

## Renesas Synergy™ Platform

# Getting Started with the Weather Panel Application

### Introduction

This application note describes a simulated Weather Panel application. The Weather Panel application is geared towards providing you with a quick out-of-box experience that demonstrates how complex multi-threaded applications with a touch screen graphical Human Machine Interface (HMI) may be developed using the Renesas Synergy™ Software Package (SSP).

The Weather Panel application runs on several different development boards including the DK-S7G2, SK-S7G2, PK-S5D9, and PE-HMI1, as shown in the following figure. These boards have different screen resolutions and a slightly different feature set. For example, more full-featured boards like the DK-S7G2 and PE-HMI provide backlight control of the LCD, while the lower cost SK-S7G2 and PK-S5D9 MCU boards do not include this ability. This application note focuses on the PE-HMI1, while highlighting the key changes required to make the application run on each of the different boards.



**Figure 1. Weather Panel Application on Several Development Boards**

This application was developed using the Synergy Software Package (SSP). The SSP is a unified, robust framework that includes driver-level support for the peripherals in Synergy ARM Cortex M4/M0+ MCUs along with ThreadX®, Express Logic's Real Time Operating system (RTOS). In addition to ThreadX, full stack support is available through Express Logic's X-Ware suite of stacks (NetX™, USBX™, GUIX™, and FileX®). 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 an RTOS such as ThreadX. While specific knowledge of ThreadX makes understanding the code easier, you should be able to easily understand the information provided in this application note if you have any previous experience with RTOS principles such as threads, message queues, semaphores,

and mutexes. For more detailed information on ThreadX features, refer to the Synergy *X-ware (ThreadX) User Manual*.

The Weather Panel application was developed using the Renesas Synergy e<sup>2</sup> studio Integrated Solution Development Environment (ISDE). This Eclipse based IDE is a free application that you can download from Renesas. While e<sup>2</sup> studio ISDE supports using multiple tools chains, this application note was built using the GCC compile tools that also come free with the e<sup>2</sup> studio ISDE environment.

While building applications under the Renesas Synergy™ 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. This application note walks you through all the steps necessary, including the following:

- Board setup
- Application overview
- Detailed explanation of the graphical screens uses
- GUIX Studio project integration
- Synergy framework configuration
- Application design highlights
- Inter-thread communication using the Synergy messaging framework
- Using the General Purpose Timer to drive a PWM backlight control signal
- Loading and running the project.

### Required Resources

- e<sup>2</sup> studio ISDE 7.3.0 or later
- Synergy Software Package (SSP) 1.6.0 or later
- IAR Embedded Workbench® for Renesas Synergy™ 8.23.3 and SSC v7.3.0
- GUIX Studio v5.4.0.0 or later

### Target Devices

- Synergy PE-HMI 1 v2.0 or later board (S7G2 Synergy MCU Group)
- Synergy DK-S7G2 v3.1 or later development board (S7G2 Synergy MCU Group)
- Synergy PK-S5D9 v1.0 or later development board (S5D9 Synergy MCU Group)
- Synergy SK-S7G2 v3.1 or later development board (S7G2 Synergy MCU Group)

## Contents

1. Board Setup.....	4
2. Application Overview.....	5
2.1 Synergy S7G2 and S5D9 MCU Peripherals used by the Weather Panel Application.....	5
2.2 Human-Machine Interface (HMI) .....	7
2.3 Weather Panel Screens .....	7
2.3.1 Large Screen Design.....	8
2.3.2 Small Screen Design.....	9
3. GUIX Studio Overview .....	9
4. Analyzing the Application .....	14
4.1 Source Code Layout.....	14
4.2 Thread Overview .....	15
4.2.1 HMI Thread.....	16
4.2.2 Thread Layout and the SSP.....	17
5. Framework Configuration .....	19
5.1 Components Tab .....	20
5.2 Threads Tab .....	22
5.3 Thread Objects.....	23
5.4 Module Configuration .....	24
5.4.1 GLCD Configuration.....	24
5.4.2 TCON Configuration.....	25
5.4.3 Using External Memory for Frame Buffer .....	27
5.4.4 e <sup>2</sup> studio Tricks.....	29
6. Application Code Highlights.....	31
6.1 Threads and Main.....	31
6.1.1 GUIX Initialization .....	32
6.1.2 Events and GUIX Message .....	33
6.2 LCD control.....	35
7. Importing and Building the Project .....	36
8. Downloading the Executable to the Target Board.....	37
9. Known Issues .....	37
10. References .....	37
Revision History.....	39

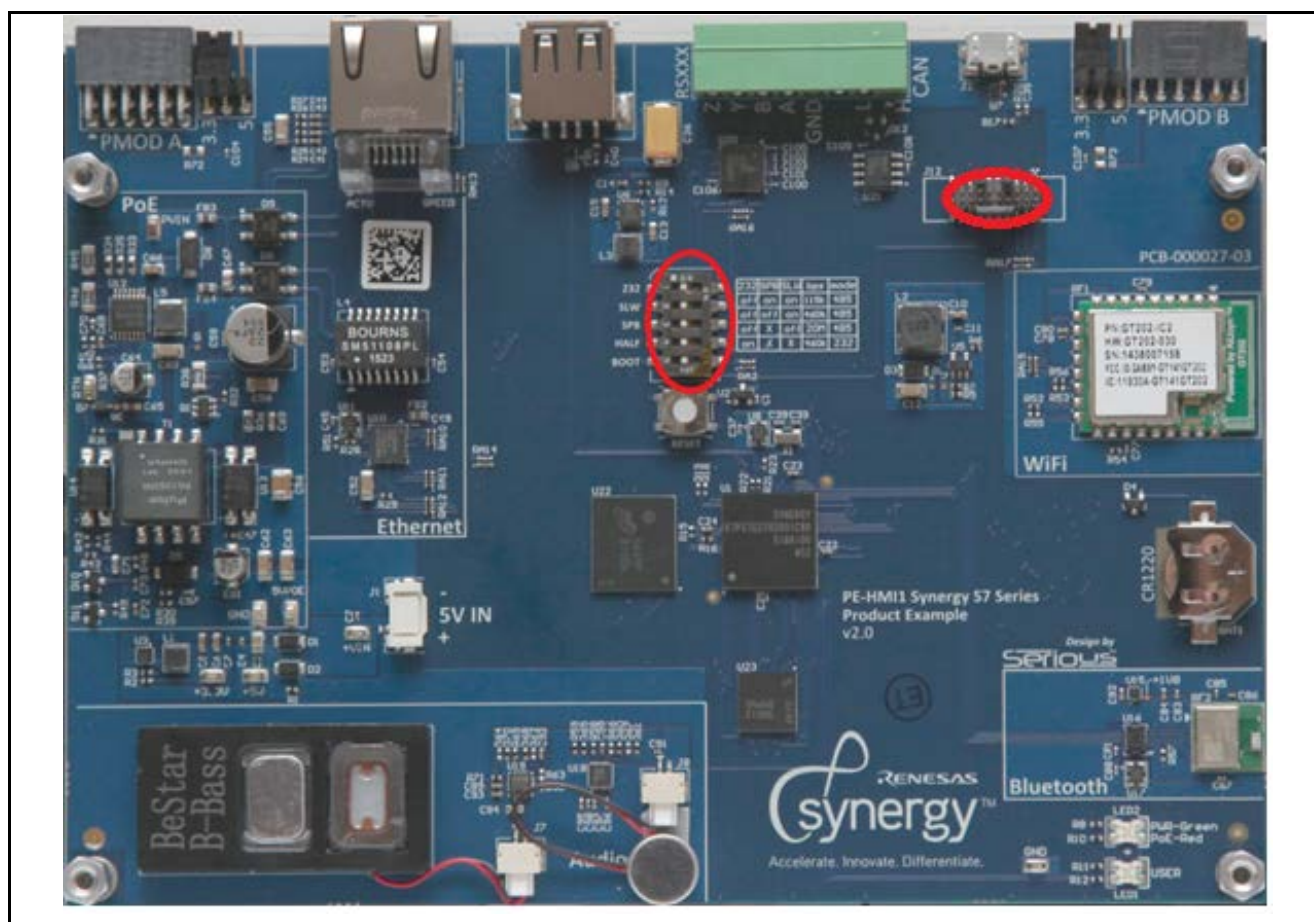
## 1. Board Setup

The PE-HMI1 board contains a few switch settings which must be configured prior to running the firmware associated with this application note. In addition to these switch settings, the boards also contain a connector to access the J-Link® programming interface.

Connect the supplied J-Link® LITE between J12 of the PE-HMI1 and the PC where you have loaded the Synergy e² studio ISDE software. J12 is marked as shown in the top left portion of Figure 2. To ensure the proper operation of the application, make sure that the DIP switch, shown marked in the middle of Figure 2, is configured as shown in Table 1.

**Table 1. Switch settings for PE-HMI1**

Switch	Setting
232	OFF
SLW	OFF
SPB	OFF
HALF	OFF
BOOT	OFF



**Figure 2. PE-HMI1 v2.0 Hardware Details for the Weather Panel Application**

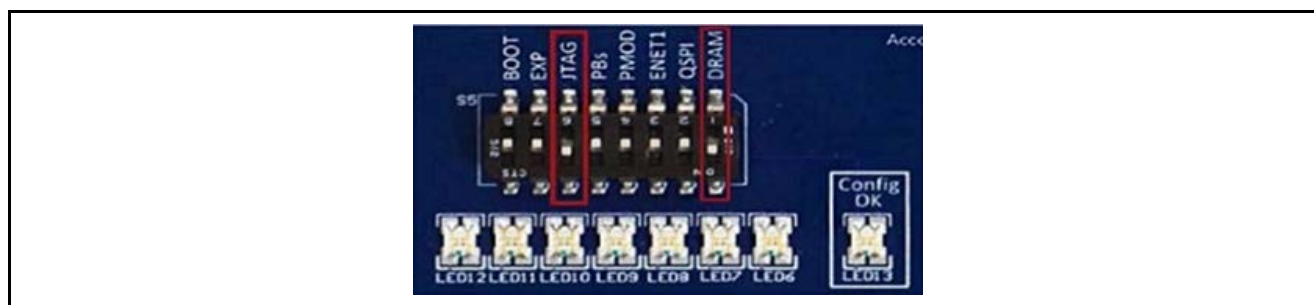
Follow the switch settings shown in Table 2 and Figure 2 for the S5 DIP switch on the DK-S7G2 V3.1 board.

Note: For the DK-S7G2 V4.1 board, the switch settings are spread across DIP switches S6, S7, S8, and S9. For DK-S7G2 V4.1, JTAG, LCD, and SDRAM need to be ON, and rest are OFF.



**Table 2. Switch Settings for DK-S7G2 V3.1 Board**

Switch	Setting
BOOT	OFF
EXP	OFF
JTAG	ON
PBs	OFF
PMOD	OFF
ENET1	OFF
QSPI	OFF
DRAM	ON


**Figure 3. DK-S7G2 V3.1 Synergy MCU Board Switch Setting**

## 2. Application Overview

One of the key goals of the Weather Panel application is to demonstrate how to build applications employing complex HMI screens using GUIX™ Studio. The following list highlights all the key features of the Weather Panel application:

- Complex HMI design using GUIX Studio
- Multi-threaded applications using the ThreadX® RTOS
  - Queue and Mutex Thread objects used
- Extensive use of Synergy Messaging framework for inter-thread communication
- GLCD configuration for various screen types/sizes
  - Frame Buffer run from internal/external memory
  - External Memory interface used
  - SPI initialization of ILI9341 Graphics Controller (SK-S7G2 and PK-S5D9 boards).
- Touch Panel, I²C touch controller driver ft5x06
  - External IRQ mapping required

In any software design, there are many ways to solve the same problem. The solution given in this application note is one approach.

### 2.1 Synergy S7G2 and S5D9 MCU Peripherals used by the Weather Panel Application

The Weather Panel application uses the Synergy S7G2 MCU or S5D9 MCU depending on the board being used. This MCU is built around an ARM Cortex-M4 device. Developing complex embedded 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 are necessary to fulfill those including the target MCU that will be 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 a considerable amount of time understanding the register map of the on-board peripherals, and writing lower level driver code necessary to expose the peripheral to the upper level application code. As we will see, most of this work has already been done in the Synergy Framework, considerably streamlining application development.

3. In addition to the on-board peripherals of the target MCU, the design often encompasses external hardware and how it is controlled. As an example, the DK-S7G2 and PE-HMI boards have LCD screens, that may be controlled directly by the on-board Graphics LCD Controller (GLCD) of the S7G2 MCU. The SK-S7G2 and PK-S5D9 Synergy MCU boards use a smaller, lower cost LCD, that requires some initialization, over a serial interface, before it can be controlled by the GLCD of the S7G2 MCU and S5D9 MCU respectively.
4. The last step usually details how an application will be structured on top of the selected hardware to accomplish the initial requirements.

The Weather Panel application requirements were first mapped to the on-board peripherals of the S7G2 or S5D9 Synergy MCU boards. Figure 4 and Figure 5 show all the internal hardware peripherals used by the Weather Panel application. This application note describes how each of these peripherals is configured using the Synergy Framework, and the considerations that were used for each peripheral as the application was being developed.



**Figure 4. S7G2 Synergy MCU Peripherals used in the Weather Panel Application**

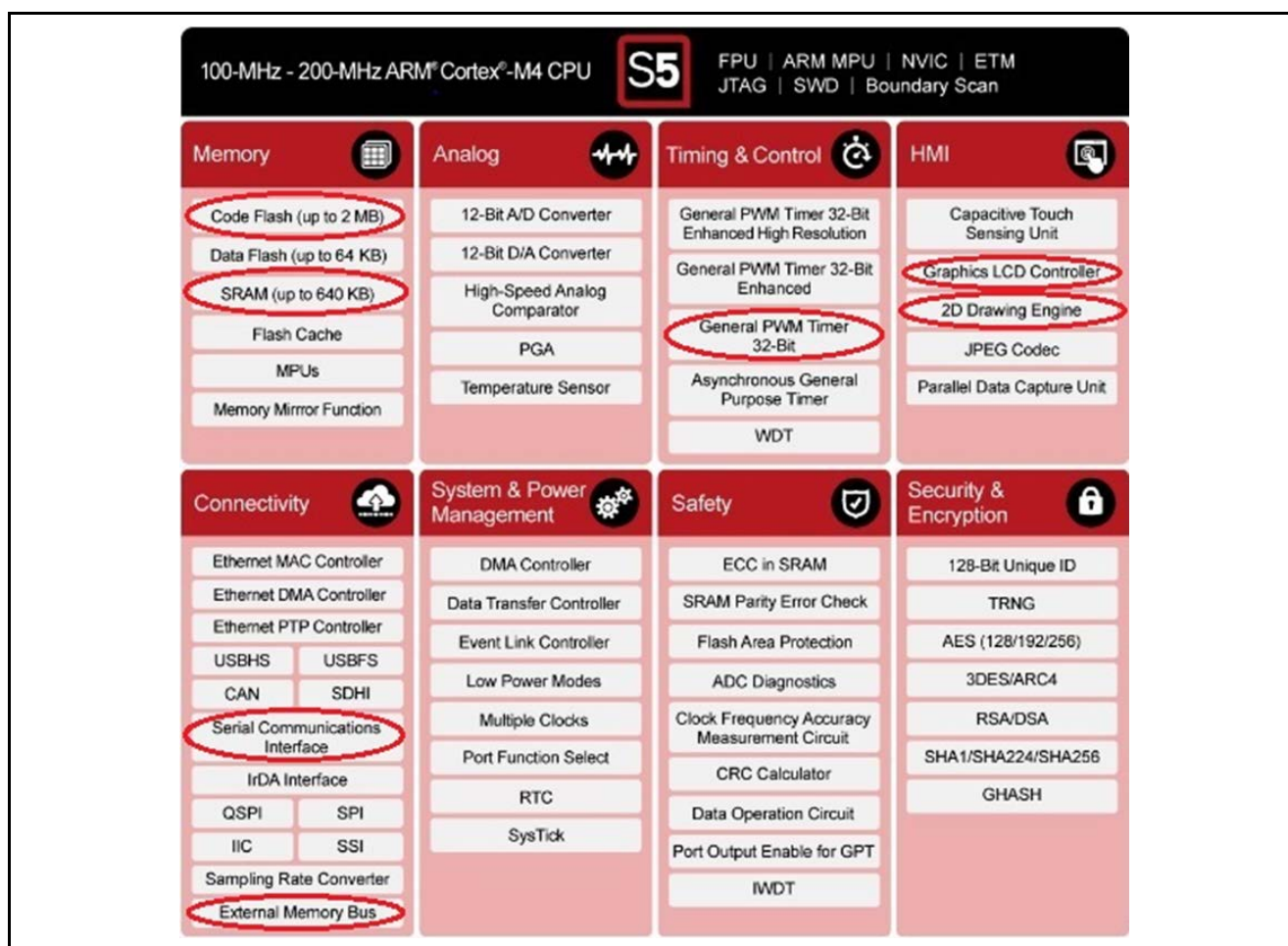


Figure 5. S5D9 Synergy MCU Peripherals Used in the Weather Panel Application

## 2.2 Human-Machine Interface (HMI)

In many 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 presentation. This means that the GUI generally does not make decisions about what to display. It is 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 GUIX Studio comes into play.

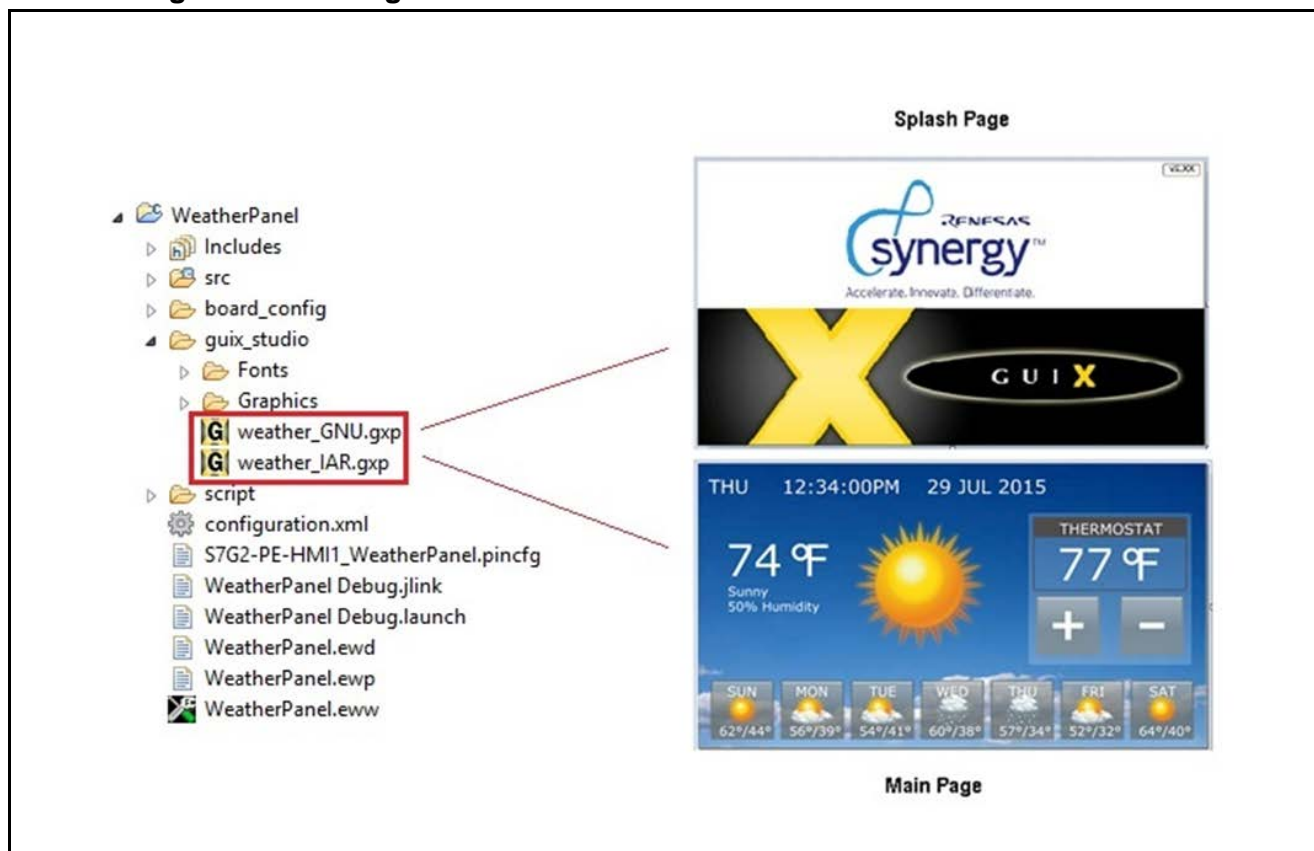
The SSP natively supports the use of GUIX from Express Logic. You may choose to use GUIX primitive calls directly in your application or choose to use the GUIX Studio to design your screens. GUIX Studio is a stand-alone tool that provides a point and click environment for generating all the screens necessary for your embedded application. Once designed, the studio outputs .c and .h files, which you then integrate into your application. All the application screens in the Weather Panel application were built using the GUIX Studio.

## 2.3 Weather Panel Screens

Screen designs are normally tailored to the size of the screen that they will be displayed on. This often necessitates multiple graphical designs when porting an application to different boards with different sized LCD screens. There are two ways to approach this problem. The first approach involves building separate static display designs for each screen resolution. GUIX Studio allows you to do this quickly. This is the approach used in this application note. The second approach involves building the screens dynamically, and sizing the windows/widgets at run time, depending on the screen resolution available. GUIX has a rich API that allows this type of dynamic screen generation, building screens dynamically.

The Weather Panel application has two different designs, one for large screens like the 4.3-inch screen found on the DK-S7G2 MCU board or the 7-inch screen found on the PE-HMI MCU board, and one for smaller screens like the one on the SK-S7G2 and PK-S5D9 Synergy MCU boards.

### 2.3.1 Large Screen Design



**Figure 6. Screen Snapshots of the Weather Panel Application**

For larger screens such as the DK-S7G2 or the PE-HMI Synergy MCU development boards, the Weather Panel application contains two main screens as shown in Figure 6, named as follows:

- Splash Page This is the first screen that appears on the HMI on boot up.
- Main Page Adjust Weather Panel settings either by selecting day, or Increase/Decrease Temperature.

While the same two screens are used for both the DK-S7G2 and PE-HMI boards, two separate graphical designs exist since it was necessary to scale down the design for the smaller screen of the SK-S7G2 and PK-S5D9 boards. A GUIX Studio project is made of various resource files (such as the fonts, images, and so forth) but, as is the case of many IDEs, the project definition itself is maintained in a single xml file with a .gxp extension. A separate .gxp file exists for all three board designs.

weather\_GNU.gxp is the GUIX Studio project to be used for e<sup>2</sup> studio.

weather\_IAR.gxp is the GUIX Studio project to be used for IAR EW for Synergy.



### 2.3.2 Small Screen Design

Figure 7 shows the small screen design used for the SK-S7G2 Synergy MCU board. The key change in this design is the elimination of weekday settings and limiting to only three days.

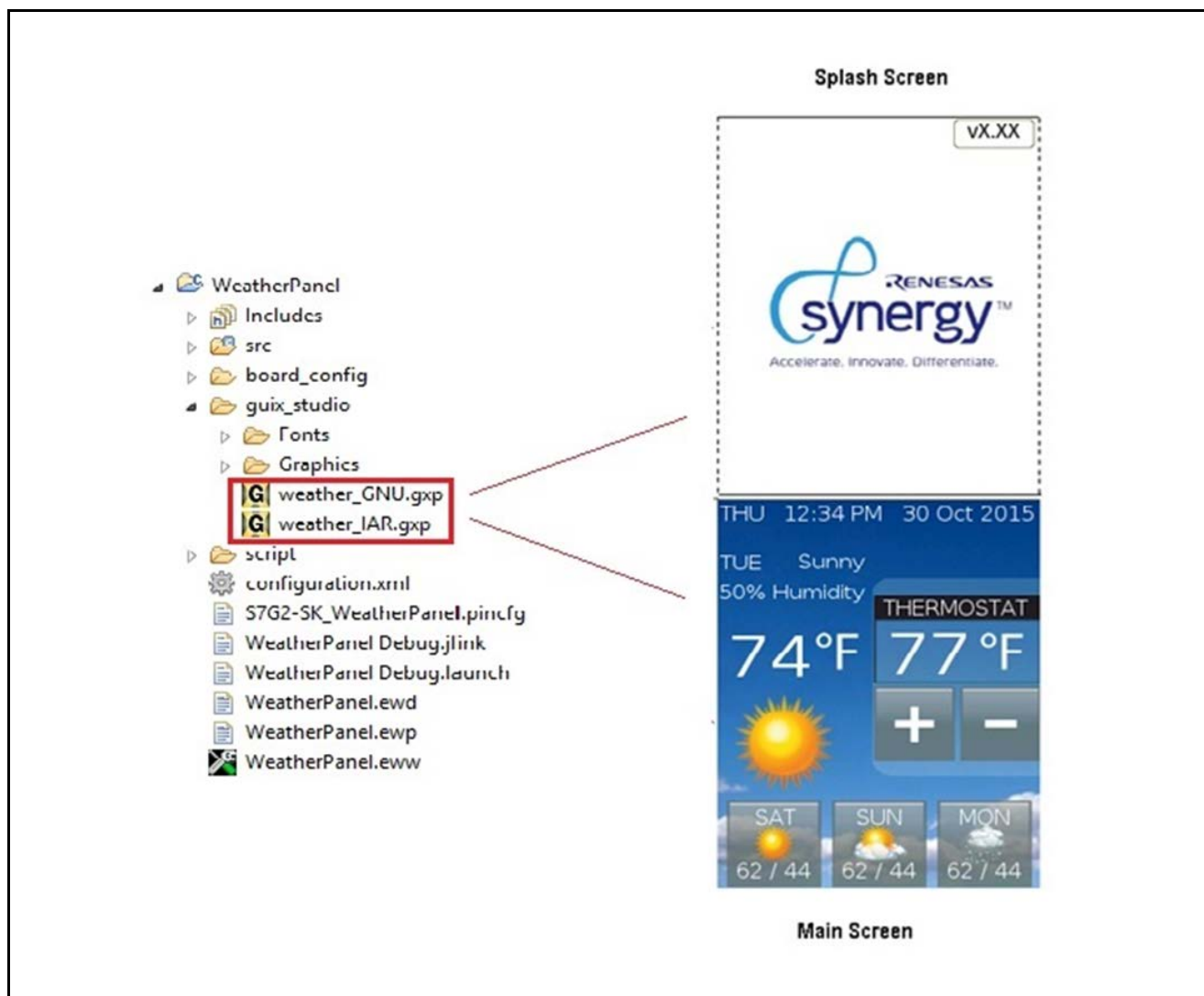


Figure 7. Screen Snapshots for the SK-S7G2 and PK-S5D9 Synergy MCU Boards

## 3. GUIX Studio Overview

This section provides an overview of how GUIX screens are designed and integrated into an SSP application using GUIX Studio. It is not meant as a replacement for the GUIX or GUIX Studio documentation. When designing graphical interfaces for the Renesas Synergy™ platform, you are encouraged to read the full documentation for GUIX and GUIX Studio while studying the associated screen handling code in the Weather Panel application.

GUIX Studio 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 such that what you see in GUIX Studio that is running on your desktop PC is what you will get on your embedded screens.

GUIX comes as a standard with a few fonts and basic graphics for things like button controls. During your screen creation phase, you may provide the GUIX with your own external images and font files to make your displays as fancy as needed. GUIX Studio also provides for the use of multi-language displays using string tables.

Note: The steps provided here are for `weather_GNU.gxp` for e<sup>2</sup> studio. The same steps need to be followed for `weather_IAR.gxp` if you are using IAR EW for Synergy.



**Figure 8. Screenshot of the Weather Panel Page being Designed in GUIX Studio**

The organization of the GUIX Studio IDE is straightforward. The center screen, known as the **Target View**, contains the screen being designed. On the upper left corner, you will find the **Project View**. This pane shows the widgets contained in your project. The order that you add items to the project determines the order that they are drawn in the final screens, so some planning is necessary. 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 View** (lower left) displays properties associated with a selected object. You may select objects from the **Project View** or from the **Target View**.

The far-right side of the GUIX Studio screen contains drop down menus for all the various resources such as Colors, Fonts, Images, Pixel maps, and Strings you used to create the screens. GUIX supports multi-language designs using string tables.

The key to making any graphical design interactive is to associate events like screen touches with the event handling code that implements the appropriate functionality. As you design your screens, you associate callback functions with your widgets. These callback functions provide the hooks necessary in your application to respond to GUI events.

GUIX Studio provides both Draw and Event callbacks. Event functions allow you to respond to typical events like touch events. Draw functions allow you to add customized drawing. The Weather Panel application only defines Event Function callbacks and then, only on the top-level windows. The callback function names are entered into the Event function field of the **Properties View** as shown in the following figure.

The Weather Panel GUIX design has two defined Event Functions, named as follows:

- `mainpage_event`
- `splashscreen_event`

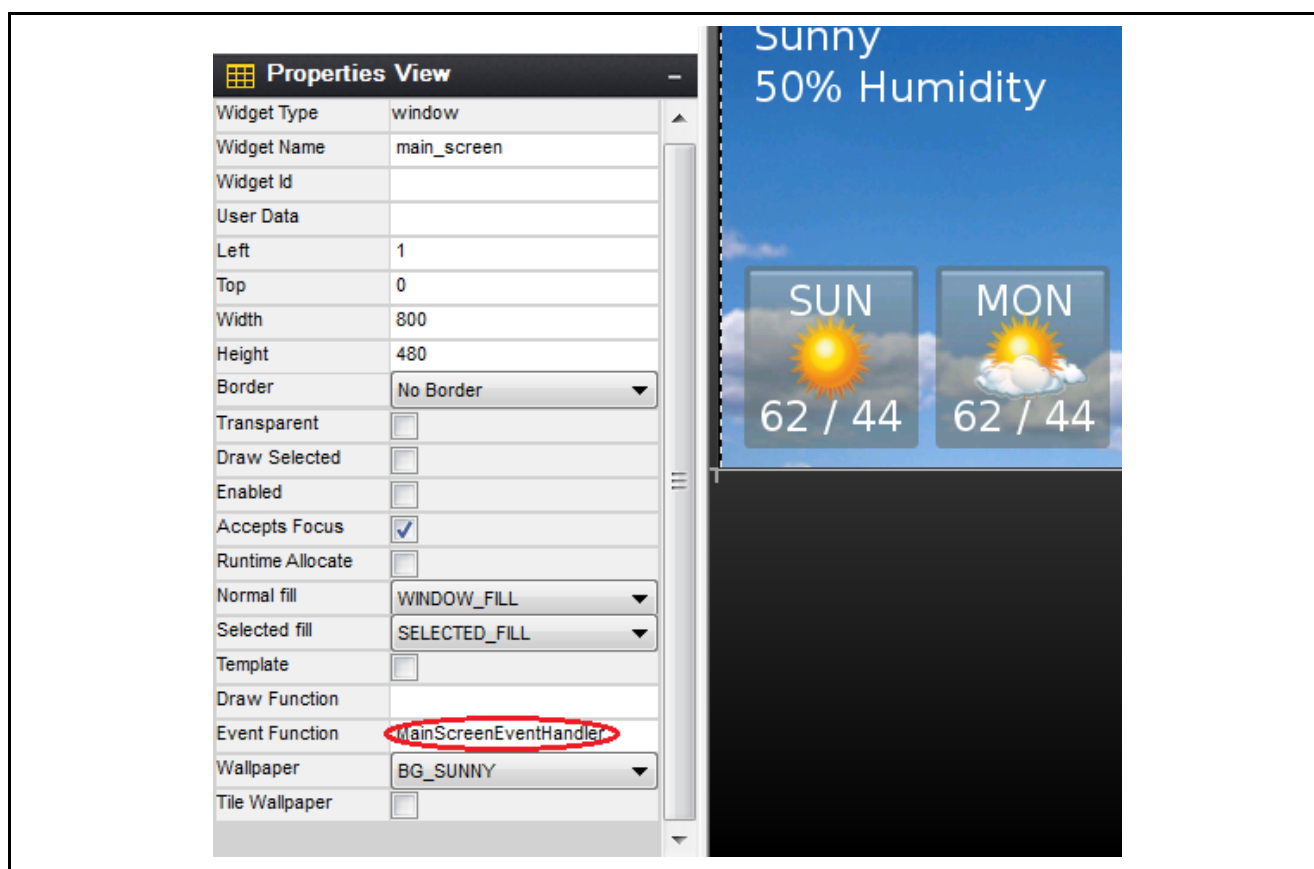


Figure 9. GUIX Studio Screen Properties View

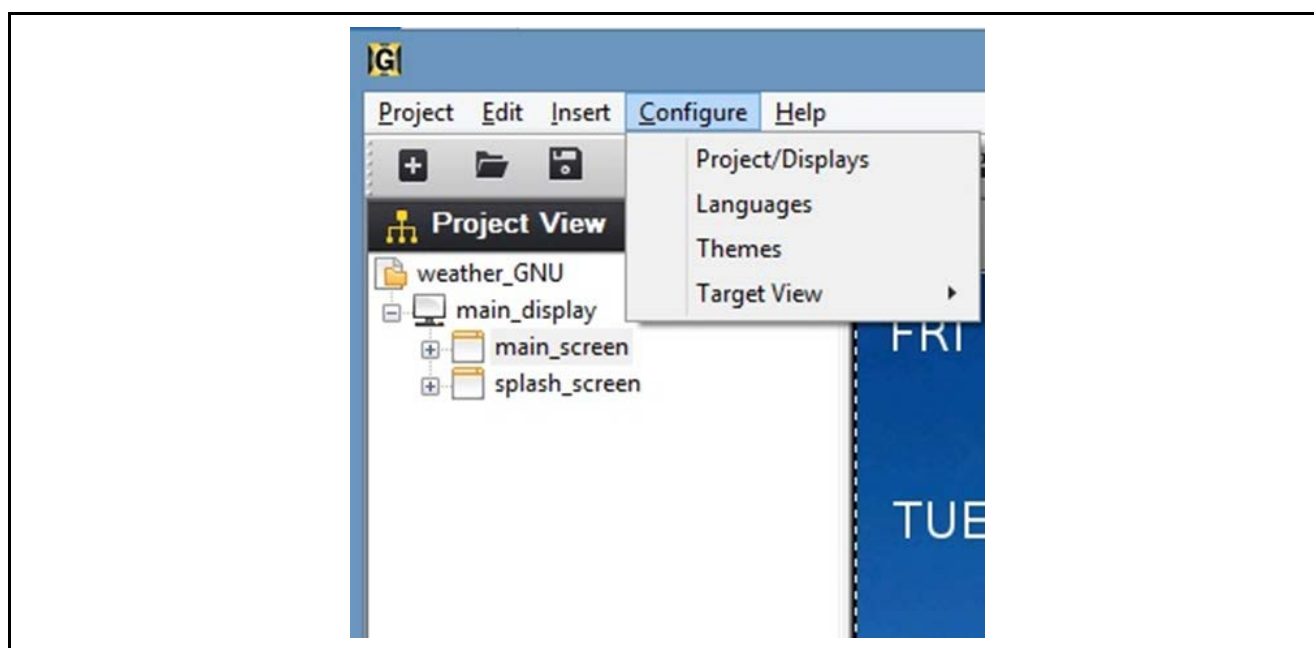
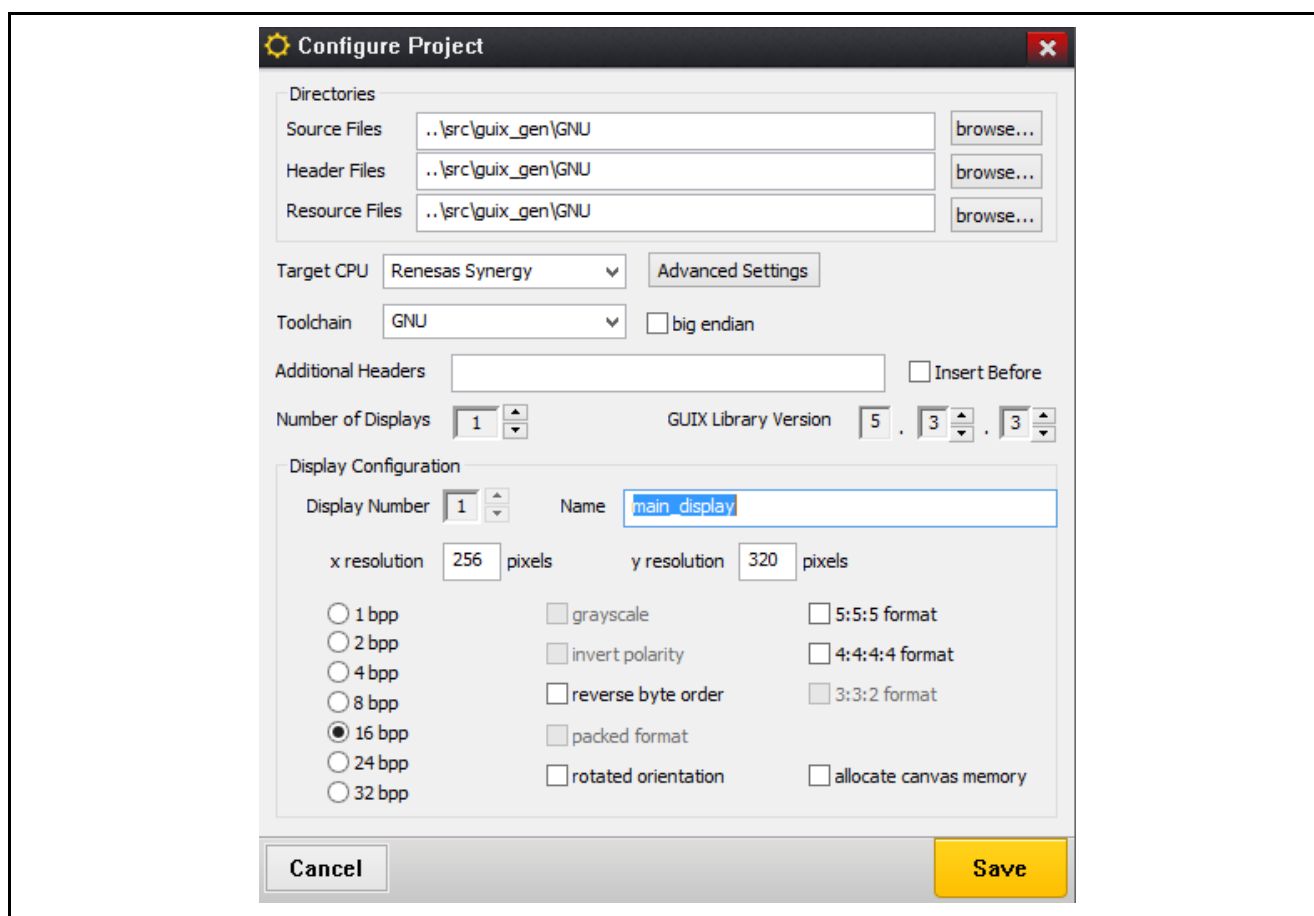


Figure 10. GUIX Studio Project Configuration View

This presents you with the Configure Project dialog box as shown in Figure 11 for e<sup>2</sup> studio and Figure 12 for IAR EW for Synergy. 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 GUIX locates the files that result from the **build** process.

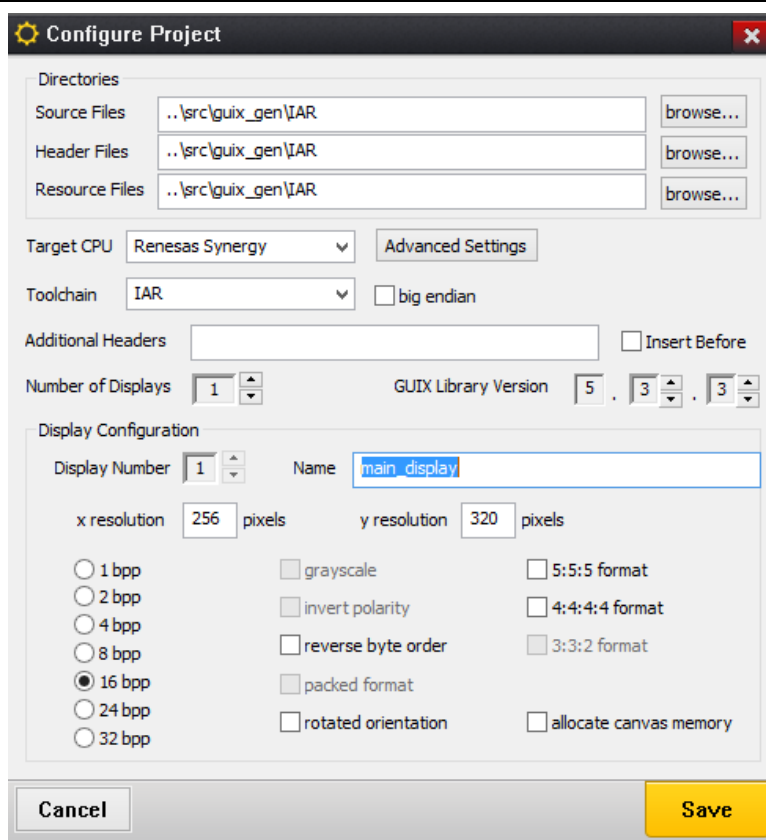
When you build your project, GUIX Studio creates .c and .h files that contain all the information necessary to render the screens you built with GUIX Studio on the LCD in your embedded Synergy application. The

**Directories** group is where you specify the default output directories for the source and header files. You may also specify a directory location where all the resource files, such as images are saved.



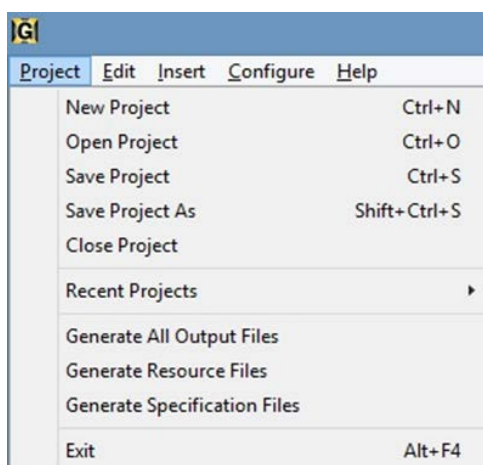
**Figure 11. GUIX Project Settings Window for e<sup>2</sup> studio**





**Figure 12. GUIX Project Settings Window for IAR EW for Synergy**

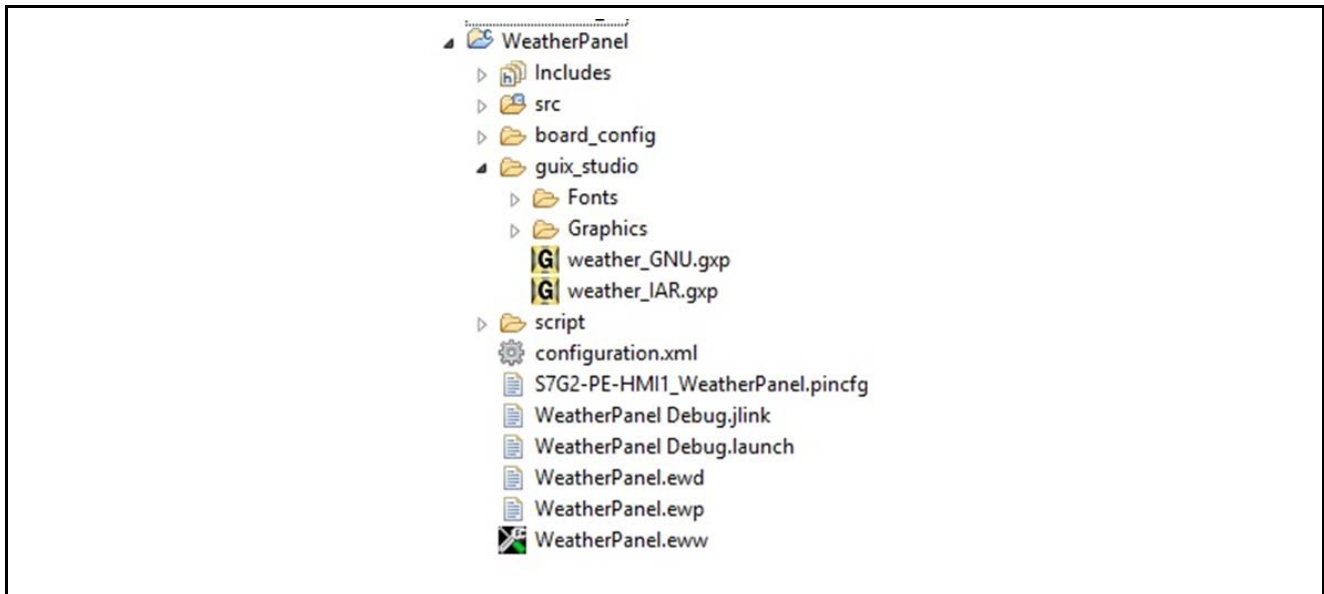
It is a good practice to save the Source, Header, and Resource files relative to the project location. This makes it easy to move projects from one location to another or from one PC to another. In the case of the Weather Panel application, you can see that all the directories are located under the `src\guix_gen\GNU` directory for e<sup>2</sup> studio `src\guix_gen\IAR` for IAR EW for Synergy.



**Figure 13. GUIX Project Generate Source Files**

When you are finished with your GUIX Studio design, you instruct GUIX Studio to generate all the output files by selecting **Project > Generate All Output Files** as shown in Figure 13.

The Weather Panel application has a `guix_studio` directory containing the original resource files and the `weather_GNU.gxp` (`weather_IAR.gxp` for IAR EW) file as shown in Figure 14. If you have GUIX Studio installed, you simply click the `weather_GNU.gxp` file to launch GUIX Studio.



**Figure 14. GUIX Project File View in the Weather Panel Application**

As mentioned earlier, the outputs of GUIX Studio are simple .c and .h source files that need to be compiled into your project. The GUIX project for the PE-HMI1 Weather Panel application contains two Event Handler functions, one for each top-level screen. GUIX automatically builds function prototypes for these callback functions in the `weather_GNU_specifications.h` for e<sup>2</sup> studio and `weather_IAR_specifications.h` for IAR EW file as follows:

```
/* Declare event process functions, draw functions, and callback functions
*/
UINT MainScreenEventHandler (GX_WINDOW *widget, GX_EVENT *event_ptr);
UINT SplashScreenEventHandler (GX_WINDOW *widget, GX_EVENT *event_ptr);
```

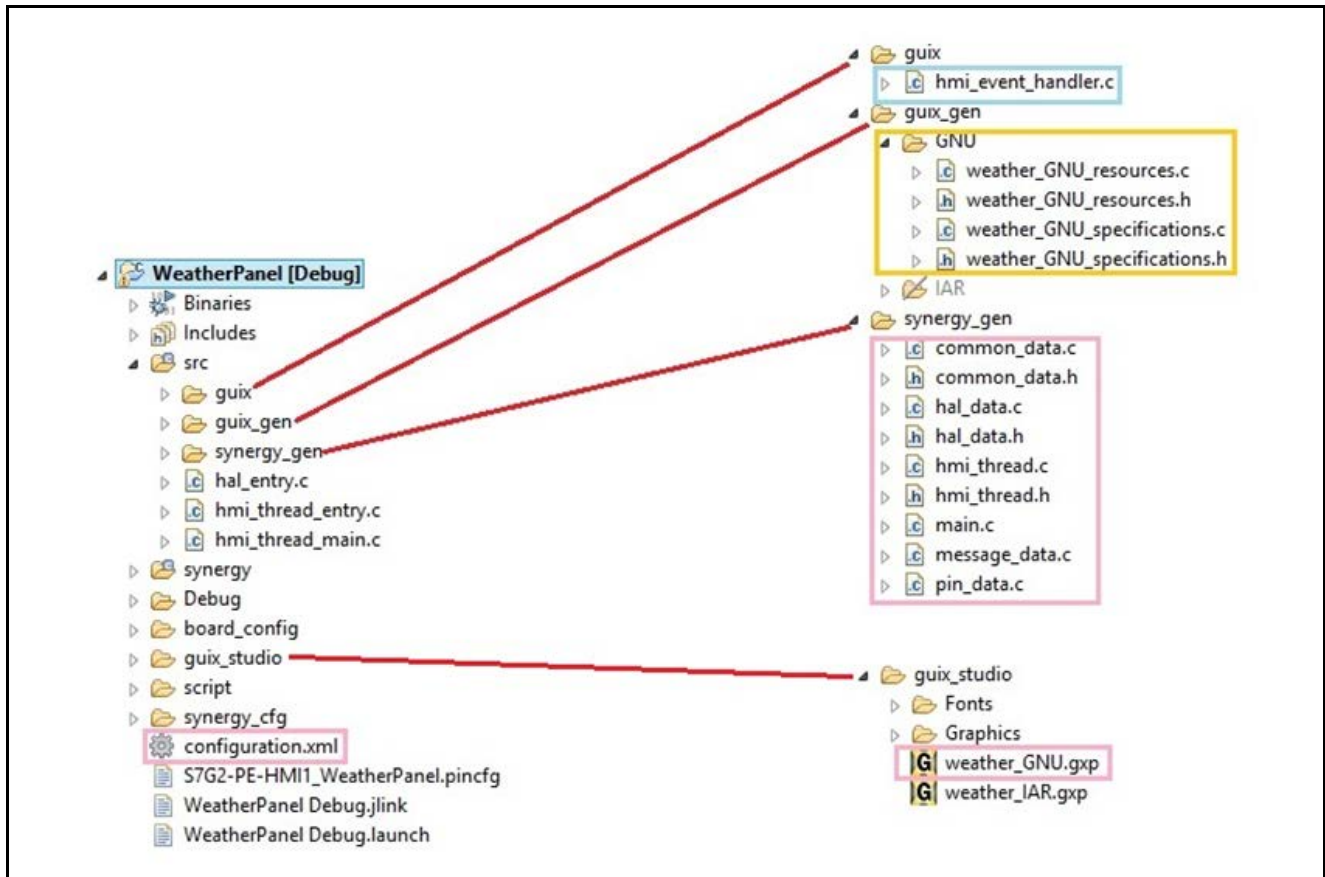
While GUIX Studio defines function prototypes for event handlers, you must create the file that contains the actual code for each of these handlers. The next section details the actual source code layout of the Weather Panel application, as also shown in Figure 15. The event handler code resides in the `hmi_event_handler.c` file.

## 4. Analyzing the 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 Synergy™ Platform applications. These include how the project is physically structured in Synergy e<sup>2</sup> 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.

### 4.1 Source Code Layout

Prior to diving into the actual application code, it is best to first understand the overall source code layout of a Synergy project. The Renesas Synergy™ Platform applications generally consist of two different types of code, your generated 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 Synergy Framework, and code that is auto-generated by GUIX Studio.



**Figure 15. Weather Panel Project Source File Layout**

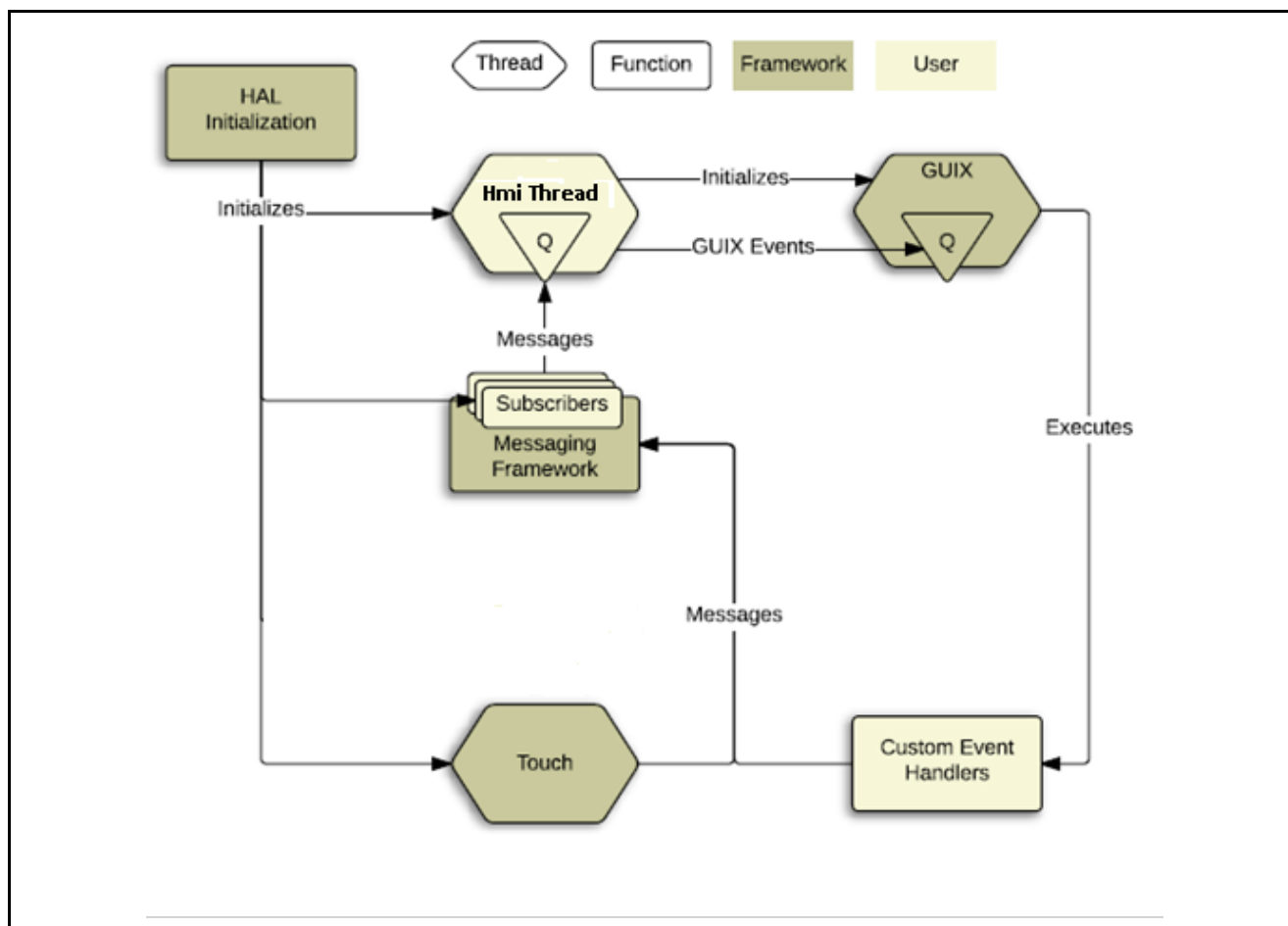
The figure above shows the source code layout for the PE-HMI1 board. Framework auto-generated code is highlighted in red, GUIX auto-generated code is highlighted in yellow, and the code you generated is highlighted in green.

Note: Most of your generated code resides in the `src` directory except for the GUIX Studio project file, `weather_GNU.gxp` (`weather_IAR.gxp` in case of IAR EW for Synergy), and the `hmi_event_handler.c` file, that contains the event handlers for the HMI.

## 4.2 Thread Overview

As mentioned in the introduction, the Weather Panel application is a multi-threaded application, running under ThreadX®. There are two origins of threads in a Synergy application, those created by you, and those created by the Framework. While it is obvious as to what threads you created, it is not always obvious as to what threads are created by the Framework. As explained in the *SSP User's Manual*, there are two principle types of modules that you add to a Synergy application, namely, Driver modules and Framework modules. Driver modules are described as RTOS aware but do not generally use any RTOS objects. Framework layer modules are free to use RTOS objects, such as semaphores or mutexes and may also create their own threads as needed.

The Weather Panel application uses a user-created thread, the HMI Thread. Threads communicate through the Synergy messaging framework, that is layered on top of the standard ThreadX message queues. The HMI Thread processes touch messages and GUIX events. The Framework Configuration section details how to add your threads to your application. The following figure shows a high-level design of the threads and messaging running on the Weather Panel application. Notice the distinction between your threads and Framework threads. As you can see from the following figure, in addition to the HMI thread, there are threads associated with GUIX and the touch controller.



**Figure 16. Weather Panel Application Message Sequence Flow**

In addition to the software components, various hardware components are also accessed through the hardware drivers provided by the Synergy Framework. These include the clock generation circuit, touch screen controller (I<sup>2</sup>C), and external interrupts unit of the Arm core processor.

#### 4.2.1 HMI Thread

The HMI thread initializes various services used by the Weather Panel application, including GUIX. On the SK-S7G2 and PK-S7G2 Synergy MCU boards, the HMI thread must also initialize the LCD screen to place it into the proper RGB mode so that it may be controlled by the GLCD peripheral of the processor.

Once this initialization is complete, the HMI thread processes touch messages, and GUIX events. If any of these inputs result in a change to the system state, the HMI Thread sends the appropriate update messages to the GUIX thread, resulting in changes to the graphical HMI. The flow chart in Figure 17 illustrates the high-level design of the HMI Thread and the message flow.



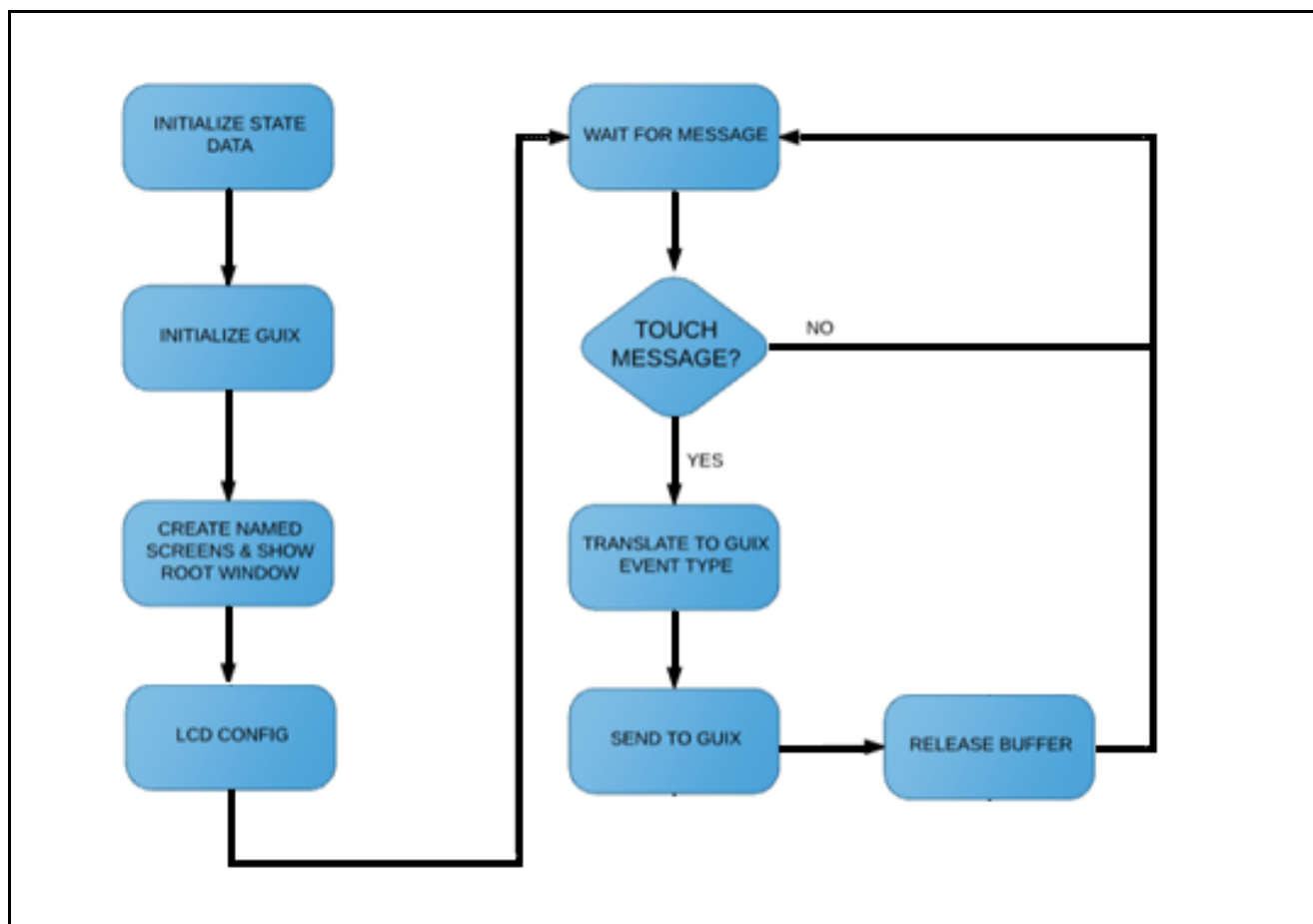


Figure 17. HMI Thread Flow

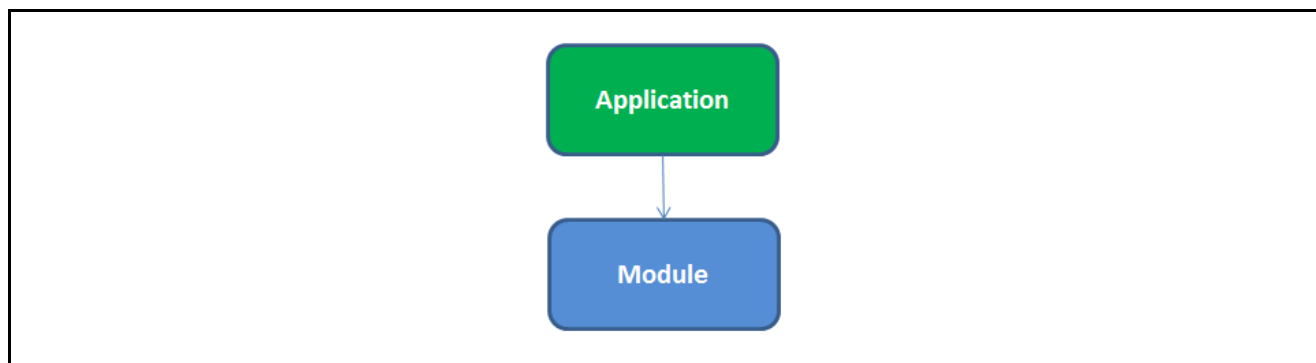
#### 4.2.2 Thread Layout and the SSP

For those new to Synergy, one of the most difficult aspects of learning how to develop complex applications will be learning the various Modules defined under the Synergy Framework, how to add them to your application, and more specifically, how these Modules are layered on top of each other to form SSP stacks.

As described in the *SSP User Manual*, Modules are the core building blocks of the SSP. Modules provide functionality upwards and may require functionality from below. The SSP comes with two predefined layers, the Driver layer, and the Framework layer. The principle difference between the two is that the Driver layer modules are peripheral drivers that are RTOS aware but do not use any RTOS objects or make any RTOS API calls, which means that the Driver layer modules may be used in applications with or without a RTOS. Framework layer modules are free to use RTOS objects such as semaphores, make RTOS API calls, or even create threads as necessary.

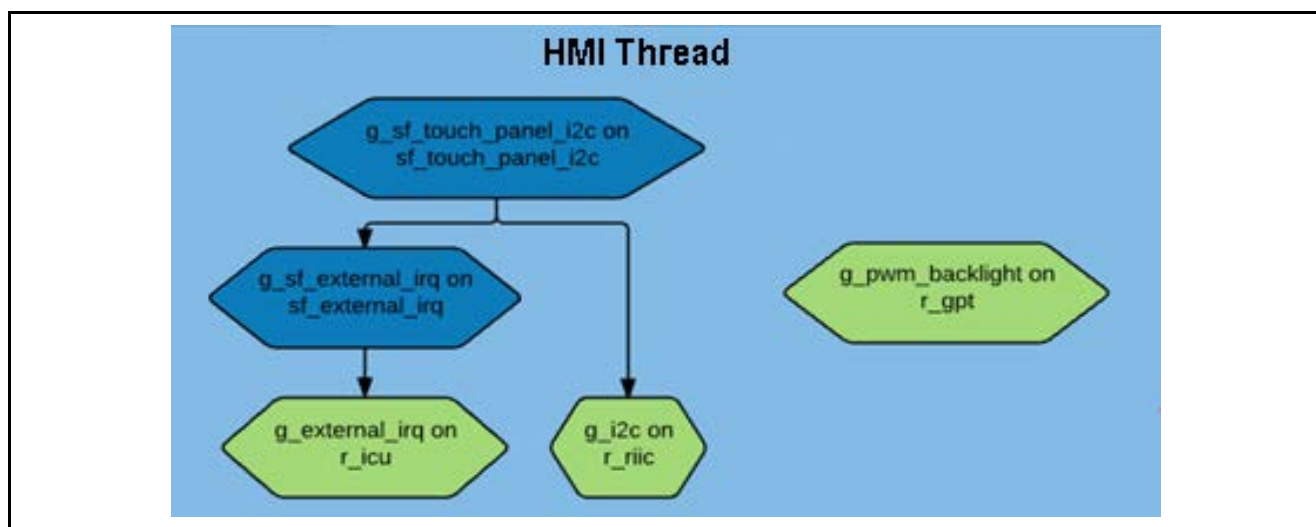
Note: Understanding SSP naming conventions will help you understand Synergy applications. Driver layer module names always start with an `rx_` prefix, while framework modules always start with a `sf_` prefix.

The simplest SSP application consists of one Module with your application on top.



**Figure 18. SSP Application and Module Layer**

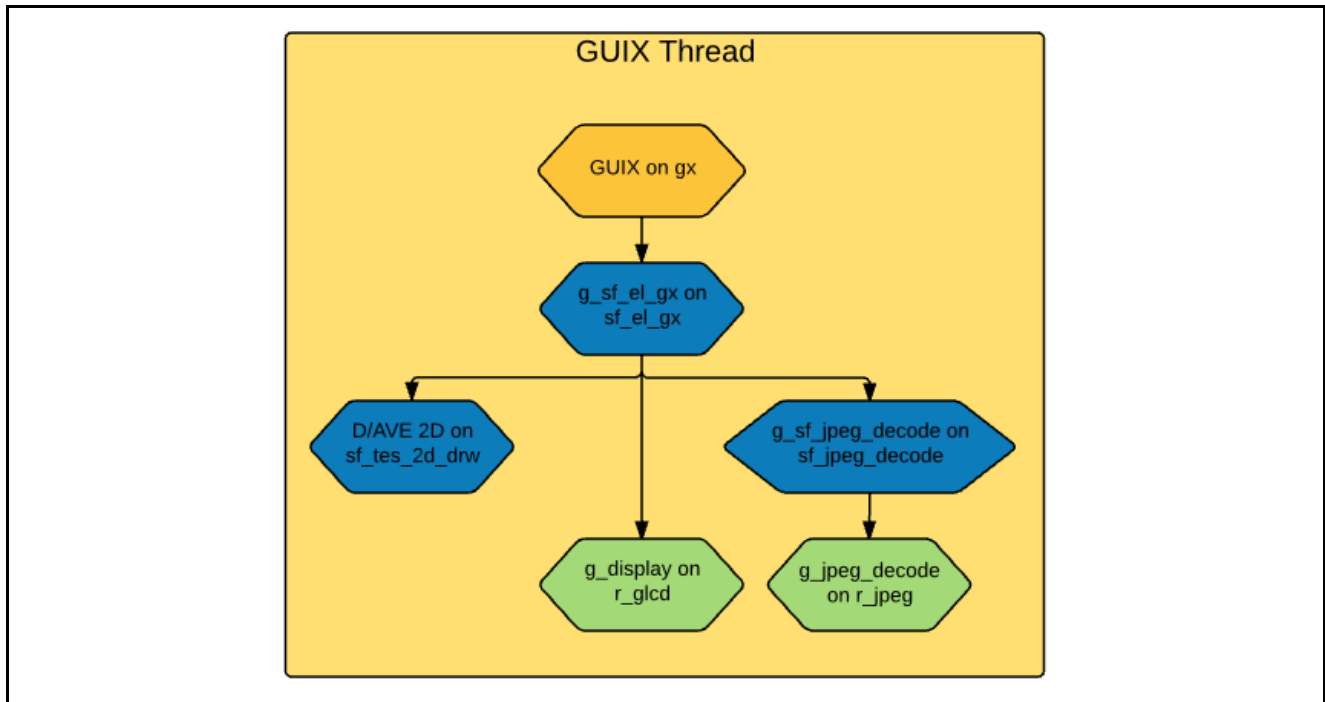
For example, in SSP, all driver **instances** have a name such as `g_gpt` attached to the instance of the driver, such as the `r_gpt` driver in this case. The first thing to recognize is that `r_gpt` is a driver level module due to the `r_` prefix. You will see in the section about Framework Configuration, how these names are assigned to a specific instance of a driver.



**Figure 19. HMI Thread Modules Representation**

These diagrams can become complicated depending on the Modules necessary to accomplish the application's goals. Here you see that the HMI Thread relies on numerous Modules, some of which are layered on top of each other, forming SSP stacks. In the above figure, Framework Modules are represented with a dark blue color, and Driver Modules are represented with a light green color.

The touch controller on most of the development boards generates an IRQ when a touch occurs. The coordinates of the touch are then communicated over the I<sup>2</sup>C bus. In the above figure, the HMI Thread uses the `sf_touch_panel_i2c` module. For interrupt processing, this module requires the `sf_external_irq` module, that in turn requires the `r_icu` module. For the I<sup>2</sup>C communication, the `sf_touch_panel_i2c` module requires the `r_iic` module.



**Figure 20. GUIX Thread Module Representation**

Figure 20 highlights some of the internal threads to fully understand your application architecture under the SSP. Even though it shows a GUIX thread, you never actually create a GUIX thread in your application. The reason is that the `sf_el_gx` module automatically creates this thread when you add the module to your application. The main reason for the GUIX thread box is to have a place holder for the modules you add to your application.

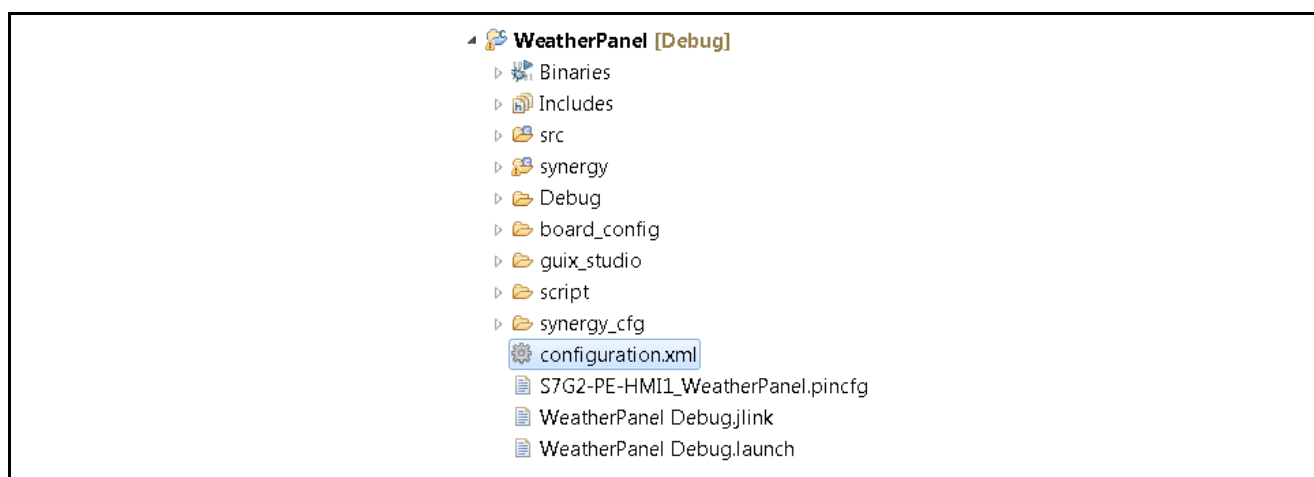
A possible shortcoming of the HMI Thread diagram, as described in the next section, is that the `sf_touch_panel_i2c` module is added under the HMI Thread.

The GUIX Thread utilizes several modules including the `r_glcd` driver module. The S7G2 and S5D9 Synergy MCU Groups include a Graphics LCD (GLCD) controller. This driver module controls that peripheral. It is also one of the more complex modules to understand. This module allows you to define many properties including the screen resolution, where the frame buffer resides, for example, internal versus external memory, the assignment of video sync signals, and so on. If your team designs embedded systems with graphical displays, you will want to have a complete understanding of this module.

## 5. Framework Configuration

One of the first things you must do when writing a Synergy application, is to configure the framework. To properly configure the framework, 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, if 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, queues, and so on that each thread will require. Once you have this information, you will be set to successfully configure the framework for your specific application needs.

In the Weather Panel application, the framework configuration is stored in a file named **configuration.xml**. Double clicking on this file brings up the **Synergy Configuration** tab for the project. It may take a few seconds for e<sup>2</sup> studio to process the xml file.

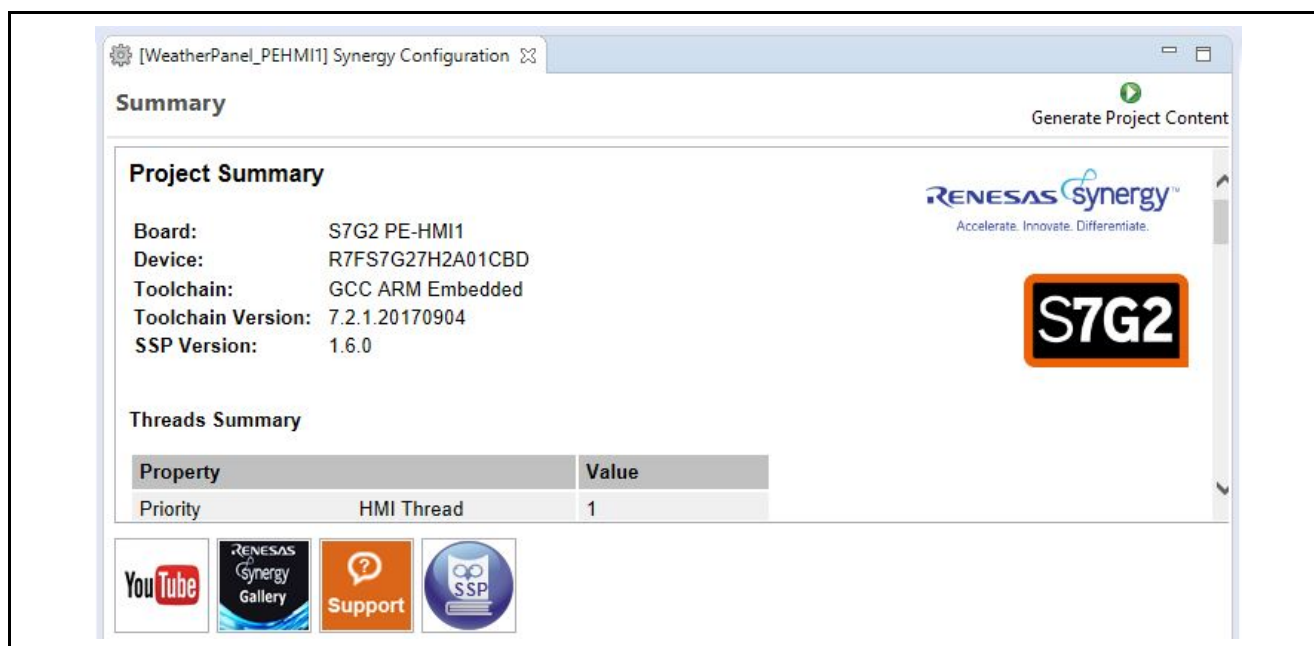


**Figure 21. Snapshot Showing configuration.xml on the Project Pane**

When building a project from scratch, this configuration tab is where you will perform the initial configuration of the Synergy framework. As you can see from the following figure, the Synergy Configuration **WeatherPanel** pane contains a **Summary** screen highlighting the items you may configure, along with a scrolling window that lists all the software components currently selected for this project. Below this scrolling window are tabs that allow you to tailor the framework to the needs of your specific application.

For the purposes of this application note, we will highlight a few of the details of the framework configuration as it pertains to the Weather Panel application. For additional details, refer to the appropriate Synergy Framework documentation, the *SSP User's Manual*, and application notes on how to configure the Synergy Framework.

When you have configured the project appropriately, click the **Generate Project Content**, the green arrow button above the summary screen, to build all the auto-generated files necessary to implement the components you defined.

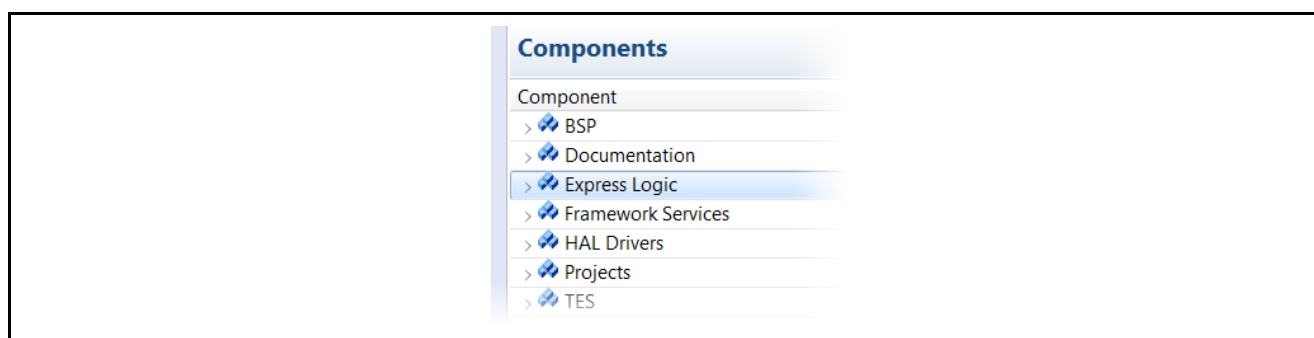


**Figure 22. Summary of the Weather Panel Application Configuration**

## 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 **Threads** tab. One of the advantages of the Synergy framework is that it will only compile in the components you choose, thereby reducing the size of your overall application. As shown in the following figure, components are broken down into seven categories.



**Figure 23. 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 Weather Panel application. One of the nice features of the **Components** tab is it gives you a description of the component and shows dependencies for each component. As an example, notice that the `sf_message` (Messaging Framework) component requires ThreadX. This dependency listing helps eliminate compile time errors that would result from failing to choose the proper dependent components when making your component selections.

**Table 3. Components Used in the Weather Panel Application**

Category	Component	Version	Description
BSP	<code>s7g2_pe_hmi1</code>	1.6.0	Board Support Package for S7G2_PE_HMI1
Express Logic	<code>gx</code>	1.6.0	Express Logic GUIX: Provides=[GUIX], Requires=[ThreadX]
	<code>tx</code>	1.6.0	Express Logic ThreadX: Provides=[ThreadX]
Framework Services	<code>sf_el_gx</code>	1.6.0	SF_EL_GX GUIX Adaption Framework: Provides=[SSP GUIX Adaption Framework], Requires=[ThreadX, GUIX]
	<code>sf_external_irq</code>	1.6.0	Framework External IRQ: Provides=[Framework External IRQ], Requires=[External IRQ, ThreadX]
	<code>sf_jpeg_decode</code>	1.6.0	Framework JPEG Decode: Provides=[SF JPEG Decode], Requires=[ThreadX, JPEG Decode]
	<code>sf_message</code>	1.6.0	Messaging Framework: Provides=[Message], Requires=[ThreadX]
	<code>sf_tes_2d_drw</code>	1.6.0	TES Dave/2d(DRW) Framework: Provides=[SF_TES_2D_DRW], Requires=[ThreadX, TES Dave/2d]
	<code>sf_touch_panel_i2c</code>	1.6.0	Framework Touch Panel using I2C: Provides=[Framework Touch Panel], Requires=[ThreadX, Message, I2C, Framework External IRQ]
HAL Drivers	<code>r_cgc</code>	1.6.0	Clock Generation Circuit: Provides=[CGC]
	<code>r_dtc</code>	1.6.0	Data Transfer Controller: Provides=[Transfer]
	<code>r_fmi</code>	1.6.0	Factory MCU Information Module: Provides=[FMI]
	<code>r_riic</code>	1.6.0	RIIC: Provides=[I2C Slave]
	<code>r_elc</code>	1.6.0	Event Link Controller: Provides=[ELC]
	<code>r_glcd</code>	1.6.0	Graphics LCD: Provides=[Display]
	<code>r_gpt</code>	1.6.0	General Purpose Timer: Provides=[Timer, GPT]
	<code>r_icu</code>	1.6.0	External IRQ: Provides=[External IRQ]
	<code>r_ioport</code>	1.6.0	I/O Port: Provides=[IO Port]
	<code>r_jpeg_decode</code>	1.6.0	JPEG Decode: Provides=[Key Matrix]
TES	<code>touch_panel_i2c_ft5x06</code>	1.6.0	Touch panel i2c ft5x06 driver
	<code>dave2d</code>	1.6.0	TES Dave/2d: Provides=[Dave/2d]

## 5.2 Threads Tab

The **Threads** tab is where you can add and review the threads that the framework automatically creates for your application. You define a new thread by clicking the  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** and then single click on the **HMI Thread**, you should see something like the screen capture shown in the next figure. This shows that the HMI thread requires multiple modules, the I<sup>2</sup>C driver which is used to read the on-touch sensor of the PE-HMI1.

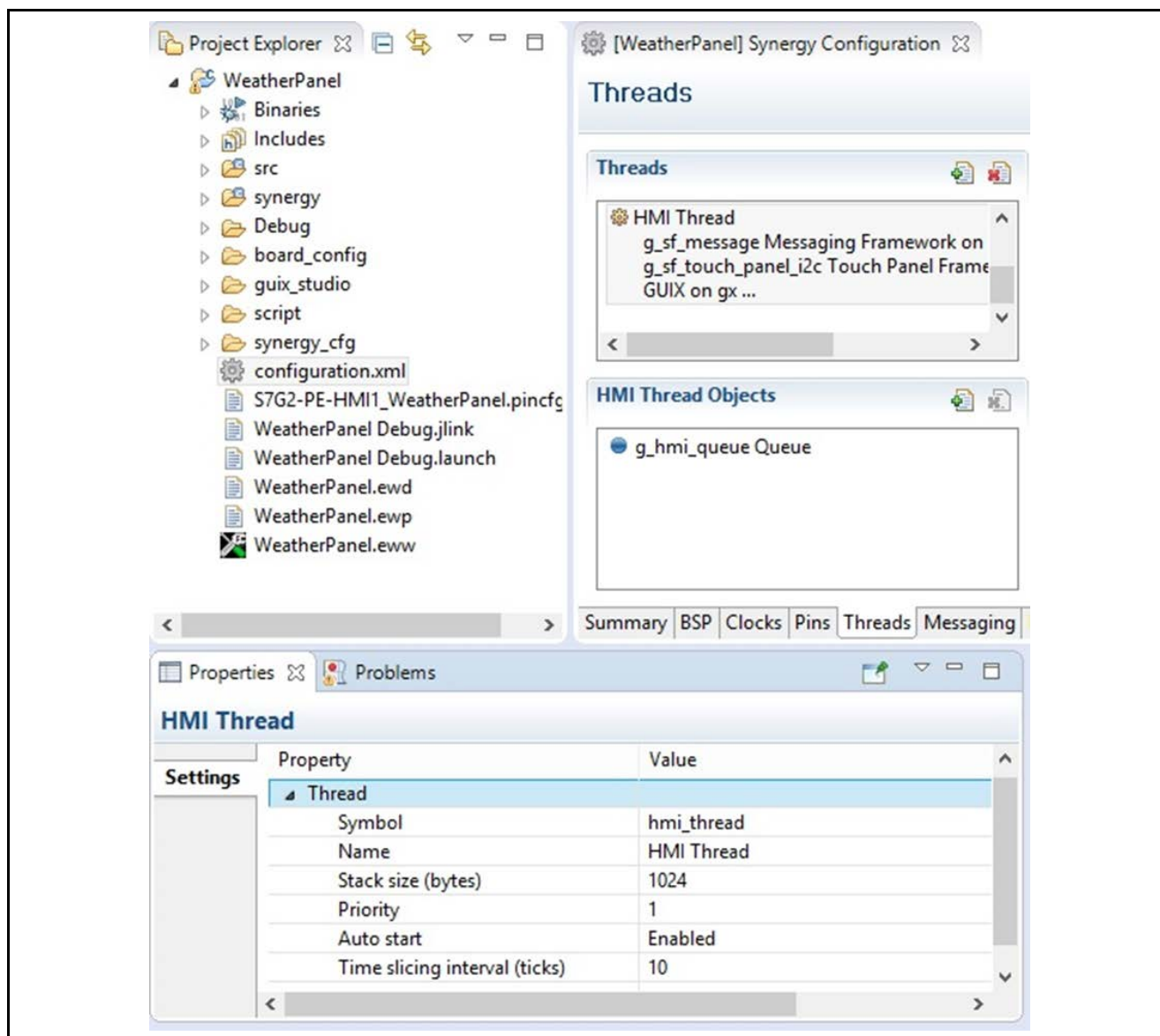


Figure 24. HMI Thread Properties

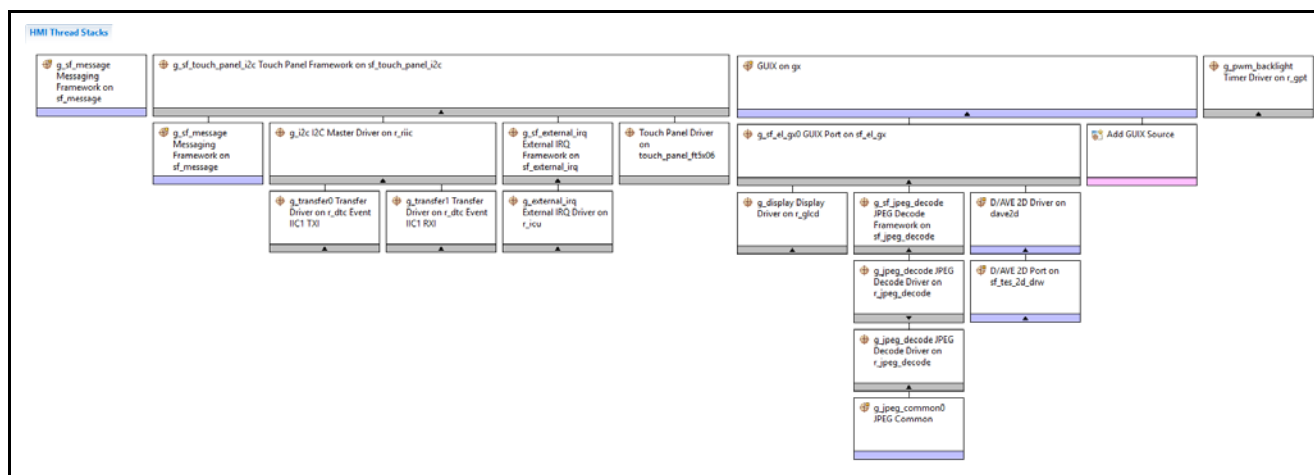


Figure 25. HMI Thread Modules for the Weather Panel Application

You can add additional Modules to any thread by clicking the (+) button. If you have chosen the appropriate components prior to adding Modules to your threads, you should not receive any errors. As an example, the figure below shows you how to add a timer to the HMI Thread. The timer is added by choosing **Driver > Timers > r\_gpt**.

If you pick a Module that you have not preselected, the appropriate component for first, the Framework automatically selects the component for you. If the framework detects errors with the Module addition, it prefaces the Module with an error. You may examine the errors by hovering over the Module name.

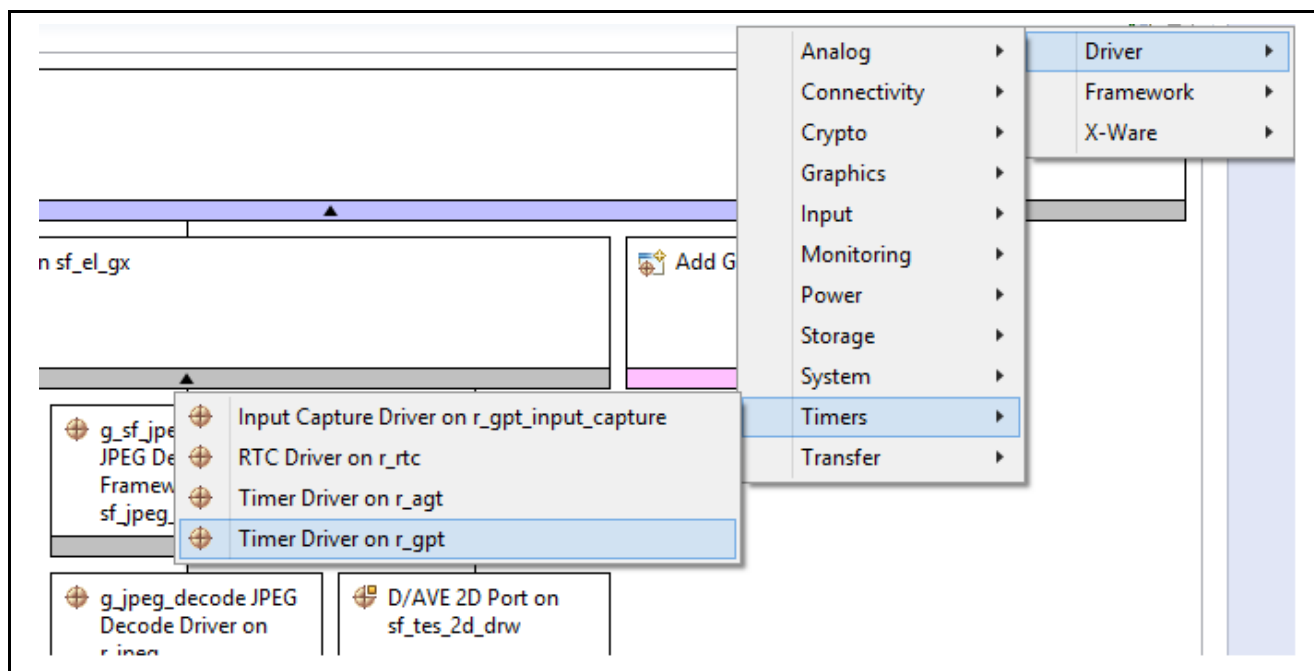
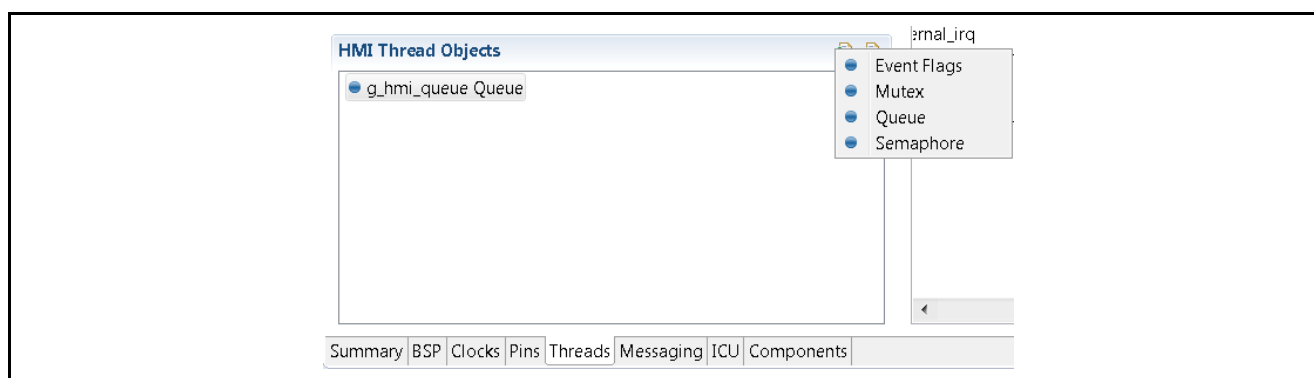


Figure 26. Adding PWM Timer to HMI Thread

### 5.3 Thread Objects

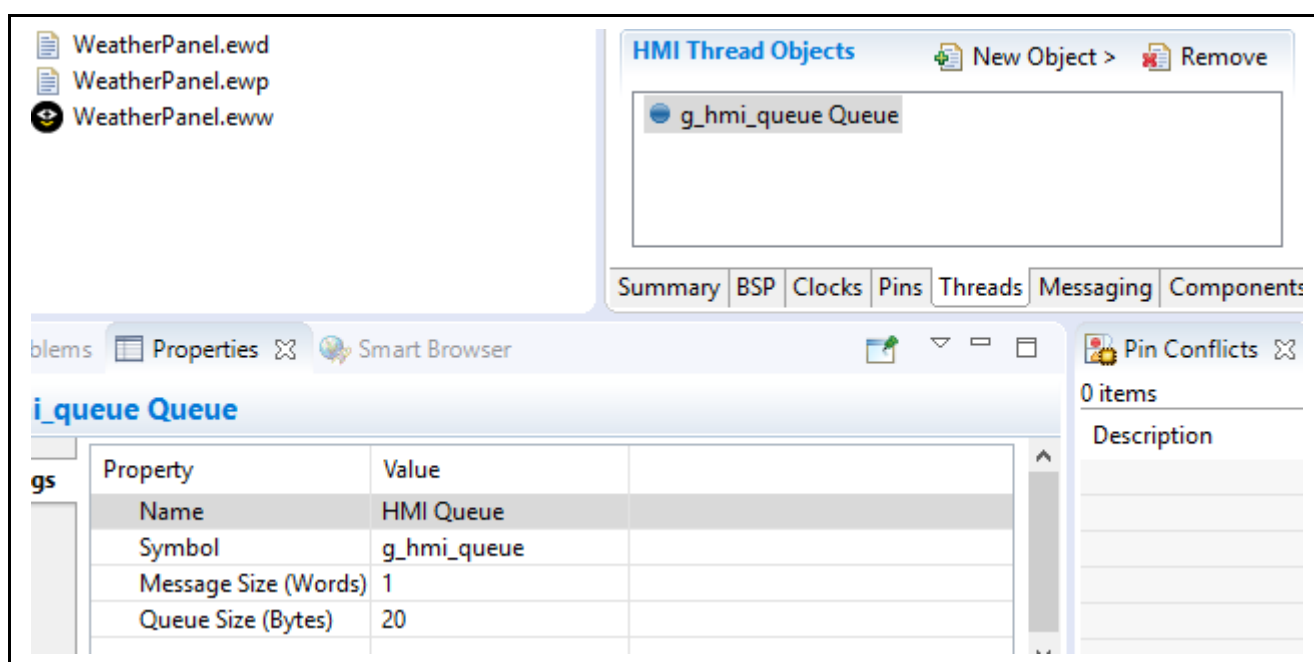
ThreadX supports various objects such as Event Flags, Mutexes, Queues, and Semaphores. If you click on **HMI Thread** in the **Threads** window, you will see that there is one Queue object, **g\_hmi\_queue**, allocated for this thread.

You can allocate additional thread objects for any thread by selecting the thread from the left-hand **Threads** window and then clicking on the (+) button next to the **Thread Objects** window. As you can see from the following figure, after clicking the button (+) in the **Thread Objects** window, you are presented with a drop-down list that allows you to add the standard thread objects supported by ThreadX.



**Figure 27. Adding Thread Objects and Objects Supported by ThreadX**

When adding, or reviewing threads, thread modules, or thread objects, you will want the **Properties** tab enabled so you can examine or change the properties associated with the item. If your **Properties** tab is not showing, you can show it by going to **Window > Show View > Other... > General** and then selecting **Properties**. As an example, Figure 28 shows that the System Queue has a message size of 1 word and a Queue size of 20 bytes. To change these values, simply update them in the **Properties** view and then click the **Generate Project Content** button to update your project code with the new value.



**Figure 28. HMI Thread Object g\_hmi Queue Properties**

## 5.4 Module Configuration

Once you have added Driver or Framework Modules to your project, you need to configure their properties. The properties are dependent on the driver(s) that you have added. Use the **Properties** tab to configure them. The Weather Panel application adds the `r_glcd` driver module. This module is used to configure the GLCD peripheral of the ARM Cortex®-M4 MCU. While the properties of each development board may differ slightly, the process of configuring these properties is generally the same on all the development boards.

### 5.4.1 GLCD Configuration

As you can see from Figure 29, selecting the `g_display` Display Driver on the `g_glcd` module under the HMI Thread Modules dialog 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 Module grouping.

The Module group is where you configure the GLCD 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_display` by default and the name of a user defined callback function if used. The Weather Panel application does not use a callback.
- **Input:** This block of module properties defines the input to the graphics controller, most notably, the size of the frame buffer, source of the dot clock, where the frame buffer is located, and others. This section allows you to define two graphics screens. The Weather Panel application only uses one screen, the Input-Graphics Screen 1 is set to be used.
- **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 to the GLCD. This divisor 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.

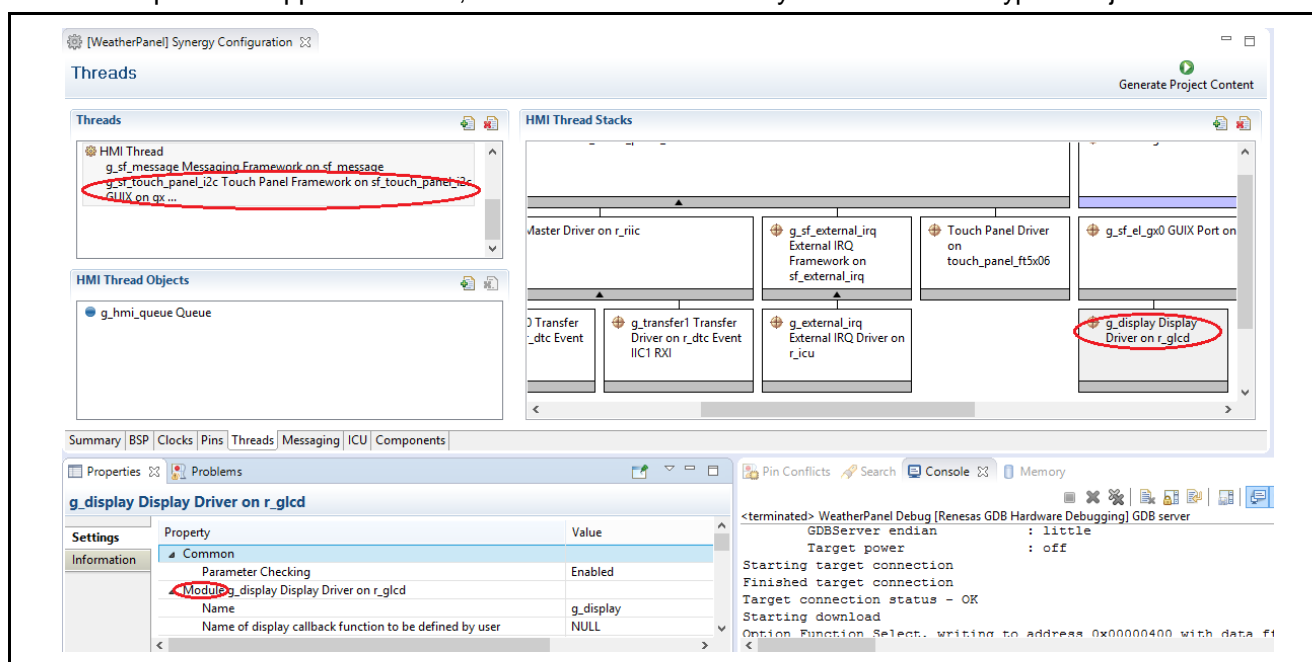


Figure 29. GLCD Properties Configuration using the Properties Tab

### 5.4.2 TCON Configuration

If you scroll down a little farther 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 dot clock that is driven directly from the PLL0UT 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.

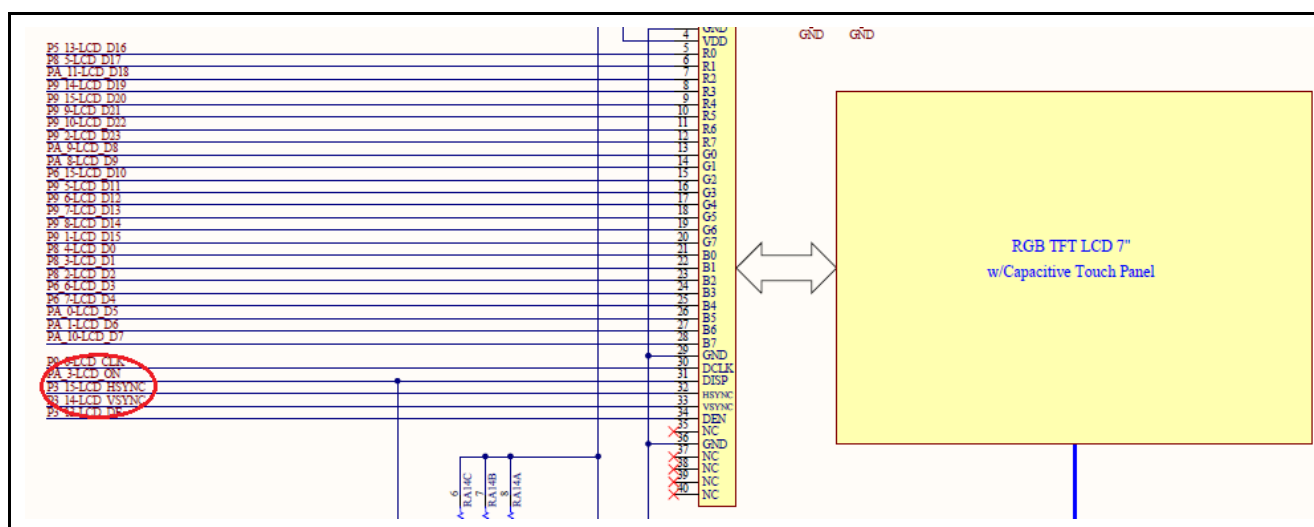
TCON - Hsync pin select	LCD_TCON0
TCON - Vsync pin select	LCD_TCON1
TCON - DataEnable pin select	LCD_TCON2
TCON - Panel clock division ratio	1/8

Figure 30. TCON Configuration for PPE-HMI1 Board LCD

To provide some flexibility, the GLCD controller of the S7G2 MCU provides two pin grouping options. 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 pick the group of pins they want to use. Picking one or the other may free up MCU pins that are necessary in some other part of the hardware design.

If you look at the schematics for the PE-HMI1 board, you can see all the pins connected to the LCD data lines. You will also notice the four pins connected to the sync signals, that are highlighted in red. The data

lines chosen by the hardware designer must match one of the two pin groupings available under the GLCD module. A little extra flexibility is provided for the LCD sync signals.

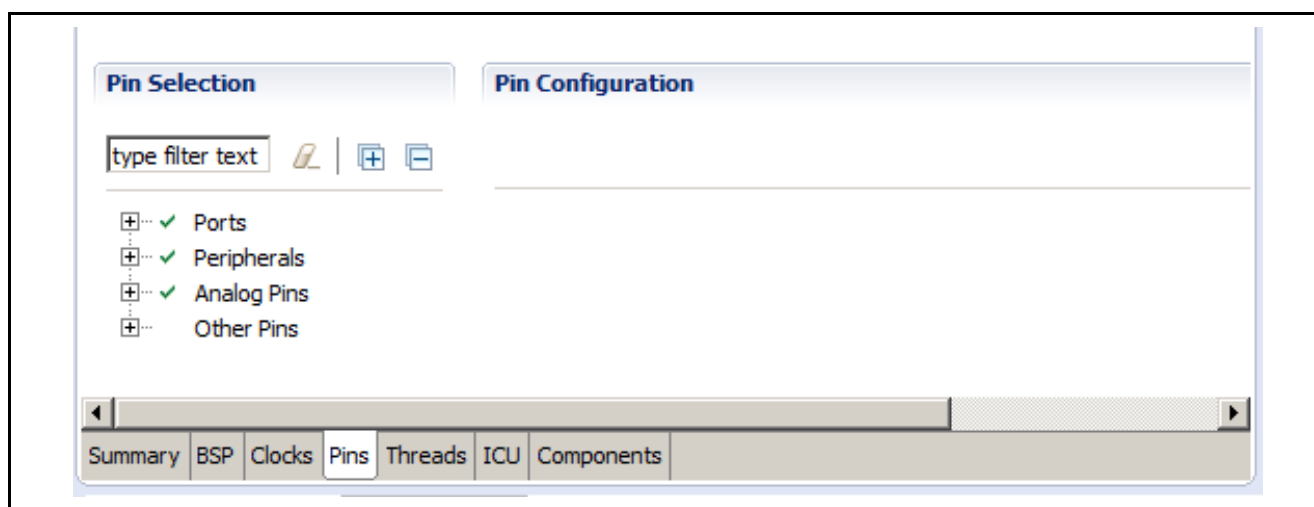


**Figure 31. PE-HMI1 LCD-specific Signals from the Schematics**

The easiest way to understand this is to go to the **Pins** tab in the Synergy Configuration window. You will see selections for **Ports**, **Peripherals**, **Analog Pins**, and **Other Pins**, as shown in Figure 31. If you expand the **Peripherals** dialog, you will see all the various ARM core peripherals that can be configured from this screen.

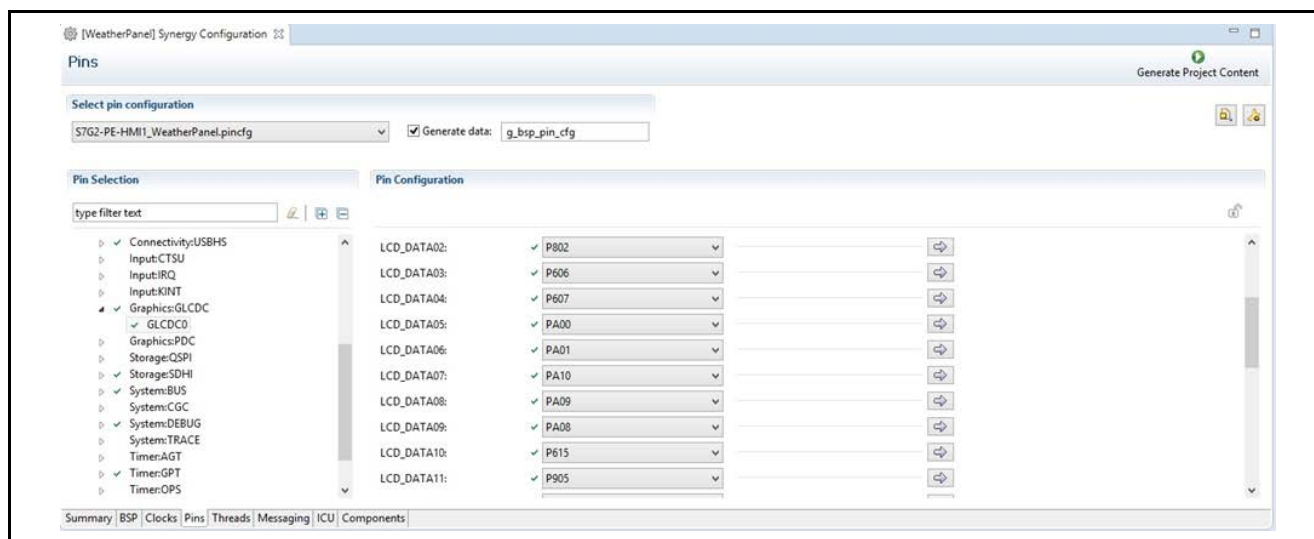
If you scroll down to the LCD\_GRAPHICS entry and click the small plus sign next to it, you will see two options GLCD\_Controller\_Pin\_Option\_A and GLCD\_Controller\_Pin\_Option\_B. There should be a green check mark next to GLCD\_Controller\_Pin\_Option\_B indicating that this is the pin group associated with driving the LCD display.

Notice that TCON0 is associated with the Port 3 Pin 15 (P315). On the schematic (P3\_15) we see that it is connected to LCD\_DE, which is the data enable pin for this screen. Referring back to Figure 30, we see TCON0 has been selected to drive the DataEnable signal.



**Figure 32. PE-HMI1 Pin Configuration Tab**

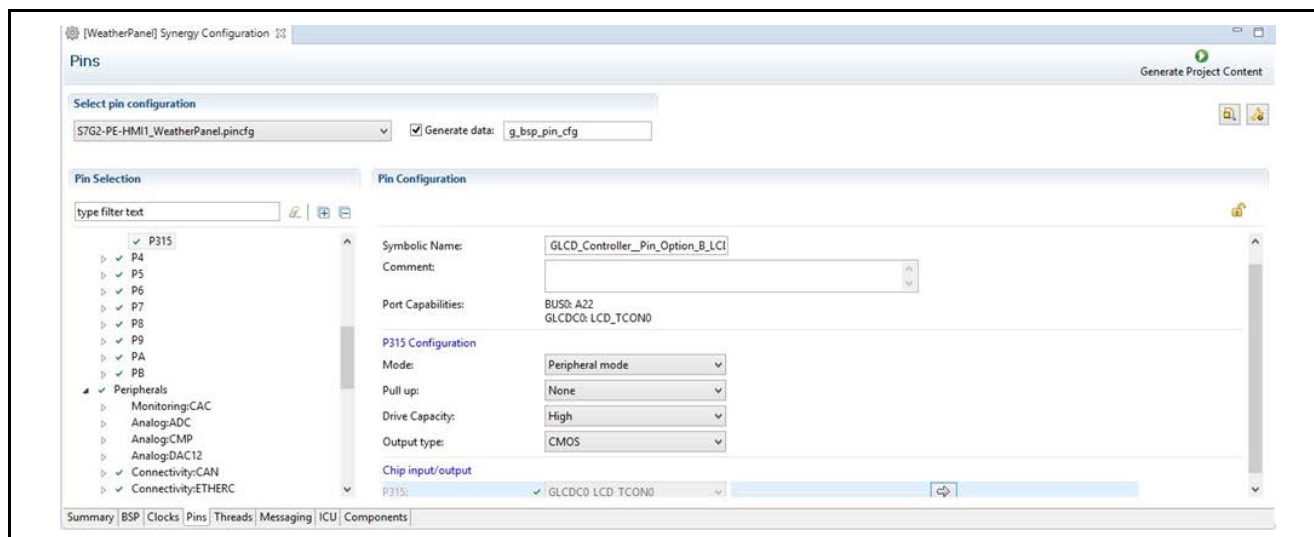
If you look at all the LCD data lines such as LCD\_DATA\_DATA00, and the pins they are connected to, they should match the pins they are connected to on the schematic. Clicking on the arrow to the right of the pin brings you directly to the associated **Pin Configuration** dialog just as if you had selected the Ports Group, and then the specific port and pin that you are interested in.



**Figure 33. LCD Pin Configuration Using Configurator**

For example, clicking on this arrow next to the `LCD_TCON0` pin should bring you to the **Pin Selection Screen** that looks like Figure 34. Notice that the Pin is appropriately set to the **Peripheral mode**. At the time of writing this application note, the pins default to no Pull Up, Low Drive Capacity, and CMOS output type. Clicking on the arrow button to the right of this screen brings you back to the associated peripheral screen.

**Note:** At the time of writing this application note, when you select option A or B of the `LCD_GRAPHICS` peripheral, you must manually enable each pin connected to your display. Using the arrow button to toggle back and forth between the **Peripheral** screen and the **Pin Configuration** screen makes this process easier.

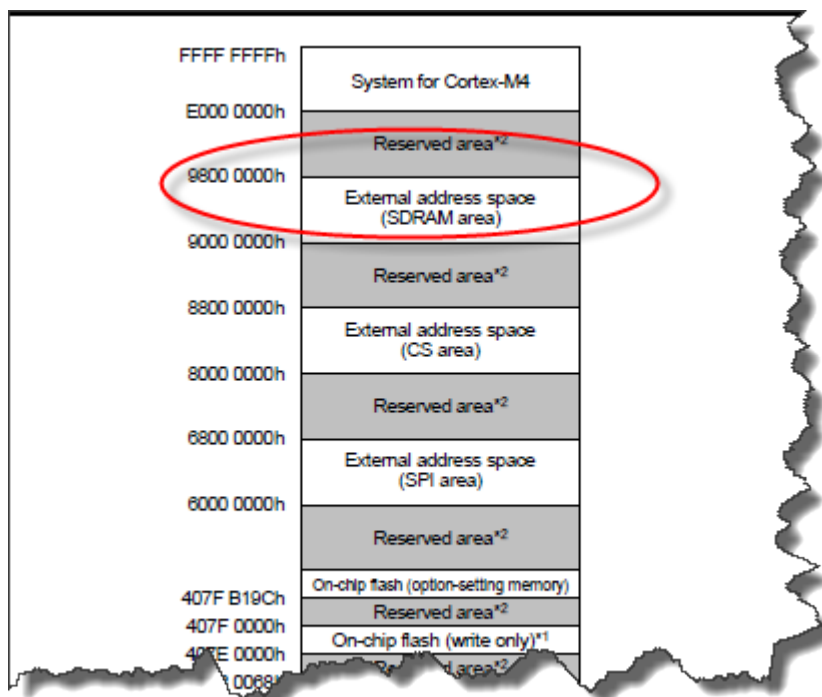


**Figure 34. Pin Configuration Tab**

### 5.4.3 Using External Memory for Frame Buffer

One of the differences between a lower cost development board, like the SK-S7G2 Synergy MCU board, and the more expensive PE-HMI Synergy MCU board, is the availability of an external memory area for the screen buffer. As the screen size and color depth increases, or a more sophisticated display strategy is used (such as ping pong frame buffering), the available internal memory of the microcontroller may not be sufficient. In this case, an external memory device is usually added to the board.

The SK-S7G2 Synergy MCU directly supports the use of an external SDRAM. Figure 35 shows an excerpt from the SK-S7G2 Synergy MCU memory map found in Chapter 4 of the *S7 Series MCU User's Manual*. You can see here that the SDRAM address space is associated with address 9000 0000h to 9800 0000h.



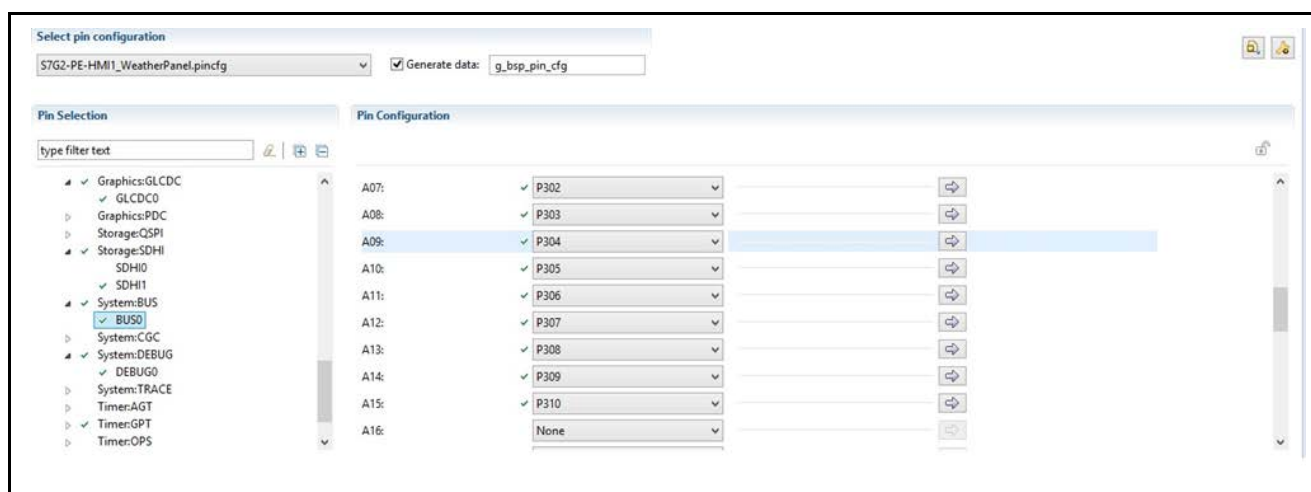
**Figure 35. SDRAM Memory Area of the PE-HMI1 Board**

Two steps are required to use the external SDRAM for your frame buffer. The first is to locate the following property under the **Properties** tab for the GLCD controller, and change it from `bss` to `sdram`. By convention, the `bss` abbreviation instructs the SSP to place the frame buffer in internal memory in a section named `bss`.

Input - Graphics screen1	Used
Input - Graphics screen1 frame buffer name	fb_background
Input - Number of Graphics screen1 frame buffer	2
Input - Section where Graphics screen1 frame buffer allocated	sdram

**Figure 36. Selecting the SDRAM for the Frame Buffer**

The second step is to configure the External Memory Interface. This is like configuring the GLCD controller that we just discussed above. Return to the **Pins** tab of the Synergy Configuration window, expand the **Peripherals** selection, and then expand the bus selection. Change the **Operation Mode** to **Enabled** and then manually enable each line used by your SDRAM by switching back and forth between the **Peripheral** view and the **Pin Configuration** view, using the arrow to the right of the pin name.



**Figure 37. Configuring External Memory Interface**

Once you have configured the External Memory Bus correctly and changed the GLCD property to point to SDRAM, you may not see any difference on your display. This may be because your current screen resolution fits fine inside the internal memory. When you change to external memory, your GLCD continues to drive the screen, just as it did before, only now, it is pulling the frame buffer from your external SDRAM. How do you know the screens are really being driven from external memory?

#### 5.4.4 e<sup>2</sup> studio Tricks

The e<sup>2</sup> 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 external SDRAM. To use this feature, make sure to connect the e<sup>2</sup> studio to your board and run the program under the debugger. Ensure that your **Memory** tab is open in the **Console** window, normally located to the bottom of the screen in **Debug** view. Click the small green plus (+) sign to add a memory monitor. You should see a **Monitor Memory** dialog as shown in Figure 38. From the ARM memory map above, enter the External Address Space associated with the SDRAM area in hex format (0x90000000) 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.

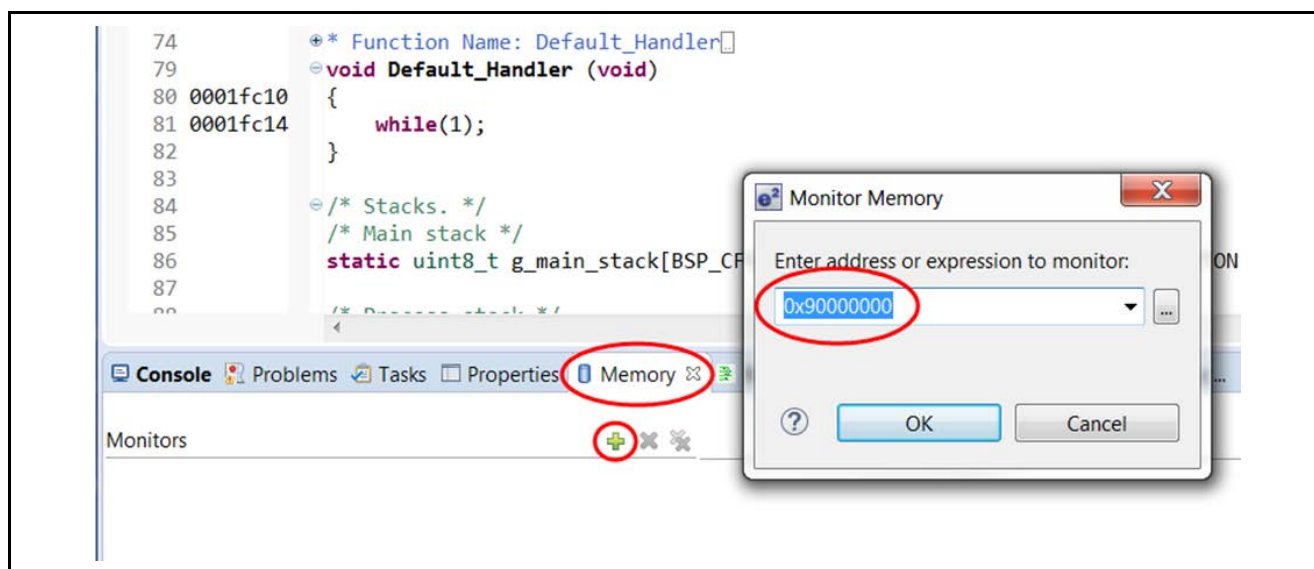


Figure 38. Using the Memory Monitor to Display the SDRAM Contents

You should now see the contents of the SDRAM 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 external SDRAM. 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.

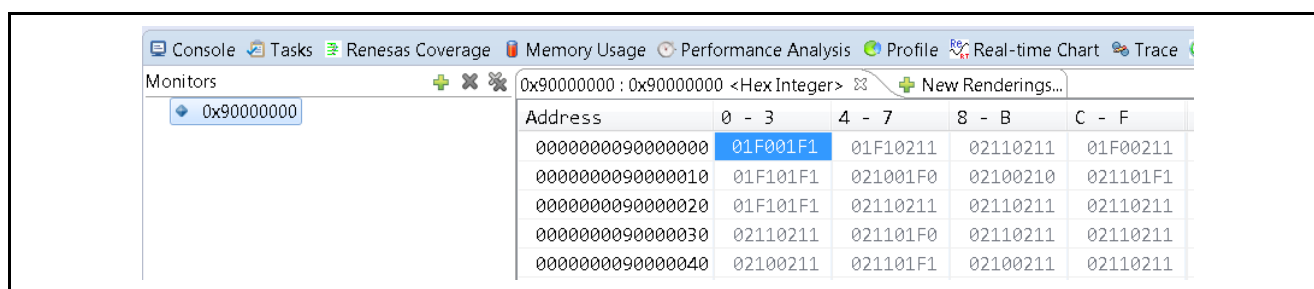
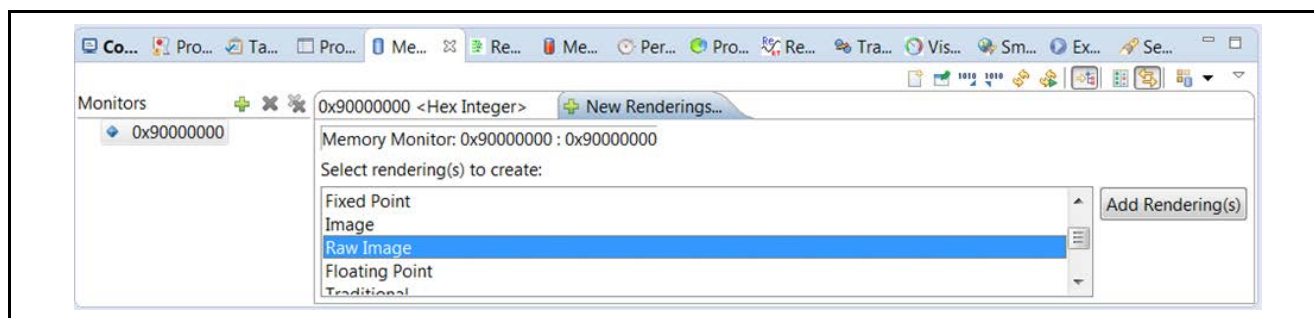


Figure 39. SDRAM Contents

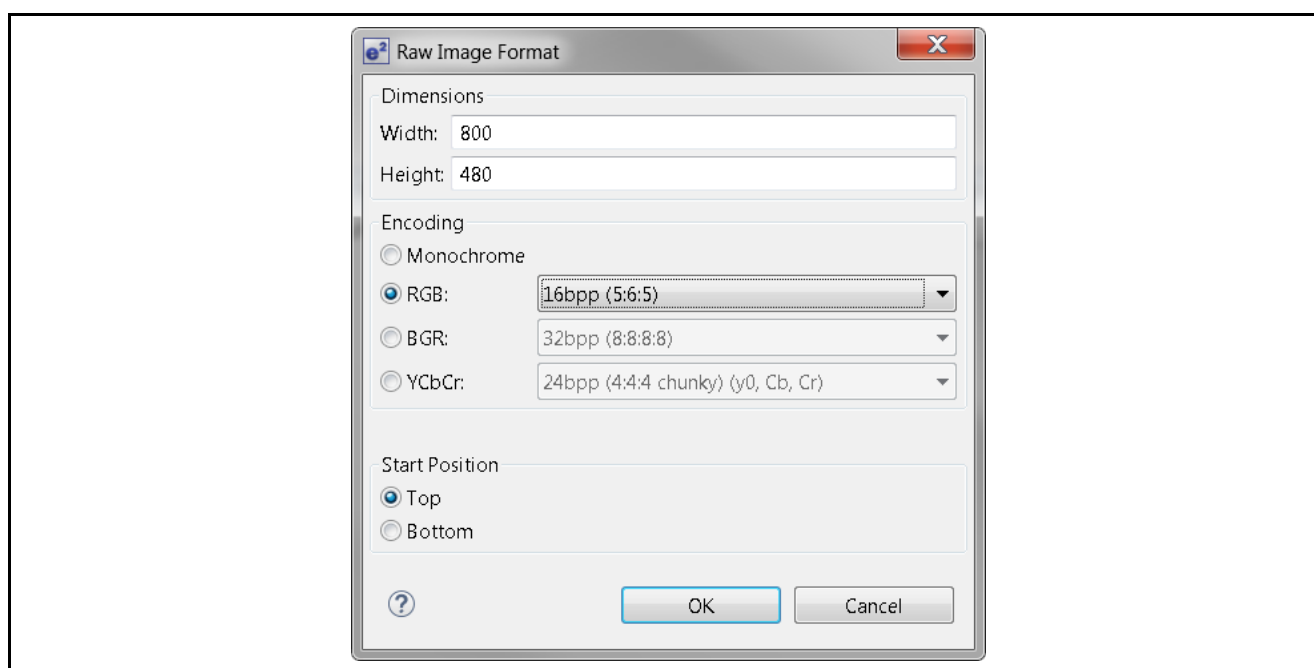
Select the **New Renderings** tab next to the memory monitor you just created, select **Raw Image** type from the list of options, and press the **Add Rendering(s)** button off to the right side of the screen.





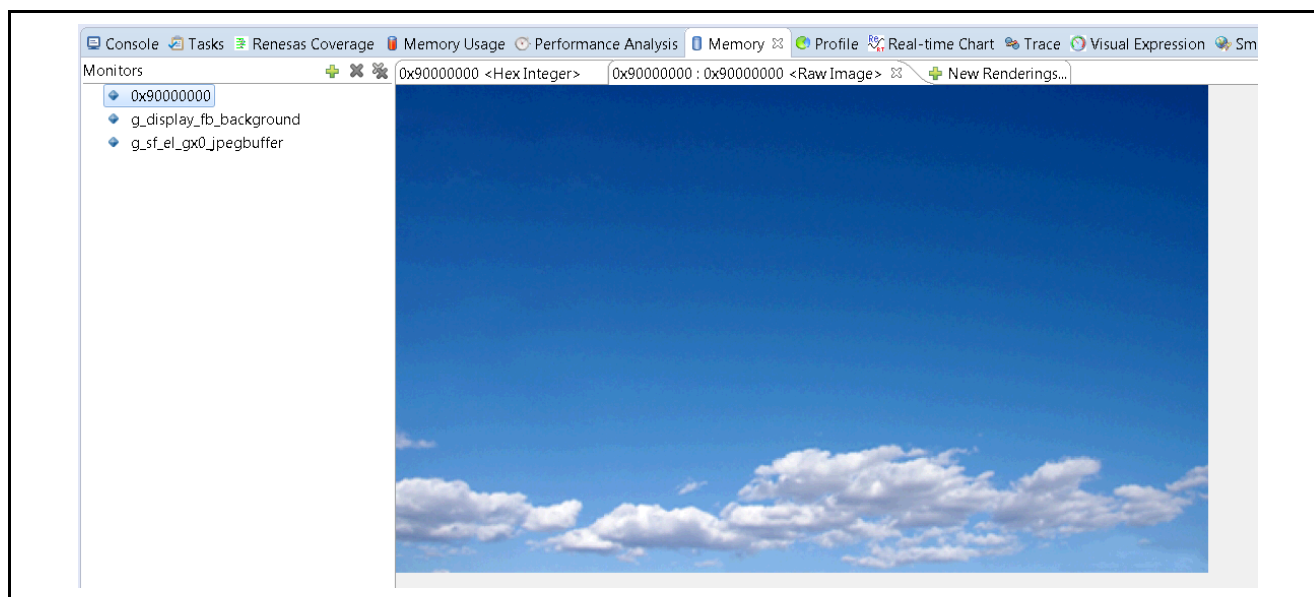
**Figure 40. GUIX Rendering Format Selection**

The **Raw Image Format** dialog box appears, that lets you enter the screen resolution Width and Height, along with the Encoding that is 16 bpp (5:6:5), in our case.

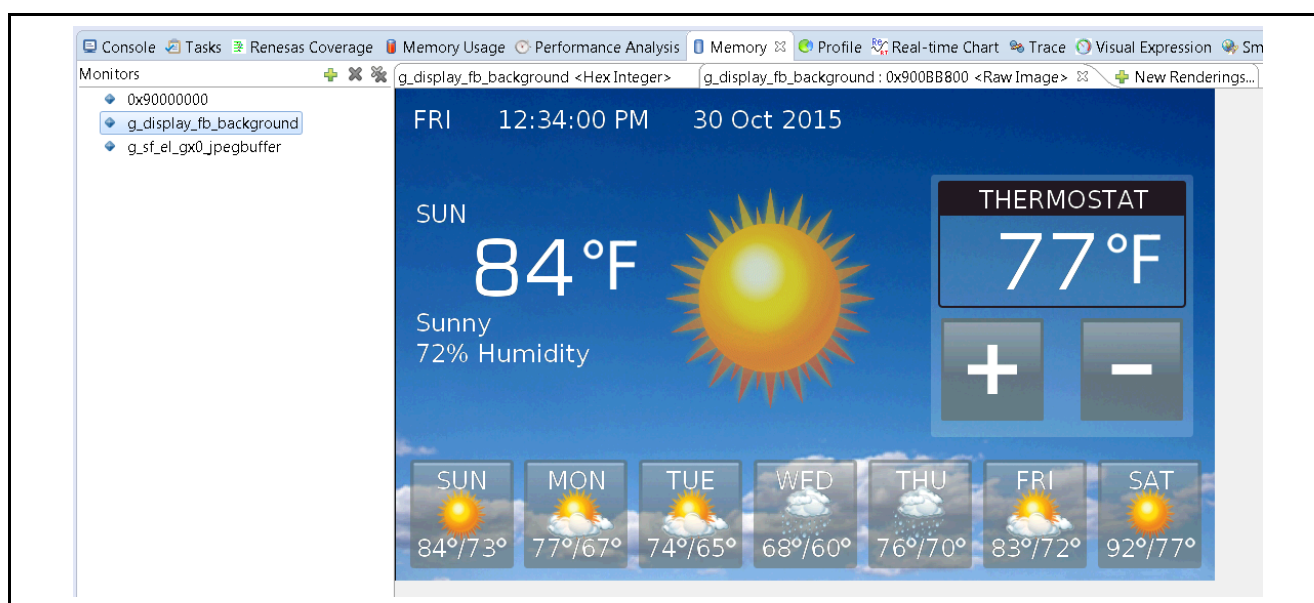


**Figure 41. Raw Image Format for Weather Application on PE-HMI1**

Once you press the **OK** key, the memory monitor presents you with the image that would be displayed at that memory address, based on the parameters you entered. You can switch back to the memory monitor tab, modify memory locations, and see the image change on both the memory monitor and the actual LCD screen. You could perform these same steps when running out of internal memory, but first, you must reference the linker map to determine where in the `.bss` section the screen memory was mapped, open a memory monitor on that location, and repeat the steps above.



**Figure 42. GUIX Rendering Seen Using e² studio Memory Monitor**



**Figure 43. GUIX Rendering Seen Using e² studio Memory Monitor**

## 6. Application Code Highlights

This section details the highlights of the Weather Panel application. The goal of the Weather Panel application is to show you how to develop more complex multi-threaded HMI applications using ThreadX and GUIX with the SSP.

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

### 6.1 Threads and Main

There are a few subtle differences when using ThreadX with the SSP environment. In a typical ThreadX application, `main()` calls `tx_kernal_enter()`, which then calls `tx_application_define()`. If you have written ThreadX applications prior to working with Synergy, you may be used to creating the main application threads and defining other resources used by the application such as queues and semaphores in `tx_application_define()`.

With the Synergy framework, `main()` is an auto-generated file which looks like the following code. In this case, `tx_application_define()` calls thread entry functions for the threads specified during the framework configuration.

```
void tx_application_define(void* first_unused_memory)
{
    hmi_thread_create ();

#ifdef TX_USER_ENABLE_TRACE
    TX_USER_ENABLE_TRACE;
#endif

    g_hal_init ();

    tx_application_define_user (first_unused_memory);
}

void main(void)
{
    __disable_irq ();
    tx_kernel_enter ();
}
```

When you create a thread using the **Threads** tab, the framework creates several files. As an example, when the **HMI Thread** was added, the framework created three files for you: `hmi_thread.h`, `hmi_thread.c`, and `hmi_thread_entry.c`, as shown in the following figure.

The first two files are auto-generated and therefore put into the `synergy_gen` folder. The `hmi_thread_entry.c` file is the entry point for the **HMI 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.

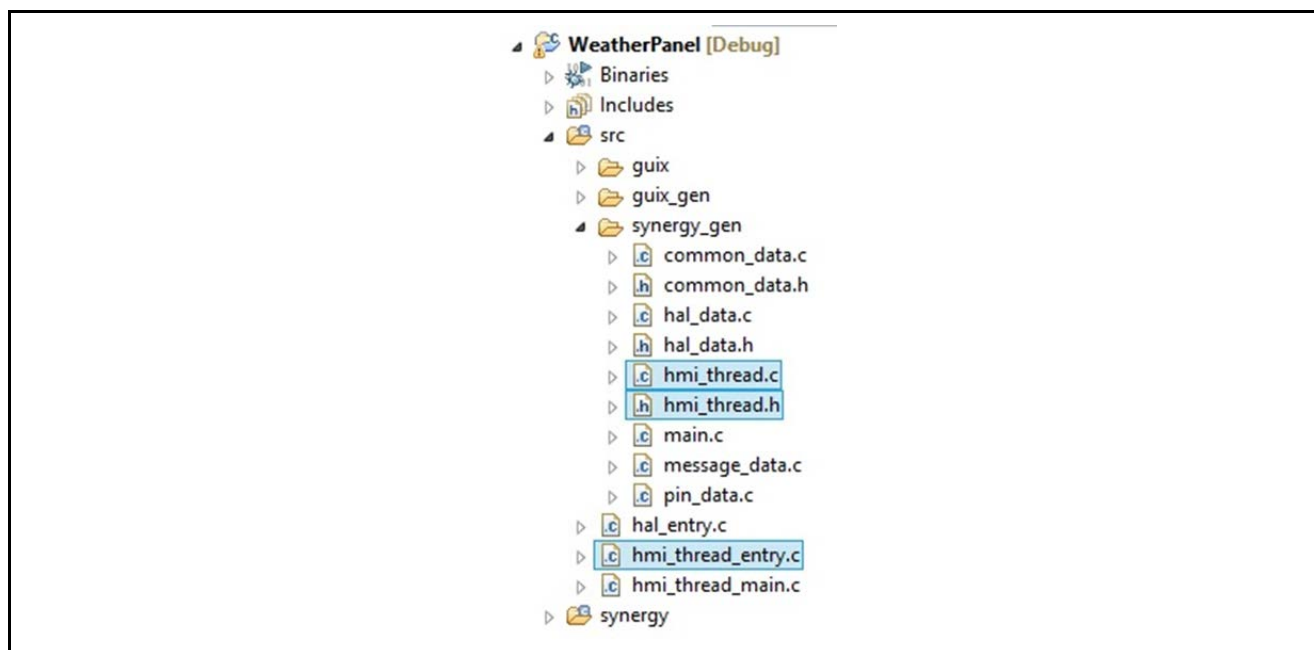


Figure 44. Framework Generated Source File Organization

### 6.1.1 GUIX Initialization

This section details the GUIX initialization. The GUIX system is not automatically initialized by the framework. Several calls are required to initialize GUIX and create the initial canvas where the drawing takes place. You

will find this initialization code at the top of the `hmi_thread_entry()` function located in the `hmi_thread_entry.c` file.

```
status = gx_system_initialize();
if(TX_SUCCESS != status)
{
    while(1);
}
/* Initializes GUIX drivers. */
err = g_sf_el_gx0.p_api->open (g_sf_el_gx0.p_ctrl, g_sf_el_gx0.p_cfg);
if(SSP_SUCCESS != err)
{
    while(1);
}
gx_studio_display_configure ( MAIN_DISPLAY,
                             g_sf_el_gx0.p_api->setup,
                             LANGUAGE_ENGLISH,
                             MAIN_DISPLAY_THEME_1,
                             &p_window_root );
err = g_sf_el_gx0.p_api->canvasInit(g_sf_el_gx0.p_ctrl, p_window_root);
if(SSP_SUCCESS != err)
{
    while(1);
}
```

### 6.1.2 Events and GUIX Message

Touching the screen in the Weather Panel application causes GUIX to invoke the specific callback function that we defined for that screen in GUIX Studio. GUIX provides the callback function with specific information about the window that caused the event, and the actual event that occurred. There are currently 46 different event types recognized by GUIX. They are defined in the `gx_api.h` file and reproduced here for convenience.

```
/* Define the pre-defined Widget event types. */
```

```
#define GX_EVENT_TERMINATE          1
#define GX_EVENT_REDRAW            2
#define GX_EVENT_SHOW              3
#define GX_EVENT_HIDE              4
#define GX_EVENT_RESIZE            5
#define GX_EVENT_SLIDE              6
#define GX_EVENT_FOCUS_GAINED      7
#define GX_EVENT_FOCUS_LOST        8
#define GX_EVENT_HORIZONTAL_SCROLL  9
#define GX_EVENT_VERTICAL_SCROLL   10
#define GX_EVENT_TIMER             11
#define GX_EVENT_PEN_DOWN          12
#define GX_EVENT_PEN_UP            13
#define GX_EVENT_PEN_DRAG          14
#define GX_EVENT_KEY_DOWN          15
#define GX_EVENT_KEY_UP            16
#define GX_EVENT_CLOSE             17
#define GX_EVENT_DESTROY           18
#define GX_EVENT_SLIDER_VALUE      19
#define GX_EVENT_TOGGLE_ON         20
#define GX_EVENT_TOGGLE_OFF        21
#define GX_EVENT_RADIO_SELECT      22
#define GX_EVENT_RADIO_DESELECT    23
#define GX_EVENT_CLICKED           24
#define GX_EVENT_LIST_SELECT       25
#define GX_EVENT_VERTICAL_FLICK    26
#define GX_EVENT_HORIZONTAL_FLICK  28
#define GX_EVENT_MOVE              29
#define GX_EVENT_PARENT_SIZED      30
#define GX_EVENT_CLOSE_POPUP       31
#define GX_EVENT_ZOOM_IN           32
#define GX_EVENT_ZOOM_OUT          33
#define GX_EVENT_LANGUAGE_CHANGE   34
#define GX_EVENT_RESOURCE_CHANGE   35
#define GX_EVENT_ANIMATION_COMPLETE 36
#define GX_EVENT_SPRITE_COMPLETE   37
#define GX_EVENT_TEXT_EDITED       40
#define GX_EVENT_TX_TIMER           41
#define GX_EVENT_FOCUS_NEXT        42
#define GX_EVENT_FOCUS_PREVIOUS    43
#define GX_EVENT_FOCUS_GAIN_NOTIFY 44
#define GX_EVENT_SELECT            45
#define GX_EVENT_DESELECT          46
```

The Weather Panel application uses just a few of these events such as `GX_EVENT_CLICKED`. GUIX passes these events as a `GX_EVENT` structure. The first element of the structure is the event type. A `GX_EVENT` allows data to be sent as part of the message. The final field, `gx_event_payload`, is a union of various data types. The Weather Panel application uses this payload to send a pointer to the current state data structure.



```
/* Define Event type. Note: the size of this structure must be less than or
equal to the constant
GX_EVENT_SIZE defined previously. */
```

```
typedef struct GX_EVENT_STRUCT
{
    ULONG    gx_event_type;          /* Global event type
*/
    USHORT   gx_event_sender;        /* ID of the event sender
*/
    USHORT   gx_event_target;        /* ID of event destination
*/
    ULONG    gx_event_display_handle;
    union
    {
        UINT    gx_event_timer_id;
        GX_POINT gx_event_pointdata;
        GX_UBYTE gx_event_uchardata[4];
        USHORT   gx_event_ushortdata[2];
        ULONG    gx_event_ulongdata;
        GX_BYTE   gx_event_chardata[4];
        SHORT     gx_event_shortdata[2];
        INT       gx_event_intdata[2];
        LONG      gx_event_longdata;
    } gx_event_payload;
} GX_EVENT;
```

## 6.2 LCD control

The PE-HMI1 display has a couple of digital controls that must be driven from the Weather Panel application. As is the case with most embedded applications, the first thing you must do is to identify the hardware dependencies and setup the appropriate drivers.

The two signals of interest in this section are the LCD\_ON and LCD\_BLEN (Blanking Enable). Figure 45 shows an excerpt from the PE-HMI1 v2.0 schematic, which shows the J5 connector. This is the connector that the LCD Screen plugs into. You will notice that the two signals list the associated MCU pins, PA 5 and PA 3. The LCD\_ON signal requires a simple Hi/Lo state that turns the LCD on and off, and the LCD\_BLEN signal requires a PWM signal, which modulates the display intensity.

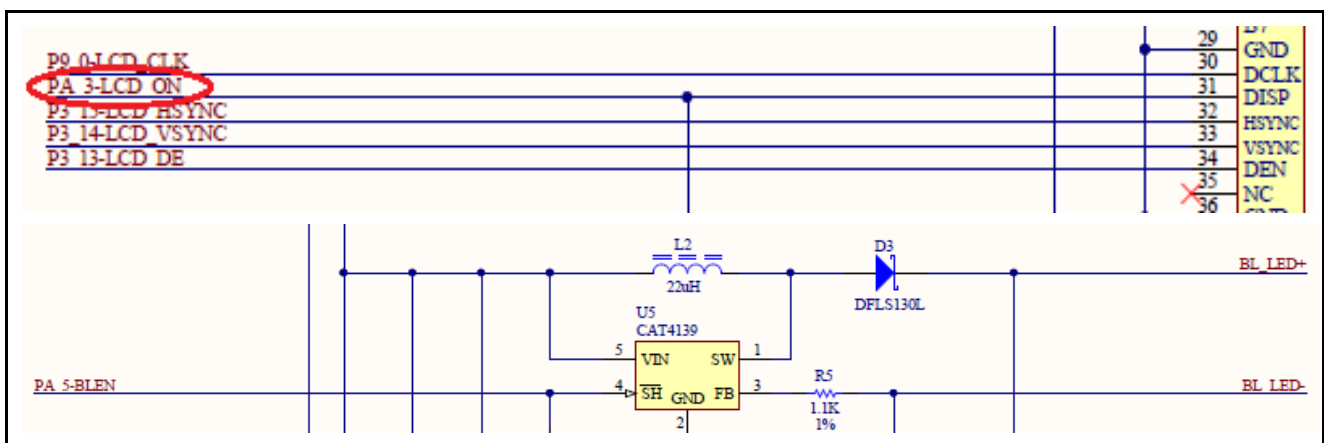
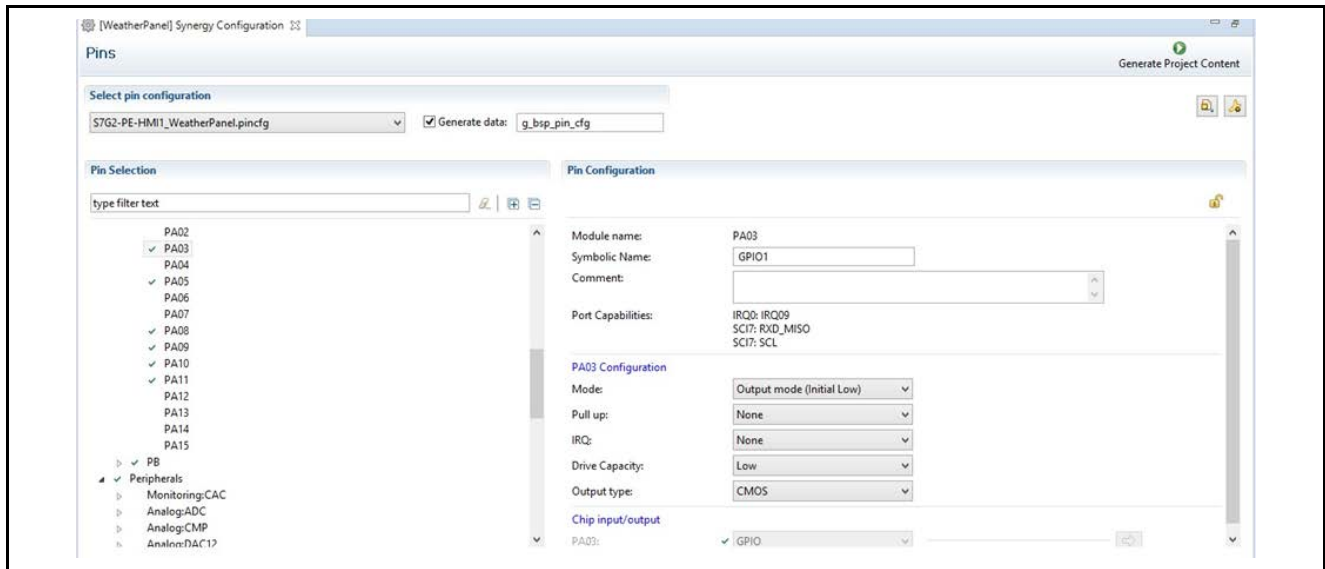


Figure 45. LCD On and Backlight Pin Details for PE-HMI1

Figure 46 shows the **Pins** tab for the PE-HMI1 Weather Panel application. The first thing you do when configuring a pin is to select the port from the **Pin Selection** dialog box on the left-hand side of the screen, in this case, PA (port A). The port selection expands to show the pins associated with the I/O port.



**Figure 46. LCD ON Pin Configuration for PE-HMI1**

In this case, we have the simplest configuration that you can have for a GPIO pin. The mode is set to Output mode, and the Chip input/output is set to GPIO. Notice that the module name is PA03, which is the naming convention that you will see in Synergy Pins configuration. Declaring an I/O pin in this manner causes the Synergy framework to create an instance of the pin in the `hal_data.c` file, that is an auto-generated file.

```
const ioport_instance_t g_ioport =
{ .p_api = &g_ioport_on_ioport, .p_cfg = NULL };
```

This provides driver-level access to the pin but it is up to the user to write the code that sets the state of the pin. In this case, all that is required for the Weather Panel application is to set the pin high to turn the LCD display on. This is accomplished during the initialization code in the `hmi_thread_entry.c` file.

```
/* Controls the GPIO pin for LCD ON. */
err = g_ioport.p_api->pinWrite(IOPORT_PORT_10_PIN_03, IOPORT_LEVEL_HIGH);
if (err)
{
    while(1);
}
```

For this demonstration application, the error handling simply loops with a `while (1)` condition if an error is returned from the `g_ioport.p_api->pinWrite()` call. This causes the HMI Thread to stop responding should an error be returned from the `pinWrite` call.

## 7. Importing and Building the Project

To bring the Weather Panel application into the e<sup>2</sup> studio ISDE, follow these steps:

Refer to the *Renesas Synergy™ Project Import Guide* (r11an0023eu0121-synergy-ssp-import-guide.pdf) included in this package for information on how to import a Synergy project.

1. Launch e<sup>2</sup> studio ISDE.
2. In the workspace launcher, browse to the workspace location of your choice.
3. Close the **Welcome** window.
4. In the ISDE go to **File > Import**.
5. In the Import Dialog Box pick **Existing Projects into Workspace**.
6. Select the Root directory of your workspace (where you placed the project).
7. Select the project you wish to import and click **Finish**.

8. Generate output files as mentioned in the GUIX Studio Overview.
  - For e<sup>2</sup> studio ISDE:  
Run **weather\_GNU.gxp** from <project workspace>\guiX\_studio folder before building the project. Refer to Figure 10 and Figure 12.
  - For IAR Embedded Workbench:  
Run **weather\_IAR.gxp** from <project workspace>\guiX\_studio folder before building the project. Refer to Figure 11 and Figure 12.
9. Click on **Generate Project Content** on the Synergy configurator window.
10. Now build the project.

## 8. Downloading the Executable to the Target Board

To connect and run the code, follow these steps:

1. Refer to the *Quick Start Guide for PE-HMI1* to setup the J-Link debugger connection from your PC to the JTAG connector on the target board.
2. Go to **Run > Debug configurations**.
3. Click **Debug**. The program will break at the reset handler.
4. Click **Yes** to switch to the **Debug perspective** when prompted by the ISDE.
5. Click **Resume** twice.

## 9. Known Issues

Each `GX_EVENT_CLICKED` sends `GX_EVENT_KEY_DOWN` event to the parent (and the parent then routes it to the selected single line text input widget). With GUIX 5.4.0.0, if a user presses a button with auto-repeat, then moves their finger outside of button boundaries (while still holding their finger pressed on the screen) and then releases it, the button will stay locked in the auto-repeat state (that is, generating `GX_EVENT_CLICKED` periodically) despite the screen being released. The only way to restore proper behavior is to press and release the button again.

## 10. References

1. *PE-HMI1 v2.0 User's Manual: Hardware*
2. *PE-HMI1-v2.0 Schematics*
3. *Renesas Synergy Software Package Datasheet*
4. *Synergy X-ware-Documents (GUIX, ThreadX)*
5. *Synergy™ Software Package SSP User's Manual v1.5.0 or later*
6. *Renesas Synergy™ Project Import Guide (r11an0023eu0120-synergy-ssp.pdf)*

## Website and Support

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

Synergy Software	<a href="http://www.renesas.com/synergy/software">www.renesas.com/synergy/software</a>
Synergy Software Package	<a href="http://www.renesas.com/synergy/ssp">www.renesas.com/synergy/ssp</a>
Software add-ons	<a href="http://www.renesas.com/synergy/addons">www.renesas.com/synergy/addons</a>
Software glossary	<a href="http://www.renesas.com/synergy/softwareglossary">www.renesas.com/synergy/softwareglossary</a>
Development tools	<a href="http://www.renesas.com/synergy/tools">www.renesas.com/synergy/tools</a>
Synergy Hardware	<a href="http://www.renesas.com/synergy/hardware">www.renesas.com/synergy/hardware</a>
Microcontrollers	<a href="http://www.renesas.com/synergy/mcus">www.renesas.com/synergy/mcus</a>
MCU glossary	<a href="http://www.renesas.com/synergy/mcuglossary">www.renesas.com/synergy/mcuglossary</a>
Parametric search	<a href="http://www.renesas.com/synergy/parametric">www.renesas.com/synergy/parametric</a>
Kits	<a href="http://www.renesas.com/synergy/kits">www.renesas.com/synergy/kits</a>
Synergy Solutions Gallery	<a href="http://www.renesas.com/synergy/solutionsgallery">www.renesas.com/synergy/solutionsgallery</a>
Partner projects	<a href="http://www.renesas.com/synergy/partnerprojects">www.renesas.com/synergy/partnerprojects</a>
Application projects	<a href="http://www.renesas.com/synergy/applicationprojects">www.renesas.com/synergy/applicationprojects</a>
Self-service support resources:	
Documentation	<a href="http://www.renesas.com/synergy/docs">www.renesas.com/synergy/docs</a>
Knowledgebase	<a href="http://www.renesas.com/synergy/knowledgebase">www.renesas.com/synergy/knowledgebase</a>
Forums	<a href="http://www.renesas.com/synergy/forum">www.renesas.com/synergy/forum</a>
Training	<a href="http://www.renesas.com/synergy/training">www.renesas.com/synergy/training</a>
Videos	<a href="http://www.renesas.com/synergy/videos">www.renesas.com/synergy/videos</a>
Chat and web ticket	<a href="http://www.renesas.com/synergy/resourcelibrary">www.renesas.com/synergy/resourcelibrary</a>

## Revision History

Rev.	Date	Description	
		Page	Summary
1.0	Oct.9.15	-	Initial version
1.10	Dec.4.15	-	In section “Create and Build the Project,” deleted the step to change the optimization level. This step is obsolete for SSP version 1.0.0-beta.3 and higher.
1.11	Jan.11.16	-	Update to remove template references
2.00	Aug.23.16	-	Update with reference to PE-HMI and ISDE 5.0.0.43
2.01	Nov.18.16	-	Minor formatting changes
2.02	Jan.10.17	-	Updated for SSP v1.2.0.b.1
2.03	Mar.21.17	-	Updated for SSP v1.2.0. Added support for PK-S5D9.
2.04	Aug.17.17	-	Updated for SSP v1.4.0
2.05	Sep.7.17	-	Final release edit
2.06	Oct.27.17	37	Known Issue added
2.07	Mar.5.18	-	Updated for SSP v1.4.0
2.08	Jun.18.18	-	Sample codes updated
2.09	Sep.21.18	-	Updated to SSP 1.5.0
2.10	Mar.21.19	-	Updated to SSP 1.6.0



## Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev. 4.0-1 November 2017)

## Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,  
Koto-ku, Tokyo 135-0061, Japan  
[www.renesas.com](http://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/](http://www.renesas.com/contact/).