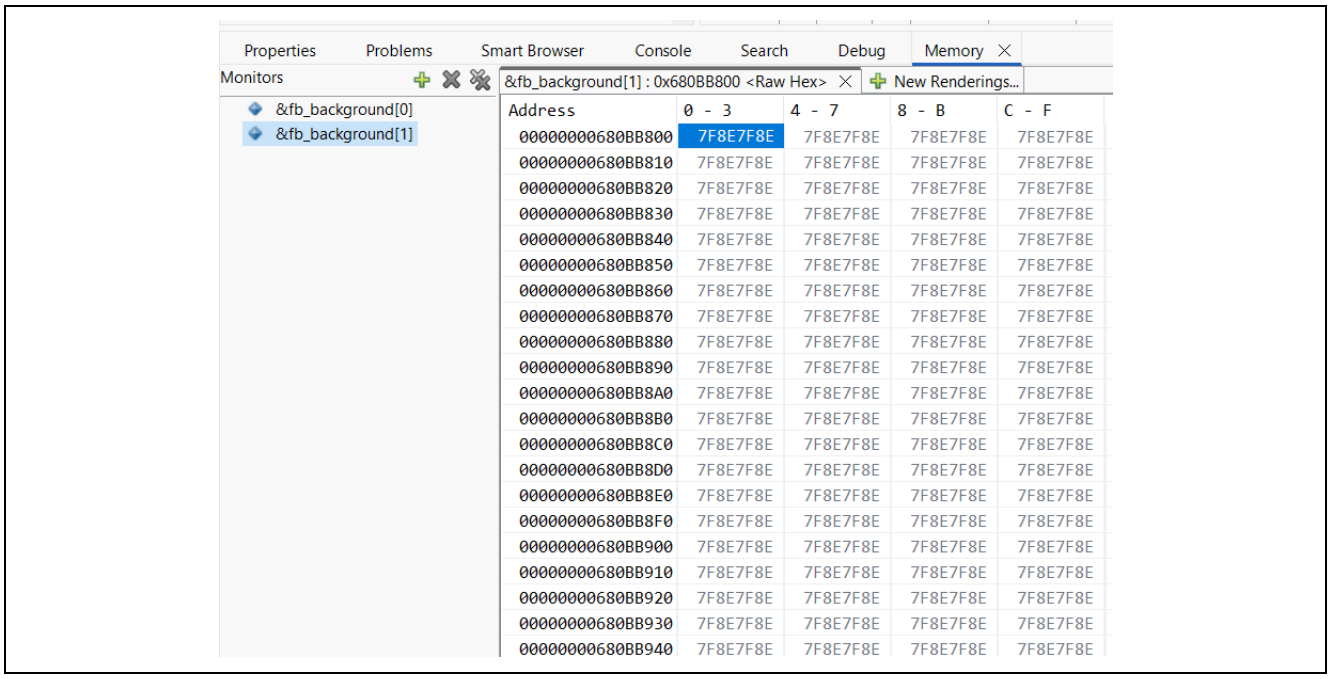

Figure 69. Framebuffer 1 Contents

Select the **New Renderings** tab next to the memory monitor you just created, select **Raw Image** type from the list of options, and press the **Add Rendering(s)** button on the right side of the screen.

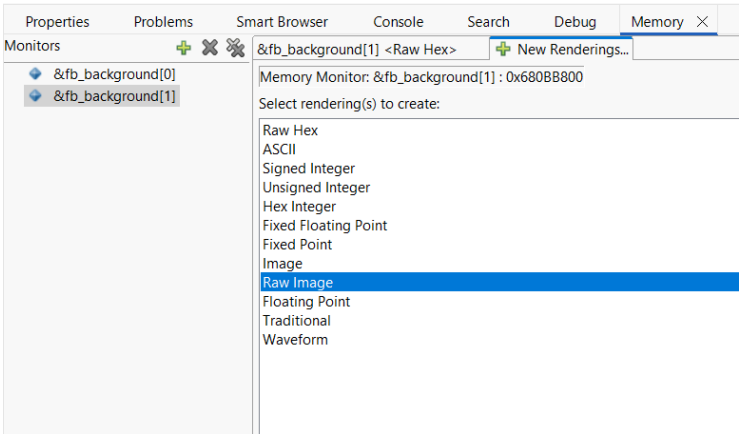

Figure 70. Rendering Format Selection

Right-click on the **Raw Image**, select **Format**, and then the **Raw Image Format** dialog box will appear, allowing you to enter the width and height of the screen resolution, along with the encoding, which is 32bpp (8:8:8:8) in this case.

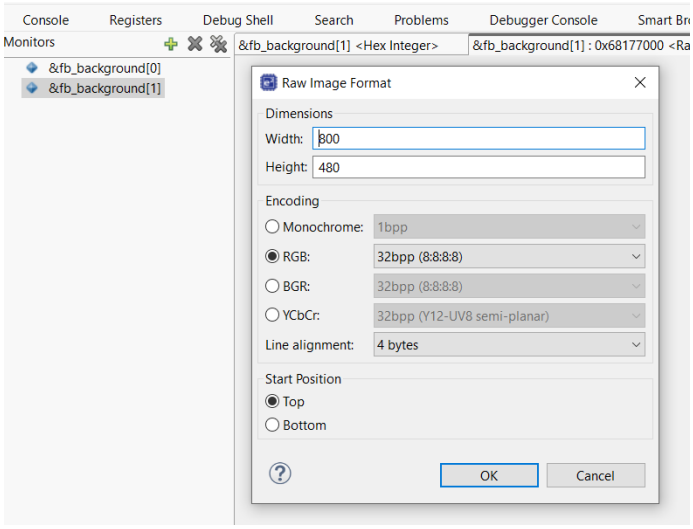

Figure 71. Raw Image Format for Graphics Application on EK-RA8E2

Once you click **OK**, the memory monitor will present you with the image that will be displayed at that memory address based on the parameters you entered.

Figure 72. Image Rendering Using e² studio Memory Monitor

10. 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	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	Aug.05.25	-	Initial version

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/.