RA2L1/RA2E1

CPK-RA2L1/RA2E1 评估板入门

要点

CPK-RA2L1/CPK-RA2E1 是用于 RA2L1/RA2E1 单片机的评估板套件。该套件可通过灵活配置软件包 (FSP) 和 e² studio IDE，对 RA2L1/RA2E1 MCU 群组的特性进行无缝评估，并对嵌入系统应用程序进行开发。本文档可与《瑞萨 RA MCU 基础知识》配套使用，旨在将该指南中有关硬件操作的部分在 CPK-RA2L1/CPK-RA2E1 评估板上进行实现。

在使用本文档之前，推荐您先学习《瑞萨 RA MCU 基础知识》，这样有助于您在本文所述的硬件实操中更快上手。

开发环境：

e² studio：2021-10 版

https://www.renesas.com/jp/zh/software-tool/e-studio?

FSP：v3.5.0

https://www.renesas.com/jp/zh/software-tool/flexible-software-package-fsp?

QE for Capacitive Touch V3.0.2
https://www.renesas.com/jp/zh/software-tool/qe-capacitive-touch-development-assistance-tool-capacitive-touch-sensors?

CPK-RA2L1
https://www.renesas.com/cpk-ra2l1

CPK-RA2E1
https://www.renesas.com/cpk-ra2e1

目录

1. 首次使用瑞萨 CPK-RA2L1/CPK-RA2E1 评估板

本章介绍首次使用瑞萨 CPK-RA2L1/CPK-RA2E1 评估板时要进行的设置。

1.1 导入 BSP（板级支持包）文件

本节介绍如何在 e² studio 上导入 BSP 文件。

Setp 1. 文件准备

准备好 BSP 文件，请从网页上下载“CPK 评估板的 BSP - FSP 3.5.0 适用”

Setp 2. 导入 BSP

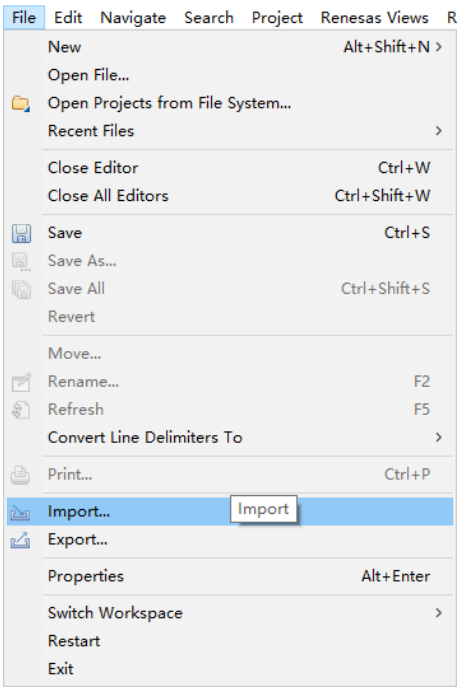
关于在 e2studio 中如何导入 BSP 文件，请参考网页上的相关文档“向 FSP 中添加 CPK 评估板的 BSP”。

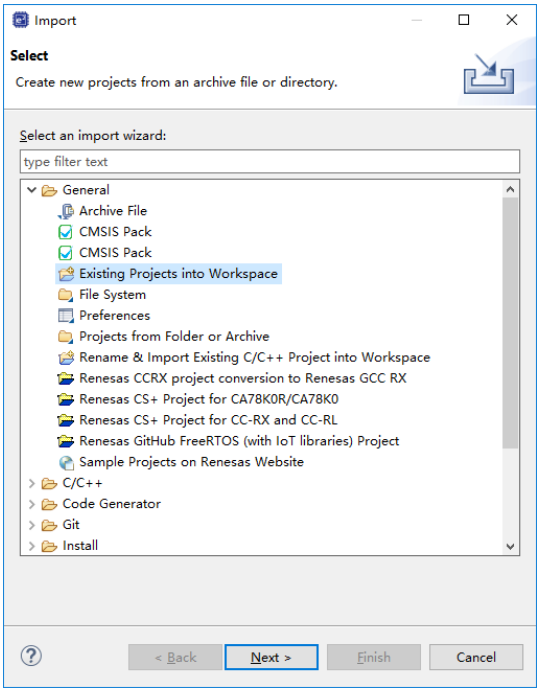
2. 下载并测试示例

本章内容基于《瑞萨 RA MCU 基础知识》中的章节 **7.2 下载并测试示例** 所作。

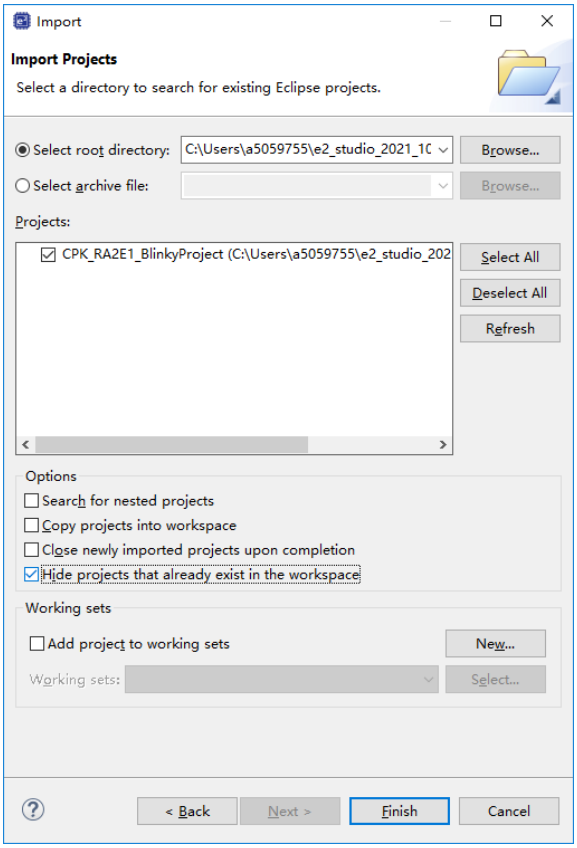
首先，从操作系统的“Start”（开始）菜单打开 e² studio。请先从网站下载项目，然后将其导入工作区。导入工作区的步骤如下：

点击“File”→“Import”





选择下载工程所在的文件夹



在我们将程序下载到评估板并运行之前，需要创建一个调试配置。单击“*Debug*”（调试）符号 旁边的小箭头，然后从下拉列表框中选择“*Debug Configurations*”（调试配置）。

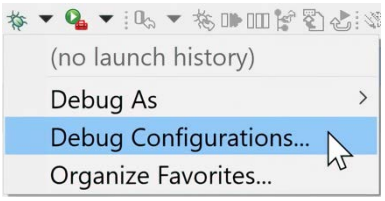


图 2-1：要开始调试，请从下拉列表框中选择“Debug Configurations”（调试配置）

在出现的窗口中，突出显示左侧树形视图中“*Renesas GDB Hardware Debugging*”（Renesas GDB 硬件调试）下的 *CPK_RA2E1_BlinkyProject Debug_Flat*。如果您为此项目使用了其他名称，请选择您使用的名称。

选择项目后，将为“Debug Configurations”（调试配置）打开一个新屏幕，其中显示相关的所有选项（请参见图 *2-2*）。由于仅用于测试目的，因此无需在此处进行任何更改，只需单击底部的“*Debug*”（调试），调试器随即启动。显示“*Confirm Perspective Switch*”（确认透视图切换）对话框后，选择“*YES*”（是）。

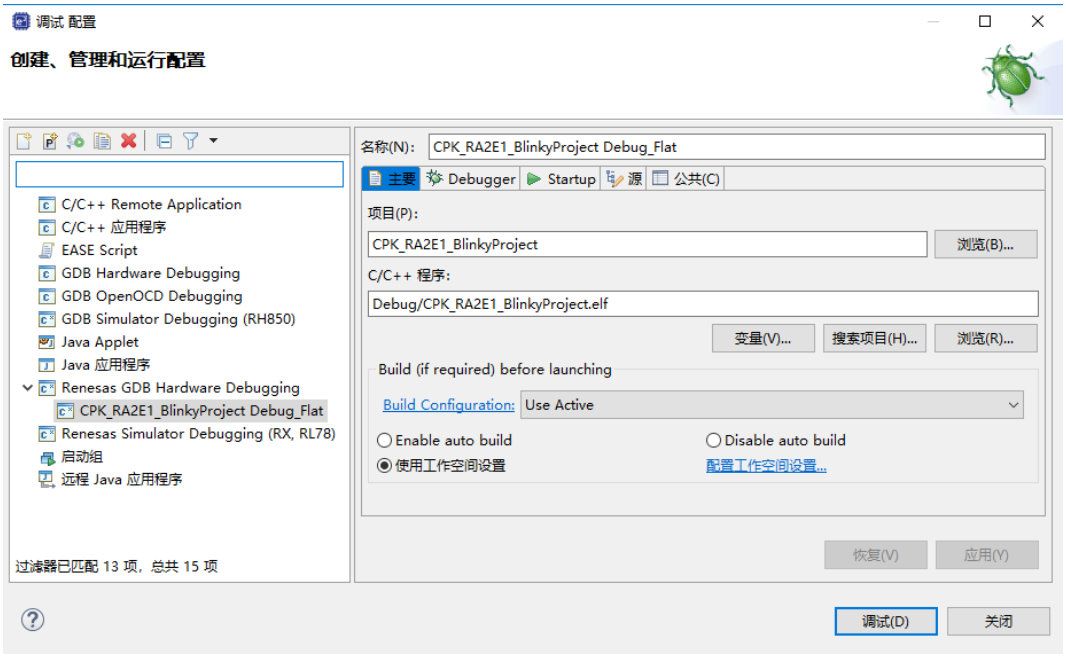


图 2-2：在“Renesas GDB Hardware Debugging”（Renesas GDB 硬件调试）下选择项目后，无需在出现的窗口中进行任何更改。

可能会出现另一个名为“*J-Link® Firmware update*”（J-Link® 固件更新）的对话框，要求为板上调试器安装新的固件版本。强烈建议单击“*Yes*”（是）来允许更新。

根据 Windows 工作站的安全设置，可能会出现一个显示安全警报的对话框窗口，通知您“Windows Defender Firewall has blocked some features of E2 Server GDB on all public and private networks.”（Windows Defender 防火墙已阻止所有公用和专用网络上的 E2 Server GDB 的某些功能。）要继续操作，请

允许 E2 Server GDB 在专用网络上通信。为此，请选中相应的复选框，然后单击“*Allow access*”（允许访问）。

打开“Debug”（调试）透视图后（请参见图 2-3），调试器会将程序指针设置为程序的入口点，即复位处理程序。单击“*Resume*”按钮，程序将运行到 `main()` 函数内 `hal_entry()` 调用行中的下一个停止处。再次单击“*Resume*”，程序将继续执行，评估板上红色 LED3（用户 LED）开始闪烁。

最后一步是单击“*Disconnect*”（断开连接）按钮，断开调试器与开发板的连接，以停止程序的执行。

现在，您已确定 e² studio 的安装可与评估板配合使用，接下来为 RA 系列单片机编写第一个程序。这将是下一章的主题。

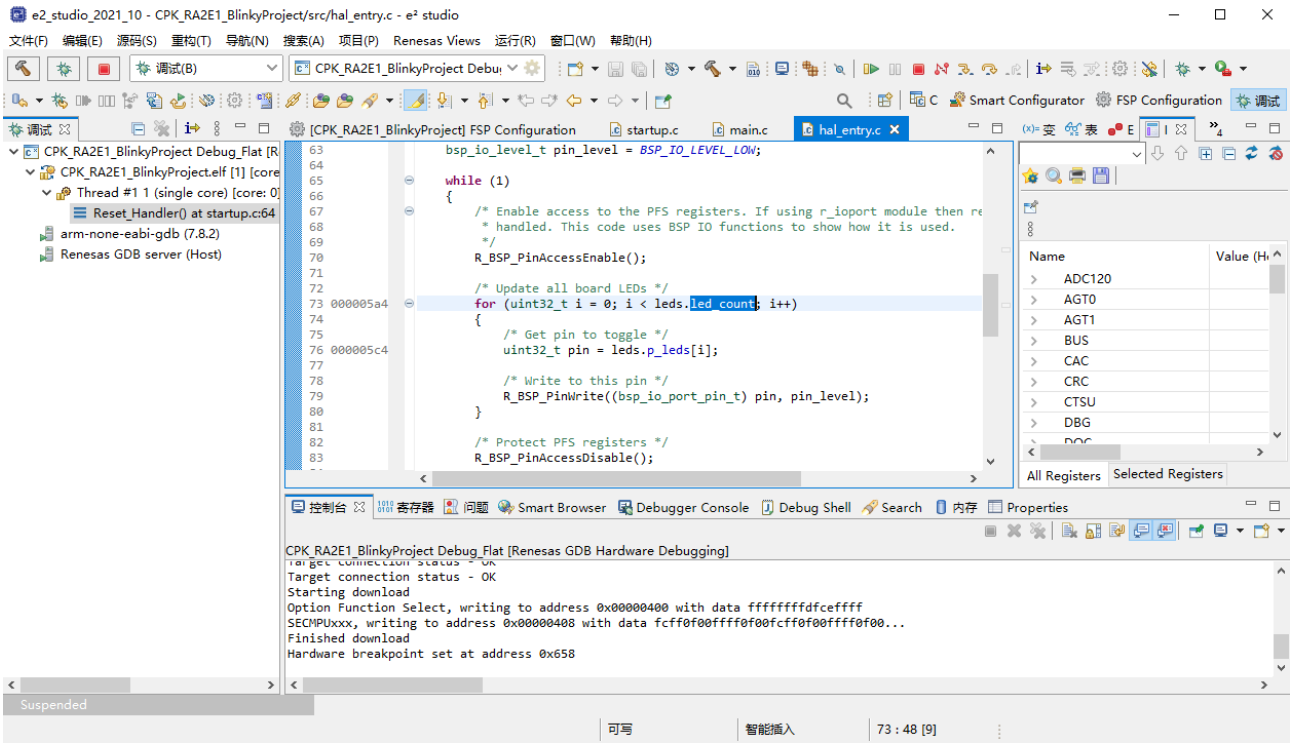


图 2-3：e² studio 的“Debug”（调试）透视图

本章要点：

- 为了将程序下载到任何硬件并进行调试，需要先创建一个调试配置。
- 加载后，调试器会将程序计数器设置为入口点。下一个停止处将位于 main() 中。

3. Hello World! – Hi Blinky!

本章内容基于《瑞萨 RA MCU 基础知识》中的章节 **8 Hello World! – Hi Blinky!** 所作。

您将在本章中学到以下内容：

- 如何从头开始为 CPK-RA2L1/RA2E1 评估板创建项目。
- 如何在 FSP 配置器中更改灵活配置软件包的设置。
- 如何编写代码以切换 CPK 上的用户 LED。
- 如何下载和测试程序。

注：本章内容以 CPK-RA2E1 举例，若使用 CPK-RA2L1，步骤和方法一致。

大多数编程语言新手曾编写的第一个程序（现在仍是）就是将字符串“Hello World”输出到标准输出设备的程序。

这些年来，LED 成为了一种商品，我们在电路板上放置了 LED，将它们的闪烁作为新的“Hello World”。

这也是本章的目标：切换 RA2L1/RA2E1 系列器件的评估板 (CPK) 上的 LED。您将使用配置器创建一个新项目，基于灵活配置软件包 (FSP) 的 API 编写代码，最后下载、调试并运行代码。

CPK-RA2L1/RA2E1 可以轻松连接外部硬件，因为大多数引脚均可通过 MCU 引脚访问区域中的公头引脚插针或电路板的系统控制和生态系统访问区域中的生态系统连接器进行访问。由于 RA2L1/RA2E1 系列 MCU 是 RA 产品家族 MCU 的 RA6 系列的超集器件，因此可以评估该系列的大多数功能，并随后将结果应用于该系列的较小同级产品。图 3-1 所示为电路板的框图，其中突出显示了主要元件。

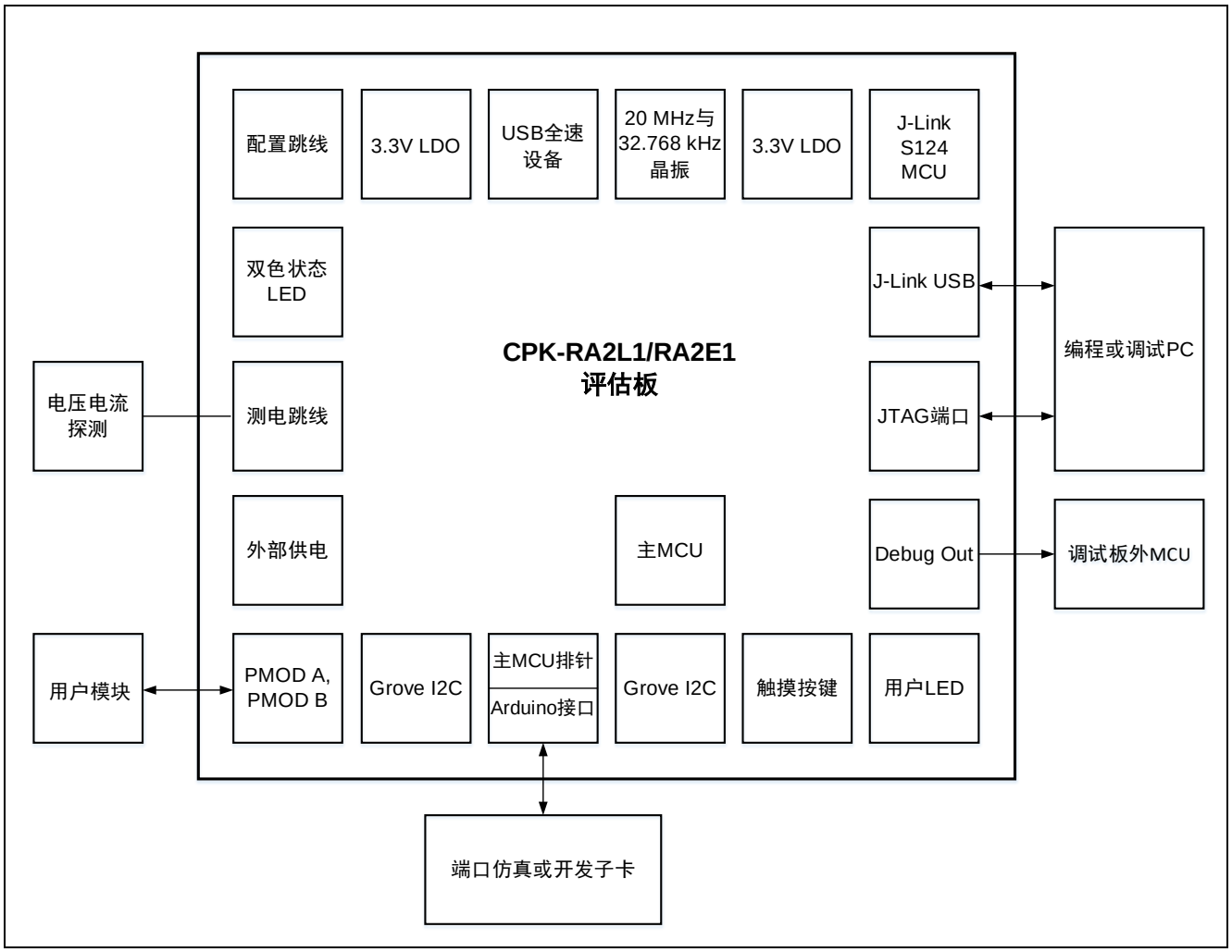


图 3-1：CPK-RA2L1/RA2E1 评估板的框图

3.1 使用项目配置器创建项目

如果尚未启动 e² studio，请从 Windows® 工作站的“Start”（开始）菜单中打开 e² studio。开发环境启动并运行后，请关闭“*Welcome*”（欢迎）屏幕（如果它在显示），因为它会挡住其他窗口。

由于在 e² studio 中为单片机编写新程序始终需要创建一个项目，因此这是您需要执行的第一步。为此，请转到“*File → New → Renesas C/C++ Project*”（文件 → 新建 → *Renesas* C/C++ 项目），或者在“Project Explorer”（项目资源管理器）视图中单击鼠标右键，然后选择“*New → Renesas C/C++ Project*”（新建 → *Renesas* C/C++ 项目）。两种方式都将打开一个对话框，询问要使用的模板。在左侧栏中选择 *Renesas RA*，再从主窗口中选择“*Renesas RA C/C++ Project*”（瑞萨 RA C/C++ 项目）。然后单击“*下一步*”。

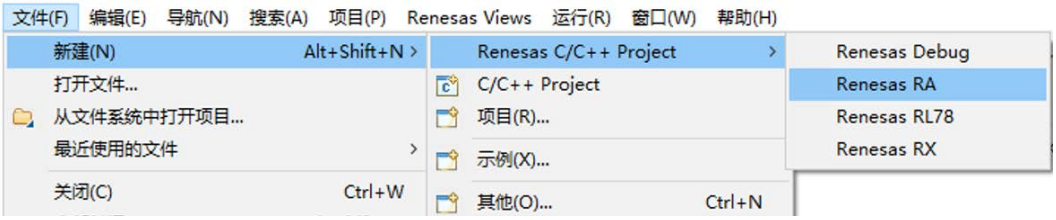


图 3-2：第一步是调用项目配置器

出现“*Project Configurator*”（项目配置器）后，为项目命名，接受项目的默认位置（将作为 e² studio 工作区），或将其更改为您偏好的文件夹。单击“*Next*”（下一步），转到“*Device and Tools Selection*”（器件和工具选择）屏幕。

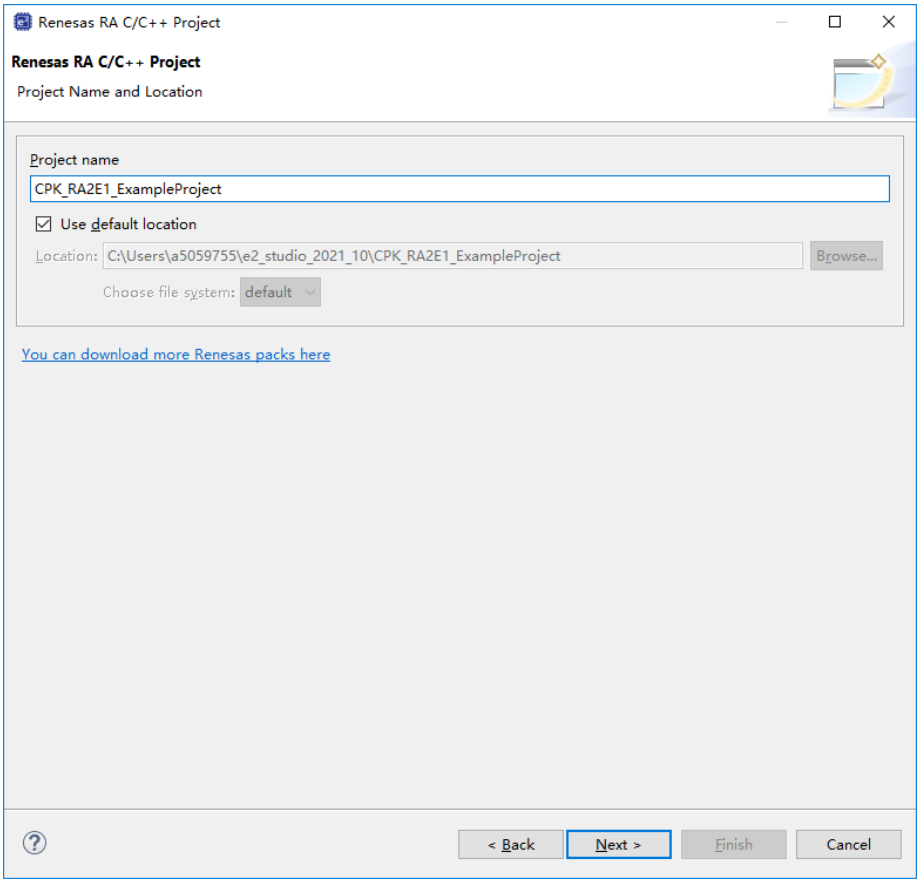


图 3-3：项目配置器的第一个屏幕主要询问项目的名称和位置

在“*Device Selection*”（器件选择）下，查找名为“*FSP Version*”（FSP 版本）的字段：它应显示与之前下载的灵活配置软件包相同的版本。从“*Board*”（电路板）下的下拉列表中选择 *CPK-RA2E1 MCU 评估板 （LQFP64*），因为这是我们用于小型“CPK_RA2E1_ExampleProject”程序的硬件。该列表通常还会包含 RA 产品家族的评估板以及“*Custom User Board*”（定制用户板）条目，并通过为所选 FSP 版本安装的 Renesas CMSIS 包文件创建。选择 *CPK-RA2E1 MCU 评估板 （LQFP64*）之后，“*R7FA2E1A92DFM*”会在“Device”（器件）旁边的文本框中自动显示。如果未显示，请浏览下拉列表，直到发现目标器件的型号为止。在“*Toolchains*”（工具链）框中，验证是否列出了 *GNU ARM® Embedded, 8.3.1.20190703* 或更高版本，以及“*Debugger*”（调试器）框中是否已选择 *J-Link® Arm*。这些字段应该会被预先填入。如果未预先填入，请修改相应项以匹配上面给出的值。

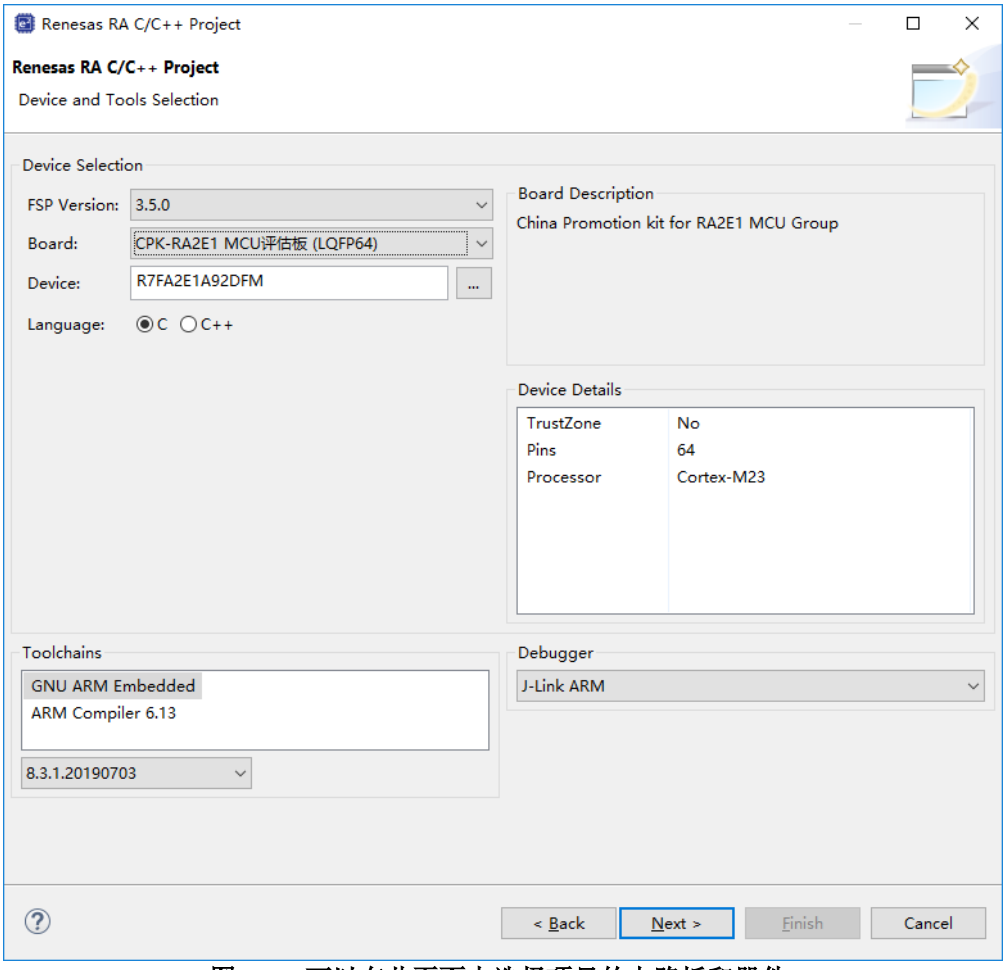


图 3-4：可以在此页面中选择项目的电路板和器件

如果一切正常，请单击“*Next*”（下一步），打开“*Build Artifact and RTOS Selection*”（构建工件和 RTOS 选择）屏幕，可以在其中设置构建的类型。只有在上一个窗口中选择了非 TrustZone 器件或为扁平化项目选择了 TrustZone 器件时，才会显示此屏幕。可用的选项包括用于创建独立 ELF（可执行和可链接格式）可执行文件的“*Executable*”（可执行文件）、用于创建目标代码库的“*Static Library*”（静态库）以及用于创建配置为与静态库一起使用的应用程序项目的“*Executable using an RA Static Library*”（使用 RA 静态库的可执行文件）。在页面右侧的下拉列表中，为项目选择是否使用实时操作系统 (RTOS)。

对于小型动手实验，请选择“*Executable*”（可执行文件）和“*No RTOS*”（无 RTOS），然后单击“下一步”。

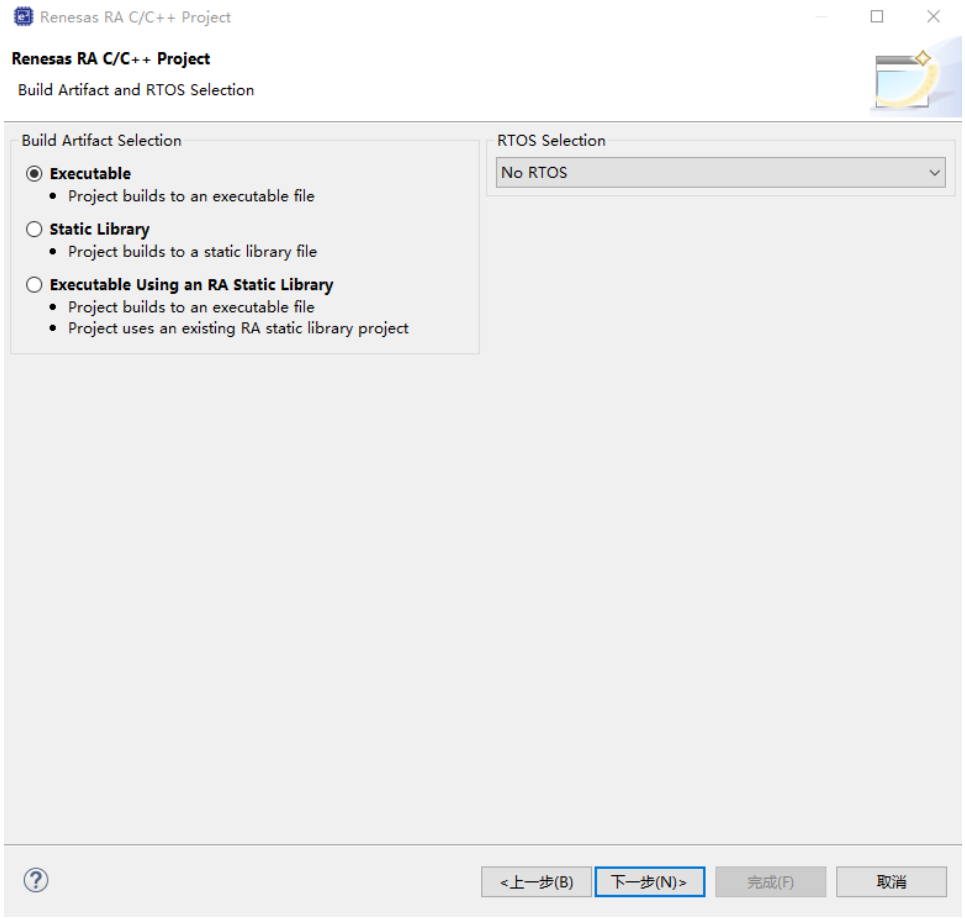


图 3-6：选择不带 RTOS 的可执行项目

这将打开“*Project Template Selection*”（项目模板选择）页面，可以在其中选择初始项目内容的模板。项目模板可能包含多个条目；至少包括适合所选电路板/器件组合的板级支持包。有些模板甚至包括一个完整的示例项目，但“Project Configurator”（项目配置器）将仅显示与您在先前屏幕上所做选择匹配的模板。在本例中，选择“*Bare Metal – Minimal*”（裸机 – 最小化）条目，以加载评估板的板级支持包。单击“*完成*”。完成项目的配置。

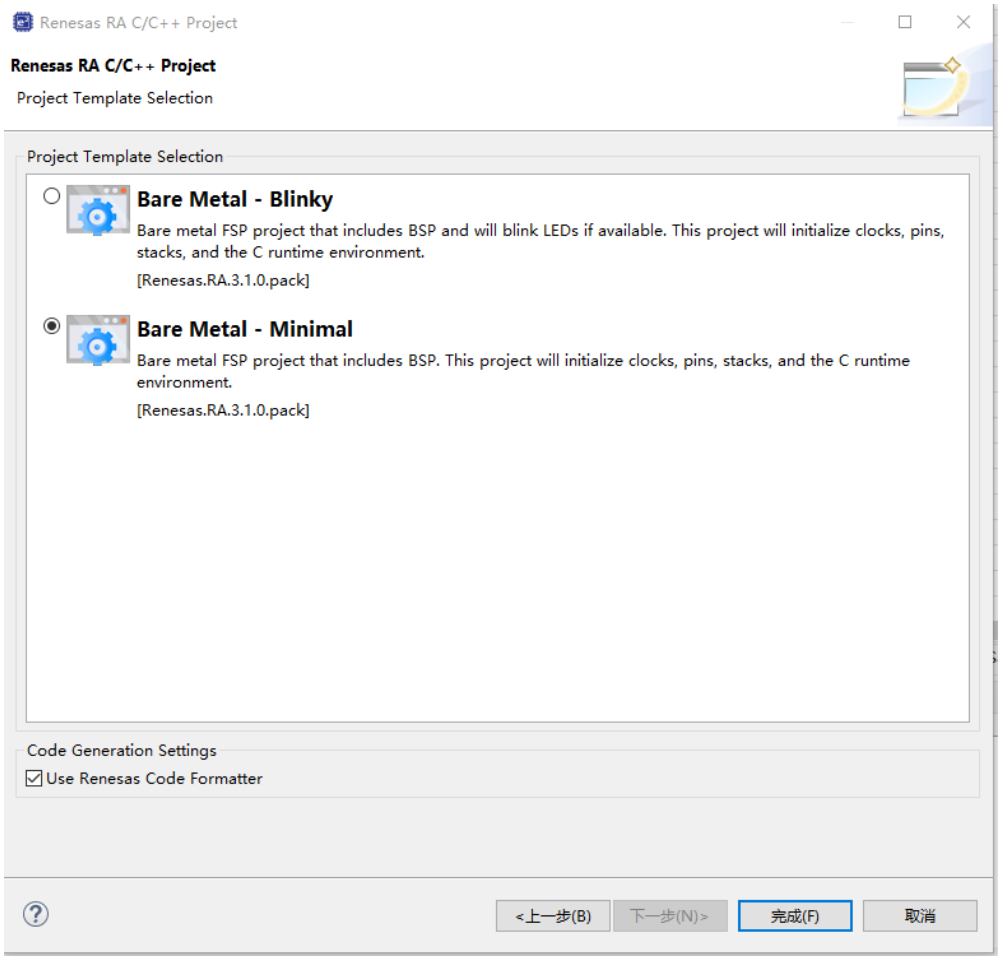


图 3-6：“Project Template Selection Page”（项目模板选择页面）将显示适合项目最初内容的模板

“*Project Configurator*”（项目配置器）将关闭并在最后一步中创建项目所需的所有文件。完成此后处理后，将出现一个对话框，询问您是否要打开“*FSP Configuration*”（FSP 配置）透视图。选择“*Open Perspective*”（打开透视图）。

3.2 使用 FSP 配置器设置运行环境

FSP 配置器启动后，将为您提供项目的只读摘要和所选软件组件的简短概述。此外，它还提供了快捷方式，可方便地访问 YouTube™ 上的瑞萨 RA 频道、Renesas.com 上的瑞萨设计与支持页面（可在其中访问文档、知识库和 Renesas Rulz 论坛）以及硬盘上的 FSP 用户手册。

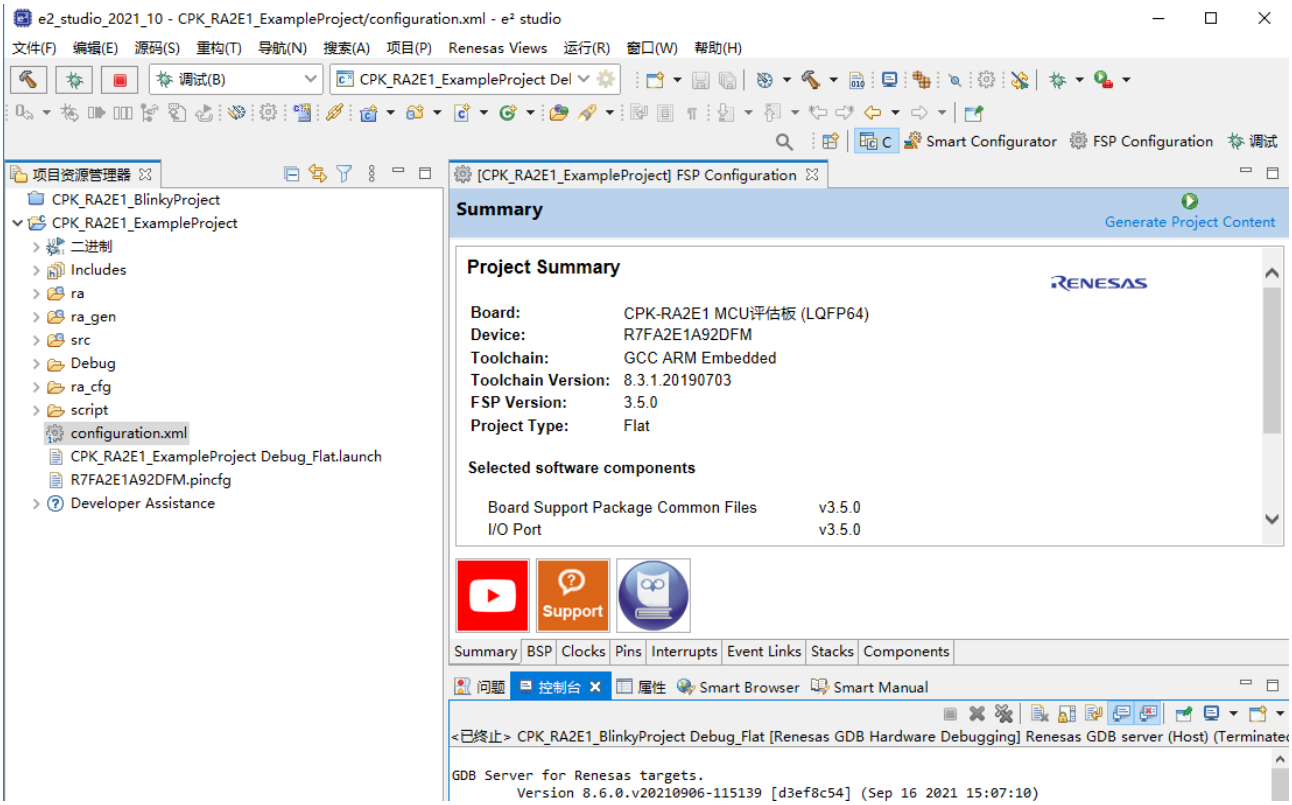


图 3-7：e² studio 内部的“FSP Configuration”（FSP 配置）透视图

在以下名为 *BSP* 的选项卡中，可以查看和编辑设置的多个方面，例如电路板和器件选择。在此选项卡的属性视图中，可以为板级支持包进行其他设置，例如，主堆栈的大小或 MCU 的某些安全功能。在之后的“*Clocks*”（时钟）选项卡中，可以为您的项目分配初始时钟配置。任何潜在的问题都将以红色突出显示，将鼠标悬停在突出显示的位置上将出现有关冲突或设置不完整的说明。

第四个选项卡“*Pins*”（引脚）涵盖了 RA MCU 的引脚分配。可以根据端口或外设列出引脚。如果设置不兼容或缺失，则配置器右侧的“*Package View*”（封装视图）会显示器件的封装，突出显示所配置的引脚并标记错误。“*Problems*”（问题）视图以及“*Pin Conflicts*”（引脚冲突）视图中也会显示这些内容。这样，便可将可能的错误减少到最低限度。

接下来是“*Interrupts*”（中断）选项卡。可以在此处指定用户定义的（即非 FSP）驱动程序如何使用单片机的中断控制器单元 (ICU)，以及将哪个中断服务程序 (ISR) 与 ICE 事件（中断）相关联。此外，还可以在此处查看分配的所有 ICU 事件的完整列表，包括由在配置器的“*Stacks*”（栈）视图中创建的 FSP 模块实例生成的 ICU 事件。

“*Event Links*”（事件链接）选项卡具有类似作用。可以在此处指定驱动程序如何在 RA 项目中使用事件链接控制器 (ELC)，并且可以声明此类驱动程序可能通过一组外设功能产生一组 ELC 事件或使用一组 ELC 事件。

需要花费大部分时间的页面为“*Stacks*”（栈）页面，可以在其中创建 RTOS 线程和内核对象，以及 FSP 软件栈。可以添加不同的对象和模块，并且可以在“*Properties*”（属性）视图中修改其属性。所有这些对象和模块都将自动插入，直到降至需要用户干预的程度为止。在这种情况下，一旦鼠标悬停在模块上，便会将需要注意的模块标记为红色，同时给出必要设置或问题的说明。如果问题解决，模块将恢复为标准颜色。“*Stacks*”（栈）视图本身以图形方式显示各种栈，可让您轻松跟踪不同的模块。在我们的示例中，仅显示了一个具有一个模块的线程：在 r_ioport 上使用 g_ioport I/O 端口驱动程序的 HAL/通用线程。它是由项目配置器自动插入的，允许我们仅用几行代码便可编写让 LED 闪烁的程序。

最后一个选项卡的名称是“*Components*”（组件），其中显示了不同的 FSP 模块并可对模块进行选择。它还列出了可用的 RA CMSIS 软件组件。不过，最好通过“*Stacks*”（栈）页面在当前项目中添加或删除模块，因为还可以在其中进行配置。

对于我们的项目，无需在 FSP 配置器中进行任何更改，因为项目配置器已经为我们进行了所有必要的设置。最后，需要创建基于当前配置的附加源代码。单击 FSP 配置器右上角的“*Generate Project Content*”（生成项目内容）按钮。此操作将从 FSP 中提取所需文件，将其调整为在配置器中进行的设置，然后将其添加到项目中。

3.3 编写前几行代码

获取所有自动生成的文件之后，接下来查看创建的内容。IDE 左侧的“*Project Explorer*”（项目资源管理器）列出了当前包含的所有内容。*ra_gen* 文件夹保存通道号等配置集。*src* 目录包含一个名为 *hal_entry.c* 的文件。这是稍后要编辑的文件。请注意，尽管在 *ra_gen* 文件夹中有一个名为 *main.c* 的文件，但用户代码必须转到 *hal_entry.c* 中。否则，如果您在 FSP 配置器中进行修改并重新创建项目内容，在 main.c 中的更改会丢失，因为每次单击“*Generate Project Content*”（生成项目内容）时，都将覆盖该文件。

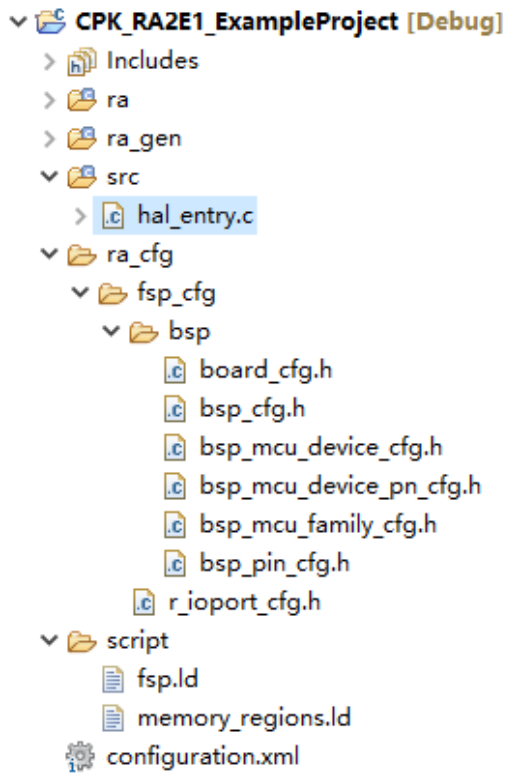


图 3-8：FSP 配置器创建所需文件后的项目树

该项目还包含几个名称中带有“ra”或“fsp”的目录，其中包含 FSP 的源文件、包含文件和配置文件。通常的规则是，不得修改这些文件夹（和子文件夹）的内容。其中包含由配置器生成的文件，在此所做的任何更改都将在下次生成或刷新项目内容时丢失。用户可编辑的源文件是直接位于 *\src* 文件夹或您添加的任何其他文件夹为根目录中的文件。

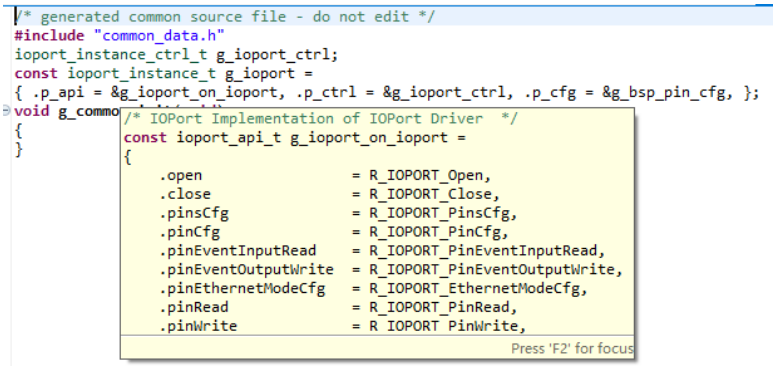
接下来，为 RA 产品家族单片机编写第一个真实源代码，实现 CPK-RA2L1/RA2E1 评估板上用户 LED（红色） 的闪烁。因此您必须通过添加代码来点亮和熄灭 LED 以及实现延时循环。

有两种选择：一种是通过接口函数来使用 API，另一种是使用 BSP 实现函数。

如果查看文件 *ra_gen\common_data.c* 中的代码，则会发现 I/O 端口驱动程序实例 `g_ioport` 具有以下定义：

```
const ioport_instance_t g_ioport = {.p_api = &g_ioport_on_ioport,
                                    .p_ctrl = &g _ioport_ctrl,
                                    .p_cfg = &g_bsp_pin_cfg, };
```

`g_ioport_on_ioport` 是一个结构体，用于声明端口可能执行的操作，将分配给 `g_ioport` 实例的 API 指针。将鼠标悬停在该结构体上，可以轻松查看其中的内容，此结构体显示了其成员之一 (`.pinWrite`) 是指向引脚写入函数的指针。



因此，要点亮 LED，可以写入：

```
g_ioport.p_api->pinwrite (&g_ioport_ctrl, pin, BSP_IO_LEVEL_LOW);
```

但这意味着实际上需要知道 用户 LED 连接到哪些 I/O 端口，以及有多少个用户 LED 可用。为此，我们可以阅读电路板的文档或仔细检查原理图以找到正确的端口。或者，也可以只依靠 FSP。创建类型为 `bsp_leds_t` 的结构体（在 *board_leds.h* 中声明）并为其分配在 *board_leds.c* 中定义的全局 BSP 结构体 `g_bsp_leds` 即可解决问题。这两个文件均位于项目的 *ra\board\ra2e1_cpk* 文件夹内。因此，以下两行代码足以获取有关评估板上 LED 的信息：

```
extern bsp_leds_t g_bsp_leds;
bsp_leds_t Leds = g_bsp_leds;
```

现在，可以使用 LED 结构体来访问电路板上的所有 LED，并使用以下语句点亮红色 LED（将端口设置为低电平将点亮 LED，将端口设置为高电平则将熄灭 LED）：

```
g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                         Leds.p_leds[BSP_LED_LED1],
                         BSP_IO_LEVEL_LOW);
```

此语句后需要有第二条语句，用于将其引脚设置为高电平以熄灭 用户 LED。
最后，需要提供一段延时以使 LED 以用户友好的方式切换。为此，可以再次调用 BSP API：

```
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
```

`R_BSP_SoftwareDelay` 函数的第一个参数是要延迟的单位数，而第二个参数是指定的基本单位，在本例中为秒。其他选项包括毫秒和微秒。

最后，由于我们想无限期地运行程序，因此必须围绕代码创建一个 `while(1)` 循环。

目前，还需要执行的操作是将以下代码行直接输入到 *hal_entry.c* 文件中的函数签名之后，替换 `/* TODO: add your own code here */` 行。对于由项目配置器和 FSP 配置器插入的其他代码，请保持不变。单片机需要借助这些代码来正常运行。

```
extern bsp_leds_t g_bsp_leds;
bsp_leds_t Leds = g_bsp_leds;

while (1)
{
    g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                             Leds.p_leds[BSP_LED_LED1],
                             BSP_IO_LEVEL_LOW);

    R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);

    g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                             Leds.p_leds[BSP_LED_LED1],
                             BSP_IO_LEVEL_HIGH);

    R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
}
```

编写代码时，始终可以使用 e² studio 的自动完成功能。只需按下 <Ctrl>-<Space>，便会出现一个窗口，显示结构体或函数可能的补全代码。如果单击一个条目，它会被自动插入代码中。

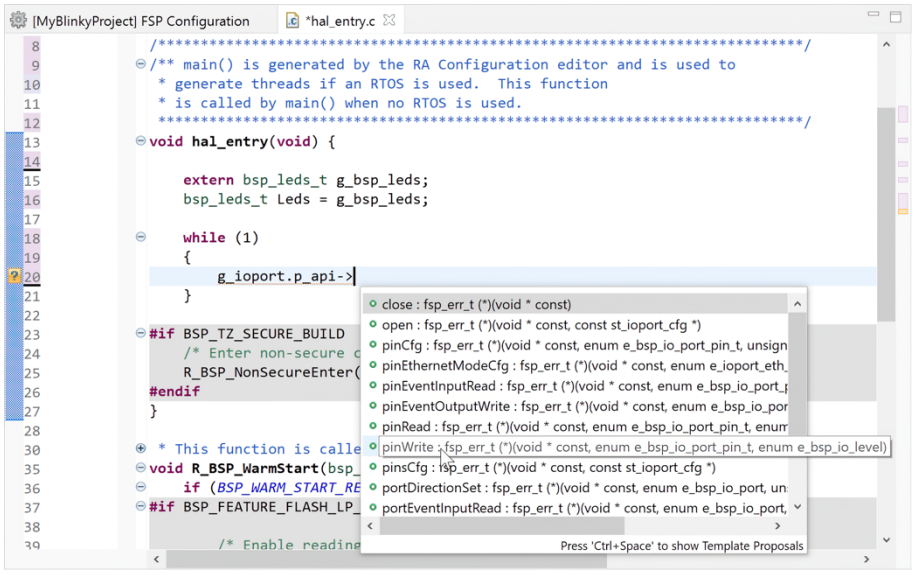


图 3-9：在变量或函数上按下 <Ctrl>-<Space> 将激活 e² studio 的代码补全功能

编写程序时，另一个有用的工具是“Developer Assistance”（开发人员帮助），可以从“Project Explorer”（项目资源管理器）中访问此工具。在使用 FSP 配置器配置了项目的软件堆栈之后，此工具将为您快速了解应用程序代码提供支持。要访问“Developer Assistance”（开发人员帮助），请先在“Project Explorer”（项目资源管理器）中展开项目，此工具随即显示。显示工具后，进一步展开树，直到看到堆栈模块及其 API。选择要使用的 API，然后将对该 API 的调用拖放到源文件中。

现在轮到您进行操作：请将上面的代码行输入到项目的 *hal_entry.c* 文件中。为此，展开项目的 *src* 文件夹，然后双击上述文件。此操作会在编辑器中将其打开。如果您不想自己输入所有内容，也可以从本手册的网站 (**www.renesas.com/ra-book**) 下载完整的项目。

3.4 编译第一个项目

编译有两种不同的配置：调试和发布。调试配置将包含调试程序所需的所有信息，例如变量和函数名，并且还将关闭编译器的某些优化，例如循环展开。这会使调试更加容易，但会增大代码大小、减慢代码执行速度。发布配置将从输出文件中除去所有这些信息，并开启完全优化，从而减小代码大小、加快代码执行速度，但是，您再也无法执行查看变量等操作（除非您知道它们在存储器中的
地址）。

对于第一次测试，可以采用调试配置（也是默认配置）。要编译项目，单击主菜单栏上的“*build*”（编译）  按钮，编译过程随即开始。如果一切正常，编译将以 0 个错误和 0 个警告结束。如果存在编译时错误，则需要返回代码，仔细检查是否正确输入了所有内容。如果未正确输入所有内容，请相应地更改代码。为了让您更轻松地定位错误，点击编译器的反馈将直接引导您进入相应的编辑器窗口（如果可能）。

程序编译成功后，会创建输出文件 *CPK_RA2E1_ExampleProject.elf*，需要先将其下载到处理器，然后才能运行和调试该文件。

3.5 下载和调试第一个项目

下一步是在评估板 (CPK) 上实际运行程序。现在需要将评估板连接到 Windows® 工作站：将电路板随附的 USB 线缆的 micro-B 端插入评估板右侧的 USB 调试端口 J11，将另一端插入 PC 上的空闲端口。POWER LED 应点亮，表示电路板已通电。Windows 操作系统可能会显示一个对话框，指示正在安装 J-Link® 板上调试器的驱动程序，此过程应自动完成。此外，还可能会出现一个窗口，询问是否更新 J-Link® 调试器。强烈建议允许进行此更新。

下载：

要下载程序，必须先创建一个调试配置。单击“Debug”（调试）符号 旁边的小箭头，然后从下拉列表框中选择“*Debug Configurations*”（调试配置）。

在出现的窗口中，突出显示“*Renesas GDB Hardware Debugging*”（瑞萨 GDB 硬件调试）下的 *CPK_RA2E1_BlinkyProject Debug_Flat*。由于项目配置器已经进行了所有必要的设置，因此无需在此对话框中进行任何更改。只需单击窗口右下角的“*Debug*”（调试）。此操作会启动调试器，将代码下载到 CPK 上的 RA2E1 MCU，并询问您是否要切换到“*Debug Perspective*”（调试透视图）。请选择“*Switch*”（切换）。“*Debug Perspective*”（调试透视图）将打开，并且程序计数器将设置为程序的入口点，即复位处理程序。此调试配置仅需要创建一次。下次只需单击“*Debug*”（调试）符号 便可启动调试器。

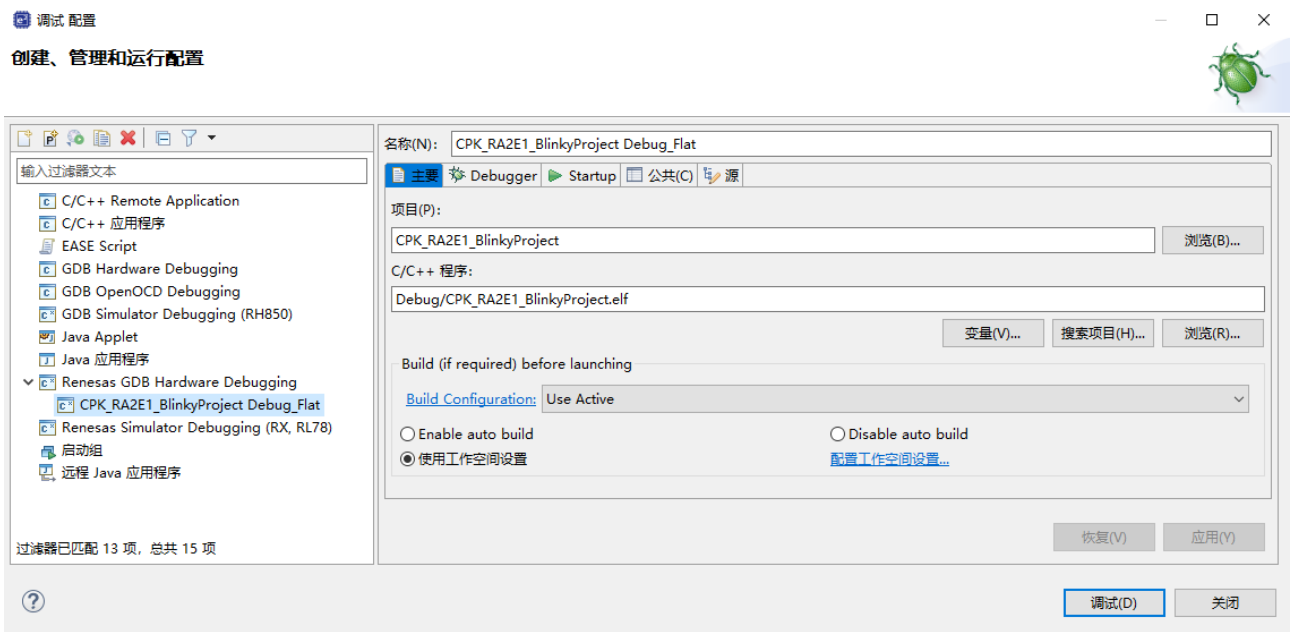


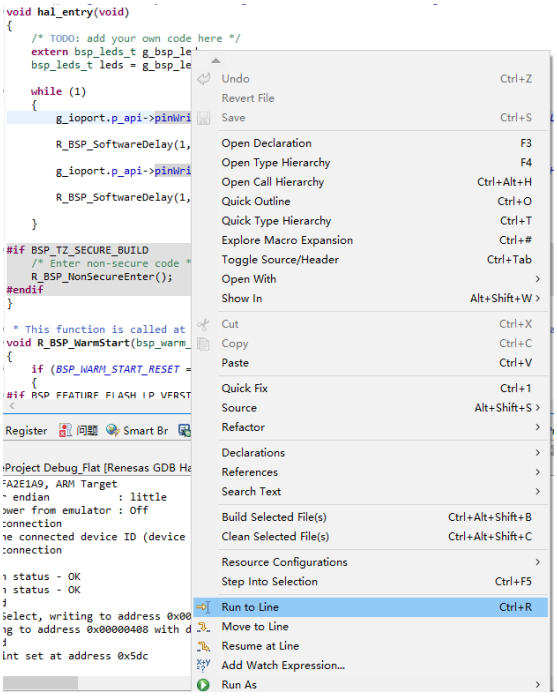
图 3-10：选择 MyBlinkyProject Debug_flat 后，无需在其他选项卡上进行任何更改

运行：

单击“*Resume*”（恢复）按钮 ，下一个停止处将处于 `main()` 中调用 `hal_entry()` 的位置。再次单击该按钮，程序将继续执行，且用户 LED1（红色）将按预期的 1 秒时间间隔闪烁。

观察结果：

如果一切正常，单击主菜单栏上的“*Suspend*”（暂停）按钮 。这将停止执行程序但不会中止调试器的连接。在编辑器视图中，激活文件 *hal_entry.c* 的选项卡，然后右键单击包含对端口的写操作的其中一行；在出现的菜单中，选择“*Run to line*”（运行至指定行）。执行将恢复，程序将在单击的行处停止。现在来看一下右侧包含变量的视图。您将看到列出的 `leds` 结构体。将其展开，浏览和分析不同的字段。调试较大的项目时，此视图会派上用场。



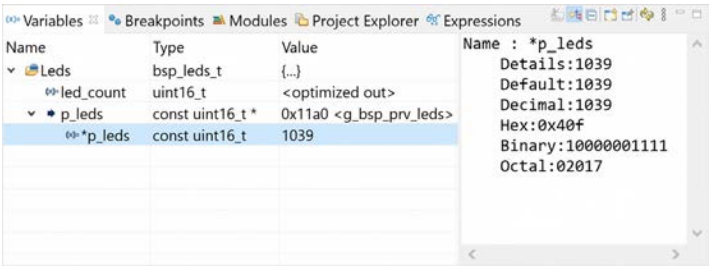


图 3-11：变量及其值可以在“Variables”（变量）视图中进行检查

最后一步是单击“*Terminate*”（终止）按钮 ，结束调试会话，以停止程序的执行。

恭喜！

您已经掌握了 RA 产品家族单片机的第一个程序！

本章要点：

- 项目配置器将创建新项目所需的所有文件和设置。
- FSP 配置器允许编程人员基于图形用户界面轻松配置 FSP 和运行环境。
- 调试配置是调试项目的必需步骤。它会自动创建，只需要激活即可。
- 实现所需功能仅需要很少的代码行。

4. 使用实时操作系统

本章内容基于《瑞萨 RA MCU 基础知识》中的章节 **9 使用实时操作系统** 所作。

您将在本章中学到以下内容：

- 什么是线程、信号量和队列，以及如何使用它们。
- 如何在 e² studio 中向程序添加线程和信号量。
- 如何在 RTOS 控制下通过按钮切换用户 LED 的状态。

上一章中的练习已经利用了瑞萨 RA 系列单片机 (MCU) 灵活配置软件包 (FSP) 的很大一部分。在本章中，您将使用 FreeRTOS™ 实时操作系统创建一个小型应用程序，利用线程控制 LED 并利用信号量实现与按钮的同步。您将亲身体验到这实际上仅需要几个步骤。

我们将从头开始创建完整的项目，因此如果您没有进行过之前的实验，请不必担心。

4.1 线程、信号量和队列

在我们实际深入进行此练习之前，需要定义将在本章和下一章中使用的一些术语，以确保我们能够达成共识。

首先，需要定义术语“线程”。如果您更习惯于“任务”这个表达方式，只需把线程看作是一种任务。有些人甚至互换使用这两个短语。当使用实时操作系统 (RTOS) 时，单片机上运行的应用程序将拆分为几个较小的半独立代码块，每个代码块通常控制程序的一个方面。这些小片段称为线程。一个应用程序中可以存在多个线程，但是在任何给定时间都只能有一个线程处于活动状态，因为 RA 系列单片机是单核器件。每个线程都有自己的堆栈空间，如果需要安全的上下文，则可以将其置于 MCU 的安全侧。每个线程还分配有优先级（相对于应用程序中的其他线程），并且可以处于不同的状态，例如运行、就绪、阻塞或暂停。在 FreeRTOS™ 中，可以通过调用 `eTaskGetState()`API 函数来查询线程的状态。线程间信号传输、同步或通信是通过信号量、队列、互斥、通知、直接任务通知或者流和消息缓冲区来实现的。

信号量是 RTOS 的资源，可用于传输事件和线程同步（以产生者－使用者方式）。使用信号量允许应用程序暂停线程，直到事件发生并发布信号量。如果没有 RTOS，就需要不断地轮询标志变量或创建代码来执行中断服务程序 (ISR) 中的某个操作，这会在相当长的一段时间内阻塞其他中断。使用信号量可快速退出 ISR 并将操作推迟到相关线程。

FreeRTOS 提供计数信号量和二进制信号量。尽管二进制信号量由于仅采用两个值（0 和 1）而非常适合实现任务之间或中断与任务之间的同步，但是计数信号量的计数范围可涵盖 0 到用户在 FSP 配置器中创建信号量期间指定的最大计数。默认值为 256，可支持设计人员执行更复杂的同步操作。

每个信号量都有两个相关的基本操作：`xSemaphoreTake()`（将使信号量递减 1）和 `xSemaphoreGive()`（将使信号量递增 1）。这两个函数有两种形式：一种是可以从中断服务程序内部调用（`xSemaphoreTakeFromISR()` 和 `xSemaphoreGiveFromThread()`）的形式，另一种则是上述可以在线程的正常上下文中调用的形式。

我们需要讨论的最后一个术语是队列，即使在本练习中不使用队列，下一章的练习中也会使用。报文队列是线程间通信的主要方法，它允许在任务之间或中断与任务之间发送消息。消息队列中可以有一条或多条消

息。数据（也可以是指向更大缓冲区的指针）会复制到队列中，即，它存储的是消息本身而非引用。新消息通常置于队列的末尾，但也可以直接发送到开头。接收到的消息将从前面开始删除。

允许的消息大小可在设计时通过 FSP 配置器指定。默认项大小为 4 个字节，默认队列长度（表示队列中可存储的项数）为 20。所有项的大小必须相同。FreeRTOS 中的队列数没有限制；惟一的限制是系统中可用的存储空间。使用 `xQueueSend()` 函数将消息放入队列中，并通过 `xQueueReceive()` 从队列中读取消息。与信号量一样，函数有两种版本：一种可以从线程的上下文调用，另一种可以从 ISR 内部调用。

4.2 使用 e2 studio 将线程添加到 FreeRTOS 中

接下来的练习也是基于 CPK-RA2E1 评估板。这次，我们将使用电路板下方的用户按钮 S1 向应用程序传输事件，应用程序将切换红色 LED 的状态进行响应。为实现目标，我们将使用 FreeRTOS，事件的处理将在线程内进行，并通过信号量进行通知。

第一步是使用项目配置器创建一个新项目。首先，转到“*File* → *New* → “*Renesas RA C/C++ Project*””（文件 → 新建 → *Renesas* C/C++ 项目）。单击“*Next*”（下一步）并在出现的屏幕上输入项目名称，例如 *CPK_RA2E1_RtosProject*。再次单击“*Next*”（下一步）。此操作将转到“*Device and Tools Selection*”（器件和工具选择）窗口。首先，选择一个电路板。选择 *CPK-RA2E1* 并将相应的器件设置为 *R7FA2E1A92DFM*（如果尚未列出）。查看工具链：它应显示为 *GCC Arm® Embedded*。单击“*Next*”（下一步）继续操作。

在当前出现的屏幕中，可以在非 TrustZone® 与安全和非安全 TrustZone 项目之间进行选择。保持“*Flat (Non-TrustZone) Project*”（扁平化（非 TrustZone）项目）处于选中状态，然后单击“*Next*”（下一步）。随即出现“*Build Artifact and RTOS Selection*”（构建工件和 RTOS 选择）窗口。保持设置不变，即在“*Build Artifact Selection*”（构建工件选择）下选择“*Executable*”（可执行文件），在“*RTOS Selection*”（RTOS 选择）下选择 *FreeRTOS*。单击“*Next*”（下一步），转到下一个名为“*Project Template Selection*”（项目模板选择）的屏幕。在此，选择“*FreeRTOS – Minimal – Static Allocation*”（FreeRTOS – 最小化 – 静态分配）。

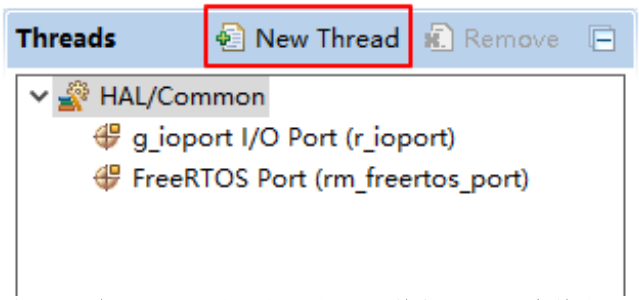


图 4-1：在 FSP 配置器出现之后，将仅显示一个线程。选择“New Thread”（新线程）按钮，添加另一个线程

最后，单击“*Finish*”（完成），在配置器生成项目后，e² studio 将询问您是否切换到“*FSP Configuration*”（FSP 配置）透视图。透视图出现后，直接转到“*Stacks*”（堆）选项卡。该选项卡将在“Threads”（线程）窗格中显示“*HAL/Common*”（HAL/通用）线程的单个条目，其中包含 I/O 端口的驱动程序。单击窗格顶部的“*New Thread*”（新线程）图标（请参见图 4-1 添加新线程）。

现在，在“*Properties*”（属性）视图中更改新线程的属性：将“*Symbol*”（符号）重命名为 *led_thread*，将“*Name*”（名称）重命名为 *LED Thread*。其他属性保持默认值。在“*LED Thread Stack*”（LED 线程堆） 窗格中，单击“*New Stack*”（新线程）按钮图标，选择“*Driver → Input → External IRQ Driver on r_icu*”（驱动程序 → 输入 → r_icu 上的外部 IRQ 驱动程序）（请参见图 4-2）。

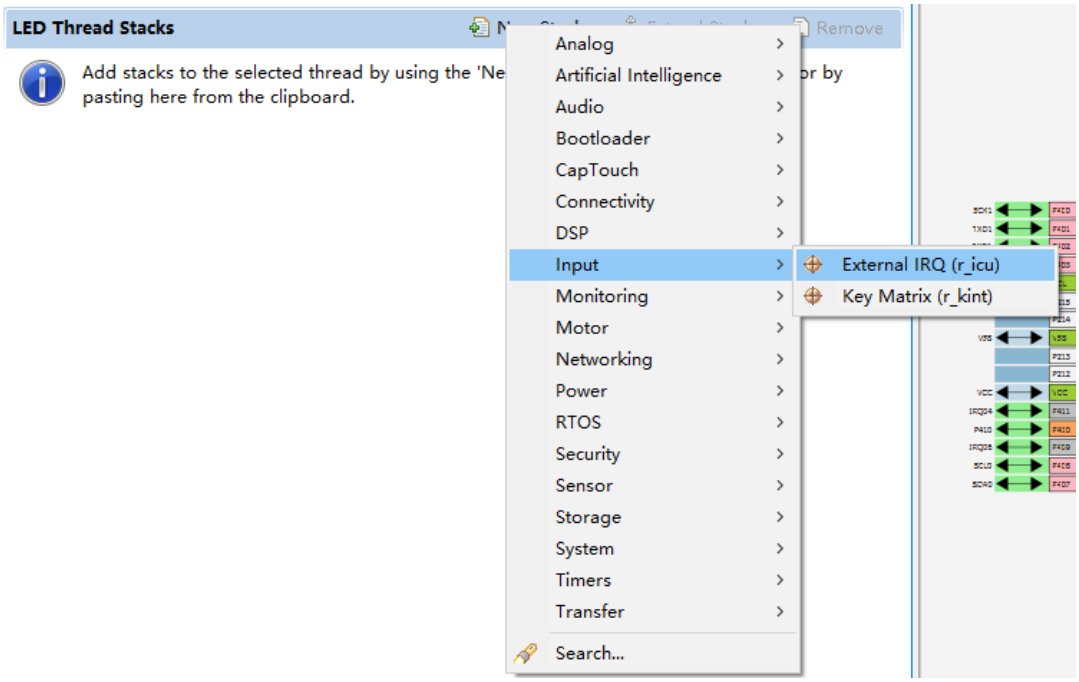


图 4-2：添加新驱动程序只需单击几下鼠标

此操作将为外部中断添加驱动程序。查看新驱动程序的“*Properties*”（属性）：

首先，请确保“*Channel*”（通道）为 3，因为 S1 所连引脚连接到 IRQ03。出于相同的原因，将名称更改为 *g_external_irq03* 或您喜欢的任何名称。

为中断分配优先级 2，启动期间 FSP 将不会允许该中断。也可以选择任何其他优先级，但开始时最好选择优先级 2，因为即使在较大的系统中，也很少会遇到中断优先级冲突。请注意，优先级 3 是为系统时钟节拍定时器 (systick) 保留的，因此不应被其他中断使用。

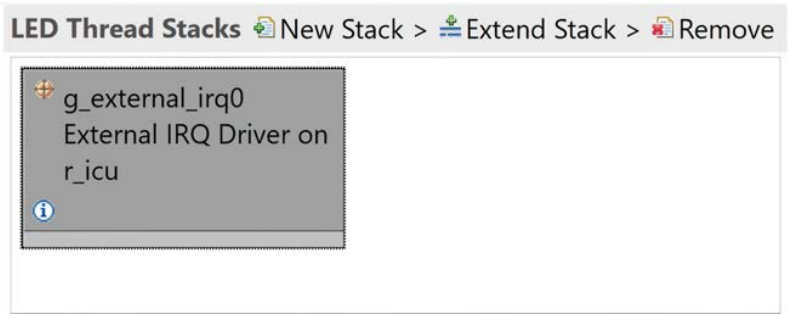


图 4-3：堆元素的灰色条表示此驱动程序是模块实例，只能由另一个 FSP 模块实例引用

将“*Trigger*”（触发器）从“*Rising*”（上升）更改为“*Falling*”（下降）以捕捉按钮激活操作，并将“*Digital Filtering*”（数字滤波）从“*Disabled*”（禁用）更改为“*Enabled*”（启用）。始终将“*Digital Filtering Sample Clock*”（数字滤波采样时钟）设置为 *PCLK / 64*。这将有助于对按钮去抖。最后，用 *external_irq03_callback* 替换 *Callback* 行中的 *NULL*。每次按下 S1 都会调用此函数。在稍后创建应用程序时，我们将为回调函数本身添加代码。图 4-4 给出了必要设置的摘要。

属性	值
Common	
Parameter Checking	Default (BSP)
Module g_external_irq03 External IRQ (r_icu)	
Name	g_external_irq03
Channel	3
Trigger	Falling
Digital Filtering	Enabled
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK / 64
Callback	external_irq03_callback
Pin Interrupt Priority	Priority 2
Pins	
IRQ03	P004

图 4-4：应用程序所需的 IRQ 驱动程序的属性

现在，只需要执行几个步骤，即可编译和下载程序。下一步是添加信号量。

为此，请在“*LED Thread Objects*”（LED 线程对象）窗格中单击“*New Object*”（新对象）按钮。如果看到的不是此窗格，而是“*HAL/Common Objects*”（HAL/通用对象）窗格，则突出显示“*Threads*”（线程）窗格中的“*LED Thread*”（LED 线程），随即将显示此窗格。添加一个*二进制信号量*，我们需要在按下按钮时通知 *LED 线程*。将信号量的“*Symbol*”（符号）属性更改为 *g_s1_semaphore*，并将“*Memory Allocation*”（存储器分配）保留为“*Static*”（静态）。现在，FSP 配置器中的“*Stacks*”（堆）选项卡的外观应类似于图 4-5。

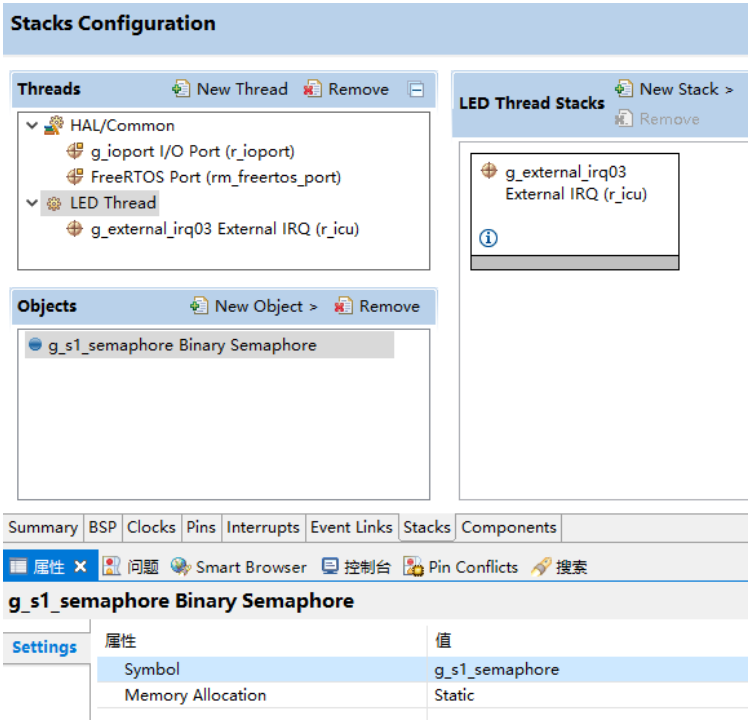


图 4-5：这是添加 LED 线程和信号量后“Stacks”（堆）选项卡应呈现的外观

FSP 配置器中的最后一步是将 S1 连接的 I/O 引脚配置为 IRQ03 输入。为此，请激活配置器中的“*Pins*”（引脚）选项卡，展开“*Ports → P0*”（端口 → P0），然后选择 *P004*。在 CPK-RA2L1/RA2E1 评估板上，这是 S1 连接的端口。在右侧的“*Pin Configuration*”（引脚配置）窗格中，为其指定*符号名称* *SW1*，并确保其他设置与图 4-6 中的设置相同。通常，配置器应该已为您完成了相关设置。如果没有完成，请相应调整。请注意，右侧的封装查看器将突出显示引脚 P004，这样便可获得引脚位置的图形参考。

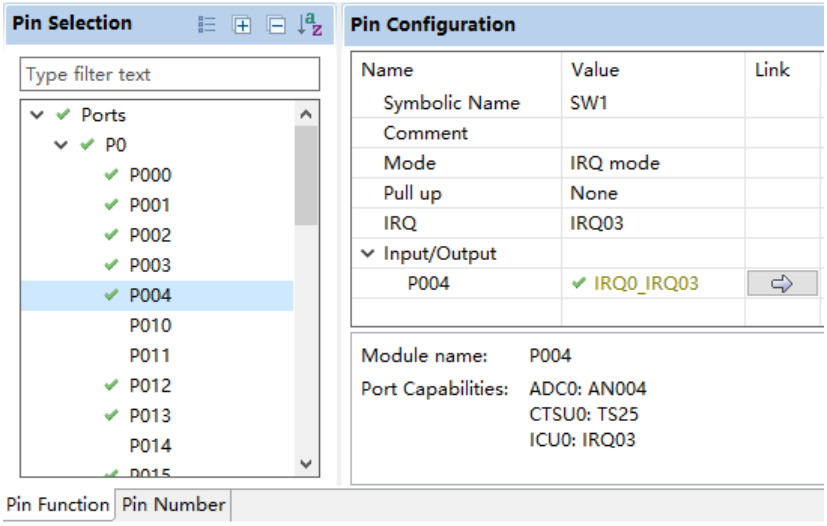


图 4-6：应已为 IRQ03 正确配置了端口 P004

完成此操作后，即完成了配置器中的设置。保存更改，然后单击其顶部的“*Generate Project Content*”（生成项目内容）图标以创建必要的文件、文件夹和设置。

需要执行的最后一项任务是添加初始化 `leds` 结构体所需的代码，编写几行代码来切换 LED 并读取信号量，然后创建将设置信号量的回调函数。可以在本章末尾查看完整代码。

由于我们正在使用 *LED 线程*处理按钮和切换 LED 的状态，因此本次需要将相关代码添加到 *led_thread_entry.c* 文件中。在“*Project Explorer*”（项目资源管理器）中双击文件名以在编辑器中将其打开。如果未显示文件，请展开项目文件夹，然后展开 *src* 目录。与第 3 章中的练习一样，为 LED 添加结构体并对其进行初始化。需要定义用户 LED（红色）所连 I/O 引脚的电平的另一个变量。将其命名为 `led_level`。该变量的类型需要采用 `ioport_level_t`，并且应初始化为 `IOPORT_LEVEL_HIGH`（在 CPK-RA2L1/RA2E1 上，“高”电平对应于“开启”）。

下一步将是打开并启用连接到板上 S1 的 IRQ03。为此，请使用 IRQ FSP 驱动程序的打开和使能功能。完成后，初始化即完成。

```
g_external_irq03.p_api->open(g_external_irq03.p_ctrl,
                             g_external_irq03.p_cfg);
 g_external_irq03.p_api->enable(g_external_irq03.p_ctrl);
```

在 `while(1)` 循环内部，需要添加一些语句并删除 `vTaskDelay(1);` 语句。先使用函数调用将 `led_level` 的值写入用户 LED（红色）的 I/O 引脚的输出寄存器，然后执行相关语句切换该引脚的电平。有几种方法可以实现这一点。自行实现，回顾第 3 章的练习或查看本章结尾的代码。不要忘记 e² studio 的智能手册功能，它会提供很大帮助！

`While(1)` 循环中的最后一条语句是调用 `xSemaphoreTake()`，将信号量的地址和常量 `portMAX_DELAY` 作为参数。后一个参数将通知 RTOS 无限期地暂停线程，直到从 IRQ03 中断服务程序调用的回调函数中释放信号量为止。

最后要执行的操作是添加回调函数本身。该函数应尽可能短，因为它将在中断服务程序的上下文中执行。编写此函数十分简单：只需转到“Project Explorer”（项目资源管理器）中的“*Developer Assistance → LED Thread → g_external_irq03 External IRQ Driver on r_icu*”（开发人员帮助 → LED 线程 → r_icu 上的 g_external_irq03 外部 IRQ 驱动程序），然后将所出现列表末尾的回调函数定义拖放到源文件中。

```
void external_irq03_callback(external_irq_callback_args_t *p_args);
```

在回调函数内，添加以下两行代码：

```
FSP_PARAMETER_NOT_USED(p_args);
xSemaphoreGiveFromISR(g_s1_semaphore, NULL);
```

第一行中的宏将告知编译器回调函数不使用参数 `p_args`，从而避免编译器发出警告，而第二行中的宏则在每次按下按钮 S1 时释放信号量。注意，必须使用 give 系列函数的中断保存版本，因为此函数调用发生在 ISR 的上下文内。此调用的第二个参数是 `*pxHigherPriorityTaskWoken`。如果可能有一个或多个任务由于信号量发生阻塞并等待该信号量变为可用状态，并且其中一个任务的优先级高于发生中断时执行的任务，则此参数将在调用 `xSemaphoreGiveFromISR`() 后变为 true。在这种情况下，应在退出中断之前执行上下文切换。由于在我们的示例中，没有其他任务依赖于此信号量，因此可以将此参数设置为 `NULL`。

完成所有代码编写后，单击“*Build*”（编译）图标（“锤子”），编译项目。如果编译后存在错误，请返回程序，借助“*Problems*”（问题）视图中显示的编译器反馈修复问题。

如果项目编译成功，请单击“*Debug*”（调试）图标旁的小箭头，选择“*Debug Configurations*”（调试配置），然后展开“*Renesas GDB Hardware Debugging*”（瑞萨 GDB 硬件调试）。选择 *MyRtosProject Debug_Flat*，或者为项目指定的名称，然后单击“*Debug*”（调试）。这样便可启动调试器。如果您需要更多相关信息，请回顾第 3 章中的相关部分。调试器启动并运行后，单击“*Resume*”（恢复）两次。现在程序正在执行，每次按下 CPK 上的 S1 时，用户 LED1（红色）都应切换状态。

最后一点：在实际应用中，应执行错误检查以确保程序正确运行。为了清楚和简洁起见，本示例中将其省略。

```
 #include “led_thread.h”

 void led_thread_entry(void *pvParameters)
 {
FSP_PARAMETER_NOT_USED (pvParameters);
extern bsp_leds_t g_bsp_leds;
bsp_leds_t Leds = g_bsp_leds;

uint8_t led_level = BSP_IO_LEVEL_HIGH;

g_external_irq03.p_api->open(g_external_irq03.p_ctrl,
                             g_external_irq03.p_cfg);
g_external_irq03.p_api->enable(g_external_irq03.p_ctrl);

while (1)
    {
        g_ioport.p_api->pinWrite(&g_ioport_ctrl,
                                 Leds.p_leds[BSP_LED_LED1],led_level);

        if (led_level == BSP_IO_LEVEL_HIGH)
        {
            led_level = BSP_IO_LEVEL_LOW;
        }
        else
        {
            led_level = BSP_IO_LEVEL_HIGH;
        }

        xSemaphoreTake(g_s1_semaphore, portMAX_DELAY);
}
 }

 /* callback function for the SW1 push button; sets the semaphore */
 void external_irq03_callback(external_irq_callback_args_t *  p_args)
 {
   FSP_PARAMETER_NOT_USED(p_args);
   xSemaphoreGiveFromISR(g_s1_semaphore, NULL);
 }
```

恭喜！

您已成功完成本练习！

本章要点：

- 通过使用全面的 API，可以轻松使用 FSP 的各个函数。
- FSP 将处理大多数与用户代码无关的内容。
- 使用 FreeRTOS™ 十分简单，因为 FSP 配置器的使用非常直观，添加线程和信号量也相当轻松。

5. 使用“Renesas QE”获取触摸按键的状态

您将在本章中学到以下内容：

- 如何使用 RA 产品家族微控制器的“灵活配置软件包”的中间件 CapTouch 进行非接触按键的检测。
- 如何使用 Renesas QE for Capacitive Touch 工具完成按键的 Tuning 过程。

在本部分，我们将使用瑞萨 RA 产品家族微控制器的“Renesas QE”工具，在按下触摸按键时，控制用户 LED1（红色）和 LED2（蓝色）的点亮状态。按下 BTN1 时，LED1（红色）；按下 BTN2 时，LED2（蓝色）；按下 BTN3 时，LED1（红色）和 LED2（蓝色）同时亮。

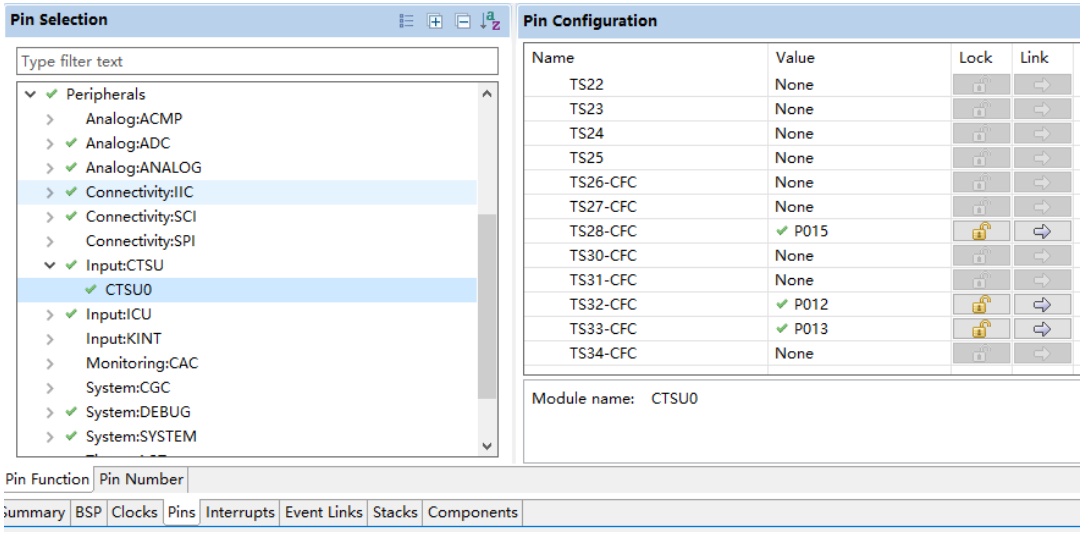
该端口的设置将在 Renesas QE 的图形界面中完成，程序员只需完成极少的编程工作。在执行该练习中的编程任务时，可再次体验到 Renesas QE 给用户提供的便利：进行触摸按键的 Tunning 非常方便。这里需要提醒的是，使用 FSP3.5.0 的情况下，需要使用 QE for Capacitive Touch 3.0.2，如何安装此工具，请参照以下链接：

https://www.renesas.com/jp/zh/document/rln/qe-capacitive-touch-v302-release-note

5.1 使用 FSP 配置器设置 CTSU 端口

如果在完成上次练习后已关闭 e² studio，请再次打开并创建一个新项目。到目前为止您应该已经掌握了 RA 的相关知识，这里将不再赘述每个步骤，因为大部分需要执行的任务在之前的实验中已经做过介绍。将新项目命名为 *CPK_RA2E1_CTUSProject*，在进入 *“Device and Tools Selection ”*（器件和工具选择）屏幕后，选择 *CPK-RA2E1* 作为电路板，我们将再次使用该评估板进行实验。在 *“Project Type Selection ”*（项目类型选择）页面，确保 *“Flat (Non-TrustZone) Project ”*（简单（非 TrustZone）项目）处于启用状态，并确保 *“RTOS Selection ”*（RTOS 选择）下的 *“No RTOS ”*（无 RTOS）条目已激活。最后，在 *“Project Template Selection ”*（项目模板选择）页面上选择 *“Bare Metal – Minimal ”*（裸机 – 最小化），然后单击 *“Finish ”*（完成）。

在项目配置器已创建项目并显示 FSP 配置器后，移至 *“Pins ”*（引脚）选项卡并打开 *“Peripherals ”*（外围设备）条目。 在外围设备列表中，滚动到 *“Input: CTSU ”*（输入： CTSU）。打开配置并确保 CTSU0 如下所示。



然后，转到 *“Stacks”*（堆）选项卡。首先，我们需要添加用于连接到触摸按键的模块。在 *“HAL/Common Stacks”*（HAL/通用堆栈）窗格上，单击 *“New Stack”*（新堆），然后选择“CapTouch → *Touch (rm_touch)”*。

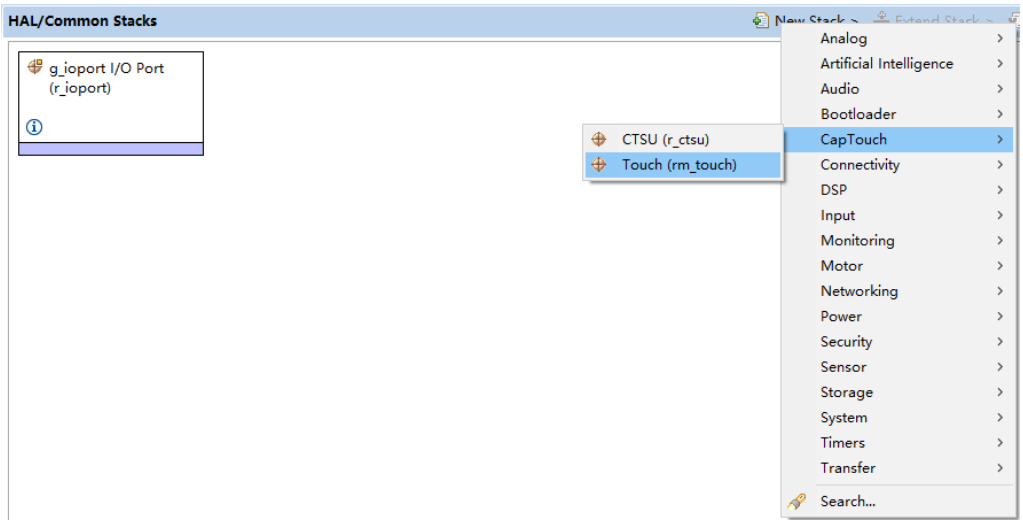
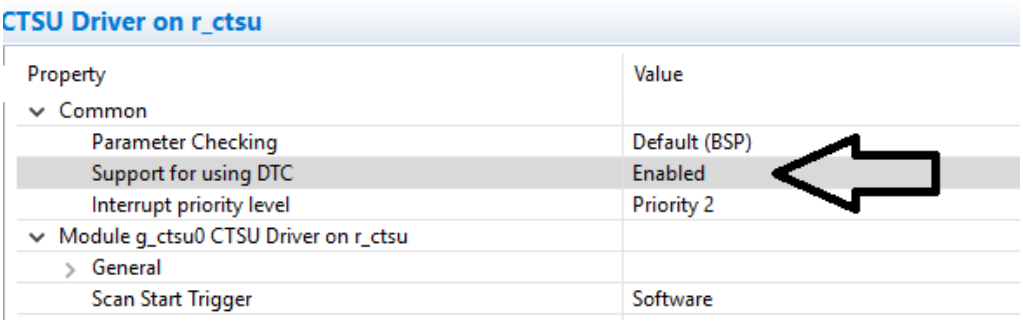
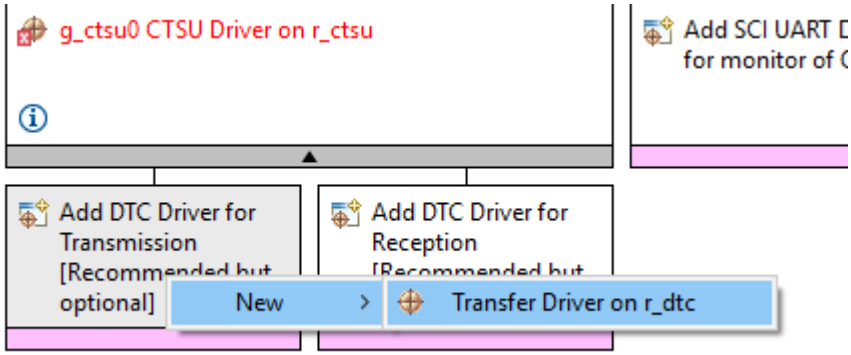


图 5-1：首先添加 CTSU 的驱动程序

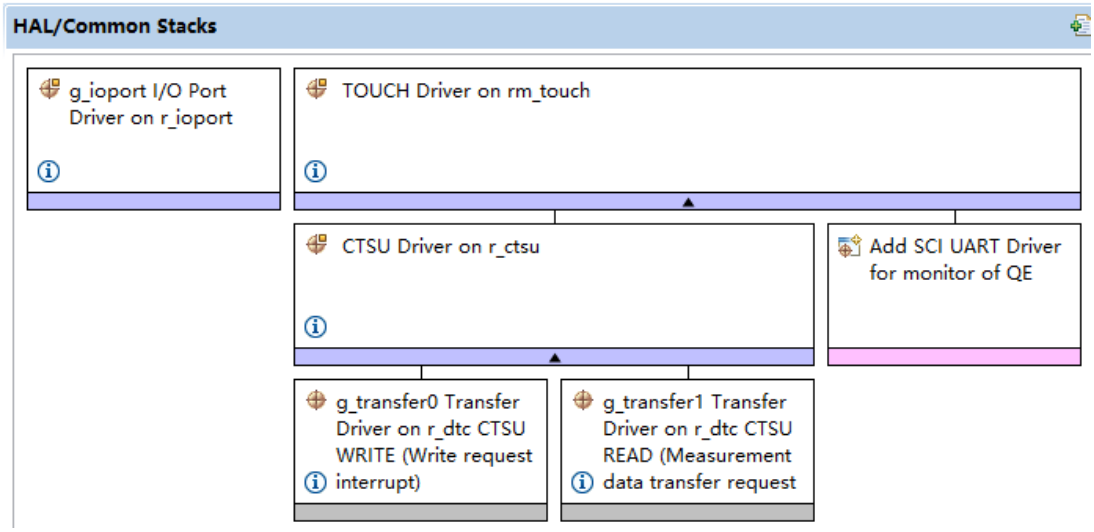
单击 驱动程序 *“CTSU (r_ctsu)”*，然后在 *“Property”*（属性）选项卡下的 *“Support for using DTC”*（支持使用 DTC） 项选择 *“Enabled”*（启用）。



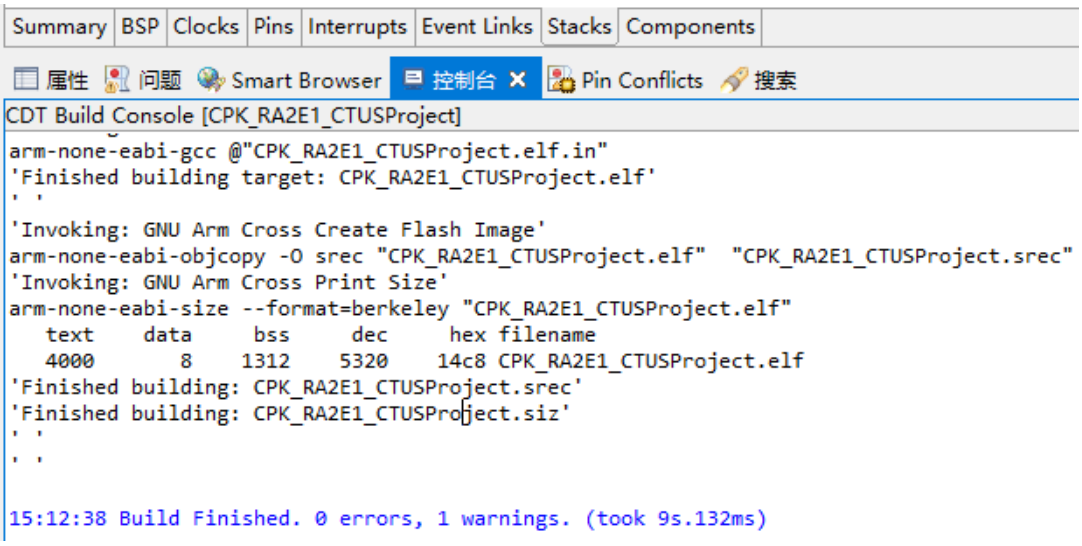
通过单击 *“Add DTC Driver”*（添加 DTC 驱动程序）框并为两个条目添加模块，将用于传输和接收的 DTC 传输驱动程序添加到模块堆栈配置中。



完成后，堆栈应如下所示：



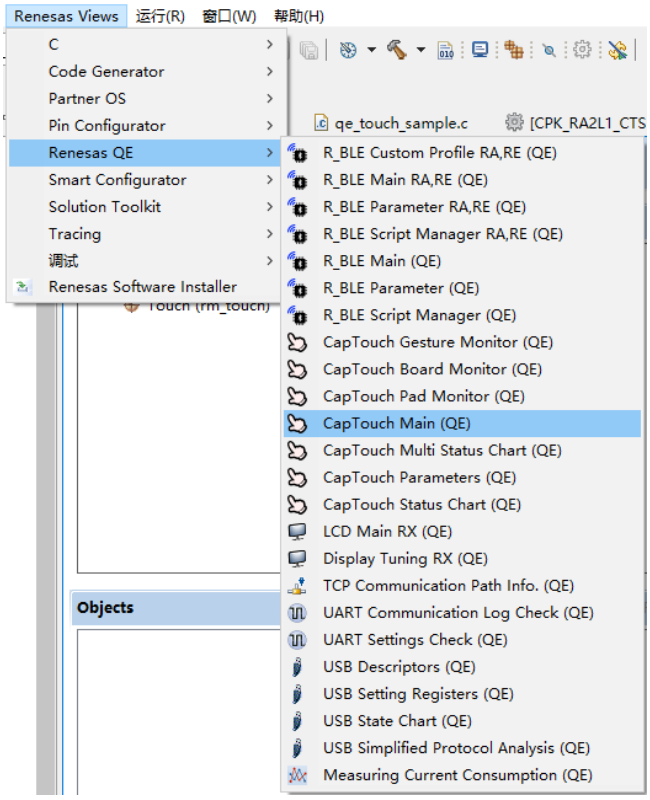
至此，已经完成了必须在 FSP 配置器中进行的设置。保存配置，然后单击屏幕右上角的 *“Generate Project Content”*（生成项目内容）按钮，以提取文件并创建所需的设置。最后一步，再次切换到 C/C++ 透视图并构建项目，在构建过程中不应显示错误。



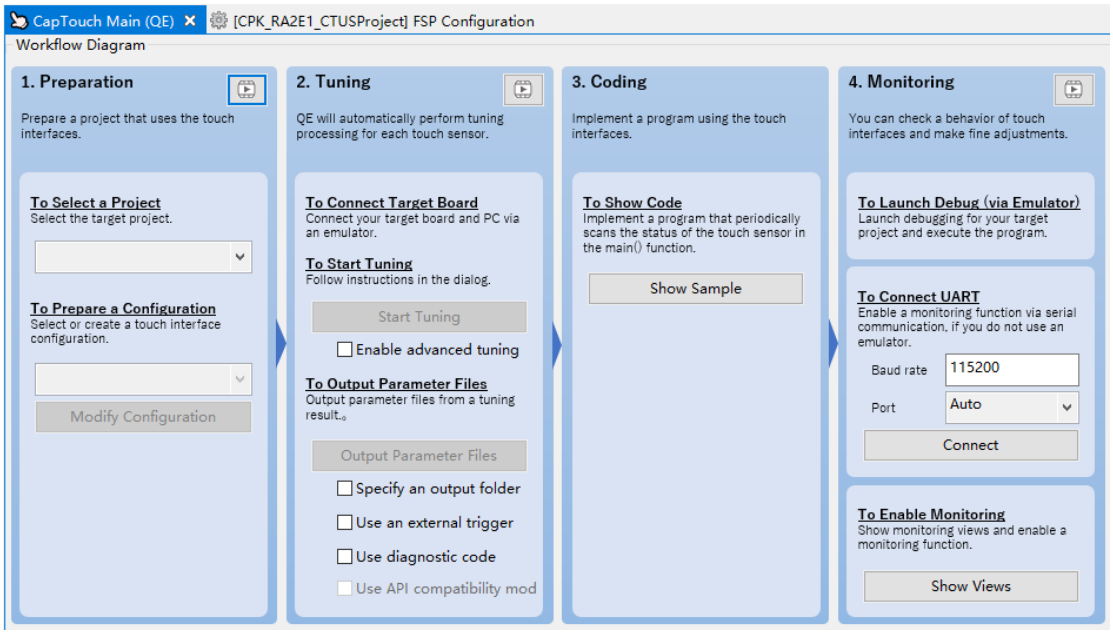
5.2 创建电容式触摸配置

下一步将是创建触摸界面。也就是将电路板及其传感器的图形表示与触摸项目中启用的物理 MCU 传感器引脚相关联。 这里，QE for Capacitive Touch RA 插件将用于设置和配置触摸界面。

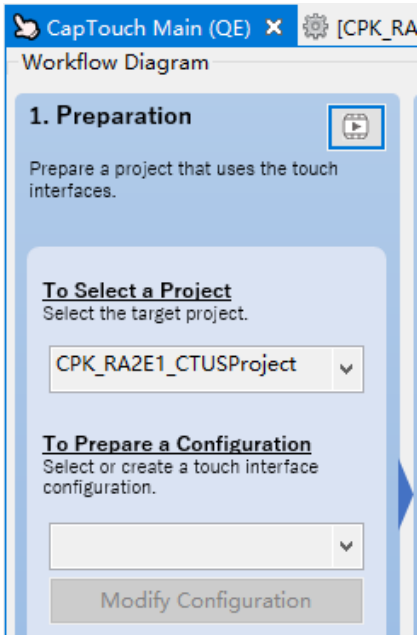
在 e2Studio IDE 中，通过 Renesas Views->Renesas QE->CapTouch Main (QE) 打开主透视图，为项目配置电容式触摸。



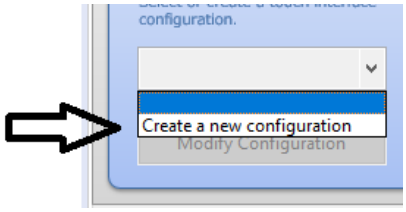
e2Studio IDE 将打开一个类似于下图的窗格：



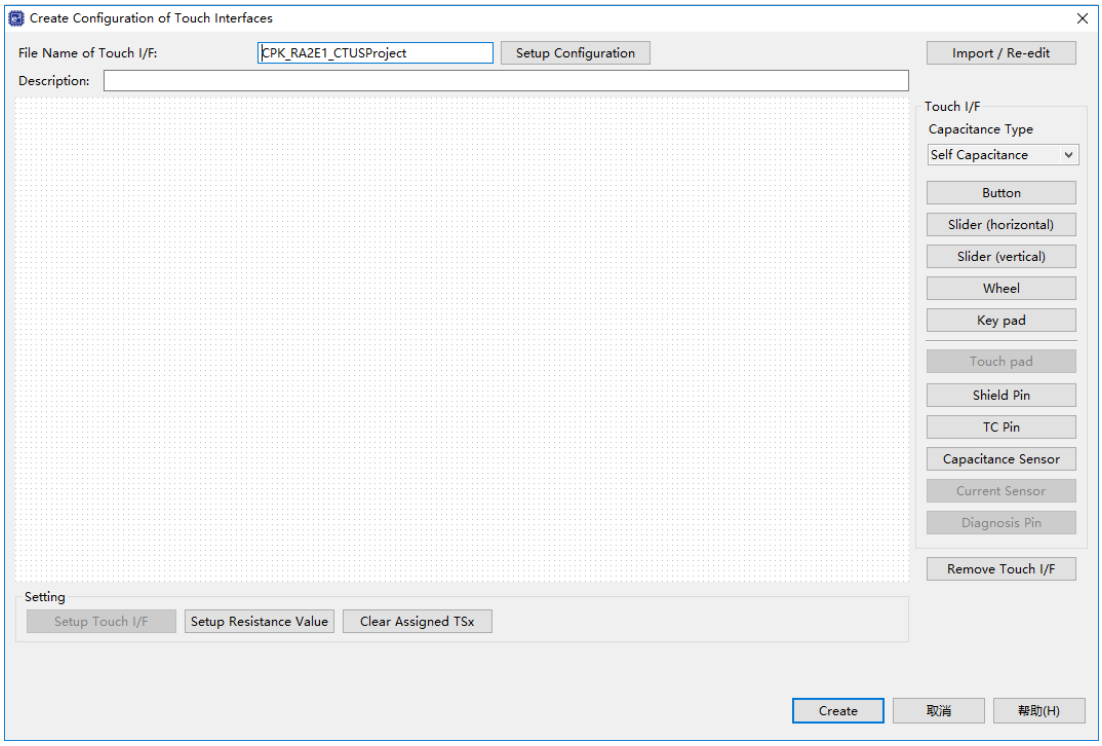
在 CapTouch Main / Sensor Tuner RA (QE) 窗格中，通过使用下拉选项卡并选择 CPK_RA2E1_CTSUProject 项目来选择要为其配置触摸界面的项目，如下所示：



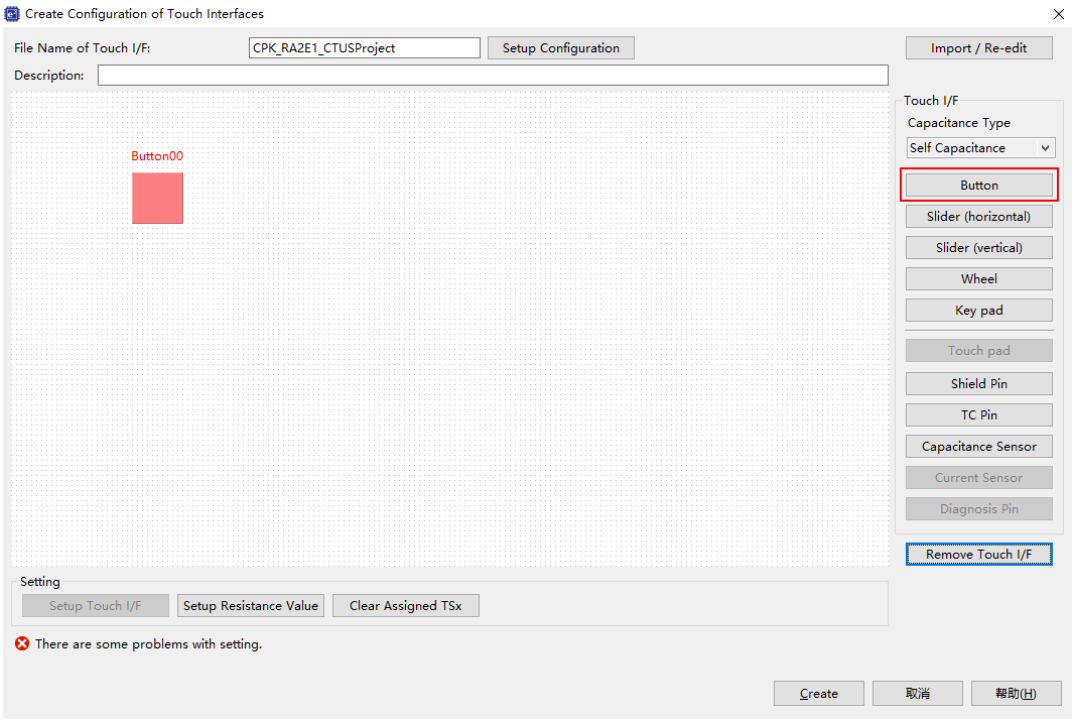
接下来，通过使用下方的下拉菜单并选择 *“Create a new configuration”*（创建新配置）来创建新的触摸配置。



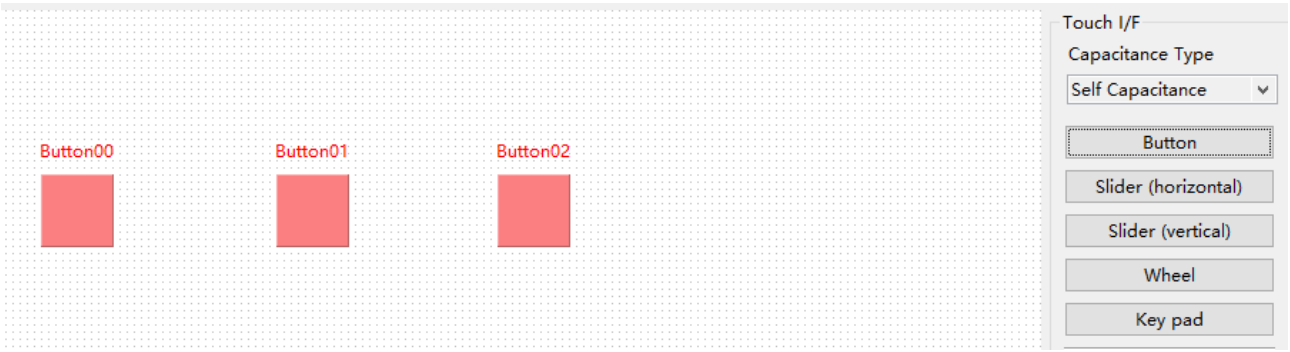
这将打开一个新菜单窗口，显示用于创建触摸界面的默认空白画布：



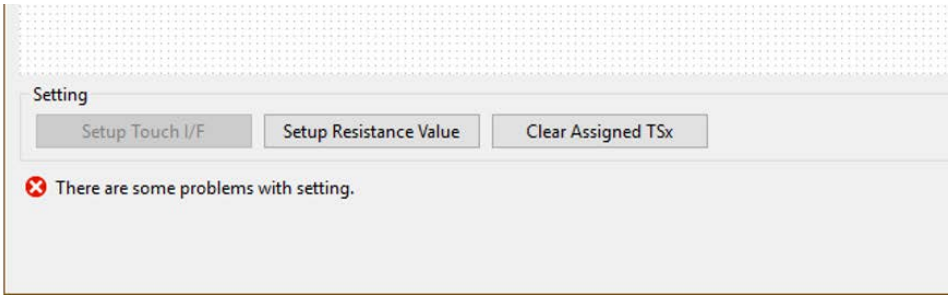
从画布右侧选择 *“Button”*（按钮）菜单项并将光标移动到画布上，将按钮添加到画布。单击鼠标左键放下按钮图标：



通过向画布添加另外两个按钮来完成配置。 添加所有三个按钮后，按 ESC 键退出。 画布将类似于下图：



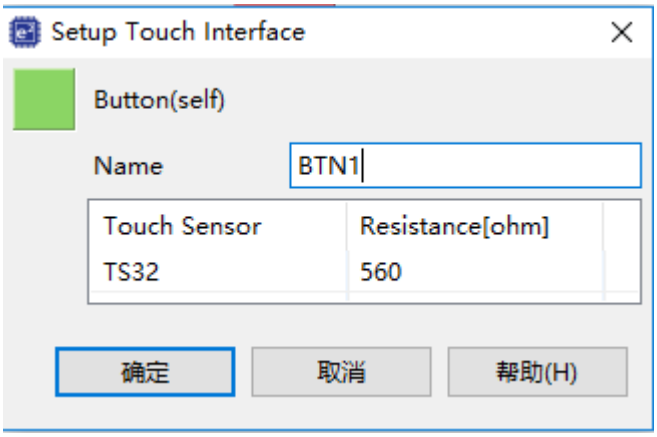
此时，画布下方会显示 RED X 及文本 *“There are some problems with setting”*（设置存在一些问题）。



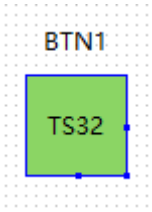
这表明画布上的按钮没有绑定到任何 MCU 传感器引脚。 按钮（以及其他已添加了的组件）也将显示为红色，这表明它们的配置存在问题。

您需为 QE 画布上的按钮指定电路板丝印上标明的名称。另外还可以看到传感器通道，方便使用画布时参考。

要在传感器焊盘和物理触摸通道引脚之间建立连接，需双击 Button00，此时将出现一个对话框。然后，基于原理图，通过下拉菜单和鼠标选择 TS32 作为 MCU 传感器以分配给该按钮。 此外，请为传感器指定一个可读的名称。这里，我们可以回顾一下之前的步骤，选择使用 BTN1 的丝印名称。这些设置可以让开发应用程序更容易，而且带有大量按钮的板子也更方便调整和使用。



请注意，当一个在 *“Smart Configurator Setup ”*（智能配置）中被配置过的有效的 MCU 传感器引脚被指定时，按钮将变为绿色，这表明它已被正确指定。此外，MCU 传感器通道也将出现在按钮图像中。 如下所示：

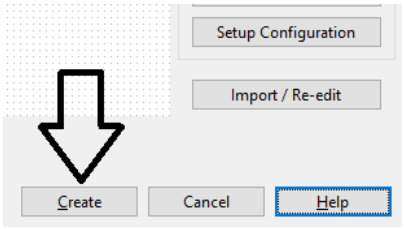


接下来，继续按如下方式进行其余连接：

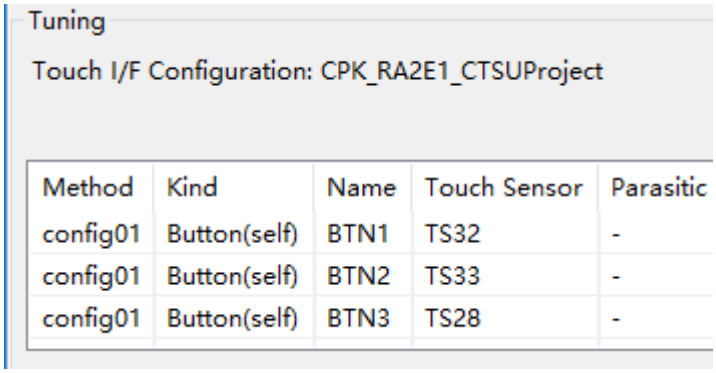
• 双击 Button01，将出现一个对话框。此时使用下拉和鼠标，选择 TS33 作为 MCU 传感器分配给该按钮，并将名称分配为 BTN2。

• 双击 Button02，将出现一个对话框。此时使用下拉和鼠标，选择 TS28 作为 MCU 传感器分配给该按钮，并将名称分配为 BTN3。

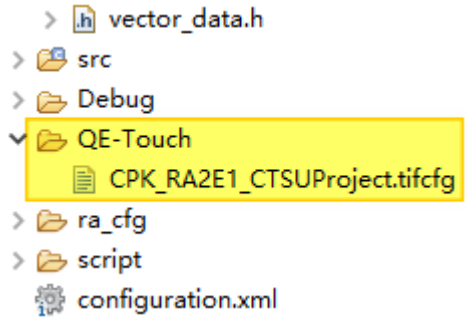
单击对话框中的 *“Create ”*（创建）， 开始设置触摸界面：



此时，CapTouch Main /Sensor Tuner (QE) 窗口的主视图窗格中会显示触摸界面的配置以及用户创建的命名。 向上拖动窗格可以看到对话框。

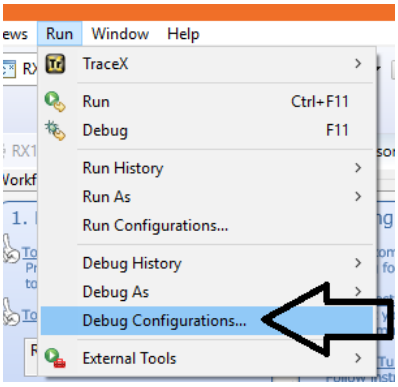


成功创建配置后，在 *“Project Explorer ”*（项目资源管理器）中展开项目后，将显示配置文件 ***Touch Interface Configuration File (.tifcfg)*** 。

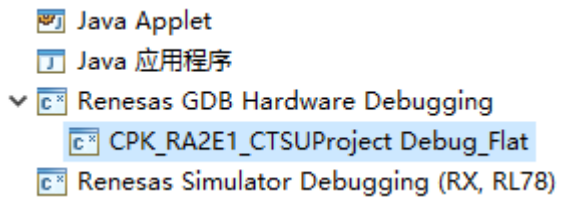


这里请注意：一旦配置了项目，便**无需**调整触摸按钮。如果用户希望继续开发其他应用程序，可以修改项目或添加额外的模块，而无需 *“Tuning ”*（调节）。 但请谨记，除非完成 *“Tuning ”*（调节）并输出所需文件，否则无法进行触摸检测操作或 API 调用。

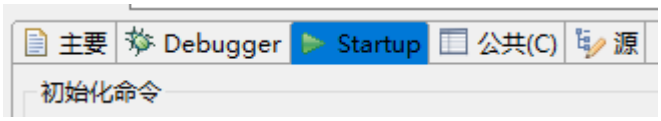
这里需要对调试选项稍作调整，以便将特殊的调优内核下载到 MCU RAM 中并在 main() 处自动中断。单击 **Run->Debug Configurations…**进行 *“Debug Configurations ”*（调试配置）。



现在将打开一个对话框。 在左侧窗格的列表中，打开 *“Renesas GDB Hardware Debugging”*（Renesas GDB 硬件调试）条目并单击选择 ***CPK_RA2E1_CTSUProject Debug_Flat***。



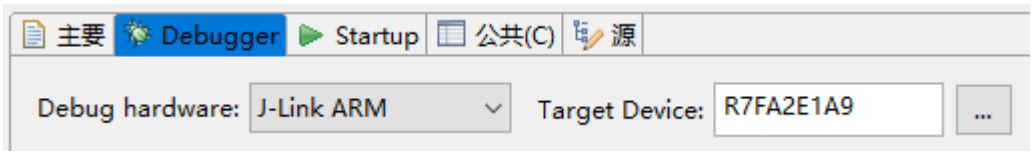
选择 *“Startup”*（启动）选项卡



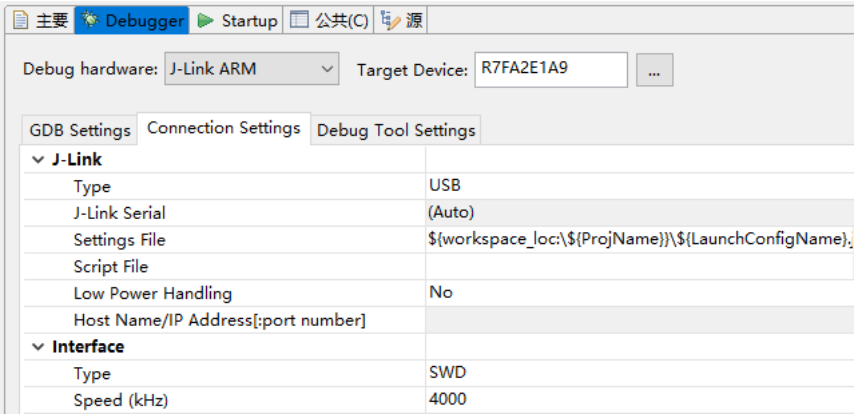
在通常设置时，要选中 “Set breakpoint at: main”（设置断点）和“Resume”（继续）两个复选框，如下所示。您可能需要在对话框中向下滚动才能看到这些复选框。

切换到 *“Common”*（通用）选项卡并确保**未选**“在后台启动”。

选择 *“Debugger”*（调试器）选项卡。 确保 *“Debugger hardware”*（调试硬件）是 **J-Link Arm**，*“Target Device”*（目标设备）是 **R7FA2E1A9**。



在 *“Debugger”*（调试器）选项卡中选择 *“Connection Settings”*（连接设置）并确保 *“Interface Type”*（接口类型）为 **SWD**。



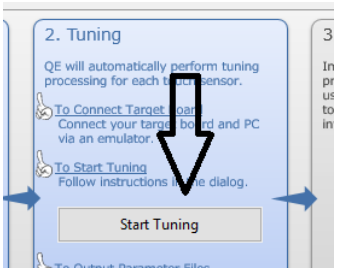
依次点击 *“Apply”*（应用）和 *“Close”*（关闭），以保存设置。 这样就完成了用于调整的项目配置和调试设置。

5.3 电容式触控项目 Tuning（调节）

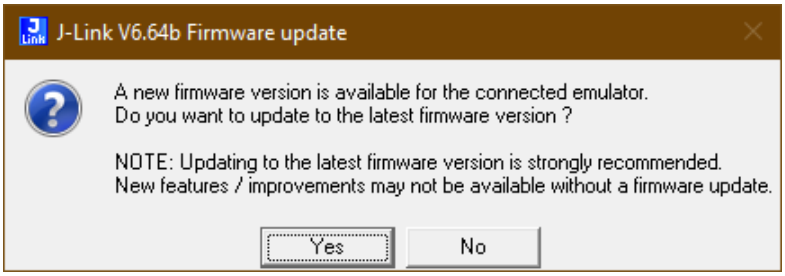
本实验将演示对用户工程进行自动 Tuning（调节）的过程。首先在 QE for Capacitive Touch RA 插件中启动 *“Tuning”*（调节）向导，在 e² studio IDE 中执行 *“Tuning”*（调节），最后完成 *“Tuning”*（调节）并将所需的参数文件输出到应用工程中。

请确保硬件已按照前面内容进行连接。

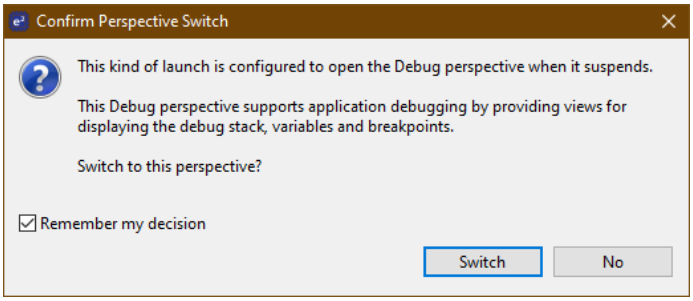
要开始自动 Tuning（调节）过程，请单击 CapTouch Main / Sensor Tuner RA (QE) e2studio IDE 中的 *“Start Tuning”*（开启调节）按钮。



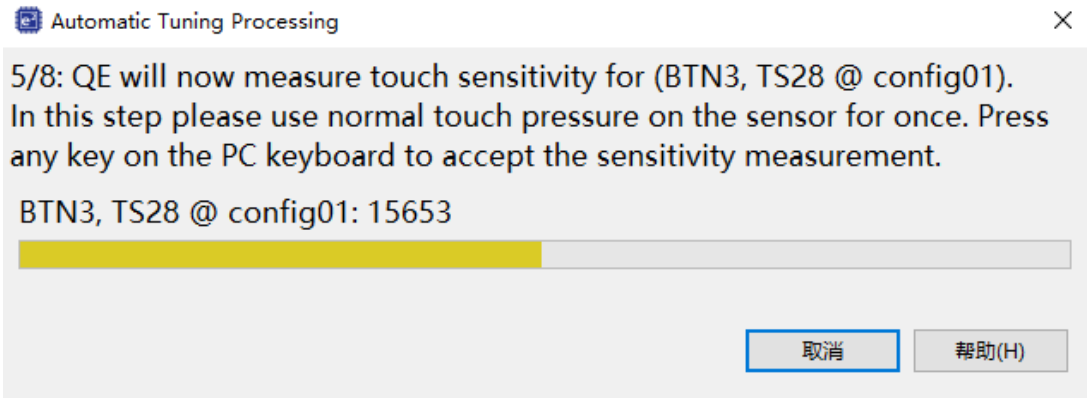
此时可能会显示以下消息以更新 JLINK 固件。 单击 Yes 更新固件，IDE 将自动执行此操作。



Tuning（调节）向导将启动目标板上的调试会话，e² studio 将显示一条消息，用以提示将切换到调试透视图。 选中 *“Remember my decision”*（记住我的决定）复选框，单击 *“Switch”*（切换）以继续调试过程和电容式触摸自动 Tuning（调节）的 QE。

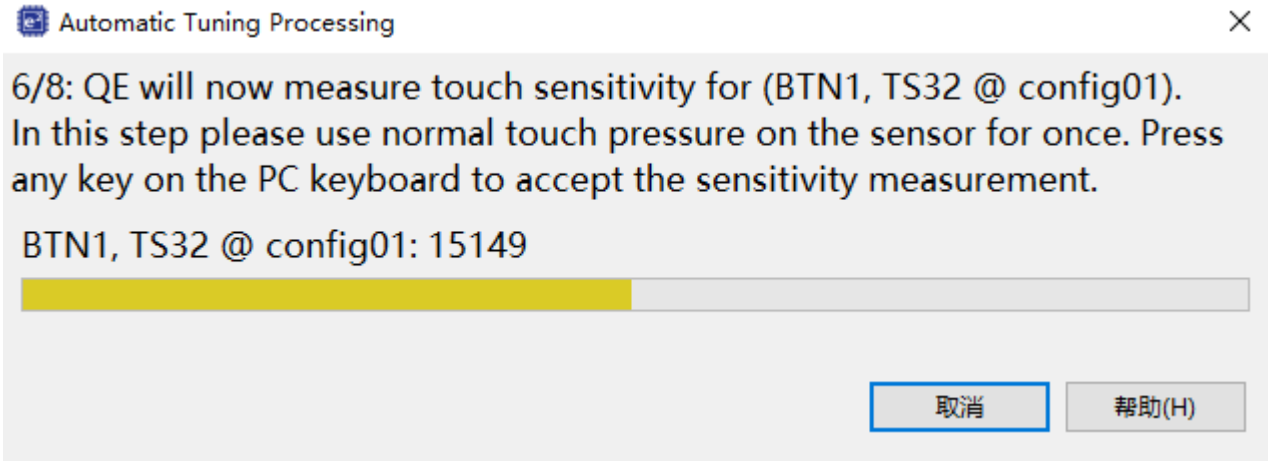


注意：经过几个自动化步骤后，您将看到包含如下所示信息的对话框。 这是 Tuning（调节）过程的触摸灵敏度测试步骤。您可以看到传入的“触摸计数”，即在传感器上看到的电容。 如果触摸板上的传感器（此时为 BTN3、TS28），将会使条形图和触摸计数增加，这是因为我们正在更改/增加触摸传感器的电容而导致的计数上升。且 Tuning（调节）总是从最低传感器编号到最高传感器编号进行。

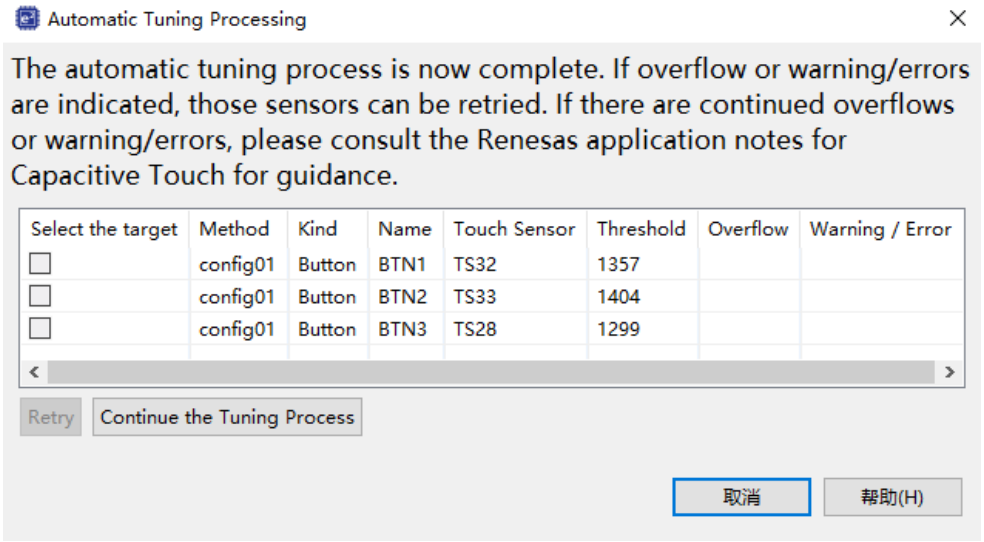


您需要在电极板 （BTN3/TS28）上的传感器上使用**一般的触摸力度**。当按下板上 BTN1 位置时，便会看到进度条向右进展，同时触摸计数也在增加。保持按压并点击电脑键盘上的任意键以接受测量，仅需一次即可。示例如下所示。 测量结果将因显示的计数而异。

对按钮 BTN1 和 BTN2 重复此过程。

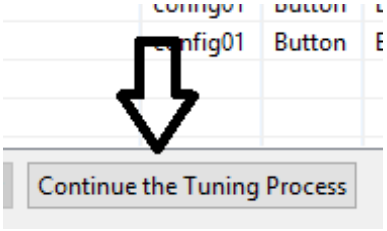


完成后，您将看到如下所示的界面。理想情况下，使用此硬件应该会看到 1000 次或更多计数的触摸阈值。这是中间件用来确定是否发生触摸事件的检测阈值。



注意：在 Tuning（调节）过程中您可能会收到“寄生电容< 10pF”的警告。如果发生这种情况，请选中选择目标框并单击重试以尝试重新 Tuning（调节）该传感器。要注意的是，要使 CTSU/CTSU2 IP 实现最佳性能，至少需要 10pF 的负载电容。低于 10pF 的负载会导致该传感器通道上的噪声敏感。

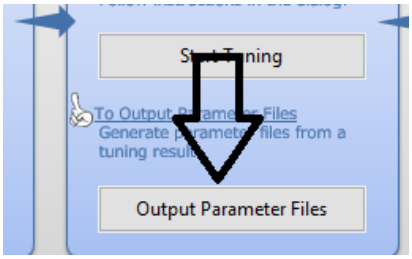
单击对话框中的 *“Continue the Tuning Process”*（继续调节）按钮。 这将退出 Tuning（调节）过程并与目标上的调试会话断开连接。 您应该返回到 e² studio IDE 中默认的 CapTouch Main (QE) RA 界面。



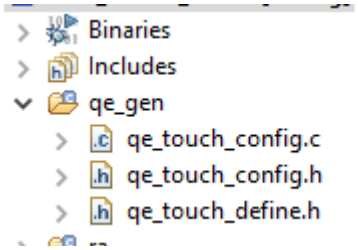
Tuning（调节）过程完成后，出现 CapTouch Main (QE) 的默认视图。这里所示的是配置传感器的 Tuning（调节）结果，是为用户提供的一种快速检查 Tuning（调节）结果的方法。

Method	Kind	Name	Touch Sensor	Parasitic Capacitance[pF]	Sensor Drive Pulse Frequency[MHz]	Threshold
config01	Button(self)	BTN1	TS32	16.889	2.0	1357
config01	Button(self)	BTN2	TS33	14.222	2.0	1404
config01	Button(self)	BTN3	TS28	16.0	2.0	1299

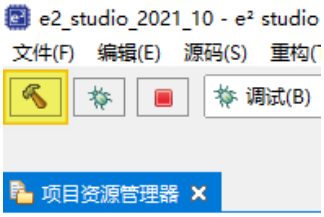
接下来就是输出 Tuning（调节）参数文件。单击按钮 *“Output Parameter Files”*（输出参数文件）。



查看 *“Project Explorer”* 窗口，您将看到文件已添加。其中包含了使用 r_ctsu 和 rm_touch FSP 模块启用触摸检测所需的 Tuning（调节）信息。



使用 IDE 左上角的锤子图标构建项目：

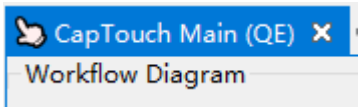


5.4 添加应用程序代码——简单方法

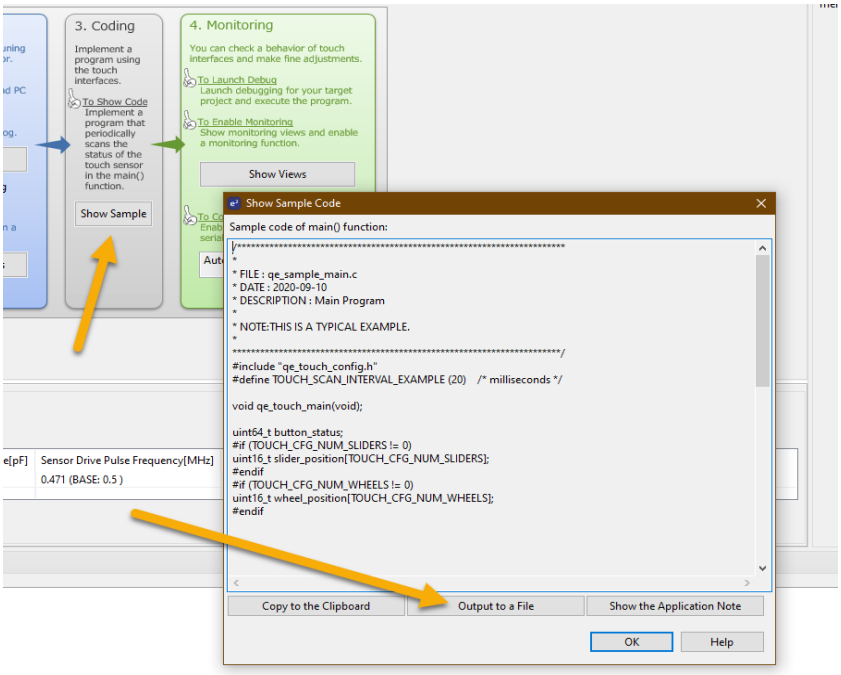
本节将向用户展示如何简单地将所需的触摸 API 添加到项目中，以执行触摸检测功能。您可以利用 QE 工具生成部分示例代码然后稍作修改，从而简化后续的应用程序设置。

在 e² Studio 中，切换到 C/C++ 界面。

在主视图中，打开 CapTouch Main (QE) 选项卡。

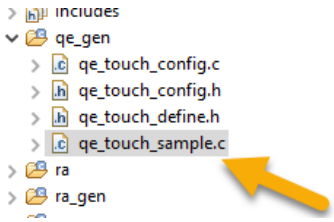


单击 *“Show Sample ”*（显示示例）按钮打开示例代码窗口，然后单击 *“Output to a file ”*（输出到文件），然后单击“OK”。



注：一般情况下，QE 工具的输出示例可以直接作为基本模板使用。但在本实验中，稍后将进行一些细微更改。

请注意，在 qe_gen 文件夹中，添加了一个源文件“qe_touch_sample.c”



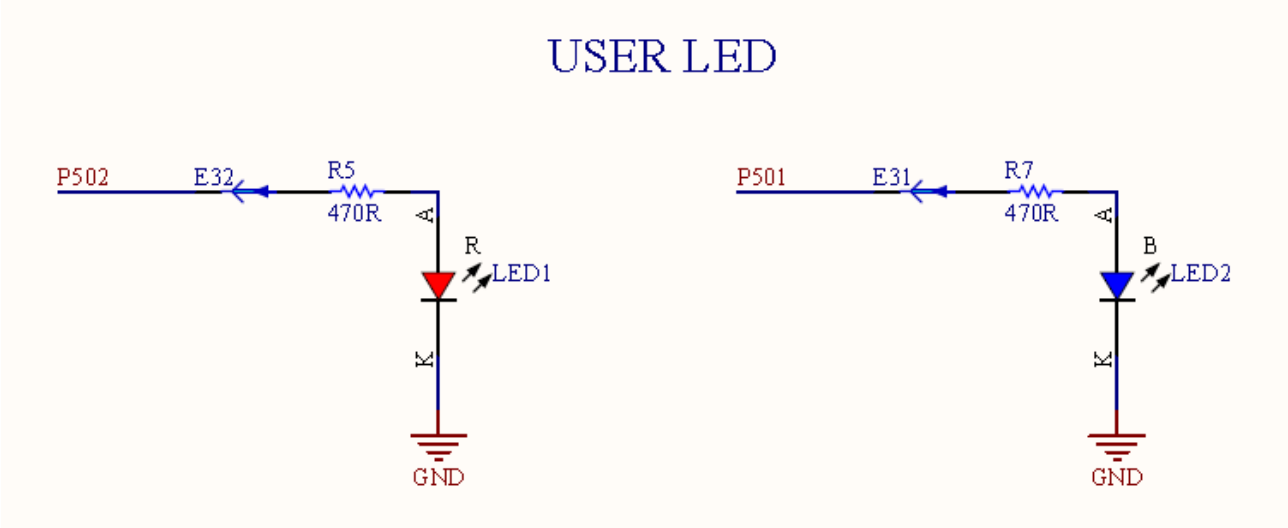
打开 src 文件夹下的 hal_entry.c，为 qe_touch_main(void)函数添加 extern 以及对该函数的调用，用来开启触摸处理。

```
1   #include "hal_data.h"
2
3   extern void qe_touch_main(void);
4
5   FSP_CPP_HEADER
6   void R_BSP_WarmStart(bsp_warm_start_event_t event);
7   FSP_CPP_FOOTER
8
10  * main() is generated by the RA Configuration editor and is used to ge
13  void hal_entry(void) {
14      /* TODO: add your own code here */
15
16      qe_touch_main();
17
18  #if BSP_TZ_SECURE_BUILD
19      /* Enter non-secure code */
20      R_BSP_NonSecureEnter();
21  #endif
```

注意：在带有大量触摸按钮的板上，如果仅仅查看 API 返回的变量，用户很难确定具体的二进制值。作为 Tuning（调节）过程和文件生成的一部分，QE 工具会创建一个预定义的“State MAsk”，再加上用户按钮的名称分配，可以使程序员快速确定由二进制结果激活的按钮。这些掩码可以在 qe_touch_define.h 文件中找到，另外还能看到返回的 uint_64 中按钮的索引以及该传感器的掩码位置。

```
Button State Mask for each configuration.
#define CONFIG01_INDEX_BTN1     (1)
#define CONFIG01_MASK_BTN1      (1ULL << CONFIG01_INDEX_BTN1)
#define CONFIG01_INDEX_BTN2     (2)
#define CONFIG01_MASK_BTN2      (1ULL << CONFIG01_INDEX_BTN2)
#define CONFIG01_INDEX_BTN3     (0)
#define CONFIG01_MASK_BTN3      (1ULL << CONFIG01_INDEX_BTN3)
```

要查看板上触摸的物理指示，可以使用板上的 LED1 和 LED2 作为此视觉提示。 P502 和 P501 在 MCU 板的示意图中表示为连接到 LED。当 GPIO 线被驱动为高电平时，LED 点亮。请注意，CPK-RA2E1 的 BSP 已经为此配置了正确的端口设置，使其成为 GPIO 输出。



打开文件 qe_touch_sample.c，在 DataGet 函数中加入如下代码：

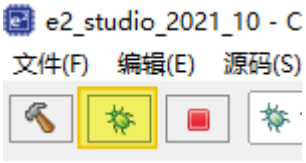
```
        if ( button_status & CONFIG01_MASK_BTN1)
        {
             R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_02,
BSP_IO_LEVEL_HIGH);
             R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_01,
BSP_IO_LEVEL_LOW);
        }
        else if ( button_status & CONFIG01_MASK_BTN2)
        {
             R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_02,
BSP_IO_LEVEL_LOW);
             R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_01,
BSP_IO_LEVEL_HIGH);
        }
        else if ( button_status & CONFIG01_MASK_BTN3)
        {
            R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_02,
BSP_IO_LEVEL_HIGH);
            R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_01,
BSP_IO_LEVEL_HIGH);
        }
        else
        {
            R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_02, BSP_IO_LEVEL_LOW);
            R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_01, BSP_IO_LEVEL_LOW);
        }
```

如图：

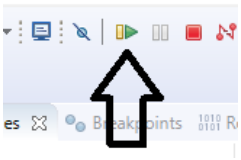
```
err = RM_TOUCH_DataGet(g_qe_touch_instance_config01.p_ctrl, &button_status, NULL, NULL);
if (FSP_SUCCESS == err)
{
    /* TODO: Add your own code here. */
    if ( button_status & CONFIG01_MASK_BTN1)
    {
        R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_02, BSP_IO_LEVEL_HIGH);
        R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_01, BSP_IO_LEVEL_LOW);
    }
    else if ( button_status & CONFIG01_MASK_BTN2)
    {
        R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_02, BSP_IO_LEVEL_LOW);
        R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_01, BSP_IO_LEVEL_HIGH);
    }
    else if ( button_status & CONFIG01_MASK_BTN3)
    {
        R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_02, BSP_IO_LEVEL_HIGH);
        R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_01, BSP_IO_LEVEL_HIGH);
    }
    else
    {
        R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_02, BSP_IO_LEVEL_LOW);
        R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_05_PIN_01, BSP_IO_LEVEL_LOW);
    }
}
```

构建项目。 显示构建结果的控制台输出不应显示错误。

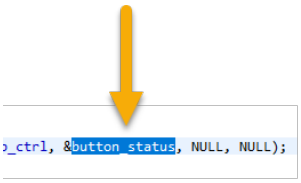
单击 e² studio 左上角的“调试”图标启动调试会话。

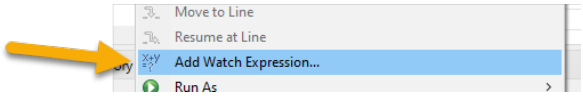


可能会在 hal_entry() 函数处停止处理。 点击“继续”图标以继续调试会话。

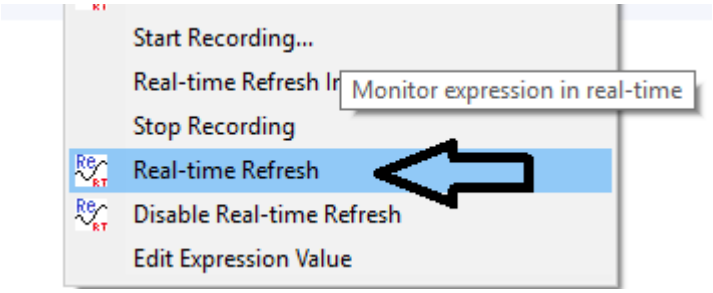


在 qe_touch_sample.c 文件中向下滚动到 while(1) 循环中的 R_TOUCH_DataGet() 函数，并将变量 button_status 添加到表达式窗口。

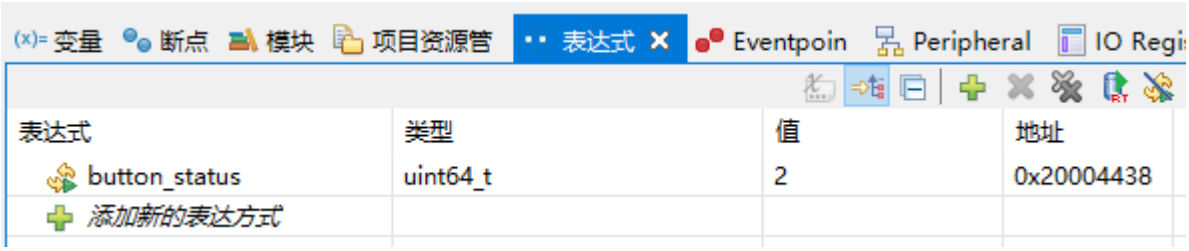




通过右键单击 button_status 变量并选择 *“Real-time Refresh ”*（实时刷新），在表达式窗口中的变量上启用实时刷新。



按下板上的 BTN1 时，“表达式”窗口中的 button_status 将出现“2”，此时 LED1 亮。



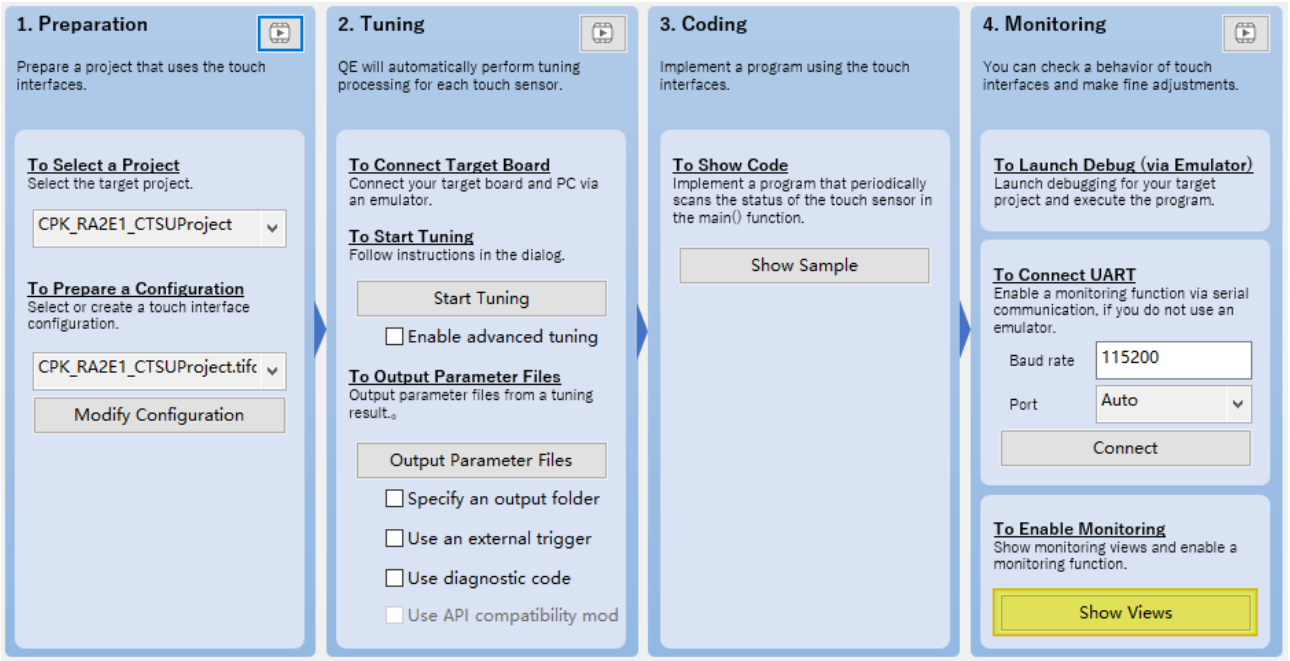
同样地，按下板上的 BTN2 时，表达式窗口中的 button_status 将出现“2”，此时 LED2 亮。按下板上的 BTN3 时，表达式窗口中的 button_status 将出现“4”，此时 LED1 和 LED2 同时亮。

5.5 使用电容式触控工具的 QE 监控触控性能

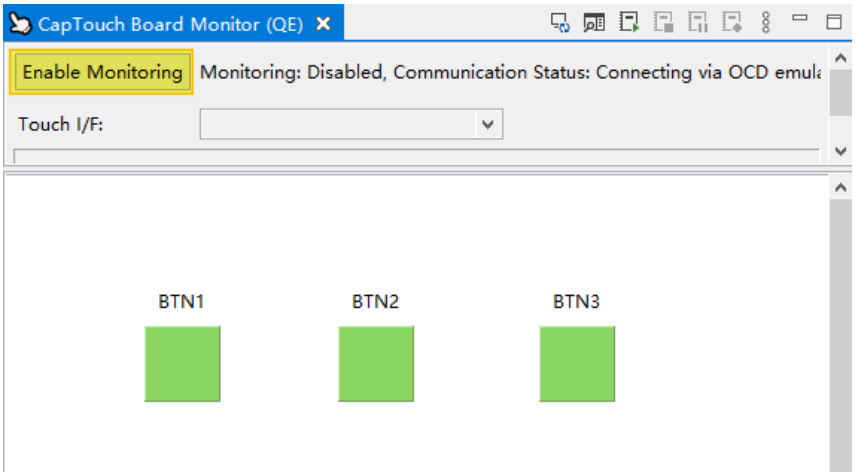
本部分将展示如何通过电容式触控 RA 工具的 QE 监控应用程序的触控性能并在 e² studio IDE 中显示该性能。 此外，用户还可修改一些对电容式触摸性能至关重要的参数。

在主视图中，打开 CapTouch Main (QE)选项卡。

点击 *“Show Views ”*（查看显示）按钮：

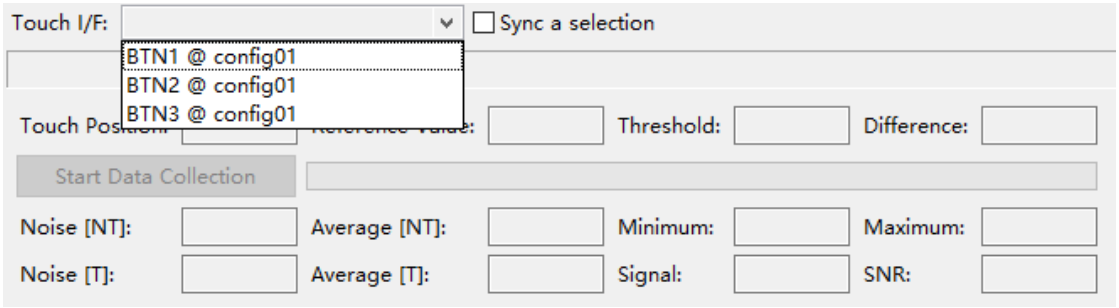


点击 *“Enable Monitoring ”* 开始监测。



您可以通过 CapTouch Board Monitor (QE)查看按钮/按键的触摸图形。 触摸其中一个传感器将会出现手形图标，如下所示：

通过下拉菜单选择 BTN1 @ config01 使用 CapTouch Status Chart (QE) 监控图形触摸灵敏度。



在这里您可以看到所选触摸传感器的当前计数值、参考值、阈值以及触摸传感器是打开还是关闭状态。下面的图例供参考：

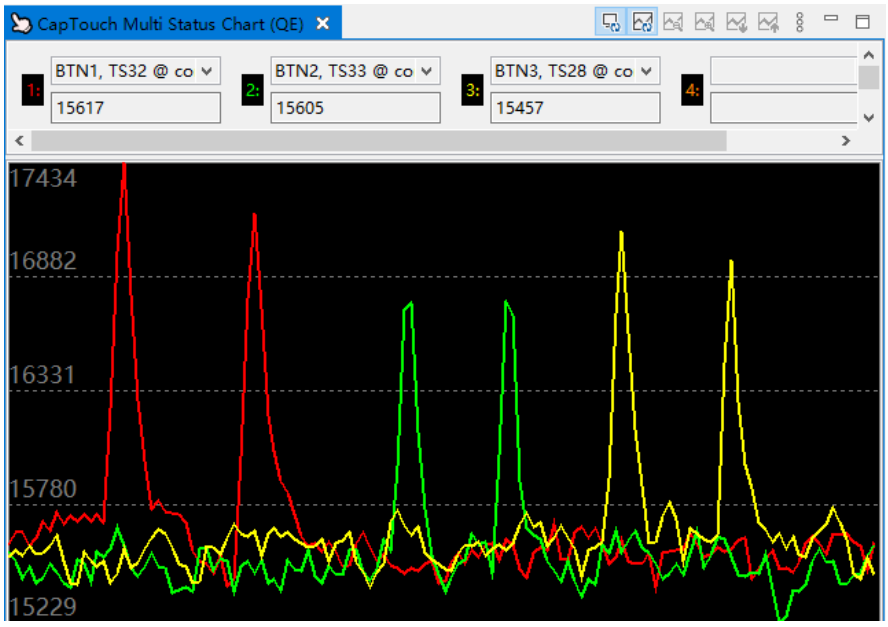
- Items in waveform are as follows.
  - Count Value
  - Reference Value
  - Touch Threshold
- Rectangles on the bottom of waveform indicates that selected Touch I/F is judged Touch-On.
- On the above of this window, Check Count Value, Reference Value, Touch Threshold and difference value as numeric value.

您还可以通过 CapTouch Multi Status Chart (QE) 视图同时查看多个传感器以评估串扰等性能。

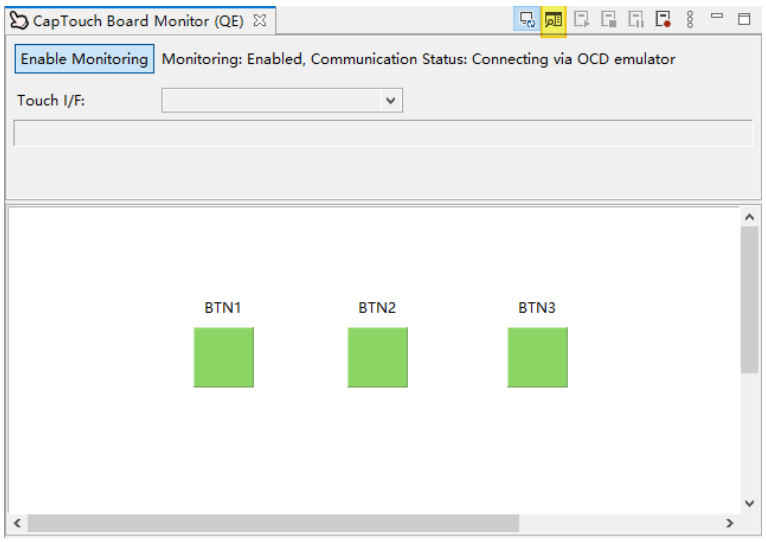
- Select Touch I/F from pull-down menu on the top of this window.
- Relation between color of waveform and [1:] – [8:] is as follows.

  [1:] [2:] [3:] [4:]

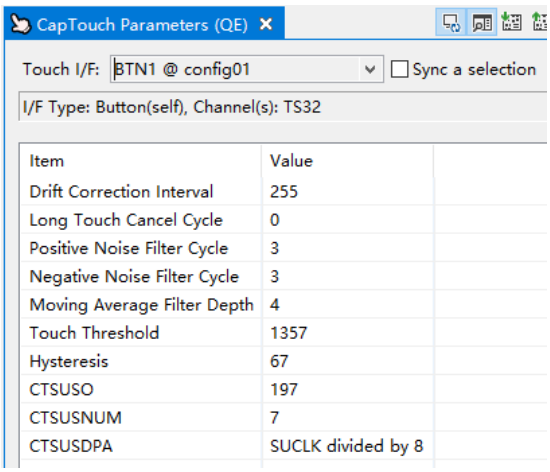
  [5:] [6:] [7:] [8:]
- On the above of this window, Check Count Value, Reference Value, Touch Threshold and difference value as numeric value.



在 CapTouch Board Monitor (QE) 的“高级设置”中可以评估触摸传感器的数值结果。单击黄色矩形框中的菜单图标以显示传感器的数字数据：



触摸传感器性能参数也可以通过 CapTouch Parameter (QE) 查看。当启用此视图的高级模式显示时，CTSU 寄存器将显示在列表中。可通过下拉菜单选择 BTN1 @ config01 查看，如下图所示：



要了解 CTSU 电容式触摸电极设计方面的知识，请参阅以下应用笔记：
“电容式传感器微控制器 CTSU 电容式触摸电极设计指南” (R30AN0389)
https://www.renesas.com/jp/zh/document/apn/capacitive-sensor-microcontrollers-ctsu-capacitive-touch-electrode-design-guide?language=zh&r=1398061

要学习更多关于使用 QE for Capacitve Touch (RA) 和 FSP 创建触摸界面的知识，电容式触摸传感器，调整触摸传感器，并在应用程序中实现触摸中间件，请参阅以下应用笔记：
“RA Family Using QE and FSP to Develop Capacitive Touch Applications” (R01AN4934)
https://www.renesas.com/jp/zh/document/apn/ra-family-using-qe-and-fsp-develop-capacitive-touch-application?language=en&r=1398061

恭喜！

您已成功完成本练习！也完成了 CPK-RA2L1/RA2E1 评估板的入门操作！

本章要点：

- 使用 FSP 配置器和 CapTouch 中间件便于增加对触摸按键的支持。

6. 《CPK-RA2L1/RA2E1 评估板入门》的文件列表

文件类型	内容	出现位置	文件名/文件夹名
文件（PDF）	用户手册	-	r01qs0056cc0100-cpk-ra2-quickguide.pdf
BSP 包	用于导入 BSP 的文件	第 1 章	CPKsBSP_FSP3.5.0.zip
样例程序	基本样例程序	第 2 章	CPK_RA2L1_BlinkyProject.rar CPK_RA2E1_BlinkyProject.rar
样例程序	LED 闪烁样例程序	第 3 章	CPK_RA2L1_ExampleProject.rar CPK_RA2E1_ExampleProject.rar
样例程序	FreeRTOS 样例程序	第 4 章	CPK_RA2L1_RtosProject.rar CPK_RA2E1_RtosProject.rar
样例程序	CTSU 样例程序	第 5 章	CPK_RA2L1_CTSUProject.rar CPK_RA2E1_CTSUProject.rar

7. 参考文献

《瑞萨 RA MCU 基础知识》
（最新版本请从瑞萨电子网页上取得）

技术信息/技术更新
（最新信息请从瑞萨电子网页上取得）

网站和咨询窗口

RA 产品家族网站

- http://www.renesas.com/ra

瑞萨学院网站

- https://academy.renesas.com

瑞萨 Rulz 中文论坛

- https://japan.renesasrulz.com/rulz-chinese/

修订记录

Rev.	发行日	修订内容	
		页	要点
1.00	2021.12	—	初版发行